МАШИННОЕ ОБУЧЕНИЕ
УЧЕБНОЕ ПОСОБИЕ

Оглавление

Глава 1. Числовые атрибуты 7
1.1 Однофакторный анализ 7
1.1.1 Меры центральной тенденции 8
1.1.2 Меры дисперсии 10
1.2 Двухфакторный анализ 15
1.2.1 Эмпирическая совместная функция распределения вероятности 15
1.2.2 Математическое ожидание и дисперсия 15
1.2.3 Меры зависимости 16
1.3 Многофакторный анализ 19
1.4. Нормальная форма 22
1.5 Нормальное распределение 24
1.5.1 Одномерное Нормальное Распределение 24
1.5.2 Многомерное Нормальное Распределение 27
Глава 2. Ядерные методы 31
2.1 Ядерные матрицы 32
2.1.1 Отображение Воспроизводящего Ядра 33
2.1.2 Отображение Эмпирического Ядра 35
2.1.3 Мерсеровское Отображение Ядра 36
2.1.3.1 Отображение Ядра При Конкретных Данных 36
2.1.3.2 Мерсеровское Отображение Ядра 37
2.2. Векторные ядра 38
2.2.1 Полиномиальное ядро 38
2.2.2 Гауссово ядро 40
2.3 Основные операции ядра в пространстве признаков 41
2.3.1 Норма точки 41
2.3.2 Расстояние между точками 41
2.3.3 Среднее в пространстве признаков 41
2.3.4 Общая дисперсия в пространстве признаков 42
2.3.5 Центрирование в пространстве признаков 42

2.3.6 Нормализация в пространстве признаков 43
2.4. Ядра для сложных объектов 44
2.4.1. Ядро спектра для строк 44
2.4.2. Ядра с диффузией на вершинах графа 45
2.4.3 Степень ядер 45
2.4.4 Ядро с экспоненциальной диффузией 46
2.4.5 Ядро с диффузией фон Неймана 47
Глава 3. Снижение размерности 48
3.1 Предпосылки 48
3.2 Метод главных компонент 50
3.2.1 Наилучшая линейная аппроксимация 50
3.2.2 Наилучшая двумерная аппроксимация 53
3.2.3 Наилучшая r-мерная аппроксимация 56
3.2.4 Геометрия метода главных компонент 59
3.3 Ядерный анализ главных компонент 61
3.4. Сингулярное разложение 65
3.4.1. Геометрия сингулярного разложения 66
3.4.2. Связь между SVD и PCA 67
Глава 4. Поиск наборов объектов 69
4.1 Часто встречающиеся наборы и ассоциативные правила 69
4.1.1 Наборы объектов и наборы тидов 69
4.1.2 Представление базы данных 69
4.1.3 Поддержка и часто встречающиеся наборы элементов 70
4.1.4 Ассоциативные правила 70
4.1.5 Майнинг наборов данных и правил 71
4.2 Алгоритмы майнинга набора данных 71
4.2.1 Генерация кандидатов 71
4.2.2 Вычисление поддержки 72
4.3 Вычислительная сложность 72
4.3.1 Поуровневый подход: алгоритм Apriori 72
4.3.2 Подход пересечения наборов тидов: алгоритм Eclat 74
4.3.3 Подход дерева частоты образцов: алгоритмов FPGrowth 77
4.3.4 Генерация правил ассоциации 78
Глава 5. Суммирование наборов 80
5.1 Максимальная и замкнутая частота наборов 80

5.2 Поиск максимальной частоты набора: алгоритм GENMAX 84
5.3 Поиск закрытых часто встречающихся наборов элементов: CHARM алгоритм. 86
5.4 Невыводимые наборы объектов 88
5.4.1 Обобщённые наборы элементов 89
5.4.2 Принцип включения-исключения 89
5.4.3 Границы поддержки набора элементов 89
5.4.4 Невыводимые наборы объектов 90
Глава 6. Анализ последовательностей 91
6.1. Частота последовательности 91
6.2 Поиск часто встречающихся последовательностей 92
6.2.1 Уровневый поиск: GSP 93
6.2.2 Вертикальный поиск последовательности: Spade 94
6.2.3 Поиск последовательностей на основе проекций: PrefixSpan 95
6.3 Поиск подстрок с помощью суффиксных деревьев 96
6.3.1 Суффиксное Дерево 96
6.3.2 Алгоритм линейного времени Укконена 97
Глава 7. Оценка паттернов и правил 101
7.1. Меры для оценки паттернов и правил 101
7.1.1. Меры по оценке правил 101
7.1.2 Меры оценки модели 104
7.1.3 Сравнение множественных правил и шаблонов 105
7.2 Проверка значимости и доверительные интервалы 107
7.2.1 Точный тест Фишера для продуктивных правил 107
7.2.2 Проверка значимости перестановками 110
Глава 8. Репрезентативная кластеризация 112
8.1 Алгоритм K-средних 112
8.2 Ядерный алгоритм K-средних 118
8.3 Кластеризация с максимизацией ожиданий 122
8.3.1 EM в одномерном пространстве 123
8.3.2 EM в *d*-мерном пространстве 125
8.3.3 Оценка максимального правдоподобия 129
8.3.4 EM подход 133
Глава 9. Иерархическая кластеризация 137
9.1 Исходные данные 137
9.2 Агломеративная иерархическая кластеризация 139

Глава 10. Кластеризация, основанная на плотности ... 145
10.1 Алгоритм DBSCAN ... 145
10.2 Оценка плотности ядра ... 147
10.2.1 Одномерная оценка плотности ... 147
10.2.2 Многомерная оценка плотности ... 151
10.2.3 Оценка плотности ближайшего соседа ... 153
10.3 Кластеризация на основе плотности: DENCLUE ... 153
Глава 11. Валидация кластеризации ... 160
11.1 Внешние меры ... 160
11.1.1 Соответствующие меры ... 161
11.1.2 Меры, основанные на энтропии ... 165
11.1.3 Парные меры ... 169
11.1.4 Корреляционные меры ... 173
11.2 Внутренние меры ... 176
11.3. Относительные меры ... 182
11.3.1 Стабильность кластера ... 185
11.3.2 Тенденция к кластеризации ... 187
Глава 12. Вероятностная классификация ... 190
12.1 Классификатор Байеса ... 190
12.1.1 Оценка априорной вероятности ... 190
12.1.2 Оценка правдоподобия ... 191
12.2. Наивный байесовский классификатор ... 193
12.3 Классификатор K-ближайших соседей. ... 197
Глава 13. Классификатор дерева решений ... 199
13.1 Деревья решений ... 199
13.2 Алгоритм дерева решений ... 201
13.2.1 Меры Оценки С Разделением Точек ... 202
13.2.2 Оценка Точек Разделения ... 204
13.2.3 Вычислительная Сложность ... 208
Глава 14. Линейный дискриминантный анализ ... 210
14.1 Оптимальный линейный дискриминант ... 210
14.2 Дискриминантный анализ ядра ... 216
Глава 15. Метод опорных векторов ... 225
15.1 Опорные вектора и расстояния (зазоры) ... 225
15.2 Случай линейного разбиения ... 228

15.3 SVM с мягким зазором: случай линейной неразделимости ... 231
15.4 Ядерный SVM: нелинейный случай ... 235
15.4 Ядерный SVM классификатор ... 237
15.5 Алгоритмы обучения SVM ... 237
15.5.1 Двойственное решение: стохастический градиентный подъем ... 238
15.5.2 Основное решение: оптимизация Ньютона ... 240
Глава 16. Оценка классификации ... 246
16.1 Классификация мер эффективности ... 246
16.1.1 Меры основанные на таблице сопряженности ... 247
16.1.2 Бинарная классификация: положительный и отрицательный класс ... 251
16.1.3 ROC анализ ... 254
16.2 Оценка классификаторов ... 261
16.2.1 K-кратная перекрестная проверка ... 261
16.2.2 Повторная выборка начальной загрузки ... 262
16.2.3 Доверительные интервалы ... 263
16.2.4 Сравнение классификаторов: парный t-тест ... 267
16.3 Разложение на смещение и разброс (Bias-variance decomposition) ... 268
16.3.1 Классификаторы ансамбля ... 272
Глава 17. Лабораторные работы ... 279
17.1 Лабораторная работа № 1 ... 279
17.2 Лабораторная работа № 2 ... 282
17.3 Лабораторная работа № 3 ... 284
17.4 Лабораторная работа № 4 ... 286
17.5 Лабораторная работа № 5 ... 289
17.6 Лабораторная работа № 6 ... 292
17.7 Лабораторная работа № 7 ... 294
17.8 Лабораторная работа № 8 ... 297
Глава 18. Практические занятия ... 300
18.1 Практическое занятие № 1 ... 300
18.2 Практическое занятие № 2 ... 301
18.3 Практическое занятие № 3 ... 302
18.4 Практическое занятие № 4 ... 303
18.5 Практическое занятие № 5 ... 305
18.6 Практическое занятие № 6 ... 307
18.7 Практическое занятие № 7 ... 309

18.8 Практическое занятие № 8 310
Глава 19. Индивидуальное домашнее задание 312

Глава 1. Числовые атрибуты

В этой главе мы обсуждаем основные статистические методы исследовательского анализа данных числовых атрибутов. Мы рассматриваем меры центральной тенденции или местоположения, меры дисперсии и меры линейной зависимости или связи между атрибутами. Мы подчеркиваем связь между вероятностным и геометрическим и алгебраическим представлениями матрицы данных.

1.1 Однофакторный анализ

Однофакторный анализ фокусируется на одном атрибуте в момент времени; таким образом, матрицу данных D можно представить как матрицу размера $n \times 1$ или просто вектор-столбец, заданный как

$$\boldsymbol{D} = \begin{pmatrix} \underline{X} \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

где X - числовой атрибут интереса, причем $x_i \in \mathbb{R}$. Предполагается, что X - случайная величина, причем каждая точка x_i $(1 \leq i \leq n)$ рассматривается как тождественная случайная величина. Мы предполагаем, что рассматриваемые данные являются случайной выборкой, взятой из X, то есть каждая переменная x_i независима и распределена аналогично X. В векторном представлении мы рассматриваем выборку как n-мерный вектор и записываем $X \in \mathbb{R}^n$.

В общем, плотность распределения вероятности $f(x)$ и кумулятивная функция распределения $F(x)$ атрибута X неизвестны. Однако мы можем оценить эти распределения непосредственно из выборки данных, что также позволит нам вычислить статистику для оценки нескольких важных параметров совокупности.

Эмпирическая кумулятивная функция распределения

Эмпирическая кумулятивная функция распределения (CDF) X задается как

$$\hat{F}(x) = \frac{1}{n}\sum_{i=1}^{n} I(x_i \leq x), \tag{1.1}$$

где

$$I(x_i \leq x) = \begin{cases} 1, & \text{если } x_i \leq x \\ 0, & \text{если } x_i > x \end{cases}$$

это индикатор, который указывает, выполняется ли данное условие или нет. Интуитивно, чтобы получить эмпирическую CDF, мы вычисляем для каждого значения $x \in \mathbb{R}$, сколько точек в выборке меньше или равно x. Эмпирическая CDF помещает значение вероятности, равное $\frac{1}{n}$, в каждую точку x_i. Обратите внимание, что мы используем обозначение

$\hat{F}$ для указания того факта, что эмпирическая CDF является оценкой для неизвестной совокупности CDF *F*.

Функция обратного кумулятивного распределения

Определим обратную кумулятивную функцию распределения или квантиль-функцию случайной величины X следующим образом:

$$F^{-1}(q) = min\{x \mid F(x) \geq q\}, \tag{1.2}$$

То есть обратная CDF дает наименьшее значение *X*, для которого q значений меньше, и $1 - q$ значений больше. Эмпирическая обратная кумулятивная функция распределения $\hat{F}^{-1}$ может быть получена из уравнения (1.1).

Эмпирическая функция вероятности

Эмпирическая функция распределения вероятности (PMF) X задается как

$$\hat{f}(x) = P(X = x) = \frac{1}{n}\sum_{i=1}^{n} I(x_i = x), \tag{1.3}$$

где

$$I(x_i = x) = \begin{cases} 1, & \text{если } x_i = x \\ 0, & \text{если } x_i \neq x \end{cases}$$

Эмпирическая PMF также помещает значение вероятности, равное $\frac{1}{n}$, в каждую точку x_i.

1.1.1 Меры центральной тенденции

Эти меры описывают концентрацию значений, «средние» значения и т.д.

Математическое ожидание

Математическое ожидание случайной величины *X* – это среднее арифметическое значение *X*. Оно представляет собой численное выражение местоположения или центральной тенденции распределения *X*.

Математическое ожидание дискретной случайной величины X определяется как

$$\mu = E[X] = \sum_{x} xf(x), \tag{1.4}$$

где *f(x)* – функция распределения вероятности *X*.

Математическое ожидание непрерывной случайной величины X определяется как

$$\mu = E[X] = \int_{-\infty}^{\infty} xf(x)dx,$$

где *f(x)* – функция распределения плотности вероятности *X*.

Выборочное среднее значение

Выборочное среднее значение – это функция $\hat{\mu}: \{x_1, x_2, \dots, x_n\} \longrightarrow \mathbb{R}$, определенная как среднее значение x_i-х:

$$\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} x_i\,, \tag{1.5}$$

Выборочное среднее значение служит для оценки неизвестного среднего значения μ для X. Оно может быть получено путем подставления эмпирической PMF $\hat{f}(x)$ в Ур. (1.4):

$$\hat{\mu} = \sum_x x\hat{f}(x) = \sum_x x\left(\frac{1}{n}\sum_{i=1}^{n} I(x_i = x)\right) = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Выборочное среднее – несмещенная оценка

Оценка $\hat{\theta}$ называется несмещенной оценкой параметра θ, если $E\left[\hat{\theta}\right] = \theta$ для любого возможного значения θ. Выборочное среднее значение является несмещенной оценкой для среднего значения μ, поскольку

$$E[\hat{\mu}] = E\left[\frac{1}{n}\sum_{i=1}^{n} x_i\right] = \frac{1}{n}\sum_{i=1}^{n} E[x_i] = \frac{1}{n}\sum_{i=1}^{n} \mu = \mu \tag{1.6}$$

где мы используем тот факт, что случайные величины x_i являются IID согласно X, что означает, что они имеют то же среднее значение μ, что и X, то есть $E[x_i] = \mu$ для всех x_i. Мы также использовали тот факт, что функция ожидания E является *линейным оператором*, то есть для любых двух случайных величин X и Y и действительных чисел a и b мы имеем $E[aX + bY] = aE[X] + bE[Y]$.

Устойчивость

Мы говорим, что статистика является устойчивой, если на нее не влияют экстремальные значения (например, выбросы) в данных. К сожалению, выборочное среднее значение не является устойчивым, поскольку одно большое значение (выброс) может исказить среднее значение. Более устойчивым показателем является усеченное среднее значение, полученное после отбрасывания небольшой части экстремальных значений на одном или обоих концах. Более того, среднее значение может несколько вводить в заблуждение, поскольку обычно это не значение, которое встречается в выборке, и оно может даже не быть значением, которое случайная величина может фактически принять (для дискретной случайной величины). Например, количество автомобилей на душу населения является целочисленной случайной величиной, но, по данным Бюро исследований транспорта США, среднее количество легковых автомобилей в США составляло 0,45 в 2008 г. (137,1 миллиона автомобилей при численности

населения). 304,4 млн). Очевидно, что нельзя владеть 0,45 машинами; это можно интерпретировать так, что в среднем на 100 человек приходится 45 автомобилей.

Медиана

Медиана случайной величины определяется как значение m такое, что

$$P(X \leq m) \geq \frac{1}{2} \text{ и } P(X \geq m) \geq \frac{1}{2}$$

Другими словами, медиана m - это «самое среднее» значение; половина значений X меньше, а половина значений X больше m. Таким образом, с точки зрения (обратной) кумулятивной функции распределения, медиана - это значение m, для которого

$$F(m) = 0.5 \text{ или } m = F^{-1}(0.5)$$

Медиана выборки может быть получена из эмпирической CDF [Ур. (1.1)] или эмпирической обратной CDF [Ур. (1.2)] вычислением

$$\hat{F}(m) = 0.5 \text{ или } m = \hat{F}^{-1}(0.5)$$

Более простой подход к вычислению медианы выборки – сначала отсортировать все значения x_i $(i \in [1, n])$ в порядке возрастания. Если n нечетное, медиана – это значение в позиции $\frac{n+1}{2}$. Если n четное, то оба значения на позициях $\frac{n}{2}$ и $\frac{n}{2} + 1$ – медианы.

В отличие от математического ожидания, медиана является устойчивой статистикой, поскольку на нее не сильно влияют экстремальные значения. Кроме того, это значение, которое встречается в выборке, и значение, которое случайная величина может фактически принять.

Мода

Мода случайной величины X - это значение, при котором функция распределения вероятности или функция распределения плотности вероятности достигает своего максимального значения, в зависимости от того, является ли X дискретной или непрерывной, соответственно.

Мода выборки – это значение, для которого эмпирическая функция распределения вероятности [Ур. (1.3)] достигает своего максимума, определяемого как

$$mode(X) = arg \max_x \hat{f}(x)$$

Мода может быть не очень полезной мерой центральной тенденции для выборки, потому что непоказательный элемент может случайно оказаться наиболее частым элементом. Кроме того, если все значения в выборке различны, каждое из них будет модой.

1.1.2 Меры дисперсии

Меры дисперсии дают представление о разбросе или изменении значений случайной величины.

Диапазон

Диапазон значений или просто диапазон случайной величины X – это разница между максимальным и минимальным значениями X, заданная как

$$r = \max\{X\} - \min\{X\}$$

Диапазон (значений) X является параметром совокупности, не путать с диапазоном функции X, которая представляет собой набор всех значений, которые X может принимать. Какой диапазон используется, должно быть ясно из контекста.

Диапазон выборки – это статистика, заданная как

$$\hat{r} = \max_{i=1}^{n}\{x_i\} - \min_{i=1}^{n}\{x_i\}$$

По определению, диапазон чувствителен к экстремальным значениям и, следовательно, не является устойчивым.

Межквартильный диапазон

Квартили – это особые значения функции квантилей [Ур. (1.2)], которые делят данные на четыре равные части. То есть квартили соответствуют значениям квантилей 0,25, 0,5, 0,75 и 1,0. Первый квартиль – это значение $q_1 = F^{-1}(0.25)$ слева от которого лежит 25% точек; второй квартиль совпадает со значением медианы $q_2 = F^{-1}(0.5)$, слева от которого лежит 50% точек; третий квартиль $q_3 = F^{-1}(0.75)$ - значение, слева от которого лежат 75% точек; а четвертый квартиль - максимальное значение X, слева от которого лежат 100% точек.

Более надежной мерой дисперсии X является межквартильный диапазон (IQR), определяемый как

$$IQR = q_3 - q_1 = F^{-1}(0.75) - F^{-1}(0.25) \tag{1.7}$$

IQR также можно рассматривать как *усеченный диапазон*, в котором мы отбрасываем 25% низких и высоких значений X. Или, иначе говоря, это диапазон средних 50% значений X. IQR является устойчивым по определению.

Образец IQR может быть получен путем включения эмпирической обратной CDF в формуле. (1.7):

$$\widehat{IQR} = \hat{q}_3 - \hat{q}_1 = \hat{F}^{-1}(0.75) - \hat{F}^{-1}(0.25)$$

Дисперсия и стандартное отклонение

Дисперсией случайной величины X показывает насколько значения X отклоняются от среднего значения X или её математического ожидания. Более формально дисперсия –это ожидаемая величина квадрата отклонения от математического ожидания, определяемая как

$$\sigma^2 = var(X) = E[(X-\mu)^2] = \begin{cases} \sum_{x} (x-\mu)^2 f(x) & - \text{ если } X \text{ дискретная} \\ \int_{-\infty}^{\infty} (x-\mu)^2 f(x)dx & - \text{если } X \text{ непрерывная} \end{cases} \tag{1.8}$$

Среднеквадратическое отклонение σ определяется как положительный квадратный корень из дисперсии σ^2.

Мы также можем записать дисперсию как разницу между ожиданием X^2 и квадратом ожидания X:

$$\sigma^2 = var(X) = E[(X-\mu)^2] = E[X^2 - 2\mu X + \mu^2] = E[X^2] - 2\mu E[X] + \mu^2 \\ = E[X^2] - 2\mu^2 + \mu^2 = E[X^2] - (E[X])^2 \tag{1.9}$$

Стоит отметить, что дисперсия – это фактически *второй момент относительно математического ожидания*, соответствующий $r = 2$, который является частным случаем *r-го момента относительно математического ожидания* для случайной величины X, определяемой как $\mathrm{E}[(\mathbb{x} - \mu)^r]$.

Выборочная дисперсия

Выборочная дисперсия определяется как

$$\hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{\mu})^2 \tag{1.10}$$

Это среднеквадратическое отклонение значений данных x_i от выборочного среднего $\hat{\mu}$, которое может быть получено путем вставки эмпирической функции вероятности $\hat{f}$ из уравнения (1.3) в уравнение. (1.8), поскольку

$$\hat{\sigma}^2 = \sum_x (x-\hat{\mu})^2\,\hat{f}(x) = \sum_x (x-\hat{\mu})^2 \left(\frac{1}{n}\sum_{i=1}^{n} I(x_i = x)\right) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{\mu})^2$$

Выборочное среднеквадратическое отклонение выборки определяется как положительный квадратный корень из дисперсии выборки:

$$\hat{\sigma} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{\mu})^2}$$

Стандартизированная оценка, также называемая *z-оценкой*, выборочного значения x_i – это количество стандартных отклонений, на которые значение отличается от математического ожидания:

$$z_i = \frac{x_i - \hat{\mu}}{\hat{\sigma}}$$

Иными словами, *z*-оценка x_i измеряет отклонение x_i от среднего значения $\hat{\mu}$ в единицах $\hat{\sigma}$.

Геометрическая интерпретация дисперсии выборки

Мы можем рассматривать выборку данных для атрибута X как вектор в n-мерном пространстве, где n - размер выборки. То есть записываем $X = (x_1, x_2, \dots, x_n)^T \in \mathbb{R}^n$. Далее, пусть

$$Z = X - \mathbf{1} \cdot \hat{\mu} = \begin{pmatrix} x_1 - \hat{\mu} \\ x_2 - \hat{\mu} \\ \vdots \\ x_n - \hat{\mu} \end{pmatrix}$$

центральный вектор атрибута, где $\mathbf{1} \in \mathbb{R}^n$ – n-мерный вектор, все элементы которого имеют значение 1. Мы можем переписать уравнение (1.10) в терминах величины Z, то есть скалярного произведения Z на себя:

$$\hat{\sigma}^2 = \frac{1}{n}\|Z\|^2 = \frac{1}{n}Z^T Z = \frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{\mu})^2 \tag{1.11}$$

Таким образом, дисперсию выборки можно интерпретировать как возведенную в квадрат величину центрального вектора атрибута или скалярное произведение центрального вектора атрибута на себя, нормализованное по размеру выборки.

Дисперсия выборочного среднего значения

Поскольку выборочное среднее значение $\hat{\mu}$ само по себе является статистикой, мы можем вычислить его среднее значение и дисперсию. Ожидаемая величина выборочного среднего значения – это просто μ, как мы видели в уравнении (1.6). Чтобы получить выражение для дисперсии выборочного среднего значения, мы используем тот факт, что все случайные величины x_i независимы, и, следовательно,

$$var\left(\sum_{i=1}^{n} x_i\right) = \sum_{i=1}^{n} var(x_i)$$

Кроме того, поскольку все x_i распределены идентично X, они имеют такую же дисперсию, как и X, то есть

$$var(x_i) = \sigma^2 \text{ для всех } i$$

Объединяя два приведенных выше факта, мы получаем

$$var\left(\sum_{i=1}^{n} x_i\right) = \sum_{i=1}^{n} var(x_i) = \sum_{i=1}^{n} \sigma^2 = n\sigma^2 \tag{1.12}$$

Далее отметим, что

$$E\left[\sum_{i=1}^{n} x_i\right] = n\mu \tag{1.13}$$

Используя уравнения (1.9), (1.12) и (1.13) дисперсия выборочного среднего значения $\hat{\mu}$ может быть вычислена как

$$var(\hat{\mu}) = E[(\hat{\mu} - \mu)^2] = E[\hat{\mu}^2] - \mu^2 = E\left[\left(\frac{1}{n}\sum_{i=1}^{n} x_i\right)^2\right] - \frac{1}{n^2}E\left[\sum_{i=1}^{n} x_i\right]^2$$
$$= \frac{1}{n^2}\left(E\left[\left(\sum_{i=1}^{n} x_i\right)^2\right] - E\left[\sum_{i=1}^{n} x_i\right]^2\right) = \frac{1}{n^2}var\left(\sum_{i=1}^{n} x_i\right) = \frac{\sigma^2}{n} \quad (1.14)$$

Другими словами, выборочное среднее значение $\hat{\mu}$ варьируется или отклоняется от математического ожидания μ пропорционально дисперсии совокупности σ^2. Однако отклонение можно уменьшить, учитывая больший размер выборки n.

Дисперсия выборки является смещенной, но асимптотически она не смещена

Выборочная дисперсия в уравнении (1.10) представляет собой смещенную оценку истинной дисперсии совокупности σ^2, то есть$E[\hat{\sigma}^2] \neq \sigma^2$. Чтобы показать это, мы используем тождество

$$\sum_{i=1}^{n}(x_i - \mu)^2 = n(\hat{\mu} - \mu)^2 + \sum_{i=1}^{n}(x_i - \hat{\mu})^2 \quad (1.15)$$

Вычисляя математическое ожидания σ^2 с помощью уравнения (1.15) на первом шаге получаем

$$E[\hat{\sigma}^2] = E\left[\frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{\mu})^2\right] = E\left[\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2\right] - E[(\hat{\mu} - \mu)^2] \quad (1.16)$$

Напомним, что случайные величины x_i являются IID согласно X, что означает, что они имеют то же математическое ожидание μ и дисперсию σ^2, что и X. Это означает, что

$$E[(x_i - \mu)^2] = \sigma^2$$

Далее, из уравнения (1.14) среднее значение выборки $\hat{\mu}$ имеет дисперсию $E[(\hat{\mu} - \mu)^2] = \frac{\sigma^2}{n}$. Подставляя это в уравнение (1.16) получаем

$$E[\hat{\sigma}^2] = \frac{1}{n}n\sigma^2 - \frac{\sigma^2}{n} = \left(\frac{n-1}{n}\right)\sigma^2$$

Выборочная дисперсия $\hat{\sigma}^2$ является смещенной оценкой σ^2, поскольку ее ожидаемое значение отличается от дисперсии генеральной совокупности в $\frac{n-1}{n}$ раз. Однако *асимптотически она не смещена*, смещение обращается в нуль при $n \longrightarrow \infty$, поскольку

$$\lim_{n\to\infty}\frac{n-1}{n} = \lim_{n\to\infty} 1 - \frac{1}{n} = 1$$

Иными словами, по мере увеличения размера выборки мы имеем

$$E[\hat{\sigma}^2] \longrightarrow \sigma^2 \text{ при } n \longrightarrow \infty$$

1.2 Двухфакторный анализ

В двухфакторном анализе рассматриваются одновременно два атрибута. Особенный интерес представляет определение связи или зависимости между атрибутами. Рассмотрим данные $\boldsymbol{D}$, представленные в матрице $n \times 2$:

$$\boldsymbol{D} = \begin{pmatrix} \boldsymbol{X_1} & \boldsymbol{X_2} \\ \hline \boldsymbol{x_{11}} & \boldsymbol{x_{12}} \\ \boldsymbol{x_{21}} & \boldsymbol{x_{22}} \\ \vdots & \vdots \\ \boldsymbol{x_{n1}} & \boldsymbol{n_2} \end{pmatrix},$$

где X_1, X_2 – интересующие нас числовые атрибуты.

Геометрически, можно представить $\boldsymbol{D}$ двумя способами: как вид из n точек или векторов в двумерном пространстве над атрибутами X_1, X_2, т.е. $\boldsymbol{x}_i = (x_{i1}, x_{i2})^T \in \mathbb{R}^2$, или как вид из двух точек или векторов над n-мерным пространством:

$$X_1 = (x_{11}, x_{21}, \dots, x_{n1})^T,$$
$$X_2 = (x_{12}, x_{22}, \dots, x_{n2})^T,$$

где каждый столбец является вектором в $\mathbb{R}^n$.

С вероятностной точки зрения, вектор-столбец $\boldsymbol{X} = (X_1, X_2)^T$ можно считать двумерной векторной случайной величиной и точки $\boldsymbol{x}_i (1 \le i \le n)$ – как случайную выборку из $\boldsymbol{X}$, т.е. $\boldsymbol{x}_i$ считаются независимыми и выбранными равновероятно из $\boldsymbol{X}$.

1.2.1 Эмпирическая совместная функция распределения вероятности

Эмпирическая совместная функция распределения вероятностей для $\boldsymbol{X}$ определена следующим образом:

$$\hat{f}(\boldsymbol{x}) = P(\boldsymbol{X} = \boldsymbol{x}) = \frac{1}{n} \sum_{i=1}^{n} I(\boldsymbol{x}_i = \boldsymbol{x}), \tag{1.17}$$

$$\hat{f}(x_1, x_2) = P(X_1 = x_1, X_2 = x_2) = \sum_{i=1}^{n} I(x_{i1} = x_1, x_{i2} = x_2),$$

где $\boldsymbol{x} = (x_1, x_2)^T$, и I – переменная-индикатор, принимающая значение 1 только когда аргумент истинный:

$$I(\boldsymbol{x_i} = \boldsymbol{x}) = \begin{cases} 1, & \text{если } x_{i1} = x_1 \text{ и } x_{i2} = x_2 \\ 0, & \text{иначе.} \end{cases}$$

Как и в однофакторном случае, функция передает вероятность $\frac{1}{n}$ на каждую точку в выборе.

1.2.2 Математическое ожидание и дисперсия

Среднее значение

Двухфакторное среднее значение определено как математическое ожидание случайной величины $\boldsymbol{X}$:

$$\boldsymbol{\mu} = E[\boldsymbol{X}] = E\left[\begin{pmatrix} X_1 \\ X_2 \end{pmatrix}\right] = E\begin{pmatrix} E[X_1] \\ E[X_2] \end{pmatrix} = \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix} \tag{1.18}$$

Другими словами, среднее – вектор из математических ожиданий по каждому атрибуту.

Вектор выборочного математического ожидания может быть получен из $\hat{f}_{X_1}$ и $\hat{f}_{X_2}$- эмпирических функций распределения вероятностей величин X_1 и X_2 по уравнению (1.5). Также его можно получить из уравнения (1.17).

$$\hat{\boldsymbol{\mu}} = \sum_{\boldsymbol{x}} \boldsymbol{x}\hat{f}(\boldsymbol{x}) = \sum_{\boldsymbol{x}} \boldsymbol{x}\left(\frac{1}{n}\sum_{i=1}^{n} I(\boldsymbol{x}_i = \boldsymbol{x})\right) = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{x}_i \tag{1.19}$$

Дисперсия

Используя (1.8), вычислим σ_1^2 – дисперсию по атрибуту X_1, σ_2^2 – по атрибуту X_2. Тогда общая *дисперсия* будет определена как:

$$\text{var}(\boldsymbol{D}) = \sigma_1^2 + \sigma_2^2$$

Аналогично, для выборки $\hat{\sigma}_1^2$ и $\hat{\sigma}_2^2$ могут быть установлены через (1.10), и *общая дисперсия выборки* есть $\hat{\sigma}_1^2 + \hat{\sigma}_2^2$.

1.2.3 Меры зависимости

Ковариантность

Ковариантность между атрибутам X_1 и X_2 показывает меру ассоциации или линейной зависимости между ними и определена как:

$$\sigma_{12} = E[(X_1 - \mu_1)(X_2 - \mu_2)] \tag{1.20}$$

Исходя из линейности отношения, имеем:

$$\begin{aligned} \sigma_{12} &= E[(X_1 - \mu_1)(X_2 - \mu_2)] \\ &= E[X_1X_2 - X_1\mu_2 - X_2\mu_1 + \mu_1\mu_2] \\ &= E[X_1X_2] - \mu_2E[X_1] - \mu_1E[X_2] + \mu_1\mu_2 \\ &= E[X_1X_2] - \mu_1\mu_2 \\ &= E[X_1X_2] - E[X_1]E[X_2] \end{aligned} \tag{1.21}$$

По сути, 1.21 – обобщение однофакторной ковариантности (ур-е 1.9) на двухфакторный случай.

Если X_1 и X_2 – независимые случайные величины, то их ковариантность равна нулю. Действительно, в этом случае

$$E[X_1X_2] = E[\dot{X}_1] \cdot E[X_2],$$

что в свою очередь означает

$$\sigma_{12} = 0.$$

Однако, обратное неверно. То есть, если $\sigma_{12} = 0$, нет достаточных оснований заявлять, что X_1 и X_2 – независимы. В этом случае между величинами нет только линейной зависимости, однако может быть зависимость более высокого порядка.

Выборочная ковариантность между X_1 и X_2 определена как

$$\hat{\sigma}_{12} = \frac{1}{n}\sum_{i=1}^{n}(x_{i1} - \hat{\mu}_1)(x_{i2} - \hat{\mu}_2) \tag{1.22}$$

Это может быть выведено заменой эмпирической совместной функции распределения $\hat{f}(x_1, x_2)$ из 1.17 в 1.20 следующим образом:

$$\begin{aligned}\hat{\sigma}_{12} &= E[(X_1 - \hat{\mu}_1)(X_2 - \hat{\mu}_2)] \\ &= \sum_{\boldsymbol{x}=(x_1,x_2)^2}(x_1 - \hat{\mu}_1)(x_2 - \hat{\mu}_2)\hat{f}(x_1, x_2) \\ &= \frac{1}{n}\sum_{\boldsymbol{x}=(x_1,x_2)^T}\sum_{i=1}^{n}(x_1 - \hat{\mu}_1)(x_2 - \hat{\mu}_2)\cdot I(x_{i1} = x_1, x_{i2} = x2) \\ &= \frac{1}{n}\sum_{i=1}^{n}(x_{i1} - \hat{\mu}_1)(x_{i2} - \hat{\mu}_2)\end{aligned}$$

Обратите внимание, что выборочная ковариантность есть обобщение выборочной дисперсии (1.10), поскольку

$$\hat{\sigma}_{11} = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu_1)(x_i - \mu_1) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu_1)^2 = \hat{\sigma}_1^2.$$

Аналогично, $\hat{\sigma}_{22} = \sigma_2^2$.

Корреляция

Корреляция между X_1 и X_2 – есть корреляция, нормализованная среднеквадратическим отклонением каждой переменной:

$$\rho_{12} = \frac{\sigma_{12}}{\sigma_1\sigma_2} = \frac{\sigma_{12}}{\sqrt{\sigma_1^2\sigma_2^2}} \tag{1.23}$$

Выборочная корреляция между атрибутами X_1 и X_2 определена как:

$$\hat{\rho} = \frac{\hat{\sigma}_{12}}{\hat{\sigma}_1\hat{\sigma}_2} = \frac{\sum_{i=1}^{n}(x_{i1} - \hat{\mu}_1)(x_{i2} - \hat{\mu}_2)}{\sqrt{\sum_{i=1}^{n}(x_{i1} - \hat{\mu}_1)^2\sum_{i=1}^{n}(x_{i2} - \hat{\mu}_2)^2}} \tag{1.24}$$

Геометрическая интерпретация выборочной ковариантности и корреляции

Пусть Z_1 и Z_2 обозначают центральные векторов атрибутов в $\mathbb{R}^n$, представленные как:

$$Z_1 = X_1 - \mathbf{1} \cdot \hat{\mu}_1 = \begin{pmatrix} x_{11} - \hat{\mu}_1 \\ x_{21} - \hat{\mu}_1 \\ \vdots \\ x_{n1} - \hat{\mu}_1 \end{pmatrix} \quad Z_2 = X_2 - \mathbf{1} \cdot \hat{\mu}_2 = \begin{pmatrix} x_{12} - \hat{\mu}_2 \\ x_{22} - \hat{\mu}_2 \\ \vdots \\ x_{n2} - \hat{\mu}_2 \end{pmatrix}$$

Тогда выборочная ковариантность (ур-е 1.22) может быть записана следующим образом:

$$\hat{\sigma}_{12} = \frac{Z_1^T Z_2}{n}$$

Другими словами, ковариантность между двумя атрибутами есть векторное произведение центральных векторов атрибутов, нормализованных по размеру выборки. Это обобщение однофакторной выборочной ковариантности (ур-е 1.11).

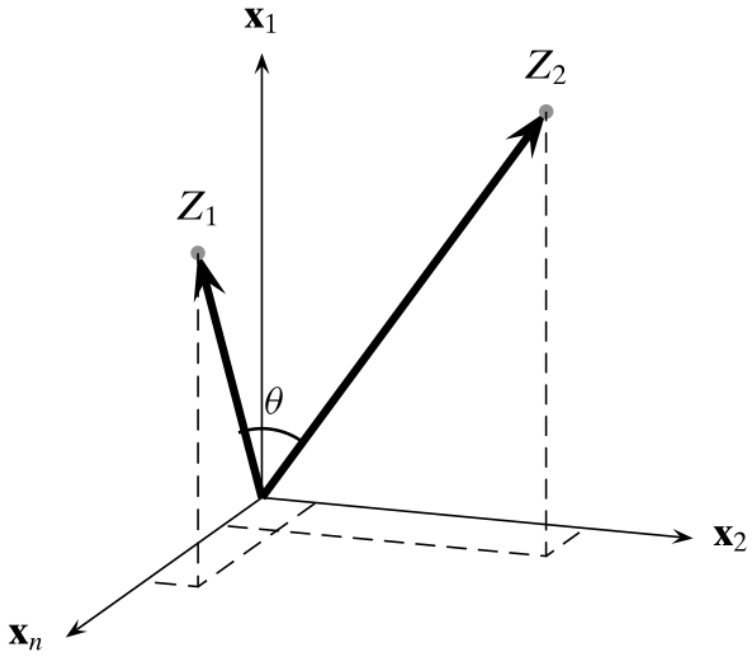


Рисунок 1.1. Геометрическая интерпретация ковариантности и корреляции. Два центральных вектора атрибутов концептуально показаны в n-мерном пространстве $\mathbb{R}^n$, натянутые на n точек.

Выборочная ковариантность (ур-е 1.24) может быть записана как:

$$\hat{\rho}_{12} = \frac{Z_1^T Z_2}{\sqrt{Z_1^T Z_1}\sqrt{Z_2^T Z_2}} = \frac{Z_1^T Z_2}{\|Z_1\| \, \|Z_2\|} = \left(\frac{Z_1}{\|Z_1\|}\right)^T \left(\frac{Z_2}{\|Z_1\|}\right) = \cos\theta \tag{1.25}$$

Таким образом, коэффициент корреляции есть косинус угла между двумя центральными векторами атрибутов, как показано на рис. 1.1.

Матрица ковариантности

Информация о дисперсии и ковариантности атрибутов X_1 и X_2 может быть обобщена в *матрицу ковариантности*:

$$\begin{aligned}\boldsymbol{\Sigma} &= E[(\boldsymbol{X}-\boldsymbol{\mu})(\boldsymbol{X}-\boldsymbol{\mu})^T] \\ &= E\left[\begin{pmatrix}X_1-\mu_1\\X_2-\mu_2\end{pmatrix}(X_1-\mu_1 \quad X_2-\mu_2)\right] \\ &= \begin{pmatrix}E[(X_1-\mu_1)(X_1-\mu_1)] & E[(X_1-\mu_1)(X_2-\mu_2)]\\E[(X_2-\mu_2)(X_1-\mu_1)] & E[(X_2-\mu_2)(X_2-\mu_2)]\end{pmatrix} \\ &= \begin{pmatrix}\sigma_1^2 & \sigma_{12}\\\sigma_{21} & \sigma_2^2\end{pmatrix}\end{aligned} \tag{1.26}$$

Поскольку $\sigma_{12} = \sigma_{21}$, $\boldsymbol{\Sigma}$ – *симметричная* матрица. На главной диагонали матрица содержит дисперсии атрибутов, в недиагональных элементах – информацию о ковариантности.

Общая дисперсия двух атрибутов – сумма главной диагонали $\boldsymbol{\Sigma}$ или *след* $\boldsymbol{\Sigma}$*:*

$$var(\boldsymbol{D}) = tr(\boldsymbol{\Sigma}) = \sigma_1^2 + \sigma_2^2$$

Из этого немедленно следует, что $\mathrm{tr}(\boldsymbol{\Sigma}) \geq 0$.

Обобщенная дисперсия двух атрибутов учитывает также ковариантность (в дополнение к дисперсии) и задана как определитель матрицы ковариантности $\boldsymbol{\Sigma}$. Обозначается как $|\boldsymbol{\Sigma}|$ или $\det(\Sigma)$. Обобщенная дисперсия неотрицательна, поскольку

$$|\boldsymbol{\Sigma}| = \det(\boldsymbol{\Sigma}) = \sigma_1^2\sigma_2^2 - \sigma_{12}^2 = \sigma_1^2\sigma_2^2 - \rho_{12}^2\sigma_1^2\sigma_2^2 = (1-\rho_{12}^2)\sigma_1^2\sigma_2^2$$

где из уравнения (1.23) $\sigma_{12} = \rho\sigma_1\sigma_2$. Обратите внимание, что т.к. $|\rho_{12}| \leq 1$, то и $\rho_{12}^2 \leq 1$, следовательно $\det(\boldsymbol{\Sigma}) \geq 1$.

Выборочная матрица ковариантности представлена в виде

$$\hat{\boldsymbol{\Sigma}} = \begin{pmatrix}\hat{\sigma}_1^2 & \hat{\sigma}_{12}\\\hat{\sigma}_{21} & \hat{\sigma}_2^2\end{pmatrix}$$

Выборочная матрица ковариантности $\hat{\boldsymbol{\Sigma}}$ имеет те же свойства, что и $\boldsymbol{\Sigma}$, т.е. симметричность, $|\hat{\boldsymbol{\Sigma}}| \geq 0$ и тоже может использоваться для получения выборочной общей и обобщенной дисперсии

1.3 Многофакторный анализ

В многофакторном анализе рассматриваются все d числовых атрибутов $X_1, X_2, \ldots, X_d$. Полные данные представляют собой матрицу размера $n \times d$, заданную как

$$\boldsymbol{D} = \begin{pmatrix}\underline{X_1} & \underline{X_2} & \ldots & \underline{X_d}\\x_{11} & x_{12} & \ldots & x_{1d}\\x_{21} & x_{22} & \ldots & x_{2d}\\\vdots & \vdots & \ddots & \vdots\\x_{n1} & x_{n2} & \ldots & x_{nd}\end{pmatrix}$$

Данные в строке могут быть представлены как набор из n точек или векторов в d-мерном пространстве атрибутов

$$\boldsymbol{x}_i = (x_{i1}, x_{i2}, \ldots, x_{id})^T \in \mathbb{R}^d$$

Данные в столбце могут быть представлены как набор из d точек или векторов в n-мерном пространстве, охватываемом точками данных

$$X_j = (x_{1j}, x_{2j}, \dots, x_{nj})^T \in \mathbb{R}^n$$

С вероятностной точки зрения атрибуты d моделируются как векторная случайная величина, $\boldsymbol{X} = (X_1, X_2, \dots, X_d)^T$, а точки x_i считаются случайной выборкой из $\mathbf{X}$, т.е. они независимы и распределены так же, как $\boldsymbol{X}$.

Среднее значение

Обобщая уравнение (1.18), *вектор многомерного среднего* получается путем взятия среднего значения каждого атрибута, заданного как

$$\mu = E[\mathbf{X}] = \begin{pmatrix} E[X_1] \\ E[X_2] \\ \vdots \\ E[X_d] \end{pmatrix} = \begin{pmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_d \end{pmatrix}$$

Обобщая уравнение (1.19), *выборочное среднее значение* берется как

$$\widehat{\boldsymbol{\mu}} = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{x}_i$$

Ковариационная матрица

Обобщая уравнение (1.26) до d измерений, информация о многомерной ковариации находится внутри симметричной ковариационной матрицы $d \times d$, которая дает ковариацию для каждой пары атрибутов:

$$\boldsymbol{\Sigma} = E[(\boldsymbol{X} - \boldsymbol{\mu})(\boldsymbol{X} - \boldsymbol{\mu})^T] = \begin{pmatrix} \sigma_1^2 & \sigma_{12} & \dots & \sigma_{1d} \\ \sigma_{21} & \sigma_2^2 & \dots & \sigma_{2d} \\ \dots & \dots & \dots & \dots \\ \sigma_{d1} & \sigma_{d2} & \dots & \sigma_d^2 \end{pmatrix}$$

Диагональный элемент σ_i^2 определяет дисперсию атрибутов для X_i, тогда как недиагональные элементы $\sigma_{ij} = \sigma_{ji}$ представляют собой ковариацию между парами атрибутов X_i и X_j.

Ковариационная матрица неотрицательна

Стоит отметить, что $\boldsymbol{\Sigma}$ – неотрицательная матрица, т.е.,

$\boldsymbol{a}^T\boldsymbol{\Sigma}\mathbf{a} \geq 0$, для любого d-мерного вектора $\boldsymbol{a}$

Чтобы увидеть это, рассмотрим следующее

$$\boldsymbol{a}^T\boldsymbol{\Sigma}\mathbf{a} = \boldsymbol{a}^T E[(\boldsymbol{X} - \boldsymbol{\mu})(\boldsymbol{X} - \boldsymbol{\mu})^T]\mathbf{a}$$
$$= E[\boldsymbol{a}^T(\boldsymbol{X} - \boldsymbol{\mu})(\boldsymbol{X} - \boldsymbol{\mu})^T\mathbf{a}]$$
$$= E[Y^2]$$

$$\geq 0$$

Где Y – случайная переменная $Y = \boldsymbol{a}^T(\boldsymbol{X} - \boldsymbol{\mu}) = \sum_{i=1}^{d} a_i(X_i - \mu_i)$, и мы используем тот факт, что математическое ожидание квадрата случайной величины неотрицательно.

$\boldsymbol{\Sigma}$ также симметрично – это означает, что все собственные значения $\boldsymbol{\Sigma}$ являются вещественными и неотрицательными. Другими словами, d собственных значений $\boldsymbol{\Sigma}$ могут быть расположены от наибольшего к наименьшему следующим образом: $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d \geq 0$. Как следствие, детерминант $\boldsymbol{\Sigma}$ неотрицателен:

$$\det(\boldsymbol{\Sigma}) = \prod_{i=1}^{d} \lambda_i \geq 0 \tag{1.27}$$

Общая и обобщенная дисперсия

Общая дисперсия задается как след ковариационной матрицы:

$$var(\boldsymbol{D}) = tr(\boldsymbol{\Sigma}) = \sigma_1^2 + \sigma_2^2 + \cdots + \sigma_d^2 \tag{1.28}$$

Будучи суммой квадратов, общая дисперсия должна быть неотрицательной.

Обобщенная дисперсия определяется как определитель ковариационной матрицы $\det(\boldsymbol{\Sigma})$, также обозначаемый, как $|\boldsymbol{\Sigma}|$. Она дает единственное значение для всего многомерного разброса. Из уравнения (1.27) имеем $\det(\boldsymbol{\Sigma}) \geq 0$.

Выборочная ковариационная матрица

Выборочная ковариационная матрица задается как

$$\widehat{\boldsymbol{\Sigma}} = E[(\boldsymbol{X} - \widehat{\boldsymbol{\mu}})(\boldsymbol{X} - \widehat{\boldsymbol{\mu}})^T] = \begin{pmatrix} \hat{\sigma}_1^2 & \hat{\sigma}_{12} & \dots & \hat{\sigma}_{1d} \\ \hat{\sigma}_{21} & \hat{\sigma}_2^2 & \dots & \hat{\sigma}_{2d} \\ \dots & \dots & \dots & \dots \\ \hat{\sigma}_{d1} & \hat{\sigma}_{d2} & \dots & \hat{\sigma}_d^2 \end{pmatrix} \tag{1.29}$$

Вместо поэтапного вычисления выборочной ковариационной матрицы мы можем получить ее с помощью матричных операций. Пусть $\boldsymbol{Z}$ представляет центрированную матрицу данных, заданную как матрицу центрированных векторов атрибутов $Z_i = X_i - \mathbf{1} \cdot \hat{\mu}_i$, где $\mathbf{1} \in \mathbb{R}^n$:

$$\boldsymbol{Z} = \boldsymbol{D} - \mathbf{1} \cdot \widehat{\boldsymbol{\mu}}^T = \begin{pmatrix} | & | & & | \\ Z_1 & Z_2 & \dots & Z_d \\ | & | & & | \end{pmatrix}$$

В качестве альтернативы центрированная матрица данных также может быть записана в терминах центрированных точек $\boldsymbol{z}_i = \boldsymbol{x}_i - \widehat{\boldsymbol{\mu}}$:

$$\boldsymbol{Z} = \boldsymbol{D} - \mathbf{1} \cdot \widehat{\boldsymbol{\mu}}^T = \begin{pmatrix} x_1^T - \widehat{\boldsymbol{\mu}}^T \\ x_2^T - \widehat{\boldsymbol{\mu}}^T \\ \vdots \\ x_n^T - \widehat{\boldsymbol{\mu}}^T \end{pmatrix} = \begin{pmatrix} - & z_1^T & - \\ - & z_2^T & - \\ & \vdots & \\ - & z_n^T & - \end{pmatrix}$$

В матричной записи выборочная ковариационная матрица может быть записана как

$$\widehat{\boldsymbol{\Sigma}} = \frac{1}{n}[\boldsymbol{Z}^T\boldsymbol{Z}] = \frac{1}{n}\begin{pmatrix} Z_1^T Z_1 & Z_1^T Z_2 & \dots & Z_1^T Z_d \\ Z_2^T Z_1 & Z_2^T Z_2 & \dots & Z_2^T Z_d \\ \dots & \dots & \ddots & \dots \\ Z_d^T Z_1 & Z_d^T Z_2 & \dots & Z_d^T Z_d \end{pmatrix} \quad (1.30)$$

Таким образом выборочная ковариационная матрица задается как попарные *внутренние или скалярные произведения* центрированных векторов атрибутов, нормализованные на размер выборки.

В терминах центрированных точек $\boldsymbol{z}_i$, выборочная ковариационная матрица также может быть записана как сумма матриц первого ранга, полученных как внешнее произведение каждой центрированной точки:

$$\widehat{\boldsymbol{\Sigma}} = \frac{1}{n}\sum_{i=1}^{n} z_i \cdot z_i^T \quad (1.31)$$

1.4. Нормальная форма

При анализе двух и более атрибутов часто бывает необходимо нормализовать значения атрибутов, особенно в тех случаях, когда значения сильно различаются по масштабу.

Нормализация диапазона

Пусть X - атрибут и $x_1, x_2, \dots x_n$ - случайная выборка из X. При нормализации диапазона каждое значение масштабируется на диапазон выборки $\hat{r}$ of X:

$$x_i' = \frac{x_i - min_i\{x_i\}}{\hat{r}} = \frac{x_i - min_i\{x_i\}}{max_i\{x_i\} - min_i\{x_i\}}$$

После преобразования новый атрибут принимает значения в диапазоне [0, 1].

Стандартная оценочная нормализация

В стандартной оценочной нормализации, называемой также z-нормализацией, каждое значение заменяется его z-оценкой:

$$x_i' = \frac{x_i - \hat{\mu}}{\hat{\sigma}}$$

где $\hat{\mu}$ - выборочное мат. ожидание и $\widehat{\sigma^2}$ - выборочная дисперсия X. После трансформации мат. ожидание новой величины станет $\hat{\mu}' = 0$, среднеквадратическое отклонение - $\hat{\sigma}' = 1$.

Пример 1.1.

Рассмотрим пример набора данных, предоставленный в таблице 1.1. Атрибуты *Возраст* и *Доход* имеют очень разные масштабы, причем последний имеет гораздо большие значения.

Таблица 1.1. Набор данных для нормализации

x_i	*Возраст* (X_1)	*Доход* (X_2)

x_1	12	300
x_2	14	500
x_3	18	1000
x_4	23	2000
x_5	27	3500
x_6	28	4000
x_7	34	4300
x_8	37	6000
x_9	39	2500
x_{10}	40	2700

Посчитаем расстояние между x_1 и x_2:

$$\|x_1 - x_2\| = \|(2, 200)^T\| = \sqrt{2^2 + 200^2} = \sqrt{40004} = 200.01$$

Как мы можем заметить, вклад атрибута *Возраст* затмевается атрибутом *Доход*.

Диапазон выборки для *Возраста* составляет $\hat{r} = 40 - 12 = 28$, с минимальным значением равным 12. После нормализации диапазона, новый атрибут задается как:

$$\text{Возраст}' = (0, 0.071, 0.214, 0.393, 0.536, 0.571, 0.786, 0.893, 0.964, 1)^T$$

Например, для точки $x_2 = (x_{21}, x_{22}) = (14, 500)$ значение $x_{21} = 14$ преобразуется в

$$x'_{21} = \frac{14 - 12}{28} = \frac{2}{28} = 0.071$$

Аналогичным образом, диапазон выборки для атрибута *Доход* составляет 6000 - 300 = 5700 с минимальным значением 300; таким образом, *Доход* преобразуется в

$$\text{Доход}' = (0, 0.035, 0.123, 0.298, 0.561, 0.649, 0.702, 1, 0.386, 0.421)^T$$

так что $x_{22} = 0.035$. Расстояние между x_1 и x_2 после нормализации диапазона представляется как

$$\|x'_1 - x'_2\| = \|(0, 0)^T - (0.071, 0.035)^T\| = \|(-0.071, -0.035)^T\| = 0.079$$

Мы можем заметить, что *Доход* больше не искажает расстояние.

Для z-нормализации сначала вычислим мат. ожидание среднеквадратическое отклонение обоих атрибутов:

	Возраст	*Доход*
$\hat{\mu}$	27.2	2680
$\hat{\sigma}$	9.77	1726.15

Возраст переходит в

$$\text{Возраст}' = (-1.56, -1.35, -0.94, -0.43, -0.02, 0.08, 0.70, 1.0, 1.21, 1.31)^T$$

Например, значение $x_{21} = 14$ для точки$x_2 = (x_{21}, x_{22}) = (14, 500)$ преобразуется в

$$x'_{21} = \frac{14 - 27.2}{9.77} = -1.35$$

Аналогичным образом для *Дохода*

$$\text{Доход}' = (-1.38, -1.26, -0.97, -0.39, 0.48, 0.77, 0.94, 1.92, -0.10, 0.01)^T$$

так что $x_{22} = -1.26$. Расстояние между x_1 и x_2 после z-нормализации представляется как

$$\|x'_1 - x'_2\| = \|(-1.56, -1.38)^T - (1.35, -1.26)^T\| = \|(-0.18, -0.12)^T\| = 0.216$$

1.5 Нормальное распределение

Нормальное распределение это одна из наиболее важных функций плотности вероятности, особенно, потому что многие физические величины примерно описываются нормальным распределением. Кроме того, выборочное распределение среднего подчиняется нормальному распределению. Нормальное распределение также играет важную роль в качестве параметрического распределения выбора в кластеризации, оценке плотности и классификации.

1.5.1 Одномерное Нормальное Распределение

Случайная величина X имеет нормальное распределение с двумя параметрами – математическое ожидание μ и дисперсия σ^2, если функция плотности вероятности X задаётся следующим образом:

$$f(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{(x-\mu)^2}{2\sigma^2}\right\}$$

Выражение $(x-\mu)^2$ показывает расстояние от величины x до математического ожидания μ, таким образом, плотность вероятности экспоненциально уменьшается в зависимости от расстояния до математического ожидания. Единственный максимум кривой плотности распределения приходится на значение математического ожидания $x = \mu$ и задаётся

как $f(\mu) = \frac{1}{\sqrt{2\pi\sigma^2}}$, что обратно пропорционально среднеквадратичному (стандартному) отклонению σ распределения.

На рисунке 1.2 изображено стандартное нормальное распределение, имеющее параметры математическое ожидание $\mu = 0$ и дисперсия $\sigma^2 = 1$. Нормальное распределение имеет характерную форму колокола, симметричного относительно математического ожидания. Рисунок также демонстрирует эффект от различных значений среднеквадратичного отклонения на форме распределения. Меньшее значение (например, $\sigma = 0.5$) приводит к более «острому» распределению, которое резко возрастает и спадает, в то время как более крупное значение (например, $\sigma = 2$) даёт более плоское распределение, которое дольше возрастает и дольше спадает. Поскольку нормальное распределение симметричное, математическое ожидание μ является также медианой и модой распределения.

Вероятностная Масса

При заданном интервале $[a, b]$ вероятностная масса нормального распределения в пределах этого интервала задается в следующем виде:

$$P(a \leq x \leq b) = \int_a^b f(x|\mu, \sigma^2)\, dx$$

В частности, нас часто интересует вероятностная масса, сосредоточенная в пределах k стандартных отклонений σ от математического ожидания μ, то есть для интервала $[\mu - k\sigma, \mu + k\sigma]$:

$$P(\mu - k\sigma \leq x \leq \mu + k\sigma) = \frac{1}{\sqrt{2\pi}\sigma} \int_{\mu-k\sigma}^{\mu+k\sigma} \exp\left\{-\frac{(x-\mu)^2}{2\sigma^2}\right\} dx$$

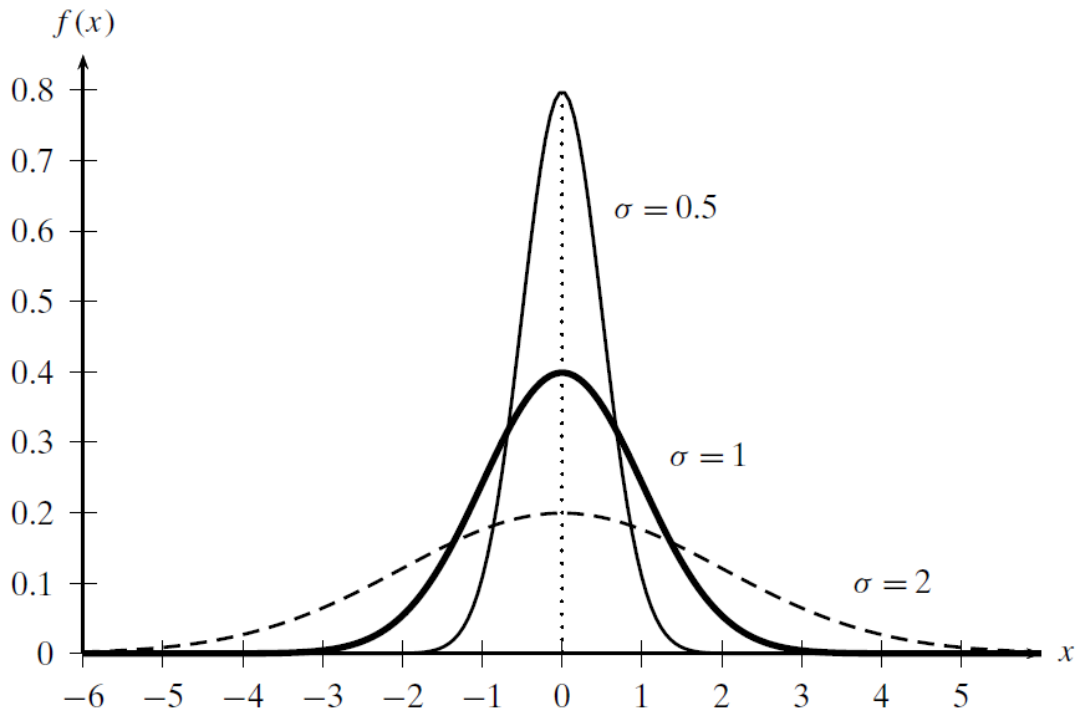


Рисунок 1.2. Нормальное распределение: $\mu = 0$, различные среднеквадратичные отклонения

С помощью изменения переменной $z = \frac{x-\mu}{\sigma}$, получаем эквивалентное выражение в терминах стандартного нормального распределения:

$$P(-k \le z \le k) = \frac{1}{\sqrt{2\pi}\sigma} \int_{-k}^{k} e^{-\frac{1}{2}z^2}\, dz = \frac{2}{\sqrt{2\pi}\sigma} \int_{0}^{k} e^{-\frac{1}{2}z^2}\, dz$$

Последнее преобразование вытекает из того факта, что $e^{-\frac{1}{2}z^2}$ – симметричная функция, следовательно, интегрирование в пределах $[-k, k]$ эквивалентно двукратному интегрированию в интервале $[0, k]$. В итоге, преобразовав ещё одну переменную $t = \frac{z}{\sqrt{2}}$, получаем:

$$P(-k \le z \le k) = 2 \cdot P\left(0 \le t \le \frac{k}{\sqrt{2}}\right) = \frac{2}{\sqrt{\pi}} \int_{0}^{\frac{k}{\sqrt{2}}} e^{-t^2}\, dt = \text{erf}\,(k/\sqrt{2}) \qquad (1.32)$$

где erf – *функция ошибки Гаусса*, определяемая как:

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_{0}^{x} e^{-t^2}\, dt$$

Пользуясь (1.32), мы можем рассчитать вероятностную массу в пределах k стандартных отклонений от математического ожидания. В частности, для $k = 1$:

$$P(\mu - \sigma \leq x \leq \mu + \sigma) = \text{erf}\left(\frac{1}{\sqrt{2}}\right) = 0.6827$$

что означает, что 68.27% всех точек лежат в пределах 1 стандартного отклонения от математического ожидания. Для $k = 2$ имеем $\text{erf}\left(\frac{2}{\sqrt{2}}\right) = 0.9545$, для $k = 3$ имеем $\text{erf}\left(\frac{3}{\sqrt{2}}\right) = 0.9973$. Таким образом, почти вся вероятностная масса нормального распределения находится в пределах $\pm 3\sigma$ от математического ожидания μ.

1.5.2 Многомерное Нормальное Распределение

Имеется d-мерный вектор случайных величин $\boldsymbol{X} = (X_1, X_2, \dots, X_d)^T$, мы говорим, что $\boldsymbol{X}$ имеет многомерное нормальное распределение (с параметрами математическое ожидание $\boldsymbol{\mu}$ и ковариационная матрица $\boldsymbol{\Sigma}$), если его совместная многомерная функция плотности вероятности задается следующим образом

$$f(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\left(\sqrt{2\pi}\right)^d \sqrt{|\boldsymbol{\Sigma}|}} \exp\left\{-\frac{(\boldsymbol{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{x} - \boldsymbol{\mu})}{2}\right\} \tag{1.33}$$

где $|\boldsymbol{\Sigma}|$ – определитель (детерминант) ковариационной матрицы. Как и в одномерном случае, выражение

$$(\boldsymbol{x_i} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{x_i} - \boldsymbol{\mu}) \tag{1.34}$$

определяет расстояние, называемое *расстоянием Махаланобиса*, между точкой $\boldsymbol{x}$ и математическим ожиданием $\boldsymbol{\mu}$, учитывающее всю дисперсионно-ковариационную информацию между переменными. Расстояние Махаланобиса является обобщением Евклидова расстояния, потому что если установить $\boldsymbol{\Sigma} = \boldsymbol{I}$, где $\boldsymbol{I}$ – единичная матрица $d \times d$ (то есть элементы главной диагонали равны 1, остальные – 0), то получится:

$$(\boldsymbol{x_i} - \boldsymbol{\mu})^T \boldsymbol{I^{-1}} (\boldsymbol{x_i} - \boldsymbol{\mu}) = \left|\left|\boldsymbol{x_i} - \boldsymbol{\mu}\right|\right|^2$$

Таким образом, Евклидово расстояние не учитывает информацию о ковариации между случайными величинами, в то время как расстояние Махаланобиса явно её учитывает.

Стандартное многомерное нормальное распределение имеет параметры $\boldsymbol{\mu} = 0$ и $\boldsymbol{\Sigma} = \boldsymbol{I}$. Рисунок 1.3 (а) демонстрирует плотность вероятности стандартного двумерного ($d = 2$) нормального распределения с параметрами

$$\boldsymbol{\mu} = 0 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

и

$$\boldsymbol{\Sigma} = \boldsymbol{I} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Это соответствует случаю, при котором обе случайные величины независимы и обе имеют стандартное нормальное распределение. Симметричная природа стандартного нормального распределения чётко показана на контурном изображении на рисунке 1.3 (b). Каждый кривая уровня представляет собой набор точек x с фиксированным значением плотности $f(x)$.

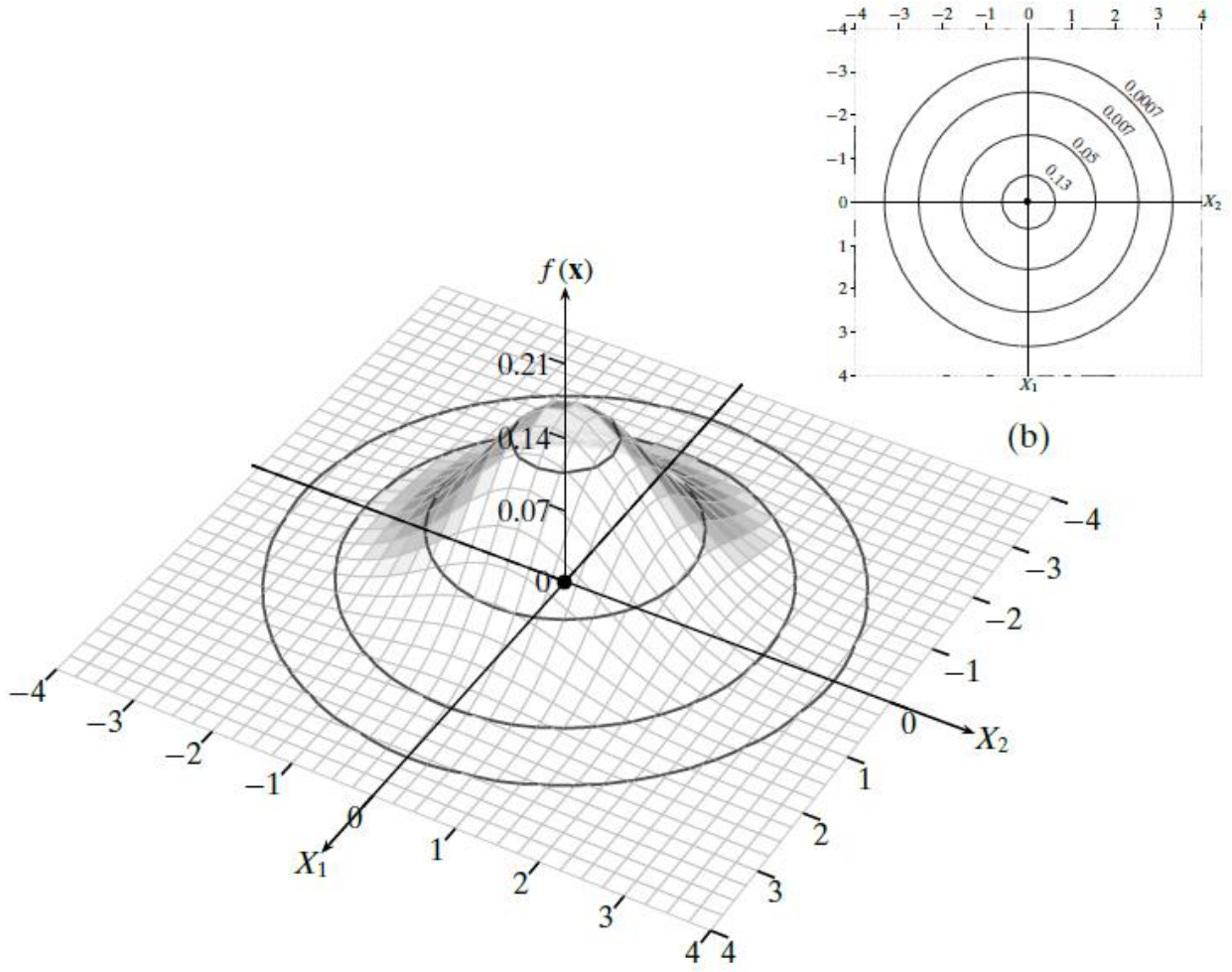


Рисунок 1.3. (a) Стандартное двумерное нормальное распределение и (b) контурное изображение. Параметры: $\boldsymbol{\mu} = (0,0)^T$, $\boldsymbol{\Sigma} = \boldsymbol{I}$

Геометрия Многомерного Нормального Распределения

Рассмотрим геометрию многомерного нормального распределения для произвольного математического ожидания $\boldsymbol{\mu}$ и ковариационной матрицы $\boldsymbol{\Sigma}$. По сравнению со стандартным нормальным распределением, можно ожидать, что контуры плотности будут смещены, масштабированы и повернуты. Сдвиг или перенос происходит из-за того, что математическое ожидание $\boldsymbol{\mu}$ не обязательно является началом координат $\mathbf{0}$. Масштабирование или искажение являются результатом дисперсий случайных величин, а вращение – результатом ковариаций.

Форма или геометрия нормального распределения становится понятной при рассмотрении спектрального разложения ковариационной матрицы. Напомним, что $\boldsymbol{\Sigma}$ – симметричная положительная полуопределённая (диагональные элементы и главные

определители неотрицательны) матрица $d \times d$. Уравнение собственного вектора для $\mathbf{\Sigma}$ выглядит следующим образом:

$$\mathbf{\Sigma} \boldsymbol{u_i} = \lambda_i \boldsymbol{u_i}$$

Здесь λ_i – собственное значение матрицы $\mathbf{\Sigma}$, вектор $\boldsymbol{u_i} \in \mathbb{R}^d$ – собственный вектор, соответствующий λ_i. Поскольку $\mathbf{\Sigma}$ – симметричная положительная полуопределённая матрица, она содержит d вещественных и неотрицательных собственных значений, которые могут быть расположены в порядке от самого большого до самого малого: $\lambda_1 \geq \lambda_2 \geq \cdots \lambda_d \geq 0$. Диагональная матрица Λ используется для записи данных собственных значений:

$$\Lambda = \begin{pmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_d \end{pmatrix}$$

Затем, собственные векторы являются единичными (нормальными) и взаимно ортогональными, то есть они ортонормированы:

$$\boldsymbol{u_i^T u_i} = 1 \quad \text{для всех } i$$

$$\boldsymbol{u_i^T u_j} = 0 \quad \text{для всех } i \neq j$$

Собственные вектора могут быть объединены в ортогональной матрице $\boldsymbol{U}$, определяемой как матрица с нормальными и взаимно ортогональными столбцами:

$$\boldsymbol{U} = \begin{pmatrix} | & | & & | \\ \boldsymbol{u_1} & \boldsymbol{u_2} & \cdots & \boldsymbol{u_d} \\ | & | & & | \end{pmatrix}$$

Спектральное разложение матрицы $\mathbf{\Sigma}$ может быть компактно представлено следующим образом:

$$\mathbf{\Sigma} = \boldsymbol{U \Lambda U^T}$$

Данное равенство может быть геометрически интерпретировано как изменение в базисных векторах. Из исходных d измерений, соответствующих d случайным величинам X_j, получаем d новых измерений $\boldsymbol{u_i}$, ковариационную матрицу $\mathbf{\Sigma}$ в исходном пространстве, в то время как ковариационная матрица $\mathbf{\Lambda}$ находится в новом пространстве координат. Так как $\mathbf{\Lambda}$ – диагональная матрица, мы можем сразу сделать вывод, что после преобразования каждое новое измерение $\boldsymbol{u_i}$ имеет дисперсию λ_i, а все ковариации равны нулю. Другими словами, в новом пространстве нормальное распределение выровнено по осям (отсутствует компонент вращения), однако искажено по каждой оси пропорционально собственным значениям λ_i, которые представляют собой дисперсию вдоль данного измерения.

Общая и Обобщённая Дисперсии

Определитель ковариационной матрицы представлен как $\det(\mathbf{\Sigma}) = \prod_{i=1}^{d} \lambda_i$. Таким образом, обобщённая дисперсия матрицы $\mathbf{\Sigma}$ – произведение её собственных значений.

Учитывая тот факт, что след квадратной матрицы инвариантен к подобным преобразованиям, таким как изменение базисов, можем сделать вывод, что общая дисперсия $var(\boldsymbol{D})$ для набора данных $\boldsymbol{D}$ является инвариантной, то есть:

$$var(\boldsymbol{D}) = tr(\boldsymbol{\Sigma}) = \sum_{i=1}^{d} \sigma_i^2 = \sum_{i=1}^{d} \lambda_i = tr(\boldsymbol{\Lambda})$$

Другими словами, $\sigma_1^2 + \cdots + \sigma_d^2 = \lambda_1 + \cdots + \lambda_d$.

Глава 2. Ядерные методы

Прежде чем мы начнём анализировать данные, важно, в первую очередь, найти такое представление данных, которое облегчит дальнейший анализ. Например, из сложных данных: текста, последовательностей, изображений и т. п., – необходимо, как правило, извлечь (или сконструировать) такой набор атрибутов или признаков (фичей), чтобы мы могли представить экземпляры данных в виде многомерных векторов. То есть, имея экземпляр данных x (например, последовательность), нам нужно найти такое отображение φ, чтобы $\varphi(x)$ было векторным представлением x. Даже если данные представляют собой числовую матрицу, может потребоваться нелинейное отображение φ такое, что $\varphi(x)$ будет вектором в соответствующем многомерном пространстве нелинейных признаков, чтобы установить нелинейные соотношения между исходными признаками. Мы будем использовать термин *«входного пространства»* для пространства входных данных x, и термин *«пространство признаков»* для векторов $\varphi(x)$. Таким образом, имея набор экземпляров данных x_i и отображение φ, мы можем преобразовать их в векторы признаков $\varphi(x_i)$, которые уже позволяют анализировать сложные данные с помощью численных методов анализа.

Отображение в пространство признаков позволяет исследовать данные с помощью алгебраического и вероятностного моделирования, в то же время, итоговое пространство, как правило, имеет довольно большую размерность (вплоть до бесконечномерных пространств). Таким образом, преобразование всех входных точек в пространство признаков может быть «дорогой» или даже невозможной задачей. Из-за высоких размерностей мы сталкиваемся с проклятием размерности, более подробно освещённом в Главе 6.

Ядерные методы избегают явного преобразования всех оригиналов x входного пространства в их образы $\varphi(x)$ в пространстве признаков. Вместо этого входные объекты представляются в виде их $n \times n$ значений попарного подобия. Функция подобия, называемая *ядром*, выбирается так, чтобы она была скалярным произведением в некотором пространстве признаков высокой размерности, но не требовала явного построения $\varphi(x)$ для вычисления. Пусть $\mathfrak{I}$ – исходное пространство, которое может состоять из произвольного набора объектов, а $\mathbf{D} = \{x_i\}_{i=1}^n \subset \mathfrak{I}$ – набор данных, состоящий из n объектов входного пространства. Мы можем представить значения попарного подобия между точками $\mathbf{D}$ как *ядерную матрицу* размера $n \times n$:

$$\mathbf{K} = \begin{pmatrix} K(\mathbf{x}_1, \mathbf{x}_1) & K(\mathbf{x}_1, \mathbf{x}_2) & \cdots & K(\mathbf{x}_1, \mathbf{x}_n) \\ K(\mathbf{x}_2, \mathbf{x}_1) & K(\mathbf{x}_2, \mathbf{x}_2) & \cdots & K(\mathbf{x}_2, \mathbf{x}_n) \\ \vdots & \vdots & \ddots & \vdots \\ K(\mathbf{x}_n, \mathbf{x}_1) & K(\mathbf{x}_n, \mathbf{x}_2) & \cdots & K(\mathbf{x}_n, \mathbf{x}_n) \end{pmatrix},$$

где $K : \mathfrak{I} \times \mathfrak{I} \to \mathbb{R}$ – *ядерная функция* над любыми двумя точками входного пространства. Однако мы требуем, чтобы K была скалярным произведением в некотором пространстве признаков. То есть, для любых $\mathbf{x}_i, \mathbf{x}_j \in \mathfrak{I}$ ядерная функция должна удовлетворять условию:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j), \tag{2.1}$$

где $\phi : \mathfrak{I} \to \mathfrak{F}$ – отображение из исходного пространства $\mathfrak{I}$ в пространство признаков $\mathfrak{F}$. Интуитивно это можно понимать так, что у нас должна быть возможность вычислить скалярное произведение только по входному представлению x, не прибегая к отображению $\varphi(x)$. Очевидно, что произвольная функция не подходит на роль ядра; допустимая ядерная функция должна сохранять условие 2.1, о чём рассказано в главе 2.1.

Важно отметить, что оператор транспонирования в скалярном произведении используется только в тех случаях, когда пространство $\mathfrak{F}$ – векторное. Когда $\mathfrak{F}$ – произвольное линейное пространство со скалярным произведением, ядро записывается следующим образом: $K(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle$, однако для простоты мы будем использовать оператор транспонирования. Если $\mathfrak{F}$ – пространство со скалярным произведением (предгильбертово), следует понимать, что:

$$\phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle.$$

Мы покажем, что подход ядерных матриц применим ко многим методам анализа данных. То есть, вместо явного отображения входного пространства в пространство признаков, данные могут быть представлены в виде ядерной матрицы $\mathbf{K}$ размера $n \times n$, а весь анализ может быть выполнен над $\mathbf{K}$. Достичь этого можно с помощью так называемого *трюка с ядром*, а именно, показать, что анализ требует только скалярного произведения $\phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$ в пространстве признаков, которое можно заменить соответствующим ядром $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$, которое, в свою очередь, легко вычислить в исходном пространстве. После вычисления ядерной матрицы нам не понадобятся входные точки $\mathbf{x}_i$, поскольку все операции, использующие скалярное произведение, могут быть выполнены над ядерной матрицей $\mathbf{K}_{n \times n}$. Отсюда непосредственно следует, что для входных данных в виде числовой матрицы $\mathbf{D}$ размерности $n \times d$, для которых было применено линейное ядро, результаты анализа $\mathbf{K}$ будут эквивалентны результатам анализа $\mathbf{D}$, поскольку в анализе задействованы только скалярные произведения. Разумеется, ядерные методы дают намного больше гибкости, поскольку мы можем использовать нелинейные ядра для нелинейного же анализа, или мы можем напрямую анализировать сложные объекты без явного построения отображения $\varphi(x)$.

Ядерные методы предлагают принципиально иной взгляд на данные. Вместо представления данных в виде векторов входного пространства или пространства признаков, в рассмотрение принимаются только значения ядра между парами точек. Ядерную матрицу можно также рассматривать как взвешенную матрицу смежности для полного графа над n входными точками. Следовательно, можно говорить о сильной связи ядерных методов с анализом графов, в частности, с алгебраической теорией графов.

2.1 Ядерные матрицы

Пусть χ обозначает пространство ввода, которым может быть любой произвольный набор объектов, и пусть $\boldsymbol{D} = [\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_n}] \subset \chi$ обозначает подмножество из n объектов входного пространства. Пусть $\phi: \chi \to \mathcal{F}$ будет отображением из входного пространства в пространство признаков $\mathcal{F}$, которое наделено скалярным произведением и нормой. Пусть $K: \chi \times \chi \to \mathbb{R}$ будет функцией, которая отображает пару входных объектов на значение их скалярного произведения в пространстве признаков, то есть $K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$, и пусть $\mathbf{K}$ будет ядерной матрицей $n \times n$, соответствующей подмножеству $\mathbf{D}$.

Функция K называется **положительно полуопределённым ядром** тогда и только тогда, когда она симметрична:

$$K(x_i, x_j) = K(x_j, x_i),$$

и соответствующая ядерная матрица **K** для любого подмножества $\mathbf{D} \subset \chi$ положительно полуопределённая, то есть:

$$\mathbf{a}^T\mathbf{Ka} \geq 0, \text{для всех векторов } \mathbf{a} \in \mathbb{R}^n,$$

из чего следует, что:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} a_i a_j K(x_i, x_j) \geq 0, \text{для всех } a_i \in \mathbb{R}, i \in [1, n] \tag{2.2}$$

Сначала удостоверимся, что если $K(x_i, x_j)$ представляет скалярное произведение $\phi(x_i)^T\phi(x_j)$ в некотором пространстве признаков, то K – положительно полуопределённое ядро. Рассмотрим любой набор данных **D**, пусть $\mathbf{K} = \{K(\boldsymbol{x_i}, \boldsymbol{x_j})\}$ – соответствующая ядерная матрица. Во-первых, K – симметрично, поскольку скалярное произведение симметрично, из чего следует, что **K** – симметрична. Во-вторых, **K** – положительно полуопределённая матрица, потому что:

$$\mathbf{a}^T\mathbf{Ka} = \sum_{i=1}^{n}\sum_{j=1}^{n} a_i a_j K(x_i, x_j)$$

$$= \sum_{i=1}^{n}\sum_{j=1}^{n} a_i a_j \phi(x_i)^T\phi(x_j)$$

$$= \left(\sum_{i=1}^{n} a_i \phi(x_i)\right)^T \left(\sum_{j=1}^{n} a_j \phi(x_j)\right)$$

$$= \left\|\sum_{i=1}^{n} a_i \phi(x_i)\right\|^2 \geq 0$$

Таким образом, K – положительно полуопределённое ядро.

Теперь покажем, что если имеется положительное полуопределённое ядро $K: \chi \times \chi \to \mathbb{R}$, то оно соответствует скалярному произведению в некотором пространстве признаков $\mathcal{F}$.

2.1.1 Отображение Воспроизводящего Ядра

Для отображения воспроизводящего ядра ϕ отобразим каждую точку $\mathbf{x} \in \chi$ в функцию на пространстве функций $[f: \chi \to \mathbb{R}]$, содержащем функции, которые отображают точки из χ в $\mathbb{R}$. Алгебраически это пространство функций представляет собой абстрактное векторное пространство, в котором каждая точка является функцией. В частности, любая точка $\mathbf{x} \in \chi$ во входном пространстве отображается в следующую функцию:

$$\phi(\mathbf{x}) = K(\mathbf{x},\cdot)$$

где $\cdot$ обозначает любой аргумент в χ. То есть, каждый объект $\mathbf{x}$ во входном пространстве отображается в *точку-функцию* $\phi(\mathbf{x})$, которая, на самом деле, является функцией $K(\mathbf{x},\cdot)$, представляющей её сходство со всеми остальными точками во входном пространстве χ.

Пусть $\mathcal{F}$ – множество всех функций или точек, которые могут быть получены как линейная комбинация любого подмножества точек-функций:

$$\mathcal{F} = span\{K(\mathbf{x},\cdot)|\mathbf{x} \in \chi\} = \{\mathbf{f} = f(\cdot) = \textstyle\sum_{i=1}^{m} \alpha_i K(\mathbf{x},\cdot) \mid m \in \mathbb{N}, \alpha_i \in \mathbb{R}, \{x_1, \dots, x_m\} \subseteq \chi\}$$

Мы используем двойное обозначение $\mathbf{f}$ и $f(\cdot)$ попеременно, чтобы подчеркнуть тот факт, что каждая точка $\mathbf{f}$ в пространстве признаков – это фактически функция $f(\cdot)$. Обратите внимание, что точка-признак $\phi(\mathbf{x}) = K(\mathbf{x},\cdot)$ по определению принадлежит $\mathcal{F}$.

Пусть $\mathbf{f}, \mathbf{g} \in \mathcal{F}$ – любые две точки в пространстве признаков:

$$\mathbf{f} = f(\cdot) = \sum_{i=1}^{m_\alpha} \alpha_i K(\mathbf{x_i},\cdot) \qquad\qquad \mathbf{g} = g(\cdot) = \sum_{j=1}^{m_\beta} \beta_j K(\mathbf{x_j},\cdot)$$

Определим скалярное произведение двух точек как

$$\mathbf{f}^T\mathbf{g} = f(\cdot)^T g(\cdot) = \sum_{i=1}^{m_\alpha}\sum_{j=1}^{m_\beta} \alpha_i \beta_j K(\mathbf{x_i}, \mathbf{x_j}) \tag{2.3}$$

Подчеркнём, что обозначение $\mathbf{f}^T\mathbf{g}$ применяется только для удобства; это обозначает внутреннее произведение $\langle \mathbf{f}, \mathbf{g} \rangle$, потому что $\mathcal{F}$ – абстрактное векторное пространство с внутренним произведением, как было описано выше.

Мы можем проверить, что скалярное произведение *билинейно*, то есть линейно по обоим аргументам, потому что

$$\mathbf{f}^T\mathbf{g} = \sum_{i=1}^{m_\alpha}\sum_{j=1}^{m_\beta} \alpha_i \beta_j K(\mathbf{x_i}, \mathbf{x_j}) = \sum_{i=1}^{m_\alpha} \alpha_i g(\mathbf{x_i}) = \sum_{j=1}^{m_\beta} \beta_j f(\mathbf{x_j})$$

Поскольку K – положительно полуопределённое ядро:

$$\|\mathbf{f}\|^2 = \mathbf{f}^T\mathbf{f} = \sum_{i=1}^{m_\alpha}\sum_{j=1}^{m_\alpha} \alpha_i \alpha_j K(\mathbf{x_i}, \mathbf{x_j}) \geq 0$$

Таким образом, пространство $\mathcal{F}$ – *предгильбертово пространство*, определяемое как пространство с нормированным внутренним произведением, потому что оно наделено симметричным билинейным скалярным произведением и нормой. Добавляя предельные точки всех сходящихся последовательностей Коши, мы можем превратить $\mathcal{F}$ *в гильбертово пространство*, определяемое как предгильбертово пространство, полное относительно нормы. Однако, рассмотрение данного материала выходит за рамки этой главы.

Пространство $\mathcal{F}$ обладает так называемым *воспроизводящим свойством*, то есть мы можем вычислить функцию $f(\cdot) = \mathbf{f}$ в точке $\mathbf{x} \in \chi$, рассчитав скалярное произведение $\mathbf{f}$ и $\phi(\mathbf{x})$:

$$\mathbf{f}^T \phi(\mathbf{x}) = f(\cdot)^T K(\mathbf{x}, \cdot) = \sum_{i=1}^{m_\alpha} \alpha_i K(\mathbf{x_i}, \mathbf{x}) = f(\mathbf{x})$$

По этой причине, пространство $\mathcal{F}$ также называется *гильбертовым пространством с воспроизводящим ядром*.

Всё, что нужно сделать – показать, что $K(\mathbf{x_i}, \mathbf{x_j})$ соответствует скалярному произведению в пространстве признаков $\mathcal{F}$. Это, действительно, так, потому что, используя выражение (2.3), для любых 2 точек-признаков $\phi(\mathbf{x_i}), \phi(\mathbf{x_j}) \in \mathcal{F}$ их скалярное произведение:

$$\phi(\mathbf{x_i})^T \phi(\mathbf{x_j}) = K(\mathbf{x_i}, \cdot)^T K(\mathbf{x_j}, \cdot) = K(\mathbf{x_i}, \mathbf{x_j})$$

Отображение воспроизводящего ядра показывает, что любое положительно полуопределённое ядро соответствует скалярному произведению в некотором пространстве признаков. Это значит, что мы можем применить хорошо знакомые алгебраические и геометрические методы, чтобы понять и проанализировать данные в этих пространствах.

2.1.2 Отображение Эмпирического Ядра

Отображение воспроизводящего ядра ϕ отображает входное пространство в потенциально бесконечномерное пространство признаков. Однако, при наличии набора данных $\mathbf{D} = \{\mathbf{x}_i\}_{i=1}^n$, мы можем получить конечномерное отображение, вычисляя ядро только в точках в $\mathbf{D}$. То есть определение отображения ϕ выглядит следующим образом:

$$\phi(\mathbf{x}) = \big(K(\mathbf{x_1}, \mathbf{x}), K(\mathbf{x_2}, \mathbf{x}), \dots, K(\mathbf{x_n}, \mathbf{x})\big)^T \in \mathbb{R}^n$$

в котором каждая точка $\mathbf{x} \in \chi$ отображается в n-мерный вектор, содержащий значения ядра $\mathbf{x}$ с каждым из объектов $\mathbf{x_i} \in \mathbf{D}$. Мы можем определить скалярное произведение в пространстве признаков следующим образом:

$$\phi(\mathbf{x_i})^T \phi(\mathbf{x_j}) = \sum_{k=1}^{n} K(\mathbf{x_k}, \mathbf{x_i}) K(\mathbf{x_k}, \mathbf{x_j}) = \mathbf{K}_i^T \mathbf{K}_j \tag{2.4}$$

где $\mathbf{K}_i$ обозначает i-ый столбец $\mathbf{K}$, который совпадает с i-ой строкой $\mathbf{K}$ (рассматривается как вектор-столбец), так как $\mathbf{K}$ – симметрична. Однако для того, чтобы отображение ϕ было допустимо, мы требуем, чтобы $\phi(\mathbf{x_i})^T \phi(\mathbf{x_j}) = K(\mathbf{x_i}, \mathbf{x_j})$, поскольку выражение (2.4) явно этому не удовлетворяет. Единственное решение – замена $\mathbf{K}_i^T \mathbf{K}_j$ в выражении (2.4) на $\mathbf{K}_i^T \mathbf{A} \mathbf{K}_j$ с некоторой положительно полуопределённой матрицей $\mathbf{A}$:

$$\mathbf{K}_i^T \mathbf{A} \mathbf{K}_j = K(\mathbf{x_i}, \mathbf{x_j})$$

Если мы можем найти такую матрицу $\mathbf{A}$, это будет означать, что для всех пар отображённых точек мы имеем

$$\{\mathbf{K}_i^T \mathbf{A} \mathbf{K}_j\}_{i,j=1}^n = \{K(\mathbf{x_i}, \mathbf{x_j})\}_{i,j=1}^n,$$

что можно записать более компактно:

$$\mathbf{KAK} = \mathbf{K}$$

Это сразу говорит о том, что мы берем $\mathbf{A} = \mathbf{K}^{-1}$, (псевдо) инверсию ядерной матрицы $\mathbf{K}$. Изменённое отображение ϕ называется *отображением эмпирического ядра*:

$$\phi(\mathbf{x}) = \mathbf{K}^{-1/2} \cdot \left(K(\mathbf{x_1}, \mathbf{x}), K(\mathbf{x_2}, \mathbf{x}), \dots, K(\mathbf{x_n}, \mathbf{x})\right)^T \in \mathbb{R}^n,$$

так данное скалярное произведение даёт:

$$\phi(\mathbf{x}_i)^T \phi(\mathbf{x_j}) = \left(\mathbf{K}^{-\frac{1}{2}}\mathbf{K_i}\right)^T \left(\mathbf{K}^{-\frac{1}{2}}\mathbf{K_j}\right)$$

$$= \mathbf{K_i^T}\left(\mathbf{K}^{-\frac{1}{2}}\mathbf{K}^{-\frac{1}{2}}\right)\mathbf{K_j}$$

$$= \mathbf{K_i^T K^{-1} K_j}$$

Для всех пар отображённых точек имеем:

$$\left\{\mathbf{K}_i^T \mathbf{K^{-1}} \mathbf{K}_j\right\}_{i,j=1}^n = \mathbf{K K^{-1} K} = \mathbf{K}$$

что и хотели получить. Однако, важно отметить, что такое представление эмпирических признаков действительно только для n точек в $\mathbf{D}$. Если точки добавляются в $\mathbf{D}$ или удаляются из $\mathbf{D}$, то отображение ядра обновляется для всех точек.

2.1.3 Мерсеровское Отображение Ядра

В общем случае, для одного и того же ядра K могут быть построены различные пространства признаков. Теперь мы опишем, как построить отображение Мерсера.

2.1.3.1 Отображение Ядра При Конкретных Данных

Мерсеровское отображение ядра наиболее понятно после рассмотрения ядерных матриц для набора данных $\mathbf{D}$ во входном пространстве. Поскольку $\mathbf{K}$ – симметричная положительно полуопределённая матрица, она содержит вещественные и неотрицательные собственные значения, и может быть разложена следующим образом:

$$\mathbf{K} = \mathbf{U \Lambda U^T},$$

где $\mathbf{U}$ – ортонормированная матрица собственных векторов $\mathbf{u}_i = (u_{i1}, u_{i2}, \dots, u_{in})^T \in \mathbb{R}^n$ (при $i = 1, \dots, n$); $\mathbf{\Lambda}$ – диагональная матрица собственных значений; при этом собственные значения обеих матриц ($\mathbf{U}$ и $\mathbf{\Lambda}$) расположены в невозрастающем порядке $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n \geq 0$:

$$\mathbf{U} = \begin{pmatrix} | & | & | & | \\ \mathbf{u_1} & \mathbf{u_2} & \cdots & \boldsymbol{u_n} \\ | & | & | & | \end{pmatrix} \qquad \mathbf{\Lambda} = \begin{pmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{pmatrix}$$

Ядерная матрица $\mathbf{K}$ может быть записана как спектральная сумма:

$$\mathbf{K} = \lambda_1 \mathbf{u_1} \mathbf{u_1^T} + \lambda_2 \mathbf{u_2} \mathbf{u_2^T} + \cdots \lambda_n \mathbf{u_n} \mathbf{u_n^T}$$

В частности, функция ядра между $\mathbf{x_i}$ и $\mathbf{x_j}$ задаётся как

$$\mathbf{K}(\mathbf{x_i}, \mathbf{x_j}) = \lambda_1 u_{1i} u_{1j} + \lambda_2 u_{2i} u_{2j} + \cdots \lambda_n u_{ni} u_{nj} = \sum_{k=1}^{n} \lambda_k u_{ki} u_{kj}, \quad (2.5)$$

где u_{ki} – означает i-ый элемент собственного вектора $\mathbf{u_k}$. Отсюда следует, что если мы определим отображение Мерсера ϕ следующим образом:

$$\phi(\mathbf{x_i}) = \left(\sqrt{\lambda_1} u_{1i}, \sqrt{\lambda_2} u_{2i}, \ldots, \sqrt{\lambda_n} u_{ni}\right)^T, \quad (2.6)$$

то $\mathbf{K}(\mathbf{x_i}, \mathbf{x_j})$ – это скалярное произведение в пространстве признаков отображённых точек $\phi(\mathbf{x_i})$ и $\phi(\mathbf{x_j})$, потому что

$$\begin{aligned} \phi(\mathbf{x_i})^T \phi(\mathbf{x_j}) &= \left(\sqrt{\lambda_1} u_{1i}, \sqrt{\lambda_2} u_{2i}, \ldots, \sqrt{\lambda_n} u_{ni}\right)\left(\sqrt{\lambda_1} u_{1j}, \sqrt{\lambda_2} u_{2j}, \ldots, \sqrt{\lambda_n} u_{nj}\right)^T \\ &= \lambda_1 u_{1i} u_{1j} + \lambda_2 u_{2i} u_{2j} + \cdots \lambda_n u_{ni} u_{nj} = \mathbf{K}(\mathbf{x_i}, \mathbf{x_j}) \end{aligned}$$

Отметим, что $\mathbf{U_i} = (u_{1i}, u_{2i}, \ldots, u_{ni})^T$ – это i-ая строка $\mathbf{U}$, можем переписать отображение Мерсера ϕ таким образом:

$$\phi(\mathbf{x_i}) = \sqrt{\mathbf{\Lambda}} \mathbf{U_i} \quad (2.7)$$

Таким образом, значение ядра – это просто скалярное произведение между масштабированными строками $\mathbf{U}$:

$$\phi(\mathbf{x_i})^T \phi(\mathbf{x_j}) = \left(\sqrt{\mathbf{\Lambda}} \mathbf{U_i}\right)^T \left(\sqrt{\mathbf{\Lambda}} \mathbf{U_j}\right) = \mathbf{U_i^T \Lambda U_j}$$

Отображение Мерсера, определённое эквивалентно в выражениях (2.6) и (2.7), очевидно, ограничивается входным набором данных $\mathbf{D}$, также как и отображение эмпирического ядра, и поэтому называется *мерсеровским отображением ядра для конкретных данных*.

2.1.3.2 Мерсеровское Отображение Ядра

Для компактных непрерывных пространств, аналогично дискретному случаю в выражении (2.5), значение ядра между любыми двумя точками может быть записано в виде бесконечного спектрального разложения:

$$K(\mathbf{x_i}, \mathbf{x_j}) = \sum_{k=1}^{\infty} \lambda_k \mathbf{u_k}(\mathbf{x_i}) \mathbf{u_k}(\mathbf{x}_j),$$

где $[\lambda_1, \lambda_2, \ldots]$ – бесконечное множество собственных значений, $\{\mathbf{u_1}(\cdot), \mathbf{u_2}(\cdot), \ldots\}$ – соответствующее множество ортогональных и нормализованных *собственных функций*, то есть, каждая функция $\mathbf{u_i}(\cdot)$ – решение интегрального уравнения

$$\int K(\mathbf{x}, \mathbf{y})\, \mathbf{u}_i(\mathbf{y}) d\mathbf{y} = \lambda_i \mathbf{u}_i(\mathbf{x})$$

и K – непрерывное положительно полуопределённое ядро, то есть, для всех функций $a(\cdot)$ с конечным квадратным интегралом (например, $\int a(\mathbf{x})^2 d\mathbf{x} < \infty$), K удовлетворяет условию

$$\int\int K(\mathbf{x_1}, \mathbf{x_2}) a(\mathbf{x}_1) a(\mathbf{x}_2) d\mathbf{x}_1 d\mathbf{x}_2 \geq 0$$

Мы можем увидеть, что это положительно полуопределённое ядро для компактных непрерывных пространств является аналогичным дискретному ядру из выражения (2.2). Кроме того, аналогично отображению Мерсера на конкретных данных [выражение (2.6)], общее мерсеровское отображение ядра выражается как

$$\phi(\mathbf{x_i}) = \left(\sqrt{\lambda_1}\mathbf{u_1}(\mathbf{x_i}), \sqrt{\lambda_2}\mathbf{u_2}(\mathbf{x_i}), \ldots\right)^T$$

со значением ядра, эквивалентным скалярному произведению между двумя отображёнными точками:

$$K(\mathbf{x_i}, \mathbf{x_j}) = \phi(\mathbf{x_i})^T \phi(\mathbf{x_j}).$$

2.2. Векторные ядра

Теперь мы рассмотрим два наиболее часто используемых векторных ядра на практике. Ядра, которые отображают (входное) векторное пространство в другое векторное пространство (признаков), называются *векторными ядрами*. Для многомерных входных данных входным векторным пространством будет d-мерное вещественное пространство $\mathbb{R}^d$. Пусть $\mathbf{D}$ содержит n входных точек $\mathbf{x}_i \in \mathbb{R}^d$, для $i = 1, 2, \ldots, n$. Обычно используемые (нелинейные) ядерные функции над векторными данными включают полиномиальное и гауссово ядра, как описано ниже.

2.2.1 Полиномиальное ядро

Полиномиальные ядра бывают двух типов: однородные и неоднородные. Пусть $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$. Однородное полиномиальное ядро определяется как

$$K_q(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x})^T \phi(\mathbf{y}) = (\mathbf{x}^T \mathbf{y})^q, \quad (2.8)$$

где q - степень многочлена. Это ядро соответствует пространству признаков, охватываемому всеми произведениями ровно q атрибутов.

Наиболее типичными случаями являются линейное (q = 1) и квадратичное (q = 2) ядра, заданные как

$$K_1(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T \mathbf{y}$$

$$K_2(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T \mathbf{y})^2$$

Неоднородное полиномиальное ядро определяется как

$$K_q(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x})^T \phi(\mathbf{y}) = (c + \mathbf{x}^T \mathbf{y})^q, \quad (2.9)$$

где q - степень многочлена, а $c \geq 0$ – некоторая постоянная. При $c = 0$ получаем однородное ядро. Когда $c > 0$, это ядро соответствует пространству признаков, охватываемому всеми произведениями не более чем q атрибутов. Это видно из биномиального разложения

$$K_q(\mathbf{x}, \mathbf{y}) = (c + \mathbf{x}^T\mathbf{y})^{\,q} = \sum_{k=0}^{q} \binom{q}{k} c^{k-q} (\mathbf{x}^T\mathbf{y})^k$$

Например, для типичного значения $\mathrm{c} = 1$ неоднородное ядро представляет собой взвешенную сумму однородных полиномиальных ядер для всех степеней до q, то есть

$$(1 + \mathbf{x}^T\mathbf{y})^{\,q} = 1 + q\mathbf{x}^T\mathbf{y} + \binom{q}{2} (\mathbf{x}^T\mathbf{y})^2 + \cdots + q(\mathbf{x}^T\mathbf{y})^{q-1} + (\mathbf{x}^T\mathbf{y})^q$$

Для полиномиального ядра можно построить отображение φ из входа в пространство признаков. Пусть $n_0, n_1, \ldots, n_d$ обозначают неотрицательные целые числа, такие что $\sum_{i=0}^{d} n_i = q$. Далее, пусть $\mathbf{n} = (n_0, n_1, \ldots, n_d)$, и пусть $|\mathbf{n}| = \sum_{i=0}^{d} n_i = q$. Кроме того, пусть $\binom{q}{\mathbf{n}}$ обозначает полиномиальный коэффициент

$$\binom{q}{\mathbf{n}} = \binom{q}{n_0, n_1, \ldots, n_d} = \frac{q!}{n_0!\, n_1! \ldots n_d!}$$

Тогда полиномиальное разложение неоднородного ядра имеет вид

$$\begin{aligned}
K_q(\mathbf{x}, \mathbf{y}) = (c + \mathbf{x}^T\mathbf{y})^q &= \left(c + \sum_{k=1}^{d} x_k y_k \right)^q = (c + x_1y_1 + \cdots + x_dy_d)^q \\
&= \sum_{|\mathbf{n}|=q} \binom{q}{\mathbf{n}} c^{n_0} (x_1y_1)^{n_1} (x_2y_2)^{n_2} \ldots (x_dy_d)^{n_d} \\
&= \sum_{|\mathbf{n}|=q} \binom{q}{\mathbf{n}} c^{n_0} ({x_1}^{n_1} {x_2}^{n_2} \ldots {x_d}^{n_d})({y_1}^{n_1} {y_2}^{n_2} \ldots {y_d}^{n_d}) \\
&= \sum_{|\mathbf{n}|=q} \left(\sqrt{a_{\mathbf{n}}} \prod_{k=1}^{d} {x_k}^{n_k} \right) \left(\sqrt{a_{\mathbf{n}}} \prod_{k=1}^{d} {y_k}^{n_k} \right) \\
&= \phi(\mathbf{x})^T \phi(\mathbf{y})
\end{aligned}$$

где $a_{\mathbf{n}} = \binom{q}{\mathbf{n}} c^{n_0}$, и сумма по всем $\mathbf{n} = (n_0, n_1, \ldots, n_d)$, такая что $|\mathbf{n}| = n_0 + n_1 + \cdots + n_d = q$. Используя обозначение $\mathbf{x}^{\mathbf{n}} = \prod_{k=1}^{d} {x_k}^{n_k}$, отображение $\phi\colon \mathbb{R}^d \longrightarrow \mathbb{R}^m$ задается как вектор

$$\phi(\mathbf{x}) = (\ldots, a_{\mathbf{n}}\mathbf{x}^{\mathbf{n}}, \ldots)^T = \left(\ldots, \sqrt{\binom{q}{\mathbf{n}} c^{n_0}} \prod_{k=1}^{d} {x_k}^{n_k}, \ldots \right)^T$$

где переменная $\mathbf{n} = (n_0, \ldots, n_d)$ охватывает все возможные значения, такие, что $|\boldsymbol{n}| = q$. Можно показать, что размерность пространства признаков задается как

$$m = \binom{d+q}{q}$$

2.2.2 Гауссово ядро

Ядро Гаусса, также называемое ядром радиальной базисной функции Гаусса (RBF), определяется как

$$K(\mathbf{x},\mathbf{y}) = exp\left\{\frac{\|\mathbf{x}-\mathbf{y}\|^2}{2\sigma^2}\right\}, \tag{2.10}$$

где $\sigma > 0$ – параметр разброса, который играет ту же роль, что и стандартное отклонение в функции нормальной плотности. Обратите внимание, что $K(\mathbf{x},\mathbf{x}) = 1$, и, кроме того, значение ядра обратно пропорционально расстоянию между двумя точками **x** и **y**.

Интересно отметить, что пространство признаков для гауссова ядра имеет бесконечную размерность. Чтобы увидеть это, обратите внимание, что экспоненциальная функция может быть записана как бесконечное разложение

$$\exp\{a\} = \sum_{n=0}^{\infty}\frac{a^n}{n!} = 1 + a + \frac{1}{2!}a^2 + \frac{1}{3!}a^3 + \cdots$$

Далее, используя $\gamma = \frac{1}{2\sigma^2}$ и отметив, что $\|\mathbf{x}-\mathbf{y}\|^2 = \|\mathbf{x}\|^2 + \|\mathbf{y}\|^2 - 2\mathbf{x}^T\mathbf{y}$, мы можем переписать гауссово ядро следующим образом

$$K(\mathbf{x},\mathbf{y}) = exp\{-\gamma\|\mathbf{x}-\mathbf{y}\|^2\} = exp\{-\gamma\|\mathbf{x}\|^2\}\cdot exp\{-\gamma\|\mathbf{y}\|^2\}\cdot exp\{2\gamma\mathbf{x}^T\mathbf{y}\}$$

В частности, последний член представляется как бесконечное разложение

$$exp\{2\gamma\mathbf{x}^T\mathbf{y}\} = \sum_{q=0}^{\infty}\frac{(2\gamma)^q}{q!}(\mathbf{x}^T\mathbf{y})^q = 1 + (2\gamma)\mathbf{x}^T\mathbf{y} + \frac{(2\gamma)^{\mathbf{2}}}{2!}(\mathbf{x}^T\mathbf{y})^{\mathbf{2}} + \cdots$$

Используя полиномиальное разложение $(\mathbf{x}^T\mathbf{y})^{\boldsymbol{q}}$, мы можем записать гауссово ядро в виде

$$\begin{aligned} K(\mathbf{x},\mathbf{y}) &= exp\{-\gamma\|\mathbf{x}\|^2\}\,exp\{-\gamma\|\mathbf{y}\|^2\}\sum_{q=0}^{\infty}\frac{(2\gamma)^q}{q!}\left(\sum_{|\mathbf{n}|=q}\binom{q}{\mathbf{n}}\prod_{k=1}^{d}(x_k y_k)^{n_k}\right) \\ &= \sum_{q=0}^{\infty}\sum_{|\mathbf{n}|=q}\left(\sqrt{a_{\mathbf{q},\mathbf{n}}}exp\{-\gamma\|\mathbf{x}\|^2\}\prod_{k=1}^{d}{x_k}^{n_k}\right)\left(\sqrt{a_{\mathbf{q},\mathbf{n}}}exp\{-\gamma\|\mathbf{y}\|^2\}\prod_{k=1}^{d}{y_k}^{n_k}\right) \\ &= \boldsymbol{\phi}(\mathbf{x})^T\boldsymbol{\phi}(\boldsymbol{y}) \end{aligned}$$

где $a_{\mathbf{q},\mathbf{n}} = \frac{(2\gamma)^q}{q!}\binom{q}{\mathbf{n}}$, и $\mathbf{n} = (n_0, n_1, \ldots, n_d)$, такая что $|\mathbf{n}| = n_0 + n_1 + \cdots + n_d = q$. Отображение в пространство признаков соответствует функции $\boldsymbol{\phi}$: $\mathbb{R}^d \longrightarrow \mathbb{R}^\infty$

$$\boldsymbol{\phi}(\mathbf{x}) = \left(\ldots, \sqrt{\frac{(2\gamma)^q}{q!}\binom{q}{\mathbf{n}}}\,exp\{-\gamma\|\mathbf{x}\|^2\}\prod_{k=1}^{d}{x_k}^{n_k}, \ldots\right)^T$$

с размерностями в пределах всех степеней $q = 0, \dots, \infty$, и с переменной $\mathbf{n} = (n_0, n_1, \dots, n_d)$ в диапазоне всех возможных значений, таких, что $|\mathbf{n}| = q$ для каждого значения q. Поскольку φ отображает входное пространство в бесконечномерное пространство признаков, мы, очевидно, не можем явно преобразовать $\mathbf{x}$ в $\varphi(\mathbf{x})$, но вычислить гауссово ядро $K(\mathbf{x}, \mathbf{y})$ просто.

2.3 Основные операции ядра в пространстве признаков

Рассмотрим основные операции анализа данных, которые могут быть выполнены только с помощью ядер, без определения $\phi(\boldsymbol{x})$

2.3.1 Норма точки

Норма точки в пространстве признаков вычисляется следующим образом:

$$\|\phi(\boldsymbol{x})\|^2 = \phi(\boldsymbol{x})^T \phi(\boldsymbol{x}) = K(\boldsymbol{x}, \boldsymbol{x})$$

что предполагает $\|\phi(\boldsymbol{x})\| = \sqrt{K(x, x)}$.

2.3.2 Расстояние между точками

Расстояние между двумя точками $\phi(\boldsymbol{x}_i)$ и $\phi(\boldsymbol{x}_j)$ может быть вычислено как

$$\begin{aligned} \left\|\phi(\boldsymbol{x}_i) - \phi(\boldsymbol{x}_j)\right\|^2 &= \|\phi(\boldsymbol{x}_i)\|^2 + \left\|\phi(\boldsymbol{x}_j)\right\|^2 - 2\phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j) \\ &= K(\boldsymbol{x}_i, \boldsymbol{x}_i) + K(\boldsymbol{x}_j, \boldsymbol{x}_j) - 2K(\boldsymbol{x}_i, \boldsymbol{x}_j) \end{aligned} \tag{2.11}$$

что предполагает

$$\delta\left(\phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j)\right) = \left\|\phi(\boldsymbol{x}_i) - \phi(\boldsymbol{x}_j)\right\| = \sqrt{K(\boldsymbol{x}_i, \boldsymbol{x}_i) + K(\boldsymbol{x}_j, \boldsymbol{x}_j) - 2K(\boldsymbol{x}_i, \boldsymbol{x}_j)}$$

Из преобразования уравнения (2.11) видно, что значение ядра можно рассматривать как меру схожести между двумя точками:

$$\frac{1}{2}\left(\|\phi(\boldsymbol{x}_i)\|^2 + \left\|\phi(\boldsymbol{x}_j)\right\|^2 \left\|\phi(\boldsymbol{x}_i) - \phi(\boldsymbol{x}_j)\right\|^2\right) = K(\boldsymbol{x}_i, \boldsymbol{x}_j) = 2\phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j)$$

Таким образом, чем больше расстояние $\left\|\phi(\boldsymbol{x}_i) - \phi(\boldsymbol{x}_j)\right\|$ между двумя точками в пространстве признаком, тем меньше значение ядра и меньше схожесть.

2.3.3 Среднее в пространстве признаков

Среднее точек в пространстве признаков определено как

$$\boldsymbol{\mu}_\phi = \frac{1}{n}\sum_{i=1}^{n} \phi(\boldsymbol{x}_i)$$

Поскольку в общем случае к $\phi(\boldsymbol{x}_i)$ доступа нет, нельзя явно вычислить среднюю точку в пространстве признаков.

Тем не менее, можно вычислить квадрат нормы среднего следующим образом:

$$\begin{aligned}\left\|\boldsymbol{\mu}_{\phi}\right\|^2 &= \boldsymbol{\mu}_{\phi}^T\boldsymbol{\mu}_{\phi} \\ &= \left(\frac{1}{n}\sum_{i=1}^{n}\phi(\boldsymbol{x}_i)\right)^T\left(\frac{1}{n}\sum_{j=1}^{n}\phi(\boldsymbol{x}_j)\right) \\ &= \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{n}\phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x}_j) \\ &= \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{n}K(\boldsymbol{x}_i,\boldsymbol{x}_j)\end{aligned} \tag{2.12}$$

Таким образом, квадрат нормы в пространстве признаков – среднее по матрице ядра $\boldsymbol{K}$.

2.3.4 Общая дисперсия в пространстве признаков

Сначала выведем формулу квадрата расстояния от точки $\phi(\boldsymbol{x}_i)$ до среднего $\boldsymbol{\mu}_{\phi}$ в пространстве признаков.

$$\begin{aligned}\left\|\phi(\boldsymbol{x}_i)-\boldsymbol{\mu}_{\phi}\right\|^2 &= \|\phi(\boldsymbol{x}_i)\|^2 - 2\phi(\boldsymbol{x}_i)^2\boldsymbol{\mu}_{\phi} + \left\|\boldsymbol{\mu}_{\phi}\right\|^2 \\ &= K(\boldsymbol{x}_i,\boldsymbol{x}_i) - \frac{2}{n}\sum_{j=1}^{n}K(\boldsymbol{x}_i,\boldsymbol{x}_j) + \frac{1}{n^2}\sum_{a=1}^{n}\sum_{b=1}^{n}K(\boldsymbol{x}_a,\boldsymbol{x}_b)\end{aligned}$$

Общая дисперсия (ур-е 1.4) в пространстве признаков получается из среднего квадрата отклонения точек от среднего:

$$\begin{aligned}\sigma_{\phi}^2 &= \frac{1}{n}\sum_{i=1}^{n}\left\|\phi(\boldsymbol{x}_i)-\boldsymbol{\mu}_{\phi}\right\|^2 \\ &= \frac{1}{n}\sum_{i=1}^{n}K(\boldsymbol{x}_i,\boldsymbol{x}_i) - \frac{2}{n}\sum_{j=1}^{n}K(\boldsymbol{x}_i,\boldsymbol{x}_j) + \frac{1}{n^2}\sum_{a=1}^{n}\sum_{b=1}^{n}K(\boldsymbol{x}_a,\boldsymbol{x}_b) \\ &= \frac{1}{n}\sum_{i=1}^{n}K(\boldsymbol{x}_i,\boldsymbol{x}_i) - \frac{2}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}K(\boldsymbol{x}_i,\boldsymbol{x}_j) + \frac{n}{n^3}\sum_{a=1}^{n}\sum_{b=1}^{n}K(\boldsymbol{x}_a,\boldsymbol{x}_b) \\ &= \frac{1}{n}\sum_{i=1}^{n}K(\boldsymbol{x}_i,\boldsymbol{x}_i) - \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}K(\boldsymbol{x}_i,\boldsymbol{x}_j)\end{aligned} \tag{2.13}$$

Другими словами, общая дисперсия в пространстве признаков – разница между средним по главной диагонали и среднем по матрице ядра $\boldsymbol{K}$**.** Заметьте, что из второй член получившегося уравнения - $\left\|\boldsymbol{\mu}_{\phi}\right\|^2$ (по 2.12).

2.3.5 Центрирование в пространстве признаков

Каждую точку в пространстве признаков можно центрировать с помощью вычитания среднего:

$$\hat{\phi}(\boldsymbol{x}_i) = \phi(\boldsymbol{x}_i) - \boldsymbol{\mu}_{\phi}$$

Поскольку в общем случае нет явного представления ни $\phi(\boldsymbol{x}_i)$, ни $\boldsymbol{\mu}_\phi$, нельзя явно центрировать точки. Но можно посчитать *центрированную матрицу ядра*, т.е. матрицу ядра на центрированными точками.

Центрированная матрица ядра задана как

$$\widehat{\boldsymbol{K}} = \left\{\widehat{K}(\boldsymbol{x}_i, \boldsymbol{x}_j)\right\}_{i,j=1}^{n}$$

где каждая ячейка соответствует ядру между центрированными точками, т.е.:

$$\begin{aligned}
\widehat{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) &= \hat{\phi}(\boldsymbol{x}_i)^T \hat{\phi}(\boldsymbol{x}_j) \\
&= \left(\phi(\boldsymbol{x}_i) - \boldsymbol{\mu}_\phi\right)^T \left(\phi(\boldsymbol{x}_j) - \boldsymbol{\mu}_\phi\right) \\
&= \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j) - \phi(\boldsymbol{x}_i)^T \boldsymbol{\mu}_\phi - \phi(\boldsymbol{x}_j)^T \boldsymbol{\mu}_\phi + \boldsymbol{\mu}_\phi^T \boldsymbol{\mu}_\phi \\
&= K(\boldsymbol{x}_i, x_j) - \frac{1}{n}\sum_{k=1}^{n} \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_k) - \frac{1}{n}\sum_{k=1}^{n} \phi(\boldsymbol{x}_j)^T \phi(\boldsymbol{x}_k) + \left\|\boldsymbol{\mu}_\phi\right\|^2 \\
&= K(\boldsymbol{x}_i, \boldsymbol{x}_j) - \frac{1}{n}\sum_{k=1}^{n} K(\boldsymbol{x}_i, \boldsymbol{x}_k) - \frac{1}{n}\sum_{k=1}^{n} K(\boldsymbol{x}_j, \boldsymbol{x}_k) + \frac{1}{n^2}\sum_{a=1}^{n}\sum_{b=1}^{n} K(\boldsymbol{x}_a, \boldsymbol{x}_b)
\end{aligned}$$

Другими словами, можно вычислить центрированную матрицу ядра с использованием одной только функции ядра. Над всеми парами точек, центрированная матрица ядра может быть записана следующим образом:

$$\begin{aligned}
\widehat{\boldsymbol{K}} &= \boldsymbol{K} - \frac{1}{n}\mathbf{1}_{n\times n}\boldsymbol{K} - \frac{1}{n}\boldsymbol{K}\mathbf{1}_{n\times n} + \frac{1}{n^2}\mathbf{1}_{n\times n}\boldsymbol{K}\mathbf{1}_{n\times n} \\
&= \left(\boldsymbol{I} - \frac{1}{n}\mathbf{1}_{n\times n}\right)\boldsymbol{K}\left(\boldsymbol{I} - \frac{1}{n}\mathbf{1}_{n\times n}\right)
\end{aligned}$$

Где $\mathbf{1}_{n\times n}$ – единичная матрица, у которой все элементы равны 1.

2.3.6 Нормализация в пространстве признаков

Частый вид нормализации – убедится, что точки в пространстве признаков имеют единичную длину с помощью замены $\phi(\boldsymbol{x}_i)$ на соответствующий единичный вектор $\phi_n(\boldsymbol{x}_i) = \frac{\phi(\boldsymbol{x}_i)}{\|\phi(\boldsymbol{x}_i)\|}$. Скалярному произведению в пространстве признаков соответствует косинус угла между двумя отображенными точки, поскольку

$$\phi_n(\boldsymbol{x}_i)^T \phi_n(\boldsymbol{x}_j) = \frac{\phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j)}{\|\phi(\boldsymbol{x}_i)\| \cdot \|\phi(\boldsymbol{x}_j)\|} = \cos\theta$$

Если отображенные точки и центрированы, и нормализованы, то скалярное произведение – корреляция между двумя точками в пространстве признаков.

Нормализованная матрица ядра $\boldsymbol{K}_n$ может быть получена с использованием только функции ядра K следующим образом:

$$\boldsymbol{K}_n(\boldsymbol{x}_i, \boldsymbol{x}_j) = \frac{\phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j)}{\|\phi(\boldsymbol{x}_i)\| \cdot \|\phi(\boldsymbol{x}_j)\|} = \frac{K(\boldsymbol{x}_i, \boldsymbol{x}_j)}{\sqrt{K(\boldsymbol{x}_i, \boldsymbol{x}_i) \cdot K(\boldsymbol{x}_j, \boldsymbol{x}_j)}}$$

Причем у $\boldsymbol{K}_n$ диагональные элементы равны 1.

Пусть $\boldsymbol{W}$ – диагональная матрица, состоящая из диагональных элементов $\boldsymbol{K}$

$$W = diag(\boldsymbol{K}) = \begin{pmatrix} K(\boldsymbol{x}_1, \boldsymbol{x}_1) & 0 & \cdots & 0 \\ 0 & K(\boldsymbol{x}_2, \boldsymbol{x}_2) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & K(\boldsymbol{x}_n, \boldsymbol{x}_n) \end{pmatrix}$$

Нормализованная матрица ядра в таком случае может быть компактно записана следующим образом:

$$\mathbf{K}_n = \boldsymbol{W}^{-1/2} \cdot \boldsymbol{K} \cdot \boldsymbol{W}^{-1/2}$$

Где $\boldsymbol{W}^{-1/2}$ – диагональная матрица, определенная как $\boldsymbol{W}^{-1/2}(\boldsymbol{x}_i, \boldsymbol{x}_i) = \frac{1}{\sqrt{K(\boldsymbol{x}_i, \boldsymbol{x}_i)}}$, со всеми остальными элементами равными 0.

2.4. Ядра для сложных объектов

Мы завершаем эту главу некоторыми примерами ядер, определенных для сложных данных, таких как строки и графы.

2.4.1. Ядро спектра для строк

Рассмотрим текст или последовательность данных, определенных над алфавитом Σ. Отображение признаков l-спектра это отображение $\varphi: \Sigma^* \rightarrow \mathbb{R}^{|\Sigma|^l}$ из множества подстрок над Σ в $|\Sigma|^l$-ное пространство, представляющее собой количество всех возможных подстрок длины l, определяемых как

$$\varphi(\mathrm{x}) = (\ldots, \#(\alpha), \ldots)^T_{\alpha \in \Sigma^l}$$

где $\#(\alpha)$ – количество вхождений строки α длиной l в x**.**

(Полное) отображение спектра является расширением отображения l-спектра, полученного путем рассмотрения всех длин l от 0 до ∞, что приводит к бесконечномерному отображению признаков $\varphi: \Sigma \rightarrow \mathbb{R}^{\infty}$:

$$\varphi(\mathrm{x}) = (\ldots, \#(\alpha), \ldots)^T_{\alpha \in \Sigma^*}$$

где $\#(\alpha)$ – количество вхождений строки α в x**.**

Ядро l-спектра между двумя строками $\mathrm{x}_i, \mathrm{x}_j$, — это просто скалярное произведение между их l-отображениями спектра:

$$K(\mathrm{x}_i, \mathrm{x}_j) = \varphi(\mathrm{x}_i)^T \varphi(\mathrm{x}_j)$$

Наивное вычисление ядра l-спектра занимает $O(|\Sigma|^l)$ времени. Однако для данной строки x длины n подавляющее большинство строк длины l имеют нулевое число вхождений,

которые можно игнорировать. Отображение l-спектра может быть эффективно вычислено за время $O(n)$ для строки длины n (при условии, что $n \gg l$), потому что может быть не более $n - l + 1$ подстрок длины l, поэтому, ядро l -спектра может вычисляться за $O(n + m)$ времени для любых двух строк длины n и m соответственно.

Отображение признаков для (полного) ядра спектра бесконечномерно, но, опять же, для данной строки x длины n, подавляющее большинство строк будет иметь нулевое число вхождений. Простая реализация отображения спектра для строки x длины n может быть вычислена за время $O(n^2)$, потому что x может иметь не более $\sum_{l=1}^{n} n - l + 1 = n(n+1)/2$ различных непустых подстрок. Ядро спектра может быть вычислено за время $O(n^2 + m^2)$ для любых двух строк длины n и m соответственно. Однако гораздо более эффективные вычисления доступны с помощью суффиксных деревьев (см. Главу 10) с общим временем $O(n + m)$.

2.4.2. Ядра с диффузией на вершинах графа

Пусть $\boldsymbol{S}$ - некоторая симметричная матрица подобия между вершинами графа $G = (V, E)$. Например, $\boldsymbol{S}$ может быть (взвешенной) матрицей смежности $\boldsymbol{A}$ [ур-е. (4.1)] или матрицей Кирхгофа (Лапласа) $\boldsymbol{L} = \boldsymbol{A} - \boldsymbol{\Delta}$ (или ее отрицание), где $\boldsymbol{\Delta}$ - матрица степеней для неориентированного графа G, определенная как $\boldsymbol{\Delta}(i, j) = d_i$ и $\boldsymbol{\Delta}(i, j) = 0$ для всех $i \neq j$ и d_i - степень узла i.

Рассмотрим сходство между любыми двумя вершинами, получаемого путем суммирования произведения сходств по пути длиной 2:

$$S^{(2)}(\mathrm{x}_i, \mathrm{x}_j) = \sum_{a=1}^{n} S(\mathrm{x}_i, \mathrm{x}_a)\, S(\mathrm{x}_a, \mathrm{x}_j) = \boldsymbol{S_i^T S_j}$$

где

$$\boldsymbol{S_i} = (S(\mathrm{x}_i, \mathrm{x}_1), S(\mathrm{x}_i, \mathrm{x}_2), \ldots, S(\mathrm{x}_i, \mathrm{x}_n))^T$$

обозначает вектор (столбец), представляющий i строку $\boldsymbol{S}$ (и, поскольку $\boldsymbol{S}$ симметричен, он также обозначает i столбец $\boldsymbol{S}$). Таким образом, по всем парам узлов матрица подобия по пути длиной 2, обозначенная как $\boldsymbol{S^{(2)}}$, задается как квадрат базовой матрицы подобия $\boldsymbol{S}$:

$$\boldsymbol{S^{(2)}} = \boldsymbol{S} \times \boldsymbol{S} = \boldsymbol{S^2}$$

В общем случае, если мы просуммируем произведение базовых сходств по всем путям длины l между двумя вершинами, мы получим матрицу подобия $\boldsymbol{S}^l$ длины l, которая является просто l-степенью **S**, то есть,

$$\boldsymbol{S^{(l)}} = \boldsymbol{S}^l$$

2.4.3 Степень ядер

Пути четной длины образуют положительно определенные ядра, но пути нечетной длины не гарантируют этого, если только базовая матрица $\boldsymbol{S}$ сама не является положительно определенной. В частности, $\boldsymbol{K} = \boldsymbol{S^2}$ допустимое ядро. Чтобы убедиться в этом, предположим, что i-строка $\boldsymbol{S}$ представляет собой отображение признаков для x_i, то есть $\varphi(\mathrm{x}_i) = S_i$. Значение

ядра между любыми двумя точками тогда является скалярным произведением в пространстве признаков.

$$K(\mathrm{x}_i, \mathrm{x}_j) = S^{(2)}(\mathrm{x}_i, \mathrm{x}_j) = S_i^T S_j = \varphi(\mathrm{x}_i)^T \varphi(\mathrm{x}_j)$$

Для произвольного пути длины l пусть $\boldsymbol{K} = \boldsymbol{S^l}$. Рассмотрим спектральное разложение матрицы $\boldsymbol{S}$:

$$\boldsymbol{S} = \boldsymbol{U\Lambda U}^T = \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \boldsymbol{u_i}\, \lambda_i \boldsymbol{u_i^T}$$

где $\boldsymbol{U}$ - ортогональная матрица собственных векторов, а $\boldsymbol{\Lambda}$ - диагональная матрица собственных чисел $\boldsymbol{S}$:

$$\boldsymbol{U} = \begin{pmatrix} | & | & & | \\ \boldsymbol{u_1} & \boldsymbol{u_2} & \dots & \boldsymbol{u_n} \\ | & | & & | \end{pmatrix} \quad \boldsymbol{\Lambda} = \begin{pmatrix} \lambda_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_n \end{pmatrix}_{\mathrm{j}}$$

Спектральное разложение матрицы $\boldsymbol{K}$ можно получить следующим образом

$$\boldsymbol{K} = \boldsymbol{S}^l = (\boldsymbol{U\Lambda U}^T)^l = \boldsymbol{U}(\boldsymbol{\Lambda}^l)\, \boldsymbol{U}^T$$

где мы использовали тот факт, что собственные векторы $\boldsymbol{S}$ и $\boldsymbol{S}^l$ идентичны, и, кроме того, собственные числа $\boldsymbol{S}^l$ заданы как $(\lambda_i)^l$ (для всех $\mathrm{i} = 1, \dots, \mathrm{n}$), где λ_i - собственное число $\boldsymbol{S}$. Чтобы $\boldsymbol{K} = \boldsymbol{S}^l$ была положительной определенной матрицей, все ее собственные числа должны быть неотрицательными, что гарантируется для всех путей четной длин. Поскольку $(\lambda_i)^l$ будет отрицательным, если l нечетно при отрицательном λ_i, пути нечетной длины приводят к положительно определенному ядру, только если $\boldsymbol{S}$ положительно определено.

2.4.4 Ядро с экспоненциальной диффузией

Вместо того, чтобы фиксировать длину пути априори, мы можем получить новое ядро между вершинами графа, рассматривая обходы всех возможных длин, при этом уменьшая вклад более длинных путей, которые приводят к ядру с экспоненциальной диффузией, определяемому как

$$\boldsymbol{K} = \sum_{l=0}^{\infty} \frac{1}{l!} \beta^l \boldsymbol{S}^l = \boldsymbol{I} + \beta \boldsymbol{S} + \frac{1}{2!}\beta^2 \boldsymbol{S}^2 + \frac{1}{3!}\beta^3 \boldsymbol{S}^3 + \cdots = exp\{\beta \boldsymbol{S}\} \quad (2.15)$$

где β - коэффициент затухания, а $exp\{\beta \boldsymbol{S}\}$ – экспонента матрицы. Ряд в правой части выше сходится при всех $\beta \geq 0$.

Подставив $\boldsymbol{S} = \boldsymbol{U\Lambda U}^T = \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \lambda_i \boldsymbol{u_i}\, \boldsymbol{u_i^T}$ в ур-е (5.15) и используя тот факт, что $\boldsymbol{UU}^T = \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \boldsymbol{u_i}\, \boldsymbol{u_i^T} = \boldsymbol{I}$, имеем

$$\boldsymbol{K} = \boldsymbol{I} + \beta \boldsymbol{S} + \frac{1}{2!}\beta^2 \boldsymbol{S}^2 + \cdots = \left(\sum_{i=1}^{n} \boldsymbol{u_i}\, \boldsymbol{u_i^T}\right) + \left(\sum_{i=1}^{n} \boldsymbol{u_i}\, \beta \boldsymbol{u_i^T}\right) + \left(\sum_{i=1}^{n} \boldsymbol{u_i}\, \frac{\beta^2}{2!} \boldsymbol{u_i^T}\right) + \cdots =$$

$$\sum_{i=1}^{n} \boldsymbol{u_i} \left(1 + \beta\lambda_i + \frac{1}{2!}\beta^2 {\lambda_i}^2 + \cdots\right) \boldsymbol{u_i^T} = \sum_{i=1}^{n} \boldsymbol{u_i}\, \exp\{\beta\lambda_i\}\, \boldsymbol{u_i^T} =$$

$$\boldsymbol{U} \begin{pmatrix} \exp\{\beta\lambda_1\} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \exp\{\beta\lambda_n\} \end{pmatrix} \boldsymbol{U}^T \quad (2.16)$$

Таким образом, собственные векторы $\boldsymbol{K}$ такие же, как и у $\boldsymbol{S}$, тогда как ее собственные числа задаются как $\exp\{\beta\lambda_i\}$, где λ_i - собственное число $\boldsymbol{S}$. Кроме того, $\boldsymbol{K}$ симметричная матрица, потому что $\boldsymbol{S}$ симметрична, а его собственные числа действительны и неотрицательны, потому что результат взятия экспоненты от действительного числа неотрицателен. Таким образом, $\boldsymbol{K}$ является положительной определенной матрицей ядра. Сложность вычисления ядра с диффузией составляет $O(n^3)$, что соответствует сложности спектрального разложение матрицы.

2.4.5 Ядро с диффузией фон Неймана

Связанное ядро, на основе степеней $\boldsymbol{S}$, является ядром с диффузией фон Неймана, определяемое как

$$\boldsymbol{K} = \sum_{l=0}^{\infty} \beta^l \boldsymbol{S}^l \quad (2.17)$$

где $\beta \geq 0$. Расширим уравнение (2.17):

$$\boldsymbol{K} = \boldsymbol{I} + \beta\boldsymbol{S} + \beta^2\boldsymbol{S}^2 + \beta^3\boldsymbol{S}^3 + \cdots = \boldsymbol{I} + \beta\boldsymbol{S}(\boldsymbol{I} + \beta\boldsymbol{S} + \beta^2\boldsymbol{S}^2 + \cdots) = \boldsymbol{I} + \beta\boldsymbol{S}\boldsymbol{K}$$

Преобразуя члены в предыдущем уравнении, мы получаем выражение в замкнутой форме для ядра фон Неймана

$$\boldsymbol{K} - \beta\boldsymbol{S}\boldsymbol{K} = \boldsymbol{I}$$

$$(\boldsymbol{I} - \beta\boldsymbol{S})\boldsymbol{K} = \boldsymbol{I}$$

$$\boldsymbol{K} = (\boldsymbol{I} - \beta\boldsymbol{S})^{-1} \quad (2.18)$$

Подставляя спектральное разложение $\boldsymbol{S} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T$ и переписывая $\boldsymbol{I} = \boldsymbol{U}\boldsymbol{U}^T$, мы имеем

$$\boldsymbol{K} = (\boldsymbol{U}\boldsymbol{U}^T - \boldsymbol{U}(\beta\boldsymbol{\Lambda})\boldsymbol{U}^T)^{-1} = (\boldsymbol{U}(\boldsymbol{I} - \beta\boldsymbol{\Lambda})\boldsymbol{U}^T)^{-1} = \boldsymbol{U}(\boldsymbol{I} - \beta\boldsymbol{\Lambda})^{-1}\boldsymbol{U}^T$$

где $(\boldsymbol{I} - \beta\boldsymbol{\Lambda})^{-1}$ - диагональная матрица, i-диагональный элемент которой равен $(1 - \beta\lambda_i)^{-1}$. Собственные вектора $\boldsymbol{K}$ и $\boldsymbol{S}$ идентичны, но собственные числа $\boldsymbol{K}$ заданы как $1 / (1 - \beta\lambda_i)$. Чтобы $\boldsymbol{K}$ было положительным определенным ядром, все его собственные числа должны быть неотрицательными, что, в свою очередь, означает, что

$$(1 - \beta\lambda_i)^{-1} \geq 0$$

$$1 - \beta\lambda_i \geq 0$$

$$\beta \leq 1 / \lambda_i$$

Далее, обратная матрица$(\boldsymbol{I} - \beta\boldsymbol{\Lambda})^{-1}$ существует, только если

$$\det(\boldsymbol{I} - \beta\boldsymbol{\Lambda}) = \prod_{i=1}^{n} 1 - \beta\lambda_i \neq 0$$

откуда следует, что $\beta \neq 1 / \lambda_i$ для всех i. Таким образом, чтобы $\boldsymbol{K}$ было допустимым ядром, мы требуем, чтобы $\leq 1 / \lambda_i$ для всех $\mathrm{i} = 1, \ldots, \mathrm{n}$. Таким образом, ядро фон Неймана будет положительно определенным, если $|\beta| \leq 1 / \mathrm{p}(\boldsymbol{S})$, где $\mathrm{p}(\boldsymbol{S}) = \max_i\{|\lambda_i|\}$, называемым спектральным радиусом $\boldsymbol{S}$ и определяемым как наибольшее собственное число $\boldsymbol{S}$ по абсолютной величине.

Глава 3. Снижение размерности

В главе 3 мы рассмотрели некоторые характеристики многомерных данных, которые подчас противоречат здравому смыслу. Например, в пространствах высокой размерности точки чаще рассредоточены по поверхности пространства или его углам, а центр практически пуст. Можно наблюдать явное увеличение ортогональных осей. Как следствие, данные высокой размерности создают определённые проблемы для анализа, хотя в некоторых применениях (например, в задачах классификации) высокие размерности могут быть полезны. Тем не менее, важно проверить, возможно ли понижение размерности без потери основных свойств исходной матрицы. Это может помочь как в интеллектуальном анализе данных, так и при их визуализации. В этой главе мы изучим методы, которые позволяют получить оптимальные проекции данных на меньшие размерности.

3.1 Предпосылки

Пусть данные представляют из себя матрицу $\mathbf{D}$ размера $n \times d$ (n точек с d признаками):

$$\mathbf{D} = \left(\begin{array}{c|cccc} X & X_1 & X_2 & \cdots & X_d \\ \mathbf{x}_1 & x_{11} & x_{12} & \cdots & x_{1d} \\ \mathbf{x}_2 & x_{21} & x_{22} & \cdots & x_{2d} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{x}_n & x_{n1} & x_{n2} & \cdots & x_{nd} \end{array}\right).$$

Каждая точка $\mathbf{x}_i = (x_{i1}, x_{i2}, \dots, x_{id})^T$ — это вектор в d-мерном пространстве с базисом $\mathbf{e}_1, \mathbf{e}_2, \dots, \mathbf{e}_d$, в котором компонент $\mathbf{e}_i$ соотносится с i-тым признаком X_i. Напомним, что стандартным базисом называется ортонормированный базис для пространства данных, то есть, базисные векторы попарно ортогональны: $\mathbf{e}_i^T \mathbf{e}_j = 0$, и имеют единичную длину: $\|\mathbf{e}_i\| = 1$.

Таким образом, для любого набора d ортонормированных векторов $\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_d$ такого, что $\mathbf{u}_i^T \mathbf{u}_j = 0$ и $\|\mathbf{u}_i\| = 1$ (или $\mathbf{u}_i^T \mathbf{u}_i = 1$), мы можем представить каждую точку $\mathbf{x}$ как линейную комбинацию:

$$\mathbf{x} = a_1 \mathbf{u}_1 + a_2 \mathbf{u}_2 + \cdots + a_d \mathbf{u}_d, \tag{3.1}$$

где вектор $\mathbf{a} = (a_1, a_2, \dots, a_d)^T$ — это координаты $\mathbf{x}$ в новом базисе. Линейную комбинацию выше можно записать в виде умножения матриц:

$$\mathbf{x} = \mathbf{U}\mathbf{a}, \tag{3.2}$$

где $\mathbf{U}$ — матрица размера $d \times d$, i-ый столбец которой содержит i-тый базисный вектор:

$$\mathbf{U} = \begin{pmatrix} | & | & | & | \\ \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_d \\ | & | & | & | \end{pmatrix}.$$

Матрица $\mathbf{U}$ является *ортогональной*, все её столбцы, будучи базисными векторами, *ортогональны*, то есть, они попарно ортогональны и имеют единичную длину:

$$\mathbf{u}_i^T \mathbf{u}_j = \begin{cases} 1, \text{если } i = j \\ 0, \text{если } i \neq j \end{cases}.$$

Поскольку $\mathbf{U}$ ортогональна, её обратная матрица равна её транспонированной:

$$\mathbf{U}^{-1} = \mathbf{U}^T,$$

следовательно: $\mathbf{U}^T\mathbf{U} = \mathbf{I}$, где $\mathbf{I}$ — единичная матрица размера $d \times d$.

Умножив ур. (3.2) слева на $\mathbf{U}^T$, получим выражение для вычисления координат $\mathbf{x}$ в новом базисе:

$$\mathbf{U}^T\mathbf{x} = \mathbf{U}^T\mathbf{U}\mathbf{a}$$

$$\mathbf{a} = \mathbf{U}^T\mathbf{x}. \tag{3.3}$$

Поскольку теоретически существует бесконечное число наборов ортонормальных базисных векторов, возникает вполне естественный вопрос: существует ли оптимальный базис для конкретного толкования оптимальности? Кроме того, часто входная размерность d достаточно велика, что может вызвать различные проблемы из-за проклятия размерности. Вполне естественно поставить вопрос о поиске пространства меньшей размерности, которое сохраняло бы основные характеристики данных. То есть, нам интересно найти r-мерное представление $\mathbf{D}$ такое, чтобы $r \ll d$, или, иными словами, рассматривая произвольную точку $\mathbf{x}$ и полагая, что базисные векторы отсортированы в порядке уменьшения значимости, мы могли бы усечь выражение (3.1) до r членов и получить следующее:

$$\mathbf{x}' = a_1\mathbf{u}_1 + a_2\mathbf{u}_2 + \cdots + a_r\mathbf{u}_r = \sum_{i=1}^{r} a_i\mathbf{u}_i\,. \tag{3.4}$$

Здесь $\mathbf{x}'$ — это проекция $\mathbf{x}$ на первые r базисных векторов, что в терминах матричных операций можно записать следующим образом:

$$\mathbf{x}' = \begin{pmatrix} | & | & | & | \\ \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_r \\ | & | & | & | \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_r \end{pmatrix} = \mathbf{U}_r\mathbf{a}_r, \tag{3.5}$$

где $\mathbf{U}_r$ — матрица, состоящая из r первых базисных векторов, а $\mathbf{a}_r$ — вектор, содержащий первые r координат. Далее, ограничив ур-е 3.3 первыми r членами, получим:

$$\mathbf{a}_\mathrm{r} = \mathbf{U}_r^T\mathbf{x}. \tag{3.6}$$

Подставив 3.6 в 3.5, можно получить окончательное выражение для проекции $\mathbf{x}$ на первые r базисных векторов:

$$\mathbf{x}' = \mathbf{U}_r\mathbf{U}_r^T\mathbf{x} = \mathbf{P}_r\mathbf{x}, \tag{3.7}$$

где $\mathbf{P}_r = \mathbf{U}_r\mathbf{U}_r^T$ — *ортогональная проекционная матрица* для подпространства над первыми r базисными векторами. То есть, $\mathbf{P}_r$ симметрична, и $\mathbf{P}_r^2 = \mathbf{P}_r$. Это легко проверить, поскольку: $\mathbf{P}_r^T = (\mathbf{U}_r\mathbf{U}_r^T)^T = \mathbf{U}_r\mathbf{U}_r^T = \mathbf{P}_r$ и $\mathbf{P}_r^2 = (\mathbf{U}_r\mathbf{U}_r^T)(\mathbf{U}_r\mathbf{U}_r^T) = \mathbf{U}_r\mathbf{U}_r^T = \mathbf{P}_r$. Здесь мы использовали тот факт, что $\mathbf{U}_r^T\mathbf{U}_r = \mathbf{I}_{r\times r}$, единичная матрица размера $r \times r$. Проекционную матрицу $\mathbf{P}_r$ можно легко записать как декомпозицию:

$$\mathbf{P}_r = \mathbf{U}_r \mathbf{U}_r^T = \sum_{i=1}^{r} \mathbf{u}_i \mathbf{u}_i^T . \quad (3.8)$$

Из выражений 3.1 и 3.4 понятно, что проекция $\mathbf{x}$ на оставшиеся размерности составляет *вектор ошибки*:

$$\epsilon = \sum_{i=r+1}^{d} a_i \mathbf{u}_i = \mathbf{x} - \mathbf{x}'.$$

Следует заметить, что $\mathbf{x}'$ и ϵ — ортогональные векторы:

$$\mathbf{x}'^T \epsilon = \sum_{i=1}^{r} \sum_{j=r+1}^{d} a_i a_j \, \mathbf{u}_i^T \mathbf{u}_j = 0,$$

что следует из ортонормированности базиса. Более того, мы можем сделать ещё более сильное утверждение: подпространство над первыми r базисными векторами:

$$S_r = span(\mathbf{u}_1, \dots, \mathbf{u}_r)$$

и подпространство над оставшимися базисными векторами:

$$S_{d-r} = span(\mathbf{u}_{r+1}, \dots, \mathbf{u}_d)$$

являются *ортогональными подпространствами*, то есть, любая пара векторов $\mathbf{x} \in S_r$ и $\mathbf{y} \in S_{d-r}$ должна быть ортогональной. Подпространство S_{d-r} называют также *ортогональным дополнением* S_r.

Цель понижения размерности — найти такой r-мерный базис, который даст наилучшее приближение $\mathbf{x}_i'$ для всех точек $\mathbf{x}_i \in \mathbf{D}$, либо уменьшить ошибку $\epsilon = \mathbf{x} - \mathbf{x}'$ для всех точек.

3.2 Метод главных компонент

Метод главных компонент (PCA) – это метод, который ищет r-мерный базис, наилучшим образом описывающий дисперсию данных. Направление с наибольшей проецируемой дисперсией называется первым главным компонентом. Ортогональное направление, которое захватывает вторую по величине проецируемую дисперсию, называется вторым главным компонентом и т.д. Как мы увидим, направление, которое максимизирует дисперсию, также минимизирует среднеквадратичную ошибку.

3.2.1 Наилучшая линейная аппроксимация

Мы начнем с $r = 1$, т.е. с одномерного подпространства или прямой $\boldsymbol{u}$, которая наилучшим образом аппроксимирует $\boldsymbol{D}$, с точки зрения дисперсии проецируемых точек. Это приведет к общей методике PCA для наилучшего $1 \leq r \leq d$ размерного базиса для $\boldsymbol{D}$.

Без потери общности, мы предполагаем, что $\boldsymbol{u}$ имеет величину $\|\boldsymbol{u}\|^2 = \boldsymbol{u}^T\boldsymbol{u} = 1$; в противном случае можно продолжать увеличивать проецируемую дисперсию, просто увеличивая величину $\boldsymbol{u}$. Мы также предполагаем, что данные центрированы так, что среднее значение $\boldsymbol{\mu} = \mathbf{0}$.

Проекция $\boldsymbol{x}_i$ на вектор $\boldsymbol{u}$ берется как

$$\boldsymbol{x_i'} = \left(\frac{\boldsymbol{u}^T\boldsymbol{x}_i}{\boldsymbol{u}^T\boldsymbol{u}}\right)\boldsymbol{u} = (\boldsymbol{u}^T\boldsymbol{x}_i)\boldsymbol{u} = a_i\boldsymbol{u}$$

где скаляр

$$a_i = \boldsymbol{u}^T\boldsymbol{x}_i$$

дает координату $\boldsymbol{x_i'}$ вдоль прямой $\boldsymbol{u}$. Заметим, что поскольку средняя точка $\boldsymbol{\mu} = \boldsymbol{0}$, ее координата вдоль $\boldsymbol{u}$ равна $\mu_{\boldsymbol{u}} = 0$.

Необходимо выбрать направление $\boldsymbol{u}$ так, чтобы дисперсия проецируемых точек была максимальной. Проецируемая дисперсия вдоль $\boldsymbol{u}$ берется как:

$$\begin{aligned}\sigma_{\boldsymbol{u}}^2 &= \frac{1}{n}\sum_{i=1}^{n}(a_i - \mu_{\boldsymbol{u}})^2 \\ &= \frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{u}^T\boldsymbol{x}_i)^2 \\ &= \frac{1}{n}\sum_{i=1}^{n}\boldsymbol{u}^T(\boldsymbol{x}_i{\boldsymbol{x}_i}^T)\boldsymbol{u} \\ &= \boldsymbol{u}^T\left(\frac{1}{n}\sum_{i=1}^{n}\boldsymbol{x}_i{\boldsymbol{x}_i}^T\right)\boldsymbol{u} \\ &= \boldsymbol{u}^T\boldsymbol{\Sigma}\boldsymbol{u}\end{aligned} \tag{3.9}$$

где $\boldsymbol{\Sigma}$ – это ковариационная матрица для центрированных данных $\boldsymbol{D}$.

Чтобы максимизировать проецируемую дисперсию, мы должны решить задачу ограниченной оптимизации, а именно, чтобы максимизировать $\sigma_{\boldsymbol{u}}^2$ с учетом ограничения $\boldsymbol{u}^T\boldsymbol{u} = 1$. Это может быть решено путем введения множителя Лагранжа α для ограничения, чтобы получить задачу безусловной максимизации.

$$\max_{\boldsymbol{u}} J(\boldsymbol{u}) = \boldsymbol{u}^T\boldsymbol{\Sigma}\boldsymbol{u} - \alpha(\boldsymbol{u}^T\boldsymbol{u} - 1) \tag{3.10}$$

Приравнивая производную $J(\boldsymbol{u})$ по $\boldsymbol{u}$ к нулевому вектору, получаем

$$\frac{\partial}{\partial \boldsymbol{u}}J(\boldsymbol{u}) = \boldsymbol{0}$$

$$\frac{\partial}{\partial \boldsymbol{u}}\left(\boldsymbol{u}^T\boldsymbol{\Sigma}\boldsymbol{u} - \alpha(\boldsymbol{u}^T\boldsymbol{u} - 1)\right) = \boldsymbol{0}$$

$$2\boldsymbol{\Sigma}\boldsymbol{u} - \boldsymbol{2}\alpha\boldsymbol{u} = \boldsymbol{0}$$

$$\boldsymbol{\Sigma}\boldsymbol{u} = \alpha\boldsymbol{u} \tag{3.11}$$

Это означает, что α является собственным значением ковариационной матрицы $\mathbf{\Sigma}$, с соответствующим собственным вектором $\boldsymbol{u}$. Далее, взяв скалярное произведение с $\boldsymbol{u}$ по обе стороны от уравнения (3.11) получаем

$$\boldsymbol{u}^T\mathbf{\Sigma}\boldsymbol{u} = \boldsymbol{u}^T\alpha\boldsymbol{u}$$

Из уравнения (3.9) имеем

$$\sigma_{\boldsymbol{u}}^2 = \alpha\boldsymbol{u}^T\boldsymbol{u}$$

$$\text{или } \sigma_{\boldsymbol{u}}^2 = \alpha \tag{3.12}$$

Таким образом, чтобы максимизировать проецируемую дисперсию $\sigma_{\boldsymbol{u}}^2$, необходимо выбрать наибольшее собственное значение матрицы $\mathbf{\Sigma}$. Другими словами, доминирующий собственный вектор $\boldsymbol{u}_1$ задает направление наибольшей дисперсии, также называемое первым главным компонентом, т.е. $\boldsymbol{u} = \boldsymbol{u}_1$. Кроме того, наибольшее собственное значение λ_1 определяет проецируемую дисперсию, т.е. $\sigma_{\boldsymbol{u}}^2 = \alpha = \lambda_1$.

Минимальная квадратичная ошибка

Теперь покажем, что направление, которое максимизирует проецируемую дисперсию, также минимизирует среднеквадратичную ошибку. Как и раньше, предположим, что набор данных $\boldsymbol{D}$, был центрирован путем вычитания среднего значения из каждой точки. Для точки $\boldsymbol{x}_i \in \boldsymbol{D}$ пусть $\boldsymbol{x}_i'$ обозначает ее проекцию вдоль направления $\boldsymbol{u}$, и пусть $\boldsymbol{\epsilon}_i = \boldsymbol{x}_i - \boldsymbol{x}_i'$ обозначает вектор ошибки. Среднеквадратичная ошибка (MSE) условия оптимизации определяется как

$$MSE(\boldsymbol{u}) = \frac{1}{n}\sum_{i=1}^{n}\|\boldsymbol{\epsilon}_i\|^2 \tag{3.13}$$

$$= \frac{1}{n}\sum_{i=1}^{n}\|\boldsymbol{x}_i - \boldsymbol{x}_i'\|^2$$

$$= \frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{x}_i - \boldsymbol{x}_i')^T(\boldsymbol{x}_i - \boldsymbol{x}_i')$$

$$= \frac{1}{n}\sum_{i=1}^{n}(\|\boldsymbol{x}_i\|^2 - 2\boldsymbol{x}_i^T\boldsymbol{x}_i' + (\boldsymbol{x}_i')^T\boldsymbol{x}_i') \tag{3.14}$$

Заметив, что $\boldsymbol{x}_i' = (\boldsymbol{u}^T\boldsymbol{x}_i)\boldsymbol{u}$, имеем

$$= \frac{1}{n}\sum_{i=1}^{n}(\|\boldsymbol{x}_i\|^2 - 2\boldsymbol{x}_i^T(\boldsymbol{u}^T\boldsymbol{x}_i)\boldsymbol{u} + ((\boldsymbol{u}^T\boldsymbol{x}_i)\boldsymbol{u})^T(\boldsymbol{u}^T\boldsymbol{x}_i)\boldsymbol{u})$$

$$= \frac{1}{n}\sum_{i=1}^{n}(\|\boldsymbol{x}_i\|^2 - 2(\boldsymbol{u}^T\boldsymbol{x}_i)(\boldsymbol{x}_i^T\boldsymbol{u}) + (\boldsymbol{u}^T\boldsymbol{x}_i)(\boldsymbol{x}_i^T\boldsymbol{u})\boldsymbol{u}^T\boldsymbol{u})$$

$$= \frac{1}{n}\sum_{i=1}^{n}\left(\|\boldsymbol{x}_i\|^2 - (\boldsymbol{u}^T\boldsymbol{x}_i)(\boldsymbol{x}_i^T\boldsymbol{u})\right)$$

$$= \frac{1}{n}\sum_{i=1}^{n}\|\boldsymbol{x}_i\|^2 - \frac{1}{n}\sum_{i=1}^{n}\boldsymbol{u}^T(\boldsymbol{x}_i\boldsymbol{x}_i^T)\boldsymbol{u}$$

$$= \frac{1}{n}\sum_{i=1}^{n}\|\boldsymbol{x}_i\|^2 - \boldsymbol{u}^T\left(\frac{1}{n}\sum_{i=1}^{n}\boldsymbol{x}_i\boldsymbol{x}_i^T\right)\boldsymbol{u}$$

$$= \sum_{i=1}^{n}\frac{\|\boldsymbol{x}_i\|^2}{\boldsymbol{n}} - \boldsymbol{u}^T\boldsymbol{\Sigma}\boldsymbol{u} \tag{3.15}$$

Заметим, что общая дисперсия центрированных данных (т.е. $\boldsymbol{\mu} = \mathbf{0}$) получается как

$$\mathrm{var}(\boldsymbol{D}) = \frac{1}{n}\sum_{i=1}^{n}\|\boldsymbol{x}_i - \mathbf{0}\|^2 = \frac{1}{n}\sum_{i=1}^{n}\|\boldsymbol{x}_i\|^2$$

Далее, имеем

$$\mathrm{var}(\boldsymbol{D}) = tr(\boldsymbol{\Sigma}) = \sum_{i=1}^{d}\sigma_i^2$$

Таким образом, можно переписать уравнение (3.15) как

$$MSE(\boldsymbol{u}) = \mathrm{var}(\boldsymbol{D}) - \boldsymbol{u}^T\boldsymbol{\Sigma}\boldsymbol{u} = \sum_{i=1}^{d}\sigma_i^2 - \boldsymbol{u}^T\boldsymbol{\Sigma}\boldsymbol{u}$$

Поскольку первый член $\mathrm{var}(\boldsymbol{D})$ является константой для данного набора данных $\boldsymbol{D}$, вектор $\boldsymbol{u}$, который минимизирует $MSE(\boldsymbol{u})$**,** максимизирует второй член, проецируемую дисперсию $\boldsymbol{u}^T\boldsymbol{\Sigma}\boldsymbol{u}$. Поскольку мы знаем, что $\boldsymbol{u}_1$, доминирующий собственный вектор матрицы $\boldsymbol{\Sigma}$, максимизирует проецируемую дисперсию, имеем

$$MSE(\boldsymbol{u}_1) = \mathrm{var}(\boldsymbol{D}) - \boldsymbol{u}_1{}^T\boldsymbol{\Sigma}\boldsymbol{u}_1 = \mathrm{var}(\boldsymbol{D}) - \boldsymbol{u}_1{}^T\lambda_1\boldsymbol{u}_1 = \mathrm{var}(\boldsymbol{D}) - \lambda_1 \tag{3.16}$$

Таким образом, главный компонент $\boldsymbol{u}_1$, являющийся направлением, которое максимизирует проецируемую дисперсию, также является направлением, которое минимизирует среднеквадратичную ошибку MSE.

3.2.2 Наилучшая двумерная аппроксимация

Теперь нас интересует наилучшее двумерное приближение к $\boldsymbol{D}$. Как и раньше, предположим, что $\boldsymbol{D}$ уже центрирован, т.ч. $\boldsymbol{\mu} = \mathbf{0}$. Мы уже вычислили направление с наибольшей дисперсией, а именно $\boldsymbol{u}_1$, которое является собственным вектором с соответствующим наибольшим собственным значением λ_1 матрицы $\boldsymbol{\Sigma}$. Теперь мы хотим найти другое направление $\mathbf{v}$, которое также максимизирует проецируемую дисперсию, но ортогонально $\boldsymbol{u}_1$. Согласно формуле (3.9) проецируемая на $\mathbf{v}$ дисперсия определяется как

$$\sigma_{\mathbf{v}}^2 = \mathbf{v}^T \mathbf{\Sigma} \mathbf{v}$$

Далее мы потребуем, чтобы $\mathbf{v}$ был единичным вектором, ортогональным $\boldsymbol{u}_1$, т.е.

$$\mathbf{v}^T \boldsymbol{u}_1 = 0$$

$$\mathbf{v}^T \mathbf{v} = 1$$

Тогда условие оптимизации определяется как

$$\max_{\mathbf{v}} J(\mathbf{v}) = \mathbf{v}^T \mathbf{\Sigma} \mathbf{v} - \alpha(\mathbf{v}^T \mathbf{v} - 1) - \beta(\mathbf{v}^T \boldsymbol{u}_1 - 0) \tag{3.17}$$

Взяв производную $J(\mathbf{v})$ по $\mathbf{v}$ и приравняв ее к нулевому вектору, получаем

$$2\mathbf{\Sigma} \mathbf{v} - 2\alpha \mathbf{v} - \beta \boldsymbol{u}_1 = \mathbf{0} \tag{3.18}$$

Если умножить левую часть на $\boldsymbol{u}_1^T$, то получим

$$2\boldsymbol{u}_1^T \mathbf{\Sigma} \mathbf{v} - 2\alpha \boldsymbol{u}_1^T \mathbf{v} - \beta \boldsymbol{u}_1^T \boldsymbol{u}_1 = 0$$

$2\mathbf{v}^T \mathbf{\Sigma} \boldsymbol{u}_1 - \beta = 0$, откуда следует, что

$$\beta = 2\mathbf{v}^T \lambda_1 \boldsymbol{u}_1 = 2\lambda_1 \mathbf{v}^T \boldsymbol{u}_1 = 0$$

При выводе выше мы использовали тот факт, что $\boldsymbol{u}_1^T \mathbf{\Sigma} \mathbf{v} = \mathbf{v}^T \mathbf{\Sigma} \boldsymbol{u}_1$, и что $\mathbf{v}$ ортогонален $\boldsymbol{u}_1$. Подставляя $\beta = 0$ в уравнение (3.18), получаем

$$2\mathbf{\Sigma} \mathbf{v} - 2\alpha \mathbf{v} = \mathbf{0}$$

$$\mathbf{\Sigma} \mathbf{v} = \alpha \mathbf{v}$$

Это означает, что $\mathbf{v}$ – это еще один собственный вектор $\mathbf{\Sigma}$. Так же, как в уравнении (3.12), имеем $\sigma_{\mathbf{v}}^2 = \alpha$. Чтобы максимизировать дисперсию по $\mathbf{v}$, мы должны выбрать $\alpha = \lambda_2$, второе по величине собственное значение $\mathbf{\Sigma}$, причем второй главный компонент задается соответствующим собственным вектором, т.е. $\mathbf{v} = \boldsymbol{u}_2$.

Общая проецируемая дисперсия

Пусть $\boldsymbol{U}_2$ – матрица, столбцы которой соответствуют двум главным компонентам

$$\boldsymbol{U}_2 = \begin{pmatrix} | & | \\ \boldsymbol{u}_1 & \boldsymbol{u}_2 \\ | & | \end{pmatrix}$$

Для точки $\boldsymbol{x}_i \in \boldsymbol{D}$ координаты в двумерном подпространстве, натянутом на $\boldsymbol{u}_1$ и $\boldsymbol{u}_2$, могут быть вычислены по формуле (3.6) следующим образом

$$\boldsymbol{a}_i = \boldsymbol{U}_2^T \boldsymbol{x}_i$$

Предположим, что каждая точка $\boldsymbol{x}_i \in \mathbb{R}^d$ в $\boldsymbol{D}$ была спроецирована для получения ее координат $\boldsymbol{a}_i \in \mathbb{R}^d$, что дало новый набор данных $\boldsymbol{A}$. Кроме того, поскольку предполагается, что $\boldsymbol{D}$ центрирован ($\boldsymbol{\mu} = \mathbf{0}$), координаты спроецированного среднего также равны нулю, т.к. $\boldsymbol{U}_2 \boldsymbol{\mu} = \boldsymbol{U}_2^T \mathbf{0} = \mathbf{0}$.

Общая дисперсия для $\boldsymbol{A}$ определяется как

$$var(\boldsymbol{A}) = \frac{1}{n}\sum_{i=1}^{n} \|\boldsymbol{a}_i - \boldsymbol{0}\|^2$$

$$= \frac{1}{n}\sum_{i=1}^{n} (\boldsymbol{U}_2^T \boldsymbol{x}_i)^T \, (\boldsymbol{U}_2^T \boldsymbol{x}_i)$$

$$= \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{x}_i^T (\boldsymbol{U}_2 \boldsymbol{U}_2^T) \boldsymbol{x}_i$$

$$= \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{x}_i^T \boldsymbol{P}_2 \boldsymbol{x}_i \tag{3.19}$$

Где $\boldsymbol{P}_2$ – матрица ортогональной проекции [уравнение (3.8)], которая определяется как

$$\boldsymbol{P}_2 = \boldsymbol{U}_2 \boldsymbol{U}_2^T = \boldsymbol{u}_1 \boldsymbol{u}_1^T + \boldsymbol{u}_2 \boldsymbol{u}_2^T$$

Подставляя это в (3.19), проецируемая общая дисперсия определяется как

$$var(\boldsymbol{A}) = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{x}_i^T \boldsymbol{P}_2 \boldsymbol{x}_i \tag{3.20}$$

$$= \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{x}_i^T (\boldsymbol{u}_1 \boldsymbol{u}_1^T + \boldsymbol{u}_2 \boldsymbol{u}_2^T) \boldsymbol{x}_i$$

$$= \frac{1}{n}\sum_{i=1}^{n} (\boldsymbol{u}_1^T \boldsymbol{x}_i)(\boldsymbol{x}_i^T \boldsymbol{u}_1) + \frac{1}{n}\sum_{i=1}^{n} (\boldsymbol{u}_2^T \boldsymbol{x}_i)(\boldsymbol{x}_i^T \boldsymbol{u}_2)$$

$$= \boldsymbol{u}_1^T \boldsymbol{\Sigma} \boldsymbol{u}_1 + \boldsymbol{u}_2^T \boldsymbol{\Sigma} \boldsymbol{u}_2 \tag{3.21}$$

Поскольку $\boldsymbol{u}_1$ и $\boldsymbol{u}_2$ – собственные вектора $\boldsymbol{\Sigma}$, имеем $\boldsymbol{\Sigma}\boldsymbol{u}_1 = \lambda_1 \boldsymbol{u}_1$ и $\boldsymbol{\Sigma}\boldsymbol{u}_2 = \lambda_2 \boldsymbol{u}_2$, т.е.

$$var(\boldsymbol{A}) = \boldsymbol{u}_1^T \boldsymbol{\Sigma} \boldsymbol{u}_1 + \boldsymbol{u}_2^T \boldsymbol{\Sigma} \boldsymbol{u}_2 = \boldsymbol{u}_1^T \lambda_1 \boldsymbol{u}_1 + \boldsymbol{u}_2^T \lambda_2 \boldsymbol{u}_2 = \lambda_1 + \lambda_2 \tag{3.22}$$

Таким образом, сумма собственных значений представляет собой общую дисперсию проецируемых точек, а первые два главных компонента максимизируют эту дисперсию.

Среднеквадратичная ошибка

Теперь мы продемонстрируем, что первые два главных компонента также минимизируют среднеквадратичную ошибку. Среднеквадратичная ошибка задается как

$$\mathrm{M}SE = \frac{1}{n}\sum_{i=1}^{n} \|\boldsymbol{x}_i - \boldsymbol{x}_i'\|^2$$

$= \frac{1}{n}\sum_{i=1}^{n} (\|\boldsymbol{x}_i\|^2 - 2\boldsymbol{x}_i^T \boldsymbol{x}_i' + (\boldsymbol{x}_i')^T \boldsymbol{x}_i')$, используя (3.14)

$= var(\boldsymbol{D}) + \frac{1}{n}\sum_{i=1}^{n} (-2\boldsymbol{x}_i^T \boldsymbol{P}_2 \boldsymbol{x}_i + (\boldsymbol{P}_2 \boldsymbol{x}_i)^T \boldsymbol{P}_2 \boldsymbol{x}_i)$, используя (3.7), что $\boldsymbol{x}_i' = \boldsymbol{P}_2 \boldsymbol{x}_i$

$$= var(\boldsymbol{D}) - \frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{x}_i^T \boldsymbol{P}_2 \boldsymbol{x}_i)$$

$$= var(\boldsymbol{D}) - var(\boldsymbol{A}), \text{используя (3.20)} \tag{3.23}$$

Таким образом, MSE минимизируется именно тогда, когда общая проецируемая дисперсия $var(\boldsymbol{A})$ максимальна. Из (3.22) имеем

$$MSE = var(\boldsymbol{D}) - \lambda_1 - \lambda_2$$

3.2.3 Наилучшая r-мерная аппроксимация

Теперь нас интересует наилучшая r-мерная аппроксимация к $\boldsymbol{D}$, где $2 < r \leq d$. Предположим, что мы уже вычислили первые $j-1$ главных компонент или собственных векторов, $\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_{j-1}$, соответствующих $j-1$ наибольшим собственным значениям матрицы $\boldsymbol{\Sigma}$, для $1 \leq j \leq r$. Чтобы вычислить j-й новый базисный вектор $\mathbf{v}$, мы должны убедиться, что он нормализован до единичной длины, т.е. $\mathbf{v}^T\mathbf{v} = 1$, и ортогонален всем предыдущим компонентам $\boldsymbol{u}_i$, т.е. $\boldsymbol{u}_i^T\mathbf{v} = 0$, для $1 \leq i < j$. Как и до этого, проецируемая дисперсия по $\mathbf{v}$ задается как

$$\sigma_{\mathbf{v}}^2 = \mathbf{v}^T\boldsymbol{\Sigma}\mathbf{v}$$

В сочетании с ограничениями на $\mathbf{v}$ это приводит к следующей задаче максимизации с множителями Лагранжа:

$$\max_{\mathbf{v}} J(\mathbf{v}) = \mathbf{v}^T\boldsymbol{\Sigma}\mathbf{v} - \alpha(\mathbf{v}^T\mathbf{v} - 1) - \sum_{i=1}^{j-1}\beta_i(\boldsymbol{u}_i{}^T\mathbf{v} - 0)$$

Взяв производную $J(\mathbf{v})$ по $\mathbf{v}$ и приравняв ее к нулевому вектору, получаем

$$2\boldsymbol{\Sigma}\mathbf{v} - 2\alpha\mathbf{v} - \sum_{i=1}^{j-1}\beta_i\boldsymbol{u}_i = \mathbf{0} \tag{3.24}$$

Если умножить левую часть на $\boldsymbol{u}_k^T$ для $1 \leq k < j$, то получим

$$2\boldsymbol{u}_k^T\boldsymbol{\Sigma}\mathbf{v} - 2\alpha\boldsymbol{u}_k^T\mathbf{v} - \beta_k\boldsymbol{u}_k^T\boldsymbol{u}_k - \sum_{\substack{i=1 \\ i\neq k}}^{j-1}\beta_i\boldsymbol{u}_k^T\boldsymbol{u}_i = 0$$

$$2\mathbf{v}^T\boldsymbol{\Sigma}\boldsymbol{u}_k - \beta_k = 0$$

$$\beta_k = 2\mathbf{v}^T\lambda_k\boldsymbol{u}_k = 2\lambda_k\mathbf{v}^T\boldsymbol{u}_k = 0$$

где мы использовали тот факт, что $\boldsymbol{\Sigma}\boldsymbol{u}_k = \lambda_k\boldsymbol{u}_k$, т.к. $\boldsymbol{u}_k$ является собственным вектором, которому соответствует k-е наибольшее собственное значение λ_k матрицы $\boldsymbol{\Sigma}$. Таким образом, мы находим, что $\beta_i = 0$ для всех $i < j$ в уравнении (3.24), откуда следует

$$\boldsymbol{\Sigma}\mathbf{v} = \alpha\mathbf{v}$$

Чтобы максимизировать дисперсию по $\mathbf{v}$, мы устанавливаем $\alpha = \lambda_j$, j-е наибольшее собственной значение матрицы $\boldsymbol{\Sigma}$, где $\mathbf{v} = \boldsymbol{u}_j$ задает j-ю главную компоненту.

Таким образом, чтобы найти наилучше r-мерное приближение к $\boldsymbol{D}$, мы вычисляем собственные значения матрицы $\boldsymbol{\Sigma}$. Поскольку $\boldsymbol{\Sigma}$ является положительно полуопределенной, все ее собственные числа должны быть неотрицательными, и мы можем отсортировать их в порядке убывания следующим образом:

$$\lambda_1 \geq \lambda_2 \geq \cdots \lambda_r \geq \lambda_{r+1} \ldots \geq \lambda_d \geq 0$$

Затем мы выбираем r наибольших собственных значений и соответствующие им собственные векторы, чтобы сформировать лучшее r-мерное приближение.

Общая проецируемая дисперсия

Пусть $\boldsymbol{U}_r$ – r-мерная матрица базисных векторов

$$\boldsymbol{U}_r = \begin{pmatrix} | & | & & | \\ \boldsymbol{u}_1 & \boldsymbol{u}_2 & \ldots & \boldsymbol{u}_r \\ | & | & & | \end{pmatrix}$$

с матрицей проекции, заданной как

$$\boldsymbol{P}_r = \boldsymbol{U}_r \boldsymbol{U}_r^T = \sum_{i=1}^{r} \boldsymbol{u}_i \boldsymbol{u}_i^T$$

Пусть $\boldsymbol{A}$ обозначает набор данных, сформированный координатами проецируемых точек в r-мерном подпространстве, т.е. $\boldsymbol{a}_i = \boldsymbol{U}_r^T \boldsymbol{x}_i$, и пусть $\boldsymbol{x}_i' = \boldsymbol{P}_r \boldsymbol{x}_i$ обозначает спроецированную точку в исходном d-мерном пространстве. Из (3.19), (3.21) и (3.22) проецируемая дисперсия определяется как

$$var(\boldsymbol{A}) = \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{x}_i^T \boldsymbol{P}_r \boldsymbol{x}_i = \sum_{i=1}^{r} \boldsymbol{u}_i^T \boldsymbol{\Sigma} \boldsymbol{u}_i = \sum_{i=1}^{r} \lambda_i$$

Таким образом, общая проецируемая дисперсия – это просто сумма r наибольших собственных значений матрицы $\boldsymbol{\Sigma}$.

Среднеквадратичная ошибка

На основании вывода для уравнения (3.23), среднеквадратичная ошибка в r-измерениях может быть записана как

$$\begin{aligned} \mathrm{M}SE &= \frac{1}{n} \sum_{i=1}^{n} \|\boldsymbol{x}_i - \boldsymbol{x}_i'\|^2 \\ &= var(\boldsymbol{D}) - var(\boldsymbol{A}) \\ &= var(\boldsymbol{D}) - \sum_{i=1}^{r} \boldsymbol{u}_i^T \boldsymbol{\Sigma} \boldsymbol{u}_i \end{aligned}$$

$$= var(\boldsymbol{D}) - \sum_{i=1}^{r} \lambda_i$$

Первые r главных компонент максимизируют проецируемую дисперсию $var(\boldsymbol{A})$ и, таким образом, они также минимизируют MSE.

Общая дисперсия

Обратите внимание, что общая дисперсия $\boldsymbol{D}$ инвариантна к изменению базисных векторов. Следовательно, имеем следующее тождество:

$$var(\boldsymbol{D}) = \sum_{i=1}^{d} \sigma_i^2 = \sum_{i=1}^{d} \lambda_i$$

Выбор размерности

Часто мы можем не знать, сколько измерений r использовать для хорошей аппроксимации. Одним из критериев выбора r является вычисление доли общей дисперсии, охваченной первыми r главными компонентами

$$f(r) = \frac{\lambda_1 + \lambda_2 + \cdots + \lambda_r}{\lambda_1 + \lambda_2 + \cdots + \lambda_d} = \frac{\sum_{i=1}^{r} \lambda_i}{\sum_{i=1}^{d} \lambda_i} = \frac{\sum_{i=1}^{r} \lambda_i}{var(\boldsymbol{D})} \tag{3.25}$$

Учитывая определенный желаемый порог дисперсии, скажем α, начиная с первого главного компонента, мы продолжаем добавлять дополнительные компоненты и останавливаемся на наименьшем значении r, для которого $f(r) \geq \alpha$. Другими словами, мы выбираем наименьшее количество измерений так, что подпространство, охватываемое этими r измерениями, захватывает по крайней мере долю α общей дисперсии. На практике α обычно устанавливается равным 0.9 или выше, т.ч. уменьшенный набор данных фиксирует не менее 90% общей дисперсии.

Алгоритм 3.1 демонстрирует псевдокод для алгоритма метода главных компонент PCA. Центрируются входные данные $\boldsymbol{D} \in \mathbb{R}^{n \times d}$ вычитанием среднего значение из каждой точки. Затем вычисляются собственные векторы и собственные значения ковариационной матрицы $\boldsymbol{\Sigma}$. Учитывая желаемый порог дисперсии α, выбирается наименьший набор измерений r, который охватывает по меньшей мере долю α общей дисперсии. В заключении, вычисляются координаты каждой точки в новом r-мерном подпространстве главных компонентов, чтобы получить новую матрицу данных $\boldsymbol{A} \in \mathbb{R}^{n \times r}$.

АЛГОРИТМ 3.1. Метод главных компонент PCA

PCA $(\mathbf{D}, \alpha)$:

1 $\boldsymbol{\mu} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i$ // compute mean

2 $\mathbf{Z} = \mathbf{D} - \mathbf{1} \cdot \boldsymbol{\mu}^T$ // center the data

3 $\boldsymbol{\Sigma} = \frac{1}{n}\left(\mathbf{Z}^T\mathbf{Z}\right)$ // compute covariance matrix

4 $(\lambda_1, \lambda_2, \ldots, \lambda_d) = \text{eigenvalues}(\boldsymbol{\Sigma})$ // compute eigenvalues

5 $\mathbf{U} = \begin{pmatrix} \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_d \end{pmatrix} = \text{eigenvectors}(\boldsymbol{\Sigma})$ // compute eigenvectors

6 $f(r) = \frac{\sum_{i=1}^{r} \lambda_i}{\sum_{i=1}^{d} \lambda_i}$, for all $r = 1, 2, \ldots, d$ // fraction of total variance

7 Choose smallest r so that $f(r) \geq \alpha$ // choose dimensionality

8 $\mathbf{U}_r = \begin{pmatrix} \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_r \end{pmatrix}$ // reduced basis

9 $\mathbf{A} = \{\mathbf{a}_i \mid \mathbf{a}_i = \mathbf{U}_r^T \mathbf{x}_i, \text{for } i = 1, \ldots, n\}$ // reduced dimensionality data

3.2.4 Геометрия метода главных компонент

Геометрически, когда $r = d$, PCA соответствует ортогональному изменению базиса, так что общая дисперсия фиксируется суммой дисперсий по каждому из главных направлений $\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_d$, и далее, все ковариации равны нулю. Это можно увидеть, посмотрев на коллективное действие полного набора главных компонентов, которые могут быть организованы в ортогональную матрицу $d \times d$.

$$\boldsymbol{U} = \begin{pmatrix} | & | & & | \\ \boldsymbol{u}_1 & \boldsymbol{u}_2 & \ldots & \boldsymbol{u}_d \\ | & | & & | \end{pmatrix}$$

$$\boldsymbol{U}^{-1} = \boldsymbol{U}^T$$

Каждый главный компонент $\boldsymbol{u}_i$ соответствует собственному вектору ковариационной матрицы $\boldsymbol{\Sigma}$, т.е.

$\boldsymbol{\Sigma}\boldsymbol{u}_i = \lambda_i \boldsymbol{u}_i$ for all $1 \leq i \leq d$

Который можно компактно записать в матричных обозначениях следующим образом:

$$\boldsymbol{\Sigma}\begin{pmatrix} | & | & & | \\ \boldsymbol{u}_1 & \boldsymbol{u}_2 & \ldots & \boldsymbol{u}_d \\ | & | & & | \end{pmatrix} = \begin{pmatrix} | & | & & | \\ \lambda_1\boldsymbol{u}_1 & \lambda_2\boldsymbol{u}_2 & \ldots & \lambda_d\boldsymbol{u}_d \\ | & | & & | \end{pmatrix}$$

$$\boldsymbol{\Sigma}\mathbf{U} = \mathbf{U}\begin{pmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_d \end{pmatrix}$$

$$\boldsymbol{\Sigma}\mathbf{U} = \mathbf{U}\boldsymbol{\Lambda} \tag{3.26}$$

Если мы умножим уравнение (3.26) слева на $\boldsymbol{U}^{-1} = \boldsymbol{U}^T$, то получим

$$\boldsymbol{U}^T\boldsymbol{\Sigma}\mathbf{U} = \boldsymbol{U}^T\mathbf{U}\boldsymbol{\Lambda} = \boldsymbol{\Lambda} = \begin{pmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_d \end{pmatrix}$$

Это означает, что если мы изменим базис на $\boldsymbol{U}$, то ковариационная матрица $\boldsymbol{\Sigma}$ изменится на аналогичную матрицу $\boldsymbol{\Lambda}$, которая фактически является ковариационной матрицей в новом базисе. Тот факт, что $\boldsymbol{\Lambda}$ является диагональной, подтверждает, что после изменения базиса все ковариации исчезают, и остается только дисперсия по каждой из главных компонент, причем дисперсия по каждому новому направлению $\boldsymbol{u}_i$ задается соответствующим собственным значением λ_i.

Стоит отметить, что в новом базисе уравнение

$$\boldsymbol{x}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{x} = 1 \tag{3.27}$$

Определяет d-мерный эллипсоид (или гиперэллипс). Собственные векторы $\boldsymbol{u}_i$ матрицы $\boldsymbol{\Sigma}$, т.е. главные компоненты, являются направлениями главных осей эллипсоида. Квадратные корни собственных значений, т.е. $\sqrt{\lambda_i}$, являются длинами полуосей эллипса.

Умножая (3.26) справа на $\boldsymbol{U}^{-1} = \boldsymbol{U}^T$, имеем

$$\boldsymbol{\Sigma} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T \tag{3.28}$$

Предполагая, что $\boldsymbol{\Sigma}$ обратимая или невырожденная, имеем

$$\boldsymbol{\Sigma}^{-1} = (\boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T)^{-1} = (\boldsymbol{U}^{-1})^T\boldsymbol{\Lambda}^{-1}\boldsymbol{U}^{-1} = \boldsymbol{U}\boldsymbol{\Lambda}^{-1}\boldsymbol{U}^T$$

Где

$$\boldsymbol{\Lambda}^{-1} = \begin{pmatrix} \frac{1}{\lambda_1} & 0 & \cdots & 0 \\ 0 & \frac{1}{\lambda_2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{1}{\lambda_d} \end{pmatrix}$$

Подставив $\boldsymbol{\Sigma}^{-1}$ в уравнение (3.27) и используя тот факт, что $\boldsymbol{x} = \boldsymbol{U}\boldsymbol{a}$ из (3.2), где $\boldsymbol{a} = (a_1, a_2, \ldots, a_d)^T$ представляет координаты $\boldsymbol{x}$ в новом базисе, получаем

$$\boldsymbol{x}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{x} = 1$$

$$(\boldsymbol{a}^T\boldsymbol{U}^T)\boldsymbol{U}\boldsymbol{\Lambda}^{-1}\boldsymbol{U}^T(\boldsymbol{U}\boldsymbol{a}) = 1$$

$$\boldsymbol{a}^T\boldsymbol{\Lambda}^{-1}\boldsymbol{a} = 1$$

$$\sum_{i=1}^{d}\frac{a_i^2}{\lambda_i} = 1$$

что и есть уравнение эллипса с центром в $\boldsymbol{0}$ и длинной полуосей $\sqrt{\lambda_i}$. Таким образом $\boldsymbol{x}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{x} = 1$ или эквивалентно $\boldsymbol{a}^T\boldsymbol{\Lambda}^{-1}\boldsymbol{a} = 1$ в новом базисе главных компонентов определяет d-мерный эллипсоид, где длины полуосей равны стандартным отклонениям (квадратный корень из дисперсии $\sqrt{\lambda_i}$) вдоль каждой оси. Точно так же уравнение $\boldsymbol{x}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{x} = s$ или эквивалентное $\boldsymbol{a}^T\boldsymbol{\Lambda}^{-1}\boldsymbol{a} = s$, для различных значений скалярной величины s, представляет концентрические эллипсоиды.

3.3 Ядерный анализ главных компонент

Анализ главных компонент может быть расширен, чтобы искать нелинейные “направления” в данных с использованием ядерных методов. Ядерный PCA ищет направления наибольшей дисперсии в пространстве признаков вместо входного пространства. То есть, вместо поиска линейных комбинаций во входном пространстве, алгоритм ищет линейные комбинации в пространстве признаков, полученном из нелинейного преобразования входного пространства. Таким образом, линейные главные компоненты пространства признаков соответствуют нелинейным направлениям во входном пространстве. Как мы увидим, с использованием *ядерного трюка* все операции могут быть выполнены в терминах входного пространства, без необходимости преобразования данных в пространстве признаков.

Пусть ϕ – отображение из входного пространства в пространство признаков. Каждой точке в пространстве признаков соответствует образ $\phi(\boldsymbol{x}_i)$ точки $\boldsymbol{x}_i$ во входном пространстве. Во входном пространстве первая главная компонента показывает направление наибольшей дисперсии; её собственный вектор совпадает с наибольшим собственным значением в матрице ковариантности. Таким же образом, через поиск собственного вектора, соответствующий наибольшему собственному значению в матрице ковариантности в пространстве признаков, можно найти первую главную компоненту в пространстве признаков $\boldsymbol{u}_1$ (с $\boldsymbol{u}_1^T\boldsymbol{u}_1 = 1$):

$$\boldsymbol{\Sigma}_{\phi} = \lambda_1 \boldsymbol{u}_1 \tag{3.29}$$

где $\boldsymbol{\Sigma}_{\phi}$ – матрица ковариантности в пространстве признаков, определенная как

$$\boldsymbol{\Sigma}_{\phi} = \frac{1}{n}\sum_{i=1}^{n} \phi(\boldsymbol{x}_i)\phi(\boldsymbol{x}_i)^T \tag{3.30}$$

Здесь предполагается, что точки центрированы, т.е. $\phi(\boldsymbol{x}_i) = \phi(\boldsymbol{x}_i) - \boldsymbol{\mu}_{\phi}$, где $\boldsymbol{\mu}_{\phi}$ – среднее в пространстве признаков.

Подставляя (3.30) в (3.29) получаем:

$$\left(\frac{1}{n}\sum_{i=1}^{n} \phi(\boldsymbol{x}_i)\phi(\boldsymbol{x}_i)^T\right)\boldsymbol{u}_1 = \lambda_1 \boldsymbol{u}_1$$

$$\frac{1}{n}\sum_{i=1}^{n} \phi(\boldsymbol{x}_i)(\phi(\boldsymbol{x}_i)^t \boldsymbol{u}_1) = \lambda_1 \boldsymbol{u}_1 \tag{3.31}$$

$$\sum_{i=1}^{n}\left(\frac{\phi(\boldsymbol{x}_i)^T\boldsymbol{u}_1}{n\lambda_1}\right)\phi(\boldsymbol{x}_i) = \boldsymbol{u}_1$$

$$\sum_{i=1}^{n} c_i \phi(\boldsymbol{x}_i) = \boldsymbol{u}_1 \tag{3.32}$$

где $c_i = \frac{\phi(\boldsymbol{x}_i)^T\boldsymbol{u}_1}{n\lambda_1}$ – скалярная величина. Из (3.32) видно, что лучшее направление в пространстве признаков, $\boldsymbol{u_1}$ есть линейная комбинация преобразованных точек, где скаляры c_i показывают важность каждой точки в направлении наибольшей дисперсии.

Теперь можно подставить (3.32) назад в (3.31) и получить

$$\left(\frac{1}{n}\sum_{i=1}^{n}\boldsymbol{\phi}(\boldsymbol{x}_i)\boldsymbol{\phi}(\boldsymbol{x}_i)^T\right)\left(\sum_{i=1}^{n}c_j\boldsymbol{\phi}(\boldsymbol{x}_i)\right)=\lambda_1\sum_{i=1}^{n}c_i\boldsymbol{\phi}(\boldsymbol{x}_i)$$

$$\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{n}c_j\boldsymbol{\phi}(\boldsymbol{x}_i)\boldsymbol{\phi}(\boldsymbol{x}_i)^T\boldsymbol{\phi}(\boldsymbol{x}_j)=\lambda_1\sum_{i=1}^{n}c_i\boldsymbol{\phi}(\boldsymbol{x}_i)$$

$$\sum_{i=1}^{n}\left(\boldsymbol{\phi}(\boldsymbol{x}_i)\sum_{j=1}^{n}c_j\boldsymbol{\phi}(\boldsymbol{x}_i)^T\boldsymbol{\phi}(\boldsymbol{x}_j)\right)=n\lambda_1\sum_{i=1}^{n}c_i\boldsymbol{\phi}(\boldsymbol{x}_i)$$

В предыдущем уравнении можно заменить скалярное произведение в пространстве признаков, $(\boldsymbol{x}_i)^T\boldsymbol{\phi}(\boldsymbol{x}_j)$, соответствующей ядерной функцией во входном пространстве, а именно $K(\boldsymbol{x}_i,\boldsymbol{x}_j)$

$$\sum_{i=1}^{n}\left(\boldsymbol{\phi}(\boldsymbol{x}_i)\sum_{j=1}^{n}c_jK(\boldsymbol{x}_i,\boldsymbol{x}_j)\right)=n\lambda_1\sum_{i=1}^{n}c_i\boldsymbol{\phi}(\boldsymbol{x}_i) \tag{3.33}$$

Здесь предполагается, что точки в пространстве признаков уже центрированы, т.е. ядерная матрица была центрирована через уравнение (2.14):

$$\boldsymbol{K}=\left(\boldsymbol{I}-\frac{1}{n}\mathbf{1}_{n\times n}\right)\boldsymbol{K}\left(\boldsymbol{I}-\frac{1}{n}\mathbf{1}_{n\times n}\right)$$

где $\boldsymbol{I}$ **–** единичная матрица, у которой элементы главной диагонали 1, а остальные 0, и $\mathbf{1}_{n\times n}$ – матрица $n\times n$, у которой все элементы – 1.

Мы заменили одно скалярное произведение ядерной функцией. Чтобы убедится, что все вычисления в пространстве признаков выполняются только в терминах скалярного произведение, возьмем любую точку $\boldsymbol{\phi}(\boldsymbol{x}_k)$ и умножим (3.33) на неё с обеих сторон чтобы получить

$$\begin{aligned}\sum_{i=1}^{n}\left(\boldsymbol{\phi}(\boldsymbol{x}_k)\boldsymbol{\phi}(\boldsymbol{x}_i)\sum_{j=1}^{n}c_jK(\boldsymbol{x}_i,\boldsymbol{x}_j)\right)&=n\lambda_1\sum_{i=1}^{n}c_i\boldsymbol{\phi}(\boldsymbol{x}_i)\boldsymbol{\phi}(\boldsymbol{x}_k)\\ \sum_{i=1}^{n}\left(K(\boldsymbol{x}_k,\boldsymbol{x}_i)\sum_{j=1}^{n}c_jK(\boldsymbol{x}_i,\boldsymbol{x}_j)\right)&=n\lambda_1\sum_{i=1}^{n}c_iK(\boldsymbol{x}_k,\boldsymbol{x}_i)\end{aligned} \tag{3.34}$$

Затем, обозначим $\boldsymbol{K}_i$ строку i центрированной ядерной матрицы, записанную как вектор-столбец:

$$\boldsymbol{K}_i=(K(\boldsymbol{x}_i,\boldsymbol{x}_1)\quad K(\boldsymbol{x}_i,\boldsymbol{x}_2)\quad\cdots\quad K(\boldsymbol{x}_i,\boldsymbol{x}_n))^T$$

Пусть $\boldsymbol{c}$ обозначает вектор-столбец весов

$$\boldsymbol{c}=(c_1\quad c_2\quad\cdots\quad c_n)^T$$

Теперь можно записать (3.34) как:

$$\sum_{i=1}^{n} K(\boldsymbol{x}_k, \boldsymbol{x}_i)\boldsymbol{K}_i^T\boldsymbol{c} = n\lambda_1\boldsymbol{K}_k^T\boldsymbol{c}$$

На самом деле, поскольку можно выбрать любые из n точек $\phi(\boldsymbol{x}_k)$ в пространстве признаков, чтобы получить (3.34), имеется набор уравнений:

$$\begin{aligned}
\sum_{i=1}^{n} K(\boldsymbol{x}_1, \boldsymbol{x}_i)\boldsymbol{K}_i^T\boldsymbol{c} &= n\lambda_1\boldsymbol{K}_1^T\boldsymbol{c} \\
\sum_{i=1}^{n} K(\boldsymbol{x}_2, \boldsymbol{x}_i)\boldsymbol{K}_i^T\boldsymbol{c} &= n\lambda_1\boldsymbol{K}_2^T\boldsymbol{c} \\
\vdots &= \vdots \\
\sum_{i=1}^{n} K(\boldsymbol{x}_2, \boldsymbol{x}_i)\boldsymbol{K}_i^T\boldsymbol{c} &= \sum_{i=1}^{n} K(\boldsymbol{x}_k, \boldsymbol{x}_i)\boldsymbol{K}_n^T\boldsymbol{c}
\end{aligned}$$

Можно представить эти n уравнений как:

$$\boldsymbol{K}^2\boldsymbol{c} = n\lambda_1\boldsymbol{K}\boldsymbol{c}$$

где $\boldsymbol{K}$ – центрированная ядерная матрица. Умножением на $\boldsymbol{K}^{-1}$ получаем:

$$\begin{aligned}
\boldsymbol{K}^{-1}\boldsymbol{K}^2\boldsymbol{c} &= n\lambda_1\boldsymbol{K}^{-1}\boldsymbol{K}\boldsymbol{c} \\
\boldsymbol{K}\boldsymbol{c} &= n\lambda_1\boldsymbol{c} \\
\boldsymbol{K}\boldsymbol{c} &= \eta_1\boldsymbol{c}
\end{aligned} \tag{3.35}$$

где $\eta_1 = n\lambda_1$. Таким образом, вектор весов $\boldsymbol{c}$ есть собственный вектор, соответствующий наибольшему собственному значению η_1 ядерной матрицы $\boldsymbol{K}$.

Как только найдено $\boldsymbol{c}$, её можно подставить обратно в (3.32), чтобы получить первую ядерную основную компоненту $\boldsymbol{u}_1$. Единственное ограничение – надо нормализовать $\boldsymbol{u}_1$ до единичного вектора следующим образом:

$$\begin{aligned}
\boldsymbol{u}_1^T\boldsymbol{u}_1 &= 1 \\
\sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j \phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x}_j) &= 1 \\
\boldsymbol{c}^T\boldsymbol{K}\boldsymbol{c} &= 1
\end{aligned}$$

Из (3.35) имеем:

$$\begin{aligned}
\boldsymbol{c}^T(\eta_1\boldsymbol{c}) &= 1 \\
\eta_1\boldsymbol{c}^T\boldsymbol{c} &= 1 \\
\|\boldsymbol{c}\|^2 &= \frac{1}{\eta_1}
\end{aligned}$$

Однако, поскольку $\boldsymbol{c}$ – собственный вектор **K,** он будет иметь единичную норму. То есть, чтобы удостоверится, что $\boldsymbol{u}_1$ – единичный вектор, надо масштабировать вектор весов $\boldsymbol{c}$ так, что его норма $\|c\| = \sqrt{\frac{1}{\eta_1}}$, что можно получить, домножив $\boldsymbol{c}$ на $\sqrt{\frac{1}{\eta_1}}$.

В общем случае, поскольку мы не отображаем входные точки в пространство признаков через ϕ, невозможно непосредственно вычислить главное направление в терминах $\phi(\boldsymbol{x}_i)$, как показано в (3.32). Однако, здесь важно, что любую точку $\phi(\boldsymbol{x})$ можно отобразить на главное направление $\boldsymbol{u}_1$ следующим образом:

$$\boldsymbol{u}_1^T \phi(\boldsymbol{x}) = \sum_{i=1}^{n} c_i \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}) = \sum_{i=1}^{n} c_i K(\boldsymbol{x}_i, \boldsymbol{x})$$

что требует только ядерные операции. Когда $\boldsymbol{x} = \boldsymbol{x}_i$ – одна из входных точек, проекция $\phi(\boldsymbol{x}_i)$ на главную компоненту $\boldsymbol{u}_1$ может быть записана как скалярное произведение

$$\boldsymbol{a}_i = \boldsymbol{u}_1^T \phi(\boldsymbol{x}_i) = \boldsymbol{K}_i^T \boldsymbol{c} \tag{3.36}$$

где $\boldsymbol{K}_i$ – вектор-столбец, соответствующий i-й строке в ядерной матрице. Таким образом, мы показали, что все вычисления, или для решения главной компоненты, или для проекции точек, могут быть выполнены с использованием одной только ядерной функции. Затем, мы можем получить дополнительные главные компоненты путём решения 3.35 для других собственных значений и собственных векторов. Другими словами, отсортировав собственные значения $\boldsymbol{K}$ в порядке убывания $\eta_1 \geq \eta_2 \geq \cdots \geq \eta_n \geq 0$, можно получить j-ю главную компоненту как соответствующий собственный вектор $\boldsymbol{c}_j$, который должен быть нормализован по норме $\|\boldsymbol{c}_j\| = \sqrt{\frac{1}{n}}$, при условии $\eta_j > 0$. И поскольку $\eta_j = n\lambda_j$, дисперсия по j-й главной компоненте определяется как $\lambda_j = \frac{\eta_j}{n}$. Алгоритм 3.2 приводит псевдокод для метода ядерного PCA.

Алгоритм 3.2. Ядерный анализ главных компонент

KERNELPCA $(\mathbf{D}, K, \alpha)$:

1 $\mathbf{K} = \left\{K(\mathbf{x}_i, \mathbf{x}_j)\right\}_{i,j=1,\ldots,n}$ // compute $n \times n$ kernel matrix

2 $\mathbf{K} = (\mathbf{I} - \frac{1}{n}\mathbf{1}_{n\times n})\mathbf{K}(\mathbf{I} - \frac{1}{n}\mathbf{1}_{n\times n})$ // center the kernel matrix

3 $(\eta_1, \eta_2, \ldots, \eta_d) = \text{eigenvalues}(\mathbf{K})$ // compute eigenvalues

4 $\begin{pmatrix}\mathbf{c}_1 & \mathbf{c}_2 & \cdots & \mathbf{c}_n\end{pmatrix} = \text{eigenvectors}(\mathbf{K})$ // compute eigenvectors

5 $\lambda_i = \frac{\eta_i}{n}$ for all $i = 1, \ldots, n$ // compute variance for each component

6 $\mathbf{c}_i = \sqrt{\frac{1}{\eta_i}} \cdot \mathbf{c}_i$ for all $i = 1, \ldots, n$ // ensure that $\mathbf{u}_i^T\mathbf{u}_i = 1$

7 $f(r) = \frac{\sum_{i=1}^{r}\lambda_i}{\sum_{i=1}^{d}\lambda_i}$, for all $r = 1, 2, \ldots, d$ // fraction of total variance

8 Choose smallest r so that $f(r) \geq \alpha$ // choose dimensionality

9 $\mathbf{C}_r = \begin{pmatrix}\mathbf{c}_1 & \mathbf{c}_2 & \cdots & \mathbf{c}_r\end{pmatrix}$ // reduced basis

10 $\mathbf{A} = \{\mathbf{a}_i \mid \mathbf{a}_i = \mathbf{C}_r^T\mathbf{K}_i, \text{for } i = 1, \ldots, n\}$ // reduced dimensionality data

3.4. Сингулярное разложение

Анализ главных компонентов является частным случаем более общего метода разложения матриц, называемого сингулярным разложением (Singular Value Decomposition - SVD). Мы знаем, что по ур-ю 3.28 PCA дает следующее разложение ковариационной матрицы:

$$\Sigma = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T \quad (3.37)$$

где ковариационная матрица была представлена через ортогональную матрицу $\boldsymbol{U}$, содержащую ее собственные вектор и диагональную матрицу $\boldsymbol{\Lambda}$, содержащую ее собственные числа (отсортированные в порядке убывания). SVD обобщает приведенную выше факторизацию для любой матрицы. В частности, для матрицы данных $\boldsymbol{D}$ размера $n \times d$ с n точками и d столбцами SVD факторизует $\boldsymbol{D}$ следующим образом:

$$\boldsymbol{D} = \boldsymbol{L}\Delta\boldsymbol{R}^{\boldsymbol{T}} \quad (3.38)$$

где $\boldsymbol{L}$ - ортогональная матрица размера $n \times n$, $\boldsymbol{R}$ - ортогональная матрица размера $d \times d$, а $\boldsymbol{\Delta}$ - «диагональная» матрица $n \times d$. Столбцы $\boldsymbol{L}$ называются левыми сингулярными векторами, а столбцы $\boldsymbol{R}$ (или строки $\boldsymbol{R}^{\boldsymbol{T}}$) называются правыми сингулярными векторами. Матрица $\boldsymbol{\Delta}$ определяется как

$$\boldsymbol{\Delta}(\mathrm{i},\mathrm{j}) = \begin{cases}\delta_i, & \text{если } i = j \\ 0, & \text{если } i \neq j\end{cases}$$

где $i = 1, \ldots, n$ и $j = 1, \ldots, d$. Элементы $\boldsymbol{\Delta}(\mathrm{i},\mathrm{i}) = \delta_i$ вдоль главной диагонали $\boldsymbol{\Delta}$ называются сингулярными числами $\boldsymbol{D}$, и все они неотрицательны. Если ранг $\boldsymbol{D}$ равен $r \leq \min(n, d)$, то существует только r ненулевых особых значений, которые, как мы предполагаем, упорядочены следующим образом:

$$\delta_1 \geq \delta_2 \geq \cdots \geq \delta_r \geq 0$$

Можно отбросить эти левые и правые сингулярные вектора, которые соответствуют нулевым сингулярным значениям, чтобы получить сокращенный SVD в следующем виде:

$$\boldsymbol{D} = \boldsymbol{L_r \Delta_r R_r^T} \quad (3.39)$$

где $\boldsymbol{L_r}$ - матрица левых сингулярных векторов размера $n \times r$, $\boldsymbol{R_r}$ - матрица правых сингулярных векторов размера $d \times r$, а $\boldsymbol{\Delta_r}$ - диагональная матрица размера $r \times r$, содержащая положительные сингулярные вектора. Уменьшенный SVD приводит непосредственно к спектральному разложению $\boldsymbol{D}$, заданному как

$$\boldsymbol{D} = \boldsymbol{L_r \Delta_r R_r^T} = \begin{pmatrix} | & | & & | \\ l_1 & l_2 & \dots & l_r \\ | & | & & | \end{pmatrix} \begin{pmatrix} \delta_1 & 0 & \cdots & 0 \\ 0 & \delta_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \delta_r \end{pmatrix} \begin{pmatrix} - & r_1^T & - \\ - & r_2^T & - \\ - & \vdots & - \\ - & r_r^T & - \end{pmatrix}$$

$$= \delta_1 l_1 r_1^T + \delta_2 l_2 r_2^T + \cdots + \delta_r l_r r_r^T = \sum_{i=1}^{r} \delta_i l_i r_i^T$$

Спектральное разложение $\boldsymbol{D}$ представляет собой сумму матриц первого ранга вида $\delta_i l_i r_i^T$. Выбрав q наибольших сингулярных значений $\delta_1, \delta_2, \dots, \delta_q$ и соответствующие левый и правый сингулярные вектора, мы получаем наилучшее приближение ранга q к исходной матрице $\boldsymbol{D}$. То есть, если $\boldsymbol{D_q}$ - матрица, определенная как

$$\boldsymbol{D_q} = \sum_{i=1}^{r} \delta_i l_i r_i^T$$

то можно показать, что $\boldsymbol{D_q}$ - это матрица ранга q, которая минимизирует следующее выражение

$$\left\| \boldsymbol{D} - \boldsymbol{D_q} \right\|_F$$

где $\|\boldsymbol{A}\|_F$ - норма Фробениуса матрицы $\boldsymbol{A}$ размера $n \times d$, определяемая как

$$\|\boldsymbol{A}\|_F = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{d} \boldsymbol{A}(i,j)^2}$$

3.4.1. Геометрия сингулярного разложения

В общем, любая матрица $\boldsymbol{D}$ размера $n \times d$ представляет собой линейное преобразование $\boldsymbol{D}: \mathbb{R}^d \to \mathbb{R}^n$ из пространства d-мерных векторов в пространство n-мерных векторов, поскольку для любого $x \in \mathbb{R}^d$ существует $y \in \mathbb{R}^n$ такое, что

$$\boldsymbol{Dx} = \boldsymbol{y}$$

Множество всех векторов $y \in \mathbb{R}^n$ таких, что $\boldsymbol{Dx} = \boldsymbol{y}$ по всем возможными $x \in \mathbb{R}^d$, называется пространством столбцов $\boldsymbol{D}$, а множество всех векторов $x \in \mathbb{R}^d$, таких что $\boldsymbol{D^T y} = \boldsymbol{x}$ по всем $y \in \mathbb{R}^n$, называется пространство строк $\boldsymbol{D}$, что эквивалентно пространству столбцов $\boldsymbol{D^T}$. Другими словами, пространство столбцов $\boldsymbol{D}$ — это набор всех векторов, которые могут быть получены как линейные комбинации столбцов $\boldsymbol{D}$, а пространство строк $\boldsymbol{D}$ - это набор всех

векторов, которые могут быть получены как линейные комбинации строк $\boldsymbol{D}$ (или столбцов $\boldsymbol{D^T}$). Также обратите внимание, что множество всех векторов $x \in \mathbb{R}^d$, таких что $\boldsymbol{Dx} = \boldsymbol{0}$, называется нулевым пространством $\boldsymbol{D}$, и, наконец, множество всех векторов $y \in \mathbb{R}^n$, таких что $\boldsymbol{D^T y} = \boldsymbol{0}$, называется левым нулевым пространством $\boldsymbol{D}$.

Одним из основных свойств сингулярного разложения является то, что оно дает основу для каждого из четырех фундаментальных пространств, связанных с матрицей $\boldsymbol{D}$. Если $\boldsymbol{D}$ имеет ранг r, это означает, что у него есть только r независимых столбцов и r независимых строк. Таким образом, r левых сингулярных векторов $l_1, l_2, \dots, l_r$, соответствующие r ненулевым сингулярным значениям $\boldsymbol{D}$ в ур-и (3.38), представляют собой основу для пространства столбцов $\boldsymbol{D}$. Остальные $n - r$ левых сингулярных векторов $l_{r+1}, l_{r+2}, \dots, l_n$, представляют собой основу для левого нулевого пространства $\boldsymbol{D}$. Для пространства строк r правых сингулярных векторов $r_1, r_2, \dots, r_r$, соответствующие r ненулевым сингулярным значениям, представляют собой базис для пространства строк $\boldsymbol{D}$, а оставшиеся $d - r$ правых сингулярных векторов r_j $(j = r + 1, \dots, d)$ представляют собой базис для нулевое пространство $\boldsymbol{D}$.

Рассмотрим сокращенное представление сингулярного разложения из ур-я (3.39). Умножив обе части уравнения справа на $\boldsymbol{R_r}$ и зная, что $\boldsymbol{R_r^T R_r} = \boldsymbol{I}_r$, где $\boldsymbol{I}_r$ - единичная матрица размера $r \times r$, имеем

$$\boldsymbol{DR}_r = \boldsymbol{L_r \Delta_r R_r^T R}_r$$

$$\boldsymbol{DR}_r = \boldsymbol{L_r \Delta_r}$$

$$\boldsymbol{DR}_r = \boldsymbol{L_r} \begin{pmatrix} \delta_1 & 0 & \cdots & 0 \\ 0 & \delta_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \delta_r \end{pmatrix}$$

$$\boldsymbol{D} \begin{pmatrix} | & | & & | \\ r_1 & r_2 & \dots & r_r \\ | & | & & | \end{pmatrix} = \begin{pmatrix} | & | & & | \\ \delta_1 l_1 & \delta_2 l_2 & \dots & \delta_r l_r \\ | & | & & | \end{pmatrix}$$

Из вышесказанного получаем, что

$\boldsymbol{DR}_i = \delta_i l_i$ для всех $i = 1, \dots, r$

Другими словами, SVD — это специальная факторизация матрицы $\boldsymbol{D}$, так что любой базисный вектор r_i для пространства строк отображается в соответствующий базисный вектор l_i в пространстве столбцов, масштабируемый сингулярным значением δ_i. Таким образом, мы можем рассматривать SVD как отображение ортонормированного базиса $(r_1, r_2, \dots, r_r)$ $\mathbb{R}^d$ (пространство строк) в ортонормированный базис $(l_1, l_2, \dots, l_r)$ $\mathbb{R}^n$ (пространство столбцов) с соответствующими осями, масштабируемыми согласно сингулярным значениям $(\delta_1, \delta_2, \dots, \delta_r)$.

3.4.2. Связь между SVD и PCA

Предположим, что матрица $\boldsymbol{D}$ центрирована, и, что она разложена с помощью SVD [ур-е. (3.38)] как $\boldsymbol{D} = \boldsymbol{L \Delta R^T}$. Рассмотрим матрицу рассеяния для $\boldsymbol{D}$, заданную как $\boldsymbol{D^T D}$. Имеем

$$\boldsymbol{D^T D} = (\boldsymbol{L \Delta R^T})^{\boldsymbol{T}} (\boldsymbol{L \Delta R^T}) = \boldsymbol{R} \Delta^T \boldsymbol{L^T L \Delta R^T} = \boldsymbol{R}(\Delta^T \Delta) \boldsymbol{R^T} = \boldsymbol{R}(\Delta_d^2) \boldsymbol{R^T} \quad (3.40)$$

где Δ_d^2 - диагональная матрица $d \times d$, определенная как $\Delta_d^2(i,i) = \delta_i^2$ для $i = 1, \ldots, d$. Причем только $r \leq \min(d,n)$ из этих собственных чисел положительны, а остальные - нули.

Поскольку ковариационная матрица центрированного $\boldsymbol{D}$ задается как $\Sigma = \frac{\mathbf{1}}{\boldsymbol{n}}\boldsymbol{D}^T\boldsymbol{D}$ и поскольку ее можно разложить как $\Sigma = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T$ через PCA [ур-е. (3.37)], имеем

$$\boldsymbol{D}^{\boldsymbol{T}}\boldsymbol{D} = n\Sigma = n\boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T = \boldsymbol{U}(n\boldsymbol{\Lambda})\boldsymbol{U}^T (3.41)$$

Приравнивая уравнение (3.40) и уравнение (3.41), мы получаем, что правые сингулярные вектора $\boldsymbol{R}$ совпадают с собственными векторами Σ. Кроме того, соответствующие сингулярные значения $\boldsymbol{D}$ связаны с собственными числами Σ через выражение

$n\lambda_i = \delta_i^2$ или $\lambda_i = \frac{\delta_i^2}{n}$ для $i = 1, \ldots, d$ (3.42)

Рассмотрим теперь матрицу $\boldsymbol{D}\boldsymbol{D}^{\boldsymbol{T}}$. Имеем

$$\boldsymbol{D}\boldsymbol{D}^{\boldsymbol{T}} = (\boldsymbol{L}\Delta\boldsymbol{R}^{\boldsymbol{T}})(\boldsymbol{L}\Delta\boldsymbol{R}^{\boldsymbol{T}})^{\boldsymbol{T}} = \boldsymbol{L}\Delta\boldsymbol{R}^{\boldsymbol{T}}\boldsymbol{R}\Delta^T\boldsymbol{L}^{\boldsymbol{T}} = \boldsymbol{L}(\Delta\Delta^T)\boldsymbol{L}^{\boldsymbol{T}} = \boldsymbol{L}\Delta_n^2\boldsymbol{L}^{\boldsymbol{T}}$$

где Δ_n^2 - диагональная матрица $d \times d$, определенная как $\Delta_n^2(i,i) = \delta_i^2$ для $i = 1, \ldots, d$. Только r из этих сингулярных значений положительны, а все остальные - нули. Таким образом, левые сингулярные вектора $\boldsymbol{L}$ являются собственными векторами матрицы $n \times n\, \boldsymbol{D}\boldsymbol{D}^{\boldsymbol{T}}$, а соответствующие собственные числа задаются как δ_i^2.

Глава 4. Поиск наборов объектов

Во многих практических задачах необходимо понять, насколько часто два или более объекта связаны между собой. Рассмотрим, например, популярный веб-сайт, который логирует весь входящий трафик в форме неких журналов посещений. Такие журналы посещений обычно хранят исходную страницу, страницу, запрошенную пользователем, время посещения, код возврата и так далее. Имея такие журналы, можно задаться вопросом, есть ли такие страницы, которые пользователи просматривают при каждом посещении сайта. Такие часто встречающиеся наборы страниц могут объяснить поведение пользователей на сайте и позволить улучшить пользовательский опыт.

Задача поиска паттернов встречается во многих сферах. Типичным примером является анализ корзины покупок, то есть, поиск таких наборов предметов, которые чаще всего покупают вместе в супермаркете. После анализа таких часто встречающихся наборов мы можем извлечь ассоциативные правила, дающие некоторую информацию о том, насколько вероятна покупка двух наборов одновременно, или при каких условиях такая покупка возможна. Например, при анализе логов сайта мы можем извлечь такое правило: «Пользователи, которые посещают набор из главной страницы, страницы ноутбуков и скидок, также часто посещают страницу корзины и оформления заказа». Отсюда мы можем предположить, что скидки повлияли на увеличение продаж ноутбуков. В случае корзины покупок можно извлечь правила вроде такого: «Клиенты, покупающие молоко и хлопья, склонны покупать бананы». По такому правилу продуктовый магазин может перенести бананы поближе к полке с хлопьями. Мы начнём эту главу с алгоритмов поиска часто встречающихся наборов, а затем покажем, как их можно использовать для извлечения ассоциативных правил.

4.1 Часто встречающиеся наборы и ассоциативные правила

4.1.1 Наборы объектов и наборы тидов

Пусть $\mathfrak{I} = \{x_1, x_2, \dots, x_m\}$ – множество элементов, называемых *объектами*. Множество $X \subseteq \mathfrak{I}$ будем называть *набором объектов*. Множеством объектов $\mathfrak{I}$ может быть, например, множество всех страниц веб-сайта или продуктов в магазине. Набор мощности (или размера) k будем называть k-набором. Далее, обозначим $\mathfrak{I}^{(k)}$ множество всех k-наборов, то есть множество всех подмножеств $\mathfrak{I}$ размера k. Пусть $\mathfrak{T} = \{t_1, t_2, \dots, t_n\}$ – другое множество элементов, называемых *идентификаторами транзакций или тидами* (англ. transaction identifiers). Множество $T \subseteq \mathfrak{T}$ назовём *набором тидов*. Будем полагать, что наборы объектов и наборы тидов отсортированы в лексикографическом порядке.

Транзакция представляет из себя кортеж вида $\langle t, X \rangle$, где $t \in \mathfrak{T}$ – уникальный идентификатор транзакции, а X – набор объектов. Множество идентификаторов $\mathfrak{T}$ может быть множеством всех покупателей в супермаркете, посетителей веб-сайта и так далее. Для удобства будем ссылаться на транзакцию $\langle t, X \rangle$ по её идентификатору t.

4.1.2 Представление базы данных

Двоичная база данных $\mathbf{D}$ – это двоичное отношение над набором объектов и идентификаторов транзакций, то есть, $\mathbf{D} \subseteq \mathfrak{T} \times \mathfrak{I}$. Будем говорить, что идентификатор $t \in \mathfrak{T}$ *содержит* объект $x \in \mathfrak{I}$ тогда и только тогда, когда $(t, x) \in \mathbf{D}$. Другими словами, $(t, x) \in \mathbf{D}$ тогда и только тогда, когда $x \in X$ входит в кортеж $\langle t, X \rangle$. Будем говорить, что идентификатор t содержит набор $X = \{x_1, x_2, \dots, x_k\}$ тогда и только тогда, когда $(t, x_i) \in \mathbf{D}$ для всех $i = 1, 2, \dots, k$.

Для множества X обозначим через 2^X множество степеней X, то есть множество всех подмножеств X. Пусть $\mathbf{i} : 2^{\mathfrak{T}} \to 2^{\mathfrak{I}}$ - функция, определённая следующим образом:

$$\mathbf{i}(T) = \{x \mid \forall t \in T, t \text{ содержит } X\}, \quad (4.1)$$

где $T \subseteq \mathfrak{T}$, а $\mathbf{i}(T)$ – множество объектов, общих для всех транзакций в наборе T. В частности, $\mathbf{i}(t)$ – набор объектов, содержащихся в транзакции $t \in \mathfrak{T}$. Заметим, что в этой части мы опускаем нотацию множеств для удобства (пишем $\mathbf{i}(t)$ вместо $\mathbf{i}(\{t\})$). Иногда удобно рассматривать двоичную базу данных $\mathbf{D}$ как *базу данных транзакций*, состоящую из кортежей вида $\langle t, \mathbf{i}(t)\rangle$, где $t \in \mathfrak{T}$. База данных транзакций или наборов объектов может рассматриваться как горизонтальное представление двоичной базы данных, в котором мы опускаем объекты, не содержащиеся в данной транзакции.

Пусть $t : 2^{\mathfrak{I}} \to 2^{\mathfrak{T}}$ – функция, определённая следующим образом:

$$\mathbf{t}(X) = \{t \mid t \in T \text{ и } t \text{ содержит } X\}, \quad (4.2)$$

где $X \subseteq \mathfrak{I}$, а $\mathbf{t}(X)$ – набор идентификаторов транзакций, который содержит все объекты из в наборе X. В частности, $\mathbf{t}(x)$ – набор идентификаторов, которые содержат единственный объект $x \in \mathfrak{I}$. Иногда базу данных $\mathbf{D}$ удобно рассматривать как базу данных идентификаторов, содержащую последовательность кортежей вида $\langle x, \mathbf{t}(x)\rangle$, где $x \in \mathfrak{I}$. База данных наборов идентификаторов – это вертикальное представление двоичной базы, где мы опускаем идентификаторы, которые не содержат данный объект.

4.1.3 Поддержка и часто встречающиеся наборы элементов

Поддержка набора X в наборе данных $\mathbf{D}$, обозначаемая как $sup(X, \mathbf{D})$ – это число транзакций в $\mathbf{D}$, которые содержат X:

$$sup(X, \boldsymbol{D}) = |\{t \mid \langle t, \mathbf{i}(t)\rangle \in \mathbf{D} \text{ и } X \subseteq \mathbf{i}(t)\}| = |\mathbf{t}(X)|.$$

Относительная поддержка X – это доля транзакций, содержащих X:

$$rsup(X, \boldsymbol{D}) = \frac{sup(X, \boldsymbol{D})}{|\boldsymbol{D}|}$$

Это оценка *совместной вероятности* элементов из X.

Набор X считается частым в $\boldsymbol{D}$, если $sup(X, \boldsymbol{D}) \geq minsup$, где $minsup$ – определяемый пользователем *минимальный уровень поддержки*. Если нет путаницы относительно базы данных $\boldsymbol{D}$, мы будем обозначать поддержку и относительную поддержку как $sup(X)$ и $rsup(X)$ соответственно. Если $minsup$ определяется дробью, предполагается относительная поддержка. Мы будем обозначать $\mathfrak{F}$ множество всех часто встречающихся наборов объектов, а $\mathfrak{F}^k$ – множество всех часто встречающихся k-наборов.

4.1.4 Ассоциативные правила

Ассоциативное правило – это выражение вида: $X \xrightarrow{s,c} Y$, где X и Y – непересекающиеся наборы объектов, то есть, $X, Y \subseteq \mathfrak{I}$ и $X \cap Y = \emptyset$. Обозначим набор $X \cap Y$ как XY. *Поддержкой* правила называется число транзакций, в которых X и Y встречаются вместе как подмножества:

$$s = sup(X \to Y) = |t(XY)| = sup(XY).$$

Относительная поддержка правила – доля транзакций, в которые входят X и Y, и она даёт оценку совместной вероятности X и Y:

$$rsup(X \to Y) = \frac{sup(XY)}{|\boldsymbol{D}|} = P(X \wedge Y).$$

Уровень доверия правила – условная вероятность того, что транзакция содержит Y, при условии, что она содержит X:

$$c = conf(X \to Y) = P(Y|X) = \frac{P(X \wedge Y)}{P(X)} = \frac{sup(XY)}{sup(X)}.$$

Правило считается *часто встречающимся*, если часто встречается набор объектов XY, то есть, $sup(XY) \geq minsup$, и *сильным*, если $conf \geq minconf$, где $minconf$ – заданный пользователем минимальный уровень доверия.

4.1.5 Майнинг наборов данных и правил

Из определения поддержки и уровня доверия правил мы можем заметить, что для генерации частых и сильных правил необходимо найти все часто встречающиеся наборы объектов, а также значение их поддержки. Формально, имея двоичную базу данных $\boldsymbol{D}$ и заданный пользователем минимальный уровень поддержки $minsup$, задача поиска часто встречающихся наборов объектов сводится к перечислению всех частых наборов, то есть, таких, у которых уровень поддержки не ниже $minsup$. Далее, имея множество часто встречающихся наборов $\mathfrak{F}$ и минимальный уровень доверия $minconf$, задача поиска ассоциативных правил сводится к нахождению всех часто встречающихся сильных правил.

4.2 Алгоритмы майнинга набора данных

Начнем с описания наивного (brute-force) алгоритма, который перебирает всевозможные $X \subseteq \mathcal{I}$, и для каждого подмножества определяет его поддержку в наборе данных $\boldsymbol{D}$. Алгоритм состоит из двух шагов: (1) генерация кандидатов и (2) вычисление поддержки.

4.2.1 Генерация кандидатов

Этот шаг генерирует все подмножества $\mathcal{I}$, называемые *кандидатами*, т.к. каждый набор объектов (itemset) потенциально – кандидат в частый образец. Пространство поиска кандидатов экспоненциально, т.к. есть $2^{|\mathcal{I}|}$ потенциально частых образцов. Полезно также отметить структуру пространства поиска наборов объектов; множество всех наборов объектов составляет решетчатую структуру, где два набора X и Y связаны, если X есть *непосредственное подмножество* Y, т.е. $X \subseteq Y$ и $|X| = |Y| - 1$. С точки зрения стратегии поиска, можно перебрать наборы в решетке через поиск в ширину (BFS, breadth-first) или в глубину (DFS, depth-first) на *префиксном дереве*, X и Y связаны, если X – непосредственное подмножество Y и префикс Y. Таким образом, можно перебрать наборы начиная с пустого множества, добавляя по одному на шаге.

4.2.2 Вычисление поддержки

На этом шаге вычисляется поддержка каждого кандидата X и определяется, является ли X частым образцом. Для каждой транзакции $\langle t, \boldsymbol{i}(t)\rangle$ в базе определяется, является ли X подмножеством $\boldsymbol{i}(t)$. Если да, поддержка X увеличивается на 1.

Псевдокод метода показан в алгоритме 4.1. Происходит перечисление наборов объектов $X \subseteq \mathcal{I}$, и для каждого набора объектов вычисляется поддержка проверкой $X \subseteq \boldsymbol{i}(t)$ для каждого $t \in \mathcal{T}$.

Алгоритм 4.1. Brute-force алгоритм

BRUTEFORCE (D, $\mathcal{I}$, *minsup*):
1 $\mathcal{F} \leftarrow \emptyset$ // set of frequent itemsets
2 **foreach** $X \subseteq \mathcal{I}$ **do**
3 $sup(X) \leftarrow$ COMPUTESUPPORT $(X, \mathbf{D})$
4 **if** $sup(X) \geq minsup$ **then**
5 $\mathcal{F} \leftarrow \mathcal{F} \cup \{(X, sup(X))\}$
6 **return** $\mathcal{F}$

COMPUTESUPPORT (X, D):
7 $sup(X) \leftarrow 0$
8 **foreach** $\langle t, \mathbf{i}(t)\rangle \in \mathbf{D}$ **do**
9 **if** $X \subseteq \mathbf{i}(t)$ **then**
10 $sup(X) \leftarrow sup(X) + 1$
11 **return** $sup(X)$

4.3 Вычислительная сложность

Вычисление поддержки в худшем случае $O(|\mathcal{I}| \cdot |\boldsymbol{D}|)$, и поскольку есть $O\left(2^{|\mathcal{I}|}\right)$ возможных кандидатов, вычислительная сложность метода brute-force $O\left(2^{|\mathcal{I}|} \cdot \mathcal{I} \cdot |\boldsymbol{D}|\right)$. Т.к. база данных $\boldsymbol{D}$ может быть очень большой, важно понять также сложность ввода-вывода (I/O). Поскольку для вычисления поддержки каждого кандидата производится полное сканирование базы, I/O сложность BruteForce составляет $O\left(2^{|\mathcal{I}|}\right)$ сканирований базы данных. Таким образом, даже для небольших пространств наборов этот подход вычислительно трудный, тогда как на практике решетка $\mathcal{I}$ может быть очень большой (например, в супермаркете обычно тысячи товаров.

Теперь рассмотрим способы систематического улучшения brute-force алгоритма.

4.3.1 Поуровневый подход: алгоритм Apriori

Метод грубой силы перечисляет все возможные наборы объектов, что приводит к лишним вычислениям, т.к. многие кандидаты не могут не являться частыми. Пусть $X, Y \subseteq \mathcal{I}$ -

два любых набора. Заметим, что если $X \subseteq Y$, то $\sup(X) \geq \sup(Y)$, что приводит к следующим наблюдениям: (1) если X – частый образец, то любое подмножество $Y \subseteq X$ тоже часто, и (2) если X не часто, то любое надмножество $Y \supseteq X$ не может быть часто. Использование этих наблюдений в *алгоритме Apriori* позволяет добиться существенного улучшения по сравнению с методом грубой силы. В алгоритме применяется поуровневый (в ширину) поиск по пространству поиска наборов, удаление всех надмножеств нечастых кандидатов и обходится генерация кандидатов с нечастыми подмножествами. Также, алгоритм *Apriori* вместо вычисления поддержки одного набора объектов исследует префиксное дерево в ширину и вычисляет поддержку всех допустимых кандидатов размера k, которые составляют уровень k префиксного дерева. Это приводит к существенному улучшению в части вычислительно сложности и сложности ввода/вывода.

Алгоритм 4.2 показывает псевдокод алгоритма Apriori. Пусть $C^{(k)}$ обозначает префиксное дерево из всех кандидатов длины k. Начало метода – заполнение уровня $C^{(1)}$ единичными объектами (префиксное дерево изначально пусто). В цикле while (строчки 5-11) происходит вычисление поддержки для текущего набора кандидатов на уровне k через процедуру ComputeSupport, которая генерирует k-подмножества каждой транзакции в базе $\boldsymbol{D}$, и для каждого такого подмножества увеличивает поддержку соответствующего кандидата в $C^{(k)}$, если такой существует. Таким образом, база сканируется только один раз на каждом уровне, и вычисляется поддержка всех кандидатов длины k. Затем, происходит удаление нечастых кандидатов (строчка 9). Пережившие это листья префиксного дерева составляют множество частых наборов объектов длины k $\mathcal{F}^{(k)}$, которые используются, чтобы сгенерировать множество кандидатов длиной $k+1$ для следующего уровня (строчка 10). Процедура ExtendPrefixTree применяет расширение на основе префикса для генерации кандидатов. Для двух частых наборов длины k $\mathrm{X_a}$ и X_b с общим префиксом длиной $k-1$ (т.е. это два листа с общим родителем), создается кандидат длиной $(k+1)$ - $X_{ab} = X_a \cup X_b$. Кандидат принимается, если он не содержит нечастого подмножества. Наконец, если набор длины k X_a не имеет расширений, он рекурсивно удаляется из префиксного дерева вместе со своими предками, тоже не имеющими расширения. Если на уровне были добавлены новые кандидаты, процесс повторяется для следующего уровня.

В худшем случае, вычислительная сложность *Apriori* всё равно $O\left(|\mathcal{I}| \cdot |\boldsymbol{D}| \cdot 2^{|\mathcal{I}|}\right)$, т.к. все наборы объектов могут быть частыми. Однако, на практике вычислительная сложность алгоритма существенно ниже из-за очистки пространства поиска. Также, с точки зрения ввода-вывода, Apriori требует $O(|\mathcal{I}|)$ сканирований БД, в отличие от $O\left(2^{|\mathcal{I}|}\right)$ при методе грубой силы. Причем на практике требуется всего l сканирований базы, где l – самый длинный частый набор объектов.

Алгоритм 4.2. Apriori

```
APRIORI (D, 𝓘, minsup):
1  ℱ ← ∅
2  𝒞^(1) ← {∅} // Initial prefix tree with single items
3  foreach i ∈ 𝓘 do Add i as child of ∅ in 𝒞^(1) with sup(i) ← 0
4  k ← 1 // k denotes the level
5  while 𝒞^(k) ≠ ∅ do
6      COMPUTESUPPORT (𝒞^(k), D)
7      foreach leaf X ∈ 𝒞^(k) do
8          if sup(X) ≥ minsup then ℱ ← ℱ ∪ {(X, sup(X))}
9          else remove X from 𝒞^(k)
10     𝒞^(k+1) ← EXTENDPREFIXTREE (𝒞^(k))
11     k ← k + 1
12 return ℱ^(k)

COMPUTESUPPORT (𝒞^(k), D):
13 foreach ⟨t, i(t)⟩ ∈ D do
14     foreach k-subset X ⊆ i(t) do
15         if X ∈ 𝒞^(k) then sup(X) ← sup(X) + 1

EXTENDPREFIXTREE (𝒞^(k)):
16 foreach leaf X_a ∈ 𝒞^(k) do
17     foreach leaf X_b ∈ SIBLING(X_a), such that b > a do
18         X_ab ← X_a ∪ X_b
           // prune candidate if there are any infrequent subsets
19         if X_j ∈ 𝒞^(k), for all X_j ⊂ X_ab, such that |X_j| = |X_ab| − 1 then
20             Add X_ab as child of X_a with sup(X_ab) ← 0
21     if no extensions from X_a then
22         remove X_a, and all ancestors of X_a with no extensions, from 𝒞^(k)
23 return 𝒞^(k)
```

4.3.2 Подход пересечения наборов тидов: алгоритм Eclat

Шаг вычисления поддержки может быть существенно улучшен, если мы индексируем базу так, чтобы можно было производить быстрые вычисления частот. Заметим, что при поуровневом подходе, чтобы вычислить поддержку, мы генерируем подмножества каждой транзакции и проверяем, не найдется ли это подмножество в префиксном дереве. Это может быть слишком тяжело, т.к. так можно сгенерировать много подмножеств, которые не существуют в префиксном дереве.

Алгоритм Eclat настраивает наборы тидов непосредственно для вычисления поддержки. Основная идея – поддержка каждого набора-кандидата может быть вычислена пересечением наборов тидов для правильно выбранных подмножеств. В общем, если есть $\boldsymbol{t}(X)$ и $\boldsymbol{t}(Y)$ для двух любых частых наборов X и Y, имеем

$$\boldsymbol{t}(XY) = \boldsymbol{t}(X) \cap \boldsymbol{t}(Y)$$

Поддержка кандидата XY есть мощность $\boldsymbol{t}(XY)$, т.е. $\sup(XY) = |\boldsymbol{t}(XY)|$. Eclat пересекает наборы тидов, только если частые наборы объектов имеют общий префикс, и проходит префиксной дерево поиском в глубину, обрабатывая наборы объектов с общим префиксом, также называемый *префиксным классов эквивалентности*.

Псевдокод алгоритма Eclat представлен на алгоритме 4.3. Он применяет вертикальное представление бинарной базы $\boldsymbol{D}$. Таким образом, вход – набор кортежей $\langle i, \boldsymbol{t}(i)\rangle$ для всех частых объектов $\mathrm{i} \in \mathcal{I}$, который составляют класс эквивалентности P (у них у всех пустой префикс); предполагается, что P содержит только частые наборы объектов. В общем, имея префиксный класс эквивалентности P, для каждого частого набора объектов $X_a \in P$, мы пытаемся пересечь его набор тидов с набора тидов всех остальных наборов объектов $X_b \in P$. Образец-кандидат $X_{ab} = X_a \cup X_b$, и мы проверяем мощность пересечения $\boldsymbol{t}(X_a) \cap \boldsymbol{t}(X_b)$ чтобы определить, является ли образец частым. Если да, то X_{ab} добавляется к новому классу эквивалентности P_a, который содержит все наборы объектов, разделяющие X_a как префикс. Рекурсивный вызов Eclat затем находит все расширения ветки X_a в дереве поиска. Процесс продолжается, пока не останется возможных расширений для всех ветвей.

Алгоритм 4.3. Eclat

```
// Initial Call: F ← ∅, P ← {⟨i, t(i)⟩ | i ∈ I, |t(i)| ≥ minsup}
ECLAT (P, minsup, F):
1 foreach ⟨X_a, t(X_a)⟩ ∈ P do
2     F ← F ∪ {(X_a, sup(X_a))}
3     P_a ← ∅
4     foreach ⟨X_b, t(X_b)⟩ ∈ P, with X_b > X_a do
5         X_ab = X_a ∪ X_b
6         t(X_ab) = t(X_a) ∩ t(X_b)
7         if sup(X_ab) ≥ minsup then
8             P_a ← P_a ∪ {⟨X_ab, t(X_ab)⟩}
9     if P_a ≠ ∅ then ECLAT (P_a, minsup, F)
```

Вычисления сложность Eclat в худшем случае $O(|\boldsymbol{D}| \cdot 2^{|\mathcal{I}|})$, поскольку может быть $2^{|\mathcal{I}|}$ частых наборов объектов, и пересечение двух наборов тидов требует максимум $O(|\boldsymbol{D}|)$ времени. Сложность ввода-вывода Eclat тяжелее определить, т.к. она зависит от размера промежуточных наборов тидов. Если t – средний размер наборов, изначальный размер базы $O(t \cdot |\mathcal{L}|)$, и общий размер всех промежуточных наборов тидов $O\left(t \cdot 2^{|\mathcal{I}|}\right)$. Таким образом, Eclat требует $\frac{t \cdot 2^{|\mathcal{I}|}}{t \cdot |\mathcal{I}|} = O\left(\frac{2^{|\mathcal{I}|}}{|\mathcal{I}|}\right)$ сканирований БД в худшем случае.

Diffset-ы: разница между наборами тидов

Алгоритм Eclat может быть существенно улучшен, если можно сократить размер промежуточных наборов тидов. Это можно сделать, если отслеживать разницы в наборах тидов вместо полных наборов тидов. Формально, пусть $X_k = \{x_1, x_2, \ldots, x_{k-1}, x_k\}$ – набор объектов длины k. Определим diffset от X_k как множество тидов, содержащих префикс $X_{\{k-1\}} = \{x_1, x_2, \ldots, x_{k-1}\}$, но не содержащих объект x_k:

$$\boldsymbol{d}(X_k) = \boldsymbol{t}(X_{k-1}) \backslash \boldsymbol{t}(X_k)$$

Рассмотрим два набора объектов длины k: $X_a = \{x_1, \dots, x_{k-1}, x_a\}$ и $X_b = \{x_1, \dots, x_{k-1}, x_b\}$, разделяющие префикс $X = \{x_1, x_2, \dots, x_{k-1}\}$. $\boldsymbol{d}(X_{ab} = X_a \cup X_b = \{x_1, \dots, x_{k-1}, x_a, x_b\})$ определен как:

$$\boldsymbol{d}(X_{ab}) = \boldsymbol{t}(X_a) \backslash \boldsymbol{t}(X_{ab}) = \boldsymbol{t}(X_a) \backslash \boldsymbol{t}(X_b) \tag{4.3}$$

Однако, заметим что

$$\boldsymbol{t}(X_a) \backslash \boldsymbol{t}(X_b) = \boldsymbol{t}(X_a) \cap \overline{\boldsymbol{t}(X_b)}$$

и, поскольку $\boldsymbol{t}(X) \cap \overline{\boldsymbol{t}(X)} = \emptyset$, можно выразить $\mathbf{d}(X_{ab})$ через $\boldsymbol{d}(X_a)$ и $\boldsymbol{d}(X_b)$ следующим образом:

$$\begin{aligned}
\boldsymbol{d}(X_{ab}) &= \boldsymbol{t}(X_a) \backslash \boldsymbol{t}(X_b) \\
&= \boldsymbol{t}(X_a) \cap \overline{\boldsymbol{t}(X_b)} \\
&= \left(\boldsymbol{t}(X_a) \cap \overline{\boldsymbol{t}(X_b)}\right) \cup \boldsymbol{t}(X) \cap \overline{\boldsymbol{t}(X)} \\
&= \left(\left(\boldsymbol{t}(X_a) \cup \boldsymbol{t}(X)\right) \cap \left(\overline{\boldsymbol{t}(X_b)} \cup \overline{\boldsymbol{t}(X)}\right)\right) \cap \left(\boldsymbol{t}(X_a) \cup \overline{\boldsymbol{t}(X)}\right) \cap \left(\overline{\boldsymbol{t}(X_b)} \cup \boldsymbol{t}(X)\right) \\
&= \left(\boldsymbol{t}(X) \cap \overline{\boldsymbol{t}(X_b)}\right) \cap \overline{\left(\boldsymbol{t}(X) \cap \overline{\boldsymbol{t}(X_a)}\right)} \cap \mathcal{T} \\
&= \boldsymbol{d}(X_b) \backslash \boldsymbol{d}(X_a)
\end{aligned}$$

Таким образом, $\mathbf{d}(X_{ab})$ может быть получен через diffset-ы подмножеств X_aи X_b, что означает, что все операции пересечения в алгоритме Eclat можно заменить на операции с diffset-ами. Тогда поддержка набора объектов-кандидата может получена:

$$\sup(X_{ab}) = \sup(X_a) - |\boldsymbol{d}(X_{ab})|$$

что следует из (4.3).

Вариант алгоритма Eclat, использующий данную оптимизацию, называется dEclat. Псевдокод алгоритма представлен в алгоритме 4.4. Вход алгоритма – все частые единичные объекты $i \in \mathcal{I}$ вместе с их $\boldsymbol{d}(i)$, которые вычисляются как

$$\boldsymbol{d}(i) = \boldsymbol{t}(\emptyset) \backslash \boldsymbol{t}(i) = \mathcal{T} \backslash \boldsymbol{t}(i)$$

Алгоритм 4.4. dEclat

```
// Initial Call: F ← ∅,
    P ← {⟨i, d(i), sup(i)⟩ | i ∈ I, d(i) = T \ t(i), sup(i) ≥ minsup}
dECLAT (P, minsup, F):
1  foreach ⟨X_a, d(X_a), sup(X_a)⟩ ∈ P do
2      F ← F ∪ {(X_a, sup(X_a))}
3      P_a ← ∅
4      foreach ⟨X_b, d(X_b), sup(X_b)⟩ ∈ P, with X_b > X_a do
5          X_ab = X_a ∪ X_b
6          d(X_ab) = d(X_b) \ d(X_a)
7          sup(X_ab) = sup(X_a) − |d(X_ab)|
8          if sup(X_ab) ≥ minsup then
9              P_a ← P_a ∪ {⟨X_ab, d(X_ab), sup(X_ab)⟩}
10     if P_a ≠ ∅ then dECLAT (P_a, minsup, F)
```

Имея класс эквивалентности P, для каждой отличной пары наборов объектов X_a и X_b мы генерируем образец-кандидат $X_{ab} = X_a \cup X_b$ и проверяем, является ли он частым с использованием diffset-ов (строчки 6-7). Чтобы найти дальнейшие расширения, использованы рекурсивные вызовы. Важно отметить, что переключение с наборов тидов на diffset-ы может быть сделано на любом рекурсивном вызове метода. В частности, если начальные наборы тидов имеют малую мощность, первый вызов может использовать пересечения наборов тидов, и переключиться на diffset-ы начиная с длины наборов объектов 2. Для ясности, эта оптимизация не приведена в псевдокоде.

4.3.3 Подход дерева частоты образцов: алгоритмов FPGrowth

Алгоритм FPGrowth индексирует БД для быстрого вычисления поддержки с использованием расширенного префиксного дерева, называемого *дерево частых образцов* (frequent pattern tree, FP-дерево). Каждый узел дерева хранит один объект и информацию о наборе объектов, составляющем объекты по пути от корня до этого узла. FP-дерево составляется следующим образом. Изначально, дерево содержит пустой элемент $\emptyset$ как корень. Затем, для каждого кортежа $\langle t, X \rangle \in \boldsymbol{D}$, где $X = \boldsymbol{i}(t)$, мы вставляем наборов объектов X в FP-дерево, увеличивая число на узлах по пути, представляющем X. Если X разделяет префикс с какой-либо ранее вставленной транзакцией, то X будет следовать по тому же пути на протяжении всего общего префикса. Для оставшихся объектов будут созданы новые узлы с числами, установленными в 1. FP-дерево завершено, когда все транзакции вставлены.

FP-дерево можно рассматривать как представление $\boldsymbol{D}$, сжатое по префиксу. Поскольку мы хотим как можно более компактное дерево, наиболее частые элементы надо расположить сверху. Поэтому FPGrowth упорядочивает элементы в порядке убывания поддержки. Т.е. сначала вычисляется поддержка единичных объектов $i \in \mathcal{T}$, затем отбрасываются нечастые элементы, и все частые элементы сортируются по убыванию поддержки. В итоге, каждый кортеж $\langle t, X \rangle \in \boldsymbol{D}$ вставляется в FP-дерево после перестановки X в порядке убывания поддержки объектов.

Как только FP-дерево сконструировано, оно служит как индекс вместо исходной базы данных. Все частые наборы могут быть получены непосредственно из дерева с помощью метода FPGrowth, псевдокод которого представлен на алгоритме 4.5. Метод принимает FP-

дерево R, сконструированное на основе БД ***D,*** и текущий префикс набора объектов P, который изначально пустой.

С помощью FP-дерева R, для каждого частого объекта i в R строятся проецируемые FP-деревья в порядке возрастания поддержки. Чтобы спроецировать R на объект i, надо найти все вхождения i в дереве, и для каждого вхождения определить путь от корня до i (строчка 13). Число в элементе i на данном пути записано в $cnt(i)$ (строчка 14), и путь вставляется в проецируемое дерево R_X, где X – набор объектов, полученный расширением префикса P объектом i. При вставке пути, числа на каждом узле в R_x по данному пути увеличиваются на $cnt(i)$, а сам объект i опускается из пути. Полученное FP-дерево – проекция набора объектов X, которая составляет текущий префикс, расширенной объектом i (строчка 9). Затем происходит рекурсивный вызов FPGrowth от R_x и нового X (строчка 16). Когда дерево R – единичный путь, такое дерево обрабатывается перечислением всех наборов объектов, являющихся подмножествами пути, и рекурсивный вызов не происходит.

Алгоритм 4.5. Алгоритм FPGrowth

```
// Initial Call: R ← FP-tree(D), P ← ∅, F ← ∅
FPGROWTH (R, P, F, minsup):
1  Remove infrequent items from R
2  if ISPATH(R) then // insert subsets of R into F
3      foreach Y ⊆ R do
4          X ← P ∪ Y
5          sup(X) ← min_{x∈Y}{cnt(x)}
6          F ← F ∪ {(X, sup(X))}
7  else   // process projected FP-trees for each frequent item i
8      foreach i ∈ R in increasing order of sup(i) do
9          X ← P ∪ {i}
10         sup(X) ← sup(i) // sum of cnt(i) for all nodes labeled i
11         F ← F ∪ {(X, sup(X))}
12         R_X ← ∅ // projected FP-tree for X
13         foreach path ∈ PATHFROMROOT(i) do
14             cnt(i) ← count of i in path
15             Insert path, excluding i, into FP-tree R_X with count cnt(i)
16         if R_X ≠ ∅ then FPGROWTH (R_X, X, F, minsup)
```

4.3.4 Генерация правил ассоциации

Имея коллекцию частых наборов объектов $\mathcal{F}$, для генерации правил ассоциации можно провести итерацию по всех наборам объектов $Z \in \mathcal{F}$, и вычислить уверенность для различных правил, которые могут быть потом выведены из набора объектов. Формально, имея частый набор объектов $Z \in \mathcal{F}$, мы смотрим на все строгие подмножества $X \subset Z$, чтобы вычислить правила вида

$$X \xrightarrow{s,c} Y, \text{ где } Y = Z \backslash X$$

где $Z \backslash X = Z - X$. Правило должно быть частым, поскольку

$$s = \sup(XY) = \sup(Z) \geq minsup$$

Таким образом, достаточно проверить, удовлетворяет ли уверенность правила пределу $minconf$. Уверенность вычисляется следующим образом:

$$c = \frac{\sup(X \cup Y)}{\sup(X)} = \frac{\sup(Z)}{\sup(X)}$$

если $c \geq minconf$, то правило – сильное. С другой стороны, если $conf(X \rightarrow Y) < c$, то $conf(W \rightarrow Z \backslash W) < C$ для всех подмножеств $W \subset X$, где $\sup(W) \geq \sup(X)$. Таким образом, можно избежать проверки подмножеств X.

Алгоритм 4.3 показывает псевдокод для алгоритма майнинга правила ассоциаций. Для каждого частого набора объектов $Z \in \mathcal{F}$, с размером хотя бы 2, мы инициализируем множество антецедентов $\mathcal{A}$ всеми непустыми подмножествами Z (строчка 2). Для каждой $X \in \mathcal{A}$ мы проверяем, является уверенность правила $X \rightarrow Z \backslash X$ хотя бы $minconf$ (строчка 7). Если да, то правило сохраняется. Иначе удаляем все подмножества $W \subset X$ из множество возможных антецедентов (строчка 10)

Алгоритм 4.6. Алгоритм AssociationRules

```
ASSOCIATIONRULES (F, minconf):
1  foreach Z ∈ F, such that |Z| ≥ 2 do
2      A ← {X | X ⊂ Z, X ≠ ∅}
3      while A ≠ ∅ do
4          X ← maximal element in A
5          A ← A \ X// remove X from A
6          c ← sup(Z)/sup(X)
7          if c ≥ minconf then
8              print X ⟶ Y, sup(Z), c
9          else
10             A ← A \ {W | W ⊂ X} // remove all subsets of X from A
```

Глава 5. Суммирование наборов

Пространство поиска для часто встречающихся наборов элементов обычно очень велико и растет экспоненциально с увеличением количества элементов. В частности, низкое минимальное значение поддержки может привести к непреодолимому количеству частых наборов элементов. Альтернативный подход, изучаемый в этой главе, состоит в том, чтобы определить сжатые представления часто встречающихся наборов элементов, которые суммируют их основные характеристики. Использование сжатых представлений может не только снизить требования к вычислениям и хранению, но также может упростить анализ добытых паттернов. В этой главе мы обсудим три из этих представлений: замкнутые, максимальные и невыводимые наборы элементов.

5.1 Максимальная и замкнутая частота наборов

Для двоичной базы данных $\mathbf{D} \subseteq \mathcal{T} \times \mathcal{I}$, над тидами $\mathcal{T}$ и элементами $\mathcal{I}$, пусть $\mathcal{F}$ обозначает набор всех частых наборов элементов, то есть

$$\mathcal{F} = \{X | X \subseteq \mathcal{I} \text{ и } \sup(X) \geq \text{minsup}\}$$

Максимально частые наборы элементов

Частое множество $X \subseteq \mathcal{F}$ называется максимальным, если оно не имеет частых надмножеств. Пусть $\mathcal{M}$ будет набором всех максимально частых наборов элементов, заданных как

$$\mathcal{M} = \{X | X \in \mathcal{F} \text{ и } \nexists Y \supset X, \text{такого что } Y \in \mathcal{F}\}$$

Множество $\mathcal{M}$ является сжатым представлением набора всех часто встречающихся элементов $\mathcal{F}$, потому что мы можем определить, является ли какой-либо набор элементов X частым или нет, используя $\mathcal{M}$. Если существует максимальный набор элементов Z такой, что $X \subseteq Z$, то X должен быть частым; в противном случае X не может быть частым. С другой стороны, мы не можем определить $\sup(X)$, используя только $\mathcal{M}$, хотя мы можем оценить его снизу, то есть $\sup(X) \geq \sup(Z)$, если $X \subseteq Z \in \mathcal{M}$.

Пример 5.1

Рассмотрим набор данных, представленный на рисунке 5.1a. Используя любой из алгоритмов, обсуждаемых в главе 8, и $minsup = 3$, мы получаем часто встречающиеся наборы элементов, показанные на рис. 5.1b. Обратите внимание, что существует 19 часто встречающихся наборов элементов из $2^5 - 1 = 31$ возможных непустых наборов элементов. Из них есть только два максимальных набора элементов,

Tid	Itemset
1	$ABDE$
2	BCE
3	$ABDE$
4	$ABCE$
5	$ABCDE$
6	BCD

Рисунок 5.1(a) – База данных транзакций

sup	Itemsets
6	B
5	E, BE
4	$A, C, D, AB, AE, BC, BD, ABE$
3	$AD, CE, DE, ABD, ADE, BCE, BDE, ABDE$

Рисунок 5.1(b) – Частые наборы

$ABDE$ и BCE. Любой другой часто встречающийся набор элементов должен быть подмножеством одного из максимальных наборов элементов. Например, мы можем определить, что ABE является частым, поскольку $ABE \subset ABDE$, и мы можем установить, что $\sup(\text{ABE}) \geq \sup(ABDE) = 3$.

Замкнутые часто встречающиеся наборы

Напомним, что функция $\mathbf{t}: 2^{\mathcal{I}} \longrightarrow 2^{\mathcal{T}}$ отображает наборы элементов в наборы тидов, и функция $\mathbf{i}: 2^{\mathcal{T}} \longrightarrow 2^{\mathcal{I}}$ отображает наборы тидов в наборы элементов. То есть, для $T \subseteq \mathcal{T}$ и $X \subseteq \mathcal{I}$, получаем

$$\mathbf{t}(X) = \{t \in \mathcal{T} \mid t \text{ содержит } X\}$$

$$\mathbf{i}(T) = \{x \in \mathcal{I} \mid \forall t \in T, t \text{ содержит } x\}$$

Определим $\mathbf{c}: 2^{\mathcal{I}} \longrightarrow 2^{\mathcal{T}}$ – оператор замыкания, заданный как

$$\mathbf{c}(X) = \mathbf{i} \circ \mathbf{t}(\text{X}) = \mathbf{i}(\mathbf{t}(X))$$

Оператор замыкания $\mathbf{c}$ отображает наборы элементов в наборы элементов и удовлетворяет следующим трем свойствам:

- *Экстенсивность:* $X \subseteq \mathbf{c}(X)$
- *Монотонность:* Если $X_i \subseteq X_j$, то $c(X_i) \subseteq c(X_j)$
- *Идемпотентность:* $\mathbf{c}\big(\mathbf{c}(X)\big) = \mathbf{c}(X)$

Набор элементов X называется замкнутым, если $\mathbf{c}(X) = X$, то есть если X является неподвижной точкой оператора замыкания $\mathbf{c}$. С другой стороны, если $X \neq \mathbf{c}(X)$, то X не замкнут, но множество $\mathbf{c}(X)$ называется его замыканием. Судя по свойствам оператора замыкания, и X, и $\mathbf{c}(X)$ имеют одинаковый набор тидов. Отсюда следует, что частое множество $X \subseteq \mathcal{F}$ является замкнутым, если оно не имеет частого надмножества с той же частотой, потому что по определению это самый большой набор элементов, общий для всех тидов в наборе $\mathbf{t}(X)$. Таким образом, набор всех замкнутых частых наборов элементов определяется как

$$\mathcal{C} = \{X | X \in \mathcal{F} \text{ и } \nexists Y \supset X, \text{такого, что } \sup(X) = \sup(Y)\} \quad (5.1)$$

Иными словами, X замкнуто, если все надмножества X имеют строго меньшую поддержку, то есть $\sup(\mathrm{X}) \geq \sup(Y)$ для всех $Y \supset X$.

Набор всех замкнутых часто встречающихся наборов элементов $\mathcal{C}$ представляет собой сжатое представление, поскольку мы можем определить, является ли набор элементов X частым, а также точную поддержку X, используя только $\mathcal{C}$. Набор X является частым, если существует замкнутый частый набор элементов $Z \in \mathcal{C}$, такой что $X \subseteq Z$. Далее, поддержка X задается как

$$\sup(X) = \max\{\sup(Z) | Z \in \mathcal{C}, X \subseteq Z\}$$

Следующие отношения поддерживаются между набором всех, замкнутых и максимально частых наборов элементов:

$$\mathcal{M} \subseteq \mathcal{C} \subseteq \mathcal{F}$$

Минимальные генераторы

Частый набор элементов X является минимальным генератором, если у него нет подмножеств с такой же поддержкой:

$$\mathcal{G} = \{X | X \in \mathcal{F} \text{и } \nexists Y \subset X, \text{такого, что } \sup(X) = \sup(Y)\}$$

Другими словами, все подмножества X имеют строго более высокую поддержку, то есть$\sup(\mathrm{X}) < \sup(Y)$ для всех $Y \subset X$. Понятие минимального генератора тесно связано с понятием замкнутых наборов элементов. Учитывая класс эквивалентности наборов элементов, которые имеют один и тот же набор тидов, замкнутый набор элементов является единственным максимальным элементом класса, а минимальные генераторы – минимальными элементами класса.

Пример 5.2

Рассмотрим пример набора данных на рисунке 5.1a. Частые замкнутые (а также максимальные) наборы элементов данных с $\mathrm{minsup} = 3$ показаны на рисунке 5.2. Мы можем видеть, например, что наборы элементов AD, DE, ABD, ADE, BDE и $ABDE$ встречаются в одних и тех же трех транзакциях, а именно 135, и, таким образом, составляют класс эквивалентности. Самый большой набор элементов среди них, а именно $ABDE$, является замкнутым набором элементов. Использование оператора замыкания дает тот же результат; имеем $\mathbf{c}(AD) = \mathbf{i}\big(\mathbf{t}(AD)\big) = \mathbf{i}(135) = ABDE$, что указывает на то, что замыкание AD является $ABDE$. Чтобы проверить, что $ABDE$ замкнут, заметьте, что $\mathbf{c}(ABDE) = \mathbf{i}\big(\mathbf{t}(ABDE)\big) = \mathbf{i}(135) = ABDE$. Минимальные элементы класса эквивалентности, а именно AD и DE,

являются минимальными генераторами. Ни одно из подмножеств этих наборов элементов не имеет одинаковых наборов элементов.

Набор всех замкнутых частых наборов элементов и соответствующий набор минимальных генераторов:

Tidset	$\mathcal{C}$	$\mathcal{G}$
1345	ABE	A
123456	B	B
1356	BD	D
12345	BE	E
2456	BC	C
135	$ABDE$	AD, DE
245	BCE	CE

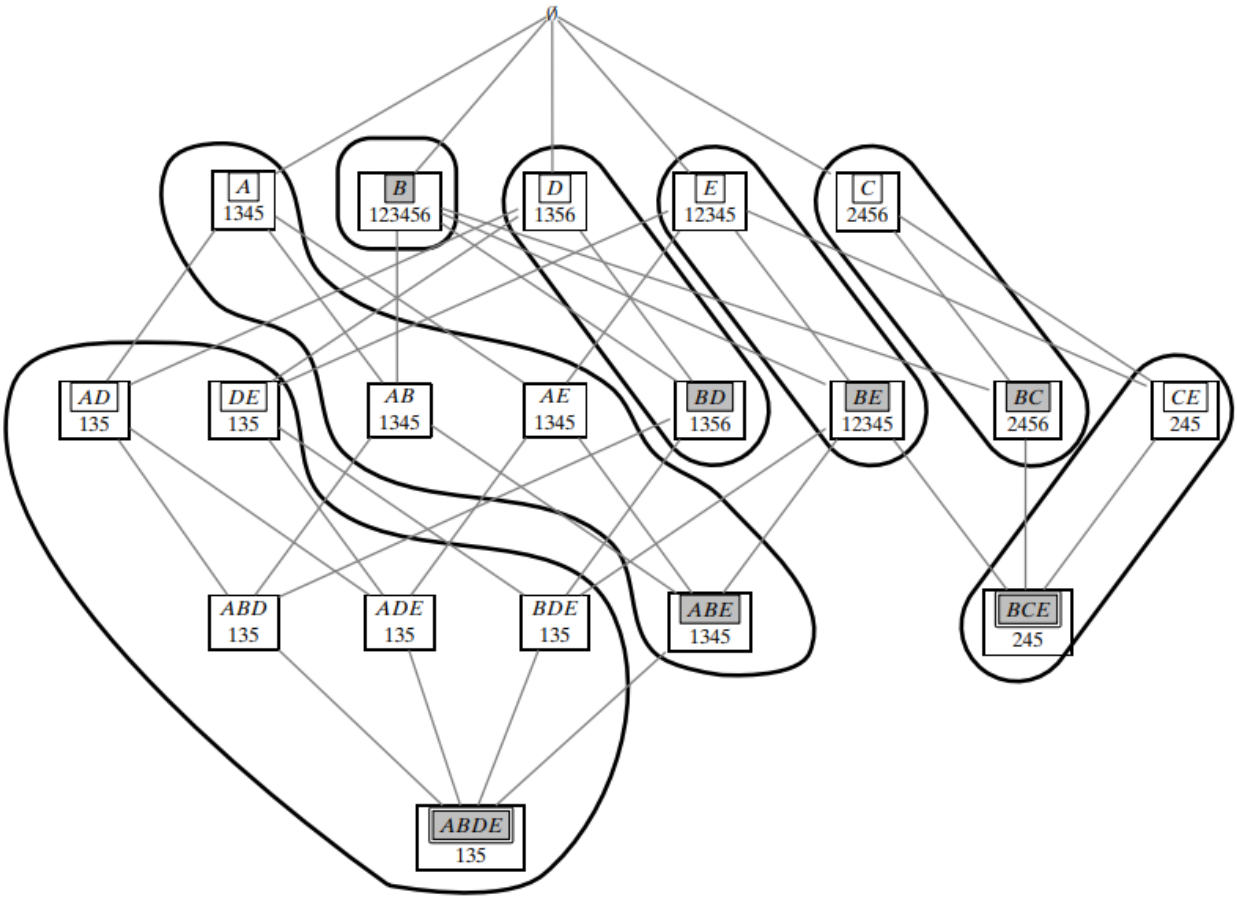


Рисунок 5.2 – Частые, замкнутые, минимальные генераторы и максимально частые наборы элементов. Наборы элементов, заключенные в рамки и закрашенные, замкнуты, тогда как наборы внутри рамок (но не закрашенные) являются минимальными генераторами; максимальные наборы элементов показаны двойными линиями.

Из замкнутых наборов элементов максимальными являются $ABDE$ и BCE. Рассмотрим набор элементов AB. Используя $\mathcal{C}$, мы можем определить, что

$$\sup(AB) = \max\{\sup(ABE) \mid \sup(ABDE)\} = \max\{4,3\} = 4$$

5.2 Поиск максимальной частоты набора: алгоритм GENMAX

Поиск максимальных наборов элементов требует дополнительных шагов, помимо простого определения частых наборов элементов. Предполагая, что набор максимально частых наборов элементов изначально пуст, то есть $\mathcal{M} = \emptyset$, каждый раз, когда мы генерируем новый набор часто встречающихся элементов X, мы должны выполнять следующие проверки максимальности

- **Проверка подмножества:** $\nexists Y \in \mathcal{M}$, такого что $X \subset Y$. Если такой Y существует, очевидно что X не максимальный. В противном случае добавляем X в $\mathcal{M}$, как потенциально максимальный набор элементов.
- **Проверка надмножества:** $\nexists Y \in \mathcal{M}$, такого что $Y \subset X$. Если такой Y существует, то Y не может быть максимальным, и мы должны исключить его из $\mathcal{M}$.

Эти две проверки на максимальность занимают время $O(|\mathcal{M}|)$, что может стать дорогостоящим, особенно по мере роста $\mathcal{M}$; таким образом, по соображениям эффективности очень важно минимизировать количество повторений этих проверок. Таким образом, любой из часто используемых алгоритмов интеллектуального анализа наборов элементов из главы 8 может быть расширен для поиска максимально частых наборов элементов путем добавления шагов проверки максимальности. Здесь мы рассматриваем метод GenMax, который основан на подходе пересечений наборов тидов Eclat. Мы увидим, что он никогда не вставляет не максимальный набор элементов в $\mathcal{M}$. Таким образом, он исключает проверки надмножества и требует только проверки подмножества для определения максимальности. Алгоритм 5.1 показывает псевдокод для GenMax. Первоначальный вызов принимает в качестве входных данных набор часто встречающихся элементов вместе с их наборами тидов, $\langle i, \mathbf{t}(i)\rangle$, и изначально пустой набор максимальных наборов элементов, $\mathcal{M}$. Для данного набора пар наборов элементов и наборов тидов, называемых IT-парами, вида $\langle X, \mathbf{t}(X)\rangle$, рекурсивный метод GenMax работает следующим образом. В строках 1–3 мы проверяем, можно ли обрезать всю текущую ветвь, проверяя, входит ли объединение всех наборов элементов, $Y = \bigcup X_i$, в некоторый максимальный паттерн $Z \in \mathcal{M}$(или содержится в нем). Если это так, то из текущей ветви невозможно сгенерировать максимальный набор элементов, и он удаляется. С другой стороны, если ветвь не отсечена, мы перекрещиваем каждую IT-пару $\langle X_i, \mathbf{t}(X_i)\rangle$ со всеми другими IT-парами $\langle X_j, \mathbf{t}(X_j)\rangle$, при $j > i$, для генерации новых кандидатов X_{ij} , которые добавляются к множеству IT-пар P_i (строки 6–9). Если P_i не пуст, выполняется рекурсивный вызов GenMax для поиска других потенциально частых расширений X_i. С другой стороны, если P_i пусто, это означает, что X_i не может быть расширен, и он потенциально максимален. В этом случае мы добавляем X_i к множеству $\mathcal{M}$ при условии, что X_i не содержится ни в одном ранее добавленном максимальном множестве $Z \in \mathcal{M}$(строка 12). Также обратите внимание, что из-за этой проверки максимальности перед вставкой любого набора элементов в $\mathcal{M}$ нам никогда не нужно удалять из него какие-либо наборы элементов. Другими словами, все наборы элементов в $\mathcal{M}$ гарантированно максимальны. По завершении GenMax набор $\mathcal{M}$ содержит последний набор всех максимально частых наборов элементов. Подход GenMax также включает ряд других оптимизаций для уменьшения проверок максимальности и улучшения вычислений поддержки. Кроме того, GenMax использует набор различий наборов тидов для

быстрого вычисления поддержки, которые были описаны в Разделе 4.2.2. Мы опускаем эти оптимизации здесь для ясности.

Алгоритм 5.1: GenMax

```
// Initial Call: M ← ∅, P ← {⟨i, t(i)⟩ | i ∈ I, sup(i) ≥ minsup}
GENMAX (P, minsup, M):
1  Y ← ⋃ X_i
2  if ∃Z ∈ M, such that Y ⊆ Z then
3      return // prune entire branch
4  foreach ⟨X_i, t(X_i)⟩ ∈ P do
5      P_i ← ∅
6      foreach ⟨X_j, t(X_j)⟩ ∈ P, with j > i do
7          X_ij ← X_i ∪ X_j
8          t(X_ij) = t(X_i) ∩ t(X_j)
9          if sup(X_ij) ≥ minsup then P_i ← P_i ∪ {⟨X_ij, t(X_ij)⟩}
10     if P_i ≠ ∅ then GENMAX (P_i, minsup, M)
11     else if ∄Z ∈ M, X_i ⊆ Z then
12         M = M ∪ X_i    // add X_i to maximal set
```

Пример 5.3

На рисунке 5.3 показано выполнение GenMaxв примере базы данных с рисунка 5.1a с использованием $\text{minsup} = 3$. Первоначально набор максимальных наборов элементов пуст. Корень дерева представляет собой начальный вызов со всеми IT-парами, состоящими из частых отдельных элементов и их наборов тидов. Сначала мы перекрещиваем $\mathbf{t}(A)$ с наборами элементов других элементов. Набор частых расширений от A

$$P_A = \{\langle AB, 1345\rangle, \langle AD, 135\rangle, \langle AE, 1345\rangle\}$$

Выбор $X_i = AB$ приводит к следующему набору расширений, а именно

$$P_{AB} = \{\langle ABD, 135\rangle, \langle ABE, 1345\rangle\}$$

Наконец, мы достигаем самого левого листа, соответствующего $P_{ABD} = \{\langle ABDE, 135\rangle\}$. На этом этапе мы добавляем $ABDE$ к набору максимально частых наборов элементов, потому что у него нет других расширений, так что $\mathcal{M} = \{ABDE\}$.

Затем поиск возвращается на один уровень, и мы пытаемся обработать ABE, который также является кандидатом на максимальное значение. Однако он содержится в $ABDE$, поэтому его удаляют. Точно так же, когда мы пытаемся обработать $P_{AD} = \{\langle ADE, 135\rangle\}$, он будет отсечен, потому что он также включен в $ABDE$, и аналогично для AE. На этом этапе все максимальные наборы элементов, начинающиеся с A, найдены, и мы переходим к ветви B. Крайняя левая ветвь B, а именно BCE, не может быть расширена дальше. Поскольку BCE не является подмножеством какого-либо максимального набора элементов в $\mathcal{M}$, мы вставляем его как максимальный набор элементов, так что $\mathcal{M} = \{ABDE, BCE\}$. Впоследствии все оставшиеся ветви включаются в один из этих двух максимальных наборов элементов и, таким образом, отсекаются.

5.3 Поиск закрытых часто встречающихся наборов элементов: CHARM алгоритм.

Поиск часто встречающихся закрытых наборов элементов требует, чтобы мы выполняли проверку закрытия, то есть является ли $X = \boldsymbol{c}(X)$. Прямая проверка закрытия может быть очень дорогой операцией, так как мы должны были бы проверить, что X является самым большим набором элементов, общим для всех тидов в $\boldsymbol{t}(X)$, то есть, $X = \bigcap_{t \in \boldsymbol{t}(X)} \boldsymbol{i}(t)$. Вместо этого мы опишем метод пересечения вертикальных наборов тидов, называемый CHARM, который выполняет более эффективную проверку закрытия. Учитывая совокупность IT-пар $\{\langle X_i , \boldsymbol{t}(X_i)\rangle\}$, выполняются следующие 3 свойства:

Свойство (1) Если $\boldsymbol{t}(X_i) = \boldsymbol{t}(X_j)$, тогда $\boldsymbol{c}(X_i) = \boldsymbol{c}(X_j) = \boldsymbol{c}(X_i \cup X_j)$, что означает, что мы можем заменить каждое вхождение X_i на $X_i \cup X_j$ и отбросить ветвь под X_j, потому что ее закрытие идентично закрытию $X_i \cup X_j$.

Свойство (2) Если $\boldsymbol{t}(X_i) \subset \boldsymbol{t}(X_j)$, тогда $\boldsymbol{c}(X_i) \neq \boldsymbol{c}(X_j)$, но $\boldsymbol{c}(X_i) = \boldsymbol{c}(X_i \cup X_j)$, что означает, что мы можем заменить каждое вхождение X_i на $X_i \cup X_j$, но не можем отбросить X_j, потому что это порождает другое закрытие. Заметим, что если $\boldsymbol{t}(X_i) \supset \boldsymbol{t}(X_j)$, тогда мы просто меняем роль X_i и X_j.

Свойство (3) Если $\boldsymbol{t}(X_i) \neq \boldsymbol{t}(X_j)$, то $\boldsymbol{c}(X_i) \neq \boldsymbol{c}(X_j) \neq \boldsymbol{c}(X_i \cup X_j)$. В этом случае мы не можем удалить ни X_i ни X_j, так как каждый из них создает свое закрытие.

Алгоритм 5.2 представляет псевдокод для Charm, который также основан на алгоритме Eclat. Он принимает в качестве входных набор данных всех часто встречающихся одиночных элементов вместе с их наборами тидов. Кроме того, изначально множество всех замкнутых наборов элементов, $\mathcal{C}$, пусто. Учитывая любое множество IT-пар $P = \{\langle X_i , \boldsymbol{t}(X_i)\rangle\}$, метод сначала сортирует их в порядке возрастания поддержки. Для каждого набора элементов X_i мы пытаемся расширить его со всеми другими элементами X_j в отсортированном порядке и применяем вышеупомянутые три свойства для отбрасывания ветвей, где это возможно. Сначала мы убедимся, что $X_{ij} = X_i \cup X_j$ является часто встречающимся, проверив мощность $\boldsymbol{t}(X_{ij})$. Если да, то мы проверяем свойства 1 и 2 (строки 8 и 12). Обратите внимание, что каждый раз, когда мы заменяем X_i на $X_{ij} = X_i \cup X_j$, мы обязательно делаем это в текущем наборе P, а также в новом наборе P_i. Только когда свойство 3 выполняется, мы добавляем новое расширение X_{ij} к набору P_i (строка 14). Если множество P_i не пусто, то мы делаем рекурсивный вызов Charm. Наконец, если X_i не является подмножеством любого замкнутого множества Z с той же поддержкой, мы можем смело добавить его к множеству замкнутых наборов элементов $\mathcal{C}$ (строка 18). Для быстрого вычисления поддержки Charm использует оптимизацию diffset; здесь она опускается.

Алгоритм 5.2: Charm

```
// Initial Call: C ← ∅,  P ← {⟨i, t(i)⟩ : i ∈ I, sup(i) ≥ minsup}
CHARM (P, minsup, C):
1  Sort P in increasing order of support (i.e., by increasing |t(X_i)|)
2  foreach ⟨X_i, t(X_i)⟩ ∈ P do
3      P_i ← ∅
4      foreach ⟨X_j, t(X_j)⟩ ∈ P, with j > i do
5          X_ij = X_i ∪ X_j
6          t(X_ij) = t(X_i) ∩ t(X_j)
7          if sup(X_ij) ≥ minsup then
8              if t(X_i) = t(X_j) then // Property 1
9                  Replace X_i with X_ij in P and P_i
10                 Remove ⟨X_j, t(X_j)⟩ from P
11             else
12                 if t(X_i) ⊂ t(X_j) then // Property 2
13                     Replace X_i with X_ij in P and P_i
14                 else // Property 3
15                     P_i ← P_i ∪ {⟨X_ij, t(X_ij)⟩}
16     if P_i ≠ ∅ then CHARM (P_i, minsup, C)
17     if ∄Z ∈ C, such that X_i ⊆ Z and t(X_i) = t(Z) then
18         C = C ∪ X_i    // Add X_i to closed set
```

Пример 5.4.

Мы проиллюстрируем алгоритм Charm для нахождения часто встречающихся закрытых наборов элементов из примера базы данных на рис. 5.1a, используя минимальный уровень поддержки (minsup = 3). На рисунке 5.4 показана последовательность шагов. Начальный набор IT-пар после сортировки на основе поддержки отображается в корне дерева поиска. Порядок сортировки – A, C, D, E и B. Сначала мы обрабатываем расширения из A, как показано на рис. 5.4 a. Так как AC не является часто встречающимся, его отбрасывают. AD является часто встречающимся, потому что $\boldsymbol{t}(A) \neq \boldsymbol{t}(D)$, мы добавляем $\langle AD, 135\rangle$ в множество P_A (свойство 3). Когда мы объединяем A с E, применяется свойство 2, и мы просто заменяем все вхождения A как в P, так и в P_A на AE. Аналогично, поскольку $\boldsymbol{t}(A) \subset \boldsymbol{t}(B)$, все текущие вхождения A, фактически AE, как в P, так и в P_A заменяются AEB. Таким образом, множество P_A содержит только один набор элементов $\{\langle ADEB, 135\rangle\}$. Когда CHARM вызывается с P_A в качестве IT-пары, он переходит прямо к строке 18 и добавляет $ADEB$ к набору закрытых наборов элементов $\mathcal{C}$. Когда вызов возвращается, мы проверяем, можно ли добавить AEB как закрытый набор элементов. AEB является подмножеством $ADEB$, но у него нет такой же поддержки, поэтому AEB также добавляется в $\mathcal{C}$. На этом этапе все закрытые наборы элементов, содержащие A, были найдены.

Алгоритм Charm переходит к оставшимся ветвям, как показано на рисунке 9.4b. Например, C обрабатывается следующим. CD нечасто встречается и поэтому сокращается. CE встречается часто и добавляется к P_C как новое расширение (через свойство 3). Поскольку $\boldsymbol{t}(C) \subset \boldsymbol{t}(B)$, все вхождения C заменяются на CB, и $P_C = \{\langle CEB, 235\rangle\}$. CEB и CB закрыты.

Вычисление продолжается таким образом, пока не будут перечислены все закрытые частые наборы элементов. Обратите внимание: когда мы переходим к DEB и выполняем проверку закрытия, мы обнаруживаем, что это подмножество $ADEB$ также имеет ту же поддержку; при этом DEB не закрывается.

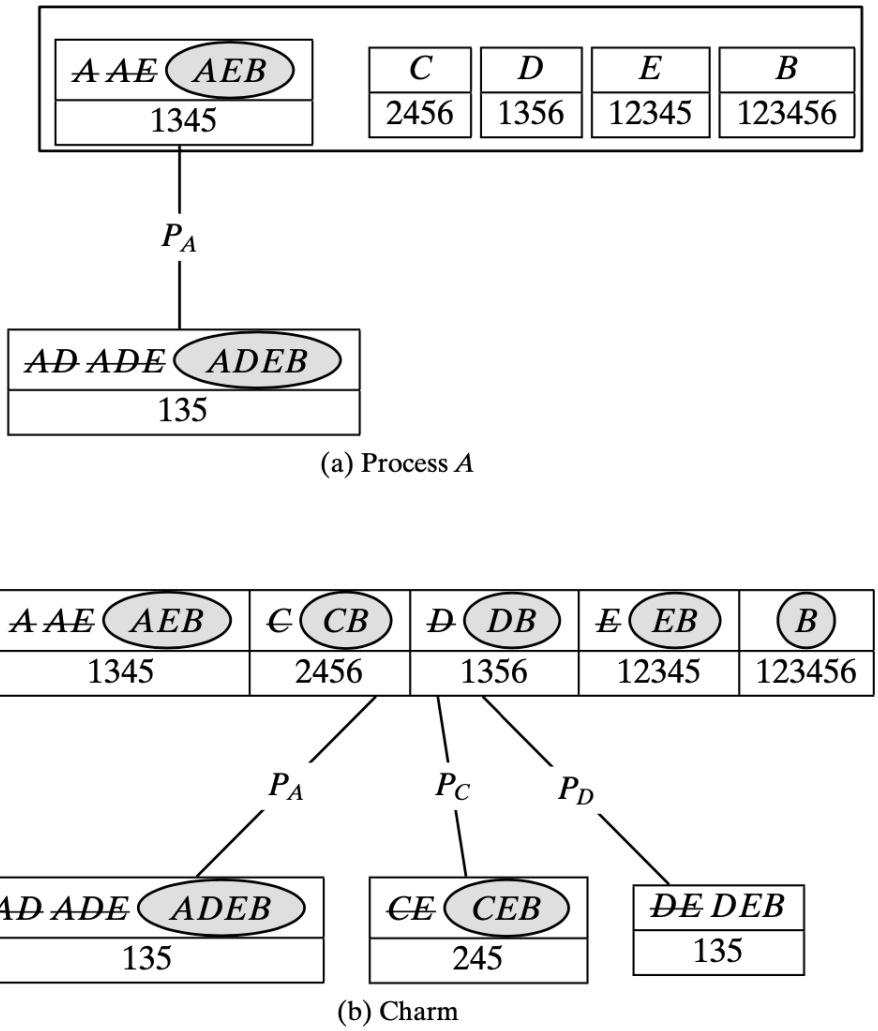


Рисунок 5.4 Поиск часто встречающихся закрытых наборов элементов. Закрытые наборы элементов показаны заштрихованными овалами. Зачеркивание представляет собой наборы элементов X_i, замененные на $X_i \cup X_j$ во время выполнения алгоритма. Редкие наборы элементов не отображаются.

5.4 Невыводимые наборы объектов

Набор объектов называется невыводимым, если его поддержка не может быть выведена из поддержки его подмножеств. Множество всех невыводимых наборов можно рассматривать как обобщённое или сжатое представление всех часто встречающихся наборов объектов.

Кроме того, точная поддержка всех остальных часто встречающихся наборов может быть выведена из этого множества.

5.4.1 Обобщённые наборы элементов

Пусть $\mathcal{T}$ – множество идентификаторов транзакций, $\mathcal{I}$ – множество объектов, X – k-набор, то есть, $X = \{x_1, x_2, \dots, x_k\}$. Рассмотрим набор тидов $\mathbf{t}(x_i)$ для каждого элемента $x_i \in X$. Эти k наборов разбивают множество всех идентификаторов на 2^k подмножеств, некоторые из которых могут быть пустыми, и каждое такое подмножество содержит идентификаторы для некоторого подмножества объектов $Y \subseteq X$, но не содержит оставшиеся объекты $Z = X \backslash Y$. Каждое такое подмножество (область) является, таким образом, набором идентификаторов обобщённого набора объектов, включающего объекты из X и их отрицания. Таким образом, этот обобщённый набор объектов можно записать как $Y\bar{Z}$, где Y состоит из обычных объектов, а Z состоит из отрицаний элементов. Мы будем называть поддержкой обощённого набора объектов $Y\bar{Z}$ число транзакций, которые содержат все объекты из Y, но не содержат объекты из Z:

$$sup(Y\bar{Z}) = |\{t \in \mathcal{T} \mid Y \subseteq \mathbf{i}(t) \text{ и } Z \cap \mathbf{i}(t) = \emptyset\}|.$$

5.4.2 Принцип включения-исключения

Пусть $Y\bar{Z}$ – обобщённый набор объектов, а $X = Y \cup Z = YZ$. Принцип включения-исключения позволяет напрямую вычислить поддержку $Y\bar{Z}$ как комбинацию поддержки всех наборов объектов W, таких, что $Y \subseteq W \subseteq X$:

$$sup(Y\bar{Z}) = \sum_{Y \subseteq W \subseteq X} -1^{|W \backslash Y|} * sup(W)\,. \tag{5.2}$$

5.4.3 Границы поддержки набора элементов

Обратите внимание, что в формуле включения-исключения из ур. 5.2 для поддержки $Y\bar{Z}$ есть все подмножества между Y и $X = Y\bar{Z}$. Иными словами, для данного k-набора X существует 2^k обобщённых наборов объектов в виде $Y\bar{Z}$, где $Y \subseteq X$ и $Z = X \backslash Y$, причём у каждого обобщённого набора есть условие для $sup(X)$ в уравнении включения-исключения: это верно для $W = X$. Поскольку поддержка каждого (обобщённого) набора должна быть не отрицательной, мы можем вывести оценку границы поддержки X из каждого из 2^k обобщённых наборов элементов, установив $sup(Y\bar{Z}) \geq 0$. Однако заметим, что в формуле (9.2) коэффициент при $sup(X)$ равен $+1$ всякий раз, когда $|X \backslash Y|$ чётно, и -1, когда $|X \backslash Y|$ нечётно. Тогда из возможных 2^k подмножеств $Y \subseteq X$ мы можем вывести 2^{k-1} нижних границ и 2^{k-1} верхних границ для $sup(X)$, полученных после $sup(Y\bar{Z}) \geq 0$ перестановкой членов в уравнении включения-исключения так, что $sup(X)$ оказывается слева, а оставшиеся члены справа:

Верхние границы ($|X \backslash Y|$ нечётные):

$$sup(X) \leq \textstyle\sum_{Y \subseteq W \subseteq X} -1^{(|X \backslash W|+1)} * sup(W)\,, \tag{5.3}$$

Нижние границы ($|X \backslash Y|$ чётные):

$$sup(X) \geq \sum_{Y \subseteq W \subseteq X} -1^{(|X \setminus W|+1)} * sup(W)\,. \quad (5.4)$$

Заметим, что разница между уравнениями (5.3) и (5.4) только в знаке неравенства, который зависит от начального подмножества Y.

5.4.4 Невыводимые наборы объектов

Пусть дан набор объектов X, пусть $Y \subseteq X$. Обозначим через $IE(Y)$ сумму:

$$IE(Y) = \sum_{Y \subseteq W \subseteq X} -1^{(|X \setminus W|+1)} * sup(W).$$

Тогда множества всех верхних и нижних границ для поддержки $sup(X)$ будут иметь вид:

$$UB(X) = \{IE(Y) \mid Y \subseteq X, |X \setminus Y| \text{ не чётное}\}$$

$$LB(X) = \{IE(Y) \mid Y \subseteq X, |X \setminus Y| \text{ чётное}\}.$$

Набор объектов X называется *невыводимым*, если $\max\{LB(X)\} \neq \min\{UB(X)\}$, то есть, поддержка X не может быть выведена из значений поддержки подмножеств, можно лишь оценить диапазон возможных значений:

$$sup(X) \in [\max\{LB(X)\}, \min\{UB(X)\}].$$

Напротив, X – выводимый набор, если $sup(X) = \max\{LB(X)\} = \min\{UB(X)\}$, поскольку в этом случае поддержка X может быть выведена напрямую из поддержки его подмножеств. Таким образом, набор всех часто встречающихся невыводимых объектов может быть задан как:

$$\mathcal{N} = \{X \in \mathcal{F} \mid \max\{LB(X)\} \neq \min\{UB(X)\}\},$$

где $\mathcal{F}$ - набор всех часто встречающихся объектов.

Глава 6. Анализ последовательностей

Многие реальные сферы, такие как биоинформатика, веб-интеллектуальный анализ и интеллектуальный анализ текста, должны иметь дело с последовательными и временными данными. Анализ последовательности помогает обнаруживать закономерности во времени или положениях в заданном наборе данных. В этой главе мы рассмотрим методы поиска частых последовательностей, которые допускают пропуски между элементами, а также методы поиска частых подстрок, которые не допускают пропусков между последовательными элементами.

6.1. Частота последовательности

Пусть Σ обозначает алфавит, определенный как конечный набор знаков или символов, и пусть $|\Sigma|$ - его мощность. Последовательность или строка определяется как упорядоченный список символов и записывается как $s = s_1 s_2 \ldots s_k$, где $s_i \in \Sigma$- символ с индексом i, также обозначаемый как $s[i]$. Здесь $|s| = k$ обозначает длину последовательности. Последовательность длины k также называется k-последовательностью. Мы используем обозначение $s[i{:}\, j] = s_i s_{i+1} \ldots s_{j-1} s_j$для обозначения подстроки или последовательности последовательных символов в позициях с i по j, где $j > i$. Определим префикс последовательности s как любую подстроку вида $s[1{:}\, i] = s_1 s_2 \ldots s_{i-1} s_i$, где $0 \leqslant i \leqslant n$. Также определим суффикс s как любую подстроку вида s $s[i{:}\, n] = s_i s_{i+1} \ldots s_{n-1} s_n$, где $1 \leqslant i \leqslant n + 1$. Обратите внимание, что $s[1{:}\, 0]$ - пустой префикс, а $s[n + 1{:}\, n]$ - пустой суффикс. Пусть Σ^* будет набором всех возможных последовательностей, которые могут быть построены с использованием символов из Σ, включая пустую последовательность $\emptyset$ (которая имеет нулевую длину).

Пусть $s = s_1 s_2 \ldots s_n$ и $r = r_1 r_2 \ldots r_m$ - две последовательности над Σ. Мы говорим, что r - подпоследовательность s, обозначаемая $r \subseteq s$, если существует взаимно однозначное отображение $\phi{:}\, [1, m] \to [1, n]$, такое что $r[i] = s[\phi(i)]$ и для любых двух позиций i, j в r, $i < j \Longrightarrow \phi(i) < \phi(j)$. Другими словами, каждая позиция в r отображается в другую позицию в s, и порядок символов сохраняется, даже если между последовательными элементами r в отображении может быть промежуточный разрыв. Если $r \subseteq s$, мы также говорим, что s содержит r. Последовательность r называется последовательной подпоследовательностью или подстрокой s при условии $r_1 r_2 \ldots r_m = s_j s_{j+1} \ldots s_{j+m-1}$, то есть $r[i{:}\, m] = s[j{:}\, j + m - 1]$, где $1 \leqslant j \leqslant n - m + 1$. Для подстрок мы не допускаем пробелов между элементами r в отображении.

Для базы данных $D = \{s_1 s_2 \ldots s_N\}$ из N последовательностей, и для некоторой последовательности r поддержка r в базе данных D определяется как общее количество последовательностей в D, содержащих r

$$\sup(r) = |\{ s_i \in D \mid r \subseteq s_i \}|$$

Относительная поддержка r — это доля последовательностей, содержащих r.

$$r\sup(r) = \sup(r)\,/N$$

Учитывая заданный пользователем порог $minsup$, мы говорим, что последовательность r часто встречается в базе данных D, если $sup(r) \geq minsup$. Частая последовательность *максимальна*, если она не является подпоследовательностью какой-либо другой частой последовательности, а частая последовательность *закрыта*, если она не является подпоследовательностью любой другой частой последовательности с той же поддержкой.

6.2 Поиск часто встречающихся последовательностей

Для поиска закономерностей в последовательностях важен порядок символов, и поэтому мы должны рассматривать все возможные *перестановки* символов как возможные частые кандидаты. Сравните это с поиском наборов, где нам нужно было только рассмотреть *комбинации элементов*. Пространство поиска последовательности может быть организовано в виде дерева поиска префиксов. Корень дерева на уровне 0 содержит пустую последовательность, причем каждый символ $x \in \Sigma$ является одним из его дочерних элементов. Таким образом, узел, помеченный последовательностью $\mathbf{s} = s_1 s_2 \ldots s_k$ на уровне k есть дети вида $\mathbf{s} = s_1 s_2 \ldots s_k s_{k+1}$ на уровне $k+1$. Другими словами, **s** – это префикс каждого дочернего элемента $\boldsymbol{s`}$**,** который также называется *расширением* **s.**

Пространство поиска подпоследовательности концептуально бесконечно, поскольку включает в себя все последовательности в Σ^*, то есть все последовательности нулевой или более длины, которые могут быть созданы с помощью символов в Σ. На практике база данных **D** состоит из последовательностей ограниченной длины. Пусть l обозначает длину самой длинной последовательности в базе данных, тогда в худшем случае нам придется рассмотреть все возможные последовательности длиной до l, что дает следующую привязку к размеру пространства поиска:

$$|\Sigma|^1 + |\Sigma|^1 + \cdots + |\Sigma|^l = O(|\Sigma|^l) \qquad (6.1)$$

так как на уровне k есть $|\Sigma|^k$ возможных подпоследовательностей длины k.

Алгоритм 6.1: Алгоритм GSP

GSP (D, Σ, *minsup*):

1 $\mathcal{F} \leftarrow \emptyset$
2 $\mathcal{C}^{(1)} \leftarrow \{\emptyset\}$ // Initial prefix tree with single symbols
3 **foreach** $s \in \Sigma$ **do** Add s as child of $\emptyset$ in $\mathcal{C}^{(1)}$ with $sup(s) \leftarrow 0$
4 $k \leftarrow 1$ // k denotes the level
5 **while** $\mathcal{C}^{(k)} \neq \emptyset$ **do**
6 COMPUTESUPPORT $(\mathcal{C}^{(k)}, \mathbf{D})$
7 **foreach** *leaf* $\mathbf{s} \in \mathcal{C}^{(k)}$ **do**
8 **if** $sup(\mathbf{r}) \geq minsup$ **then** $\mathcal{F} \leftarrow \mathcal{F} \cup \{(\mathbf{r}, sup(\mathbf{r}))\}$
9 **else** remove $\mathbf{s}$ from $\mathcal{C}^{(k)}$
10 $\mathcal{C}^{(k+1)} \leftarrow$ EXTENDPREFIXTREE $(\mathcal{C}^{(k)})$
11 $k \leftarrow k+1$
12 **return** $\mathcal{F}^{(k)}$

COMPUTESUPPORT $(\mathcal{C}^{(k)}, \mathbf{D})$:

13 **foreach** $\mathbf{s}_i \in \mathbf{D}$ **do**
14 **foreach** $\mathbf{r} \in \mathcal{C}^{(k)}$ **do**
15 **if** $\mathbf{r} \subseteq \mathbf{s}_i$ **then** $sup(\mathbf{r}) \leftarrow sup(\mathbf{r}) + 1$

EXTENDPREFIXTREE $(\mathcal{C}^{(k)})$:

16 **foreach** *leaf* $\mathbf{r}_a \in \mathcal{C}^{(k)}$ **do**
17 **foreach** *leaf* $\mathbf{r}_b \in$ CHILDREN(PARENT($\mathbf{r}_a$)) **do**
18 $\mathbf{r}_{ab} \leftarrow \mathbf{r}_a + \mathbf{r}_b[k]$ // extend $\mathbf{r}_a$ with last item of $\mathbf{r}_b$
 // prune if there are any infrequent subsequences
19 **if** $\mathbf{r}_c \in \mathcal{C}^{(k)}$, ***for all*** $\mathbf{r}_c \subset \mathbf{r}_{ab}$, *such that* $|\mathbf{r}_c| = |\mathbf{r}_{ab}| - 1$ **then**
20 Add $\mathbf{r}_{ab}$ as child of $\mathbf{r}_a$ with $sup(\mathbf{r}_{ab}) \leftarrow 0$
21 **if** *no extensions from* $\mathbf{r}_a$ **then**
22 remove $\mathbf{r}_a$, and all ancestors of $\mathbf{r}_a$ with no extensions, from $\mathcal{C}^{(k)}$
23 **return** $\mathcal{C}^{(k)}$

6.2.1 Уровневый поиск: GSP

Мы можем разработать эффективный алгоритм поиска последовательностей, который ищет дерево префиксов последовательностей с помощью поиска по уровням или по ширине. Учитывая множество частых последовательностей на уровне k, мы генерируем все возможные расширения последовательностей или кандидатов на уровне $k+1$. Затем мы вычисляем поддержку каждого кандидата и отсекаем те, которые не являются частыми. Поиск прекращается, когда более частые расширения невозможны.

Псевдокод для метода уровневого поиска обобщенной последовательности (GSP) показан в алгоритме 10.1. Он использует антимонотонное свойство поддержки для отсечения паттернов кандидатов, то есть: никакая суперсеквенция нечастой последовательности не может быть частой, и все подпоследовательности частой последовательности должны быть частыми. Префиксное дерево поиска на уровне k обозначается $C^{(k)}$. Первоначально $C^{(1)}$ включает в себя все символы в Σ. Учитывая текущий набор кандидатов k-последовательностей

$C^{(k)}$, метод сначала вычисляет их поддержку (строка 6). Для каждой последовательности базы данных $\mathbf{s}_i \in \mathbf{D}$, мы проверяем, является ли кандидат-последовательность $\mathbf{r} \in C^{(k)}$ подпоследовательностью $\mathbf{s}_i$. Если это так, то мы увеличиваем поддержку $\mathbf{r}$. Как только частые последовательности на уровне k найдены, мы генерируем кандидатов на уровень $k+1$ (строка 10). Для расширения, каждый лист $\mathbf{r}_a$ расширяется последним символом любого другого листа $\mathbf{r}_b$, который имеет один и тот же префикс (т.е. имеет одного и того же родителя), чтобы получить новый префикс. кандидат $(k+1)$-последовательность $\mathbf{r}_{ab} = \mathbf{r}_a + \mathbf{r}_b[k]$ (строка 18). Если новый кандидат $\mathbf{r}_{ab}$ содержит любую нечастую k-последовательность, мы отсекаем его.

Вычислительная сложность GSP равна $O(|\Sigma|^l)$, согласно ур-ю (10.1), где l – длина самой длинной частой последовательности. Сложность ввода-вывода равна $O(l \cdot \mathbf{D})$, потому что мы вычисляем поддержку целого уровня в одном сканировании базы данных.

6.2.2 Вертикальный поиск последовательности: Spade

Алгоритм Spade (англ. «лопата») использует вертикальное представление базы данных для поиска последовательностей. Идея состоит в том, чтобы записать для каждого символа идентификаторы последовательности и позиции, в которых он встречается. Для каждого символа $s \in \Sigma$ мы сохраняем набор кортежей вида $\langle i, pos(s)\rangle$, где $pos(s)$ – набор позиций в последовательности базы данных $\mathbf{s}_i \in \mathbf{D}$, где появляется символ s. Пусть $L(s)$ обозначает множество таких кортежей позиции последовательностей для символа s, который мы называем списком позиций. Набор списков позиций для каждого символа $s \in \Sigma$ образует вертикальное представление входной базы данных. В целом, учитывая k-последовательность $\mathbf{r}$, ее список позиций L(r) поддерживает список позиций для вхождений последнего символа $\mathbf{r}[k]$ в каждой последовательности базы данных $\mathbf{s}_i$, при условии $\mathbf{r} \subseteq \mathbf{s}_i$. Поддержка последовательности $\mathbf{r}$ – это просто число различных последовательностей, в которых есть $\mathbf{r}$, то есть $sup(\mathbf{r}) = |L(s)|$.

Поддержка вычислений в Spade осуществляется с помощью последовательных операций соединения. Даны списки позиций для любых двух k-последовательностей $\mathbf{r}_a$ и $\mathbf{r}_b$, которые имеют одинаковую длину префикса (k-1), идея состоит в том, чтобы выполнить последовательные соединения в списках позиций для вычисления поддержки новой $(k+1)$ длины последовательности кандидатов $\mathbf{r}_{ab} = \mathbf{r}_a + \mathbf{r}_b[k]$. Дан кортеж $\langle i, pos(\mathbf{r}_b[k])\rangle \in L(\mathbf{r}_b)$, мы сначала проверяем, существует ли кортеж $\langle i, pos(\mathbf{r}_a[k])\rangle \in L(\mathbf{r}_a)$, то есть обе последовательности должны происходить в одной и той же последовательности базы данных $\mathbf{s}_i$. Далее, для каждой позиции $p \in pos(\mathbf{r}_b[k])$ мы проверяем, существует ли позиция $q \in pos(\mathbf{r}_a[k])$ так, что $q < p$. Если да, то это означает, что символ $\mathbf{r}_b[k]$ происходит после последней позиции $\mathbf{r}_a$и, таким образом, мы сохраняем p как верное вхождение $\mathbf{r}_{ab}$. Список позиций $L(\mathbf{r}_{ab})$ включает в себя все такие случаи. Обратите внимание, как мы отслеживаем позиции только для последнего символа в последовательности кандидатов. Это происходит потому, что мы расширяем последовательности из общего префикса, поэтому нет необходимости отслеживать все вхождения символов в префикс. Последовательное соединение обозначим как $L(\mathbf{r}_{ab}) = L(\mathbf{r}_a) \cap L(\mathbf{r}_b)$.

Основное преимущество вертикального подхода заключается в том, что он позволяет использовать различные стратегии поиска в пространстве поиска последовательности, включая поиск по ширине или глубине. В алгоритме 10.2 показан псевдокод для Spade. Учитывая набор последовательностей P, которые совместно используют один и тот же

префикс, а также их списки позиций, метод создает новый класс эквивалентности префикса P_a для каждой последовательности $\mathbf{r}_a \in P$ путем выполнения последовательных соединений с каждой последовательностью $\mathbf{r}_b \in P$, включая самосоединения. После удаления нечастых расширений, новый класс эквивалентности P_a обрабатывается рекурсивно.

Алгоритм 6.2: Алгоритм Spade

```
// Initial Call: F ← ∅, k ← 0,
    P ← {⟨s, L(s)⟩ | s ∈ Σ, sup(s) ≥ minsup}
SPADE (P, minsup, F, k):
1 foreach r_a ∈ P do
2     F ← F ∪ {(r_a, sup(r_a))}
3     P_a ← ∅
4     foreach r_b ∈ P do
5         r_ab = r_a + r_b[k]
6         L(r_ab) = L(r_a) ∩ L(r_b)
7         if sup(r_ab) ≥ minsup then
8             P_a ← P_a ∪ {⟨r_ab, L(r_ab)⟩}
9     if P_a ≠ ∅ then SPADE (P, minsup, F, k + 1)
```

6.2.3 Поиск последовательностей на основе проекций: PrefixSpan

Пусть **D** обозначает базу данных, а $s \in \Sigma$ – любой символ. Проецируемая база данных относительно s, обозначаемая $\mathbf{D}_s$, получается путем нахождения первого вхождения s в $\mathbf{s}_i$, скажем, в позиции p. Далее мы сохраняем в $\mathbf{D}_s$, только суффикс $\mathbf{s}_i$, начинающегося с позиции $p+1$. Кроме того, любые нечастые символы удаляются из суффикса. Это делается для каждой последовательности $\mathbf{s}_i \in \mathbf{D}$.

Основная идея PrefixSpan состоит в том, чтобы вычислить поддержку только отдельных символов в проецируемой базе данных $\mathbf{D}_s$, а затем выполнить рекурсивные проекции на частые символы в первую очередь по глубине. Метод PrefixSpan описан в разделе Алгоритм 10.3. В нём **r** – это частая подпоследовательность, и $\mathbf{D}_\mathbf{r}$ – ожидаемый набор данных для **r**. Изначально **r** пуст и $\mathbf{D}_\mathbf{r}$ представляет из себя весь входной набор данных **D**. Учитывая базу данных (проецируемых) последовательностей $\mathbf{D}_\mathbf{r}$, PrefixSpan сначала находит все частые символы в проецируемом наборе данных. Для каждого такого символа s мы расширяем **r**, добавляя s, чтобы получить новую частую подпоследовательность $\mathbf{r}_s$. Затем мы создаем проецируемый набор данных $\mathbf{D}_s$, проецируя $\mathbf{D}_\mathbf{r}$ на символ s. Затем рекурсивно вызывается PrefixSpan с $\mathbf{r}_s$ и $\mathbf{D}_\mathbf{s}$,

Алгоритм 6.3: Алгоритм PrefixSpan

```
// Initial Call: D_r ← D, r ← ∅, F ← ∅
PREFIXSPAN (D_r, r, minsup, F):
1 foreach s ∈ Σ such that sup(s, D_r) ≥ minsup do
2     r_s = r + s // extend r by symbol s
3     F ← F ∪ {(r_s, sup(s, D_r))}
4     D_s ← ∅ // create projected data for symbol s
5     foreach s_i ∈ D_r do
6         s'_i ← projection of s_i w.r.t symbol s
7         Remove any infrequent symbols from s'_i
8         Add s'_i to D_s if s'_i ≠ ∅
9     if D_s ≠ ∅ then PREFIXSPAN (D_s, r_s, minsup, F)
```

6.3 Поиск подстрок с помощью суффиксных деревьев

Теперь мы рассмотрим эффективные методы поиска частых подстрок. Пусть **s** – последовательность длиной n, тогда в **s** содержится не более $O(n^2)$ возможных различных подстрок в **s**. Чтобы убедиться в этом, рассмотрим подстроки длины w, из которых в s имеется n-w+1 возможных подстрок в **s**. Сложив все длины подстрок, получим

$$\sum_{w=1}^{n}(n-w+1) = n + (n-1) + \cdots + 2 + 1 = O(n^2)$$

Это гораздо меньшее пространство поиска по сравнению с подпоследовательностями, и поэтому мы можем разработать более эффективные алгоритмы для решения задачи поиска частых подстрок. Фактически, мы можем найти все частые подстроки в худшем случае за $O(Nn^2)$ времени для набора данных $\mathbf{D} = \{\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_k\}$ с N последовательностями.

6.3.1 Суффиксное Дерево

Пусть Σ – алфавит и $\$ \notin \Sigma$ – терминальный символ, используемый для обозначения конца строки. Используя последовательность **s**, мы добавляем терминальный символ так, что $\mathbf{s} = s_1 s_2 \ldots s_n s_{n+1}$, где $s_{n+1} = \$$, а j-й суффикс задается как $\mathbf{s}[j{:}\,n+1] = s_j \ldots s_n s_{n+1}$. *Суффиксальное дерево* последовательностей в базе данных **D**, обозначаемое T, хранит все суффиксы для каждого $\mathbf{s}_i \in \mathbf{D}$ в древовидной структуре, где суффиксы, имеющие общий префикс, лежат на одном и том же пути от корень дерева. Подстрока, полученная путем объединения всех символов от корневого узла до узла v, называется *меткой узла* v и обозначается как $L(v)$. Подстрока, которая появляется на ребре (v_a, v_b), называется *меткой ребра* и обозначается как $L(v_a, v_b)$. Суффиксальное дерево имеет два типа узлов: внутренние и конечные узлы. Внутренний узел в дереве суффиксов (за исключением корня) имеет по крайней мере два дочерних элемента, где каждая метка ребра к дочернему элементу начинается с другого символа. Поскольку терминальный символ уникален, то в дереве суффиксов столько же листьев, сколько уникальных суффиксов во всех последовательностях. Каждый узел-лист соответствует суффиксу одной или нескольких последовательностей в **D**.

Нетрудно получить квадратичное время и объем для алгоритма построения дерева суффиксов. Изначально дерево суффиксов T пусто. Далее, для каждой последовательности $\mathbf{s}_i \in \mathbf{D}$, $|\mathbf{s}_i| = n_i$, мы генерируем все ее суффиксы $\mathbf{s}_i[j{:}\,n+1]$, при $1 \le j \le n_i$, и вставляем

каждый суффикс в дерево, следуя от корня, пока либо не доберемся до листьев, либо не встретим несоответствие в один из символов вдоль ребра. Если мы достигаем листа, мы вставляем пару *(i, j)* в лист, отмечая, что это j-й суффикс последовательности $\mathbf{s}_i$. Если мы нашли несоответствие в одном из символов, скажем, в позиции $p>j$, мы добавляем внутреннюю вершину непосредственно перед несовпадением и создаем новый узел-лист, содержащий *(i, j)* с меткой ребра $\mathbf{s}_i[j: n+1]$.

С точки зрения сложности времени и требуемой памяти, алгоритм, описанный выше, требует $O(Nn^2)$ время и памяти, где N – число последовательностей в **D**, а n – самая длинная длина последовательности. Временная сложность вытекает из того, что метод всегда вставляет новый суффикс, начиная с корня дерева суффиксов. Это означает, что в худшем случае он сравнивает $O(n)$ символов на вставку суффикса, давая наихудшую границу $O(n^2)$ для всех n суффиксов. Сложность памяти возникает из-за того, что каждый суффикс явно представлен в дереве, занимая $n + (n-1) + \cdots + 2 + 1 = O(n^2)$ памяти. Для N последовательностей в базе данных, мы получаем $O(Nn^2)$ как наихудший случай временных границ и границ памяти.

Частые Подстроки

Как только суффиксальное дерево построено, мы можем вычислить все частые подстроки, проверяя, сколько различных последовательностей появляется в листовом узле или во внутреннем узле. Метки узлов для узлов с поддержкой хотя бы *minsup* дают набор частых подстрок; все префиксы таких меток узлов также являются частыми. Дерево суффиксов также может поддерживать специальные запросы для поиска всех вхождений в базу данных любой запрашиваемой подстроки **q**. Для каждого символа в **q** мы следуем по пути от корня до тех пор, пока не увидим все символы в **q** или пока не обнаружим несоответствие в любой позиции. Если **q** найден, то набор листьев под этим путем является списком вхождений **q**. С другой стороны, если есть несоответствие, это означает, что запрос не проходит в базе данных. С точки зрения сложности времени запроса, поскольку мы должны сопоставить каждый символ в **q**, мы сразу же получаем $O(|\mathbf{q}|)$ в качестве временной границы (предполагая, что $|\Sigma|$ – это константа), которая *не зависит* от размера базы данных. Перечисление всех совпадения занимают дополнительное время, для общей временной сложности $O(|\mathbf{q}| + k)$, если есть k совпадений.

6.3.2 Алгоритм линейного времени Укконена

Теперь мы представляем линейный алгоритм времени и памяти для построения суффиксных деревьев. Сначала мы рассмотрим, как построить дерево суффиксов для одной последовательности $\mathbf{s} = s_1 s_2 \ldots s_n s_{n+1}$, причем $s_{n+1} = \$$. Дерево суффиксов для всего набора данных из N последовательностей может быть получено путем вставки каждой последовательности по одной.

Достижение линейной памяти

Давайте посмотрим, как уменьшить требования к памяти суффиксного дерева. Если алгоритм хранит все символы на каждой метке ребра, то сложность памяти равна $O(n^2)$, и мы также не можем достичь линейного времени построения. Хитрость заключается в том, чтобы явно не хранить все метки ребер, а использовать метод сжатия ребер, когда мы храним только начальную и конечную позиции метки ребер во входной строке **s**. То есть, если метка ребра задана как $\mathbf{s}[i: j]$, то мы представляем её как интервал $[i, j]$.

С точки зрения сложности памяти обратите внимание, что, когда мы добавляем новый суффикс к дереву T, оно может создать не более одного нового внутреннего узла. Поскольку существует *n* суффиксов, в T есть *n* листьев и не более *n* внутренних узлов. Имея не более *2n* узлов, дерево имеет не более *2n–1* ребер, и поэтому общее пространство, необходимое для хранения интервала для каждого ребра, равно $2(2n - 1) = 4n - 2 = O(n)$.

Достижение Линейного Времени

Метод Укконена – это *онлайн* алгоритм, то есть при заданной строке $\mathbf{s} = s_1 s_2 \ldots s_n\$$ он строит полное суффиксальное дерево поэтапно. Фаза i строит дерево до i-го символа в **s**, то есть обновляет дерево суффиксов из предыдущей фазы, добавляя следующий символ s_i. Пусть T_i , обозначим суффиксальное дерево вплоть до i-го префикса $\mathbf{s}[i:j]$ с $1 \leq j \leq n$. Алгоритм Укконена строит T_i из T_{i-1} , убедившись, что все суффиксы, включая текущий символ s_i, находятся в новом промежуточном дереве T. Другими словами, на i-й фазе он вставляет в дерево T_i все суффиксы $\mathbf{s}[i:j]$ от $j = 1$ до $j = i$. Каждая такая вставка называется j-м расширением i-й фазы. Как только мы обработаем терминальный символ в позиции $n+1$, мы получим конечное суффиксальное дерево T для **s**.

Алгоритм 6.4 показывает код для простой реализации подхода Укконена. Этот метод имеет кубическую временную сложность, поскольку для получения T_i из T_{i-1} требуется $O(i^2)$ времени, а для последней фазы требуется $O(n^2)$ времени. При *n* фазах общее время составляет $O(n^3)$. Наша цель состоит в том, чтобы показать, что это время может быть сведено только к $O(n)$ с помощью оптимизаций, описанных в следующих параграфах.

Неявные суффиксы

Эта оптимизация утверждает, что в фазе *i*, если *j*-е расширение $\mathbf{s}[j:i]$ найдено в дереве, то любые последующие расширения также будут найдены, и, следовательно, нет необходимости обрабатывать дальнейшие расширения в фазе i. Таким образом, суффиксальное дерево T в конце фазы i имеет неявные суффиксы, соответствующие расширениям $j+1$ через *i*. Важно отметить, что все суффиксы станут явными, когда мы впервые столкнемся с новой подстрокой, которая еще не существует в дереве. Это наверняка произойдет в фазе $n+1$, когда мы обрабатываем терминальный символ \$, так как он не может встречаться больше нигде в **s** (в конце концов, $\$ \notin \Sigma$).

Алгоритм 6. 4: Наивный алгоритм Укконена

```
NAIVEUKKONEN (s):
1 n ← |s|
2 s[n+1] ← $ // append terminal character
3 T ← ∅ // add empty string as root
4 foreach i = 1, ..., n+1 do // phase i - construct T_i
5     foreach j = 1, ..., i do // extension j for phase i
          // Insert s[j:i] into the suffix tree
6         Find end of the path with label s[j:i-1] in T
7         Insert s_i at end of path;
8 return T
```

Неявные расширения

Пусть текущая фаза равна i, а $l \leq i-1$ – последняя явная фаза. Суффикс в предыдущем дереве T_{i-1}. Все явные суффиксы в T_{i-1} имеют реберные метки вида $[x: i+1]$, ведущие к соответствующим листовым узлам, где начальная позиция x является узлом, но конечная позиция должна быть i-1, потому что s_{i-1} был добавлен в конец этих путей в фазе i-1. В текущей фазе i мы должны были бы расширить эти пути, добавив s_i в конце. Однако вместо явного увеличения всех конечных позиций мы можем заменить конечную позицию указателем e, который отслеживает текущую обрабатываемую фазу. Если мы заменим $[x: i-1]$ на $[x: e]$, то на фазе i, если мы установим e=i, то сразу же все l существующих суффиксов будут *неявно* расширены до $[x: i]$. Таким образом, в одной операции приращения e мы, по сути, позаботились о расширениях от 1 до l для фазы i.

Хитрость пропуска/подсчета

Для j-го расширения фазы i мы должны искать подстроку $\mathbf{s}[j: i-1]$, так что мы можем добавить s_i в конец. Однако обратите внимание, что эта строка должна существовать в T_{i-1}, потому что мы уже обработали символ s_{i-1} на предыдущем этапе. Таким образом, вместо поиска каждого символа в $\mathbf{s}[j: i-1]$, начиная с корня, мы сначала подсчитываем количество символов на ребре, начинающемся с символов s_i; пусть эта длина равна m. Если m больше длины подстроки (т.е. если $m > i-j$), то подстрока должна заканчиваться на этом ребре, поэтому мы просто переходим в позицию $i-j$ и вставляем s_i. С другой стороны, если $m < i-j$, то мы можем перейти непосредственно к дочернему узлу, скажем v_c, и искать оставшуюся строку $\mathbf{s}[j+m: i-1]$ из v_c, используя ту же технику пропуска/подсчета. При такой оптимизации стоимость расширения становится пропорциональной количеству узлов на пути, а не количеству символов в $\mathbf{s}[j: i-1]$.

Суффиксальные ссылки

Мы видели, что с помощью оптимизации пропуска/подсчета мы можем искать подстроку $\mathbf{s}[j: i-1]$, следуя узлам от родителя к потомку. Однако каждый раз нам все равно приходится начинать с корневого узла. Мы можем избежать поиска от корня через использование *суффиксных ссылок*. Для каждого внутреннего узла v_a мы поддерживаем связь с внутренним узлом v_b, где $L(v_b)$ – непосредственный суффикс $L(v_a)$. В расширении j-1 пусть v_p обозначает внутренний узел, под которым мы находим $\mathbf{s}[j: i-1]$, и пусть m – длина метки узла v_p. Чтобы вставить j-е расширение $\mathbf{s}[j: i]$, мы следуем по суффиксной ссылке из v_p, в другой узел, скажем v_s, и ищем оставшуюся подстроку $\mathbf{s}[j+m-1: i-1]$ из v_s. Использование суффиксных ссылок позволяет нам прыгать внутри дерева для различных расширений, в отличие от поиска из корня каждый раз. В качестве заключительного замечания, если расширение j создает новый внутренний узел, тогда его суффиксная ссылка будет указывать на новый внутренний узел, который будет создан во время расширения j+1.

Алгоритм 6. 5: Алгоритм Укконена

```
UKKONEN (s):
1  n ← |s|
2  s[n+1] ← $ // append terminal character
3  T ← ∅ // add empty string as root
4  l ← 0 // last explicit suffix
5  foreach i = 1,...,n+1 do // phase i - construct T_i
6  |  e ← i // implicit extensions
7  |  foreach j = l+1,...,i do // extension j for phase i
   |  |  // Insert s[j:i] into the suffix tree
8  |  |  Find end of s[j:i-1] in T via skip/count and suffix links
9  |  |  if s_i ∈ T then // implicit suffixes
10 |  |  |  break
11 |  |  else
12 |  |  |  Insert s_i at end of path
13 |  |  |  Set last explicit suffix l if needed
14 return T
```

Псевдокод оптимизированного алгоритма Укконена показан на разделе Алгоритм 6.5. Важно отметить, что он достигает линейного времени и памяти только со всеми оптимизациями в сочетании, а именно неявными расширениями (строка 6), неявными суффиксами (строка 9), ссылками пропуска/подсчета и суффиксными ссылками для вставки расширений в T (строка 8).

Алгоритм Укконена имеет временную сложность $O(n)$ для последовательности длины n, поскольку он выполняет только постоянный объем работы (амортизируемый), чтобы сделать каждый суффикс явным. Обратите внимание, что для каждой фазы определенное число расширений выполняется неявно, просто увеличивая e. После i расширений от $j = 1$ до $j = i$, скажем, что l выполняются неявно. Для остальных расширений мы останавливаемся в первый раз, когда некоторый суффикс неявно находится в дереве; пусть это расширение будет k. Таким образом, фаза i должна добавлять явные суффиксы только для суффиксов l+1 через k-1. Для создания каждого явного суффикса мы выполняем постоянное число операций, которое включает в себя следование за ближайшей суффиксной связью, пропуск/подсчет для поиска первого несоответствия и вставку, если это необходимо, нового конечного узла суффикса. Поскольку каждый лист становится явным только один раз, а количество шагов пропуска/подсчета ограничено $O(n)$ по всему дереву, мы получаем наихудший случай времени алгоритма $O(n)$. Таким образом, общее время по всей базе данных из N последовательностей равно $O(Nn)$, если n – самая длинная длина последовательности.

Глава 7. Оценка паттернов и правил

В этой главе мы обсудим, как оценивать значимость предложенных частых паттернов, а также правила ассоциации, вытекающие из них. В идеале тематические шаблоны и правила должны удовлетворять желаемым свойствам, таким как краткость, новизна, полезность и так далее. Мы выделяем несколько критериев и методов оценки, которые направлены на количественную оценку различных свойств добытых результатов. Как правило, вопрос о том, интересен ли шаблон или правило, является в значительной степени субъективным. Однако мы, безусловно, можем попытаться устранить правила и закономерности, которые не являются статистически значимыми. В этой главе также рассматриваются методы проверки статистической значимости и получения доверительных границ для статистического значения теста.

7.1. Меры для оценки паттернов и правил

Пусть $\mathcal{I}$ - набор элементов, $\mathcal{T}$ - набор тидов и пусть $\boldsymbol{D} \subseteq \mathcal{T} \times \mathcal{I}$- двоичная база данных. Напомним, что правило ассоциации - это выражение $X \longrightarrow Y$, где X и Y - это наборы элементов, т.е. $X, Y \subseteq \mathcal{I}$ и $X \cap Y = \emptyset$. Мы называем X антецедентом правила, а Y - следствием.

Набор тидов для набора элементов X - это набор всех тидов, содержащих X, заданный как

$$t(X) = \{t \in \mathcal{T} \mid X \text{ содержится в } t\}$$

Таким образом, поддержка X равна $\sup(X) = |t(X)|$. В последующем обсуждении мы используем краткую форму XY для обозначения объединения XY наборов элементов X и Y.

Учитывая часто встречающийся набор элементов $Z \in F$, где F - множество всех часто встречающихся наборов элементов, мы можем вывести различные правила ассоциации, рассматривая каждое собственное подмножество Z как антецедент, а остальные элементы как консеквент, то есть для каждого $Z \in F$, мы можем вывести набор правил вида $X \longrightarrow Y$, где $X \subset Z$ и $Y = Z \setminus X$.

7.1.1. Меры по оценке правил

Различные меры интересности правил пытаются количественно определить зависимость между следствием и антецедентом. Ниже мы рассмотрим некоторые общие меры оценки правил, начиная с поддержки и достоверность.

Поддержка (Support)

Поддержка правила определяется как количество транзакций, содержащих как X, так и Y, то есть

$$\sup(X \longrightarrow Y) = sup(XY) = |t(XY)| \qquad (7.1)$$

Относительная поддержка — это доля транзакций, содержащих как X, так и Y, то есть эмпирическая совместная вероятность элементов, входящих в правило.

$$r\sup(X \longrightarrow Y) = P(XY) = r\sup(XY) = \frac{\sup(XY)}{|D|}$$

Обычно нас интересуют частые правила с $sup(X \longrightarrow Y) \geq minsup$, где $minsup$ - минимальный порог поддержки, определяемый пользователем. Когда минимальная поддержка указывается в виде дроби, подразумевается относительная поддержка. Обратите внимание, что (относительная) поддержка - это симметричная мера, потому что $\sup(X \longrightarrow Y) = \sup(Y \longrightarrow X)$.

Достоверность (Confidence)

Достоверность правила — это условная вероятность того, что транзакция содержит консеквент Y, при условии, что он содержит антецедент X:

$$conf(X \longrightarrow Y) = P(Y \mid X) = \frac{P(XY)}{P(X)} = \frac{rsup(XY)}{rsup(X)} = \frac{\sup(XY)}{\sup(X)}$$

Обычно нас интересуют правила с высокой степенью достоверности с $\text{conf}(X \longrightarrow Y) \geq minconf$, где $minconf$ — это минимальное значение достоверности, определяемое пользователем. Достоверность не является симметричной мерой, потому что по определению она зависит от антецедента.

Лифт (Lift)

Лифт определяется как отношение наблюдаемой совместной вероятности X и Y к ожидаемой совместной вероятности, если бы они были статистически независимыми, то есть

$$lift(X \longrightarrow Y) = \frac{P(XY)}{P(X) \cdot P(Y)} = \frac{rsup(XY)}{rsup(X) \cdot rsup(Y)} = \frac{conf(X \longrightarrow Y)}{rsup(Y)}$$

Один из распространенных способов использования лифта - измерить неожиданность правила. Значение лифта, близкое к 1, означает, что ожидается поддержка правила с учетом поддержки его компонентов. Обычно мы ищем значения, которые намного больше (т. е. выше ожидаемого) или меньше 1 (т. е. ниже ожидаемого).

Обратите внимание, что лифт — это симметричная мера, и она всегда больше или равна достоверности, потому что это достоверность, деленная на вероятность консеквента. Лифт также не является нижним набором, т.е. если предположить, что $X' \subset X$ и $Y' \subset Y$, может случиться так, что $lift(X' \longrightarrow Y')$ может быть выше $lift(X \longrightarrow Y)$. Лифт может быть подвержен шуму в небольших наборах данных, поскольку редкие или нечастые наборы элементов, которые встречаются всего несколько раз, могут иметь очень высокие значения лифта.

Усиление (Leverage, Рычаг)

Усиление измеряет разницу между наблюдаемой и ожидаемой совместной вероятностью XY при условии, что X и Y независимы.

$$\text{leverage}\ (X \longrightarrow Y) = P(XY) - P(X) \cdot P(Y) = \text{rsup}(XY) - \text{rsup}(X) \cdot \text{rsup}(Y)$$

Усиление дает «абсолютную» меру того, насколько неожиданно правило, и его следует использовать вместе с лифтом. Как и лифт — симметричный.

Коэффициент Жаккара (Jaccard)

Коэффициент Жаккара измеряет сходство между двумя наборами. При применении в качестве меры оценки правил он вычисляет сходство между наборами значений X и Y:

$$jaccard(X \longrightarrow Y) = \frac{|t(X) \cap t(Y)|}{|t(X) \cup t(Y)|} = \frac{sup(XY)}{sup(X) + sup(Y) - sup(XY)} = \frac{P(XY)}{P(X) + P(Y) — P(XY)}$$

Жаккар - симметричная мера.

Убежденность (Conviction)

Все меры оценки правил, которые мы рассмотрели выше, используют только совместную вероятность X и Y. Определим $\neg X$ как событие, в котором X не содержится в транзакции, то есть $X \nsubseteq t \in \mathcal{T}$, и аналогично для $\neg Y$. Как правило, существует четыре возможных события в зависимости от возникновения или отсутствия наборов элементов X и Y, как показано в таблице непредвиденных обстоятельств, представленной в табл.7.1.

Таблица 7.1 — Таблица непредвиденных обстоятельств для X и Y

	Y	$\neg Y$	
X	$sup(XY)$	$sup(X\neg Y)$	$sup(X)$
$\neg X$	$sup(\neg XY)$	$sup(\neg X\neg Y)$	$sup(\neg X)$
	$sup(Y)$	$sup(\neg Y)$	$\|D\|$

Убежденность измеряет ожидаемую ошибку правила, то есть, как часто X встречается в транзакции, а Y - нет. Таким образом, это мера силы правила по отношению к дополнению консеквента, определяемого как

$$conv(X \longrightarrow Y) = \frac{P(X) \cdot P(\neg Y)}{P(X\neg Y)} = \frac{1}{lift(X \longrightarrow \neg Y)}$$

Если совместная вероятность $X\neg Y$ меньше ожидаемой при независимости от X и $\neg Y$, тогда убежденность высока, и наоборот. Это асимметричная мера.

Из таблицы 7.1 видно, что $P(X) = P(XY) + P(X\neg Y)$, откуда следует, что $P(X\neg Y) = P(X) - P(XY)$. Далее, $P(\neg Y) = 1 - P(Y)$. Таким образом, мы имеем

$$conv(X \longrightarrow Y) = \frac{P(X) \cdot P(\neg Y)}{P(X) - P(XY)} = \frac{P(\neg Y)}{1 - P(XY)/P(X)} = \frac{1 - r\,\mathrm{sup}(Y)}{1 - conf(X \longrightarrow Y)}$$

Мы заключаем, что убежденность бесконечна, если достоверность едина. Если X и Y независимы, то убежденность равна 1.

Соотношение шансов (Odds Ratio)

В соотношении шансов используются все четыре записи из таблицы непредвиденных обстоятельств, представленной в табл. 7.1. Давайте разделим набор данных на две группы транзакций – т.е., которые содержат X, и те, которые не содержат X. Определим шансы Y в этих двух группах следующим образом:

$$odds(Y \mid X) = \frac{P(XY)/P(X)}{P(X\neg Y)/P(X)} = \frac{P(XY)}{P(X\neg Y)}$$

$$odds(Y \mid \neg X) = \frac{P(\neg XY)/P(\neg X)}{P(\neg X\neg Y)/P(\neg X)} = \frac{P(\neg XY)}{P(\neg X\neg Y)}$$

Тогда соотношение шансов определяется как отношение этих двух шансов:

$$\text{oddsratio}(X \longrightarrow Y) = \frac{\text{odds}(Y|X)}{\text{odds}(Y|\neg X)} = \frac{P(XY)\cdot P(\neg X\neg Y)}{P(X\neg Y)\cdot P(\neg XY)} = \frac{sup(XY)\cdot sup(\neg X\neg Y)}{sup(X\neg Y)\cdot sup(\neg XY)}$$

Соотношение шансов — это симметричная мера, и если X и Y независимы, тогда оно имеет значение 1. Таким образом, значения, близкие к 1, могут указывать на небольшую зависимость между X и Y. Отношение шансов больше 1 означает более высокие шансы Y встречается в присутствии X в отличие от его дополнения$\neg X$, тогда как шансы меньше единицы подразумевают более высокие шансы того, что Y встречается с $\neg X$.

7.1.2 Меры оценки модели

Теперь мы рассмотрим на показателях оценки паттернов.

Поддержка (Support)

Самыми основными мерами являются поддержка и относительная поддержка, указывающие количество и долю транзакций в D, которые содержат набор элементов X:

$$\sup(X) = |t(X)| \quad rsup(X) = \frac{\sup(X)}{|D|}$$

Лифт (Lift)

Подъем k-элементного множества $X = \{x_1, x_2, \ldots, x_k\}$ в наборе данных D определяется как

$$lift(X, D) = \frac{P(X)}{\prod_{i=1}^{k} P(x_i)} = \frac{r\sup(X)}{\prod_{i=1}^{k} r\sup(x_i)} \quad (7.2)$$

то есть отношение наблюдаемой совместной вероятности элементов в X к ожидаемой совместной вероятности, если бы все элементы $x_i \in X$ были независимыми.

Мы можем далее обобщить понятие лифта набора X, рассмотрев все различные способы его разбиения на непустые и непересекающиеся подмножества. Например, предположим, что множество $\{X_1, X_2, \ldots, X_q\}$ является q-разбиением X, т. е. разбиением X на q непустых и

непересекающихся наборов элементов X_i, таких что $X_i \cap X_j = \emptyset$ и $\cup_i X_i = X$. Определим обобщенный лифт X над разбиениями размера q как:

$$lift_q(X) = \min_{X_1 \ldots, X_q} \left\{ \frac{P(X)}{\prod_{i=1}^{q} P(X_i)} \right\}$$

Это наименьшее значение лифта по всем q-разбиениям X. В этом свете $lift(X) = lift_k(X)$, то есть лифт - это значение, полученное из уникального k-разбиения X.

Меры, основанные на правилах

Учитывая набор элементов X, мы можем оценить его, используя меры оценки правил, рассматривая все возможные правила, которые могут быть сгенерированы из X. Пусть Θ - некоторая мера оценки правил. Мы генерируем все возможные правила из X в форме $X_1 \longrightarrow X_2$ и $X_2 \longrightarrow X_1$, где множество $\{X_1, X_2\}$ является 2-разбиением или двудольным разбиением X. Затем мы вычисляем меру Θ для каждого такого правила и используем суммарную статистику, такую как среднее, максимальное и минимальное значение для характеристики X. Если Θ - симметричная мера, то $\Theta(X_1 \longrightarrow X_2) = \Theta(X_2 \longrightarrow X_1)$, и мы должны учитывать только половину правил. Например, если Θ — это повышение по правилу, то мы можем определить среднее, максимальное и минимальное значения подъема для X следующим образом:

$$AvgLift(X) = \underset{X_1, X_2}{avg}\{lift(X_1 \longrightarrow X_2)\}$$
$$MaxLift(X) = \underset{X_1, X_2}{max}\{lift(X_1 \longrightarrow X_2)\}$$
$$MinLift(X) = \underset{X_1, X_2}{min}\{lift(X_1 \longrightarrow X_2)\}$$

Мы также можем сделать то же самое для других мер правил, таких как усиление, достоверность и так далее. В частности, когда мы используем правило лифта, то $MinLift(X)$ идентичен обобщенному лифту $lift_2(X)$ по всем 2-разбиениям X.

7.1.3 Сравнение множественных правил и шаблонов

Теперь обратим наше внимание на сравнение различных правил и шаблонов. В общем, количество часто используемых наборов элементов и правил связывания может быть очень большим, и многие из них могут быть не очень актуальными. Мы выделяем случаи, когда некоторые шаблоны и правила могут быть сокращены, поскольку информация, содержащаяся в них, может быть заменена другими, более важными.

Сравнение наборов элементов

При сравнении нескольких наборов элементов мы можем сосредоточиться на максимальных наборах элементов, удовлетворяющих определенному свойству, или мы можем рассмотреть закрытые наборы элементов, которые захватывают всю информацию о поддержке. Мы рассматриваем эти и другие меры в следующих параграфах.

Максимальные наборы предметов. Часто встречающийся набор элементов X является максимальным, если все его надмножества не являются частыми, то есть X является максимальным, если и только если

$$\sup(\mathrm{X}) \geq \text{ minsup , и для всех } \mathrm{Y} \supset \mathrm{X}, \sup(\mathrm{Y}) < \text{ minsup}$$

Учитывая набор часто встречающихся наборов элементов, мы можем выбрать только максимальные из них, особенно среди тех, которые уже удовлетворяют некоторым другим ограничениям на меры оценки модели, такие как лифт или усиление.

Закрытые наборы предметов и минимальные генераторы Набор элементов X *закрытый*, если все его надмножества имеют строго меньшую поддержку, т. е.

$$sup(X) > \ sup(Y) \text{ для всех } Y \supset X$$

Набор элементов X является минимальным генератором, если все его подмножества имеют строго более высокий уровень поддержки, т. е.

$$sup(X) < \ sup(Y) \text{ для всех } Y \subset X$$

Если набор X не является минимальным генератором, то это означает, что у него есть некоторые избыточные элементы, то есть мы можем найти некоторое подмножество $Y \subset X$, которое можно заменить еще меньшим подмножеством $W \subset Y$, не меняя носителя X , т. е. существует такое $W \subset Y$, что

$$\sup(X) = \sup\big(Y \cup (X \backslash Y)\big) = \sup\big(W \cup (X \backslash Y)\big)$$

Можно показать, что все подмножества минимального генератора сами должны быть минимальными генераторами.

Продуктивные наборы предметов. Набор элементов X является *продуктивным*, если его относительная поддержка выше, чем ожидаемая относительная поддержка по всем его разделам, при условии, что они независимы. Более формально, пусть $|X| \geq 2$, и пусть $\{X_1, X_2\}$ является двудольным X. Мы говорим, что X продуктивно, если

$$rsup(X) > rsup(X_1) \times rsup(X_2), \text{ для всех двудольных } \{X_1, X_2\} \text{ из } X \quad (7.3)$$

Это сразу означает, что X является продуктивным, если его минимальный лифт больше единицы, поскольку

$$MinLift(X) = \min_{X_1, X_2} \left\{ \frac{rsup(X)}{rsup(X_1) \cdot rsup(X_2)} \right\} > 1$$

Что касается усиления, X является продуктивным, если его минимальное усиление нуля, потому что

$$\text{MinLeverage } (X) = \min_{X_1, X_2} \{rsup(X) - rsup(X_1) \times rsup(X_2)\} > 0$$

Сравнение правил

Учитывая два правила $R: X \longrightarrow Y$ и $R': W \longrightarrow Y$, которые имеют один и тот же консеквент, мы говорим, что R *более конкретен*, чем R', или, что эквивалентно, что R' *более общий*, чем R, при условии $W \subset X$.

Неизбыточные правила. Мы говорим, что правило $R: X \longrightarrow Y$ является избыточным, если существует более общее правило $R': W \longrightarrow Y$, которое имеет тот же носитель, то есть $W \subset X$ и $sup(R) = sup(R')$. С другой стороны, если $sup(R) < sup(R')$ по всем его обобщениям R', то R неизбыточно.

Правила улучшения и продуктивности. Определим *улучшение* правила $X \longrightarrow Y$ следующим образом

$$imp(X \longrightarrow Y) = conf(X \longrightarrow Y) - \max_{W \subset X}\{conf(W \longrightarrow Y)\}$$

Улучшение определяет минимальную разницу между достоверностью правила и любого из его обобщений. Правило $R: X \longrightarrow Y$ является продуктивным, если его улучшение больше нуля, что означает, что для всех более общих правил $R': W \longrightarrow Y$ имеем $conf(R) > conf(R')$. С другой стороны, если существует более общее правило R' с $conf(RR') > conf(R)$, то R *непродуктивно*. Если правило является избыточным, оно также непродуктивно, поскольку его улучшение равно нулю.

Чем меньше улучшение правила $R: X \longrightarrow Y$, тем больше вероятность, что оно будет непродуктивным. Мы можем обобщить это понятие, чтобы рассмотреть правила, которые имеют хотя бы некоторый минимальный уровень улучшения, то есть мы можем потребовать, чтобы $imp(X \longrightarrow Y) \geq \mathrm{t}$, где t - определяемый пользователем минимальный порог улучшения.

7.2 Проверка значимости и доверительные интервалы

Теперь рассмотрим, как получить оценку статистической значимости шаблонов и правил, а также как вычислить доверительные интервалы для этой оценки.

7.2.1 Точный тест Фишера для продуктивных правил

Начнём с точного теста Фишера для улучшения правил. То есть, мы напрямую проверим, является ли правило $R: X \to Y$ продуктивным, сравнив его доверительность с доверительностью каждого из его обобщений: $R': W \to Y$, в том числе с тривиальным правилом $\emptyset \to Y$.

Пусть $R: X \to Y$ – ассоциативное правило. Рассмотрим его обобщение $R': W \to Y$, где $W = X \backslash Z$ – новый антецедент, полученный удалением из X подмножества $Z \subseteq X$. Имея исходный датасет **D**, при условии наличия W, мы можем сформировать таблицу сопряжённости размера 2×2 между Z и консеквентом Y, как показано в таблице 7.2. Значения в ячейках таблицы формируются следующим образом:

$$a = sup(WZY) = \sup(XY) \qquad b = sup(WZ\neg Y) = sup(X\neg Y)$$

$$c = sup(W\neg XY) \qquad d = sup(W\neg Z\neg Y)$$

Таблица 7.2. Таблица сопряжённости для Z и Y при условии $W = X \backslash Z$.

W	Y	$\neg Y$	
Z	a	b	$a + b$
$\neg Z$	c	d	$c + d$

	$a + c$	$b + d$	$n = sup(W)$

Здесь a обозначает количество транзакций, которые содержат и X, и Y, b обозначает количество транзакций, которые содержат X, но не содержат Y, c обозначает количество транзакций с W и Y, но без Z, и, наконец, d обозначает количество транзакций, содержащих W, но без Z и Y. Предельные значения заданы следующим образом:

пределы рядов: $a + b = sup(WZ) = sup(X), \quad c + d = sup(W \neg Z)$

пределы столбцов: $a + c = sup(WY), \quad b + d = sup(W \neg Y)$

где пределы рядов задают частоту появления W с и без Z, а пределы рядов определяют количество вхождений W с и без Y. Наконец, мы можем видеть, что сумма всех ячеек – просто $n = a + b + c + d = sup(W)$. Заметим, что при $Z = X$ мы получим $W = \emptyset$, а таблица сопряжённости выродится в 7.2.

Имея таблицу сопряжённости при условии W, нам интересно получить отношение шансов, полученное сравнением наличия и отсутствия Z, то есть:

$$oddratio = \frac{a/(a+b)}{b/(a+b)} / \frac{c/(c+d)}{d/(c+d)} = \frac{ad}{bc}. \tag{7.4}$$

Напомним, что отношение шансов измеряет шансы X, то есть, W и Z, встречающиеся с Y, против подмножества W без Z, встречающиеся с Y. При нулевой гипотезе H_0 (в предположении, что Z и Y независимы при условии W) отношение шансов равно единице. Это легко видеть, так как в предположении независимости счётчик в ячейке таблицы сопряжённости равен произведению предельных значений соответствующего ряда и столбца, делённых на n, то есть, при H_0:

$$a = (a + b)(a + c)/n \qquad b = (a + b)(b + d)/n$$

$$c = (c + d)(a + c)/n \qquad d = (c + d)(b + d)/n$$

Подставляя эти значения в ур-е (7.4), получим:

$$oddsratio = \frac{ad}{bc} = \frac{(a+b)(c+d)(b+d)(a+c)}{(a+c)(b+d)(a+b)(c+d)}.$$

Таким образом, нулевая гипотеза соответствует $H_0: oddsratio = 1$, а альтернативная – $H_a: oddsratio > 1$. Если при нулевой гипотезе мы сделаем дополнительное предположение, зафиксировав пределы рядов и столбцов, тогда a однозначно определяет значения b, c и d, а функция вероятности для наблюдения a в таблице сопряжённости задаётся гипергеометрическим распределением. Напомним, что гипергеометрическое распределение

задаёт вероятность выбрать s успехов в t испытаниях, если мы делаем выборку *без замены* из конечного набора размера T, в котором всего S успехов:

$$P(s|t,S,T) = \binom{S}{s} \cdot \binom{T-S}{t-s} / \binom{T}{t}.$$

В нашем случае мы трактуем появление Z как успех. Размер выборки – $T = sup(W) = n$, поскольку мы предполагаем, что W встречается всегда, а общее количество успехов – это поддержка Z при условии W, то есть, $S = a + b$. В $t = a + c$ испытаниях гипергеометрическое распределение даёт вероятность $s = a$ успехов:

$$P(a|(a+c),(a+b),n) = \frac{\binom{a+b}{a} \cdot \binom{n-(a+b)}{(a+c)-a}}{\binom{n}{a+c}} = \frac{\binom{a+b}{a} \cdot \binom{c+d}{c}}{\binom{n}{a+c}}$$

$$= \frac{(a+b)!\,(c+d)!}{a!\,b!\,c!\,d!} \Big/ \frac{n!}{(a+c)!\,\big(n-(a+c)\big)!}$$

$$= \frac{(a+b)!\,(c+d)!\,(a+c)!\,(b+d)!}{n!\,a!\,b!\,c!\,d!}. \qquad (7.5)$$

Наша цель – сопоставить нулевую гипотезу H_0 о том, что $oddsratio = 1$, с альтернативной гипотезой H_a: $oddsratio > 1$. Поскольку a однозначно определяет остальные ячейки при фиксированных предельных значениях рядов и столбцов, из ур-я (7.4) мы видим, что чем больше a, тем больше отношение шансов, а следовательно, и больше доказательств для H_a. Мы можем получить p-значение для таблицы сопряжённости так же, как в таблице 7.2, суммируя все члены ур-я (7.5) с возможными значениями a и больше:

$$p - value(a) = \sum_{i=0}^{\min(b,c)} P(a+i|(a+c),(a+b),n)$$

$$= \sum_{i=0}^{\min(b,c)} \frac{(a+b)!\,(c+d)!\,(a+c)!\,(b+d)!}{n!\,(a+i)!\,(b-i)!\,(c-i)!\,(d+i)!},$$

что следует из того факта, что, увеличивая количество a на i, мы, поскольку предельные значения рядов и столбцов фиксированы, должны уменьшить b и c на i, а d – увеличить на i, как показано в таблице 7.1. Чем меньше p-значение, тем выше уверенность в том, что отношение шансов выше единицы, и, таким образом, мы можем отклонить нулевую гипотезу H_0, если $p - value \leq \alpha$, где α – порог значимости (например, $\alpha = 0.01$). Этот тест известен как *точный тест Фишера*.

Таким образом, чтобы проверить, продуктивно ли правило $R: X \to Y$, нам необходимо вычислить $p - value(a) = p - value(sup(XY))$ таблицы сопряжённости, полученной из каждого обобщения правила: $R': W \to Y$, где $W = X \backslash Z$ для $Z \subseteq X$. Если $p - value\big(sup(XY)\big) > \alpha$ для хотя бы одного такого сравнения, тогда мы можем отклонить правило $R: X \to Y$ как непродуктивное. С другой стороны, если $p - value\big(sup(XY)\big) \leq \alpha$ для всех обобщений, тогда R продуктивное. Заметим, однако, что при $|X| = k$ существует $2^k - 1$ обобщений, и, чтобы избежать экспоненциальной сложности для больших антецедентов, мы, как правило,

ограничиваемся только непосредственными обобщениями в следующей форме: $R' : X \backslash z \rightarrow Y$, где $z \in X$ – одно из значений атрибута а антецеденте. Однако мы включаем тривиальное правило $\emptyset \rightarrow Y$, поскольку условная вероятность $P(Y|X) = conf(X \rightarrow Y)$ должна быть выше априорной вероятности $P(Y) = conf(\emptyset \rightarrow Y)$.

Проверка множества гипотез

Имея исходный датасет **D**, мы можем получить экспоненциально большое количество правил, которые необходимо проверить на продуктивность. Так мы попадаем в проблему проверки множества гипотез, то есть, просто из-за большого количества гипотез некоторое количество непродуктивных правил пройдёт порог $p - value \leq \alpha$ со случайной вероятностью. Способ избежать этого – использовать поправку Бонферрони для уровня значимости, которая явно берёт в рассмотрение количество экспериментов, выполненных во время проверки гипотез. Вместо использования заданного порога α необходимо использовать скорректированный порог $\alpha' = \frac{\alpha}{\#r}$, где $\#r$ – количество (или его оценка) правил, которые необходимо протестировать. Такая поправка гарантирует, что частота обнаружения ложных правил ограничена α, где ложное обнаружение означает утверждение, что правило продуктивно, хотя это не так.

7.2.2 Проверка значимости перестановками

Проверка *перестановками или рандомизацией* позволяет определить распределение заданной статистики Θ путём многократных случайных изменений наблюдаемых данных, чтобы получить случайное количество датасетов, которые затем можно использовать для проверки значимости. В контексте оценки шаблонов, имея датасет **D**, мы генерируем k случайных перемешанных датасетов $\mathbf{D_1}, \mathbf{D_2}, \ldots, \mathbf{D_k}$. После этого мы можем произвести различные проверки значимости. Например, для заданного паттерна или правила мы можем проверить, являются ли они статистически значимыми, вычислив эмпирическую функцию распределения для тестовой статистики Θ путём вычисления её значения Θ_i в i-том датасете $\mathbf{D}_i$ для всех $i \in [1, k]$. Для этих значений мы можем сгенерировать выборочную функцию распределения:

$$\hat{F}(x) = \hat{P}(\Theta \leq x) = \frac{1}{k} \sum_{i=1}^{k} I(\Theta_i \leq x),$$

где I – индикаторная функция, принимающая значение 1, когда её аргумент – истина, и 0 в обратном случае. Пусть Θ – значение тестовой статистики для входного датасета **D**, тогда $p - value(\Theta)$, то есть, вероятность получить значение выше Θ, может быть вычислено так:

$$p - value(\Theta) = 1 - F(\Theta).$$

Имея заданный уровень значимости α, при значении $p - value(\Theta) > \alpha$ мы принимаем нулевую гипотезу о том, что правило/паттерн не являются статистически значимыми. С другой стороны, при $p - value(\Theta) \leq \alpha$ мы можем отклонить нулевую гипотезу и заключить, что паттерн является значимым, поскольку значение выше Θ крайне маловероятно. Проверка перестановками может быть применена для оценки всего набора правил и паттернов. Например, мы можем проверить список часто встречающихся наборов объектов, сравнивая число часто встречающихся наборов объектов в **D** с распределением числа часто

встречающихся наборов объектов, полученных опытным путём из перемешанных датасетов $\mathbf{D}_i$. Мы можем также провести этот анализ как функцию $minsup$, и так далее.

Случайный обмен

Ключевой вопрос в генерации перемешанных датасетов $\mathbf{D}_i$ – какие характеристики входного датасета $\mathbf{D}$ нам необходимо сохранить. Подход случайного обмена сохраняет неизменными пределы столбцов и рядов заданного датасета, то есть, перемешанные датасеты сохраняют поддержку каждого объекта (предел столбцов), а также число объектов в каждой транзакции (предел ряда). Имея датасет $\mathbf{D}_i$, мы случайным образом создаём k датасетов с теми же значениями пределов столбцов и рядов. Затем мы ищем часто встречающиеся паттерны в $\mathbf{D}$ и проверяем, отличается ли статистика шаблонов от полученных в перемешанном датасете. Если разница не значительна, мы можем заключить, что паттерны возникают исключительно из пределов ряда и столбца, а не из каких-либо интересных свойств данных.

Для заданной двоичной матрицы $\mathbf{D} \subseteq \mathcal{T} \times \mathcal{I}$ метод случайного обмена предлагает обменять две ненулевые ячейки матрицы посредством *перестановки*, которая оставляет пределы ряда и столбца неизменными. Чтобы показать, как работает такая перестановка, рассмотрим две любые транзакции $t_a, t_b \in \mathcal{T}$ и два объекта $i_a, i_b \in \mathcal{I}$ такие, что $(t_a, i_a), (t_b, i_b) \in \mathbf{D}$ и $(t_a, i_b), (t_b, i_a) \notin \mathbf{D}$, что соответствует подматрице $\mathbf{D}$ размера 2×2:

$$\mathbf{D}(t_a, i_a; t_b, i_b) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

После операции перестановки мы получим новую подматрицу:

$$\mathbf{D}(t_a, i_b; t_b, i_a) = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix},$$

где мы обменяли элементы из $\mathbf{D}$ так, что $(t_a, i_b), (t_b, i_a) \in \mathbf{D}$ и $(t_a, i_a), (t_b, i_b) \notin \mathbf{D}$. Мы будем обозначать эту операцию следующим образом: $Swap(t_a, i_a; t_b, i_b)$. Отметим, что перестановка не изменяет пределы ряда и столбца, и таким образом мы можем сгенерировать перемешанный датасет с теми же суммами рядов и столбцов, что и в $\mathbf{D}$, последовательностью перестановок. Алгоритм 7.1 представляет собой псевдокод для генерации датасетов, перемешанных случайным обменом. Алгоритм выполняет i перестановок, выбирая две пары $(t_a, i_a), (t_b, i_b) \in \mathbf{D}$ случайным образом; перестановка успешна, только если $(t_a, i_b), (t_b, i_a) \notin \mathbf{D}$.

Алгоритм 7.1: Генерация датасета случайного обмена.

```
SwapRandomization(t, D ⊆ T × I):
while t > 0 do
    Select pairs (t_a, i_a), (t_b, i_b) ∈ D randomly
    if (t_a, i_b) ∉ D and (t_b, i_a) ∉ D then
        D ← D \ {(t_a, i_a), (t_b, i_b)} ∪ {(t_a, i_b), (t_b, i_a)}
    t = t − 1
return D
```

Глава 8. Репрезентативная кластеризация

Для набора данных с n точками в d-мерном пространстве, $\mathbf{D} = \{\mathbf{x}_i\}_{i=1}^n$, и количеств желаемых кластеров k, цель репрезентативной кластеризации состоит в том, чтобы разделить набор данных на k групп или кластеров, обозначающихся как $C = \{C_1, C_2, \dots, C_k\}$. Далее, для каждого кластера C_i существует репрезентативная точка, которая суммирует кластер. Обычно выбирается среднее значение (также называемое центроидом) $\boldsymbol{\mu}_i$ всех точек в кластере, т. е.,

$$\boldsymbol{\mu}_i = \frac{1}{\boldsymbol{n}_i} \sum_{x_j \in C_i} \mathbf{x}_j ,$$

где $\boldsymbol{n}_i = |C_i|$ – количество точек в кластере C_i.

Полный перебор или исчерпывающий алгоритм поиска хорошей кластеризации состоит в том, чтобы просто сгенерировать все возможные разбиения n точек на k кластеров, произвести оценку оптимизации для каждого из них и сохранить кластеризацию, которая дает лучший результат. Точное количество способов разбить n точек на k непустых и непересекающихся частей можно получить с помощью чисел Стирлинга второго рода, заданных как:

$$S(n,k) = \frac{1}{k!}\sum_{t=0}^{k}(-1)^t \binom{k}{t}(k-t)^n$$

Неформально каждую точку можно назначить любому из k кластеров, поэтому существует не более k^n возможных кластеров. Однако любая перестановка k кластеров в рамках данной кластеризации дает эквивалентную кластеризацию; следовательно, имеется $O(k^n/k!)$ кластеризаций n точек в k групп. Понятно, что исчерпывающее перечисление и оценка всех возможных кластеров практически неосуществимы. В этой главе мы описываем два подхода к репрезентативной кластеризации, а именно алгоритм K-средних и EM-алгоритм.

8.1 Алгоритм K-средних

Для кластеризации $C = \{C_1, C_2, \dots, C_k\}$, нам нужна некоторая функция оценки, которая оценивает её качество. Эта функция оценки суммы квадратов ошибок определяется как:

$$SSE(C) = \sum_{i=1}^{k} \sum_{\mathbf{x}_j \in C_i} \left\|\mathbf{x}_j - \boldsymbol{\mu}_i\right\|^2 \tag{8.1}$$

Цель состоит в том, чтобы найти кластеризацию, которая минимизирует оценку SSE:

$$C^* = \arg\min_C \{SSE(C)\}$$

K-средних использует жадный итерационный подход для поиска кластеризации, которая минимизирует цель SSE [уравнение. (8.1)]. Таким образом, он может сходиться к локальным оптимумам вместо глобально оптимальной кластеризации.

K-средних инициализирует средние кластеров путем случайной генерации k точек в пространстве данных. Обычно это делается путем генерации равномерной случайной величины в пределах диапазона для каждого измерения. Каждая итерация K-средних состоит

из двух шагов: (1) назначение кластера и (2) обновление центроида. Для k средних кластера на этапе назначения кластера каждой точке $\mathbf{x}_j \in \mathbf{D}$ присваивается ближайшее среднее значение, что вызывает кластеризацию, причем каждый кластер C_i содержит точки, которые ближе к $\boldsymbol{\mu}_i$, чем любое другое среднее значение кластера. То есть каждая точка $\mathbf{x}_j$ относится к кластеру C_{j^*}, где

$$j^* = \arg \min_{i=0}^{k} \left\{ \left\| \mathbf{x}_j - \boldsymbol{\mu}_i \right\|^2 \right\} \tag{8.2}$$

Для набора кластеров $C = \{C_1, C_2, \ldots, C_k\}$, на этапе обновления центроида, новые средние значения вычисляются для каждого кластера из точек в C_i. Этапы назначения кластера и обновления центроида выполняются итеративно, пока мы не достигнем фиксированной точки или локальных минимумов. С практической точки зрения можно предположить, что K-средние сошлись, если центроиды не меняются от одной итерации к другой. Например, мы можем остановиться, если $\sum_{i=1}^{k} \left\| \boldsymbol{\mu}_i^t - \boldsymbol{\mu}_i^{t-1} \right\|^2 \leq \epsilon$, где $\epsilon > 0$ – порог сходимости, t обозначает текущую итерацию, и $\boldsymbol{\mu}_i^t$ – среднее для кластера C_i в итерации t.

Поскольку метод K-средних начинается со случайного предположения для начальных центроидов, K-средних обычно запускается несколько раз, и для отчета окончательной кластеризации выбирается запуск с наименьшим значением SSE. Также стоит отметить, что K-средних генерирует кластеры выпуклой формы, поскольку область в пространстве данных, соответствующем каждому кластеру, может быть получена как пересечение полупространств, возникающих из гиперплоскостей, которые делят пополам и перпендикулярны линейным сегментам, которые соединяют пары центроидов кластера.

С точки зрения вычислительной сложности K-средних, мы можем видеть, что этап назначения кластера занимает $O(nkd)$ времени, потому что для каждой из n точек мы должны вычислить расстояние до каждого из k кластеров, что требует d операций в d размерностях. Шаг повторного вычисления центроида занимает $O(nd)$ времени, потому что мы должны добавить в общей сложности n d-мерных точек. Предполагая, что существует t итераций, общее время для K-средних задается как $O(nkd)$. С точки зрения затрат ввода-вывода для этого требуется $O(t)$ полных сканирований базы данных, потому что мы должны читать всю базу данных на каждой итерации.

Пример 8.1. Рассмотрим одномерные данные, показанные на рисунке 8.1a. Предположим, что мы хотим кластеризовать данные в k = 2 кластера. Пусть начальные центроиды равны $\mu_1 = 2$ и $\mu_2 = 4$. На первой итерации мы сначала вычисляем кластеры, присваивая каждой точке ближайшее среднее значение, чтобы получить

$$C_1 = \{2,3\}, C_2 = \{4,10,11,12,20,25,30\}$$

затем мы обновляем средние следующим образом:

$$\mu_1 = \frac{2+3}{2} = \frac{5}{2} = 2.5$$

$$\mu_2 = \frac{4+10+11+12+20+25+30}{7} = \frac{112}{7} = 16$$

Новые центроиды и кластеры после первой итерации показаны на рисунке 8.1b. Для второго шага мы повторяем шаги назначения кластера и обновления центроида, как показано на рисунке 8.1c, чтобы получить новые кластеры:

$$C_1 = \{2,3,4\}, C_2 = \{10,11,12,20,25,30\}$$

и новые средние

$$\mu_1 = \frac{2+3+4}{3} = \frac{9}{3} = 3$$

$$\mu_2 = \frac{10+11+12+20+25+30}{6} = \frac{108}{6} = 18$$

Полный процесс до схождения показан на рисунке 8.1. Конечные кластеры представлены как

$$C_1 = \{2,3,4,11,12\}, C_2 = \{20,25,30\}$$

с представителями $\mu_1 = 7$ и $\mu_2 = 25$.

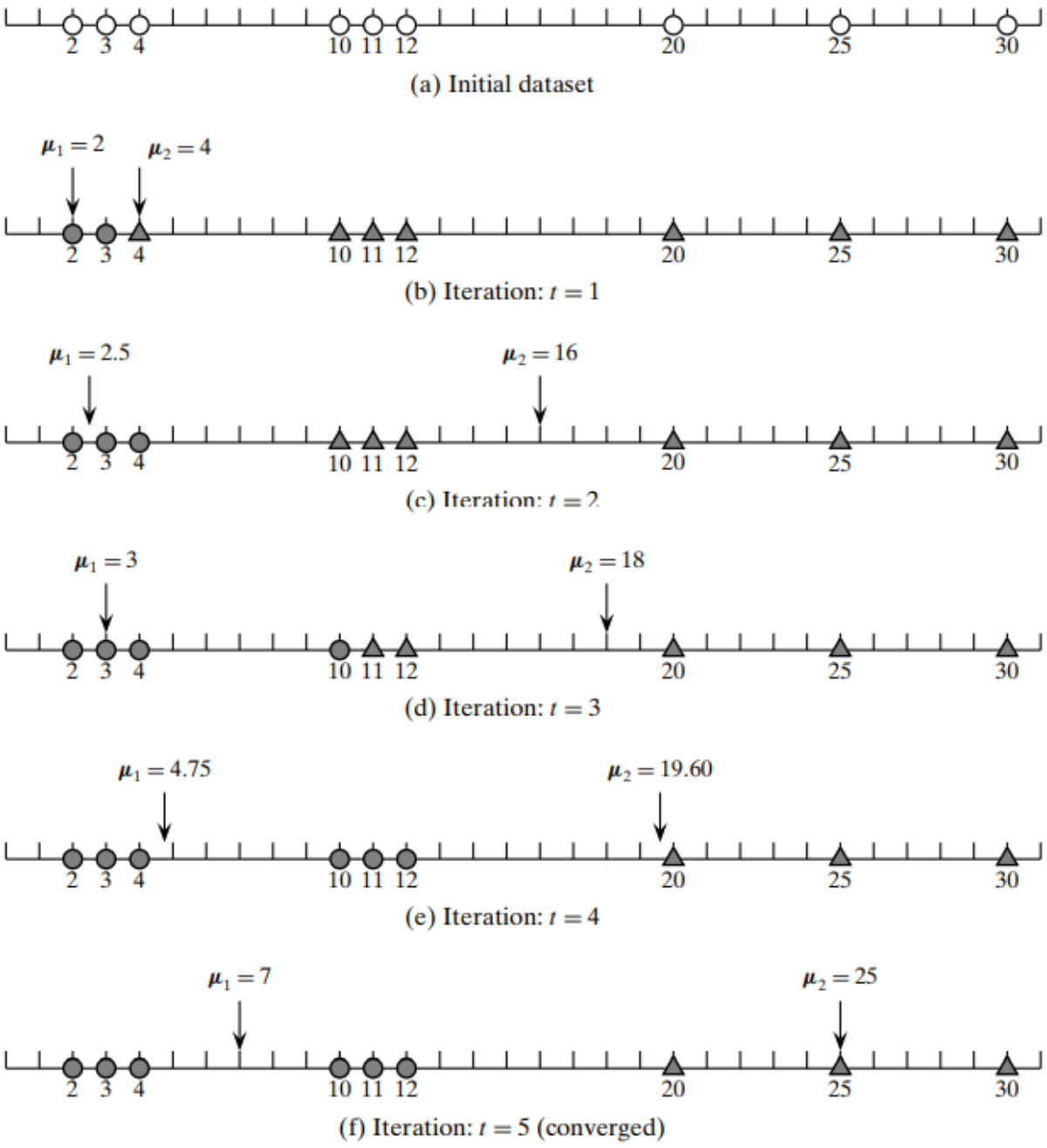


Рисунок 8.1 К-средние в одном измерении

Алгоритм 8.1: К-средних

K-MEANS $(\mathbf{D}, k, \epsilon)$:

$t = 0$

Randomly initialize k centroids: $\boldsymbol{\mu}_1^t, \boldsymbol{\mu}_2^t, \ldots, \boldsymbol{\mu}_k^t \in \mathbb{R}^d$

repeat

  $t \leftarrow t + 1$

  $C_j \leftarrow \emptyset$ for all $j = 1, \cdots, k$

  // Cluster Assignment Step

  foreach $\mathbf{x}_j \in \mathbf{D}$ **do**

    $j^* \leftarrow \arg\min_i \left\{ \left\| \mathbf{x}_j - \boldsymbol{\mu}_i^{t-1} \right\|^2 \right\}$ // Assign $\mathbf{x}_j$ to closest centroid

    $C_{j^*} \leftarrow C_{j^*} \cup \{\mathbf{x}_j\}$

  // Centroid Update Step

  foreach $i = 1$ *to* k **do**

    $\boldsymbol{\mu}_i^t \leftarrow \frac{1}{|C_i|} \sum_{\mathbf{x}_j \in C_i} \mathbf{x}_j$

until $\sum_{i=1}^{k} \left\| \boldsymbol{\mu}_i^t - \boldsymbol{\mu}_i^{t-1} \right\|^2 \leq \epsilon$

Пример 8.2(K-средних в двух измерениях). На рисунке 8.2 мы проиллюстрировали алгоритм K-средних для набора данных Iris, используя первые два основных компонента в качестве двух измерений. Ирис имеет n = 150 точек, и мы хотим найти k = 3 кластера, соответствующих трем типам ирисов. Случайная инициализация кластера дает

$$\boldsymbol{\mu}_1 = (-0.98, -1.24)^T, \boldsymbol{\mu}_2 = (-2.96, 1.16)^T, \boldsymbol{\mu}_3 = (-2.96, 1.16)^T$$

как показано на рисунке 8.2a. С этими начальными кластерами K-средних требуется восемь итераций для схождения. На рисунке 8.2b показаны кластеры и их средние значения после одной итерации:

$$\boldsymbol{\mu}_1 = (1.56, -0.08)^T, \boldsymbol{\mu}_2 = (-2.86, 0.53)^T, \boldsymbol{\mu}_3 = (-1.50, -0.05)^T$$

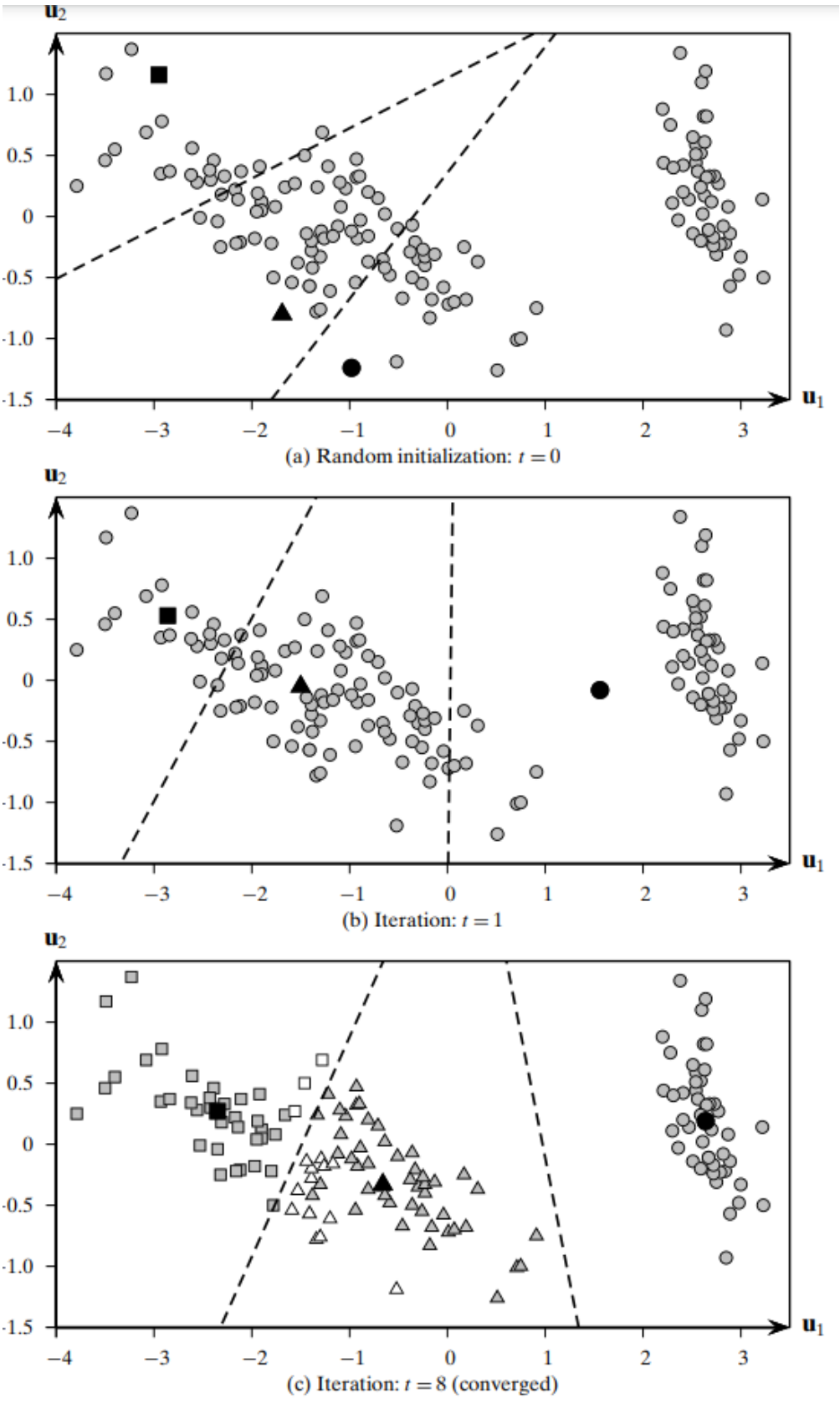


Рисунок 8.2. К-средние в двух измерениях

На рис. 8.2c показаны кластеры при сходимости. Окончательные средние следующие:

$$\boldsymbol{\mu}_1 = (2.64,0.19)^T, \boldsymbol{\mu}_2 = (-2.35,0.27)^T, \boldsymbol{\mu}_3 = (-0.66,-0.33)^T$$

На рисунке 8.2 средние значения кластера показаны черными точками, также показаны выпуклые области пространства данных, соответствующие каждому из трех кластеров. Пунктирные линии (гиперплоскости) - это серединные перпендикулярные отрезки прямых, соединяющих два центра кластеров. Получившееся выпуклое разбиение точек составляет кластеризацию.

На рисунке 8.2c показаны последние три кластера: C_1 в виде окружностей, C_2 в виде квадратов и C_3 в виде треугольников. Белые точки указывают на неправильную группировку по сравнению с известными типами ирисов. Таким образом, мы видим, что C_1 идеально соответствует iris-setosa, а большинство точек C_2 соответствует iris-virginica, а C_3 - iris-versicolor. Например, три точки (белые квадраты) типа iris-versicolor ошибочно сгруппированы вC_2, а 14 точек от iris-virginica ошибочно сгруппированы в C_3 (белые треугольники). Конечно, поскольку метка класса Iris не используется в кластеризации, разумно ожидать, что мы не получим идеальной кластеризации.

8.2 Ядерный алгоритм K-средних

В K-средних разделяющая граница между кластерами линейна. Ядерный K-средних позволяет извлекать нелинейные границы между кластерами с помощью хитрости с ядром, описанной в главе 5. Таким образом, метод может быть использован для обнаружения невыпуклых кластеров.

В ядерном K-средних основная идея состоит в концептуальном отображении точки данных $\mathbf{x}_i$ во входном пространстве в точку $\phi(\mathbf{x}_i)$ в некотором многомерном пространстве признаков посредством соответствующего нелинейного отображения ϕ. Однако трюк с ядром позволяет нам выполнять кластеризацию в пространстве признаков исключительно в терминах функции ядра $K(\mathbf{x}_i, \mathbf{x}_j)$, которая может быть вычислена во входном пространстве, но соответствует скалярному (или внутреннему) произведению $\phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j)$ в пространстве признаков.

Предположим пока, что все точки $\mathbf{x}_i \in \mathbf{D}$ сопоставлены с соответствующими отображениями $\phi(\mathbf{x}_i)$ в пространстве признаков. Пусть $\mathbf{K} = \{K(\mathbf{x}_i, \mathbf{x}_j)\}_{i,j=1,\ldots,n}$ обозначает симметричную ядерную матрицу $n \times n$, где $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j)$. Пусть $\{C_1, \ldots, C_k\}$ задает разбиение n точек на k кластеров, и пусть соответствующие средние значения кластеров в пространстве признаков задаются как $\{\boldsymbol{\mu}_i^{\phi}, \ldots, \boldsymbol{\mu}_k^{\phi}\}$, где

$$\boldsymbol{\mu}_i^{\phi} = \frac{1}{n_i}\sum_{\mathbf{x}_j \in C_i}\phi(\mathbf{x}_j)$$

среднее кластера C_i в пространстве признаков и $n_i = |C_i|$.

В пространстве признаков ядерная K-средних сумма квадратов отклонений может быть записана как

$$\min_C SSE(C) = \sum_{i=1}^{k} \sum_{\mathbf{x}_j \in C_i} \left\| \phi(\mathbf{x}_j) - \boldsymbol{\mu}_i^{\phi} \right\|^2$$

Расширяя ядерную SSE в терминах функции ядра, мы получаем

$$SSE(C) = \sum_{i=1}^{k} \sum_{\mathbf{x}_j \in C_i} \left\| \phi(\mathbf{x}_j) - \boldsymbol{\mu}_i^{\phi} \right\|^2$$

$$= \sum_{i=1}^{k} \sum_{\mathbf{x}_j \in C_i} \left\| \phi(\mathbf{x}_j) \right\|^2 - 2\phi(\mathbf{x}_j)^T \boldsymbol{\mu}_i^{\phi} + \left\| \boldsymbol{\mu}_i^{\phi} \right\|^2$$

$$= \sum_{i=1}^{k} \left(\left(\sum_{\mathbf{x}_j \in C_i} \left\| \phi(\mathbf{x}_j) \right\|^2 \right) - 2n_i \left(\frac{1}{n_i} \sum_{\mathbf{x}_j \in C_i} \phi(\mathbf{x}_j) \right)^T \boldsymbol{\mu}_i^{\phi} + n_i \left\| \boldsymbol{\mu}_i^{\phi} \right\|^2 \right)$$

$$= \left(\sum_{i=1}^{k} \sum_{\mathbf{x}_j \in C_i} \phi(\mathbf{x}_j)^T \phi(\mathbf{x}_j) \right) - \left(\sum_{i=1}^{k} n_i \left\| \boldsymbol{\mu}_i^{\phi} \right\|^2 \right)$$

$$= \sum_{i=1}^{k} \sum_{\mathbf{x}_j \in C_i} K(\mathbf{x}_j, \mathbf{x}_j) - \sum_{i=1}^{k} \frac{1}{n_i} \sum_{\mathbf{x}_a \in C_i} \sum_{\mathbf{x}_b \in C_i} K(\mathbf{x}_a, \mathbf{x}_b)$$

$$= \sum_{j=1}^{n} K(\mathbf{x}_j, \mathbf{x}_j) - \sum_{i=1}^{k} \frac{1}{n_i} \sum_{\mathbf{x}_a \in C_i} \sum_{\mathbf{x}_b \in C_i} K(\mathbf{x}_a, \mathbf{x}_b) \tag{8.3}$$

Таким образом, ядерная K-средних SSE целевая функция может быть выражена исключительно в терминах функции ядра. Как и K-средних, для минимизации целевой функции SSE мы применяем жадный итеративный подход. Основная идея состоит в том, чтобы присвоить каждой точке ближайшее среднее значение в пространстве признаков, что приведет к новой кластеризации, которая, в свою очередь, может быть использована для получения новых оценок для средних значений кластера. Однако основная трудность заключается в том, что мы не можем явно вычислить среднее значение каждого кластера в пространстве признаков. К счастью, явное получение средних кластера не требуется; все операции могут быть выполнены в терминах функции ядра $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$.

Рассмотрим расстояние от точки $\phi(\mathbf{x}_j)$ до среднего значения $\boldsymbol{\mu}_i^{\phi}$ в пространстве признаков, которое можно вычислить как

$$\left\| \phi(\mathbf{x}_j) - \boldsymbol{\mu}_i^{\phi} \right\|^2 = \left\| \phi(\mathbf{x}_j) \right\|^2 - 2\phi(\mathbf{x}_j)^T \boldsymbol{\mu}_i^{\phi} + \left\| \boldsymbol{\mu}_i^{\phi} \right\|^2$$

$$= \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j) - \frac{2}{n_i} \sum_{\mathbf{x}_a \in C_i} \phi(\mathbf{x}_j)^T \phi(\mathbf{x}_a) + \frac{1}{n_i^2} \sum_{\mathbf{x}_a \in C_i} \sum_{\mathbf{x}_b \in C_i} \phi(\mathbf{x}_a)^T \phi(\mathbf{x}_b)$$

$$= K(\mathbf{x}_j, \mathbf{x}_j) - \frac{2}{n_i} \sum_{\mathbf{x}_a \in C_i} K(\mathbf{x}_a, \mathbf{x}_j) + \frac{1}{n_i^2} \sum_{\mathbf{x}_a \in C_i} \sum_{\mathbf{x}_b \in C_i} K(\mathbf{x}_a, \mathbf{x}_b) \tag{8.4}$$

Таким образом, расстояние от точки до среднего значения кластера в пространстве признаков можно вычислить, используя только ядерные операции. На этапе назначения кластера ядерного K-средних мы присваиваем точку ближайшему среднему кластеру следующим образом:

$$C^*(\mathbf{x}_j) = \arg\min_i \left\{ \left\| \phi(\mathbf{x}_j) - \boldsymbol{\mu}_i^{\phi} \right\|^2 \right\}$$

$$= \arg\min_i \left\{ K(\mathbf{x}_j, \mathbf{x}_j) - \frac{2}{n_i} \sum_{\mathbf{x}_a \in C_i} K(\mathbf{x}_a, \mathbf{x}_j) + \frac{1}{n_i^2} \sum_{\mathbf{x}_a \in C_i} \sum_{\mathbf{x}_b \in C_i} K(\mathbf{x}_a, \mathbf{x}_b) \right\}$$

$$= \arg\min_i \left\{ \frac{1}{n_i^2} \sum_{\mathbf{x}_a \in C_i} \sum_{\mathbf{x}_b \in C_i} K(\mathbf{x}_a, \mathbf{x}_b) - \frac{2}{n_i} \sum_{\mathbf{x}_a \in C_i} K(\mathbf{x}_a, \mathbf{x}_j) \right\} \tag{8.5}$$

где мы опускаем член $K(\mathbf{x}_j, \mathbf{x}_j)$, потому что он остается одинаковым для всех k кластеров и не влияет на решение о назначении кластера. Также обратите внимание, что первый член – это просто среднее попарное ядерное значение для кластера C_i, которое не зависит от точки $\mathbf{x}_j$. Фактически, это квадрат нормы кластерного среднего в пространстве признаков. Второй член в два раза больше среднего значения ядра для точек в C_i относительно $\mathbf{x}_j$.

Алгоритм 8.2 показывает псевдокод для метода ядерного K-средних. Он начинается с первоначального случайного разбиения точек на k кластеров. Затем он итеративно обновляет назначения кластера, переназначая каждую точку ближайшему среднему значению в пространстве признаков с помощью уравнения (8.5). Чтобы облегчить вычисление расстояния, он сначала вычисляет среднее значение ядра, то есть квадрат нормы среднего значения кластера, для каждого кластера (для цикла в строке 5). Затем он вычисляет среднее значение ядра для каждой точки $\mathbf{x}_j$ с точками в кластере C_i (для цикла в строке 7). На этапе назначения основного кластера эти значения используются для вычисления расстояния xj от каждого из кластеров C_i и $\mathbf{x}_j$ присваиваются ближайшему среднему значению. Эта информация о переназначении используется для повторного разделения точек на новый набор кластеров. То есть все точки $\mathbf{x}_j$, которые ближе к среднему значению C_i, составляют новый кластер для следующей итерации. Этот итерационный процесс повторяется до сходимости.

Алгоритм 8.2: Ядерный К-средних

KERNEL-KMEANS($\mathbf{K}, k, \epsilon$):

$t \leftarrow 0$

$\mathcal{C}^t \leftarrow \{C_1^t, \ldots, C_k^t\}$// Randomly partition points into k clusters

repeat

  $t \leftarrow t+1$

  foreach $C_i \in \mathcal{C}^{t-1}$ **do** // Compute squared norm of cluster means

    $\text{sqnorm}_i \leftarrow \frac{1}{n_i^2} \sum_{\mathbf{x}_a \in C_i} \sum_{\mathbf{x}_b \in C_i} K(\mathbf{x}_a, \mathbf{x}_b)$

  foreach $\mathbf{x}_j \in \mathbf{D}$ **do** // Average kernel value for $\mathbf{x}_j$ and C_i

    foreach $C_i \in \mathcal{C}^{t-1}$ **do**

      $\text{avg}_{ji} \leftarrow \frac{1}{n_i} \sum_{\mathbf{x}_a \in C_i} K(\mathbf{x}_a, \mathbf{x}_j)$

  // Find closest cluster for each point

  foreach $\mathbf{x}_j \in \mathbf{D}$ **do**

    foreach $C_i \in \mathcal{C}^{t-1}$ **do**

      $d(\mathbf{x}_j, C_i) \leftarrow \text{sqnorm}_i - 2 \cdot \text{avg}_{ji}$

    $j^* \leftarrow \arg\min_i \{d(\mathbf{x}_j, C_i)\}$

    $C_{j^*}^t \leftarrow C_{j^*}^t \cup \{\mathbf{x}_j\}$ // Cluster reassignment

  $\mathcal{C}^t \leftarrow \{C_1^t, \ldots, C_k^t\}$

until $1 - \frac{1}{n} \sum_{i=1}^k \left| C_i^t \cap C_i^{t-1} \right| \leq \epsilon$

Для проверки сходимости мы проверяем, есть ли какие-либо изменения в кластерных назначениях точек. Количество точек, не меняющих кластеры, задается как сумма $\sum_{i=1}^k \left| C_i^t \cap C_i^{t-1} \right|$, где t задает текущую итерацию. Доля точек, переназначенных другому кластеру в текущей итерации, задается как

$$\frac{n - \sum_{i=1}^k \left| C_i^t \cap C_i^{t-1} \right|}{n} = 1 - \frac{1}{n} \sum_{i=1}^k \left| C_i^t \cap C_i^{t-1} \right|$$

Ядерный K-средних останавливается, когда доля точек с новыми назначениями кластера падает ниже некоторого порога $\epsilon \geq 0$. Например, можно выполнять итерацию, пока никакие точки не изменяют кластеры.

Вычислительная сложность

Вычисление среднего значения ядра для каждого кластера C_i занимает время $O(n^2)$ по всем кластерам. Вычисление среднего значения ядра для каждой точки по отношению к каждому из k кластеров также занимает $O(n^2)$ времени. Наконец, вычисление ближайшего среднего для каждой точки и переназначения кластера занимает $O(kn)$ времени. Общая вычислительная сложность ядерного K-средних составляет $O(tn^2)$, где t – это количество итераций до сходимости. Сложность I/O составляет $O(t)$ проходов по ядерной матрице **K**.

8.3 Кластеризация с максимизацией ожиданий

Метод k-средних является примером «жесткой» кластеризации, где каждая точка может принадлежать только одному кластеру. Теперь мы обобщаем подход для рассмотрения «мягкого» распределения в кластеры, так что у каждой точки есть вероятность принадлежать каждому кластеру.

Пусть **D** состоит из n точек $\boldsymbol{x}_j$ в d-мерном пространстве $\mathbb{R}^d$. Пусть X_a – случайная величина, соответствующая a-ому атрибуту. Мы также используем X_a для обозначения a-того вектора столбца, соответствующий n выборкам данных из X_a. Пусть $\mathbf{X} = (X_1, X_2, \dots, X_d)$ – векторная случайная величина, определенная через d-атрибуты с помощью x_j, являющимися образцами данных из **X**.

Модель Гауссовой смеси

Мы предполагаем, что каждый кластер характеризуется многомерным нормальным распределением, то есть:

$$f_i(\mathbf{x}) = f\left(\mathbf{x}_j \middle| \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\right) = \frac{1}{(2\pi)^{-\frac{d}{2}} \left|\boldsymbol{\Sigma}_i^{-1}\right|^{\frac{1}{2}}} exp\left\{-\frac{(\mathbf{x}_j - \boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}_i^{-1} (\mathbf{x}_j - \boldsymbol{\mu}_i)}{2}\right\} \quad (8.6)$$

где $\boldsymbol{\mu}_i \in \mathbb{R}^d$ – среднее значение кластера и $\boldsymbol{\Sigma}_i \in \mathbb{R}^{d \times d}$ – ковариационная матрица неизвестные параметры. $f_i(\mathbf{x})$ – плотность вероятности **x**, принадлежности к кластеру C_i. Мы предполагаем, что функция плотности вероятности **X** задается как *модель гауссовой смеси* для всех k нормалей кластеров, определяемых как

$$f(\mathbf{x}) = \sum_{j=1}^{k} f_i(\mathbf{x}) P(C_i) = \sum_{j=1}^{k} f(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) P(C_i) \quad (8.7)$$

где априорные вероятности $P(C_i)$ называются *параметрами смеси*, которые должны удовлетворять условию

$$\sum_{i=1}^{k} P(C_i) = 1$$

Таким образом, модель гауссовой смеси характеризуется средним значением $\boldsymbol{\mu}_i$, ковариационной матрицей $\boldsymbol{\Sigma}_i$, и вероятностью смеси $P(C_i)$ для каждого из k нормальных распределений. Мы компактно запишем множество всех параметров модели в виде:

$$\boldsymbol{\theta} = \{\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1, P(C_1), \dots, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k, P(C_k)\}$$

Оценка максимального правдоподобия

С учетом набора данных **D**, определим правдоподобие $\boldsymbol{\theta}$ как условную вероятность данных **D** с учетом параметров модели $\boldsymbol{\theta}$, обозначенную как $P(\mathbf{D}|\boldsymbol{\theta})$. Потому что каждая из n точек x_j считается случайной выборкой из X (т. е. независимой, с распределением как **X**), правдоподобие $\boldsymbol{\theta}$ задается как

$$P(\mathbf{D}|\boldsymbol{\theta}) = \prod_{j=1}^{k} f(\mathbf{x}_j)$$

Целью оценки максимального правдоподобия (MLE) является выбор параметров $\boldsymbol{\theta}$, которые максимизируют вероятность, то есть:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\{P(\mathbf{D}|\boldsymbol{\theta})\}$$

Максимизация логарифма функции правдоподобия – это типичное решение, поскольку оно превращает произведение по точкам в суммирование, а максимальное значение правдоподобия и логарифмической вероятности совпадают. Т.е. MLE максимизирует

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\{\ln P(\mathbf{D}|\boldsymbol{\theta})\}$$

где *функция логарифмического правдоподобия* задается как:

$$\ln P(\mathbf{D}|\boldsymbol{\theta}) = \sum_{j=1}^{n} \ln f(\mathbf{x}_j) = \sum_{j=1}^{n} \ln\left(\sum_{i=1}^{k} f(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) P(C_i)\right) \qquad (8.8)$$

Непосредственная максимизация логарифмической вероятности над $\boldsymbol{\theta}$ – весьма трудная задача. Вместо этого мы можем использовать подход максимизации ожидания (EM) для нахождения оценок максимального правдоподобия для параметров $\boldsymbol{\theta}$. EM - это двухэтапный итерационный подход, который начинается с начального предположения для параметров $\boldsymbol{\theta}$. Учитывая текущие оценки для $\boldsymbol{\theta}$, на *шаге ожидания* EM вычисляет апостериорные вероятности кластера $P(C_i|\mathbf{x}_j)$ теорему Байеса:

$$P(C_i|\mathbf{x}_j) = \frac{P(C_i \text{ and } \mathbf{x}_j)}{P(\mathbf{x}_j)} = \frac{P(\mathbf{x}_j|C_i)P(C_i)}{\sum_{a=1}^{k} P(\mathbf{x}_j|C_a)P(C_a)}$$

Поскольку каждый кластер моделируется как многомерное нормальное распределение [ур-е (8.6)], вероятность $\mathbf{x}_j$ данного кластера можно получить, рассматривая небольшой интервал $\varepsilon > 0$ с центром в точке $\mathbf{x}_j$, следующим образом:

$$P(\mathbf{x}_j|C_i) \simeq 2\varepsilon \cdot f(\mathbf{x}_j|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) = 2\varepsilon \cdot f_i(\mathbf{x}_j)$$

Апостериорная вероятность C_i с $\mathbf{x}_j$:

$$P(C_i|\mathbf{x}_j) = \frac{f_i(\mathbf{x}_j)P(C_i)}{\sum_{a=1}^{k} f_a(\mathbf{x}_j)P(C_a)} \qquad (8.9)$$

И $P(C_i|\mathbf{x}_j)$ можно рассматривать как вес или вклад точки $\mathbf{x}_j$ в кластеризацию C_i. Далее, на этапе максимизации, используя веса $P(C_i|\mathbf{x}_j)$ EM заново пересчитывает $\boldsymbol{\theta}$, то есть он заново оценивает параметры $\boldsymbol{\mu}_i$, $\boldsymbol{\Sigma}_i$, и $P(C_i)$ для каждого кластера C_i. Пересчитанное среднее значение задается как средневзвешенное значение всех точек, пересчитанная ковариационная матрица задается как взвешенная ковариация по всем парам измерений, а пересчитанная априорная вероятность для каждого кластера задается как доля весов, которые вносят вклад в этот кластер. В разделе 8.3.3 мы формально выводим выражения для оценок MLE для параметров кластера, а в разделе 8.3.4 мы более подробно описываем общий подход EM. Мы начнем с применения алгоритма кластеризации EM для одномерных и общих d-мерных случаев.

8.3.1 EM в одномерном пространстве

Рассмотрим набор данных $\mathbf{D}$, состоящий из одного атрибута $\mathbf{X}$, где каждая точка $\boldsymbol{x}_j \in \mathbb{R}$ $(j = 1, \ldots, n)$ – случайная выборка из $\mathbf{X}$. Для модели смеси [ур-е 8.7], мы используем одномерные нормали для каждого кластера:

$$f_i(\mathbf{x}) = f\left(\mathrm{x}_j \middle| \mu_i, \sigma_i^2\right) = \frac{1}{\sqrt{2\pi}\sigma_j} exp\left\{-\frac{\left(\mathbf{x}_j - \boldsymbol{\mu}_i\right)^2}{2\sigma_i^2}\right\}$$

с параметрами кластера μ_i, σ_i^2 и $P(C_i)$. Подход EM состоит из трех этапов: инициализации, шага ожидания и шага максимизации.

Инициализация

Для каждого кластера C_i для $(i = 1, 2, \ldots, k)$, мы можем случайным образом инициализировать параметры кластер μ_i, σ_i^2 и $P(C_i)$. Среднее значение μ_i выбирается равномерно случайным образом из диапазона возможных значений для **X**. Обычно предполагается, что начальная дисперсия задается в виде $\sigma_i^2 = 1$. Наконец, предыдущие вероятности кластера инициализируются в $P(C_i) = \frac{1}{k}$, так что каждый кластер имеет равную вероятность.

Шаг ожидания

Предположим, что для каждого из k кластеров мы имеем оценку параметров, а именно: среднее значение μ_i, дисперсия σ_i^2 и априорная вероятность $P(C_i)$. Учитывая эти значения апостериорные вероятности кластеров вычисляются с помощью ур-я (8.9):

$$P\left(C_i \middle| \mathbf{x}_j\right) = \frac{f\left(\mathrm{x}_j \middle| \mu_i, \sigma_i^2\right) P(C_i)}{\sum_{a=1}^{k} f\left(\mathrm{x}_j \middle| \mu_a, \sigma_a^2\right) P(C_a)}$$

Для удобства мы используем обозначение $w_{ij} = P(C_i|x_j)$, рассматривая апостериорные вероятности как вес или вклад точки x_j в кластер C_i. Далее, пусть:

$$\boldsymbol{w}_i = (w_1, w_2, \ldots, w_n)^T$$

обозначим весовой вектор для кластера C_i через все n точек.

Шаг максимизации

При условии, что все апостериорные значения вероятности или веса $w_{ij} = P(C_i|x_j)$ известны, шаг максимизации, как следует из названия, вычисляет оценки максимального правдоподобия параметров кластера путем пересчета μ_i, σ_i^2 и $P(C_i)$. Пересчитанное значение для среднего значения кластера, μ_i, вычисляется как средневзвешенное значение из всех пунктов:

$$\mu_i = \frac{\sum_{j=1}^{n} w_{ij} \mathrm{x}_j}{\sum_{j=1}^{n} w_{ij}}$$

В терминах вектора веса $\boldsymbol{w}_i$ и вектора атрибутов $X = (x_1, x_2, \ldots, x_n)^T$ - мы можем переписать как:

$$\mu_i = \frac{\boldsymbol{w}_i{}^T X}{\boldsymbol{w}_i{}^T \mathbf{1}}$$

Пересчитанное значение кластерной дисперсии вычисляется как взвешенная дисперсия по всем точкам:

$$\sigma_i^2 = \frac{\sum_{j=1}^{n} w_{ij}(x_j - \mu_i)^2}{\sum_{j=1}^{n} w_{ij}}$$

Пусть $Z_i = X - \mu_i \mathbf{1} = (x_1 - \mu_i, x_2 - \mu_i, \ldots, x_n - \mu_i)^T = (z_{1i}, z_{2i}, \ldots, z_{in})^T$

будет отцентрированным вектором атрибутов для кластера C_i, и пусть Z_i^s –квадратный вектор, заданный как $Z_i = ({z_{i1}}^2, {z_{i2}}^2, \ldots, {z_{in}}^2)^T$. Дисперсия может быть компактно выражена в терминах произведения между вектором веса и квадратом центрированного вектора:

$$\sigma_i^2 = \frac{\boldsymbol{w}_i{}^T Z_i^{s\,T}}{\boldsymbol{w}_i{}^T \mathbf{1}}$$

Наконец, априорная вероятность кластера C_i пересчитывается как доля веса, принадлежащего C_i, от общего веса:

$$P(C_i) = \frac{\sum_{j=1}^{n} w_{ij}}{\sum_{a=1}^{k}\sum_{j=1}^{n} w_{aj}} = \frac{\sum_{j=1}^{k} w_{ij}}{\sum_{j=1}^{k} 1} = \frac{\sum_{j=1}^{k} w_{ij}}{n} \qquad (8.10)$$

где мы воспользовались тем фактом, что

$$\sum_{i=1}^{n} w_{ij} = \sum_{i=1}^{n} P(C_i|x_j) = 1$$

В векторной нотации априорная вероятность может быть записана как:

$$P(C_i) = \frac{\boldsymbol{w}_i{}^T \mathbf{1}}{n}$$

Итерация

Начиная с исходного набора значений параметров кластера μ_i, σ_i^2 и $P(C_i)$ для всех i $= 1, \ldots,$ k, алгоритм EM применяет шаг ожидания для вычисления весов $w_{ij} = P(C_i|x_j)$. Эти значения затем используются на этапе максимизации для вычисления обновленных параметров кластера μ_i, σ_i^2 и $P(C_i)$. Как шаг ожидания, так и максимизация применяются итеративно до тех пор, пока, например, не произойдет конвергенция, например, пока средние не изменятся очень мало от одной итерации к другой.

8.3.2 EM в *d*-мерном пространстве

Теперь рассмотрим метод ЭМ в d измерениях, где каждый кластер

характеризуется многомерным нормальным распределением [ур-е. (8.6)], с параметрами $\boldsymbol{\mu}_i$, $\boldsymbol{\Sigma}_i$ и $P(C_i)$. Таким образом, для каждого кластера C_i, нам нужно оценить d-мерный вектор среднего:

$$\boldsymbol{\mu}_i = (\mu_1, \mu_2, \ldots, \mu_n)^T$$

и ковариационной матрицей $d \times d$:

$$\boldsymbol{\Sigma} = \begin{pmatrix} (\sigma_1^i)^2 & \sigma_{12}^i & \cdots & \sigma_{1d}^i \\ \sigma_{21}^i & (\sigma_2^i)^2 & & \sigma_{1d}^i \\ \vdots & & \ddots & \vdots \\ \sigma_{d1}^i & \sigma_{d2}^i & \cdots & (\sigma_d^i)^2 \end{pmatrix}$$

Поскольку ковариационная матрица симметрична, мы должны оценить: $\binom{d}{2} = \frac{d(d-1)}{2}$ попарных ковариаций и d дисперсии, в общей сложности $\frac{d(d+1)}{2}$ параметры для $\boldsymbol{\Sigma}_i$. Такое количество может быть большим для практических целей, потому что у нас может быть недостаточно данных, чтобы надежно оценить все из них. Например, если d = 100, то мы должны оценить $100 * \frac{101}{2} = 5050100$ * 101/2 = 5050 параметров! Одним из упрощений является предположение, что все измерения являются независимыми, что приводит к диагональной ковариационной матрице:

$$\boldsymbol{\Sigma}_i = \begin{pmatrix} (\sigma_1^i)^2 & 0 & \cdots & 0 \\ 0 & (\sigma_2^i)^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & (\sigma_d^i)^2 \end{pmatrix}$$

При предполагаемой независимости у нас есть только d параметров для оценки диагональной ковариационной матрицы.

Инициализация

Для каждого кластера C_i, где $i = 1, \ldots, k$, мы случайным образом инициализируем среднее значение $\boldsymbol{\mu}_i$ путем выбора наугад значения μ_{ia} для каждого измерения X_a равномерно распределённого на диапазоне X_a. Ковариационная матрица инициализируется как матрица тождества $d \times d$ $\boldsymbol{\Sigma}_i = \mathbf{I}$. Наконец, апостериорные вероятности кластера инициализируются $P(C_i) = 1$, так что каждый кластер имеет равную вероятность.

Шаг ожидания

На шаге ожидания мы вычисляем апостериорную вероятность кластера C_i для точки $\boldsymbol{x}_j$, используя ур-е. (8.9), при $i = 1, \ldots, k$ и $j = 1, \ldots, n$. Как и прежде, мы используем сокращенные обозначение $w_{ij} = P(C_i|x_j)$, чтобы обозначить тот факт, что $P(C_i|x_j)$ можно рассматривать как вес вклада точки $\boldsymbol{x}_j$ в кластер C_i, и мы используем обозначение $\boldsymbol{w}_i = (w_1, w_2, \ldots, w_n)^T$ чтобы определить весовой вектор для кластера C, через все n точек.

Шаг максимизации

При заданных весах w_{ij}, на этапе максимизации мы повторно вычисляем, $\boldsymbol{\mu}_i$, $\boldsymbol{\Sigma}_i$ и $P(C_i)$. Среднее значение $\boldsymbol{\mu}_i$ для кластера C_i можно оценить как:

$$\boldsymbol{\mu}_i = \frac{\sum_{j=1}^n w_{ij} \mathbf{x}_j}{\sum_{j=1}^n w_{ij}} \qquad (8.11)$$

которое может быть компактно выражено в матричном виде как:

$$\boldsymbol{\mu}_i = \frac{\mathbf{D}^T \boldsymbol{w}_i}{\boldsymbol{w}_i{}^T \mathbf{1}}$$

Пусть $Z_i = \mathbf{D} - \mathbf{1}\boldsymbol{\mu}_i^T$ – центрированная матрица данных для кластера C_i. Пусть $\boldsymbol{z}_{ji} = \boldsymbol{x}_j - \boldsymbol{\mu}_i \in \mathbf{R}^d$ – j-я центрированную точку в Z_i. Мы можем выразить $\boldsymbol{\Sigma}_i$:

$$\boldsymbol{\Sigma}_i = \frac{\sum_{j=1}^{n} w_{ij} \boldsymbol{z}_{ji} \boldsymbol{z}_{ji}^T}{\boldsymbol{w}_i^T \mathbf{1}} \tag{8.12}$$

Учитывая попарное представление атрибутов, ковариация между измерениями X_a и X_b оценивается как:

$$\sigma_{ab}^i = \frac{\sum_{j=1}^{n} w_{ij} \left(\mathrm{x}_{ja} - \mu_{ia}\right)\left(\mathrm{x}_{jb} - \mu_{ib}\right)}{\sum_{j=1}^{n} w_{ij}}$$

где x_{ja} и μ_{ia} – значения a-го измерения для $\boldsymbol{x}_j$ и $\boldsymbol{\mu}_i$, соответственно.

Наконец, априорная вероятность $P(C_i)$ для каждого кластера – это то же самое, что и одномерный случай [ур-е. (8.10)], приведенное в виде:

$$P(C_i) = \frac{\sum_{j=1}^{k} w_{ij}}{n} = \frac{\boldsymbol{w}_i^T \mathbf{1}}{n} \tag{8.13}$$

Формальный вывод этих переоценок для $\boldsymbol{\mu}_i$ [ур-е. (8.11)], $\boldsymbol{\Sigma}_i$ [ур-е (8.12)] и $P(C_i)$ [ур-е (8.13)] приводится в разделе 8.3.3.

Алгоритм кластеризации ЭМ

Псевдокод для многомерного алгоритма кластеризации ЭМ приведен в

алгоритме 8.3. После инициализации $\boldsymbol{\mu}_i$, $\boldsymbol{\Sigma}_i$ и $P(C_i)$ для всех $i = 1, \ldots, k$, шаги ожидания и максимизации повторяются до тех пор, пока не произойдет конвергенция. Для проверки конвергенции мы проверяем, что $\sum_i \left\| \boldsymbol{\mu}_i^t - \boldsymbol{\mu}_i^{t-1} \right\|^2 \leq \varepsilon$, где $\varepsilon > 0$ – порог сходимости, и t обозначает итерацию. Другими словами, итерационный процесс продолжается до тех пор, пока изменение в среднем кластера не станет очень маленьким.

Алгоритм 8.3: Алгоритм максимизации ожидания (EM)

EXPECTATION-MAXIMIZATION $(\mathbf{D}, k, \epsilon)$:

$t \leftarrow 0$

// Initialization

Randomly initialize $\boldsymbol{\mu}_1^t, \ldots, \boldsymbol{\mu}_k^t$

$\boldsymbol{\Sigma}_i^t \leftarrow \mathbf{I}, \forall i = 1, \ldots, k$

$P^t(C_i) \leftarrow \frac{1}{k}, \forall i = 1, \ldots, k$

repeat

  $t \leftarrow t + 1$

  // Expectation Step

  for $i = 1, \ldots, k$ *and* $j = 1, \ldots, n$ **do**

    $w_{ij} \leftarrow \frac{f(\mathbf{x}_j | \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) \cdot P(C_i)}{\sum_{a=1}^{k} f(\mathbf{x}_j | \boldsymbol{\mu}_a, \boldsymbol{\Sigma}_a) \cdot P(C_a)}$ // posterior probability $P^t(C_i | \mathbf{x}_j)$

  // Maximization Step

  for $i = 1, \ldots, k$ **do**

    $\boldsymbol{\mu}_i^t \leftarrow \frac{\sum_{j=1}^{n} w_{ij} \cdot \mathbf{x}_j}{\sum_{j=1}^{n} w_{ij}}$ // re-estimate mean

    $\boldsymbol{\Sigma}_i^t \leftarrow \frac{\sum_{j=1}^{n} w_{ij} (\mathbf{x}_j - \boldsymbol{\mu}_i)(\mathbf{x}_j - \boldsymbol{\mu}_i)^T}{\sum_{j=1}^{n} w_{ij}}$ // re-estimate covariance matrix

    $P^t(C_i) \leftarrow \frac{\sum_{j=1}^{n} w_{ij}}{n}$ // re-estimate priors

until $\sum_{i=1}^{k} \left\| \boldsymbol{\mu}_i^t - \boldsymbol{\mu}_i^{t-1} \right\|^2 \leq \epsilon$

Вычислительная сложность

Для шага ожидания, чтобы вычислить апостериорные вероятности кластера, нам нужно инвертировать $\boldsymbol{\Sigma}_i$ и вычислить его определитель $|\boldsymbol{\Sigma}_i||$, который занимает $O(d^3)$. Для k кластеров время равно $O(kd^3)$. На шаге ожидания, оценка плотности $f(\mathbf{x}_j | \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$ требуется $O(d^2)$, для n точек и k кластеров это потребует $O(knd^2)$. На шаге максимизации, время обновления для $\boldsymbol{\Sigma}_i$ для k кластеров по требует $O(knd^2)$. Таким образом, вычислительная сложность метода EM, составляет $O(t(kd^3 + knd^2))$где t-число итераций. Если мы используем диагональную ковариационную матрицу, то обратная матрица и определитель могут быть вычислены за $O(d)$. Вычисление плотности в точке занимает $O(d)$ времени, так что время для шага ожидания равно $O(knd)$. Шаг максимизации также занимает $O(knd)$ для повторной оценки $\boldsymbol{\Sigma}_i$. Таким образом, общее время для диагональной ковариационной матрицы равно $O(tknd)$.. Сложность ввода-вывода для алгоритма EM – это $O(t)$, полное сканирование базы данных, потому что мы читаем весь набор точек в каждой итерации.

Метод K-средних как частный случай EM

Хотя мы предположили нормальную модель смеси для кластеров, подход EM может быть использован с другими моделями распределения плотности кластеров $P(\mathbf{x}_j | C_i)$. Например, k-среднее можно рассматривать как частный случай алгоритма EM, полученный следующим образом:

$$P(\mathbf{x}_j | C_i) = \begin{cases} 1 \text{ если } C_i = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \left\{ \left\| \mathbf{x}_j - \boldsymbol{\mu}_a \right\|^2 \right\} \\ \text{иначе } 0 \end{cases}$$

Используя ур-е (8.9), апостериорная вероятность $P(C_i | \mathbf{x}_j)$ задается какЖ

$$P(C_i|\mathbf{x}_j) = \frac{P(\mathbf{x}_j|C_i)P(C_i)}{\sum_{a=1}^{k} P(\mathbf{x}_j|C_a)P(C_a)}$$

Можно видеть, что если $P(\mathbf{x}_j|C_i) = 0$, тогда $P(C_i|\mathbf{x}_j) = 0$. В противном случае, если $P(\mathbf{x}_j|C_i) = 1$, то $P(C_i|\mathbf{x}_a) = 0$ для всех $a \neq i$, и таким образом $P(C_i|\mathbf{x}_j) = \frac{1 \cdot P(C_i)}{1 \cdot P(C_i)} = 1$. Если сложить все это вместе, то апостериорная вероятность задается в виде:

$$P(C_i|\mathbf{x}_j) = \begin{cases} 1 \text{ если } \mathbf{x}_j \in C_i, \text{т. е. } C_i = \underset{C_a}{\operatorname{argmax}} \left\{ \|\mathbf{x}_j - \boldsymbol{\mu}_a\|^2 \right\} \\ \text{иначе } 0 \end{cases}$$

Понятно, что для метода k-средних параметры кластера равны $\boldsymbol{\mu}_i$и $P(C_i)$, мы можем игнорировать ковариационную матрицу.

8.3.3 Оценка максимального правдоподобия

В этом разделе мы получаем оценки максимального правдоподобия для параметров кластера $\boldsymbol{\mu}_i$, $\boldsymbol{\Sigma}_i$ и $P(C_i)$. Мы делаем это, беря производную от логарифмической функции правдоподобия по отношению к каждому из этих параметров и устанавливая производную в ноль. Частичная производная логарифмической функции правдоподобия [ур-е. (8.8)] в отношении какого-то параметра $\boldsymbol{\theta}_i$ для кластера C_i задается как:

$$\begin{aligned} \frac{\partial}{\partial \boldsymbol{\theta}_i} \ln\big(P(\mathbf{D}|\boldsymbol{\theta})\big) &= \frac{\partial}{\partial \boldsymbol{\theta}_i} \sum_{j=1}^{n} \ln\Big(f\big(\mathbf{x}_j\big)\Big) \\ &= \sum_{j=1}^{n} \left(\frac{1}{f(\mathbf{x}_j)} \frac{\partial f(\mathbf{x}_j)}{\partial \boldsymbol{\theta}_i} \right) \\ &= \sum_{j=1}^{n} \left(\frac{1}{f(\mathbf{x}_j)} \sum_{j=1}^{n} \frac{\partial}{\partial \boldsymbol{\theta}_i} \Big(f(\mathbf{x}_j|\boldsymbol{\mu}_a, \boldsymbol{\Sigma}_a) P(C_a) \Big) \right) \\ &= \sum_{j=1}^{n} \left(\frac{1}{f(\mathbf{x}_j)} \frac{\partial}{\partial \boldsymbol{\theta}_i} \big(f(\mathbf{x}_j|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) P(C_i) \big) \right) \end{aligned}$$

Последний шаг вытекает из того, что поскольку $\boldsymbol{\theta}_i$ является параметром для i-го кластера компоненты смеси для других кластеров являются константами относительно $\boldsymbol{\theta}_i$. Используя тот факт, что $|\boldsymbol{\Sigma}_i| = \frac{\mathbf{1}}{|\boldsymbol{\Sigma}_i{}^{-1}|}$, многомерная нормальная плотность в ур-и (8.6) может быть записана следующим образом:

$$f(\mathbf{x}_j|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) = (2\pi)^{-\frac{d}{2}} \big|\boldsymbol{\Sigma}_i{}^{-1}\big|^{\frac{1}{2}} exp\{g(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)\} \qquad (8.14)$$

где

$$g(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) = -\tfrac{1}{2} \big(\mathbf{x}_j - \boldsymbol{\mu}_i\big)^T \boldsymbol{\Sigma}_i{}^{-1} \big(\mathbf{x}_j - \boldsymbol{\mu}_i\big) \qquad (8.15)$$

Таким образом, производная логарифмической функции правдоподобия может быть записана в виде

$$\frac{\partial}{\partial \boldsymbol{\theta}_i}\ln\big(P(\mathbf{D}|\boldsymbol{\theta})\big)$$

$$= \tfrac{1}{2}\textstyle\sum_{j=1}^{n}\frac{1}{f(\mathbf{x}_j)}\frac{\partial}{\partial \boldsymbol{\theta}_i}\Big((2\pi)^{-\frac{d}{2}}\big|\boldsymbol{\Sigma}_i{}^{-1}\big|^{\frac{1}{2}}exp\{g(\boldsymbol{\mu}_i,\boldsymbol{\Sigma}_i)\}P(C_i)\Big) \qquad (8.16)$$

Далее мы используем тот факт, что

$$\frac{\partial}{\partial \boldsymbol{\theta}_i}exp\{g(\boldsymbol{\mu}_i,\boldsymbol{\Sigma}_i)\} = exp\{g(\boldsymbol{\mu}_i,\boldsymbol{\Sigma}_i)\}\frac{\partial}{\partial \boldsymbol{\theta}_i}g(\boldsymbol{\mu}_i,\boldsymbol{\Sigma}_i) \qquad (8.17)$$

Оценка среднего значения

Для получения оценки максимального правдоподобия для среднего значения $\boldsymbol{\mu}_i$, мы должны взять производную логарифмического правдоподобия относительно $\boldsymbol{\theta}_i = \boldsymbol{\mu}_i$. Согласно ур-ю (8.16), единственное слагаемое, включающее $\boldsymbol{\mu}_i$, это $exp\{g(\boldsymbol{\mu}_i,\boldsymbol{\Sigma}_i)\}$. Используя тот факт, что

$$\frac{\partial}{\partial \boldsymbol{\mu}_i}exp\{g(\boldsymbol{\mu}_i,\boldsymbol{\Sigma}_i)\} = \boldsymbol{\Sigma}_i{}^{-1}\big(\mathbf{x}_j - \boldsymbol{\mu}_i\big) \qquad (8.18)$$

и используя ур-е. (8.17), частичная производная логарифмического правдоподобия [ур-е. (8.16)] в отношении $\boldsymbol{\mu}_i$:

$$\frac{\partial}{\partial \boldsymbol{\mu}_i}\ln\big(P(\mathbf{D}|\boldsymbol{\theta})\big)$$

$$= \sum_{j=1}^{n}\frac{1}{f(\mathbf{x}_j)}(2\pi)^{-\frac{d}{2}}\big|\boldsymbol{\Sigma}_i{}^{-1}\big|^{\frac{1}{2}}exp\{g(\boldsymbol{\mu}_i,\boldsymbol{\Sigma}_i)\}P(C_i)\boldsymbol{\Sigma}_i{}^{-1}\big(\mathbf{x}_j - \boldsymbol{\mu}_i\big)$$

$$= \sum_{j=1}^{n}\frac{f\big(\mathbf{x}_j\big|\boldsymbol{\mu}_i,\boldsymbol{\Sigma}_i\big)P(C_i)}{f(\mathbf{x}_i)}\boldsymbol{\Sigma}_i{}^{-1}\big(\mathbf{x}_j - \boldsymbol{\mu}_i\big)$$

$$= \sum_{j=1}^{n} w_{ij}\boldsymbol{\Sigma}_i{}^{-1}\big(\mathbf{x}_j - \boldsymbol{\mu}_i\big)$$

где мы использовали ур-я (8.14) и (8.9), а также тот факт, что

$$w_{ij} = P\big(C_i|\mathbf{x}_j\big) = \frac{f\big(\mathbf{x}_j\big|\boldsymbol{\mu}_i,\boldsymbol{\Sigma}_i\big)P(C_i)}{f(\mathbf{x}_i)}$$

Устанавливая частную производной логарифмического правдоподобия на нулевой вектор и умножая обе стороны на $\boldsymbol{\Sigma}_i$, мы получаем:

$\sum_{j=1}^{n} w_{ij}(\mathbf{x}_j - \boldsymbol{\mu}_i) = 0$, это означает, что

$\sum_{j=1}^{n} w_{ij}\mathbf{x}_j = \boldsymbol{\mu}_i \sum_{j=1}^{n} w_{ij}$, и поэтому

$$\boldsymbol{\mu}_i = \frac{\sum_{j=1}^{n} w_{ij}\mathbf{x}_j}{\sum_{j=1}^{n} w_{ij}} \tag{8.19}$$

это именно та формула переоценки, которую мы использовали в ур-и. (8.11).

Оценка ковариационной матрицы

Для переоценки ковариационной матрицы $\boldsymbol{\Sigma}_i$, возьмем частную производную от ур-я (8.16) по $\boldsymbol{\Sigma}_i^{-1}$, используя правила для дифференциации $\left|\boldsymbol{\Sigma}_i^{-1}\right|^{\frac{1}{2}} exp\{g(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)\}$.

Используя тот факт, что для любой квадратной матрицы **A** мы имеем $\frac{\partial|\mathbf{A}|}{\partial\mathbf{A}} = |\mathbf{A}| \cdot (\mathbf{A}^{-1})^T$, то производная от $\left|\boldsymbol{\Sigma}_i^{-1}\right|^{\frac{1}{2}}$ в отношении $\boldsymbol{\Sigma}_i^{-1}$:

$$\frac{\partial|\boldsymbol{\Sigma}_i^{-1}|^{\frac{1}{2}}}{\partial\boldsymbol{\Sigma}_i^{-1}} = \frac{1}{2} \cdot \left|\boldsymbol{\Sigma}_i^{-1}\right|^{-\frac{1}{2}} \cdot \left|\boldsymbol{\Sigma}_i^{-1}\right| \cdot \boldsymbol{\Sigma}_i = \frac{1}{2}\left|\boldsymbol{\Sigma}_i^{-1}\right|^{\frac{1}{2}} \cdot \boldsymbol{\Sigma}_i \tag{8.20}$$

Далее, используя тот факт, что для квадратной матрицы $\mathbf{A} \in \mathbf{R}^{d\times d}$ векторы $\mathbf{a}, \mathbf{b} \in \mathbf{R}^d$, у нас есть $\frac{\partial}{\partial\mathbf{A}}\mathbf{a}^T\mathbf{A}\mathbf{b} = \mathbf{a}\mathbf{b}^T$, производная от $exp\{g(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)\}$ по $\boldsymbol{\Sigma}_i^{-1}$ получается из ур-я (8.17) следующим образом:

$$\frac{\partial}{\partial\boldsymbol{\Sigma}_i^{-1}} exp\{g(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)\} = -\frac{1}{2} exp\{g(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)\}(\mathbf{x}_j - \boldsymbol{\mu}_i)(\mathbf{x}_j - \boldsymbol{\mu}_i)^T \tag{8.21}$$

Используя правила вывода на ур-ях (8.20) и (8.21), получаем

$$\begin{aligned}
&\frac{\partial}{\partial\boldsymbol{\Sigma}_i^{-1}}\left|\boldsymbol{\Sigma}_i^{-1}\right|^{\frac{1}{2}} exp\{g(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)\} \\
&= \frac{1}{2}\left|\boldsymbol{\Sigma}_i^{-1}\right|^{\frac{1}{2}}\boldsymbol{\Sigma}_i exp\{g(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)\} - \frac{1}{2}\left|\boldsymbol{\Sigma}_i^{-1}\right|^{-\frac{1}{2}} exp\{g(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)\}\left(\mathbf{x}_j - \boldsymbol{\mu}_i\right)\left(\mathbf{x}_j - \boldsymbol{\mu}_i\right)^T \\
&= \frac{1}{2}\left|\boldsymbol{\Sigma}_i^{-1}\right|^{\frac{1}{2}} exp\{g(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)\}(\boldsymbol{\Sigma}_i - \left(\mathbf{x}_j - \boldsymbol{\mu}_i\right)\left(\mathbf{x}_j - \boldsymbol{\mu}_i\right)^T)
\end{aligned} \tag{8.22}$$

Подставляя ур-е (8.22) в ур-е (8.16) производная логарифмической функции правдоподобия относительно $\boldsymbol{\Sigma}_i^{-1}$ задается как:

$$\begin{aligned}
&\frac{\partial}{\partial\boldsymbol{\Sigma}_i^{-1}}\ln\left(P(\mathbf{D}|\boldsymbol{\theta})\right) \\
&= \frac{1}{2}\sum_{j=1}^{n}\frac{(2\pi)^{-\frac{d}{2}}\left|\boldsymbol{\Sigma}_i^{-1}\right|^{\frac{1}{2}} exp\{g(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)\}P(C_i)}{f(\mathbf{x}_i)}\left(\boldsymbol{\Sigma}_i - \left(\mathbf{x}_j - \boldsymbol{\mu}_i\right)\left(\mathbf{x}_j - \boldsymbol{\mu}_i\right)^T\right) \\
&= \frac{1}{2}\sum_{j=1}^{n}\frac{f\left(\mathbf{x}_j|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\right)P(C_i)}{f(\mathbf{x}_i)}\left(\boldsymbol{\Sigma}_i - \left(\mathbf{x}_j - \boldsymbol{\mu}_i\right)\left(\mathbf{x}_j - \boldsymbol{\mu}_i\right)^T\right) \\
&= \frac{1}{2}\sum_{j=1}^{n} w_{ij}\left(\boldsymbol{\Sigma}_i - \left(\mathbf{x}_j - \boldsymbol{\mu}_i\right)\left(\mathbf{x}_j - \boldsymbol{\mu}_i\right)^T\right)
\end{aligned}$$

Устанавливая производную на нулевую матрицу $\mathbf{0}_{d\times d}$, мы можем решить для $\boldsymbol{\Sigma}_i$:

$$\sum_{j=1}^{n} w_{ij}\left(\boldsymbol{\Sigma}_i - \left(\mathbf{x}_j - \boldsymbol{\mu}_i\right)\left(\mathbf{x}_j - \boldsymbol{\mu}_i\right)^T\right) = \mathbf{0}_{d\times d}, \text{ из чего следует, что}$$

$$\boldsymbol{\Sigma}_i = \frac{\sum_{j=1}^{n} w_{ij}(\mathbf{x}_j - \boldsymbol{\mu}_i)(\mathbf{x}_j - \boldsymbol{\mu}_i)^T}{\sum_{j=1}^{n} w_{ij}} \qquad (8.23)$$

Таким образом, мы видим, что оценка максимального правдоподобия для ковариационной матрицы задается как взвешенная форма внешнего произведения в ур-и (8.12).

Оценка априорной вероятности: параметры смеси

Для получения оценки максимального правдоподобия для параметров смеси или априорных вероятностей $P(C_i)$, мы должны взять частную производную логарифмического правдоподобия [ур-е (8.16)] в отношении $P(C_i)$. Однако мы должны ввести множитель Лагранжа α для ограничения $\sum_{a=1}^{k} P(C_a) = 1$. Таким образом, мы берем производное:

$$\frac{\partial}{\partial P(C_i)}\left(\ln\left(P(\mathbf{D}|\boldsymbol{\theta})\right) + \alpha\left(\sum_{a=1}^{k} P(C_a) - 1\right)\right) \qquad (8.24)$$

Частная производная логарифмического правдоподобия в ур-и (8.16) по $P(C_i)$:

$$\frac{\partial}{\partial P(C_i)}\ln\left(P(\mathbf{D}|\boldsymbol{\theta})\right) = \sum_{j=1}^{n} \frac{f\left(\mathbf{x}_j|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\right)}{f(\mathbf{x}_i)}$$

Таким образом, производная в ур-и (8.24):

$$\left(\sum_{j=1}^{n} \frac{f\left(\mathbf{x}_j|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\right)}{f(\mathbf{x}_i)}\right) + \alpha$$

Устанавливая производную в ноль и умножая с обеих сторон на $P(C_i)$, мы получаем

$$\sum_{j=1}^{n} \frac{f\left(\mathbf{x}_j|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\right)P(C_i)}{f(\mathbf{x}_i)} = -\alpha P(C_i)$$

$$\sum_{j=1}^{n} w_{ij} = -\alpha P(C_i) \qquad (8.25)$$

Делая суммирование ур-я (8.25) по всем кластерам, получим:

$$\sum_{i=1}^{i}\sum_{j=1}^{n} w_{ij} = -\alpha \sum_{i=1}^{n} P(C_i)$$

$$\text{or } n = -\alpha \qquad (8.26)$$

Последний шаг вытекает из того, что $\sum_{j=1}^{k} w_{ij} = 1$. Подставляя ур-е (8.26) в ур-е (8.25), получаем оценку максимального правдоподобия для $P(C_i)$ следующим образом:

$$P(C_i) = \frac{\sum_{j=1}^{n} w_{ij}}{n} \qquad (8.27)$$

что соответствует формуле в ур-и (8.13).

Мы видим, что все три параметра $\boldsymbol{\mu}_i$, $\boldsymbol{\Sigma}_i$ и $P(C_i)$ для кластера C_i зависят от весов w_{ij}, которые соответствуют апостериорным вероятностям кластера $P(C_i|\mathbf{x}_j)$. Таким образом, уравнения (8.19), (8.23) и (8.27) не представляют собой замкнутое решение для максимизации логарифмической функции правдоподобия. Вместо этого мы используем итеративный подход EM для вычисления w_{ij} на шаге ожидания, а затем мы повторно оцениваем $\boldsymbol{\mu}_i$, $\boldsymbol{\Sigma}_i$ и $P(C_i)$ на этапе максимизации. Далее мы опишем структуру EM более подробно.

8.3.4 EM подход

Непосредственная максимизация логарифмической функции правдоподобия весьма трудна [ур-е. (8.8)], потому что слагаемое смеси появляется внутри логарифма. Проблема в том, что для любой точки $\mathbf{x}_j$ мы не знаем, из какого нормального или смешанного компонента она происходит. Предположим, что мы знали эту информацию, то есть предположим, что каждая точка $\mathbf{x}_j$ имеет связанное значение, указывающее на кластер, который генерирует точку. Как мы увидим, гораздо легче максимизировать логарифмическую вероятность, учитывая эту информацию.

Категориальный атрибут, соответствующий метке кластера, может быть смоделирован как векторная случайная величина $\mathbf{C} = (C_1, C_2, \ldots, C_k)$, где C_i является случайной величиной Бернулли. Если заданная точка генерируется из кластера C_i, то $C_i = 1$, в противном случае $C_i = 0$. Параметр $P(C_i)$ – это вероятность $P(C_i = 1)$. Потому что каждая точка может быть сгенерирована только из одного кластера, если $C_a = 1$ для данной точки, тогда $C_i = 0$ для всех $i \neq a$. отсюда следует, что $\sum_{i=1}^{k} P(C_i) = 1$.

Для каждой точки $\mathbf{x}_j$, пусть его кластерный вектор равен $\mathbf{c}_j = (\mathrm{c}_{j1}, \mathrm{c}_{j2}, \ldots, \mathrm{c}_{jk})^T$. Только один компонент из $\mathbf{c}_j$ имеет значение 1. Если $\mathbf{c}_{ji} = 11$, это означает, что $C_i = 1$, то есть кластер C_i генерирует точка $\mathbf{x}_j$. Вероятностная массовая функция для $\mathbf{C}$ задается в виде:

$$P(\mathbf{C} = \mathbf{c}_j) = \prod_{i=1}^{k} P(C_i)^{\mathbf{c}_{ji}}$$

Учитывая кластерную информацию $\mathbf{c}_j$ для каждой точки $\mathbf{x}_j$, функция плотности условной вероятности для $\mathbf{X}$ задается в виде:

$$f(\mathbf{x}_j|\mathbf{c}_j) = \prod_{i=1}^{k} f(\mathbf{x}_j|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)^{\mathbf{c}_{ji}}$$

Только один кластер может генерировать $\mathbf{x}_j$, пусть C_a, в этом случае $\mathrm{c}_{ja} = 1$, и приведенное выше выражение упростилось бы до $f(\mathbf{x}_j|\mathbf{c}_j) = f(\mathbf{x}_j|\boldsymbol{\mu}_a, \boldsymbol{\Sigma}_a)$.

Пара $(\mathbf{x}_j, \mathbf{c}_j)$ является случайной выборкой, полученной из совместного распределения вектора случайных величин $\mathbf{X} = (X_1, X_2, \ldots, X_d)$ и $\mathbf{C} = (C_1, C_2, \ldots, C_k)$, соответствующие d атрибутам данных и k атрибутам кластера. Совместная функция плотности $\mathbf{X}$ и $\mathbf{C}$ задается в виде:

$$f(\mathbf{x}_j \text{ and } \mathbf{c}_j) = f(\mathbf{x}_j|\mathbf{c}_j)P(\mathbf{c}_j) = \prod_{i=1}^{k}\left(f(\mathbf{x}_j|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)P(C_i)\right)^{\mathbf{c}_{ji}}$$

Логарифмическая вероятность для данных, приведенных в кластерной информации, выглядит следующим образом:

$$\ln\left(P(\mathbf{D}|\boldsymbol{\theta})\right) = \ln\prod_{i=1}^{k} f(\mathbf{x}_j \text{ and } \mathbf{c}_j|\boldsymbol{\theta})$$

$$= \sum_{j=1}^{n} \ln f(\mathbf{x}_j \text{ and } \mathbf{c}_j|\boldsymbol{\theta})$$

$$= \sum_{j=1}^{n} \ln\left(\prod_{i=1}^{k}\left(f(\mathbf{x}_j|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)P(C_i)\right)^{\mathbf{c}_{ji}}\right)$$

$$= \sum_{j=1}^{n}\sum_{i=1}^{k} \mathrm{c}_{ji}\left(\ln f(\mathbf{x}_j|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) + \ln P(C_i)\right) \qquad (8.28)$$

Шаг ожидания

На шаге ожидания мы вычисляем ожидаемое значение логарифмической вероятности для помеченных данных, приведенных в уравнении. (8.28). Ожидание недостающей кластерной информации $\mathbf{c}_j$ описывается с помощью $\boldsymbol{\mu}_i$, $\boldsymbol{\Sigma}_i$ и $P(C_i)$ и фиксированного $\mathbf{x}_j$. Вследствие линейности ожидания, ожидаемое значение логарифмической вероятности задается в виде:

$$E[ln\ P\ (\mathbf{D}|\boldsymbol{\theta})] = \sum_{j=1}^{n}\sum_{i=1}^{k} E\left[\mathrm{c}_{ji}\right]\left(\ln f(\mathbf{x}_j|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) + \ln P(C_i)\right)$$

Ожидаемое значение $E\left[\mathrm{c}_{ji}\right]$ может быть вычислено как

$$E\left[\mathrm{c}_{ji}\right] = 1 \cdot P(\mathrm{c}_{ji} = 1|\mathbf{x}_j) + 0 \cdot P(\mathrm{c}_{ji} = 0|\mathbf{x}_j) = P(\mathrm{c}_{ji} = 1|\mathbf{x}_j) = P(C_i|\mathbf{x}_j)$$

$$= \frac{P(C_i|\mathbf{x}_j)P(C_i)}{P(\mathbf{x}_j)} = \frac{f(\mathbf{x}_j|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)P(C_i)}{f(\mathbf{x}_j)}$$

$$= w_{ji} \qquad (8.29)$$

Таким образом, на шаге ожидания мы используем значения $\boldsymbol{\theta} = \{\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i, P(C_i)\}_{i=1}^{k}$, чтобы оценить апостериорные вероятности или веса w_{ji} для каждой точки для каждого кластера. Используя $E\left[\mathrm{c}_{ji}\right] = w_{ji}$, ожидаемое значение логарифмической функции правдоподобия можно переписать следующим образом:

$$E[ln\ P\ (\mathbf{D}|\boldsymbol{\theta})] = \sum_{j=1}^{n}\sum_{i=1}^{k} w_{ij}\left(\ln f(\mathbf{x}_j|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) + \ln P(C_i)\right) \qquad (8.30)$$

Шаг максимизации

На шаге максимизации мы максимизируем ожидаемое значение логарифмического правдоподобия [ур-е (8.30)]. Беря производную по отношению к $\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i$ или $P(C_i)$ мы можем не обращать внимания на слагаемые всех остальных кластеров. Производная от ур-я (8.30) в отношении $\boldsymbol{\mu}_i$ задается как:

$$\frac{\partial}{\partial \boldsymbol{\mu}_i} E[ln\ P\ (\mathbf{D}|\boldsymbol{\theta})]$$

$$= \frac{\partial}{\partial \boldsymbol{\mu}_i} \sum_{j=1}^{n} w_{ij} \ln f\left(\mathbf{x}_j \middle| \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\right)$$

$$= \sum_{j=1}^{n} w_{ij} \frac{1}{f\left(\mathbf{x}_j \middle| \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\right)} f\left(\mathbf{x}_j \middle| \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\right) {\boldsymbol{\Sigma}_i}^{-1} \left(\mathbf{x}_j - \boldsymbol{\mu}_i\right)$$

$$= \sum_{j=1}^{n} w_{ij} {\boldsymbol{\Sigma}_i}^{-1} \left(\mathbf{x}_j - \boldsymbol{\mu}_i\right)$$

где мы использовали наблюдение, что

$$\frac{\partial}{\partial \boldsymbol{\mu}_i} f\left(\mathbf{x}_j \middle| \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\right) = f\left(\mathbf{x}_j \middle| \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\right) {\boldsymbol{\Sigma}_i}^{-1} \left(\mathbf{x}_j - \boldsymbol{\mu}_i\right)$$

что следует из ур-й (8.14), (8.17) и (8.18). Устанавливая производную от

ожидаемого значения логарифмического правдоподобия в нулевой вектор и умножая с обеих сторон на $\boldsymbol{\Sigma}$, мы получаем

$$\boldsymbol{\mu}_i = \frac{\sum_{i=1}^{n} w_{ij} \mathbf{x}_j}{\sum_{i=1}^{n} w_{ji}}$$

что соответствует формуле в ур-и (8.11).

Используя уравнения (8.22) и (8.14), получаем производную от ур-я (8.30) от ${\boldsymbol{\Sigma}_i}^{-1}$ следующим образом:

$$\frac{\partial}{\partial {\boldsymbol{\Sigma}_i}^{-1}} \ln E[P(\mathbf{D}|\boldsymbol{\theta})]$$

$$= \sum_{i=1}^{n} w_{ij} \cdot \frac{1}{f\left(\mathbf{x}_j \middle| \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\right)} \cdot \frac{1}{2} f\left(\mathbf{x}_j \middle| \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\right) \left(\boldsymbol{\Sigma}_i - \left(\mathbf{x}_j - \boldsymbol{\mu}_i\right)\left(\mathbf{x}_j - \boldsymbol{\mu}_i\right)^T\right)$$

$$= \frac{1}{2} \sum_{i=1}^{n} w_{ij} \cdot \left(\boldsymbol{\Sigma}_i - \left(\mathbf{x}_j - \boldsymbol{\mu}_i\right)\left(\mathbf{x}_j - \boldsymbol{\mu}_i\right)^T\right)$$

Устанавливая производную на нулевую матрицу $d \times d$ и решеная для $\boldsymbol{\Sigma}_i$:

$$\boldsymbol{\Sigma}_i = \frac{\sum_{i=1}^{n} w_{ij} \cdot \left(\mathbf{x}_j - \boldsymbol{\mu}_i\right)\left(\mathbf{x}_j - \boldsymbol{\mu}_i\right)^T}{\sum_{i=1}^{n} w_{ij}}$$

это то же самое, что и в ур-и (8.12).

Используя множитель Лагранжа α для ограничения $\sum_{i=1}^{k} P(C_i) = 1$, и отмечая, что в логарифмической функции правдоподобия [ур-е. (8.30)], слагаемое $\ln f\left(\mathbf{x}_j \middle| \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\right)$ – постоянная величина, дифференцируя по $P(C_i)$, мы получаем следующее:

$$\frac{\partial}{\partial P(C_i)}\left(\ln E[P(\mathbf{D}|\boldsymbol{\theta})] + \alpha\left(\sum_{a=1}^{k} P(C_a) - 1\right)\right) = \frac{\partial}{\partial P(C_i)}\left(w_{ij} \ln P(C_i) + \alpha P(C_i)\right)$$

$$= \left(\sum_{j=1}^{n} w_{ij} \frac{1}{P(C_i)}\right) + \alpha$$

Устанавливая производную в ноль, получим:

$$\sum_{j=1}^{n} w_{ij} = -\alpha P(C_i)$$

Используя тот же вывод, что и в ур-и (8.26) получаем

$$P(C_i) = \frac{\sum_{j=1}^{n} w_{ij}}{n}$$

что идентично формуле переоценки в ур-и (8.13).

Глава 9. Иерархическая кластеризация

Для n точек в d-мерном пространстве, цель иерархической кластеризации – создать последовательность вложенных разделов, которую можно удобно визуализировать с помощью дерева или иерархии кластеров, также называемой дендрограммой кластеров. Кластеры в иерархии варьируются от слабо детализированных до сильно детализированных – нижний уровень дерева (листья) состоит из каждой точки в своем собственном кластере, тогда как самый высокий уровень (корень) состоит из всех точек в одном кластере. Обе эти группы можно рассматривать как тривиальные кластеры. На каком-то промежуточном уровне мы можем найти значимые кластеры. Если пользователь предоставляет k, желаемое количество кластеров, мы можем выбрать уровень, на котором будет находиться k кластеров.

9.1 Исходные данные

Для датасета $\mathbf{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, где $\mathbf{x}_i \in \mathbb{R}^d$, кластеризация $C = \{C_1, C_2, \ldots, C_k\}$ это разделение $\mathbf{D}$, при котором каждый кластер представляет собой набор точек $C_i \subseteq \mathbf{D}$ такой, что кластеры попарно не пересекаются $C_i \cap C_j = \emptyset$(для всех $i \neq j$), и $\bigcup_{i=1}^{k} C_i = \mathbf{D}$. Кластеризация $\mathcal{A} = \{\mathcal{A}_1, \ldots, \mathcal{A}_r\}$ считается вложенной в другую кластеризацию $\mathcal{B} = \{\mathcal{B}_1, \ldots, \mathcal{B}_s\}$, тогда и только тогда, когда $r > s$ и для каждого $\mathcal{A}_i \in \mathcal{A}$ существует кластер $\mathcal{B}_j \in \mathcal{B}$ такой, что $\mathcal{A}_i \subseteq \mathcal{B}_j$. Иерархическая кластеризация дает последовательность из n вложенных разделов, начиная от тривиальной кластеризации $C_1 = \{\{\mathbf{x}_1\}, \ldots, \{\mathbf{x}_n\}\}$, где каждая точка находится в отдельном кластере по отношении к другой тривиальной кластеризации $C_n = \{\{\mathbf{x}_1\}, \ldots, \{\mathbf{x}_n\}\}$, где все точки располагаются в одном кластере. В общем случае, кластеризация C_{t-1} вложена в кластеризацию C_t. Дендрограмма кластеризации – это бинарное дерево с корнем, которое фиксирует эту вложенную структуру с ребрами между кластером $C_i \in C_{t-1}$ и кластером $C_j \in C_t$, если C_i вложен в C_j, то есть, если $C_i \subset C_j$. Таким образом, дендрограмма фиксирует всю последовательность вложенных кластеров.

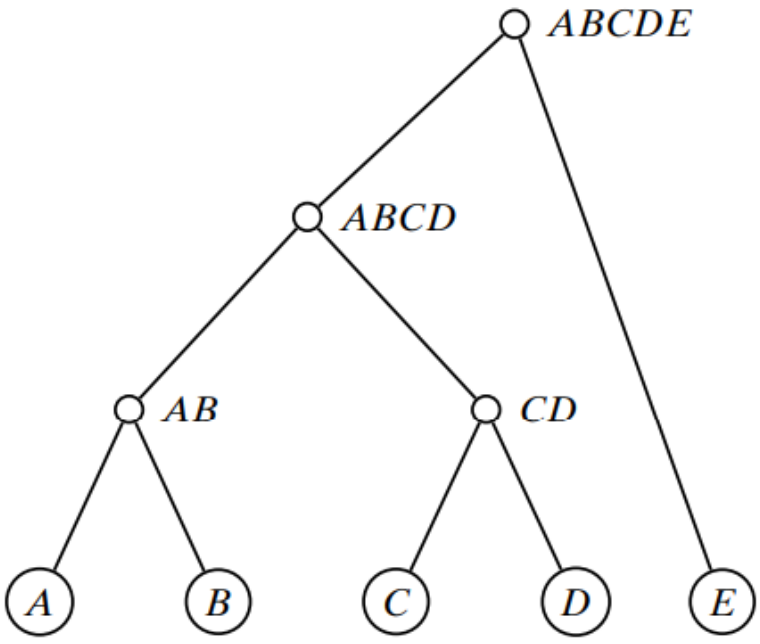


Рисунок 9.1 – Дендрограмма иерархической кластеризации

Пример 9.1. На рисунке 9.1 показан пример иерархической кластеризации пяти отмеченных точек: *A, B, C, D* и *E*. Дендрограмма представляет следующую последовательность вложенных разделов:

Кластеризация	**Кластеры**
C_1	$\{A\},\{B\},\{C\},\{\mathrm{D}\},\{E\}$
C_2	$\{AB\},\{C\},\{\mathrm{D}\},\{E\}$
C_3	$\{AB\},\{C\mathrm{D}\},\{E\}$
C_4	$\{ABC\mathrm{D}\},\{E\}$
C_5	$\{ABC\mathrm{D}E\}$

где $C_i \subset C_{t-1}$ для $t = 2, \ldots, 5$. Мы предполагаем, что *A* и *B* объединены до *C* и *D*.

Количество иерархических кластеризаций

Количество различных вложенных или иерархических кластеризаций соответствует количеству различных двоичных корневых деревьев или дендрограмм с n листами с различными метками. Любое дерево с t вершинами имеет $t - 1$ ребро. Кроме того, любое корневое двоичное дерево с m листьями имеет $m - 1$ внутренних узлов. Таким образом, дендрограмма с m листовыми узлами имеет всего $t = m + m - 1 = 2m - 1$ узлов и, следовательно, $t - 1 = 2m - 2$ ребер. Чтобы подсчитать количество различных топологий дендрограммы, давайте рассмотрим, как мы можем расширить дендрограмму с m листами, добавив дополнительный лист, чтобы получить дендрограмму с $m + 1$ листом. Обратите внимание, что мы можем добавить дополнительный лист, разделив (т. е. создав ответвление) любое из $2m - 2$ ребер. Кроме того, мы также можем добавить новый лист в качестве потомка нового корня, получая $2m - 2 + 1 = 2m - 1$ новых дендрограмм с $m + 1$ листом. Таким образом, общее количество различных дендрограмм с n листами получается с помощью следующего произведения:

$$\prod_{m=1}^{n-1}(2m-1) = 1 \times 3 \times 5 \times 7 \times \ldots \times (2n-3) = (2n-3)!! \qquad (9.1)$$

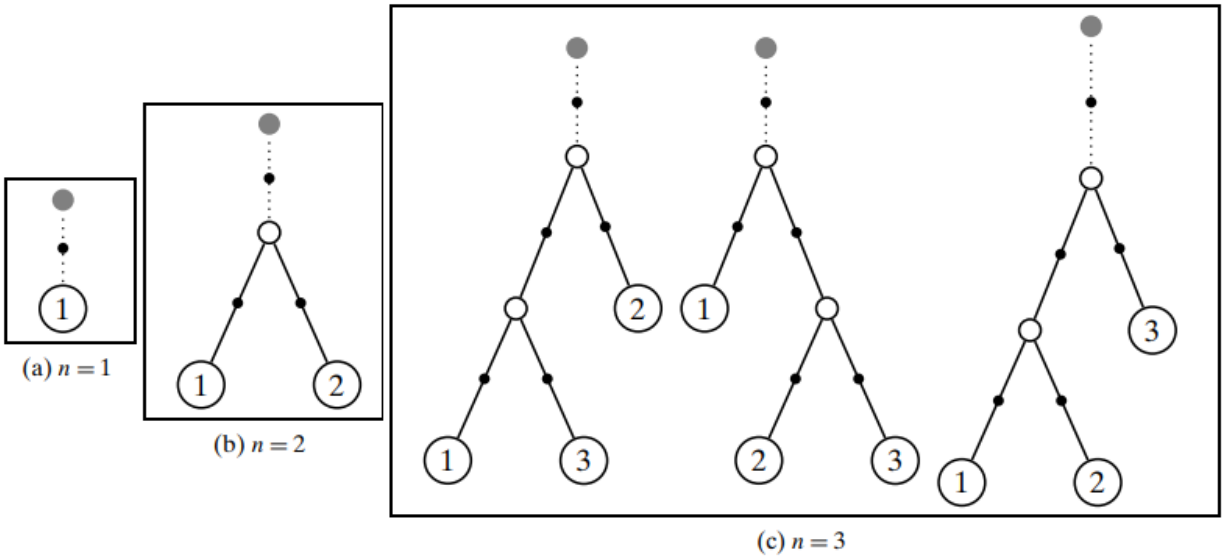


Рисунок 9.2 – Количество иерархических кластеризаций

Индекс m в формуле (9.1) увеличивается до $n - 1$, потому что последний член в произведении обозначает количество дендрограмм, которые можно получить, когда мы расширяем дендрограмму с $n - 1$ листом, добавляя еще один лист, чтобы получить дендрограммы с n листами.

Таким образом, количество возможных иерархических кластеризаций задается как $(2n - 3)!!$, что дает очень быстрый рост. Очевидно, что наивный подход к перечислению всех возможных иерархических кластеризаций просто невозможен.

Пример 9.2. На рисунке 9.2 показано количество деревьев с одним, двумя и тремя листьями. Серые узлы – это виртуальные корни, а черные точки указывают места, где можно добавить новый лист. Возможно только одно дерево с одним листом, как показано на рисунке 9.2a. Его можно расширить только одним способом, чтобы получить уникальное дерево с двумя листьями, показанное на рис. 9.2b. Однако у этого дерева есть три возможных места, где можно добавить третий лист. Каждый из этих случаев показан на рисунке 9.2c. Далее мы можем видеть, что каждое из деревьев с m = 3 листьями имеет пять мест, где можно добавить четвертый лист и так далее, что подтверждает уравнение для количества иерархических кластеризаций в уравнении (9.1).

9.2 Агломеративная иерархическая кластеризация

В агломеративной иерархической кластеризации мы начинаем с каждой из n точек в отдельном кластере. Мы многократно объединяем два ближайших кластера, пока все точки не станут членами одного кластера, как показано в псевдокоде, приведенном в алгоритме 9.1. Формально, имея набор кластеров $C = \{C_1, C_2, \dots, C_m\}$, мы находим ближайшую пару кластеров C_i и C_j и объединяем их в новый кластер $C_{ij} = C_i \cup C_j$. Затем мы обновляем набор кластеров, удаляя C_i и C_j и добавляя C_{ij}, таким образом $\mathrm{C} = \big(\mathrm{C} \setminus \{C_i, C_j\}\big) \cup \{C_{ij}\}$. Мы повторяем процесс до тех пор, пока C не будет содержать только один кластер. Поскольку количество кластеров уменьшается на один на каждом этапе, этот процесс приводит к последовательности n

вложенных кластеров. Если надо, мы можем остановить процесс слияния, когда останется ровно k кластеров.

Алгоритм 9.1: Агломеративная иерархическая кластеризация

```
AGGLOMERATIVECLUSTERING(D, k):
1 C ← {C_i = {x_i} | x_i ∈ D} // Each point in separate cluster
2 Δ ← {δ(x_i, x_j): x_i, x_j ∈ D} // Compute distance matrix
3 repeat
4     Find the closest pair of clusters C_i, C_j ∈ C
5     C_ij ← C_i ∪ C_j // Merge the clusters
6     C ← (C \ {C_i, C_j}) ∪ {C_ij} // Update the clustering
7     Update distance matrix Δ to reflect new clustering
8 until |C| = k
```

Расстояние между кластерами

Основным этапом алгоритма является определение ближайшей пары кластеров. Некоторые меры расстояния, такие как одиночная связь, полная связь, среднее по группе и другие, обсуждаемые в следующих параграфах, могут использоваться для вычисления расстояния между любыми двумя кластерами. Расстояние между кластерами в конечном итоге основано на расстоянии между двумя точками, которое обычно вычисляется с использованием евклидова расстояния или L_2-нормы

$$\delta(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_2 = \left(\sum_{i=1}^{d} (x_i - y_i)^2\right)^{1/2}$$

Тем не менее, можно использовать другие метрики расстояния или, если она доступна, матрицу расстояний, задаваемую пользователем.

Метод одиночной связи

Для двух кластеров C_i и C_j расстояние между ними, обозначаемое $\delta(C_i, C_j)$, определяется как минимальное расстояние между точкой в C_i и точкой в C_j.

$$\delta(C_i, C_j) = \min\{\delta(\boldsymbol{x}, \boldsymbol{y}) \mid \boldsymbol{x} \in C_i, \boldsymbol{y} \in C_j\}$$

Название *одиночная связь* происходит из наблюдения, что если мы выберем минимальное расстояние между точками в двух кластерах и соединим эти точки, то (как правило) между этими кластерами будет существовать только одна связь, потому что все другие пары точек будут дальше.

Метод полной связи

Расстояние между двумя кластерами определяется как максимальное расстояние между точкой в C_i и точкой в C_j:

$$\delta(C_i, C_j) = \max\{\delta(\boldsymbol{x}, \boldsymbol{y}) \mid \boldsymbol{x} \in C_i, \boldsymbol{y} \in C_j\}$$

Название *полная связь* передает тот факт, что если мы соединим все пары точек из двух кластеров с расстоянием не более $\delta(C_i, C_j)$, то все возможные пары будут соединены, то есть мы получим полную связь.

Метод средней связи

Расстояние между двумя кластерами определяется как среднее попарное расстояние между точкой в C_i и точкой в C_j:

$$\delta(C_i, C_j) = \frac{\sum_{\boldsymbol{x} \in C_i} \sum_{\boldsymbol{y} \in C_i} \delta(\boldsymbol{x}, \boldsymbol{y})}{n_i \cdot n_j}$$

Где $n_i = |C_i|$ обозначает количество точек в кластере C_i.

Центроидный метод

Расстояние между двумя кластерами определяется как расстояние между средними или центроидами двух кластеров:

$$\delta(C_i, C_j) = \delta(\boldsymbol{\mu}_i, \boldsymbol{\mu}_j) \tag{9.2}$$

где $\boldsymbol{\mu}_i = \frac{1}{n_i} \sum_{\boldsymbol{x} \in C_i} \boldsymbol{x}$.

Минимальная дисперсия: метод Уорда

Расстояние между двумя кластерами определяется как увеличение суммы квадратов ошибок (SSE) при объединении двух кластеров. SSE для данного кластера C_i задается как:

$$SSE_i = \sum_{\boldsymbol{x} \in C_i} \|\boldsymbol{x} - \boldsymbol{\mu}_i\|^2$$

Которая также может быть записана как

$$SSE_i = \sum_{\boldsymbol{x} \in C_i} \|\boldsymbol{x} - \boldsymbol{\mu}_i\|^2$$

$$= \sum_{\boldsymbol{x} \in C_i} \boldsymbol{x}^T \boldsymbol{x} - 2 \sum_{\boldsymbol{x} \in C_i} \boldsymbol{x}^T \boldsymbol{\mu}_i + \sum_{\boldsymbol{x} \in C_i} \boldsymbol{\mu}_i{}^T \boldsymbol{\mu}_i$$

$$= \left(\sum_{\boldsymbol{x} \in C_i} \boldsymbol{x}^T \boldsymbol{x} \right) - n_i \boldsymbol{\mu}_i{}^T \boldsymbol{\mu}_i \tag{9.3}$$

SSE для кластеризации $C = \{C_1, \ldots, C_m\}$ задается как

$$SSE = \sum_{i=1}^{m} SSE_i = \sum_{i=1}^{m} \sum_{\boldsymbol{x} \in C_i} \|\boldsymbol{x} - \boldsymbol{\mu}_i\|^2$$

Мера Уорда определяет расстояние между двумя кластерами C_i и C_j как чистое изменение значения SSE, когда мы объединяем C_i и C_j в C_{ij}

$$\delta(C_i, C_j) = \Delta SSE_{ij} = SSE_{ij} - SSE_i - SSE_j \tag{9.4}$$

Мы можем получить более простое выражение для меры Уорда, подставив уравнение (9.3) в уравнение (9.4) и отмечая, что, поскольку $C_{ij} = C_i \cup C_j$ и $C_i \cap C_j = \emptyset$, мы имеем $|C_{ij}| = n_{ij} = n_i + n_j$, и поэтому

$$\begin{aligned}
&\delta(C_i, C_j) = \Delta SSE_{ij} \\
&= \sum_{\boldsymbol{z} \in C_{ij}} \|\boldsymbol{z} - \boldsymbol{\mu}_{ij}\|^2 - \sum_{\boldsymbol{x} \in C_i} \|\boldsymbol{x} - \boldsymbol{\mu}_i\|^2 - \sum_{\mathbf{y} \in C_j} \|\mathbf{y} - \boldsymbol{\mu}_j\|^2 \\
&= \sum_{\boldsymbol{z} \in C_{ij}} \boldsymbol{z}^T\boldsymbol{z} - n_{ij}\boldsymbol{\mu}_{ij}^T\boldsymbol{\mu}_{ij} - \sum_{\boldsymbol{x} \in C_i} \boldsymbol{x}^T\boldsymbol{x} + n_i\boldsymbol{\mu}_i^T\boldsymbol{\mu}_i - \sum_{\mathbf{y} \in C_j} \mathbf{y}^T\mathbf{y} + n_j\boldsymbol{\mu}_j^T\boldsymbol{\mu}_j \\
&= n_i\boldsymbol{\mu}_i^T\boldsymbol{\mu}_i + n_j\boldsymbol{\mu}_j^T\boldsymbol{\mu}_j - (n_i + n_j)\boldsymbol{\mu}_{ij}^T\boldsymbol{\mu}_{ij}
\end{aligned} \tag{9.5}$$

Последний шаг следует из того, что $\sum_{\boldsymbol{z} \in C_{ij}} \boldsymbol{z}^T\boldsymbol{z} = \sum_{\boldsymbol{x} \in C_i} \boldsymbol{x}^T\boldsymbol{x} + \sum_{\mathbf{y} \in C_j} \mathbf{y}^T\mathbf{y}$. Отмечая, что

$$\boldsymbol{\mu}_{ij} = \frac{n_i\boldsymbol{\mu}_i + n_j\boldsymbol{\mu}_j}{n_i + n_j}$$

мы получаем

$$\boldsymbol{\mu}_{ij}^T\boldsymbol{\mu}_{ij} = \frac{1}{(n_i + n_j)^2}\left(n_i^2\boldsymbol{\mu}_i^T\boldsymbol{\mu}_i + 2n_in_j\boldsymbol{\mu}_i^T\boldsymbol{\mu}_j + n_j^2\boldsymbol{\mu}_j^T\boldsymbol{\mu}_j\right)$$

Подставив вышеизложенное в уравнение (9.5) окончательно получаем

$$\begin{aligned}
&\delta(C_i, C_j) = \Delta SSE_{ij} \\
&= n_i\boldsymbol{\mu}_i^T\boldsymbol{\mu}_i + n_j\boldsymbol{\mu}_j^T\boldsymbol{\mu}_j - \frac{1}{(n_i + n_j)}\left(n_i^2\boldsymbol{\mu}_i^T\boldsymbol{\mu}_i + 2n_in_j\boldsymbol{\mu}_i^T\boldsymbol{\mu}_j + n_j^2\boldsymbol{\mu}_j^T\boldsymbol{\mu}_j\right) \\
&= \frac{n_i(n_i + n_j)\boldsymbol{\mu}_i^T\boldsymbol{\mu}_i + n_j(n_i + n_j)\boldsymbol{\mu}_j^T\boldsymbol{\mu}_j - n_i^2\boldsymbol{\mu}_i^T\boldsymbol{\mu}_i - 2n_in_j\boldsymbol{\mu}_i^T\boldsymbol{\mu}_j - n_j^2\boldsymbol{\mu}_j^T\boldsymbol{\mu}_j}{n_i + n_j} \\
&= \frac{n_in_j(\boldsymbol{\mu}_i^T\boldsymbol{\mu}_i - 2\boldsymbol{\mu}_i^T\boldsymbol{\mu}_j + \boldsymbol{\mu}_j^T\boldsymbol{\mu}_j)}{n_i + n_j} \\
&= \left(\frac{n_in_j}{n_i + n_j}\right)\|\boldsymbol{\mu}_i - \boldsymbol{\mu}_j\|^2
\end{aligned}$$

Таким образом, мера Уорда является взвешенной версией меры среднего расстояния, потому что, если мы используем евклидово расстояние, среднее расстояние в уравнении (9.2) можно переписать как

$$\delta(\boldsymbol{\mu}_i, \boldsymbol{\mu}_j) = \|\boldsymbol{\mu}_i - \boldsymbol{\mu}_j\|^2 \tag{9.6}$$

Мы можем видеть, что единственная разница в том, что мера Уорда взвешивает расстояние между средними значениями как половину гармонического среднего размера кластеров, где гармоническое среднее двух чисел n_1 и n_2 задано как $\frac{2}{\frac{1}{n_1}+\frac{1}{n_2}} = \frac{2n_1n_2}{n_1+n_2}$.

Обновление матрицы расстояний

Каждый раз, когда два кластера C_i и C_j объединяются в C_{ij}, нам необходимо обновить матрицу расстояний путем пересчета расстояний от вновь созданного кластера C_{ij} до всех других кластеров C_r ($r \neq i$ и $r \neq j$). Формула Ланса-Вильямса дает общее уравнение для пересчета расстояний для всех мер кластерной близости, которые мы рассматривали ранее:

$$\delta(C_{ij}, C_r) = \alpha_i \cdot \delta(C_i, C_r) + \alpha_j \cdot \delta(C_j, C_r) + \beta \cdot \delta(C_i, C_j) + \gamma \cdot |\delta(C_i, C_r) - \delta(C_j, C_r)| \quad (9.7)$$

Коэффициенты α_i, α_j, β и γ различаются от одной меры к другой. Пусть $n_i = |C_i|$ обозначает мощность кластера C_i; тогда коэффициенты для различных мер расстояния будут такими, как показано в таблице 8.1.

Таблица 8.1. Формула Ланса-Вильямса для кластерной близости

Measure	α_i	α_j	β	γ
Single link	$\frac{1}{2}$	$\frac{1}{2}$	0	$-\frac{1}{2}$
Complete link	$\frac{1}{2}$	$\frac{1}{2}$	0	$\frac{1}{2}$
Group average	$\frac{n_i}{n_i+n_j}$	$\frac{n_j}{n_i+n_j}$	0	0
Mean distance	$\frac{n_i}{n_i+n_j}$	$\frac{n_j}{n_i+n_j}$	$\frac{-n_i \cdot n_j}{(n_i+n_j)^2}$	0
Ward's measure	$\frac{n_i+n_r}{n_i+n_j+n_r}$	$\frac{n_j+n_r}{n_i+n_j+n_r}$	$\frac{-n_r}{n_i+n_j+n_r}$	0

Вычислительная сложность

В агломеративной кластеризации нам необходимо вычислить расстояние от каждого кластера до всех остальных кластеров, и на каждом шаге количество кластеров уменьшается на 1. Первоначально для создания матрицы попарного расстояния требуется $O(n^2)$ времени, если только оно не указано как вход в алгоритм.

На каждом этапе слияния расстояния от объединенного кластера до других кластеров должны быть пересчитаны, тогда как расстояния между другими кластерами остаются прежними. Это означает, что на шаге t мы вычисляем $O(n-t)$ расстояний. Другая основная операция – найти ближайшую пару в матрице расстояний. Для этого мы можем сохранить n^2 расстояний в структуре данных куча, что позволяет нам найти минимальное расстояние за время $O(1)$; создание кучи занимает время $O(n^2)$. Удаление/обновление расстояний из кучи занимает время $O(\log n)$ для каждой операции, что составляет общее время на всех этапах

слияния $O(n^2 \log n)$. Таким образом, вычислительная сложность иерархической кластеризации составляет $O(n^2 \log n)$.

Глава 10. Кластеризация, основанная на плотности

Методы классификации на основе репрезентативности, такие как метод k-средних и метод максимизации ожидания, применимы для поиска эллипсоидных или, в лучшем случае, выпуклых кластеров. Для невыпуклых кластеров же, например, для показанного на рис. 10.1, эти методы дают ошибочное разделение на кластеры, поскольку точки из разных кластеров могут оказаться ближе, чем точки из одного кластера. Методы, основанные на плотности, которые будут рассмотрены в этой части, способны правильно находить такие кластеры.

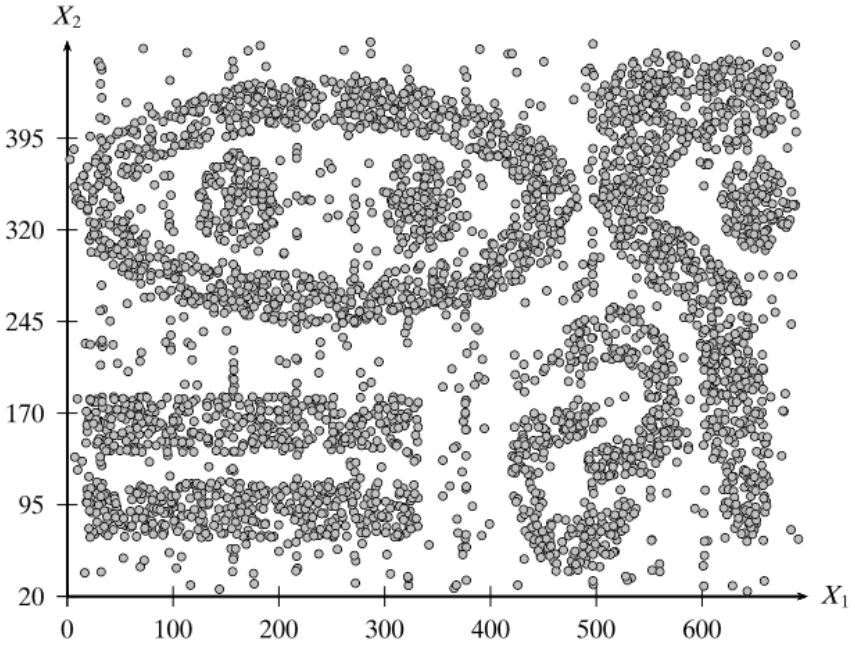


Рисунок 10.1 – Пример датасета для кластеризации, основанной на плотности.

10.1 Алгоритм DBSCAN

Кластеризация, основанная на плотности, использует не расстояние между точками, а локальную плотность точек для определения кластеров. Будем называть *ϵ-соседями* точки $\mathbf{x} \in \mathbb{R}^d$ сферу радиуса ϵ вокруг этой точки, заданную следующим образом:

$$N_\epsilon(\mathbf{x}) = B_d(\mathbf{x}, \epsilon) = \{\mathbf{y} \mid \delta(\mathbf{x}, \mathbf{y}) \leq \epsilon\},$$

где $\delta(\mathbf{x}, \mathbf{y})$ представляет собой расстояние между точками $\mathbf{x}$ и $\mathbf{y}$. Как правило, речь идёт о евклидовом расстоянии, то есть, $\delta(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_2$, но могут применяться и другие метрики.

Мы будем называть точку $\mathbf{x} \in \mathbf{D}$ *базовой (ядерной)*, если хотя бы $minpts$ точек являются её ϵ-соседями. Другими словами, точка $\mathbf{x}$ является базовой, если $|N_\epsilon(\mathbf{x})| \geq minpts$, где $minpts$ – задаваемая пользователем локальная плотность или порог частоты. *Граничной точкой* называется точка, которая не удовлетворяет порогу $minpts$, то есть, для неё $|N_\epsilon(\mathbf{x})| \leq minpts$, но при этом она является ϵ-соседней для некоторой базовой точки $\mathbf{z}$, то есть, $\mathbf{x} \in N_\epsilon(\mathbf{z})$.

Наконец, если точка не является ни базовой, ни граничной, она считается *точкой шума* или выбросом.

Будем считать, что точка $\mathbf{x}$ *напрямую достижима по плотности* из точки $\mathbf{y}$, если $\mathbf{x} \in N_\epsilon(\mathbf{y})$ и $\mathbf{y}$ является базовой точкой. Точка $\mathbf{x}$ *достижима по плотности* из точки $\mathbf{y}$, если существует последовательность точек $\mathbf{x}_0, \mathbf{x}_1, \ldots \mathbf{x}_l$ такая, что $\mathbf{x} = \mathbf{x}_0$, $\mathbf{y} = \mathbf{x}_l$ и $\mathbf{x}_i$ напрямую достижима из $\mathbf{x}_{i-1}$ для всех $i = 1, \ldots, l$. Другими словами, существует множество базовых точек, ведущих от $\mathbf{y}$ к $\mathbf{x}$. Заметим, что отношение достижимости по плотности асимметричное или направленное. Определим любые две точки $\mathbf{x}$ и $\mathbf{y}$ как *связанные по плотности*, если существует базовая точка $\mathbf{z}$ такая, что и $\mathbf{x}$, и $\mathbf{y}$ достижимы из $\mathbf{z}$. Кластером, основанным на плотности, называется максимальное множество точек, связанных по плотности.

Псевдокод основанного на плотности метода кластеризации DBSCAN (Density-based spatial clustering for applications with noise, основанная на плотности пространственная кластеризация для приложений с шумами) приведён в алгоритме 10.1. Сначала DBSCAN вычисляет ϵ-соседей $N_\epsilon(\mathbf{x}_i)$ для всех точек $\mathbf{x}_i$ датасета $\mathbf{D}$, а затем проверяет, являются ли они базовыми (строки 2-5). Кроме того, алгоритм присваивает всем точкам значение идентификатора кластера $id(\mathbf{x}_i) = \emptyset$, отмечая, что они не принадлежат ни одному кластеру. Далее, начиная с каждой базовой точки, не присвоенной ни одному кластеру, метод рекурсивно ищет все точки, связанные по плотности с исходной, и присваивает их одному кластеру (строка 10). Некоторые граничные точки могут быть достижимы из базовых из более чем одного кластера, они могут быть присвоены любому кластеру или всем (если допускается пересечение кластеров). Те точки, которые не принадлежат ни одному кластеру, помечаются как выбросы или шум.

Алгоритм 10.1 – Алгоритм кластеризации, основанный на плотности

DBSCAN (D, ϵ, $minpts$):
1 $Core \leftarrow \emptyset$
2 **foreach** $\mathbf{x}_i \in \mathbf{D}$ **do** // Find the core points
3 Compute $N_\epsilon(\mathbf{x}_i)$
4 $id(\mathbf{x}_i) \leftarrow \emptyset$ // cluster id for $\mathbf{x}_i$
5 **if** $N_\epsilon(\mathbf{x}_i) \geq minpts$ **then** $Core \leftarrow Core \cup \{\mathbf{x}_i\}$
6 $k \leftarrow 0$ // cluster id
7 **foreach** $\mathbf{x}_i \in Core$, *such that* $id(\mathbf{x}_i) = \emptyset$ **do**
8 $k \leftarrow k + 1$
9 $id(\mathbf{x}_i) \leftarrow k$ // assign $\mathbf{x}_i$ to cluster id k
10 DENSITYCONNECTED $(\mathbf{x}_i, k)$
11 $\mathcal{C} \leftarrow \{C_i\}_{i=1}^k$, where $C_i \leftarrow \{\mathbf{x} \in \mathbf{D} \mid id(\mathbf{x}) = i\}$
12 $Noise \leftarrow \{\mathbf{x} \in \mathbf{D} \mid id(\mathbf{x}) = \emptyset\}$
13 $Border \leftarrow \mathbf{D} \setminus \{Core \cup Noise\}$
14 **return** $\mathcal{C}, Core, Border, Noise$

DENSITYCONNECTED (x, k):
15 **foreach** $\mathbf{y} \in N_\epsilon(\mathbf{x})$ **do**
16 $id(\mathbf{y}) \leftarrow k$ // assign $\mathbf{y}$ to cluster id k
17 **if** $\mathbf{y} \in Core$ **then** DENSITYCONNECTED $(\mathbf{y}, k)$

DBSCAN можно рассматривать как поиск связных компонент в графе, где вершины соответствуют базовым точкам в датасете, и существует (ненаправленное) ребро между двумя вершинами (базовыми точками), если расстояние между ними меньше ϵ, то есть, каждая из них является ϵ-соседом для другой. Связанные компоненты такого графа соответствуют базовым точкам каждого кластера. Далее, каждая точка включает в свой кластер все граничные точки, которые являются её соседями.

Единственное ограничение DBSCAN – чувствительность к выбору ϵ, в частности, в тех случаях, когда кластеры имеют разную плотность. Если выбрать ϵ слишком маленьким, более разреженные кластеры будут классифицированы как шум. Если значение ϵ слишком велико, более плотные кластеры могут быть слиты в один. Иными словами, если в датасете есть кластеры с разными локальными плотностями, то единственного значения ϵ может быть недостаточно.

Вычислительная сложность

Основные вычислительные затраты DBSCAN – это вычисление всех ϵ-соседей каждой точки. Если размерность датасета невелика, это можно эффективно сделать, используя структуру пространственного индекса за время $O(n \log(n))$. Если размерность велика, вычисление соседей каждой точки требует времени $O(n^2)$. После вычисления $N_\epsilon(\mathbf{x})$ алгоритму нужен только один проход по всем точкам для поиска связанных по плотности кластеров. Таким образом, вычислительная сложность алгоритма DBSCAN в худшем случае – $O(n^2)$.

10.2 Оценка плотности ядра

Существует тесная связь между кластеризацией на основе плотности и оценкой плотности. Целью оценки плотности является определение неизвестной функции плотности вероятности путем нахождения плотных областей точек, которые, в свою очередь, могут использоваться для кластеризации. Оценка плотности ядра — это непараметрический метод, который не предполагает какой-либо фиксированной вероятностной модели кластеров, как в случае K-средних или модели смеси, принятой в алгоритме EM. Вместо этого он пытается напрямую вывести базовую плотность вероятности в каждой точке набора данных.

10.2.1 Одномерная оценка плотности

Предположим, что X является непрерывной случайной величиной, и пусть x_1, $x_2, \ldots, x_n$ будет случайной выборкой, взятой из лежащей в основе функции плотности вероятности f(x), которая предполагается неизвестной. Мы можем напрямую оценить кумулятивную функцию распределения по данным, посчитав, сколько точек меньше или равно x:

$$\hat{F}(x) = \frac{1}{n}\sum_{i=1}^{n} I(x_i \leq x)$$

где I – индикаторная функция, которая имеет значение 1 только тогда, когда ее аргумент истинен, и 0 в противном случае. Мы можем оценить функцию плотности, взяв производную от $\hat{F}(x)$, рассматривая окно небольшой ширины h с центром в точке x, то есть:

$$\hat{f}(x) = \frac{\hat{F}\left(x + \frac{h}{2}\right) - \hat{F}\left(x - \frac{h}{2}\right)}{h} = \frac{\frac{k}{n}}{h} = \frac{k}{nh} \tag{10.1}$$

где k – количество точек, которые лежат в окне шириной h с центром в x, то есть в закрытом интервале $[x - \frac{h}{2},\ x\ +\ \frac{h}{2}]$. Таким образом, оценка плотности – это отношение доли точек в окне (k/n) к объему окна (h). Здесь h играет роль «влияния». То есть при большом h оценивается плотность вероятности в большом окне с учетом множества точек, что приводит к сглаживанию оценки. С другой стороны, если h мало, то рассматриваются только точки в непосредственной близости от x. В общем, нам нужно небольшое значение h, но не слишком маленькое, так как в этом случае никакие точки не попадут в окно, и мы не сможем получить точную оценку плотности вероятности.

Оценщик ядра

Оценка плотности ядра основывается на функции ядра K, которая является неотрицательной, симметричной и интегрируется с 1, то есть $K(x) \geq 0, K(-x) = K$для всех значений x и $\int K(x)dx = 1$. Таким образом, K – это, по сути, функция плотности вероятности. Обратите внимание, что K не следует путать с положительно полуопределенным ядром..

Дискретное ядро оценку плотности $\hat{f}(x)$ из уравнения (10.1) также можно переписать в терминах функции ядра следующим образом:

$$\hat{f}(x) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right)$$

где **дискретная функция ядра** K вычисляет количество точек в окне шириной h и определяется как:

$$K(z) = \begin{cases} 1 \text{ If } |z| \leq \frac{1}{2} \\ 0 \text{ В противном случае} \end{cases} \tag{10.2}$$

Мы видим, что если $|z| = \left|\frac{x - x_i}{h}\right| \leq \frac{1}{2}$, то точка x_i находится в пределах окна шириной h с центром в x, так как

$$\left|\frac{x - x_i}{h}\right| \leq \frac{1}{2} \text{ подразумевает что } -\frac{1}{2} \leq \frac{x - x_i}{h} \leq \frac{1}{2}, \text{или}$$
$$-\frac{h}{2} \leq x - x_i \leq \frac{h}{2}, \text{и наконец}$$
$$x - \frac{h}{2} \leq x_i \leq x + \frac{h}{2}$$

Пример 10.1.

На рисунке 10.2 показаны оценки плотности ядра с использованием дискретного ядра для различных значений параметра влияния h для одномерного набора данных Iris,

содержащего атрибут длины чашелистника. По оси абсцисс отложены n = 150 точек данных. Поскольку несколько точек имеют одно и то же значение, они отображаются в стопке, где высота стопки соответствует частоте этого значения.

Когда h мало, как показано на рисунке 10.2a, функция плотности имеет много локальных максимумов или мод. Однако, когда мы увеличиваем h с 0,25 до 2, количество мод уменьшается, пока h не станет достаточно большим, чтобы получить унимодальное распределение, как показано на рисунке 10.2d. Мы можем заметить, что дискретное ядро дает негладкую (или зубчатую) функцию плотности.

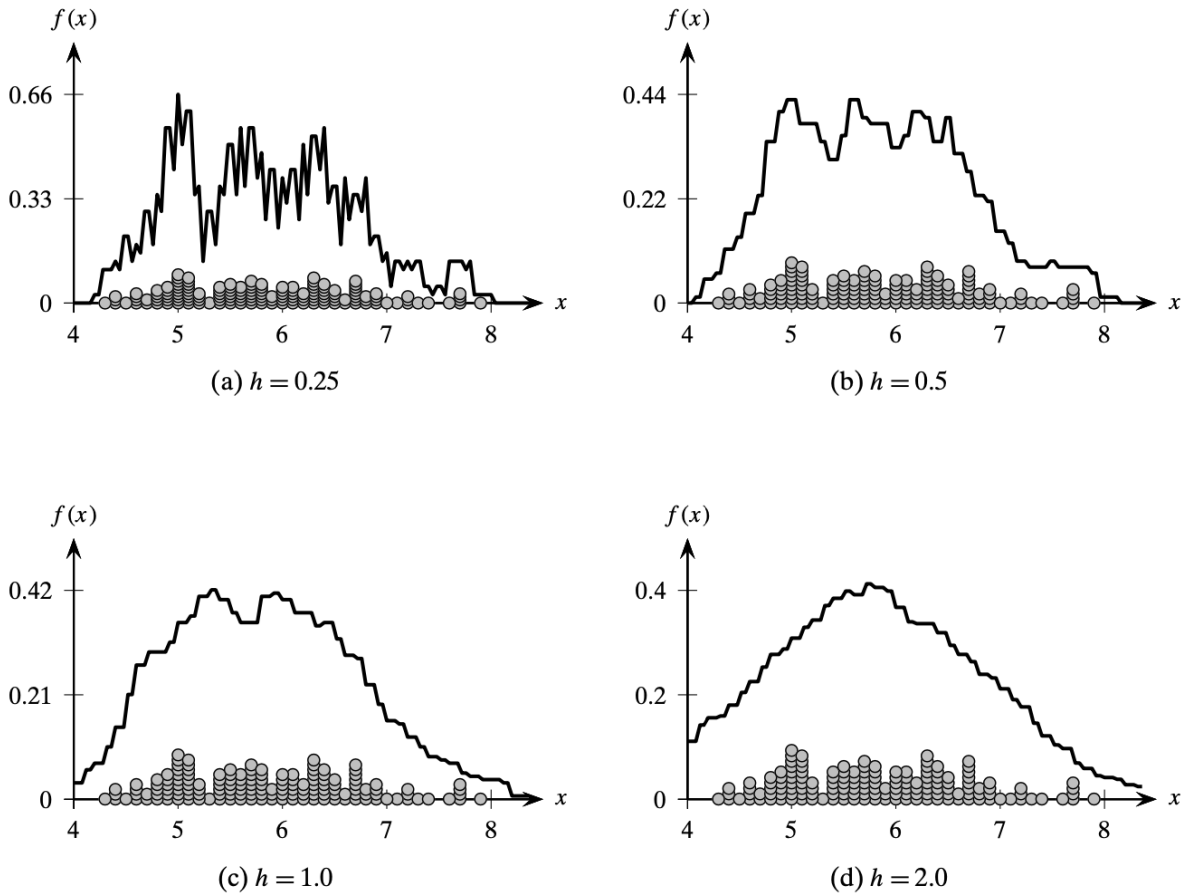


Рисунок 10.2 Оценка плотности ядра: дискретное ядро (изменяющееся h).

Гауссово ядро ширина h – параметр, обозначающий разброс или гладкость оценки плотности. Если разброс слишком большой, мы получаем более усредненное значение. Если он слишком маленький, нам не хватает точек в окне. Кроме того, функция ядра в уравнении (10.2) оказывает большое влияние. Для точек внутри окна ($|z| \leq \frac{1}{2}$) существует чистый вклад в оценку вероятности $\hat{f}(x)$ составляет $\frac{1}{nh}$. С другой стороны, точки вне окна ($|z| > \frac{1}{2}$) дают 0.

Вместо дискретного ядра мы можем определить более плавный переход влияния через гауссово ядро:

$$K(z) = \frac{1}{\sqrt{2\pi}} \exp\left\{-\frac{z^2}{2}\right\}$$

Таким образом, мы имеем

$$K\left(\frac{x - x_i}{h}\right) = \frac{1}{\sqrt{2\pi}} \exp\left\{-\frac{(x - x_i)^2}{2h^2}\right\}$$

Здесь x, который находится в центре окна, играет роль среднего, а h действует как стандартное отклонение.

Пример 10.2.

На рис. 10.3 показана одномерная функция плотности для одномерного набора данных Iris (по длине чашелистика) с использованием ядра Гаусса. Показаны графики для возрастающих значений параметра разброса h. Точки данных показаны сгруппированными по оси x с высотой, соответствующей частотам значений.

Поскольку h изменяется от 0,1 до 0,5, мы можем видеть сглаживающий эффект увеличения h на функцию плотности. Например, при h = 0,1 имеется много локальных максимумов, а при h = 0,5 - только один пик плотности. По сравнению со случаем дискретного ядра, показанным на рисунке 10.2, мы можем ясно видеть, что ядро Гаусса дает гораздо более гладкие оценки без разрывов.

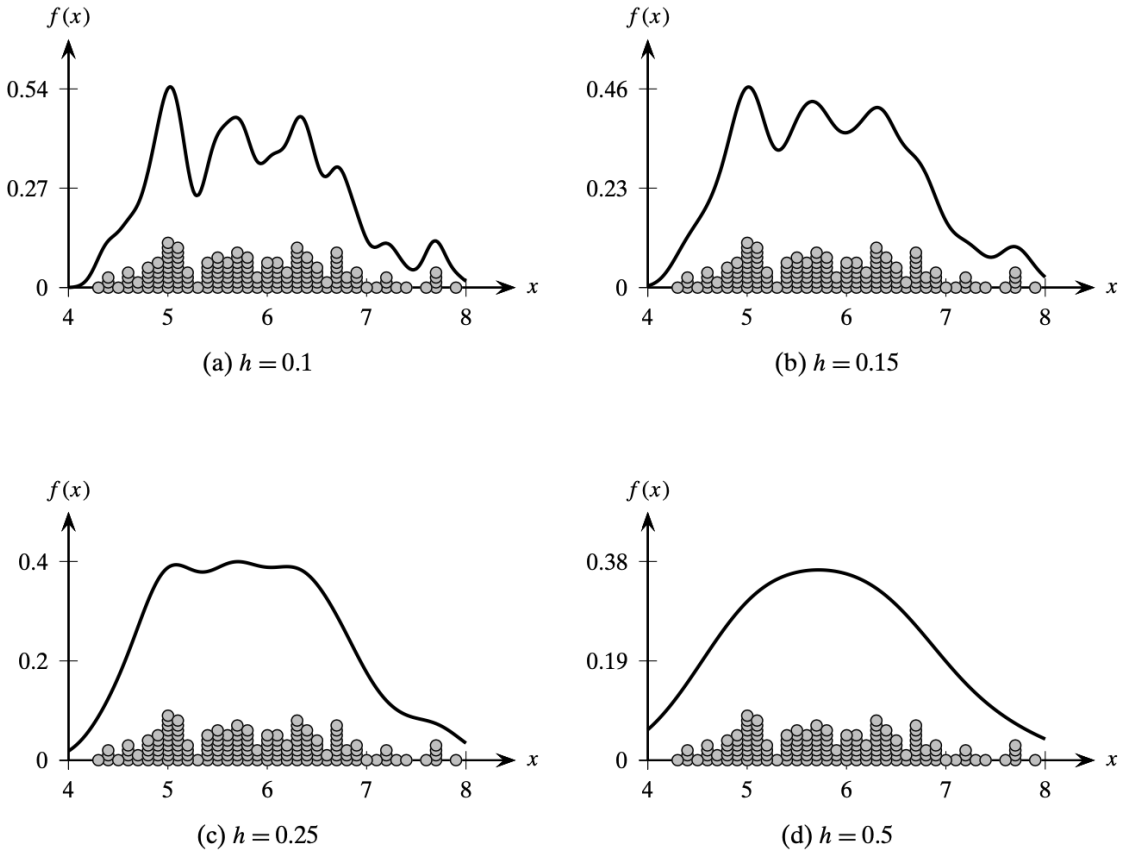


Рисунок 10.3 Оценка плотности ядра: ядро Гаусса (изменяющееся h).

10.2.2 Многомерная оценка плотности

Чтобы оценить плотность вероятности в d-мерной точке $\mathrm{x} = (\mathrm{x}_1, \mathrm{x}_2, \dots, \mathrm{x}_\mathrm{n})^\mathrm{T}$, мы определяем d-мерное «окно» как гиперкуб в d измерениях, то есть гиперкуб с центром в x с длиной ребра h. Объем такого d-мерного гиперкуба задается как

$$\mathrm{vol}\big(\mathrm{H}_\mathrm{d}(\mathrm{h})\big) = \mathrm{h}^\mathrm{d}$$

Затем плотность оценивается как доля веса точки, лежащая в d-мерном окне с центром в x, деленная на объем гиперкуба:

$$\hat{\mathrm{f}}(\mathbf{x}) = \frac{1}{\mathrm{nh}^2}\sum_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{K}\left(\frac{\mathbf{x}-\mathbf{x_i}}{\mathrm{h}}\right) \tag{10.3}$$

где функция многомерного ядра K удовлетворяет условию $\int \mathrm{K}(\mathbf{z})\mathrm{d}\mathbf{z} = 1$

Дискретное ядро для любого d-мерного вектора $\mathrm{x} = (\mathrm{x}_1, \mathrm{x}_2, \dots, \mathrm{x}_\mathrm{n})^\mathrm{T}$ дискретная ядерная функция в d-измерениях задается как

$$\mathrm{K}(\mathrm{z}) = \begin{cases} 1 \text{ If } |\mathrm{z}_\mathrm{j}| \le \frac{1}{2}, \text{для всех размеров } \mathrm{j} = 1, \dots, \mathrm{d} \\ 0 \text{ В противном случае} \end{cases}$$

Для $\mathrm{z} = \frac{\mathrm{x}-\mathrm{x_i}}{\mathrm{h}}$ мы видим, что ядро вычисляет количество точек внутри гиперкуба с центром в x, потому что $\mathrm{K}\left(\frac{\mathrm{x}-\mathrm{x_i}}{\mathrm{h}}\right) = 1$, тогда и только тогда $\left|\frac{\mathrm{x_j}-\mathrm{x_{ij}}}{\mathrm{h}}\right| \le \frac{1}{2}$ для всех размеров j. Таким образом, каждая точка в гиперкубе дает вес $\frac{1}{\mathrm{n}}$ для оценки плотности.

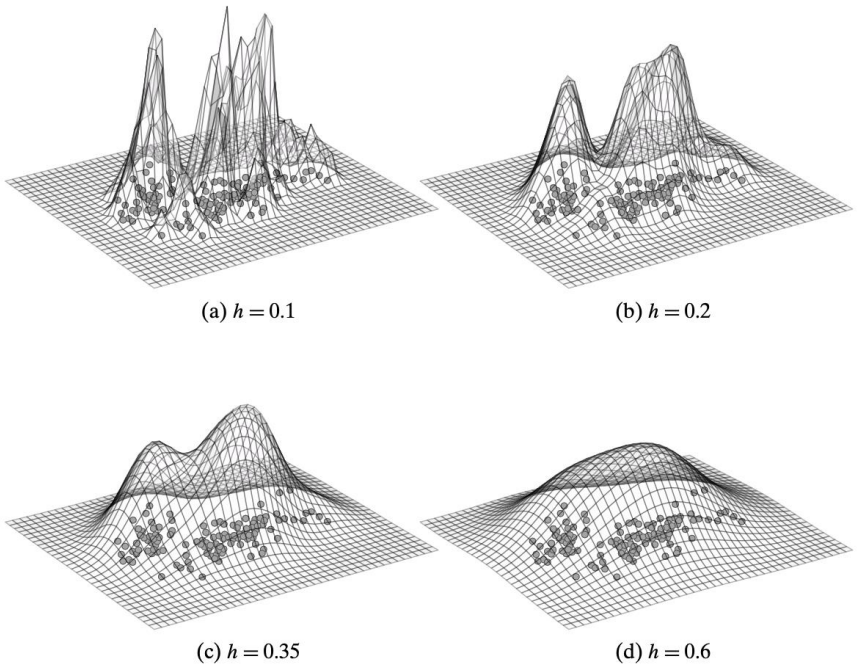


Рисунок 10.4 Оценка плотности: набор данных 2D Iris (различается по h).

Гауссово ядро d-мерное гауссово ядро задается как

$$K(\mathbf{z}) = \frac{1}{(2\pi)^{\frac{d}{2}}} \exp\left\{-\frac{\mathbf{z}^T\mathbf{z}}{2}\right\} \tag{10.4}$$

где мы предполагаем, что ковариационная матрица является единичной матрицей $d \times d$, то есть $\mathbf{\Sigma} = \mathbf{I}_d$. Подставляя $\mathbf{z} = \frac{x - x_i}{h}$ в уравнение (10.4), получаем

$$K\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right) = \frac{1}{(2\pi)^{d/2}} \exp\left\{-\frac{(\mathbf{x} - \mathbf{x}_i)^T(\mathbf{x} - \mathbf{x}_i)}{2h^2}\right\}$$

Каждая точка вносит свой вклад в оценку плотности, обратно пропорциональную ее расстоянию от $\mathbf{x}$, уменьшенному параметром ширины h.

Пример 10.3.

На рисунке 10.5 показана функция плотности вероятности для набора данных 2D радужной оболочки глаза, содержащего атрибуты длины и ширины чашелистика, с использованием ядра Гаусса. Как и ожидалось, для малых значений h функция плотности имеет несколько локальных максимумов, тогда как для больших значений количество максимумов уменьшается, и в конечном итоге для достаточно большого значения мы получаем унимодальное распределение.

Пример 10.4.

На рисунке 10.6 показана оценка плотности ядра для набора данных на основе плотности на рисунке 10.1 с использованием ядра Гаусса с h = 20. Можно четко различить, что пики плотности близко соответствуют областям с более высокой плотностью точек.

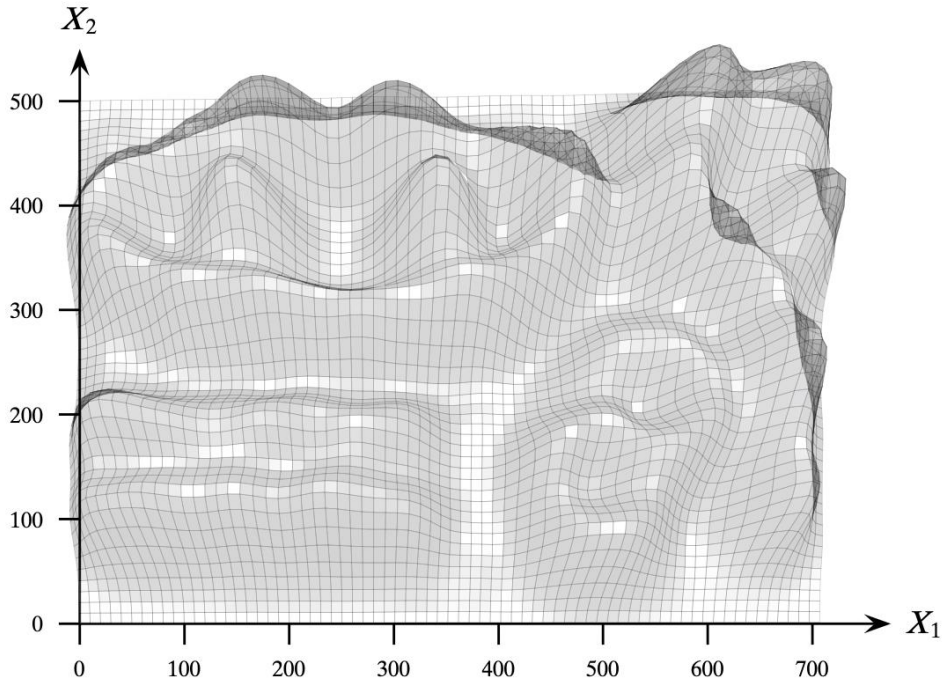


Рисунок 10.6 Оценка плотности: набор данных на основе плотности.

10.2.3 Оценка плотности ближайшего соседа

В предыдущей формулировке оценки плотности мы неявно фиксировали объем, фиксируя ширину h, и мы использовали функцию ядра, чтобы узнать количество или вес точек, которые лежат внутри области фиксированного объема. Альтернативный подход к оценке плотности состоит в том, чтобы зафиксировать k, количество точек, необходимых для оценки плотности, и позволить изменяться объему окружающей области для размещения этих k точек. Этот подход называется подходом k ближайших соседей (KNN) к оценке плотности. Как и оценка плотности ядра, оценка плотности KNN также является непараметрическим подходом.

Учитывая k, количество соседей, мы оцениваем плотность в точке x следующим образом:

$$\hat{f}(\mathbf{x}) = \frac{k}{n\,\mathrm{vol}\left(S_d(h_{\mathbf{x}})\right)}$$

где $h_{\mathbf{x}}$ – это расстояние от $\mathbf{x}$ до k-го ближайшего соседа, а $\mathrm{vol}\left(S_d(h_{\mathbf{x}})\right)$ – объем d-мерной гиперсферы $S_d(h_{\mathbf{x}})$ с центром в $\mathbf{x}$ и радиусом $h_{\mathbf{x}}$ [уравнение (6.4)]. Другими словами, ширина (или радиус) $h_{\mathbf{x}}$ теперь является переменной, которая зависит от $\mathbf{x}$ и выбранного значения k.

10.3 Кластеризация на основе плотности: DENCLUE

Заложив основы ядерной оценки плотности, мы можем разработать общую формулировку кластеризации на основе плотности. Основной подход состоит в том, чтобы найти пики в ландшафте плотности с помощью оптимизации на основе градиента и найти области с плотностью выше заданного порога.

Аттракторы плотности и градиент

Точка $\boldsymbol{x}^*$ называется аттрактором плотности, если она является локальным максимумом функции плотности вероятности *f*. Аттрактор плотности можно найти с помощью подхода градиентного подъема, начиная с некоторой точки $\boldsymbol{x}$. Идея состоит в том, чтобы вычислить градиент плотности, направление наибольшего увеличения плотности, и двигаться в направлении градиента небольшими шагами, пока мы не достигнем локальных максимумов.

Градиент в точке $\boldsymbol{x}$ может быть вычислен как многомерная производная оценки плотности вероятности в уравнении (10.3) в виде

$$\nabla\hat{f}(\boldsymbol{x}) = \frac{\partial}{\partial \boldsymbol{x}}\hat{f}(\boldsymbol{x}) = \frac{1}{nh^2}\sum_{i=1}^{n}\frac{\partial}{\partial \boldsymbol{x}}K\left(\frac{\boldsymbol{x}-\boldsymbol{x_i}}{h}\right) \tag{10.5}$$

Для ядра Гаусса [Ур. (10.4)], имеем

$$\frac{\partial}{\partial \boldsymbol{x}}K(\boldsymbol{z}) = \left(\frac{1}{(2\pi)^{\frac{d}{2}}}exp\left\{-\frac{\boldsymbol{z}^T\boldsymbol{z}}{2}\right\}\right)\cdot -z\cdot\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}} = K(\boldsymbol{z})\cdot -z\cdot\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}}$$

Полагая $\boldsymbol{z} = \frac{\boldsymbol{x}-\boldsymbol{x}_i}{h}$ выше, получаем

$$\frac{\partial}{\partial \boldsymbol{x}}K\left(\frac{\boldsymbol{x}-\boldsymbol{x}_i}{h}\right) = K\left(\frac{\boldsymbol{x}-\boldsymbol{x}_i}{h}\right)\cdot\left(\frac{\boldsymbol{x}-\boldsymbol{x}_i}{h}\right)\cdot\left(\frac{1}{h}\right)$$

что следует из того, что $\frac{\partial}{\partial \boldsymbol{x}}\left(\frac{\boldsymbol{x}-\boldsymbol{x}_i}{h}\right) = \left(\frac{1}{h}\right)$. Подставляя вышеуказанное в формулу (10.5) градиент в точке $\boldsymbol{x}$ задается как

$$\nabla\hat{f}(\boldsymbol{x}) = \frac{1}{nh^{d+2}}\sum_{i=1}^{n}\frac{\partial}{\partial \boldsymbol{x}}K\left(\frac{\boldsymbol{x}-\boldsymbol{x_i}}{h}\right)\cdot(\boldsymbol{x}_i-\boldsymbol{x}) \tag{10.6}$$

Это уравнение можно представить как состоящее из двух частей: вектора $(\boldsymbol{x}_i-\boldsymbol{x})$ и скалярного значения *влияния* $K\left(\frac{\boldsymbol{x}-\boldsymbol{x}_i}{h}\right)$. Для каждой точки $\boldsymbol{x}_i$ мы сначала вычисляем направление от $\boldsymbol{x}$, то есть вектор $(\boldsymbol{x}_i-\boldsymbol{x})$. Затем мы масштабируем его, используя значение ядра Гаусса в качестве веса $K\left(\frac{\boldsymbol{x}-\boldsymbol{x}_i}{h}\right)$. Наконец, вектор $\nabla\hat{f}(\boldsymbol{x})$ является чистым влиянием в точке x, как показано на рисунке 10.7, то есть взвешенной суммой векторов разности.

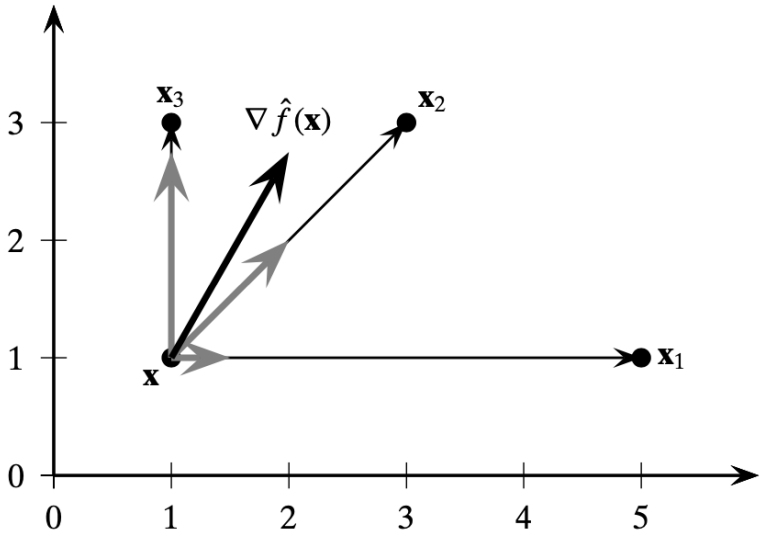


Рисунок 10.7 Вектор градиента $\nabla\hat{f}(x)$ (показан толстым черным), полученный как сумма векторов разности $(x_i - x)$ (показан серым).

Мы говорим, что $\boldsymbol{x}^*$ является аттрактором плотности для $\boldsymbol{x}$ или, альтернативно, что $\boldsymbol{x}$ является плотностью, сходящейся к $\boldsymbol{x}^*$, если процесс восхождения на холм, начатый в $\boldsymbol{x}$, сходится к $\boldsymbol{x}^*$. То есть существует последовательность точек $\boldsymbol{x} = \boldsymbol{x}_0 \to \boldsymbol{x}_1 \to \cdots \to \boldsymbol{x}_m$, начиная с $\boldsymbol{x}$ и заканчивая $\boldsymbol{x}_m$, такое, что $\left|\left|\boldsymbol{x}_m - \boldsymbol{x}^*\right|\right| \leq \epsilon$, то есть $\boldsymbol{x}_m$ сходится к аттрактору $\boldsymbol{x}^*$.

Типичный подход заключается в использовании метода градиентного подъема для вычисления $\boldsymbol{x}^*$, то есть, начиная с $\boldsymbol{x}$, мы итеративно обновляем его на каждом шаге t с помощью правила обновления:

$$x_{t+1} = x_t + \delta \cdot \nabla\hat{f}(\boldsymbol{x})$$

где $\delta > 0$ – размер шага. То есть каждая промежуточная точка получается после небольшого перемещения в направлении вектора градиента. Однако подход градиентного подъема может медленно сходиться. Вместо этого можно напрямую оптимизировать направление движения, установив градиент [Ур. (10.6)] к нулевому вектору:

$$\nabla\hat{f}(\boldsymbol{x}) = 0$$

$$\frac{1}{nh^{d+2}}\sum_{i=1}^{n} K\left(\frac{\boldsymbol{x} - \boldsymbol{x_i}}{h}\right) \cdot (\boldsymbol{x}_i - \boldsymbol{x}) = 0$$

$$\boldsymbol{x} \cdot \sum_{i=1}^{n} K\left(\frac{\boldsymbol{x} - \boldsymbol{x_i}}{h}\right) = \sum_{i=1}^{n} K\left(\frac{\boldsymbol{x} - \boldsymbol{x_i}}{h}\right)\boldsymbol{x}_i$$

$$\boldsymbol{x} = \frac{\sum_{i=1}^{n} K\left(\frac{\boldsymbol{x} - \boldsymbol{x_i}}{h}\right)\boldsymbol{x}_i}{\sum_{i=1}^{n} K\left(\frac{\boldsymbol{x} - \boldsymbol{x_i}}{h}\right)}$$

Точка $\boldsymbol{x}$ задействована как слева, так и справа выше. Однако его можно использовать для получения следующего правила итеративного обновления:

$$\boldsymbol{x}_{t+1} = \frac{\sum_{i=1}^{n} K\left(\frac{\boldsymbol{x}_t - \boldsymbol{x}_i}{h}\right) \boldsymbol{x}_i}{\sum_{i=1}^{n} K\left(\frac{\boldsymbol{x}_t - \boldsymbol{x}_i}{h}\right)} \tag{10.7}$$

где t обозначает текущую итерацию, а $\boldsymbol{x}_{t+1}$ - обновленное значение для текущего вектора $\boldsymbol{x}_t$. Это правило прямого обновления является, по сути, средневзвешенным влиянием (вычисленным с помощью функции ядра K) каждой точки $\boldsymbol{x}_i \in \mathbf{D}$ на текущую точку $\boldsymbol{x}_t$. Правило прямого обновления приводит к гораздо более быстрой сходимости процесса подъема на холм.

Центральный кластер

Кластер $\boldsymbol{C} \subseteq \mathbf{D}$ называется центральным кластером, если все точки x $\in C$ являются плотностью, сходящейся к единственному аттрактору плотности $\boldsymbol{x}^*$, такому, что $\hat{f}(\boldsymbol{x}^*) \geq \xi$, где ξ – минимальная пороговая плотность, определяемый пользователем. Другими словами,

$$\hat{f}(\boldsymbol{x}^*) = \frac{1}{nh^d} \sum_{i=1}^{n} K\left(\frac{\boldsymbol{x} - \boldsymbol{x}_i}{h}\right) \geq \xi$$

Кластер на основе плотности

Кластер $\boldsymbol{C} \subseteq \mathbf{D}$ произвольной формы называется кластером на основе плотности, если существует набор плотностных аттракторов $x_1^*, x_2^*, \ldots, x_m^*$, таких что

1. Каждая точка $\boldsymbol{x} \in \boldsymbol{C}$ притягивается к некоторому аттрактору $\boldsymbol{x}^*$.
2. Каждый аттрактор плотности имеет плотность выше ξ. То есть $\hat{f}(\boldsymbol{x}^*) \geq \xi$.
3. Любые два аттрактора плотности x_i^* и x_j^* достижимы по плотности, то есть существует путь из x_i^* в x_j^* такой, что для всех точек **y** на пути $\hat{f}(\boldsymbol{x}^*) \geq \xi$.

Алгоритм DENCLUE

Псевдокод для DENCLUE показан в алгоритме 10.2. Первым шагом является вычисление аттрактора плотности $\boldsymbol{x}^*$ для каждой точки $\boldsymbol{x}$ в наборе данных (строка 4). Если плотность в $\boldsymbol{x}^*$ выше минимального порога плотности ξ, аттрактор добавляется к набору аттракторов A. Точка данных $\boldsymbol{x}$ также добавляется к набору точек $\boldsymbol{R}(\boldsymbol{x}^*)$, сходящихся к $\boldsymbol{x}^*$ (строка 9). На втором этапе DENCLUE находит все максимальные подмножества аттракторов $\boldsymbol{C} \subseteq A$, такие, что любая пара аттракторов в $\boldsymbol{C}$ является достижимыми по плотности друг из друга (строка 11). Эти максимальные подмножества взаимно достижимых аттракторов образуют начало для каждого кластера на основе плотности. Наконец, для каждого аттрактора $\boldsymbol{x}^* \in \boldsymbol{C}$ мы добавляем к кластеру все точки $\boldsymbol{R}(\boldsymbol{x}^*)$, которые сходятся к $\boldsymbol{x}^*$, что приводит к окончательному набору кластеров $\boldsymbol{C}$.

Алгоритм 10.2: DENCLUE

DENCLUE $(\mathbf{D}, h, \xi, \epsilon)$:

$\mathcal{A} \leftarrow \emptyset$

foreach $\mathbf{x} \in \mathbf{D}$ **do** // find density attractors

  $\mathbf{x}^* \leftarrow \text{FINDATTRACTOR}(\mathbf{x}, \mathbf{D}, h, \epsilon)$

  if $\hat{f}(\mathbf{x}^*) \geq \xi$ **then**

    $\mathcal{A} \leftarrow \mathcal{A} \cup \{\mathbf{x}^*\}$

    $R(\mathbf{x}^*) \leftarrow R(\mathbf{x}^*) \cup \{\mathbf{x}\}$

$\mathcal{C} \leftarrow \{\text{maximal } C \subseteq \mathcal{A} \mid \forall \mathbf{x}_i^*, \mathbf{x}_j^* \in C, \mathbf{x}_i^* \text{ and } \mathbf{x}_j^* \text{ are density reachable}\}$

foreach $C \in \mathcal{C}$ **do** // density-based clusters

  foreach $\mathbf{x}^* \in C$ **do** $C \leftarrow C \cup R(\mathbf{x}^*)$

return $\mathcal{C}$

FINDATTRACTOR $(\mathbf{x}, \mathbf{D}, h, \epsilon)$:

$t \leftarrow 0$

$\mathbf{x}_t \leftarrow \mathbf{x}$

repeat

  $\mathbf{x}_{t+1} \leftarrow \frac{\sum_{i=1}^{n} K\left(\frac{\mathbf{x}_t - \mathbf{x}_i}{h}\right) \cdot \mathbf{x}_t}{\sum_{i=1}^{n} K\left(\frac{\mathbf{x}_t - \mathbf{x}_i}{h}\right)}$

  $t \leftarrow t + 1$

until $\|\mathbf{x}_t - \mathbf{x}_{t-1}\| \leq \epsilon$

return $\mathbf{x}_t$

Метод FINDATTRACTOR реализует процесс восхождения с использованием правила прямого обновления [Ур. (10.7)], что приводит к быстрой сходимости. Чтобы еще больше ускорить вычисление влияния, можно вычислить значения ядра только для ближайших соседей $\boldsymbol{x}_t$. То есть мы можем проиндексировать точки в наборе данных **D**, используя структуру пространственного индекса, чтобы мы могли быстро вычислить всех ближайших соседей $\boldsymbol{x}_t$ в пределах некоторого радиуса r. Для ядра Гаусса мы можем установить $r = h \cdot z$, где h - параметр влияния, который играет роль стандартного отклонения, а z указывает количество стандартных отклонений. Пусть $B_d(\boldsymbol{x}_t, r)$ обозначает множество всех точек в **D**, которые лежат внутри d-мерного шара радиуса r с центром в $\boldsymbol{x}_t$. Правило обновления на основе ближайшего соседа может быть выражено как

$$\boldsymbol{x}_{t+1} = \frac{\sum_{\boldsymbol{x}_i \in B_d(\boldsymbol{x}_t, r)} K\left(\frac{\boldsymbol{x}_t - \boldsymbol{x}_i}{h}\right) \boldsymbol{x}_i}{\sum_{\boldsymbol{x}_i \in B_d(\boldsymbol{x}_t, r)} K\left(\frac{\boldsymbol{x}_t - \boldsymbol{x}_i}{h}\right)}$$

который можно использовать в строке 20 алгоритма 10.2. Когда размерность данных невысока, это может привести к значительному ускорению. Однако эффективность быстро ухудшается с увеличением числа измерений. Это связано с двумя эффектами. Во-первых, поиск $B_d(\boldsymbol{x}_t, r)$ сводится к линейному сканированию данных, занимающему время $O(n)$ для каждого запроса. Во-вторых, из-за проклятия размерности (см. Главу 6) почти все точки кажутся одинаково близкими к $\boldsymbol{x}_t$, тем самым сводя на нет любые преимущества вычисления ближайших соседей.

Пример 10.5

На рисунке 10.8 показана кластеризация DENCLUE для двухмерного набора данных Iris, включающего атрибуты длины и ширины чашелистника. Результаты были получены при $h = 0{,}2$ и $\xi = 0{,}08$ с использованием ядра Гаусса. Кластеризация получается путем определения порога функции плотности вероятности на рисунке 10.4b при $\xi = 0{,}08$. Два пика соответствуют двум последним кластерам. В то время как *iris setosa* хорошо разделена, трудно разделить два других типа *Ирисов*.

Пример 10.6

На рисунке 10.9 показаны кластеры, полученные с помощью DENCLUE на основе набора данных по плотности из рисунка 10.1. Используя параметры h = 10 и $\xi = 9.5 \times 10^{-5}$, с гауссовым ядром, мы получаем восемь кластеров. Рисунок получен путем разрезания функции плотности на значение плотности ξ; отображаются только области выше этого значения. Все кластеры идентифицированы правильно, за исключением двух полукруглых кластеров в правом нижнем углу, которые кажутся объединенными в один кластер.

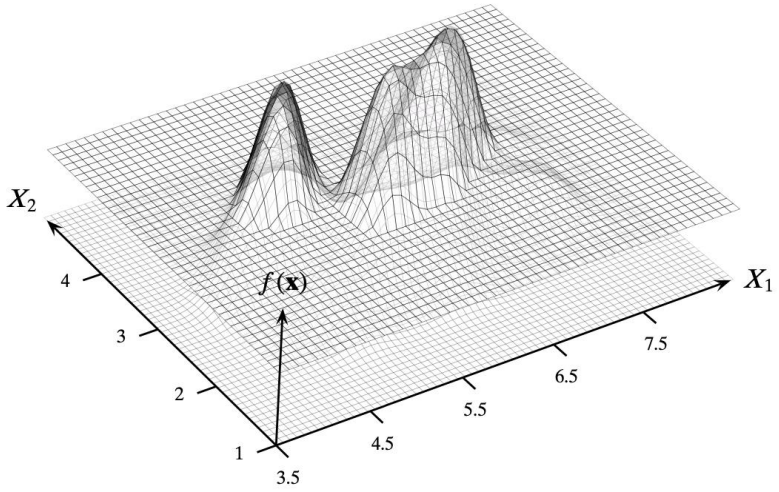


Рисунок 10.8 Ирис 2D датасет

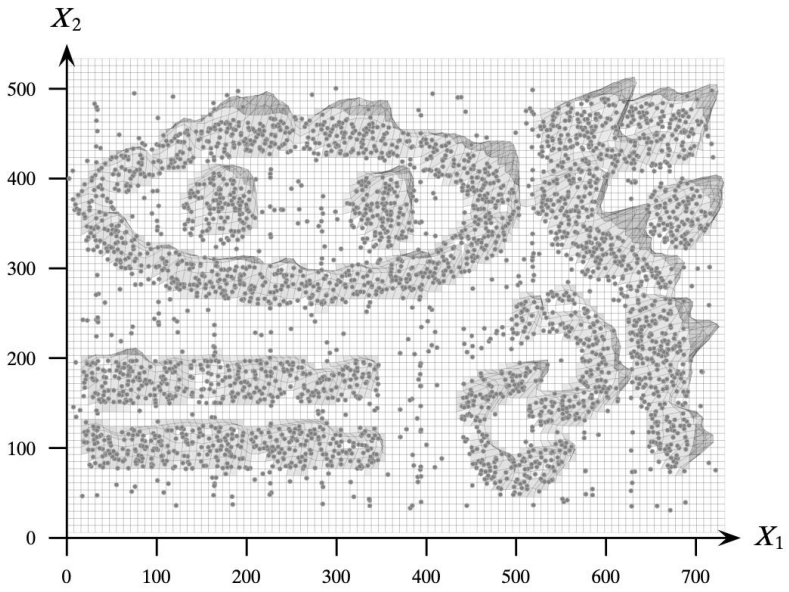


Рисунок 10.9 Датасет на основе плотности

DENCLUE: Особые случаи

Можно показать, что DBSCAN является частным случаем подхода к кластеризации на основе общей оценки плотности ядра, DENCLUE. Если мы положим $h = \in$ и ξ = *minpts*, то использование дискретного ядра DENCLUE дает точно такие же кластеры, как DBSCAN. Каждый аттрактор плотности соответствует центральной точке, а набор связанных основных точек определяет аттракторы кластера на основе плотности. Также можно показать, что K-средних является частным случаем кластеризации на основе плотности для соответствующих значений h и ξ, с аттракторами плотности, соответствующими центроидам кластеров. Кроме того, стоит отметить, что подход на основе плотности может создавать иерархические кластеры путем изменения порога ξ. Например, уменьшение ξ может привести к слиянию нескольких кластеров, обнаруженных при более высоких значениях порогов. В то же время это также может приводить к новым кластерам, если пиковая плотность удовлетворяет более низкому значению ξ.

Вычислительная сложность

Время выполнения для DENCLUE является наименьшим из-за стоимости процесса восхождения. Для каждой точки $\boldsymbol{x} \in \mathbf{D}$ поиск аттрактора плотности занимает время $O(nt)$, где t - максимальное количество итераций подъема на холм. Это связано с тем, что каждая итерация занимает $O(n)$ времени для вычисления суммы функции влияния по всем точкам $\boldsymbol{x}_i \in$ D. Таким образом, общая стоимость вычисления аттракторов плотности составляет $O(n^2t)$. Мы предполагаем, что при разумных значениях h и ξ имеется всего несколько аттракторов плотности, то есть $|A| = m \ll n$. Стоимость поиска максимально достижимых подмножеств аттракторов составляет $O(m^2)$, а конечные кластеры могут быть получены за $O(n)$ время.

Глава 11. Валидация кластеризации

Существует множество различных методов кластеризации в зависимости от типа искомых кластеров и присущих им характеристик данных. Учитывая разнообразие алгоритмов кластеризации и их параметров, важно разработать объективные подходы к оценке результатов кластеризации. Проверка и оценка кластера включает в себя три основные задачи: *оценка кластеризации* направлена на оценку совершенства или качества кластеризации, *стабильность кластеризации* стремится понять чувствительность результата кластеризации к различным алгоритмическим параметрам, например, количеству кластеров, и *оценка тенденции кластеризации* – пригодность применения кластеризации в первую очередь, то есть, есть ли у данных внутренняя структура группировки. Для каждой из вышеупомянутых задач предлагается ряд критериев достоверности и статистики, которые можно разделить на три основных типа:

Внешний: внешние меры проверки используют критерии, которые не являются неотъемлемыми для набора данных. Это может быть в форме предшествующих или определенных экспертами знаний о кластерах, например, меток классов для каждой точки.

Внутренний: меры внутренней проверки используют критерии, которые выводятся из самих данных. Например, мы можем использовать внутрикластерные и межкластерные расстояния для получения показателей компактности кластера (например, насколько похожи точки в одном кластере?) И разделения (например, насколько далеко друг от друга находятся точки в разных кластерах?).

Относительный: меры относительной проверки направлены на прямое сравнение различных кластеров, обычно полученных с помощью разных настроек параметров для одного и того же алгоритма.

В этой главе мы изучаем некоторые из основных методов кластеризации валидации и оценки, охватывающей все три типа показателей.

11.1 Внешние меры

Как следует из названия, внешние меры предполагают, что правильная или достоверная кластеризация известна априори. Истинные метки кластера играют роль внешней информации, которая используется для оценки данной кластеризации. В общем, мы не знаем правильной кластеризации; однако внешние меры могут служить способом проверки и подтверждения различных методов. Например, наборы данных классификации, которые определяют класс для каждой точки, могут использоваться для оценки качества кластеризации. Аналогичным образом, синтетические наборы данных с известной кластерной структурой могут быть созданы для оценки различных алгоритмов кластеризации путем количественной оценки степени, в которой они могут восстанавливать известные группировки.

Пусть $\boldsymbol{D} = \{\boldsymbol{x}_i\}_{i=1}^n$ - набор данных, состоящий из n точек в d-мерном пространстве, разбитых на k кластеров. Пусть $y_i \in \{1, 2, \dots, k\}$ обозначают истинную кластеризацию или информацию о метке кластера для каждой точки. Истинная кластеризация задается как $T = \{T_1, T_2, \dots, T_k\}$, где кластер T_j состоит из всех точек с меткой j, т.е. $T_j = \{\boldsymbol{x}_i \in \boldsymbol{D} | \, y_i = j\}$. Кроме того, пусть $C = \{C_1, C_2, \dots, C_r\}$ обозначает кластеризацию того же набора данных в r кластеров, полученных с помощью некоторого алгоритма кластеризации, и пусть $\hat{y}_i \in \{1, 2, \dots, r\}$ обозначает метку кластера для $\boldsymbol{x}_i$. Для ясности в дальнейшем мы будем называть T истинным

разделением, а каждый T_i – разделом. Мы будем называть C кластеризацией, причем каждый C_i назовем кластером. Поскольку предполагается, что достоверная информация известна, обычно методы кластеризации будут выполняться с правильным числом кластеров, то есть с $r = k$. Однако, чтобы сохранить более общее обсуждение, мы позволяем r отличаться от k.

Меры внешней оценки пытаются зафиксировать степень, в которой точки из одного раздела появляются в одном кластере, и степень, в которой точки из разных разделов сгруппированы в разные кластеры. Обычно существует компромисс между этими двумя целями, который либо явно фиксируется мерой, либо подразумевается при ее вычислении. Все внешние меры полагаются на $r \times k$ *таблицу сопряженности* **N**, которая индуцируется кластеризацией C и истинным разделением T, определяемая следующим образом

$$\boldsymbol{N}(i,j) = n_{ij} = |C_i \cap T_j|$$

Другими словами, счетчик n_{ij} обозначает количество точек, общих для кластера C_i и истинного разделения $\boldsymbol{T}_j$. Далее, для ясности положим $n_i = |C_i|$ обозначает количество точек в кластере C_i, и пусть $m_j = |T_j|$ обозначает количество точек в разбиении T_j. Таблица сопряженности может быть вычислена из T и C за время $O(n)$, исследуя метки разделов и кластеров, y_i и $\hat{y}_i$, для каждой точки $\boldsymbol{x}_i \in \boldsymbol{D}$ и увеличивая соответствующий счетчик $n_{y_i \hat{y}_i}$.

11.1.1 Соответствующие меры

Чистота

Чистота определяет степень, в которой кластер C_i содержит объекты только из одного раздела. Другими словами, он измеряет, насколько «чистым» является каждый кластер. Чистота кластера C_i определяется как

$$purity_i = \frac{1}{n_i} \max_{j=1}^{k} \{n_{ij}\}$$

Чистота кластеризации C определяется как взвешенная сумма значений чистоты по кластерам:

$$purity = \sum_{i=1}^{r} \frac{n_i}{n} purity_i = \frac{1}{n} \sum_{i=1}^{r} \max_{j=1}^{k} \{n_{ij}\}$$

где отношение n_i обозначает долю точек в кластере C_i. Чем выше чистота C, тем лучше согласие с истинным разделением. Максимальное значение чистоты - 1, когда каждый кластер содержит точки только из одного раздела. Когда $r = k$, значение чистоты 1 указывает на идеальную кластеризацию с взаимно-однозначным соответствием между кластерами и разделами. Однако чистота может быть равна 1 даже при $r > k$, когда каждый из кластеров является подмножеством разбиения на основе истинности. Когда $r < k$, чистота никогда не может быть 1, потому что хотя бы один кластер должен содержать точки из более чем одного раздела.

Максимальное соответствие

Мера максимального соответствия выбирает отображение между кластерами и разделами, так что сумма числа общих точек (n_{ij}) максимизируется, при условии, что только один кластер может соответствовать заданному разделу. Это отличается от чистоты, когда два разных кластера могут иметь один и тот же мажоритарный раздел.

Формально мы рассматриваем таблицу сопряженности как полный взвешенный двудольный граф $G = (V, E)$, где каждое разбиение и кластер является узлом, то есть $V = C \cup T$, и существует ребро $(C_i, T_j) \in E$ с весом $w(C_i, T_j) = n_{ij}$ для всех $C_i \in C$ и $T_j \in T$. Совпадение M в G – это подмножество E, такое, что ребра в M попарно несмежные, то есть они не имеют общей вершины. Максимальная мера соответствия определяется как *максимальное соответствие веса* в G:

$$match = \underset{M}{argmax}\left\{\frac{w(M)}{n}\right\}$$

где вес совпадающего M – это просто сумма всех весов ребер в M, заданная как $w(M) = \sum_{e \in M} w(e)$. Максимальное совпадение может быть вычислено за время $O(|V|^2 \cdot |E|) = O((r+k)^2 rk)$, что эквивалентно $O(k^4)$, если $r = O(k)$.

F-Мера

Учитывая кластер C_i, пусть j_i обозначает разбиение, содержащее максимальное количество точек из C_i, то есть $j_i = max^k{}_{j=1}\{n_{ij}\}$. *Точность* кластера C_i такая же, как и его чистота:

$$prec_i = \frac{1}{n_i} \underset{j=1}{\overset{k}{max}}\{n_{ij}\} = \frac{n_{ij_i}}{n_i}$$

Он измеряет долю точек в C_i от мажоритарного разбиения T_{ij}.

Отзыв кластера C_i определяется как

$$recall_i = \frac{n_{ij_i}}{|T_{j_i}|} = \frac{n_{ij_i}}{m_{j_i}}$$

где $m_{j_i} = |T_{j_i}|$. Это измеряет долю точки в разделе T_{j_i}, совместно используемую с кластером C_i.

F-мера – это среднее гармоническое значение точности и отзыва для каждого кластера. Следовательно, F-мера для кластера C_i имеет вид

$$F_i = \frac{2}{\frac{1}{prec_i} + \frac{1}{recall_i}} = \frac{2 \cdot prec_i \cdot recall_i}{prec_i + recall_i} = \frac{2n_{ij_i}}{n_i + m_{j_i}} \tag{11.1}$$

F-мера для кластеризации C – это среднее значение F-меры по кластерам:

$$F = \frac{1}{r}\sum_{i=1}^{r} F_i$$

Таким образом, F-мера пытается сбалансировать точность и отзыв значения по всем кластерам. Для идеальной кластеризации, когда $r = k$, максимальное значение F-меры равно 1.

Пример 11.1

На рисунке 11.1 показаны две разные кластеризации, полученные с помощью алгоритма K-средних для набора данных Iris, с использованием первых двух основных компонентов в качестве двух измерений. Здесь $n = 150$ и $k = 3$. Визуальный осмотр подтверждает, что рисунок 11.1a лучше кластеризован, чем рисунок 11.1b. Теперь мы исследуем, как различные меры, основанные на таблицах сопряженности, могут быть использованы для оценки этих двух кластеров.

Рассмотрим кластеризацию на рисунке 11.1a. Три кластера обозначены разными символами; серые точки находятся в правильном разделе, тогда как белые неправильно сгруппированы по сравнению с типами наземных ирисовых диафрагм. Например, C_3 в основном соответствует разделу T_3 (Iris-virginica), но у него есть три точки (белые треугольники) от T_2. Полная таблица сопряженности выглядит следующим образом:

	iris-setosa T_1	iris-versicolor T_2	iris-virginica T_3	n_i
C_1(squares)	0	47	14	61
C_2(circles)	50	0	0	50
C_3(triangles)	0	3	36	39
m_j	50	50	50	$n = 150$

Чтобы вычислить чистоту, мы сначала отмечаем для каждого кластера раздел с максимальным перекрытием. У нас есть соответствие (C_1, T_2), (C_2, T_1) и (C_3, T_3). Таким образом, чистота выражается как

$$purity = \frac{1}{150}(47 + 50 + 36) = \frac{133}{150} = 0.887$$

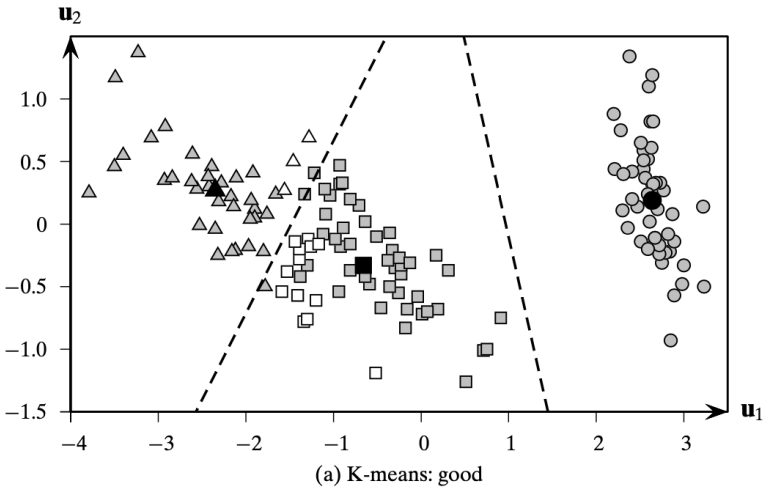
Для этой таблицы сопряженности мера максимального соответствия дает тот же результат, поскольку приведенное выше соответствие фактически является сопоставлением максимального веса. Таким образом, *match* = 0.887.

Кластер C_1 содержит $n_1 = 47 + 14 = 61$ точек, тогда как соответствующее ему разбиение T_2 содержит $m_2 = 47 + 3 = 50$ точек. Таким образом, точность и отзыв для C_1 задаются как

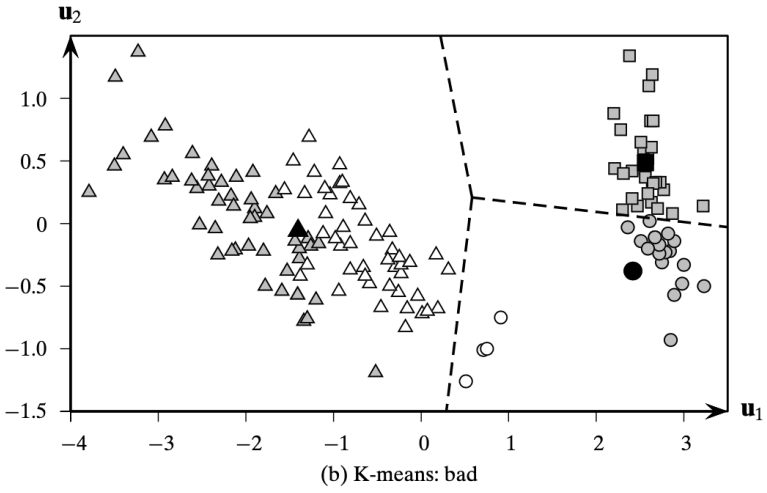
$$prec_1 = \frac{47}{61} = 0.77$$
$$recall_1 = \frac{47}{50} = 0.94$$

Следовательно, F-мера для C_1 есть

$$F_1 = \frac{2 \cdot 0.77 \cdot 0.94}{0.77 + 0.94} = \frac{1.45}{1.71} = 0.85$$



(a) K-means: good



(b) K-means: bad

Рисунок 11.1 K-средних: набор данных основных компонентов Iris

Мы также можем напрямую вычислить F1, используя уравнение (11.1)

$$F_1 = \frac{2 \cdot n_{12}}{n_1 + m_2} = \frac{2 \cdot 47}{61 + 50} = \frac{94}{111} = 0.85$$

Аналогичным образом получаем F_2 = 1,0 и F_3 = 0,81. Таким образом, значение F-меры для кластеризации задается как

$$F = \frac{1}{3}(F_1 + F_2 + F_3) = \frac{2.66}{3} = 0.88$$

Для кластеризации на рисунке 11.1b у нас есть следующая таблица сопряженности:

	iris-setosa T_1	iris-versicolor T_2	iris-virginica T_3	n_i
C_1	30	0	0	30
C_2	20	4	0	24
C_3	0	46	50	96
m_j	50	50	50	$n = 150$

Для меры чистоты раздел, с которым каждый кластер имеет наибольшее количество точек, задается как (C_1, T_1), (C_2, T_1) и (C_3, T_3). Таким образом, значение чистоты для этой кластеризации равно

$$purity = \frac{1}{150}(30 + 20 + 50) = \frac{100}{150} = 0.67$$

Мы видим, что как C_1, так и C_2 выбирают раздел T_1 как максимально перекрывающийся раздел. Однако соответствие максимального веса отличается; это дает соответствие (C_1, T_1), (C_2, T_2) и (C_3, T_3), и, таким образом

$$match = \frac{1}{150}(30 + 4 + 50) = \frac{84}{150} = 0.56$$

В таблице ниже сравниваются различные меры на основе сопряженности для двух кластеров, показанных на рисунке 11.1.

	purity	*match*	*F*
(a) Good	0.887	0.887	0.885
(b) Bad	0.667	0.560	0.658

Как и ожидалось, хорошая кластеризация на рис. 11.1a имеет более высокие баллы по чистоте, максимальному соответствию и F-мере.

11.1.2 Меры, основанные на энтропии

Условная энтропия

Энтропия кластеризации *C* определяется как

$$H(C) = -\sum_{i=1}^{r} p_{C_i} \log p_{C_i}$$

где $p_{C_i} = \frac{n_i}{n}$ – вероятность кластера C_i. Аналогично энтропия разбиения *T* определяется как

$$H(T) = -\sum_{j=1}^{k} p_{T_j} \log p_{T_j}$$

где p_{T_j} – вероятность разбиения T_j.

Специфическая для кластера энтропия T, то есть условная энтропия T относительно кластера C_i, определяется как

$$H(T|C_i) = -\sum_{j=1}^{k}\left(\frac{n_{ij}}{n_i}\right)\log\left(\frac{n_{ij}}{n_i}\right)$$

Условная энтропия T с учетом кластеризации C затем определяется как взвешенная сумма:

$$H(T|C) = \sum_{i=1}^{r}\frac{n_i}{n}H(T|C_i) = -\sum_{i=1}^{r}\sum_{j=1}^{k}\frac{n_{ij}}{n}\log\left(\frac{n_{ij}}{n_i}\right) = -\sum_{i=1}^{r}\sum_{j=1}^{k}p_{ij}\log\left(\frac{p_{ij}}{p_{C_i}}\right) \quad (11.2)$$

где $p_{ij} = \frac{n_{ij}}{n}$ – вероятность того, что точка в кластере *i* также принадлежит разделу *j*. Чем больше членов кластера разбито на различные разделы, тем выше условная энтропия. Для идеальной кластеризации значение условной энтропии равно нулю, тогда как наихудшее возможное значение условной энтропии – $\log k$. Далее, расширяя уравнение (11.2), мы видим, что

$$H(T|C) = -\sum_{i=1}^{r}\sum_{j=1}^{k}p_{ij}\left(\log p_{ij} - \log p_{C_i}\right) = -\left(\sum_{i=1}^{r}\sum_{j=1}^{k}p_{ij}\log p_{ij}\right) + \sum_{i=1}^{r}\left(\log p_{C_i}\sum_{j=1}^{k}p_{ij}\right) =$$

$$= -\sum_{i=1}^{r}\sum_{j=1}^{k}p_{ij}\log p_{ij} + \sum_{j=1}^{k}p_{C_i}\log p_{C_i} = H(C,T) - H(C) \quad (11.3)$$

где $H(T|C) = -\sum_{i=1}^{r}\sum_{j=1}^{k}p_{ij}\log p_{ij}$ – совместная энтропия C и T. Условная энтропия $H(T|C)$, таким образом, измеряет оставшуюся энтропию T при кластеризации C. В частности, $H(T|C) = 0$ тогда и только тогда, когда T полностью определяется C, что соответствует идеальной кластеризации. С другой стороны, если C и T независимы друг от друга, то $H(T|C) = H(T)$, что означает, что C не предоставляет никакой информации о T.

Нормализованная Взаимная Информация

Взаимная информация пытается количественно определить количество общей информации между кластеризацией C и разделением T, и она определяется как

$$I(C,T) = \sum_{i=1}^{r}\sum_{j=1}^{k}p_{ij}\log\left(\frac{p_{ij}}{p_{C_i}\cdot p_{T_j}}\right) \quad (11.4)$$

Он измеряет зависимость между наблюдаемой совместной вероятностью p_{ij} C и T и ожидаемой совместной вероятностью $p_{C_i}\cdot p_{T_j}$ в предположении независимости. Когда C и T

являются независимыми тогда $p_{ij} = p_{C_i} \cdot p_{T_j}$, и, таким образом, $I(C,T) = 0$. Однако верхней границы взаимной информации не существует.

Расширяя уравнение (11.4), заметим, что $I(C,T) = H(C) + H(T) - H(C,T)$. Используя уравнение (11.3), получаем два эквивалентных выражения:

$$I(C,T) = H(T) - H(T|C)$$

$$I(C,T) = H(C) - H(C|T)$$

Наконец, поскольку $H(C|T) \geq 0$ и $H(T|C) \geq 0$, имеем неравенства $I(C,T) \leq H(C)$ и $I(C,T) \leq H(T)$. Мы можем получить нормализованную версию взаимной информации, рассматривая отношения $I(C,T)$ / $H(C)$ и $I(C,T)$ / $H(T)$, оба из которых могут превзойти большинство из них. *Нормализованная взаимная информация* (NMI) определяется как среднее геометрическое этих двух соотношений:

$$NMI(C,T) = \sqrt{\frac{I(C,T)}{H(C)} \cdot \frac{I(C,T)}{H(T)}} = \frac{I(C,T)}{\sqrt{H(C) \cdot H(T)}}$$

Значение NMI находится в диапазоне [0,1]. Значения, близкие к 1, указывают на хорошую кластеризацию.

Вариация информации

Этот критерий основан на взаимной информации между кластеризацией *C* и истинным разбиением *T* и их энтропии; это определяется как

$$VI(C,T) = \big(H(T) - I(C,T)\big) + \big(H(C) - I(C,T)\big) = H(T) + H(C) - 2I(C,T) \qquad (11.5)$$

Вариация информации (VI) равна нулю только тогда, когда *C* и *T* идентичны. Таким образом, чем ниже значение VI, тем лучше кластеризация *C*.

Используя эквивалентность $I(C,T) = H(T) - H(T|C) = H(C) - H(C|T)$, мы также можем выразить уравнение (11.5) как

$$VI(C,T) = H(T|C) + H(C|T)$$

Наконец, учитывая, что $H(T|C) = H(T|C) - H(C)$, другое выражение для VI дается как

$$VI(C,T) = 2H(T,C) - H(T) - H(T)$$

Пример 11.2

Мы продолжаем с примера 2, в котором сравниваются две кластеризации, показанные на рисунке 11.1. Для мер, основанных на энтропии, мы используем основание 2 для логарифмов; формулы действительны для любой базы как таковой.

Для кластеризации на рисунке 11.1a у нас есть следующая таблица сопряженности:

	iris-setosa T_1	iris-versicolor T_2	iris-virginica T_3	n_i
C_1	0	47	14	61
C_2	50	0	0	50
C_3	0	3	36	39
m_j	50	50	50	$n = 100$

Рассмотрим условную энтропию для кластера C_1:

$$H(T, C_1) = -\frac{0}{61}\log_2\left(\frac{0}{61}\right) - \frac{47}{61}\log_2\left(\frac{47}{61}\right) - \frac{14}{61}\log_2\left(\frac{14}{61}\right) = -0 - 0.77\log_2(0.77) - 0.23\log_2(0.23) = 0.29 + 0.49 = 0.78$$

Аналогичным образом получаем $H(T, C_2)$ = 0 и $H(T, C_3)$ = 0,39. Условная энтропия для кластеризации *C* тогда задается как

$$H(T|C) = \frac{61}{150}\cdot 0.78 + \frac{50}{150}\cdot 0 + \frac{30}{150}\cdot 0.39 = 0.32 + 0 + 0.10 = 0.42$$

Чтобы вычислить нормализованную взаимную информацию, обратите внимание, что

$$H(T) = -3\left(\frac{50}{150}\log_2\left(\frac{50}{150}\right)\right) = 1.585$$

$$H(C) = -\left(\frac{61}{150}\log_2\left(\frac{61}{150}\right) + \frac{50}{150}\log_2\left(\frac{50}{150}\right) + \frac{39}{150}\log_2\left(\frac{39}{150}\right)\right) = 0.528 + 0.528 + 0.505 = 1.561$$

$$I(C,T) = \frac{47}{150}\log_2\left(\frac{47\cdot 150}{61\cdot 50}\right) + \frac{14}{150}\log_2\left(\frac{14\cdot 150}{61\cdot 50}\right) + \frac{50}{150}\log_2\left(\frac{50\cdot 150}{50\cdot 50}\right) + \frac{3}{150}\log_2\left(\frac{3\cdot 150}{39\cdot 50}\right) + \frac{36}{150}\log_2\left(\frac{36\cdot 150}{39\cdot 50}\right) = 0.379 - 0.05 + 0.528 - 0.042 + 0.353 = 1.167$$

Таким образом, значения NMI и VI равны

$$NMI(C,T) = \frac{I(C,T)}{\sqrt{H(C)\cdot H(T)}} = \frac{1.167}{\sqrt{1.585 \times 1.561}} = 0.742$$

$$VI(C,T) = H(T) + H(C) - 2I(C,T) = 1.585 + 1.561 - 2\cdot 1.161 = 0.812$$

Мы можем точно так же вычислить эти показатели для другой кластеризации на рисунке 11.1b, таблица сопряженности которой показана в примере 2.

В таблице ниже сравниваются меры, основанные на энтропии, для двух кластеров, показанных на рисунке 11.1.

	$H(\mathcal{T}\|\mathcal{C})$	*NMI*	*VI*
(a) Good	0.418	0.742	0.812
(b) Bad	0.743	0.587	1.200

Как и ожидалось, хорошая кластеризация на рисунке 11.1a имеет более высокий балл для нормализованной взаимной информации и более низкий балл для условной энтропии и вариативности информации.

11.1.3 Парные меры

Учитывая кластеризацию *C* и исходное разбиение *T*, парные меры используют информацию о разделах и метках кластера по всем парам точек. Пусть $\boldsymbol{x}_i, \boldsymbol{x}_j \in \boldsymbol{D}$ - любые две точки, $i \neq j$. Пусть y_i обозначает истинную метку разбиения, а $\hat{y}_i$ обозначает метку кластера для точки $\boldsymbol{x}_i$. Если и $\boldsymbol{x}_i$ и $\boldsymbol{x}_j$ принадлежат одному кластеру, то есть $\hat{y}_i = \hat{y}_j$, мы называем это *положительным* событием, а если они не принадлежат одному кластеру, то есть $\hat{y}_i \neq \hat{y}_j$, мы называем это *отрицательным* событием. В зависимости от того, согласованы ли метки кластера и метки разделов, можно рассмотреть четыре возможности:

- Истинно положительные значения: $\boldsymbol{x}_i$ и $\boldsymbol{x}_j$ принадлежат одному и тому же разделу в *T*, и они также находятся в одном кластере в *C*. Это истинно положительная пара, потому что положительное событие, $\hat{y}_i = \hat{y}_j$, соответствует основной истине, $y_i = y_j$. Количество истинно положительных пар определяется как

$$TP = \left|\left\{\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right) : y_i = y_j \; and \; \hat{y}_i = \hat{y}_j\right\}\right|$$

- Ложно отрицательные: $\boldsymbol{x}_i$ и $\boldsymbol{x}_j$ принадлежат одному и тому же разделу в *T*, но они не принадлежат одному и тому же кластеру в *C*. То есть отрицательное событие, $\hat{y}_i \neq \hat{y}_j$, не соответствует истине, $y_i = y_j$. Таким образом, эта пара является ложноотрицательной, а количество всех ложноотрицательных пар определяется как

$$FN = \left|\left\{\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right) : y_i \neq y_j \; and \; \hat{y}_i \neq \hat{y}_j\right\}\right|$$

- Ложно положительные: $\boldsymbol{x}_i$ и $\boldsymbol{x}_j$ не принадлежат одному и тому же разделу в *T*, но они принадлежат одному и тому же кластеру в *C*. Эта пара является ложноположительной, потому что положительное событие $\hat{y}_i = \hat{y}_j$ на самом деле ложно, т. Е. не согласуется с разделением основной истины, которое указывает, что $y_i \neq y_j$. Количество ложноположительных пар определяется как

$$FP = \left|\left\{\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right) : y_i \neq y_j \; and \; \hat{y}_i = \hat{y}_j\right\}\right|$$

- Истинно отрицательные: $\boldsymbol{x}_i$ и $\boldsymbol{x}_j$ не принадлежат одному и тому же разделу в *T*, и они не принадлежат одному и тому же кластеру в *C*. Таким образом, эта пара является истинно отрицательным, то есть $\hat{y}_i \neq \hat{y}_j$ и $y_i \neq y_j$. Количество таких истинно отрицательных пар определяется как

$$TN = \left|\left\{\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right) : y_i \neq y_j \; and \; \hat{y}_i \neq \hat{y}_j\right\}\right|$$

Поскольку имеется $N = \binom{n}{2} = \frac{n(n-1)}{2}$ пар точек, мы имеем следующую идентичность:

$$N = TP + FN + FP + TN \tag{11.6}$$

Наивное вычисление предыдущих четырех случаев времени $O(n^2)$. Однако их можно вычислить более эффективно, используя таблицу сопряженности $\boldsymbol{N} = \{n_{ij}\}$, где $1 \leq i \leq r$ и $1 \leq j \leq k$. Количество истинных положительных результатов определяется как

$$TP = \sum_{i=1}^{r}\sum_{j=1}^{k}\binom{n_{ij}}{2} = \sum_{i=1}^{r}\sum_{j=1}^{k}\frac{n_{ij}(n_{ij}-1)}{2} = \frac{1}{2}\left(\sum_{i=1}^{r}\sum_{j=1}^{k}n_{ij}^2 - \sum_{i=1}^{r}\sum_{j=1}^{k}n_{ij}\right) =$$
$$= \frac{1}{2}\left(\left(\sum_{i=1}^{r}\sum_{j=1}^{k}n_{ij}^2\right) - n\right) \tag{11.7}$$

Это следует из того факта, что каждая пара точек среди n_{ij} имеет одну и ту же метку кластера (i) и одну и ту же метку раздела (j). Последний шаг следует из того факта, что сумма всех записей в таблице сопряженности должна быть добавлена к n, то есть, $\sum_{i=1}^{r}\sum_{j=1}^{k}n_{ij} = n$.

Чтобы вычислить общее количество ложноотрицательных результатов, мы удаляем количество истинных положительных результатов из числа пар, принадлежащих одному разделу. Поскольку две точки $\boldsymbol{x}_i$ и $\boldsymbol{x}_j$, принадлежащие одному и тому же разделу, имеют $y_i = y_j$, если мы удалим истинные положительные точки, то есть пары с $\hat{y}_i = \hat{y}_j$, у нас останутся пары, для которых $\hat{y}_i \neq \hat{y}_j$, то есть ложно негативные. Таким образом, мы имеем

$$FN = \sum_{j=1}^{k}\binom{m_j}{2} - TP = \frac{1}{2}\left(\sum_{j=1}^{k}m_j^2 - \sum_{j=1}^{k}m_j - \sum_{i=1}^{r}\sum_{j=1}^{k}n_{ij}^2 + n\right) =$$
$$= \frac{1}{2}\left(\sum_{j=1}^{k}m_j^2 - \sum_{i=1}^{r}\sum_{j=1}^{k}n_{ij}^2\right) \tag{11.8}$$

Последний шаг следует из того, что $\sum_{j=1}^{k}m_j = n$

Количество ложных срабатываний можно получить аналогичным образом, вычтя количество истинных срабатываний из количества пар точек, находящихся в одном кластере:

$$FP = \sum_{i=1}^{r}\binom{n_i}{2} - TP = \frac{1}{2}\left(\sum_{i=1}^{r}n_i^2 - \sum_{i=1}^{r}\sum_{j=1}^{k}n_{ij}^2\right) \tag{11.9}$$

Наконец, количество истинно негативных срабатываний может быть получено с помощью уравнения (11.6) следующим образом:

$$TN = N - (TP + FN + FP) = \frac{1}{2}\left(n^2 - \sum_{i=1}^{r}n_i^2 - \sum_{j=1}^{k}m_j^2 + \sum_{i=1}^{r}\sum_{j=1}^{k}n_{ij}^2\right) \tag{11.10}$$

Каждое из четырех значений может быть вычислено за время $O(rk)$. Поскольку таблица сопряженности может быть получена за линейное время, общее время для вычисления четырех значений составляет $O(n + rk)$, что намного лучше, чем наивная граница $O(n^2)$. Далее мы рассмотрим попарные меры оценки, основанные на этих четырех значениях.

Коэффициент Жаккара

Коэффициент Жаккара измеряет долю истинно положительных пар точек, но после игнорирования истинных отрицательных. Это определяется следующим образом:

$$Jaccard = \frac{TP}{TP + FN + FP} \tag{11.11}$$

Для идеальной кластеризации C (т.е. полного согласия с разбиением T) коэффициент Жаккара имеет значение 1, так как в этом случае нет ложных срабатываний или ложных отрицаний. Коэффициент Жаккара асимметричен с точки зрения истинных положительных и отрицательных сторон, поскольку он игнорирует истинные отрицательные стороны. Другими словами, он подчеркивает сходство в терминах пар точек, которые принадлежат друг другу как при кластеризации, так и при разбиении по основанию, но не учитывает пары точек, которые не принадлежат друг другу.

Статистика Рэнда

Статистика Рэнда измеряет долю истинно положительных и истинно отрицательных результатов по всем парам точек; это определяется как

$$Rand = \frac{TP + FN}{N} \tag{11.12}$$

Симметричная статистика Rand измеряет долю пар точек, в которых совпадают C и T. Кластеризация префектов имеет значение 1 для статистики.

Мера Фаулкса-Мальлоу

Определяет общую *попарную точность* и значения *попарного отзыва* для кластеризации C следующим образом:

$$prec = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

Точность измеряет долю истинных или правильно сгруппированных пар точек по сравнению со всеми парами точек в одном кластере. С другой стороны, отзыв измеряет долю правильно помеченных пар точек по сравнению со всеми парами точек в том же разделе.

Мера Фаулкса-Малльлоу (FM) определяется как среднее геометрическое для попарной точности и отзыва.

$$FM = \sqrt{prec \cdot recall} = \frac{TP}{\sqrt{(TP + FN)(TP + FP)}} \tag{11.13}$$

Мера FM также асимметрична с точки зрения истинных положительных и отрицательных сторон, поскольку игнорирует истинные отрицательные стороны. Его максимальное значение также равно 1, что достигается при отсутствии ложных срабатываний или отрицательных результатов.

Пример 11.3

Продолжим пример 2 и снова рассмотрим таблицу сопряженности для кластеризации на рисунке 11.1a:

	iris-setosa T_1	**iris-versicolor** T_2	**iris-virginica** T_3
C_1	0	47	14
C_2	50	0	0
C_3	0	3	36

Используя уравнение (11.7), мы можем получить количество истинных положительных результатов следующим образом:

$$TP = \binom{47}{2} + \binom{14}{2} + \binom{50}{2} + \binom{3}{2} + \binom{36}{2} = 1081 + 91 + 1225 + 3 + 630 = 3030$$

Используя уравнения (11.8), (11.9) и (11.10) получаем

$$FN = 645$$
$$FP = 766$$
$$TN = 6734$$

Обратите внимание, что всего существует $N = \binom{150}{2} = 11175$ пар точек.

Теперь мы можем вычислить различные попарные меры для оценки кластеризации. Коэффициент Жаккара [уравнение (11.11)], статистика Рэнда [уравнение (11.12)] и мера Фаулкса – Малльлоу [уравнение (11.13)], задаются как

$$Jaccard = \frac{3030}{3030 + 645 + 766} = \frac{3030}{4441} = 0.68$$
$$Rand = \frac{3030 + 6734}{11175} = \frac{9764}{11175} = 0.87$$
$$FM = \frac{3030}{\sqrt{3675 \cdot 3796}} = \frac{3030}{3735} = 0.81$$

Используя таблицу сопряженности для кластеризации на рисунке 11.1b из примера 2, мы получаем

$$TP = 2891$$

$$FN = 784$$
$$FP = 2380$$
$$TN = 5120$$

В таблице ниже сравниваются различные меры, основанные на сопряженности, для двух кластеров на рисунке 11.1.

	Jaccard	*Rand*	*FM*
(a) Good	0.682	0.873	0.811
(b) Bad	0.477	0.717	0.657

Как и ожидалось, кластеризация на рисунке 11.1a имеет более высокие баллы по всем трем показателям.

11.1.4 Корреляционные меры

Пусть **X** и **Y** две симметричные матрицы размера $n \times n$, и пусть $N = \binom{n}{2}$. Пусть **x**, **y** $\in \mathbb{R}^N$ обозначают векторы, полученные линеаризацией верхних треугольных элементов (исключая главную диагональ) **X** и **Y** (например, построчно) соответственно. Пусть μ_X обозначает поэлементное среднее *x*, заданное как

$$\mu_X = \frac{1}{N}\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \boldsymbol{X}(i,j) = \frac{1}{N}\boldsymbol{x}^T\boldsymbol{x}$$

и пусть $\mathbf{z}_x$ обозначает центрированный вектор **x**, определенный как

$$\boldsymbol{z}_x = \boldsymbol{x} - \mathbf{1}\cdot\mu_X$$

где $\mathbf{1} \in \mathbb{R}^N$ – вектор всех единиц. Аналогично, пусть μ_Y будет поэлементным средним **y**, а $\mathbf{z}_y$ - центрированным вектором **y**.

Статистика Гильберта определяется как усредненное поэлементное произведение между **X** и **Y**

$$\Gamma = \frac{1}{N}\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \boldsymbol{X}(i,j)\cdot\boldsymbol{Y}(i,j) = \frac{1}{N}\boldsymbol{x}^T\boldsymbol{y} \tag{11.14}$$

Нормализованная статистика Гильберта определяется как поэлементная корреляция между **X** и **Y**

$$\Gamma_n = \frac{\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}(\boldsymbol{X}(i,j)-\mu_X)(\boldsymbol{Y}(i,j)-\mu_Y)}{\sqrt{\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}(\boldsymbol{X}(i,j)-\mu_X)^2 \quad \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}(\boldsymbol{Y}[\boldsymbol{i}]-\mu_Y)^2}} = \frac{\sigma_{XY}}{\sqrt{\sigma_X^2\sigma_Y^2}}$$

где σ_X^2 и σ_Y^2 - дисперсии, а σXY - ковариация векторов x и y, определяемых как

$$\sigma_X^2 = \frac{1}{N}\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}(\boldsymbol{X}(i,j)-\mu_X)^2 = \frac{1}{N}\boldsymbol{z}_x^T\boldsymbol{z}_x = \frac{1}{N}\|\boldsymbol{z}_x\|^2$$

$$\sigma_Y^2 = \frac{1}{N}\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}(\boldsymbol{Y}(i,j) - \mu_Y)^2 = \frac{1}{N}\boldsymbol{z}_y^T\boldsymbol{z}_y = \frac{1}{N}\|\boldsymbol{z}_y\|^2$$

$$\sigma_{XY}^2 = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}(\boldsymbol{X}(i,j) - \mu_X)(\boldsymbol{Y}(i,j) - \mu_Y) = \frac{1}{N}\boldsymbol{z}_x^T\boldsymbol{z}_y$$

Таким образом, нормализованная статистика Гильберта может быть переписана как

$$\Gamma_n = \frac{\boldsymbol{z}_x^T\boldsymbol{z}_y}{\|\boldsymbol{z}_x\| \cdot \|\boldsymbol{z}_y\|} = \cos\theta \tag{11.15}$$

где θ – угол между двумя центрированными векторами $\boldsymbol{z}_x$ и $\boldsymbol{z}_y$. Отсюда сразу следует, что Γ_n изменяется от −1 до +1.

Когда **X** и **Y** являются произвольными матрицами размера $n \times n$, приведенные выше выражения можно легко изменить, чтобы охватить все n^2 элементов двух матриц. (Нормализованная) статистика Гильберта может использоваться в качестве меры внешней оценки с соответствующим образом определенными матрицами **X** и **Y**, как описано ниже.

Дискретизированная статистика Гильберта

Пусть **T** и **C** - матрицы размера $n \times n$, определенные как

$$\boldsymbol{T}(i,j) = \begin{cases} 1 \text{ если } y_i = y_j, i \neq j \\ 0 \text{ иначе} \end{cases}$$

$$\boldsymbol{C}(i,j) = \begin{cases} 1 \text{ если } \hat{y}_i = \hat{y}_j, i \neq j \\ 0 \text{ иначе} \end{cases}$$

Кроме того, пусть **t**, **c** $\in \mathbb{R}^N$ обозначает N-мерные векторы, составляющие верхние треугольные элементы (исключая диагональ) **T** и **C** соответственно, где $N = \binom{n}{2}$ обозначает количество различных пар точек. Наконец, пусть $\mathbf{z}_t$ и $\mathbf{z}_c$ обозначают центрированные векторы **t** и **c**.

Дискретизированная статистика Гильберта вычисляется по формуле (11.14), положив **x** = **t** и **y** = **c**:

$$\Gamma = \frac{1}{N}\boldsymbol{t}^T\boldsymbol{c} = \frac{TP}{N} \tag{11.16}$$

Поскольку *i*-й элемент **t** равен 1 только тогда, когда *i*-я пара точек принадлежит одному и тому же разделу, и аналогично *i*-й элемент **c** равен 1 только тогда, когда *i*-я пара точек также принадлежит к тому же кластеру, точечное произведение $\boldsymbol{t}^T\boldsymbol{c}$ – это просто количество истинно положительных значений, и, таким образом, значение эквивалентно доле всех пар, которые являются истинно положительными. Отсюда следует, что чем выше соответствие между истинным разбиением *T* и кластеризацией *C*, тем выше значение.

Нормализованная дискретная статистика Гильберта

Нормализованная версия дискретизированной статистики Гильберта – это просто корреляция между **t** и **c** [уравнение (11.15)]:

$$\Gamma_n = \frac{\boldsymbol{z}_t^T \boldsymbol{z}_c}{\|\boldsymbol{z}_t\| \cdot \|\boldsymbol{z}_c\|} = \cos\theta \tag{11.17}$$

Обратите внимание, что $\mu_T = \frac{1}{N}\boldsymbol{t}^T\boldsymbol{t}$ – это доля пар точек, принадлежащих одному и тому же разделу, то есть с $y_i = y_j$, независимо от того, совпадает ли $\hat{y}_i$ с $\hat{y}_j$ или нет. Таким образом, мы имеем

$$\mu_T = \frac{\boldsymbol{t}^T\boldsymbol{t}}{N} = \frac{TP + FN}{N}$$

Аналогичным образом, $\mu_C = \frac{1}{N}\boldsymbol{c}^T\boldsymbol{c}$ – это доля пар точек, принадлежащих одному кластеру, то есть с $\hat{y}_i = \hat{y}_j$, независимо от того, совпадает ли y_i и y_j или нет, так что

$$\mu_C = \frac{c^T c}{N} = \frac{TP + FP}{N}$$

Подставляя их в числитель в формуле (11.17), получаем

$$\boldsymbol{z}_t^T\boldsymbol{z}_c = (\boldsymbol{t} - \mathbf{1}\cdot\mu_T)^T(\boldsymbol{c} - \mathbf{1}\cdot\mu_C) = \boldsymbol{t}^T\boldsymbol{c} - \mu_C\boldsymbol{t}^T\mathbf{1} - \mu_T\boldsymbol{c}^T\mathbf{1} + \mathbf{1}^T\mathbf{1}\mu_T\mu_C =$$
$$= \boldsymbol{t}^T\boldsymbol{c} - N\mu_C\mu_T - N\mu_T\mu_C + N\mu_T\mu_C = \boldsymbol{t}^T\boldsymbol{c} - N\mu_T\mu_C =$$
$$= TP - N\mu_T\mu_C \tag{11.18}$$

где $\mathbf{1} \in \mathbb{R}^N$ – вектор всех единиц. Мы также использовали тождества $\boldsymbol{t}^T\mathbf{1} = \boldsymbol{t}^T\boldsymbol{t}$ и $\boldsymbol{c}^T\mathbf{1} = \boldsymbol{c}^T\boldsymbol{c}$. Аналогичным образом мы можем вывести

$$\|\boldsymbol{z}_t\| = \boldsymbol{z}_t^T\boldsymbol{z}_t = \boldsymbol{t}^T\boldsymbol{t} - N\mu_T^2 = N\mu_T - N\mu_T^2 = N\mu_T(1-\mu_T) \tag{11.19}$$

$$\|\boldsymbol{z}_c\| = \boldsymbol{z}_c^T\boldsymbol{z}_c = \boldsymbol{c}^T\boldsymbol{c} - N\mu_C^2 = N\mu_C - N\mu_C^2 = N\mu_C(1-\mu_C) \tag{11.20}$$

Подключая уравнения (11.18), (11.19) и (11.20) в уравнение. (11.17) нормализованная дискретизированная статистика Гильберта может быть записана как

$$\Gamma_n = \frac{\frac{TP}{N} - \mu_T\mu_C}{\sqrt{\mu_T\mu_C(1-\mu_T)(1-\mu_C)}} \tag{11.21}$$

поскольку $\mu_T = \frac{TP+FN}{N}$ нормализованная статистика Γ_n может быть вычислена с использованием только значений *TP, FN* и *FP*. Максимальное значение Γ_n = +1 получается, когда нет ложных срабатываний или отрицательных результатов, то есть, когда *FN* = *FP* = 0. Минимальное значение Γ_n = −1 - это когда нет истинных положительных и отрицательных результатов, то есть когда *TP* = *TN* = 0.

Пример 11.4

Продолжая пример 11.3, для хорошей кластеризации на рисунке 11.1a мы имеем

$$TP = 3030$$

$$FN = 645$$

$$FP = 766$$
$$TN = 6734$$

Из этих значений получаем

$$\mu_T = \frac{TP + FN}{N} = \frac{3675}{11175} = 0.33$$
$$\mu_C = \frac{TP + FP}{N} = \frac{3796}{11175} = 0.34$$

Используя уравнения (11.16) и (11.21) значения статистики Гильберта равны

$$\Gamma = \frac{3030}{11175} = 0.271$$
$$\Gamma_n = \frac{0.27 - 0.33 \cdot 0.34}{\sqrt{0.33 \cdot 0.34 \cdot (1 - 0.33) \cdot (1 - 0.34)}} = \frac{0.159}{0.222} = 0.717$$

Аналогично, для плохой кластеризации на рисунке 11.1b мы имеем

$$TP = 2891$$
$$FN = 784$$
$$FP = 2380$$
$$TN = 5120$$

а значения для дискретизированной статистики Гильберта даются как

$$\Gamma = 0.258$$
$$\Gamma_n = 0.442$$

Мы наблюдаем, что хорошая кластеризация имеет более высокие значения, хотя нормализованная статистика более разборчива, чем ненормализованная версия, то есть хорошая кластеризация имеет гораздо более высокое значение Γ_n, чем плохая кластеризация, тогда как разница в Γ для двух кластеров составляет не так много.

11.2 Внутренние меры

Меры внутренней оценки не прибегают к прямому разделению, которое является типичным сценарием при кластеризации набора данных. Следовательно, для оценки качества кластеризации внутренние меры должны использовать понятия внутрикластерного сходства или компактности, в отличие от понятий межкластерного разделения, обычно с компромиссом для достижения этих двух целей. Внутренние меры основаны на *матрице расстояний* $n \times n$, также называемой *матрицей близости*, всех попарных расстояний между n точками:

$$\boldsymbol{W} = \{\delta(x_i, x_j)\}_{i,j=1}^{n} \tag{11.22}$$

где

$$\delta(x_i, x_j) = \|x_i - x_j\|_2$$

Евклидово расстояние между x_i, $x_j \in D$, хотя можно использовать и другие метрики расстояния. Поскольку **W** симметрична и $\delta(x_i, x_j) = 0$, обычно во внутренних мерах используются только верхние треугольные элементы **W** (исключая диагональ).

Матрицу близости **W** можно также рассматривать как матрицу смежности взвешенного полного графа *G* по *n* точкам, то есть с узлами $V = \{x_i \mid x_j \in D\}$, ребрами $E = \{(x_i, x_j) \mid x_i, x_j \in D\}$,, а веса ребер $w_{ij} = W(i, j)$ для всех x_i, $x_j \in D$. Таким образом, существует тесная связь между мерами внутренней оценки и задачами кластеризации графов, которые мы рассмотрели в главе 16.

Что касается внутренних мер, мы предполагаем, что у нас нет доступа к исходному разделению. Вместо этого мы предполагаем, что нам дана кластеризация $C = \{C_1, \ldots, C_k)$, состоящая из $r = k$ кластеров, причем кластер C_i содержит $n_i = |C_i|$ точек. Пусть $\hat{y}_i \in \{1, 2, \ldots, k\}$ обозначает метку кластера для точки x_i. Кластеризацию *C* можно рассматривать как k-образное разрез в G, потому что $C_i \neq 0$ для всех i, $C_i \cap C_j \neq 0$ для всех i, j и $\cup_i C_i = V$. Учитывая подмножества S, R $\subset$ V, определим W (S, R) как сумму весов на всех ребрах с одной вершиной в S и другой в R, заданных как

$$W(S, R) = \sum_{x_i \in S} \sum_{x_j \in R} w_{ij}$$

Также, если S $\subseteq$ V, обозначим через $\bar{S}$ дополнительный набор вершин, то есть $\bar{S}$ = V - S.

Внутренние меры основаны на различных функциях внутрикластерных и межкластерных весов. В частности, обратите внимание, что сумма всех внутрикластерных весов по всем кластерам задается как

$$W_{in} = \frac{1}{2} \sum_{i=1}^{k} W(C_i, C_i) \tag{11.23}$$

Мы делим на 2, потому что каждое ребро в C_i учитывается дважды при суммировании $W(C_i, C_i)$. Также обратите внимание, что сумма всех межкластерных весов задается как

$$W_{out} = \frac{1}{2} \sum_{i=1}^{k} W(C_i, \bar{C}_i) = \sum_{i=1}^{k-1} \sum_{j>1} W(C_i, C_j) \tag{11.24}$$

Здесь мы тоже делим на 2, потому что каждое ребро учитывается дважды при суммировании по кластерам. Количество отдельных внутрикластерных ребер, обозначенных N_{in}, и межкластерных ребер, обозначенных N_{out}, задается как

$$N_{in} = \sum_{i=1}^{k} \binom{n_i}{2} = \frac{1}{2} \sum_{i=1}^{k} n_i (n_i - 1)$$

$$N_{out} = \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} n_i \cdot n_j = \frac{1}{2} \sum_{i=1}^{k} \sum_{\substack{j=i+1 \\ j \neq i}}^{k} n_i \cdot n_j$$

Отметим, что общее количество различных пар точек N удовлетворяет тождеству

$$N = N_{in} + N_{out} = \binom{n}{2} = \frac{1}{2} n(n-1)$$

BetaCV Мера

Мера BetaCV — это отношение среднего внутрикластерного расстояния к среднему межкластерному расстоянию:

$$BetaCV = \frac{W_{in}/N_{in}}{W_{out}/N_{out}} = \frac{N_{out}}{N_{in}} \cdot \frac{W_{out}}{W_{in}} = \frac{N_{out}}{N_{in}} \frac{\sum_{i=1}^{k} W(C_i, C_i)}{\sum_{i=1}^{k} W(C_i, \overline{C_i})}$$

Чем меньше отношение BetaCV, тем лучше кластеризация, поскольку это указывает на то, что внутрикластерные расстояния в среднем меньше, чем межкластерные.

C-index

Пусть $W_{min}(N_{in})$ будет суммой наименьших расстояний N_{in} в матрице близости **W**, где N_{in} - общее количество внутрикластерных ребер или пар точек. Пусть $W_{max}(N_{in})$ будет суммой наибольших расстояний N_{in} в **W**.

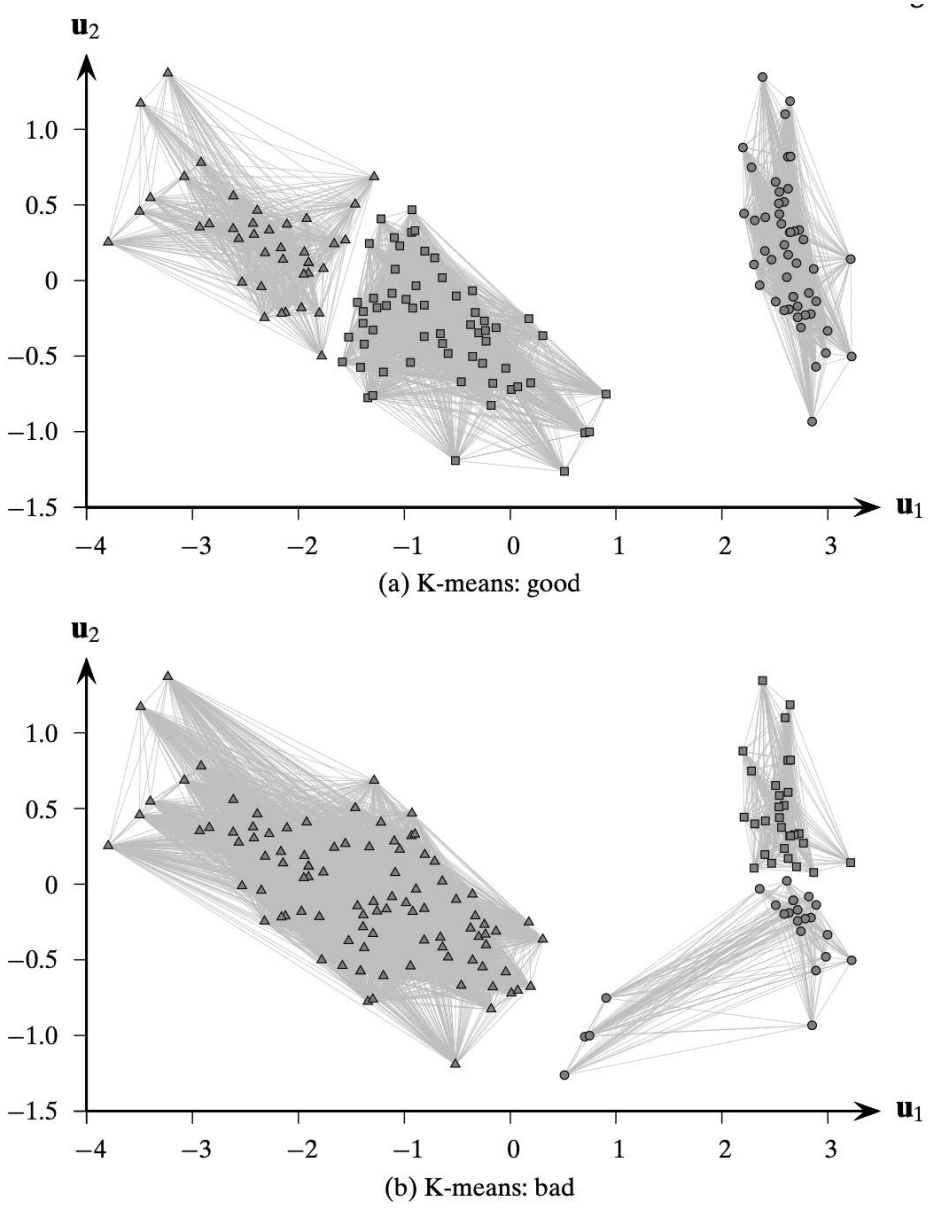


Рисунок 11.2 Кластеризации как графы: Ирис.

C-index измеряет, в какой степени кластеризация объединяет точки N_{in}, которые являются ближайшими в k кластерах. Он определяется как

$$Cindex = \frac{W_{in} - W_{min}(N_{in})}{W_{max}(N_{in}) - W_{min}(N_{in})}$$

где W_{in} - сумма всех внутрикластерных расстояний [ур. (11.23)]. C-index находится в диапазоне [0,1]. Чем меньше C-index, тем лучше кластеризация, поскольку он указывает на более компактные кластеры с относительно меньшими расстояниями внутри кластеров, а не между кластерами.

Нормализованная мера разреза

Нормализованная цель разреза [Ур. (16.17)] для кластеризации графов также может использоваться в качестве меры оценки внутренней кластеризации:

$$NC = \sum_{i=1}^{k} \frac{W(C_i, \overline{C}_i)}{vol(C_i)} = \sum_{i=1}^{k} \frac{W(C_i, \overline{C}_i)}{W(C_i, V)}$$

где $vol(C_i) = W(C_i, V)$ - объем кластера C_i, то есть суммарные веса на ребрах с хотя бы одним концом в кластере. Однако, поскольку мы используем матрицу близости или расстояния W вместо матрицы сходства A, чем выше нормализованное значение разреза, тем лучше.

Чтобы убедиться в этом, воспользуемся наблюдением, что $W(C_i, V) = W(C_i, C_i) + W(C_i, \overline{C}_i)$, так что

$$NC = \sum_{i=1}^{k} \frac{W(C_i, \overline{C}_i)}{W(C_i, C_i) + W(C_i, \overline{C}_i)} = \sum_{i=1}^{k} \frac{W(C_i, \overline{C}_i)}{\frac{W(C_i, C_i)}{W(C_i, \overline{C}_i)} + 1}$$

Мы можем видеть, что *NC* максимизируется, когда отношения $\frac{W(C_i, C_i)}{W(C_i, \overline{C}_i)}$ (по k кластерам) настолько малы, насколько это возможно, что происходит, когда внутрикластерные расстояния намного меньше по сравнению с межкластерными расстояниями, то есть когда кластеризация хорошая. Максимально возможное значение NC равно *k*.

Модульность

Модульность для кластеризации графов [Ур. (16.26)] также может использоваться как внутренняя мера:

$$Q = \sum_{i=1}^{k} \left(\frac{W(C_i, C_i)}{W(V, V)} - \left(\frac{W(C_i, V)}{W(V, V)} \right)^2 \right)$$

где

$$W(V, V) = \sum_{i=1}^{k} W(C_i, V) = \sum_{i=1}^{k} W(C_i, C_i) + \sum_{i=1}^{k} W(C_i, \overline{C}_i) = 2(W_{in} + W_{out})$$

Последний шаг следует из Ур. (11.23) и (11.24). Модульность измеряет разницу между наблюдаемой и ожидаемой долей весов на краях внутри кластеров. Поскольку мы используем матрицу расстояний, чем меньше показатель модульности, тем лучше кластеризация, что указывает на то, что внутрикластерные расстояния ниже ожидаемых.

Индекс Данна

Индекс Данна определяется как соотношение между минимальным расстоянием между парами точек из разных кластеров и максимальным расстоянием между парами точек из одного кластера. Более формально у нас есть

$$Dunn = \frac{W_{out}^{min}}{W_{in}^{max}}$$

Где W_{out}^{min} - минимальное межкластерное расстояние:

$$W_{out}^{min} = min_{i,j>i}\{w_{ab}|x_a \in C_i, x_b \in C_j\}$$

И W_{in}^{max} - максимальное внутрикластерное расстояние:

$$W_{in}^{max} = max_i\{w_{ab}|x_a, x_b \in C_i\}$$

Чем больше индекс Данна, тем лучше кластеризация, потому что это означает, что даже самое близкое расстояние между точками в разных кластерах намного больше, чем самое дальнее расстояние между точками в одном кластере. Однако индекс Данна может быть нечувствительным, поскольку минимальные межкластерные и максимальные внутрикластерные расстояния не охватывают всю информацию о кластеризации.

Индекс Дэвиса – Болдина

Пусть μ_i обозначает кластерное среднее, заданное как

$$\mu_i = \frac{1}{n_i}\sum_{x_j \in C_j} x_j \tag{11.25}$$

Далее, пусть σ_{μ_i} обозначает дисперсию или разброс точек вокруг среднего кластера, заданный как

$$\sigma_{\mu_i} = \sqrt{\frac{\sum_{x_j \in C_j} \delta\left(x_j, \mu_i\right)^2}{n_i}} = \sqrt{var(C_i)}$$

где $var(C_i)$ - полная дисперсия [Ур. (1.4)] кластера C_i.

Мера Дэвиса – Болдина для пары кластеров C_i и C_j определяется как отношение

$$DB_{ij} = \frac{\sigma_{\mu_i} + \sigma_{\mu_j}}{\delta(\sigma_{\mu_i}, \sigma_{\mu_j})}$$

DB_{ij} измеряет, насколько компактны кластеры по сравнению с расстоянием между средними значениями кластеров. Индекс Дэвиса – Болдина тогда определяется как

$$DB = \frac{1}{k}\sum_{i=1}^{k} \max_{j \neq i}\{DB_{ij}\}$$

То есть для каждого кластера C_i мы выбираем кластер C_j, который дает наибольшее отношение DB_{ij}. Чем меньше значение DB, тем лучше кластеризация, поскольку это означает, что кластеры хорошо разделены (т. Е. Расстояние между средними значениями кластеров большое), и каждый кластер хорошо представлен своим средним значением (т.е. имеет небольшой разброс).

Коэффициент Силуэта

Коэффициент силуэта является мерой сплоченности и разделения кластеров и основан на разнице между средним расстоянием до точек в ближайшем кластере и до точек в том же кластере. Для каждой точки x_i вычисляем коэффициент силуэта s_i как

$$s_i = \frac{\mu_{out}^{min}(x_i) - \mu_{in}(x_i)}{max\{\mu_{out}^{min}(x_i), \mu_{in}(x_i)\}} \tag{11.26}$$

где $\mu_{in}(x_i)$ - среднее расстояние от x_i до точек в собственном кластере $\hat{y}_i$:

$$\mu_{in}(x_i) = \frac{\sum_{x_j \in C_{\hat{y}_j}, j \neq i} \delta(x_i, x_j)}{n_{\hat{y}_i} - 1}$$

а $\mu_{out}^{min}(x_i)$ - среднее расстояние от xi до точек ближайшего кластера:

$$\mu_{out}^{min}(x_i) = \min_{j \neq \hat{y}_i} \left\{ \frac{\sum_{y \in C_j} \delta(x_i, y)}{n_j} \right\}$$

Значение s_i точки лежит в интервале [−1, + 1]. Значение, близкое к +1, указывает, что x_i гораздо ближе к точкам в собственном кластере и далеко от других кластеров. Значение, близкое к нулю, указывает на то, что x_i близко к границе между двумя кластерами. Наконец, значение, близкое к -1, указывает, что x_i намного ближе к другому кластеру, чем его собственный кластер, и, следовательно, точка может быть неправильно кластеризована.

Коэффициент силуэта определяется как среднее значение s_i по всем точкам:

$$SC = \frac{1}{n} \sum_{i=1}^{n} s_i \tag{11.27}$$

Значение, близкое к +1, указывает на хорошую кластеризацию.

Статистика Гильберта

Статистика Гильберта [Ур. (11.14)], и его нормализованная версия Γ_n [Ур. (11.15)], могут использоваться как меры внутренней оценки, если **X** = **W** является парной матрицей расстояний, а также путем определения **Y** как матрицы расстояний между кластерами:

$$Y = \left\{ \delta \left(\mu_{\hat{y}_i}, \mu_{\hat{y}_j} \right) \right\}_{i,j=1}^{n} \tag{11.28}$$

Поскольку **W** и **Y** симметричны, и Γ, и Γ_n вычисляются по их верхним треугольным элементам.

11.3. Относительные меры

Относительные меры используются для сравнения различных кластеров, полученных путем варьирования разных параметров для одного и того же алгоритма, например, для выбора количества кластеров k.

Коэффициент силуэта (Silhouette Coefficient)

Коэффициент силуэта [Ур. (11.26)] для каждой точки s_j, и среднее значение SC [уравнение. (11.27)], можно использовать для оценки количества кластеров в данных. Подход

состоит из построения значений s_j в порядке убывания для каждого кластера и отметки общего значения SC для конкретного значения k, а также значений SC по кластерам:

$$SC_i = \frac{1}{n_i} \sum_{x_j \in C_i} s_j$$

Затем мы можем выбрать значение k, которое дает наилучшую кластеризацию, при этом многие точки имеют высокие значения s_j в каждом кластере, а также высокие значения для SC и $SC_i (1 \leq i \leq k)$.

Индекс Калински – Харабаса

Учитывая набор данных $D = \{x_i\}_{i=1}^n$, матрица рассеяния для D имеет вид

$$S = n\Sigma = \sum_{j=1}^{n} (x_j - \mu)(x_j - \mu)^T$$

где $\mu = \sum_{j=1}^{n} x_j$- среднее значение, а Σ - ковариационная матрица. Матрицу разброса можно разложить на две матрицы: $S = S_W + S_B$, где S_W — это матрица внутрикластерного рассеяния, а S_B - это матрица межкластерного рассеяния, заданная как

$$S_W = \sum_{i=1}^{k} \sum_{x_j \in C_i} (x_j - \mu_i)(x_j - \mu_i)^T$$

$$S_B = \sum_{i=1}^{k} n_i (\mu_i - \mu)(\mu_i - \mu)^T$$

$\mu_i = \frac{1}{n_i} \sum_{x_j \in C_i} x_j$ - среднее значение для кластера C_i

Критерий отношения дисперсии Калински – Харабаса (CH) для данного значения k определяется следующим образом

$$CH(k) = \frac{tr(S_B)/(k-1)}{tr(S_W)/(n-k)} = \frac{n-k}{k-1} \cdot \frac{tr(S_B)}{tr(S_W)}$$

где $tr(S_W)$ и $tr(S_B)$ — это следы (сумма диагональных элементов) внутрикластерных и межкластерных матриц рассеяния.

Для хорошего значения k мы ожидаем, что разброс внутри кластера будет меньше по сравнению с разбросом между кластерами, что должно привести к более высокому значению $CH(k)$. С другой стороны, нам не нужно очень большое значение k; таким образом, член $\frac{n-k}{k-1}$ ставить в невыгодное положение большие значения k. Мы могли бы выбрать значение k, которое максимизирует $CH(k)$. В качестве альтернативы, мы можем построить график значений CH и ожидать большого увеличения значения с последующим небольшим усилением или его отсутствием. Например, мы можем выбрать значение $k > 3$, которое минимизирует член

$$\Delta(k) = \big(CH(k+1) - CH(k)\big) - \big(CH(k) - CH(k-1)\big)$$

Интуиция заключается в том, что мы хотим найти значение k, для которого $CH(k)$ намного выше, чем $CH(k-1)$, и есть только небольшое улучшение или уменьшение значения $CH(k+1)$.

Статистика разрыва

Статистика разрыва сравнивает сумму внутрикластерных весов W_{in} [Ур. (11.23)] для различных значений k с их ожидаемыми значениями, предполагающими отсутствие явной структуры кластеризации, что формирует нулевую гипотезу.

Пусть C_k будет кластеризацией, полученной для указанного значения k с использованием выбранного алгоритма кластеризации. Пусть $W_{in}^k(D)$ обозначает сумму внутрикластерных весов (по всем кластерам) для C_k во входном наборе данных D. Мы хотим вычислить вероятность наблюдаемого значения W_k при нулевой гипотезе о том, что точки случайным образом размещаются в то же пространстве данных, что и D. К сожалению, распределение выборки W_{in} неизвестно. Кроме того, оно зависит от количества кластеров k, количества точек n и других характеристик D.

Чтобы получить эмпирическое распределение для W_{in}, мы прибегаем к моделированию процесса выборки методом Монте-Карло. То есть мы генерируем t случайных выборок, содержащих n случайно распределенных точек в том же d-мерном пространстве данных, что и входной набор данных D. То есть для каждого измерения D, скажем X_j, мы вычисляем его диапазон $[min(X_j), max(X_j)]$ и генерируем значения для n точек (для j-го измерения) равномерно случайным образом в заданном диапазоне. Обозначим через $R_i \in \mathbb{R}^{n\,x\,d}, 1 \leq i \leq t$, i -й образец. Пусть $W_{in}^k(R_i)$ обозначает сумму внутрикластерных весов для данной кластеризации R_i в k кластеров. Из каждого образца набора данных R_i мы генерируем кластеризацию для различных значений k, используя тот же алгоритм, и записываем внутрикластерные значения $W_{in}^k(R_i)$. Пусть $\mu_W(k)$ и $\mu_W(k)$ обозначают среднее и стандартное отклонение этих внутрикластерных весов для каждого значения k, заданного как

$$\mu_W(k) = \frac{1}{t}\sum_{l=1}^{t} \log W_{in}^k(R_i)$$

$$\sigma_W(k) = \sqrt{\frac{1}{t}\sum_{i=1}^{t}\Big(\log W_{in}^k(R_i) - \mu_W(k)\Big)^2}$$

где мы используем логарифм значений W_{in}, так как они могут быть довольно большими.

Статистика разрыва для данного k тогда определяется как

$$gap(k) = \mu_W(k) - \log W_{in}^k(D)$$

Он измеряет отклонение наблюдаемого значения W_{in}^k от его ожидаемого значения при нулевой гипотезе. Мы можем выбрать значение k, которое дает статистику наибольшего разрыва, поскольку оно указывает на структуру кластеризации вдали от равномерного распределения точек. Более надежный подход - выбрать k следующим образом:

$$k^* = \arg\min_k \{gap(k) \geq gap(k+1) - \sigma_W(k+1)\}$$

То есть мы выбираем наименьшее значение k так, чтобы статистика разрыва находилась в пределах одного стандартного отклонения разрыва при $k + 1$.

11.3.1 Стабильность кластера

Основная идея стабильности кластера заключается в том, что кластеризация, полученная из нескольких наборов данных, отобранных из того же базового распределения, что и D, должна быть аналогичной или «стабильной». Подход кластерной стабильности можно использовать для поиска хороших значений параметров для данного алгоритма кластеризации; мы сосредоточимся на задаче поиска хорошего значения k, правильного количества кластеров.

Совместное распределение вероятностей для D обычно неизвестно. Поэтому для выборки набора данных из одного и того же распределения мы можем попробовать различные методы, включая случайные возмущения, подвыборку или повторную выборку с начальной загрузкой. Давайте рассмотрим подход начальной загрузки; мы генерируем t выборок размера n путем выборки из D с заменой, что позволяет выбирать одну и ту же точку, возможно, несколько раз, и, таким образом, каждая выборка D_i будет отличаться. Затем для каждого образца D_i мы запускаем один и тот же алгоритм кластеризации с разными значениями кластера k в диапазоне от 2 до k^{max}.

Пусть $\mathcal{C}_\mathrm{k}(\mathrm{D_i})$ обозначает кластеризацию, полученную из выборки $\mathrm{D_i}$, для данного значения k. Затем метод сравнивает расстояние между всеми парами кластеризации $\mathcal{C}_\mathrm{k}(\mathrm{D_i})$ и $\mathcal{C}_\mathrm{k}(\mathrm{D_j})$ с помощью некоторой функции расстояния. Некоторые из показателей внешней кластерной оценки можно использовать в качестве меры расстояния, задав, например, $\mathcal{C} = \mathcal{C}_\mathrm{k}(\mathrm{D_i})$ и $T = \mathcal{C}_\mathrm{k}(D_j)$ или наоборот. По этим значениям мы вычисляем ожидаемое попарное расстояние для каждого значения k. Наконец, значение k^*, которое показывает наименьшее отклонение между кластеризацией, полученной из повторно дискретизированных наборов данных, является лучшим выбором для k, поскольку оно демонстрирует наибольшую стабильность.

Однако существует одна сложность при оценке расстояния между парой кластеризации $\mathcal{C}_\mathrm{k}(\mathrm{D_i})$ и $\mathcal{C}_\mathrm{k}(\mathrm{D_j})$, а именно то, что базовые наборы данных $\mathrm{D_i}$ и $\mathrm{D_j}$ различны. То есть набор кластеризованных точек отличается, потому что каждый образец отличается. Перед вычислением расстояния между двумя кластерами мы должны ограничить кластеризацию только точками, общими для $\mathrm{D_i}$ и $\mathrm{D_j}$, обозначенных как D_{ij}. Поскольку выборка с заменой позволяет использовать несколько экземпляров одной и той же точки, мы также должны учитывать это при создании D_{ij}. Для каждой точки x_a во входном наборе данных D пусть m_i^a и m_j^a обозначают количество вхождений x_a в $\mathrm{D_i}$ и $\mathrm{D_j}$, соответственно.

Зададим

$$D_{ij} = D_i \cap D_j = \left\{ m^a \text{ экземпляр } x_a \mid x_a \in D, m^a = \min\{m_i^a, m_j^a\} \right\} \qquad (11.30)$$

То есть общий набор данных D_{ij} создается путем выбора минимального количества экземпляров точки x_a в $\mathrm{D_i}$ или $\mathrm{D_j}$.

Алгоритм 11.1 — Алгоритм кластеризации устойчивости для выбора k

CLUSTERINGSTABILITY $(A, t, k^{\max}, \mathbf{D})$:

1 $n \leftarrow |\mathbf{D}|$
// Generate t samples
2 **for** $i = 1, 2, \ldots, t$ **do**
3 $\quad \mathbf{D}_i \leftarrow$ sample n points from $\mathbf{D}$ with replacement
// Generate clusterings for different values of k
4 **for** $i = 1, 2, \ldots, t$ **do**
5 $\quad$ **for** $k = 2, 3, \ldots, k^{\max}$ **do**
6 $\quad\quad \mathcal{C}_k(\mathbf{D}_i) \leftarrow$ cluster $\mathbf{D}_i$ into k clusters using algorithm A
// Compute mean difference between clusterings for each k
7 **foreach** *pair* $\mathbf{D}_i, \mathbf{D}_j$ *with* $j > i$ **do**
8 $\quad \mathbf{D}_{ij} \leftarrow \mathbf{D}_i \cap \mathbf{D}_j$ // create common dataset using Eq. (17.30)
9 $\quad$ **for** $k = 2, 3, \ldots, k^{\max}$ **do**
10 $\quad\quad d_{ij}(k) \leftarrow d\big(\mathcal{C}_k(\mathbf{D}_i), \mathcal{C}_k(\mathbf{D}_j), \mathbf{D}_{ij}\big)$ // distance between clusterings
11 **for** $k = 2, 3, \ldots, k^{\max}$ **do**
12 $\quad \mu_d(k) \leftarrow \frac{2}{t(t-1)} \sum_{i=1}^{t} \sum_{j>i} d_{ij}(k)$ // expected pairwise distance
// Choose best k
13 $k^* \leftarrow \arg\min_k \{\mu_d(k)\}$

Алгоритм 11.1 показывает псевдокод метода стабильности кластеризации для выбора наилучшего значения k. Он принимает в качестве входных данных алгоритм кластеризации A, количество выборок t, максимальное количество кластеров k^{max} и входной набор данных D. Сначала он генерирует t выборок начальной загрузки и кластеризует их, используя алгоритм A. Затем он вычисляет расстояние между кластерами для каждой пары наборов данных Di и Dj для каждого значения k. Наконец, метод вычисляет ожидаемое попарное расстояние $\mu_d(k)$ в строке 12. Мы предполагаем, что функция расстояния кластеризации d является симметричной. Если d не является симметричным, то ожидаемую разницу следует вычислять по всем упорядоченным парам, то есть $\mu_d(k) = \frac{1}{r(t-1)} \sum_{i=1}^{r} \sum_{j \neq i} d_{ij}(k)$

Вместо функции расстояния d мы также можем оценить стабильность кластеризации с помощью меры подобия, и в этом случае после вычисления среднего сходства между парами кластеризации для данного k мы можем выбрать наилучшее значение k^* как такое, которое максимизирует ожидаемое подобие $\mu_s(k)$. В общем, те внешние меры, которые дают более низкие значения для лучшего согласования между $\mathcal{C}_{\mathrm{k}}(D_{\mathrm{i}})$ и $\mathcal{C}_{\mathrm{k}}\big(D_{\mathrm{j}}\big)$, могут использоваться как функции расстояния, тогда как те, которые дают более высокие значения для лучшего согласования, могут использоваться как функции подобия. Примеры функций расстояния включают нормализованную взаимную информацию, вариацию информации и условную энтропию (которая является асимметричной). Примеры функций подобия включают статистику Жаккара, Фаулкса – Маллоуса, статистику Гильберта и так далее.

11.3.2 Тенденция к кластеризации

Тенденция к кластеризации или кластеризация направлена на определение того, есть ли в наборе данных D какие-либо значимые группы для начала. Обычно это сложная задача, учитывая разные определения того, что значит быть кластером, например, секционированный, иерархический, на основе плотности, на основе графа и так далее. Даже если мы исправим тип кластера, определение соответствующей нулевой модели (например, модели без какой-либо структуры кластеризации) для данного набора данных все равно будет сложной задачей. Кроме того, если мы определим, что данные являются кластеризованными, тогда мы все еще сталкиваются с вопросом, сколько существует кластеров. Тем не менее, все же стоит оценить возможность кластеризации набора данных; мы рассмотрим некоторые подходы, чтобы ответить на вопрос, являются ли данные кластеризованными или нет.

Пространственная гистограмма

Один из простых подходов - сопоставить d-мерную пространственную гистограмму входного набора данных D с гистограммой из выборок, случайно сгенерированных в том же пространстве данных. Пусть $X_1, X_2, \ldots X_d$ обозначают d измерений. Учитывая b, количество ячеек для каждого измерения, мы делим каждое измерение X_j на b ячеек равной ширины и просто подсчитываем, сколько точек лежит в каждой из ячеек d-размерности b^d. Из этой пространственной гистограммы мы можем получить эмпирическую функцию совместной вероятности и массы (EPMF - empirical joint probability mass function) для набора данных D, которая является приближением неизвестной совместной функции плотности вероятности. EPMF задается как

$$f(i) = P\big(x_j \in \text{ cell i } i\big) = \frac{\big|\{x_j \in cell\ i\}\big|}{n}$$

где $\mathrm{i} = (i_1, i_2, \ldots, i_d)$ обозначает индекс ячейки, а i_j обозначает индекс ячейки по измерению X_j.

Затем мы генерируем t случайных выборок, каждая из которых содержит n точек в том же d-мерном пространстве, что и входной набор данных D. То есть для каждого измерения X_j мы вычисляем его диапазон $[min(X_j), max(X_j)]$ и генерируем значения равномерно случайным образом в пределах данного диапазона. Обозначим через R_j j-ю такую случайную выборку. Затем мы можем вычислить соответствующий EPMF $g_j(i)$ для каждого R_j $1 \leq j \leq t$.

Наконец, мы можем вычислить, насколько распределение f отличается от g_j (для $j = 1, \ldots, t$), используя дивергенцию Кульбака – Лейблера (KL) от f к g_j, определяемую как

$$KL\big(f \mid g_j\big) = \sum_i f(i) \log\left(\frac{f(i)}{g_j(i)}\right) \qquad (11.31)$$

Дивергенция KL равна нулю только тогда, когда f и g_j - одинаковые распределения. Используя эти значения расхождения, мы можем вычислить, насколько набор данных D отличается от случайного набора данных.

Основное ограничение этого подхода заключается в том, что по мере увеличения размерности количество ячеек (b^d) увеличивается экспоненциально, а при фиксированном размере выборки n большинство ячеек будет пустым или будет иметь только одну точку, что затрудняет оценку расхождение. Метод также чувствителен к выбору параметра b. Вместо гистограмм и соответствующих EPMF мы также можем использовать методы оценки плотности (см. Раздел 10.2), чтобы определить совместную функцию плотности вероятности (PDF) для набора данных D и посмотреть, чем она отличается от PDF для случайных наборов данных. Однако проклятие размерности также создает проблемы для оценки плотности.

Распределение расстояния

Вместо того, чтобы пытаться оценить плотность, другой подход к определению кластеризации состоит в сравнении попарных расстояний между точками от D с расстояниями от случайно сгенерированных выборок R_j из нулевого распределения. То есть мы создаем EPMF из матрицы близости W для D [Ур. (11.22)] путем объединения расстояний в интервалы b:

$$f(i) = P\big(\, w_{pq} \in bin\ i \mid x_p, x_q \in D, p < q \,\big) = \frac{\big|\{w_{pq} \in bin\ i\}\big|}{n(n-1)/2}$$

Точно так же для каждого из образцов R_j мы можем определить EPMF для попарных расстояний, обозначенных g_j. Наконец, мы можем вычислить KL-расходимости между f и g_j, используя ур. (11.31). Ожидаемое расхождение указывает на степень, в которой D отличается от нулевого (случайного) распределения.

Статистика Хопкинса

Статистика Хопкинса — это тест с разреженной выборкой на пространственную случайность. Учитывая набор данных D, содержащий n точек, мы генерируем t случайных подвыборок R_i по m точек в каждой, где $m \ll n$. Эти образцы берутся из того же пространства данных, что и D, и генерируются равномерно случайным образом по каждому измерению. Кроме того, мы также генерируем t подвыборок из m точек непосредственно из D, используя выборку без замены. Обозначим через D_i $i-$ю прямую подвыборку. Затем мы вычисляем минимальное расстояние между каждой точкой $x_j \in D_i$ и точками в D

$$\delta_{min}(x_j) = \min_{x_i \in D, x_i \neq x_j} \{\delta(x_j, x_i)\}$$

Точно так же мы вычисляем минимальное расстояние $\delta_{min}(y_j)$ между точкой $y_j \in R_i$ и точками в D.

Статистика Хопкинса (в d-измерениях) для i -й пары выборок R_i и D_i затем определяется как

$$HS_i = \frac{\sum_{y_j \in R_i} \big(\delta_{min}(y_j)\big)^d}{\sum_{y_j \in R_i} \big(\delta\, min(y_j)\big)^d + \sum_{x_j \in D_i} \big(\delta_{min}(x_j)\big)^d}$$

Эта статистика сравнивает распределение ближайших соседей случайно сгенерированных точек с таким же распределением для случайных подмножеств точек из D. Если данные хорошо сгруппированы, мы ожидаем, что значения $\delta_{min}(x_j)$ будут меньше по сравнению со значениями $\delta_{min}(y_j)$ и в этом случае HSi стремится к 1. Если расстояние до обоих ближайших соседей одинаково, тогда HS_i принимает значения, близкие к 0.5, что указывает на то, что данные, по существу, случайны и явной кластеризации нет. Наконец, если значения $\delta_{min}(x_j)$ больше по сравнению со значениями $\delta_{min}(y_j)$, то HS_i стремится к 0 и указывает на точечное отталкивание без кластеризации. Затем из t различных значений HS_i мы можем вычислить среднее значение и дисперсию статистики, чтобы определить, является ли D кластеризируемым или нет.

Глава 12. Вероятностная классификация

12.1 Классификатор Байеса

Пусть набор обучающих данных **D** состоит из n точек x_i в d-мерном пространстве, и пусть y_i обозначает класс для каждой точки, где $y_i \in \{c_1, \dots, c_k\}$. Классификатор Байеса напрямую использует теорему Байеса для прогнозирования класса для нового тестового экземпляра x. Он оценивает апостериорную вероятность $\mathrm{P}(c_i|x)$ для каждого класса c_i и выбирает класс, который имеет наибольшую вероятность. Прогнозируемый класс для x задается как:

$$\hat{y} = argmax_{c_i}\{\mathrm{P}(c_i|x)\} \tag{12.1}$$

Теорема Байеса позволяет нам инвертировать апостериорную вероятность в терминах правдоподобия и априорной вероятности следующим образом:

$$\mathrm{P}(c_i|x) = \frac{\mathrm{P}(x|c_i) \cdot P(c_i)}{P(x)}$$

где $\mathrm{P}(x|c_i)$ – *вероятность правдоподобия*, определяемая как вероятность наблюдения x при условии, что истинным классом является c_i, $P(c_i)$ - *априорная вероятность* класса c_i, а $P(x)$ - вероятность наблюдения x любого из k классов, заданных как

$$P(x) = \sum_{j=1}^{k} \mathrm{P}\left(x\middle|c_j\right) \cdot P(c_j)$$

Поскольку $P(x)$ фиксирован для данной точки, правило Байеса [Ур. (12.1)] можно переписать как

$$\begin{gathered} \hat{y} = argmax_{c_i}\{\mathrm{P}(c_i|x)\} = \\ argmax_{c_i}\left\{\frac{\mathrm{P}(x|c_i) \cdot P(c_i)}{P(x)}\right\} = \\ argmax_{c_i}\{\mathrm{P}(x|c_i) \cdot P(c_i)\} \end{gathered} \tag{12.2}$$

Другими словами, предсказанный класс существенно зависит от вероятности этого класса с учетом его априорной вероятности.

12.1.1 Оценка априорной вероятности

Чтобы классифицировать точки, мы должны оценить вероятность и априорные вероятности непосредственно из набора обучающих данных **D**. Пусть $\mathbf{D}_i$ обозначает подмножество точек в **D**, которые помечены классом c_i:

$$\boldsymbol{D_i} = \left\{x_j \in D\middle|x_j \text{ имеет класс } y_j = c_i\right\}$$

Пусть размер набора данных **D** задан как |**D**| = n, и пусть размер каждого подмножества $\mathbf{D}_i$, зависящего от класса, задается как | $\mathbf{D}_i$ | = n_i. Априорную вероятность для класса c_i можно оценить следующим образом:

$$\hat{P}(c_i) = \frac{n_i}{n}$$

12.1.2 Оценка правдоподобия

Чтобы оценить вероятность $\mathrm{P}(x|c_i)$, мы должны оценить совместную вероятность x по всем d измерениям, то есть мы должны оценить $P(x = (x_1, x_2, \dots, x_d)|c_i)$.

Числовые атрибуты

Предполагая, что все измерения числовые, мы можем оценить совместную вероятность x с помощью либо непараметрического, либо параметрического подхода. Мы рассматриваем непараметрический подход в разделе 12.3.

В параметрическом подходе мы обычно предполагаем, что каждый класс c_i нормально распределен вокруг некоторого среднего μ_i с соответствующей ковариационной матрицей Σ_i, оба из которых оцениваются по D_i. Таким образом, для класса c_i плотность вероятности в точке x задается как

$$f_i(x) = f(x|\mu_i, \Sigma_i) = \frac{1}{\left(\sqrt{2\pi}\right)^d \sqrt{|\Sigma_i|}} \exp\left\{-\frac{(x-\mu_i)^T \Sigma_i^{-1}(x-\mu_i)}{2}\right\} \tag{12.3}$$

Поскольку c_i характеризуется непрерывным распределением, вероятность любой данной точки должна быть равна нулю, т. е. $\mathrm{P}(x|c_i) = 0$. Однако мы можем вычислить вероятность, рассматривая небольшой интервал $\epsilon > 0$ с центром в x:

$$\mathrm{P}(x|c_i) = 2\epsilon \cdot f_i(x)$$

Тогда апостериорная вероятность определяется как

$$\mathrm{P}(c_i|x) = \frac{2\epsilon \cdot f_i(x)P(c_i)}{\sum_{j=1}^{k} 2\epsilon \cdot f_j(x)P(j)} = \frac{f_i(x)P(c_i)}{\sum_{j=1}^{k} f_j(x)P(c_j)} \tag{12.4}$$

Кроме того, поскольку $\sum_{j=1}^{k} f_j(x)P(c_j)$ остается фиксированным для x, мы можем предсказать класс для x, изменив уравнение (12.2) следующим образом:

$$\hat{y} = argmax_{c_i}\{f_i(x)P(c_i)\}$$

Чтобы классифицировать числовую контрольную точку x, байесовский классификатор оценивает параметры с помощью выборочного среднего и выборочной ковариационной матрицы. Выборочное среднее для класса c_i можно записать как

$$\hat{\mu}_i = \frac{1}{n_i} \sum_{x_j \in D_i} x_j$$

и выборочная матрица ковариации для каждого класса может быть оценена с помощью уравнения (2.30) следующим образом

$$\hat{\Sigma}_i = \frac{1}{n_i} Z_i^T Z_i$$

где Z_i - центрированная матрица данных для класса c_i, заданная как $Z_i = D_i - 1 \cdot \hat{\mu}_i^T$. Эти значения могут быть использованы для оценки плотности вероятности в уравнении (12.3) как $\hat{f}_i(x) = f\left(x|\hat{\mu}_i, \hat{\Sigma}_i\right)$.

Алгоритм 12.1 показывает псевдокод байесовского классификатора. Учитывая входной набор данных **D**, метод оценивает априорную вероятность, среднее значение и матрицу ковариации для каждого класса. Для тестирования с заданной контрольной точкой x он просто возвращает класс с максимальной апостериорной вероятностью. В стоимости обучения преобладает этап вычисления ковариационной матрицы, который занимает время $O(nd^2)$.

Алгоритм 12.1: Классификатор Байеса

BAYESCLASSIFIER ($\mathbf{D} = \{(\mathbf{x}_j, y_j)\}_{j=1}^n$):

for $i = 1, \ldots, k$ **do**

  $\mathbf{D}_i \leftarrow \{\mathbf{x}_j \mid y_j = c_i, j = 1, \ldots, n\}$ // class-specific subsets

  $n_i \leftarrow |\mathbf{D}_i|$ // cardinality

  $\hat{P}(c_i) \leftarrow n_i / n$ // prior probability

  $\hat{\boldsymbol{\mu}}_i \leftarrow \frac{1}{n_i} \sum_{\mathbf{x}_j \in \mathbf{D}_i} \mathbf{x}_j$ // mean

  $\mathbf{Z}_i \leftarrow \mathbf{D}_i - \mathbf{1}_{n_i} \hat{\boldsymbol{\mu}}_i^T$ // centered data

  $\hat{\boldsymbol{\Sigma}}_i \leftarrow \frac{1}{n_i} \mathbf{Z}_i^T \mathbf{Z}_i$ // covariance matrix

return $\hat{P}(c_i), \hat{\boldsymbol{\mu}}_i, \hat{\boldsymbol{\Sigma}}_i$ *for all* $i = 1, \ldots, k$

TESTING (**x and** $\hat{P}(c_i)$, $\hat{\boldsymbol{\mu}}_i$, $\hat{\boldsymbol{\Sigma}}_i$, **for all** $i \in [1, k]$):

$\hat{y} \leftarrow \arg\max_{c_i} \{ f(\mathbf{x} | \hat{\boldsymbol{\mu}}_i, \hat{\boldsymbol{\Sigma}}_i) \cdot P(c_i) \}$

return $\hat{y}$

Категориальные атрибуты

Если атрибуты являются категориальными, вероятность правдоподобия может быть вычислена с использованием подхода категориального моделирования данных, представленного в главе 3. Формально пусть X_j будет категориальным атрибутом в области $dom(X_j) = \{a_{j1}, a_{j2}, \ldots, a_{jm_j}\}$, то есть атрибут X_j может принимать m_j различных категориальных значений. Каждый категориальный атрибут X_j моделируется как m_j -мерная многомерная случайная величина Бернулли X_j, которая принимает m_j различных векторных значений $e_{j1}, e_{j2}, \ldots, e_{jm_j}$, где e_{jr} - r-й стандартный базисный вектор в $\mathbb{R}^{m_j}$ и соответствует r-му значению или символу $a_{jr} \in dom(X_j)$. Весь d-мерный набор данных моделируется как векторная случайная величина $X = (X_1, X_2, \ldots, X_d)^T$. Пусть $d' = \sum_{j=1}^d m_j$; категориальная точка $x = (x_1, x_2, \ldots, x_d)^T$ поэтому представляется как d'-мерный двоичный вектор.

$$v = \begin{pmatrix} v_1 \\ \vdots \\ v_d \end{pmatrix} = \begin{pmatrix} e_{1r_1} \\ \vdots \\ e_{dr_d} \end{pmatrix}$$

где $v_j = e_{jr_j}$ при условии, что $x_j = a_{jr_j}$ — это r_j-е значение в области определения X_j. Вероятность категориальной точки x получается из совместной функции вероятности (PMF) для векторной случайной величины X:

$$\mathrm{P}(x|c_i) = f(v|c_i) = f\left(X_1 = e_{1r_1}, \ldots, X_d = e_{dr_d} \middle| c_i\right) \tag{12.5}$$

Вышеупомянутую совместную PMF можно оценить непосредственно из данных $\mathbf{D}_i$ для каждого класса c_i следующим образом:

$$\hat{f}(v|c_i) = \frac{n_i(v)}{n_i}$$

где $n_i(v)$ - количество раз, когда значение v встречается в классе c_i. К сожалению, если вероятность в точке v равна нулю для одного или обоих классов, это приведет к нулевому значению апостериорной вероятности. Чтобы избежать нулевых вероятностей, один из подходов состоит в том, чтобы ввести небольшую априорную вероятность для всех возможных значений векторной случайной величины X. Один простой подход - принять *псевдосчет*, равный 1 для каждого значения, то есть предположить, что каждое значение X встречается, по крайней мере, один раз, и увеличить этот базовый счетчик, равный 1, фактическим числом появлений наблюдаемого значения v в классе c_i. Скорректированная вероятность в v тогда дается как

$$\hat{f}(v|c_i) = \frac{n_i(v) + 1}{n_i + \prod_{j=1}^{d} m_j} \tag{12.6}$$

где $\prod_{j=1}^{d} m_j$ дает количество возможных значений **X**. Расширение кода в алгоритме 12.1 для включения категориальных атрибутов относительно простое; все, что требуется, — это вычислить совместную PMF для каждого класса, используя уравнение. (12.6).

Проблемы

Основная проблема с классификатором Байеса заключается в отсутствии достаточного количества данных для надежной оценки совместной плотности вероятности или функции вероятности, особенно для многомерных данных. Например, для числовых атрибутов мы должны оценить ковариации $O(d^2)$, и по мере увеличения размерности это требует от нас оценки слишком большого количества параметров. Для категориальных атрибутов мы должны оценить совместную вероятность для всех возможных значений v, заданных как $\prod_j |dom(X_j)|$. Даже если каждый категориальный атрибут имеет только два значения, нам нужно будет оценить вероятность для 2^d значений. Однако, поскольку для v может быть не более n различных значений, большинство отсчетов будут равны нулю. Для решения некоторых из этих проблем мы можем использовать сокращенный набор параметров на практике, как описано далее.

12.2. Наивный байесовский классификатор

Ранее мы видели, что полный байесовский подход чреват проблемами, связанными с оценкой, особенно с большим количеством измерений. Наивный байесовский подход основан на простом предположении, что все атрибуты независимы. Это приводит к гораздо более простому, но удивительно эффективному классификатору на практике. Допущение о независимости подразумевает, что вероятность принадлежности может быть разложена на произведение размерных вероятностей:

$$P(\mathbf{x} \mid c_i) = P(x_1, x_2, \ldots, x_d \mid c_i) = \prod_{j=1}^{d} P(x_j \mid c_i) \tag{12.7}$$

Числовые признаки

Для числовых признаков мы делаем стандартное допущение, что каждый из них является нормально распределенным в любом классе c_i. Пусть μ_{ij} и σ_{ij}^2 обозначают среднее значение и дисперсию для признака X_j в классе c_i. Вероятность принадлежности к классу c_i для измерения X_j определяется как:

$$P(x_j \mid c_i) \propto f(x_j \mid \mu_{ij}, \sigma_{ij}^2) = \frac{1}{\sqrt{2\pi}\sigma_{ij}} \exp\left\{-\frac{(x_j - \mu_{ij})^2}{2\sigma_{ij}^2}\right\}$$

В данном случае, наивное допущение соответствует установлению всех ковариаций равным нулю в $\mathbf{\Sigma}_i$, то есть

$$\mathbf{\Sigma}_i = \begin{pmatrix} \sigma_{i1}^2 & 0 & \dots & 0 \\ 0 & \sigma_{i2}^2 & \dots & 0 \\ \vdots & \vdots & \ddots & \\ 0 & 0 & \dots & \sigma_{id}^2 \end{pmatrix}$$

Это дает

$$|\mathbf{\Sigma}_i| = \det(\mathbf{\Sigma}_i) = \sigma_{i1}^2 \sigma_{i2}^2 \cdots \sigma_{id}^2 = \prod_{j=1}^{d} \sigma_{ij}^2$$

Далее получаем

$$\mathbf{\Sigma}_i^{-1} = \begin{pmatrix} \frac{1}{\sigma_{i1}^2} & 0 & \dots & 0 \\ 0 & \frac{1}{\sigma_{i2}^2} & \dots & 0 \\ \vdots & \vdots & \ddots & \\ 0 & 0 & \dots & \frac{1}{\sigma_{id}^2} \end{pmatrix}$$

при условии, что $\sigma_{ij}^2 \neq 0$ для всех j. Окончательно,

$$(\mathbf{x} - \boldsymbol{\mu}_i)^T \mathbf{\Sigma}_i^{-1} (\mathbf{x} - \boldsymbol{\mu}_i) = \sum_{j=1}^{d} \frac{(x_j - \mu_{ij})^2}{\sigma_{ij}^2}$$

Подстановка этого в формулу (12.3) дает нам

$$P(\mathbf{x} \mid c_i) = \frac{1}{(\sqrt{2\pi})^d \sqrt{\prod_{j=1}^{d} \sigma_{ij}^2}} \exp\left\{-\sum_{j=1}^{d} \frac{\left(x_j - \mu_{ij}\right)^2}{2\sigma_{ij}^2}\right\}$$

$$= \prod_{j=1}^{d} \left(\frac{1}{\sqrt{2\pi}\sigma_{ij}} \exp\left\{-\frac{\left(x_j - \mu_{ij}\right)^2}{2\sigma_{ij}^2}\right\}\right)$$

$$= \prod_{j=1}^{d} P\left(x_j \mid c_i\right)$$

что эквивалентно формуле (12.7). Другими словами, совместная вероятность была разложена на произведение вероятностей по каждому измерение, как того требует допущение о независимости.

Наивный байесовский классификатор использует выборочное среднее $\hat{\mu}_i = (\hat{\mu}_{i1}, \ldots, \hat{\mu}_{id})^T$ и *диагональ* выборочной ковариационной матрицы $\hat{\boldsymbol{\Sigma}}_i = diag(\sigma_{i1}^2, \ldots, \sigma_{id}^2)$ для каждого класса c_i. Таким образом всего $2d$ параметром должны быть оценены в соответствии с выборочным средним и выборочной дисперсией для каждого измерения X_j.

В алгоритме 12.2 продемонстрирован псевдокод для наивного байесовского классификатора. Учитывая входной набор данных **D**, метод оценивает априорную вероятность и среднее значение для каждого класса. Затем он вычисляет дисперсию $\hat{\sigma}_{ij}^2$ для каждого из атрибутов X_j со всеми d дисперсиями для класса c_i, хранящегося в векторе $\hat{\sigma}_i$. Дисперсия для атрибута X_j получается путем предварительного центрирования данных для класса $\mathbf{D}_i$ через $\mathbf{Z}_i = \mathbf{D}_i - \mathbf{1} \cdot \hat{\boldsymbol{\mu}}_i^T$. Обозначим через Z_{ij} центрированные данных для класса c_i соответствующего атрибута X_j. Тогда дисперсия выражается как $\hat{\sigma} = \frac{1}{n_i} Z_{ij}^T Z_{ij}$

Алгоритм 12.2: Наивный классификатор Байеса

NAIVEBAYES ($\mathbf{D} = \{(\mathbf{x}_j, y_j)\}_{j=1}^{n}$):

for $i = 1, \ldots, k$ **do**
  $\mathbf{D}_i \leftarrow \{\mathbf{x}_j \mid y_j = c_i, j = 1, \ldots, n\}$ // class-specific subsets
  $n_i \leftarrow |\mathbf{D}_i|$ // cardinality
  $\hat{P}(c_i) \leftarrow n_i / n$ // prior probability
  $\hat{\boldsymbol{\mu}}_i \leftarrow \frac{1}{n_i} \sum_{\mathbf{x}_j \in \mathbf{D}_i} \mathbf{x}_j$ // mean
  $\mathbf{Z}_i = \mathbf{D}_i - \mathbf{1} \cdot \hat{\boldsymbol{\mu}}_i^T$ // centered data for class c_i
  for $j = 1, .., d$ **do** // class-specific variance for X_j
    $\hat{\sigma}_{ij}^2 \leftarrow \frac{1}{n_i} Z_{ij}^T Z_{ij}$ // variance
  $\hat{\boldsymbol{\sigma}}_i = \left(\hat{\sigma}_{i1}^2, \ldots, \hat{\sigma}_{id}^2\right)^T$ // class-specific attribute variances
return $\hat{P}(c_i), \hat{\boldsymbol{\mu}}_i, \hat{\boldsymbol{\sigma}}_i$ for all $i = 1, \ldots, k$

TESTING (**x and** $\hat{P}(c_i)$, $\hat{\boldsymbol{\mu}}_i$, $\hat{\boldsymbol{\sigma}}_i$, **for all** $i \in [1, k]$):

$\hat{y} \leftarrow \arg\max_{c_i} \left\{ \hat{P}(c_i) \prod_{j=1}^{d} f(x_j | \hat{\mu}_{ij}, \hat{\sigma}_{ij}^2) \right\}$

return $\hat{y}$

Обучение наивного байесовского классификатора происходит очень быстро, вычислительная сложность - *O(nd)*. Для тестирования с заданной контрольной точкой **x** он просто возвращает класс с максимальной апостериорной вероятностью, полученной как произведение правдоподобия для каждого измерения и априорной вероятности класса.

Категориальные признаки

Допущение о независимости приводит к упрощению совместной функции массы вероятности в уравнении (12.5), которое можно переписать следующим образом:

$$P(\mathbf{x} \mid c_i) = \prod_{j=1}^{d} P(x_j \mid c_i) = \prod_{j=1}^{d} f\left(\mathbf{X}_j = \mathbf{e}_{jr_j} \mid c_i\right)$$

Где $f\left(\mathbf{X}_j = \mathbf{e}_{jr_j} \mid c_i\right)$ – функция массы вероятности для X_j, которую можно оценить из $\mathbf{D}_i$, как показано далее:

$$\hat{f}(\mathbf{v}_j \mid c_i) = \frac{n_i(\mathbf{v}_j)}{n_i}$$

где $n_i(\mathbf{v}_j)$ – наблюдаемая частого значения $\mathbf{v}_j = \mathbf{e}_j r_j$, соответственно r_j категориальное значение a_{jrj} для атрибута X_j в классе c_i. Как и в полном байесовском классификаторе, если количество равно нулю, мы можем использовать метод псевдоподсчета для полученной априорной вероятности. Скорректированная оценка с псевдоподсчетом выглядит следующим образом

$$\hat{f}(\mathbf{v}_j \mid c_i) = \frac{n_i(\mathbf{v}_j) + 1}{n_i + m_j}$$

где $m_j=|dom(X_j)|$. Расширить код в алгоритме 12.2 для включения категориальных атрибутов несложно.

12.3 Классификатор K-ближайших соседей.

В предыдущих разделах мы рассмотрели параметрический подход к оценке правдоподобия $P(\mathbf{x}|c_i)$. В этом разделе мы рассматриваем непараметрический подход, который не делает никаких предположений о базовой совместной функции плотности вероятности. Вместо этого он напрямую использует выборку данных для оценки плотности. Мы проиллюстрируем непараметрический подход, используя оценку плотности ближайших соседей, которая приводит к классификатору *K*-ближайших соседей (KNN).

Пусть **D** - обучающий набор данных, содержащий n точек $\mathbf{x}_i \in \mathbb{R}^d$, и пусть $\mathbf{D}_i$ обозначает подмножество точек в **D**, которые помечены классом c_i, где $n_i=|\mathbf{D}_i|$. Для контрольной точки $\mathbf{x} \in \mathbb{R}^d$ и K, количества рассматриваемых соседей, пусть r обозначает расстояние от **x** до K-го ближайшего соседа в **D**.

Рассмотрим d-мерный гипершар радиуса r вокруг тестовой точки x, определенный как

$$B_d(\mathbf{x}, r) = \{\mathbf{x}_i \in \mathbf{D} \mid \delta(\mathbf{x}, \mathbf{x}_i) \leq r\}$$

Здесь $\delta(\mathbf{x}, \mathbf{x}_i)$ - это расстояние между **x** и $\mathbf{x}_i$, которое обычно считается евклидовым расстоянием, т.е. $\delta(\mathbf{x}, \mathbf{x}_i) = \|\mathbf{x} - \mathbf{x}_i\|_2$. Однако можно использовать и другие метрики расстояния. Мы предполагаем, что $|B_d(\mathbf{x}, r)| = K$.

Пусть K_i обозначает количество точек среди K ближайших соседей точки **x**, помеченных классом c_i, т. е.

$$K_i = \{\mathbf{x}_j \in B_d(\mathbf{x}, r) \mid y_j = c_i\}$$

Плотность условной вероятности класса в точке x можно оценить как долю точек из класса ci, лежащих внутри гипершара, деленную на его объем, т. е.

$$\hat{f}(\mathbf{x} \mid c_i) = \frac{K_i/n_i}{V} = \frac{K_i}{n_i V}$$

где $V = \text{vol}(B_d(\mathbf{x}, r))$ - объем d-мерного гипершара [уравнение (6.4)].

Используя уравнение (12.4) апостериорная вероятность $P(c_i \mid \mathbf{x})$ может быть оценена как

$$P(c_i \mid \mathbf{x}) = \frac{\hat{f}(\mathbf{x} \mid c_i)\hat{P}(c_i)}{\sum_{j=1}^{k} \hat{f}(\mathbf{x} \mid c_j)\hat{P}(c_j)}$$

Однако, так как $\hat{P}(c_i) = \frac{n_i}{n}$, мы имеем

$$\hat{f}(\mathbf{x} \mid c_i)\hat{P}(c_i) = \frac{K_i}{n_i V} \cdot \frac{n_i}{n} = \frac{K_i}{nV}$$

Таким образом, апостериорная вероятность представляется как

$$P(c_i \mid \mathbf{x}) = \frac{\frac{K_i}{nV}}{\sum_{j=1}^{k} \frac{K_j}{nV}} = \frac{K_i}{K}$$

Наконец, предсказанный класс для x равен

$$\hat{y} = \arg \max_{c_i} \{P(c_i \mid \mathbf{x})\} = \arg \max_{c_i} \left\{\frac{K_i}{K}\right\} = \arg \max_{c_i} \{K_i\}$$

Поскольку K фиксировано, классификатор KNN предсказывает класс **x** как класс большинства среди своих K ближайших соседей.

Глава 13. Классификатор дерева решений

Пусть обучающий набор $\mathbf{D} = \{\mathbf{x}_i, y_i\}_{i=1}^n$ состоит из n точек в d-мерном пространстве, причем y_i - метка класса для точки $\mathbf{x}_i$. Мы предполагаем, что измерения или атрибуты X_j являются числовыми или категориальными и что существует k различных классов, так что $y_i \in \{c_1, c_2, \ldots, c_k\}$. Классификатор дерева решений - это рекурсивная модель дерева на основе секционирования, которая предсказывает класс $\hat{y}_i$ для каждой точки $\mathbf{x}_i$. Пусть **R** обозначает пространство данных, охватывающее множество входных точек **D**. Дерево решений использует осепараллельную гиперплоскость для разделения пространства данных **R** на два результирующих полупространства или области, скажем $\mathbf{R}_1$ и $\mathbf{R}_2$, что также вызывает разбиение входных точек на $\mathbf{D}_1$ и $\mathbf{D}_2$ соответственно. Каждая из этих областей рекурсивно разбивается с помощью осепараллельных гиперплоскостей до тех пор, пока точки внутри индуцированного разбиения не станут относительно чистыми с точки зрения их меток классов, то есть большинство точек принадлежат к одному и тому же классу. Результирующая иерархия разделенных решений образует модель дерева решений, с конечными узлами, помеченными классом большинства среди точек в этих регионах. Чтобы классифицировать новую тестовую точку, мы должны рекурсивно оценить, к какому полупространству она принадлежит, пока не достигнем листового узла в дереве решений, в котором мы предсказываем ее класс как метку листа.

13.1 Деревья решений

Дерево решений состоит из внутренних узлов, представляющих решения, соответствующие гиперплоскостям или точкам разделения (т. е. в каком полупространстве находится данная точка), и конечных узлов, представляющих области или разделы пространства данных, которые помечены классом большинства. Область характеризуется подмножеством точек данных, лежащих в этой области.

Осепараллельные гиперплоскости

Гиперплоскость h(x) определяется как множество всех точек **x**, удовлетворяющих следующему уравнению:

$$h(\mathbf{x}) : \mathbf{w}^T\mathbf{x} + b = 0 \qquad (13.1)$$

Здесь $\mathbf{w} \in R^d$ - *весовой вектор*, нормальный к гиперплоскости, а b - смещение гиперплоскости от начала координат. Дерево решений рассматривает только *осепараллельные гиперплоскости*, то есть весовой вектор должен быть параллелен одному из исходных измерений или осям X_j. Иными словами, весовой вектор w *априори* ограничен одним из стандартных базисных векторов $\{\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_d\}$, где $\mathbf{e}_i \in R^d$ имеет 1 для j-го измерения и 0 для всех других измерений. Если $\mathbf{x} = (x_1, x_2, \ldots, x_d)^T$ и предполагая $\mathbf{w} = \mathbf{e}_j$, мы можем переписать уравнение (13.1) как

$h(\mathbf{x}) : \mathbf{e}_j^T\mathbf{x} + b = 0$, из чего следует

$h(\mathbf{x}) : x_j + b = 0$

где выбор смещения b дает различные гиперплоскости вдоль размерности X_j.

Точки Разделения

Гиперплоскость определяет точку принятия решения или разделения, поскольку она разбивает пространство данных **R** на два полупространства. Все точки x, такие, h(**x**) ≤ 0, находятся на гиперплоскости или на одной стороне гиперплоскости, тогда как все точки, такие, что h(**x**) > 0, находятся на другой стороне. Точки разделения, связанного с осью параллельной гиперплоскости можно записать как h(**x**) ≤ 0, что означает, что $x_i + b \leq 0$ или $x_i \leq -b$. Поскольку x_i - это некоторое значение из измерения X_j и смещение b может быть выбрано как любое значение, общая форма точки разделения для числового атрибута X_j задается как $X_j \leq v$, где v = - b - некоторое значение в области атрибута X_j. Таким образом, решение или точка разделения $X_j \leq v$ разбивает пространство входных данных R на две области $\mathbf{R}_Y$ и $\mathbf{R}_N$, которые обозначают множество *всех возможных точек*, удовлетворяющих решению, и тех, которые не удовлетворяют.

Раздел Данных

Каждое разбиение **R** на $\mathbf{R}_Y$ и $\mathbf{R}_N$ также вызывает двоичное разбиение соответствующих входных точек данных **D**. То есть точка разбиения вида $X_j \leq v$ вызывает разбиение данных

$\mathbf{D}_Y = \{x \mid x \in \mathbf{D}, xj \leq v\}$

$\mathbf{D}_N = \{x \mid x \in \mathbf{D}, xj > v\}$

где $\mathbf{D}_Y$ - подмножество точек данных, лежащих в области $\mathbf{R}_Y$, а $\mathbf{D}_N$ - подмножество входных точек, лежащих в области $\mathbf{R}_N$.

Чистота

Чистота области $\mathbf{R}_j$ определяется в терминах смеси классов для точек в соответствующем разделе данных $\mathbf{D}_j$. Формально, чистота — это доля точек с меткой большинства в $\mathbf{D}_j$, т. е.

$$purity(D_j) = max_i \left\{ \frac{n_{ji}}{n_j} \right\} \tag{13.2}$$

где $n_j = |\mathbf{D}_j|$ - общее количество точек данных в области $\mathbf{R}_j$, а $n_{j\,i}$ - количество точек в $\mathbf{D}_j$ с меткой класса c_i.

Категориальные Атрибуты

В дополнение к числовым атрибутам дерево решений может также обрабатывать категориальные данные. Для категориального атрибута X_j точки разделения или решения имеют вид $X_j \in V$, где $V \subset \text{dom}(X_j)$, а $\text{dom}(X_j)$ обозначает область для X_j. Интуитивно это расщепление можно считать категориальным аналогом гиперплоскости. Это приводит к двум "полупространствам", одна область $\mathbf{R}_Y$ состоит из точек **x**, удовлетворяющих условию $x_i \in V$, а другая область $\mathbf{R}_N$ состоит из точек, удовлетворяющих условию x_i V.

Правила Принятия Решений

Одним из преимуществ деревьев решений является то, что они создают модели, которые относительно легко интерпретировать. В частности, дерево может быть прочитано как набор

решающих правил, причем антецедент каждого правила включает в себя решения по внутренним узлам вдоль пути к листу, а его следствием является метка листового узла. Далее, поскольку все области не пересекаются и охватывают все пространство, набор правил можно интерпретировать как набор альтернатив или дизъюнкций.

13.2 Алгоритм дерева решений

Псевдокод для построения модели дерева решений показан в алгоритме 13.1. Он принимает в качестве входных данных обучающий набор данных **D** и два параметра η и π, где η - размер листа, а π - порог чистоты листа. Различные точки разделения оцениваются для каждого атрибута в **D**. числовые решения имеют вид $Xj \leq v$ для некоторого значения v в диапазоне значений атрибута Xj, а категориальные решения имеют вид $Xj \in V$ для некоторого подмножества значений в области Xj . Наилучшая точка разбиения выбирается для разбиения данных на два подмножества, $\mathbf{D}_Y$ и $\mathbf{D}_N$, где $\mathbf{D}_Y$ соответствует всем точкам $x \in \mathbf{D}$, удовлетворяющим решению разбиения, а $\mathbf{D}_N$ соответствует всем точкам, не удовлетворяющим решению разбиения. Затем метод дерева решений рекурсивно вызывается на $\mathbf{D}_Y$ и $\mathbf{D}_N$. Для остановки процесса рекурсивного секционирования можно использовать ряд условий остановки. Самое простое условие основано на размере раздела **D**. Если число точек n в **D** падает ниже заданного пользователем порога размера η, то мы останавливаем процесс разбиения и делаем **D** листом. Это условие предотвращает чрезмерную подгонку модели к обучающему набору, избегая моделировать очень маленькие подмножества данных. Одного размера недостаточно, потому что, если раздел уже чистый, тогда нет смысла разбивать его дальше. Таким образом, рекурсивное разбиение также прекращается, если чистота D выше порога чистоты π. далее приводится подробная информация о том, как оцениваются и выбираются точки разделения.

Алгоритм 13.1: Алгоритм дерева решений

```
DECISIONTREE (D, η, π):
n ← |D| // partition size
n_i ← |{x_j | x_j ∈ D, y_j = c_i}| // size of class c_i
purity(D) ← max_i { n_i / n }
if n ≤ η or purity(D) ≥ π then // stopping condition
    c* ← argmax_{c_i} { n_i / n } // majority class
    create leaf node, and label it with class c*
    return
(split point*, score*) ← (∅, 0) // initialize best split point
foreach (attribute X_j) do
    if (X_j is numeric) then
        (v, score) ← EVALUATE-NUMERIC-ATTRIBUTE(D, X_j)
        if score > score* then (split point*, score*) ← (X_j ≤ v, score)
    else if (X_j is categorical) then
        (V, score) ← EVALUATE-CATEGORICAL-ATTRIBUTE(D, X_j)
        if score > score* then (split point*, score*) ← (X_j ∈ V, score)
// partition D into D_Y and D_N using split point*, and call
   recursively
D_Y ← {x ∈ D | x satisfies split point*}
D_N ← {x ∈ D | x does not satisfy split point*}
create internal node split point*, with two child nodes, D_Y and D_N
DECISIONTREE(D_Y); DECISIONTREE(D_N)
```

13.2.1 Меры Оценки С Разделением Точек

Учитывая точку разделения вида $X_j \leq v$ или $X_j \in V$ для числового или категориального атрибута соответственно, нам нужен объективный критерий оценки точки разделения. Интуитивно мы хотим выбрать точку разделения, которая дает лучшее разделение или различение между различными метками классов.

Энтропия

Энтропия, как правило, измеряет количество беспорядка или неопределенности в системе. В классификационной установке разбиение имеет более низкую энтропию (или низкий беспорядок), если оно относительно чисто, то есть если большинство точек имеют одну и ту же метку. С другой стороны, разделение имеет более высокую энтропию (или больше беспорядка), если метки классов смешаны, и нет класса большинства как такового.

Энтропия множества помеченных точек **D** определяется следующим образом:

$$H(D) = -\sum_{i=1}^{k} P(C_i|\boldsymbol{D}) \log_2 P(c_i|\boldsymbol{D}) \tag{13.3}$$

где P (c_i |**D**) - вероятность класса c_i в **D**, A k-число классов. Если область чиста, то есть имеет точки из одного класса, то энтропия равна нулю. С другой стороны, если все классы

перемешаны и каждый появляется с равной вероятностью P (c_i |**D**) = $\frac{1}{k}$, то энтропия имеет наибольшее значение, H(**D**) = $\log_2$ k.

Предположим, что точка разделения разбивает **D** на $\mathbf{D}_Y$ и $\mathbf{D}_N$. Определите энтропию разделения как взвешенную энтропию каждого из результирующих разбиений, заданную как

$$H(\boldsymbol{D}_Y, \boldsymbol{D}_N) = \frac{n_Y}{n} H(\boldsymbol{D}_Y) + \frac{n_N}{n} H(\boldsymbol{D}_N) \tag{13.4}$$

где n = | **D** | - количество точек в **D**, а n_Y = |$\mathbf{D}_Y$| и n_N = |$\mathbf{D}_N$ | - количество точек в $\mathbf{D}_Y$ и $\mathbf{D}_N$.

Чтобы увидеть, приводит ли точка разделения к уменьшению общей энтропии, мы определяем информационный прирост для данной точки разделения следующим образом:

$$Gain(\boldsymbol{D}, \boldsymbol{D}_Y, \boldsymbol{D}_N) = H(\boldsymbol{D}) - H(\boldsymbol{D}_Y, \boldsymbol{D}_N) \tag{13.5}$$

Чем выше прирост информации, тем больше уменьшение энтропии и тем лучше точка разделения. Таким образом, учитывая точки разбиения и соответствующие им разбиения, мы можем оценить каждую точку разбиения и выбрать ту, которая дает наибольший информационный выигрыш.

Индекс Джини

Другой распространенной мерой для оценки чистоты точки разделения является индекс Джини, определяемый следующим образом:

$$G(\boldsymbol{D}) = 1 - \sum_{i=1}^{k} P(c_i|\boldsymbol{D})^2 \tag{13.6}$$

Если разбиение чисто, то вероятность мажоритарного класса равна 1, а вероятность всех остальных классов равна 0, и, таким образом, индекс Джини равен 0. С другой стороны, когда каждый класс представлен одинаково, с вероятностью P (c_i |**D**) = $\frac{1}{k}$, то индекс Джини имеет значение $\frac{k-1}{k}$. Таким образом, более высокие значения индекса Джини указывают на больший беспорядок, а более низкие значения указывают на больший порядок с точки зрения меток классов.

Мы можем вычислить взвешенный индекс Джини точки разделения следующим образом:

$$G(\boldsymbol{D}_Y, \boldsymbol{D}_N) = \frac{n_Y}{n} G(\boldsymbol{D}_Y) + \frac{n_N}{n} G(\boldsymbol{D}_N)$$

где n, n_Y и n_N обозначают количество точек в областях **D**, $\mathbf{D}_Y$ и $\mathbf{D}_N$ соответственно. Чем ниже значение индекса Джини, тем лучше точка разделения.

Вместо энтропии и индекса Джини для оценки расщеплений можно использовать и другие показатели. Например, мера деревьев классификации и регрессии (CART) задается следующим образом

$$CART(\boldsymbol{D}_Y, \boldsymbol{D}_N) = 2\frac{n_Y}{n}\frac{n_N}{n}\sum_{i=1}^{k}|P(c_i|\boldsymbol{D}_Y) - P(c_i|\boldsymbol{D}_N)| \quad (13.7)$$

Таким образом, эта мера предпочитает точку разбиения, которая максимизирует разницу между массовой функцией вероятности класса для двух разбиений; чем выше мера CART, тем лучше точка разбиения.

13.2.2 Оценка Точек Разделения

Все меры оценки точки разделения, такие как энтропия [Ур. (13.3)], индекс Джини [Ур. (13.6)] и CART [Ур. (13.7)], рассмотренные в предыдущем разделе, зависят от класса вероятностной массовой функции (ВМФ) для **D**, а именно P (c_i |**D**), и класса ВМФ для результирующих разбиений $\mathbf{D}_Y$ и $\mathbf{D}_N$, а именно P (c_i |$\mathbf{D}_Y$) и P (c_i |$\mathbf{D}_N$). Обратите внимание, что мы должны вычислить класс ВМФ для всех возможных точек разделения; оценка каждой из них независимо приведет к значительным вычислительным затратам. Вместо этого можно постепенно вычислить ВМФ, как описано в следующих параграфах.

Числовые Атрибуты

Если X является числовым атрибутом, мы должны оценить точки разделения вида $X \leq$ v. даже если мы ограничим v лежащим в пределах диапазона значений атрибута X, для V все равно существует бесконечное число вариантов выбора. Один из разумных подходов состоит в том, чтобы рассматривать только средние точки между двумя последовательными различными значениями X в выборке D. Это происходит потому, что точки разбиения вида X $\leq$ v для v $\in$ [x_a , x_b), где x_a и x_b - два последовательных различных значения X в **D**, производят одно и то же разбиение **D** на $\mathbf{D}_Y$ и $\mathbf{D}_N$ и , таким образом, дают одни и те же оценки. Поскольку для X может быть не более n различных значений, необходимо учитывать не более n-1 средних значений.

Пусть {v_1, . . . , v_m } обозначает множество всех таких средних точек, таких что v1 < v2 < · · · < vm. Для каждой точки x $\leq$ V, то есть, чтобы оценить класс ВМФ:

$$\hat{P}(c_i|\boldsymbol{D}_Y) = \hat{P}(c_i|X \leq v) \quad (13.8)$$

$$\hat{P}(c_i|\boldsymbol{D}_Y) = \hat{P}(c_i|X > v) \quad (13.9)$$

Пусть *I* () - индикаторная переменная, которая принимает значение 1 только тогда, когда ее аргумент истинен, и 0 в противном случае. Используя теорему Байеса, мы имеем

$$\hat{P}(c_i|X \leq v) = \frac{\hat{P}(X \leq v|c_i)\hat{P}(c_i)}{\hat{P}(X \leq v)} = \frac{\hat{P}(X \leq v|c_i)\hat{P}(c_i)}{\sum_{j=1}^{k}\hat{P}(X \leq v|c_j)\hat{P}(c_j)} \quad (13.10)$$

Априорная вероятность для каждого класса в **D** может быть оценена следующим образом:

$$\hat{P}(c_i) = \frac{1}{n}\sum_{j=1}^{n} I(y_i = c_i) = \frac{n_i}{n} \tag{13.11}$$

где y_j -класс для точки $\mathbf{x}_j$, n = |**D**| - общее количество точек, а n_i - количество точек в **D** с классом c_i . Определите N_{vi} как число точек $xj \leq v$ с классом c_i , где x_j - значение точки данных x_j для атрибута X, заданное как

$$N_{vi} = \sum_{j=1}^{n} I(x_j \leq v \text{ and } y_i = c_i) \tag{13.12}$$

Затем мы можем оценить P ($X \leq v|ci$) следующим образом:

$$\begin{aligned} \hat{P}(X \leq v|c_i) = \frac{\hat{P}(X \leq v \text{ and } c_i)}{\hat{P}(c_i)} &= \left(\frac{1}{n}\sum_{j=1}^{n} I\big(x_j \leq v \text{ and } y_j = c_i\big)\right)/(n_i/n) \\ &= \frac{N_{vi}}{n_i} \end{aligned} \tag{13.13}$$

Подставляя уравнения. (13.11) и (13.13) в уравнение (13.10), и используя уравнение (13.8), мы имеем

$$\hat{P}(c_i|\boldsymbol{D}_Y) = \hat{P}(c_i|X \leq v) = \frac{N_{vi}}{\sum_{j=1}^{k} N_{vi}} \tag{13.14}$$

Мы можем оценить $\hat{P}(X > v|c_i)$ следующим образом:

$$\hat{P}(X > v|c_i) = 1 - \hat{P}(X \leq v|c_i) = 1 - \frac{N_{vi}}{n_i} = \frac{n_i - N_{vi}}{n_i} \tag{13.15}$$

Используя уравнения (13.11), (13.15) класс ВМФ $\hat{P}(c_i|\boldsymbol{D}_N)$ рассчитывается:

$$\hat{P}(c_i|\boldsymbol{D}_N) = \hat{P}(c_i|X > v) = \frac{\hat{P}(X > v|c_i)\hat{P}(c_i)}{\sum_{j=1}^{k} \hat{P}(X > v|c_i)\hat{P}(c_j)} = \frac{n_i - N_{vi}}{\sum_{j=1}^{k}(n_j - N_{vi})} \tag{13.16}$$

Алгоритм 13.2 показывает метод оценки точки разделения для числовых атрибутов. Цикл for в строке 4 перебирает все точки и вычисляет значения средних точек v и количество точек N_{vi} из класса c_i таким образом, что $xj \leq v$. Цикл for в строке 12 перечисляет все возможные точки разделения вида $X \leq v$, по одной для каждой средней точки v, и оценивает их с помощью критерия усиления [Ур. (13.5)]; лучшая точка разделения и оценка записываются и возвращаются. Можно также использовать любые другие оценочные меры. Однако для индекса Джини и CART более низкий балл лучше, в отличие от прироста, где более высокий балл лучше.

С точки зрения вычислительной сложности начальная сортировка значений X (строка 1) занимает время O (n log n). Стоимость вычисления средних точек и специфичных для класса отсчетов N_{vi} занимает время O (nk) (для цикла в строке 4). Стоимость вычисления балла также ограничена O (nk), поскольку общее число средних точек v может быть не более n (для цикла в строке 12). Таким образом, общая стоимость оценки числового атрибута равна O(n log n + nk). Игнорируя k, поскольку это обычно небольшая константа, общая стоимость выведения числовой оценки точки разделения равна O(n log n).

EVALUATE-NUMERIC-ATTRIBUTE $(\mathbf{D}, X)$:

1 sort $\mathbf{D}$ on attribute X, so that $x_j \leq x_{j+1}, \forall j = 1, \ldots, n-1$
2 $\mathcal{M} \leftarrow \emptyset$ // set of midpoints
3 **for** $i = 1, \ldots, k$ **do** $n_i \leftarrow 0$
4 **for** $j = 1, \ldots, n-1$ **do**
5 **if** $y_j = c_i$ **then** $n_i \leftarrow n_i + 1$ // running count for class c_i
6 **if** $x_{j+1} \neq x_j$ **then**
7 $v \leftarrow \frac{x_{j+1} + x_j}{2}$; $\mathcal{M} \leftarrow \mathcal{M} \cup \{v\}$ // midpoints
8 **for** $i = 1, \ldots, k$ **do**
9 $N_{vi} \leftarrow n_i$ // Number of points such that $x_j \leq v$ and $y_j = c_i$
10 **if** $y_n = c_i$ **then** $n_i \leftarrow n_i + 1$
 // evaluate split points of the form $X \leq v$
11 $v^* \leftarrow \emptyset$; $score^* \leftarrow 0$ // initialize best split point
12 **forall** $v \in \mathcal{M}$ **do**
13 **for** $i = 1, \ldots, k$ **do**
14 $\hat{P}(c_i|\mathbf{D}_Y) \leftarrow \frac{N_{vi}}{\sum_{j=1}^{k} N_{vj}}$
15 $\hat{P}(c_i|\mathbf{D}_N) \leftarrow \frac{n_i - N_{vi}}{\sum_{j=1}^{k} n_j - N_{vj}}$
16 $score(X \leq v) \leftarrow Gain(\mathbf{D}, \mathbf{D}_Y, \mathbf{D}_N)$ // use Eq. (19.5)
17 **if** $score(X \leq v) > score^*$ **then**
18 $v^* \leftarrow v$; $score^* \leftarrow score(X \leq v)$
19 **return** $(v^*, score^*)$

Категориальные Атрибуты

Если X является категориальным атрибутом, то мы оцениваем точки расщепления вида X ∈ V, где V ⊂ dom (X) и V ∅. Другими словами, рассматриваются все различные разбиения множества значений X. Поскольку точка разделения X ∈ V дает то же разбиение, что и X ∈ V, где V = dom (X) \ V является дополнением к V, общее число различных разбиений задается как

$$\sum_{i=1}^{[m/2]} \binom{m}{i} = O(2^{m-1}) \tag{13.17}$$

где m-число значений в области X, то есть m = / dom (X)|. Таким образом, число возможных точек расщепления для рассмотрения экспоненциально в m, что может создавать

проблемы, если m велико. Одно из упрощений состоит в том, чтобы ограничить V размером один, так что существует только m расщепленных точек вида Xj ∈ {v}, где v ∈ dom(Xj).

Чтобы оценить данный раскол точке x ∈ V, мы должны вычислить следующую вероятность массового функций класса:

$$P(c_i|\boldsymbol{D}_Y) = P(c_i|X \in V)$$

$$P(c_i|\boldsymbol{D}_N) = P(c_i|X \notin V)$$

Используя теорему Байеса, мы имеем

$$P(c_i|X \in V) = \frac{P(X \in V|c_i)P(c_i)}{P(X \in V)} = \frac{P(X \in V|c_i)P(c_i)}{\sum_{j=1}^{k} P(X \in V|c_i)P(c_i)}$$

Однако заметим, что данная точка x может принимать только одно значение в области X, и поэтому значения v ∈ dom(X) являются взаимоисключающими. Поэтому мы имеем

$$P(X \in V|c_i) = \sum_{v \in V} P(X = v|c_i)$$

и мы можем переписать $P(c_i|\boldsymbol{D}_Y)$ как

$$P(c_i|\boldsymbol{D}_Y) = \frac{\sum_{v \in V} P(X = v|c_i)P(c_i)}{\sum_{j=1}^{k} \sum_{v \in V} P(X = v|c_j)P(c_i)} \tag{13.18}$$

Определим n_{vi} как число точек $\mathbf{x}_j \in \mathbf{D}$ со значением $x_j = v$ для атрибута X и классом $y_j = c_i$:

$$n_{vi} = \sum_{j=1}^{n} I(x_j = v \text{ and } y_j = c_i) \tag{13.19}$$

Класс условных эмпирических ВМФ для X тогда задается как

$$\hat{P}(X = v|c_i) = \frac{\hat{P}(X = v \text{ and } c_i)}{\hat{P}(c_i)} = \left(\frac{1}{n}\sum_{j=1}^{n} I\big(x_j = v \text{ and } y_i = c_i\big)\right) / \,(c_i/\,n) = n_{vi}/n_i \tag{13.20}$$

Заметим, что априорные вероятности класса могут быть оценены с помощью уравнения (13.11), как обсуждалось для разбиения $\mathbf{D}_Y$ для точки разбиения X ∈ V задается как

$$\hat{P}(c_i|\boldsymbol{D}_Y) = \frac{\sum_{v \in V} \hat{P}(X = v|c_i)\hat{P}(c_i)}{\sum_{j=1}^{k} \sum_{v \in V} \hat{P}(X = v|c_j)\hat{P}(c_j)} = \frac{\sum_{v \in V} n_{vi}}{\sum_{j=1}^{k} \sum_{v \in V} n_{uj}} \tag{13.21}$$

Аналогичным образом класс ВМФ для раздела $\mathbf{D}_N$ задается как

$$\hat{P}(c_i|\boldsymbol{D}_N) = \hat{P}(c_i|X \notin V) = \frac{\sum_{v \notin V} n_{vi}}{\sum_{j=1}^{k} \sum_{v \notin V} n_{vj}} \tag{13.22}$$

Алгоритм 13.3 показывает метод оценки точки разделения для категориальных атрибутов. Цикл for в строке 4 перебирает все точки и вычисляет n_{vi}, то есть число точек, имеющих значение $v \in dom(X)$ и класс c_i. Цикл for в строке 7 перечисляет все возможные точки разбиения вида $X \in V$ для $V \subset dom(X)$, такие что $|V| \leq l$, где l - заданный пользователем параметр, обозначающий максимальную мощность V. Например, чтобы контролировать количество точек разбиения, мы также можем ограничить V одним элементом, то есть $l = 1$, так что разбиения имеют вид $V \in \{v\}$, с $v \in dom(X)$. Если $l = \lfloor m/2 \rfloor$, то мы должны рассмотреть все возможные различные разбиения V. учитывая точку разбиения $X \in V$, метод оценивает ее с помощью информационного усиления [Ур. (13.5)], хотя также можно использовать любой другой критерий оценки. Лучшая точка разделения и оценка записываются и возвращаются.

С точки зрения вычислительной сложности специфичные для класса подсчеты для каждого значения n_{vi} занимают O (n) времени (для цикла в строке 4). При $m = |dom(X)|$ максимальное число разбиений V равно $O(2^{m-1})$, и поскольку каждая точка разбиения может быть оценена за время O(mk), цикл for В строке 7 занимает время $O(mk2^{m-1})$. Таким образом, общая стоимость категориальных атрибутов равна $O(n + mk2^{m-1})$. Если мы предположим, что $2^{m-1} = O(n)$, то есть если мы связываем максимальный размер V с $l = O(\log n)$, то стоимость категориальных расщеплений ограничена как O (n log n), игнорируя k.

EVALUATE-CATEGORICAL-ATTRIBUTE $(\mathbf{D}, X, l)$:

1 **for** $i = 1, \ldots, k$ **do**
2 $n_i \leftarrow 0$
3 **forall** $v \in dom(X)$ **do** $n_{vi} \leftarrow 0$
4 **for** $j = 1, \ldots, n$ **do**
5 **if** $x_j = v$ *and* $y_j = c_i$ **then** $n_{vi} \leftarrow n_{vi} + 1$ `// frequency statistics`
`// evaluate split points of the form` $X \in V$
6 $V^* \leftarrow \emptyset$; $score^* \leftarrow 0$ `// initialize best split point`
7 **forall** $V \subset dom(X)$, *such that* $1 \leq |V| \leq l$ **do**
8 **for** $i = 1, \ldots, k$ **do**
9 $\hat{P}(c_i|\mathbf{D}_Y) \leftarrow \frac{\sum_{v \in V} n_{vi}}{\sum_{j=1}^{k} \sum_{v \in V} n_{vj}}$
10 $\hat{P}(c_i|\mathbf{D}_N) \leftarrow \frac{\sum_{v \notin V} n_{vi}}{\sum_{j=1}^{k} \sum_{v \notin V} n_{vj}}$
11 $score(X \in V) \leftarrow Gain(\mathbf{D}, \mathbf{D}_Y, \mathbf{D}_N)$ `// use Eq. (19.5)`
12 **if** $score(X \in V) > score^*$ **then**
13 $V^* \leftarrow V$; $score^* \leftarrow score(X \in V)$
14 **return** $(V^*, score^*)$

13.2.3 Вычислительная Сложность

Для анализа вычислительной сложности метода дерева решений в алгоритме 13.1 мы предполагаем, что стоимость оценки всех точек разбиения для числового или категориального

атрибута - O ($n \log n$), где $n = |\mathbf{D}|$ - размер набора данных. Учитывая **D**, алгоритм дерева решений оценивает все атрибуты d со стоимостью (d n log n). Общая стоимость зависит от глубины дерева решений. В худшем случае дерево может иметь глубину n, и таким образом общая стоимость равна O($d\ n^2 \log n$).

Глава 14. Линейный дискриминантный анализ

Учитывая помеченные данные, состоящие из d-мерных точек x_i вместе с их классами y_i , цель линейного дискриминантного анализа (LDA) состоит в том, чтобы найти вектор w, который максимизирует разделение между классами после проекции на w. Вспомним из Главы 7, что первым главным составляющим является вектор, который максимизирует прогнозируемую дисперсию точек. Ключевое различие между анализом главных компонент и LDA заключается в том, что первый имеет дело с неразмеченными данными и пытается максимизировать дисперсию, тогда как второй имеет дело с помеченными данными и пытается максимизировать различие между классами.

14.1 Оптимальный линейный дискриминант

Предположим, что набор данных **D** состоит из n помеченных точек $\{x_i, y_i\}$, где $\mathbf{x}_i \in R^d$ и $y_i \in \{c_1, c_2..., c_k\}$. Пусть $\mathbf{D}_i$ обозначает подмножество точек, помеченных классом c_i, т.е., $\mathbf{D}_i = \{\mathbf{x}_j \mid y_j = c_i\}$, и пусть $|\mathbf{D}_i| = n_i$ обозначает количество точек с классом ci. Мы предполагаем, что существует только k = 2 классов. Таким образом, набор данных **D** может быть разбит на $\mathbf{D}_1$ и $\mathbf{D}_2$.

Пусть **w** единичный вектор, то есть, $\mathbf{w}^T\mathbf{w} = 1$. Проекция любой d-мерной точки $\boldsymbol{x}_i$ на вектор $\boldsymbol{w}$ задаётся как

$$\boldsymbol{x}_i' = \left(\frac{\boldsymbol{w}^T\boldsymbol{x}_i}{\boldsymbol{w}^T\boldsymbol{w}}\right)\boldsymbol{w} = (\boldsymbol{w}^T\boldsymbol{x}_i)\boldsymbol{w} = a_i\boldsymbol{w}$$

где a_i задает смещение или координату $\boldsymbol{x}_i'$ вдоль линии $\boldsymbol{w}$:

$$a_i = \boldsymbol{w}^T\boldsymbol{x}_i$$

Таким образом, множество n скаляров $\{a_1, a_2..., a_n\}$ представляет собой отображение из R^d в R, то есть из исходного d-мерного пространства в 1-мерное пространство (вдоль **w**).

Пример 14.1. Рассмотрим рис. 14.1, на котором показан 2-мерный набор данных Iris с длиной и шириной чашелистика в качестве атрибутов, а iris-setosa как класс c_1 (круги), а два других типа Iris как класс c_2 (треугольники). Есть $n_1 = 50$ точек в c_1 и $n_2 = 100$ точек в c_2. Один возможный вектор **w** показан вместе с проекцией

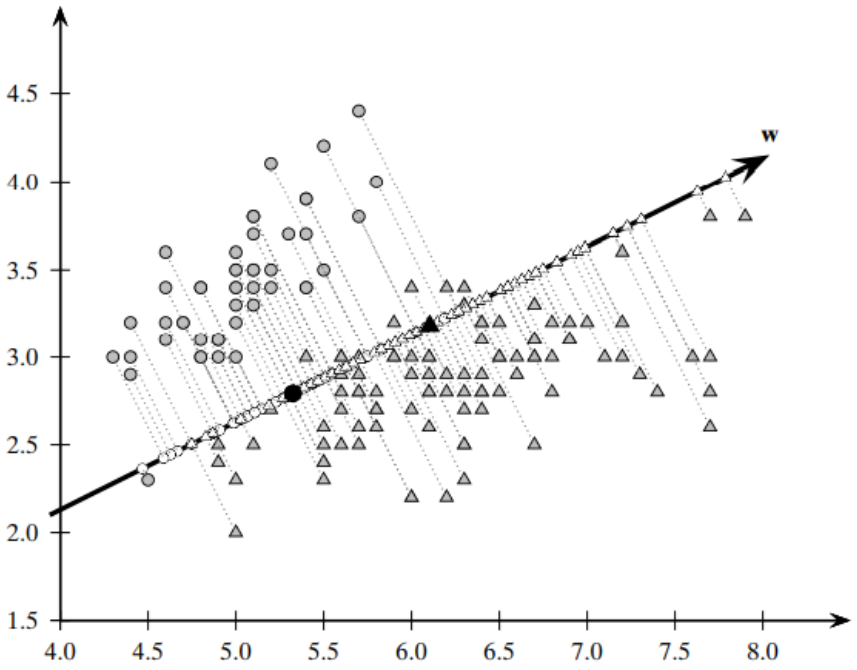


Рисунок 14.1. Проекция на ***w***.

из всех точек на **w**. Прогнозируемые средние значения этих двух классов показаны чёрным цветом. Здесь **w** было переведено так, что оно проходит через среднее значение всех данных. Можно заметить, что **w** не очень хорошо различает эти два класса, потому что проекции точек на **w** являются смешанными в терминах их классовых меток. Оптимальное направление линейного дискриминанта показано на рис. 14.2.

Каждая координата точки a_i имеет связанную с ней метку исходного класса y_i, и таким образом мы можем вычислить для каждого из двух классов среднее значение проецируемых точек следующим образом:

$$m_1 = \frac{1}{n_1}\sum_{x_i \in D_1} a_i = \frac{1}{n_1}\sum_{x_i \in D_1} w^T x_i = w^T\left(\frac{1}{n_1}\sum_{x_i \in D_1} x_i\right) = w^T \mu_1$$

где μ_1 среднее значение всех точек в $\mathbf{D}_1$. Аналогично, мы можем получить

$$m_2 = w^T \mu_2$$

Другими словами, среднее значение проецируемых точек совпадает с проекцией среднего.

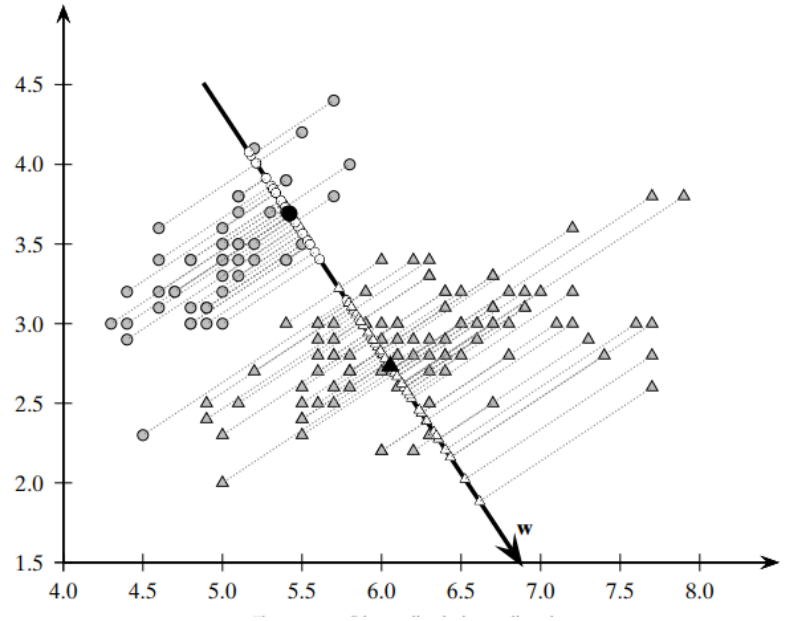


Рисунок 14.2. Линейное дискриминантное направление **w**.

Чтобы максимизировать разделение между классами, представляется разумным максимизировать разницу между прогнозируемыми средними, $|m_1 - m_2|$. Однако этого недостаточно. Для хорошего разделения дисперсия прогнозируемых точек для каждого класса также не должна быть слишком большой. Большая дисперсия привела бы к возможному совпадению точек двух классов из-за большого разброса точек, и поэтому мы можем не иметь хорошего разделения. LDA максимизирует разделение, гарантируя, что разброс s_i^2 для проецируемых точек внутри каждого класса является маленьким, где разброс определяется как

$$s_i^2 = \sum_{x_j \in D_i} (a_j - m_i)^2$$

Разброс – это общее квадратичное отклонение от среднего арифметического, в отличие от дисперсии, которая является средним отклонением от среднего арифметического. Другими словами

$$s_i^2 = n_i \sigma_i^2$$

где $n_i = |\mathbf{D}_i|$ - размер, а σ_i^2 дисперсия для класса c_i.

Мы можем включить два критерия линейного дискриминантного анализа, а именно максимизацию расстояния между прогнозируемыми средними и минимизацию суммы прогнозируемого разброса, в единый критерий максимизации, называемый задачей линейного дискриминантного анализа Фишера:

$$\max_w J(w) = \frac{(m_1 - m_2)^2}{s_1^2 + s_2^2} \quad (14.1)$$

Цель линейного дискриминантного анализа состоит в том, чтобы найти вектор **w,** который максимизирует $J(\mathbf{w})$, то есть направление, которое максимизирует разделение между

двумя средними m_1 и m_2, и минимизирует общий разброс $s_1^2 + s_2^2$ двух классов. Вектор **w** также называется оптимальным линейным дискриминантом. Задача оптимизации [Ур. (14.1)] находится в проекционном пространстве. Чтобы решить ее, мы должны переписать ее в терминах входных данных, как описано далее.

Заметим, что мы можем переписать $(m_1 - m_2)^2$следующим образом:

$$(m_1 - m_2)^2 = \left(w^T(\mu_1 - \mu_2)\right)^2 = w^T\left((\mu_1 - \mu_2)(\mu_1 - \mu_2)^T\right)w = w^T B w \quad (14.2)$$

где $B = (\mu_1 - \mu_2)(\mu_1 - \mu_2)^T$ матрица ранга d ×d, называемая матрицей рассеяния между классами.

Что касается прогнозируемого разброса для класса c_1, то мы можем вычислить его следующим образом:

$$s_i^2 = \sum_{x_i \in D_1} \left(a_j - m_i\right)^2 = \sum_{x_i \in D_1} (w^T x_i - w^T \mu_1)^2 = \sum_{x_i \in D_1} (w^T (x_i - \mu_1))^2$$

$$= w^T \left(\sum_{x_i \in D_1} (x_i - \mu_1)(x_i - \mu_1)^T \right) w = w^T S_1 w \quad (14.3)$$

где $\mathbf{S}_1$ матрица рассеяния для $\mathbf{D}_1$. Аналогично, мы можем получить

$$s_2^2 = w^T S_2 w \quad (14.4)$$

Обратите внимание еще раз, что матрица рассеяния, по существу, такая же, как и ковариационная матрица, но вместо того, чтобы записывать среднее отклонение от среднего арифметического, она записывает общее отклонение, то есть

$$S_i = n_i \Sigma_i \quad (14.5)$$

Совмещая уравнения (14.3) и (14.4), знаменатель в уравнении (14.1) может быть переписан как:

$$s_1^2 + s_2^2 = w^T S_1 w + w^T S_2 w = w^T (S_1 + S_2) w = w^T S w \quad (14.6)$$

где $\mathbf{S} = \mathbf{S}_1 + \mathbf{S}_2$ обозначает матрицу рассеяния внутри класса для объединённых данных. Так как S_1 и $\mathbf{S}_2$ симметричны положительно определённым d × d матрицам, **S** имеет те же свойства.

Используя уравнения (14.2) и (14.6), запишем целевую функцию линейного дискриминантного анализа [ур. (14.1)] следующим образом

$$\max_W J(w) = \frac{w^T B w}{w^T S w} \quad (14.7)$$

Чтобы решить задачу для наилучшего направления **w**, мы дифференцируем целевую функцию относительно **w** и устанавливаем результат равным нулю. Нам явно не приходится иметь дело с ограничением, что $w^T w = 1$, потому что в ур. (14.7) члены, связанные с величиной **w**, отменяются в числителе и знаменателе.

Напомним, что если f (x) и g(x) – две функции, то мы имеем

$$\frac{d}{dx}\left(\frac{f(x)}{g(x)}\right) = \frac{f'(x)g(x) - g'(x)f(x)}{g(x)^2}$$

где $f'(x)$ обозначает производную от $f(x)$. Принимая производную от ур. (14.7) относительно вектора w и установка результата на нулевой вектор дает нам

$$\frac{d}{dw}J(w) = \frac{2Bw(w^TSw) - 2Sw(w^TBw)}{(w^TSw)^2} = 0$$

что даёт

$$Bw(w^TSw) = Sw(w^TBw)$$

$$Bw = Sw\left(\frac{w^TBw}{w^TSw}\right)$$

$$Bw = J(w)Sw$$

$$Bw = \lambda Sw \quad (14.8)$$

где $\lambda = J(\boldsymbol{w})$. Ур. (14.8) представляет собой обобщённую задачу на собственные значения где λ обобщенного собственного значения **B** и **S**; собственному значению λ удовлетворяет уравнение det(**B** − λ**S**) = 0. Потому что цель к максимизации задачи [Ур. (14.7)], J(**w**) = λ должна быть выбрана по величине обобщенного собственного значения и w для соответствующего собственного вектора. Если S невырождено, то есть если S^{-1} существует, то ур. (14.8) приводит к стандартному уравнению собственных значений – собственных векторов, как

$$Bw = \lambda Sw$$

$$S^{-1}Bw = \lambda S^{-1}Sw$$

$$(S^{-1}B)w = \lambda w \quad (14.9)$$

Таким образом, если S^{-1} существует, то $\lambda = J(\boldsymbol{w})$ является собственным значением, а w собственный вектор матрицы S^{-1}B. Чтобы максимизировать $J(w)$, мы ищем наибольшее собственное значение λ, а соответствующий доминирующий собственный вектор **w** задает наилучший линейный дискриминантный вектор.

Алгоритм 14.1 показывает псевдокод для линейного дискриминантного анализа. Здесь мы предполагаем, что существует два класса и что S невырожденная (т.е., S^{-1} существует). Вектор 1_{n_i} - это вектор всех единиц с соответствующей размерностью для каждого класса, то есть $1_{n_i} \in R^{n_i}$ для класса i = 1,2. После деления D на две группы D_1 и D_2, линейный дискриминантный анализ переход к вычислению матриц рассеяния между классами и внутри классов B и S. Оптимальный вектор линейного дискриминанта получается как доминирующий собственный вектор S^{-1}B. С точки зрения вычислительной сложности вычисление S занимает O(nd^2) времени, а вычисление доминирующей пары собственное значение -собственный вектор занимает $O(d^3)$ времени в худшем случае. Таким образом, общее время равно $O(d^3 + nd^2)$.

Алгоритм 14.1. Линейный дискриминантный анализ

LINEARDISCRIMINANT $(\mathbf{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n})$:

1 $\mathbf{D}_i \leftarrow \{\mathbf{x}_j \mid y_j = c_i, j = 1, \ldots, n\}, i = 1, 2$ // class-specific subsets
2 $\boldsymbol{\mu}_i \leftarrow \text{mean}(\mathbf{D}_i), i = 1, 2$ // class means
3 $\mathbf{B} \leftarrow (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T$ // between-class scatter matrix
4 $\mathbf{Z}_i \leftarrow \mathbf{D}_i - \mathbf{1}_{n_i} \boldsymbol{\mu}_i^T, i = 1, 2$ // center class matrices
5 $\mathbf{S}_i \leftarrow \mathbf{Z}_i^T \mathbf{Z}_i, i = 1, 2$ // class scatter matrices
6 $\mathbf{S} \leftarrow \mathbf{S}_1 + \mathbf{S}_2$ // within-class scatter matrix
7 $\lambda_1, \mathbf{w} \leftarrow \text{eigen}(\mathbf{S}^{-1}\mathbf{B})$ // compute dominant eigenvector

Пример 14.2 (Линейный дискриминантный анализ). Рассмотрим 2-х мерные данные Iris (с атрибутами длина чашелистика и ширина чашелистика) показанные в примере 14.1. Класс c_1, соответствующий iris-setosa, имеет n_1 = 50 баллов, тогда как другой класс c_2 имеет n_2 = 100 points. Приведены средние значения для двух классов c_1 и c_2, и их разность даны в виде

$$\boldsymbol{\mu}_1 = \begin{pmatrix} 5.01 \\ 3.42 \end{pmatrix} \boldsymbol{\mu}_2 = \begin{pmatrix} 6.26 \\ 2.87 \end{pmatrix} \boldsymbol{\mu}_1 - \boldsymbol{\mu}_2 = \begin{pmatrix} -1.256 \\ 0.546 \end{pmatrix}$$

Матрица рассеяния между классами

$$\mathbf{B} = (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T = \begin{pmatrix} -1.256 \\ 0.546 \end{pmatrix} (-1.256 \quad 0.546) = \begin{pmatrix} 1.587 & -0.693 \\ -0.693 & 0.303 \end{pmatrix}$$

матрица рассеяния внутри класса

$$\mathbf{S}_1 = \begin{pmatrix} 6.09 & 4.91 \\ 4.91 & 7.11 \end{pmatrix} \mathbf{S}_2 = \begin{pmatrix} 43.5 & 12.09 \\ 12.09 & 10.96 \end{pmatrix} \quad \mathbf{S} = \mathbf{S}_1 + \mathbf{S}_2 = \begin{pmatrix} 49.58 & 17.01 \\ 17.01 & 18.08 \end{pmatrix}$$

S невырожденное, а его обратная величина задается как

$$\mathbf{S}^{-1} = \begin{pmatrix} 0.0298 & -0.028 \\ -0.028 & 0.0817 \end{pmatrix}$$

Поэтому мы имеем

$$\mathbf{S}^{-1}\mathbf{B} = \begin{pmatrix} 0.0298 & -0.028 \\ -0.028 & 0.0817 \end{pmatrix} \begin{pmatrix} 1.587 & -0.693 \\ -0.693 & 0.303 \end{pmatrix} = \begin{pmatrix} 0.066 & -0.029 \\ -0.100 & 0.044 \end{pmatrix}$$

Направление наибольшего разделения между c_1 и c_2 является доминирующим собственным вектором, соответствующим наибольшему собственному значению матрицы S^{-1}B. Решение таково:

$$J(\mathbf{w}) = \lambda_1 = 0.11$$
$$\mathbf{w} = \begin{pmatrix} 0.551 \\ -0.834 \end{pmatrix}$$

На рис. 14.2 показано оптимальное линейное дискриминантное направление **w**, переведенное в среднее значение данных. Прогнозируемые средние значения для этих двух классов показаны черным цветом. Мы можем чётко заметить, что вдоль **w** круги появляются вместе как группа и довольно хорошо отделены от треугольников. За исключением одной

внешней окружности, соответствующей точке $(4.5,2.3)^T$, все точки в c_1 идеально отделены от точек в c_2.

Для сценария двух классов, если S невырожденное, мы можем непосредственно решить $\mathbf{w}$ без вычисления собственных значений и собственных векторов. Заметим, что $\mathbf{B} = (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T$ является d × d матрицей ранга один, и поэтому Bw должен указывать в том же направлении, что $(\mu_1 - \mu_2)$, потому что

$$Bw = \big((\mu_1 - \mu_2)(\mu_1 - \mu_2)^T\big)w = (\mu_1 - \mu_2)((\mu_1 - \mu_2)^T w) = b(\mu_1 - \mu_2)$$

где $b = (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T\mathbf{w}$ просто скалярный множитель.

Затем мы можем переписать ур. (14.9) следующим образом

$$Bw = \lambda Sw$$

$$b(\mu_1 - \mu_2) = \lambda Sw$$

$$w = \frac{b}{\lambda}S^{-1}(\mu_1 - \mu_2)$$

Потому что $\frac{b}{\lambda}$ это просто скаляр, который мы можем решить для лучшего линейного дискриминанта как

$$\mathbf{w} = \mathbf{S}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) \quad (14.10)$$

Как только направление $\mathbf{w}$ найдено, мы можем нормализовать его как единичный вектор. Таким образом, вместо решения для собственного значения/собственного вектора в случае двух классов мы сразу получаем направление w, используя ур. (14.10). Интуитивно направление, которое максимизирует разделение между классами, можно рассматривать как линейное преобразование, (по $\mathbf{S}^{-1}$) вектора, соединяющего два средних класса $(\mu_1 - \mu_2)$.

Пример 14.3. Продолжая пример 14.2, мы можем непосредственно вычислить w следующим образом:

$$\begin{aligned}\mathbf{w} &= \mathbf{S}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) \\ &= \begin{pmatrix} 0.066 & -0.029 \\ -0.100 & 0.044 \end{pmatrix}\begin{pmatrix} -1.246 \\ 0.546 \end{pmatrix} = \begin{pmatrix} -0.0527 \\ 0.0798 \end{pmatrix}\end{aligned}$$

После нормализации получится:

$$\mathbf{w} = \frac{\mathbf{w}}{\|\,\mathbf{w}\,\|} = \frac{1}{0.0956}\begin{pmatrix} -0.0527 \\ 0.0798 \end{pmatrix} = \begin{pmatrix} -0.551 \\ 0.834 \end{pmatrix}$$

Обратите внимание, что даже если знак для $\mathbf{w}$ изменился на противоположный по сравнению с тем, что был в примере 14.2, они представляют одно и то же направление; отличается только скалярный множитель.

14.2 Дискриминантный анализ ядра

Дискриминантный анализ ядра, как и линейный дискриминантный анализ, пытается найти направление, которое максимизирует разделение между классами. Однако он делает это в пространстве функций с помощью функций ядра.

Задан набор данных , $\mathbf{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$где x_i точка во входном пространстве, а $y_i \in \{c_1, c_2\}$ является метка класса, пусть $\mathbf{D}_i = \{\mathbf{x}_j \mid y_j = c_i\}$ обозначим подмножество данных ограниченному классу c_i, и пусть $n_i = |\mathbf{D}_i|$. Далее, пусть φ($\mathbf{x}_i$) обозначает соответствующую точку в пространстве признаков и пусть K - функция ядра.

Цель ядра линейного дискриминантного анализа в том, чтобы найти вектор направления **w** в пространстве признаков, который максимизирует

$$\max_{\mathbf{w}} J(\mathbf{w}) = \frac{(m_1 - m_2)^2}{s_1^2 + s_2^2} \quad (14.11)$$

где m_1 и m_2 прогнозируемые средние арифметические значения, а s_1^2 и s_2^2 прогнозируемые значения рассеяния в пространстве функций. Сначала мы покажем, что **w** можно выразить как линейную комбинацию точек в пространстве признаков, а затем преобразуем задачу линейного дискриминантного анализа в терминах матрицы ядра.

Оптимальный линейный дискриминант: линейная комбинация характерных точек

Среднее арифметическое значение для класса c_i в пространстве признаков задается как

$$\boldsymbol{\mu}_i^{\phi} = \frac{1}{n_i} \sum_{\mathbf{x}_j \in \mathbf{D}_i} \phi(\mathbf{x}_j) \quad (14.12)$$

а ковариационная матрица для класса c_i в пространстве признаков

$$\boldsymbol{\Sigma}_i^{\phi} = \frac{1}{n_i} \sum_{\mathbf{x}_j \in \mathbf{D}_i} \left(\phi(\mathbf{x}_j) - \boldsymbol{\mu}_i^{\phi}\right)\left(\phi(\mathbf{x}_j) - \boldsymbol{\mu}_i^{\phi}\right)^T$$

Используя вывод, аналогичный уравнению (14.2) мы получаем выражение для матрицы рассеяния между классами в пространстве признаков

$$\mathbf{B}_{\phi} = \left(\boldsymbol{\mu}_1^{\phi} - \boldsymbol{\mu}_2^{\phi}\right)\left(\boldsymbol{\mu}_1^{\phi} - \boldsymbol{\mu}_2^{\phi}\right)^T = \mathbf{d}_{\phi}\mathbf{d}_{\phi}^T \quad (14.13)$$

где $\mathbf{d}_{\phi} = \boldsymbol{\mu}_1^{\phi} - \boldsymbol{\mu}_2^{\phi}$ разность между средними векторами двух классов. Аналогично, используя уравнения (14.5) и (14.6) матрица рассеяния внутри класса в пространстве признаков задается следующим образом

$$\mathbf{S}_{\phi} = n_1 \boldsymbol{\Sigma}_1^{\phi} + n_2 \boldsymbol{\Sigma}_2^{\phi}$$

S_{ϕ} -симметричная положительная полуопределенная матрица, размером d × d, где d-размерность пространства признаков. Из уравнения (14.9) мы заключаем, что наилучшим линейным дискриминантным вектором **w** в пространстве признаков является доминирующий собственный вектор, удовлетворяющий выражению

$$\left(\mathbf{S}_{\phi}^{-1}\mathbf{B}_{\phi}\right)\mathbf{w} = \lambda\mathbf{w} \quad (14.14)$$

Где мы предполагаем, что S_ϕ не вырожден. Пусть δ_i обозначает i-е собственное значение, а u_i i-ый собственный вектор S_ϕ, при i = 1,...,d. Собственное разложение S_ϕ даёт S_ϕ = U$\mathbf{\Delta}U^T$, с обратным значением S_ϕ ,заданным как $\mathbf{S}_\phi^{-1} = \mathbf{U}\mathbf{\Delta}^{-1}\mathbf{U}^T$. Здесь U матрица, столбцы которой являются собственными векторами S_ϕ а $\mathbf{\Delta}$ диагональная матрица собственных значений S_ϕ. Таким образом, $\mathbf{S}_\phi^{-1}$ может быть выражена как спектральная сумма

$$\mathbf{S}_\phi^{-1} = \sum_{r=1}^{d} \frac{1}{\delta_r}\mathbf{u}_r\mathbf{u}_r^T$$

Включая уравнения (14.13) и (14.15) в уравнении (14.14) получаем:

$$\lambda\mathbf{w} = \left(\sum_{r=1}^{d} \frac{1}{\delta_r}\mathbf{u}_r\mathbf{u}_r^T\right)\mathbf{d}_\phi\mathbf{d}_\phi^T\mathbf{w} = \sum_{r=1}^{d} \frac{1}{\delta_r}\left(\mathbf{u}_r\left(\mathbf{u}_r^T\mathbf{d}_\phi\right)\left(\mathbf{d}_\phi^T\mathbf{w}\right)\right) = \sum_{r=1}^{d} b_r\mathbf{u}_r$$

где$b_r = \frac{1}{\delta_r}\left(\mathbf{u}_r^T\mathbf{d}_\phi\right)\left(\mathbf{d}_\phi^T\mathbf{w}\right)$ скалярное значение. Используя деривацию подобную той, что в уравнении (7.32), r-й собственный вектор S_ϕ может быть выражен как линейная комбинация характерных точек, скажем $\mathbf{u}_r = \sum_{j=1}^{n} c_{rj}\phi\left(\mathbf{x}_j\right)$, где

c_{rj} скалярный коэффициент. Таким образом мы можем переписать w

$$\begin{aligned}\mathbf{w} &= \frac{1}{\lambda}\sum_{r=1}^{d} b_r \left(\sum_{j=1}^{n} c_{rj}\phi(\mathbf{x}_j)\right) \\ &= \sum_{j=1}^{n} \phi(\mathbf{x}_j)\left(\sum_{r=1}^{d} \frac{b_r c_{rj}}{\lambda}\right) \\ &= \sum_{j=1}^{n} a_j \phi(\mathbf{x}_j)\end{aligned}$$

Где $a_j = \sum_{r=1}^{d} b_r c_{rj}/\lambda$ скалярное значение для характерной точки $\phi(\mathbf{x}_j)$. Поэтому вектор направления w может быть выражен как линейная комбинация точек в пространстве признаков.

Задача линейного дискриминантного анализа через матрицу ядра

Теперь мы перепишем задачу ядра линейного дискриминантного анализа [Ур. (14.11)] в терминах матрицы ядра. Проецируя среднее значение для класса c_i приведенное в уравнении (14.12) на направление линейного дискриминанта **w**, мы имеем

$$\begin{aligned}m_i = \mathbf{w}^T\boldsymbol{\mu}_i^{\phi} &= \left(\sum_{j=1}^{n} a_j\phi(\mathbf{x}_j)\right)^T \left(\frac{1}{n_i}\sum_{\mathbf{x}_k\in\mathbf{D}_i} \phi(\mathbf{x}_k)\right) \\ &= \frac{1}{n_i}\sum_{j=1}^{n}\sum_{\mathbf{x}_k\in\mathbf{D}_i} a_j\phi(\mathbf{x}_j)^T\phi(\mathbf{x}_k) \\ &= \frac{1}{n_i}\sum_{j=1}^{n}\sum_{\mathbf{x}_k\in\mathbf{D}_i} a_j K(\mathbf{x}_j,\mathbf{x}_k) \\ &= \mathbf{a}^T\mathbf{m}_i \qquad (14.16)\end{aligned}$$

Где $\mathbf{a} = (a_1, a_2, \ldots, a_n)^T$ весовой вектор, и

$$\mathbf{m}_i = \frac{1}{n_i}\begin{pmatrix} \sum_{\mathbf{x}_k\in\mathbf{D}_i} K(\mathbf{x}_1,\mathbf{x}_k) \\ \sum_{\mathbf{x}_k\in\mathbf{D}_i} K(\mathbf{x}_2,\mathbf{x}_k) \\ \vdots \\ \sum_{\mathbf{x}_k\in\mathbf{D}_i} K(\mathbf{x}_n,\mathbf{x}_k) \end{pmatrix} = \frac{1}{n_i}\mathbf{K}^{c_i}\mathbf{1}_{n_i} \qquad (14.17)$$

Где $\mathbf{K}^{c_i}$ является n × n_i подмножеством ядра матрицы, ограниченного столбцами, принадлежащими точкам только в $\mathbf{D}_i$, и $\mathbf{1}_{n_i}$ является n_i-мерным вектором вес записи

которого равны единице. Таким образом, вектор n-длины m_i сохраняет для каждой точки D свое среднее значение ядра по отношению к указанным точкам $\mathbf{D}_i$.

Мы можем переписать разделение между проецируемыми средними в пространстве признаков следующим образом:

$$\begin{aligned}(m_1 - m_2)^2 &= \left(\mathbf{w}^T\boldsymbol{\mu}_1^{\phi} - \mathbf{w}^T\boldsymbol{\mu}_2^{\phi}\right)^2 \\ &= (\mathbf{a}^T\mathbf{m}_1 - \mathbf{a}^T\mathbf{m}_2)^2 \\ &= \mathbf{a}^T(\mathbf{m}_1 - \mathbf{m}_2)(\mathbf{m}_1 - \mathbf{m}_2)^T\mathbf{a} \\ &= \mathbf{a}^T\mathbf{M}\mathbf{a} \qquad (14.18)\end{aligned}$$

где $\mathbf{M} = (\mathbf{m}_1 - \mathbf{m}_2)(\mathbf{m}_1 - \mathbf{m}_2)^T$ матрица рассеяния между классами.

Мы также можем вычислить прогнозируемый разброс для каждого класса, s_1^2 и s_2^2, исключительно в терминах функции ядра, как

$$s_1^2 = \sum_{\mathbf{x}_i \in \mathbf{D}_1} \left\|\mathbf{w}^T\phi(\mathbf{x}_i) - \mathbf{w}^T\boldsymbol{\mu}_1^{\phi}\right\|^2$$

$$= \sum_{\mathbf{x}_i \in \mathbf{D}_1} \left\|\mathbf{w}^T\phi(\mathbf{x}_i)\right\|^2 - 2\sum_{\mathbf{x}_i \in \mathbf{D}_1} \mathbf{w}^T\phi(\mathbf{x}_i)\cdot\mathbf{w}^T\boldsymbol{\mu}_1^{\phi} + \sum_{\mathbf{x}_i \in \mathbf{D}_1} \left\|\mathbf{w}^T\boldsymbol{\mu}_1^{\phi}\right\|^2$$

$$= \left(\sum_{\mathbf{x}_i \in \mathbf{D}_1} \left\|\sum_{j=1}^{n} a_j\phi(\mathbf{x}_j)^T\phi(\mathbf{x}_i)\right\|^2\right) - 2\cdot n_1\cdot\left\|\mathbf{w}^T\boldsymbol{\mu}_1^{\phi}\right\|^2 + n_1\cdot\left\|\mathbf{w}^T\boldsymbol{\mu}_1^{\phi}\right\|^2$$

$$= \left(\sum_{\mathbf{x}_i \in \mathbf{D}_1} \mathbf{a}^T\mathbf{K}_i\mathbf{K}_i^T\mathbf{a}\right) - n_1\cdot\mathbf{a}^T\mathbf{m}_1\mathbf{m}_1^T\mathbf{a}$$

используя ур.(14.16)

$$= \mathbf{a}^T\left(\left(\sum_{\mathbf{x}_i \in \mathbf{D}_1} \mathbf{K}_i\mathbf{K}_i^T\right) - n_1\mathbf{m}_1\mathbf{m}_1^T\right)\mathbf{a}$$

$$= \mathbf{a}^T\mathbf{N}_1\mathbf{a}$$

где $\mathbf{K}_i$ i-ый столбец ядра матрицы, а $\mathbf{N}_1$ матрица рассеяния класса для c_1. Пусть $K(x_i, x_j) = K_{ij}$. Мы можем выразить $\mathbf{N}_1$ более компактно в матричных обозначениях следующим образом:

$$\mathbf{N}_1 = \left(\sum_{\mathbf{x}_i \in \mathbf{D}_1} \mathbf{K}_i\mathbf{K}_i^T\right) - n_1\mathbf{m}_1\mathbf{m}_1^T$$

$$= (\mathbf{K}^{c_1})\left(\mathbf{I}_{n_1} - \frac{1}{n_1}\mathbf{1}_{n_1\times n_1}\right)(\mathbf{K}^{c_1})^T \qquad (14.19)$$

где $\mathbf{I}_{n_1}$ это тождественная матрица $n_1 \times n_1$, а $\mathbf{1}_{n_1\times n_1}$ матрица $n_1 \times n_1$, все записи которой равны 1.

Аналогично получаем $s_2^2 = \mathbf{a}^T\mathbf{N}_2\mathbf{a}$, где

$\mathbf{N}_2 = (\mathbf{K}^{c_2})\left(\mathbf{I}_{n_2} - \frac{1}{n_2}\mathbf{1}_{n_2 \times n_2}\right)(\mathbf{K}^{c_2})^T$где $\mathbf{I}_{n_2}$ это тождественная матрица $n_2 \times n_2$, а $\mathbf{1}_{n_2 \times n_2}$ матрица $n_2 \times n_2$, все записи которой равны 1.

Затем сумма прогнозируемых значений рассеяния задаётся в виде:

$$s_1^2 + s_2^2 = \mathbf{a}^T(\mathbf{N}_1 + \mathbf{N}_2)\mathbf{a} = \mathbf{a}^T\mathbf{N}\mathbf{a} \quad (14.20)$$

где N - матрица рассеяния n × n внутри класса

При замене уравнений (14.18) и (14.20) в уравнении (14.11) получим условие максимизации ядра линейного дискриминантного анализа

$$\max_{\mathbf{w}} J(\mathbf{w}) = \max_{\mathbf{a}} J(\mathbf{a}) = \frac{\mathbf{a}^T\mathbf{M}\mathbf{a}}{\mathbf{a}^T\mathbf{N}\mathbf{a}}$$

Обратите внимание, что все термины в приведенном выше выражении включают только фун
вектор, соответствующий наибольшему собственному значению обобщенной задачи на собственны

$$\mathbf{M}\mathbf{a} = \lambda_1\mathbf{N}\mathbf{a} \qquad (14.21)$$

Если N невырожденный, то a-доминирующий собственный вектор, соответствующий наибольшему собственному значению для системы

$$(\mathbf{N}^{-1}\mathbf{M})\mathbf{a} = \lambda_1\mathbf{a}$$

Как и в случае линейного дискриминантного анализа [ур. (14.10)], когда есть только два класса, нам не нужно решать для собственного вектора, потому что a может быть получено непосредственно:

$$\mathbf{a} = \mathbf{N}^{-1}(\mathbf{m}_1 - \mathbf{m}_2)$$

Как только **a** получено, мы можем нормализовать **w** как единичный вектор, гарантируя, что

$\mathbf{w}^T\mathbf{w} = 1$, из чего следует, что

$$\sum_{i=1}^{n}\sum_{j=1}^{n} a_i a_j \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j) = 1, \text{или}$$

$$\mathbf{a}^T\mathbf{K}\mathbf{a} = 1$$

Иными словами, мы можем гарантировать, что **w** является единичным вектором, если масштабируем **a** на $\frac{1}{\sqrt{\mathbf{a}^T\mathbf{K}\mathbf{a}}}$.

В конечном итоге, мы можем спроецировать любую точку x на дискриминантное направление следующим образом:

$$w^T\phi(x) = \sum_{j=1}^{n} a_j\phi(x_j)^T\phi(x) = \sum_{j=1}^{n} a_j K(x_j, x) \quad (14.22)$$

В алгоритме 14.2 приводится псевдокод для дискриминантного анализа ядра. Суть метода заключается в вычислении ядерной матрицы **K** размера $n \times n$ и ядерных матриц $\mathbf{K}^{ci}$

размером $n \times n_i$, для каждого класса c_i. После вычисления матриц межклассовой и внутриклассовой инвариантности **M** и **N**, вектор весов **a** получается как доминирующий собственный вектор $\mathbf{N}^{-1}\mathbf{M}$. Последний шаг масштабирует **a** так, чтобы **w** был нормализован до единичной длины. Сложность дискриминантного анализа ядра $O(n^3)$, при этом основными шагами являются вычисление **N** и нахождение доминирующего собственного вектора $\mathbf{N}^{-1}\mathbf{M}$, оба шага занимают время $O(n^3)$.

Алгоритм 14.2: Дискриминантный анализ ядра

KERNELDISCRIMINANT $(\mathbf{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n, K)$:

1 $\mathbf{K} \leftarrow \left\{K(\mathbf{x}_i, \mathbf{x}_j)\right\}_{i,j=1,\ldots,n}$ // compute $n \times n$ kernel matrix

2 $\mathbf{K}^{c_i} \leftarrow \left\{\mathbf{K}(j,k) \mid y_k = c_i, 1 \le j,k \le n\right\}, i = 1,2$ // class kernel matrix

3 $\mathbf{m}_i \leftarrow \frac{1}{n_i}\mathbf{K}^{c_i}\mathbf{1}_{n_i}, i = 1,2$ // class means

4 $\mathbf{M} \leftarrow (\mathbf{m}_1 - \mathbf{m}_2)(\mathbf{m}_1 - \mathbf{m}_2)^T$ // between-class scatter matrix

5 $\mathbf{N}_i \leftarrow \mathbf{K}^{c_i}(\mathbf{I}_{n_i} - \frac{1}{n_i}\mathbf{1}_{n_i \times n_i})(\mathbf{K}^{c_i})^T, i = 1,2$ // class scatter matrices

6 $\mathbf{N} \leftarrow \mathbf{N}_1 + \mathbf{N}_2$ // within-class scatter matrix

7 $\lambda_1, \mathbf{a} \leftarrow \text{eigen}(\mathbf{N}^{-1}\mathbf{M})$ // compute weight vector

8 $\mathbf{a} \leftarrow \frac{\mathbf{a}}{\sqrt{\mathbf{a}^T\mathbf{K}\mathbf{a}}}$ // normalize **w** to be unit vector

Пример 14.4 (Дискриминантный анализ ядра). Рассмотрим двухмерный набор данных Iris, включающий атрибуты длины и ширины наружной доли околоцветника. На рисунке 14.3а показаны точки, спроецированные на первые две главные компоненты. Точки были разделены на два класса: c_1 (кружки) соответствует разноцветным типам Iris, а c_2 (треугольники) соответствует двум другим типам Iris. Здесь $n_1 = 50$ и $n_2 = 100$, всего $n = 150$ точек.

Так как c_1 окружен точками в c_2, хорошего линейного дискриминанта не найти. Вместо этого мы применяем дискриминантный анализ ядра с использованием однородного квадратичного ядра

$$K(\mathbf{x}_i, \mathbf{x}_j) = \left(\mathbf{x}_i^T\mathbf{x}_j\right)^2$$

Вычислим **a** с помощью уравнения (14.21)

$$\lambda_1 = 0.0511$$

Однако мы не выводим вектор **a**, потому что он лежит в $\mathbb{R}^{150}$. На рисунке 14.3а показаны контуры постоянных проекций на лучший дискриминант ядра. Контуры получены путем решения уравнения (14.22), то есть решая $\mathbf{w}^T\phi(\mathbf{x}) == \sum_{j=1}^n a_j K(\mathbf{x}_j, \mathbf{x}) = c$ для различных значений скаляров c. Контуры гиперболические, поэтому они образуют пары, начинающиеся от центра. Например, первая кривая слева и справа от начала координат $(0,0)^T$ образует один и тот же контур, то есть точки вдоль обеих кривых имеют одинаковое значение при проецировании на **w**. Можно заметить, что все контуры или пары кривых, начинающиеся с четвертой кривой (слева и справа) от центра относятся к классу c_2, тогда как первые три контура относятся в основном к классу c_1, что указывает на хорошее разделение с однородным квадратичным ядром.

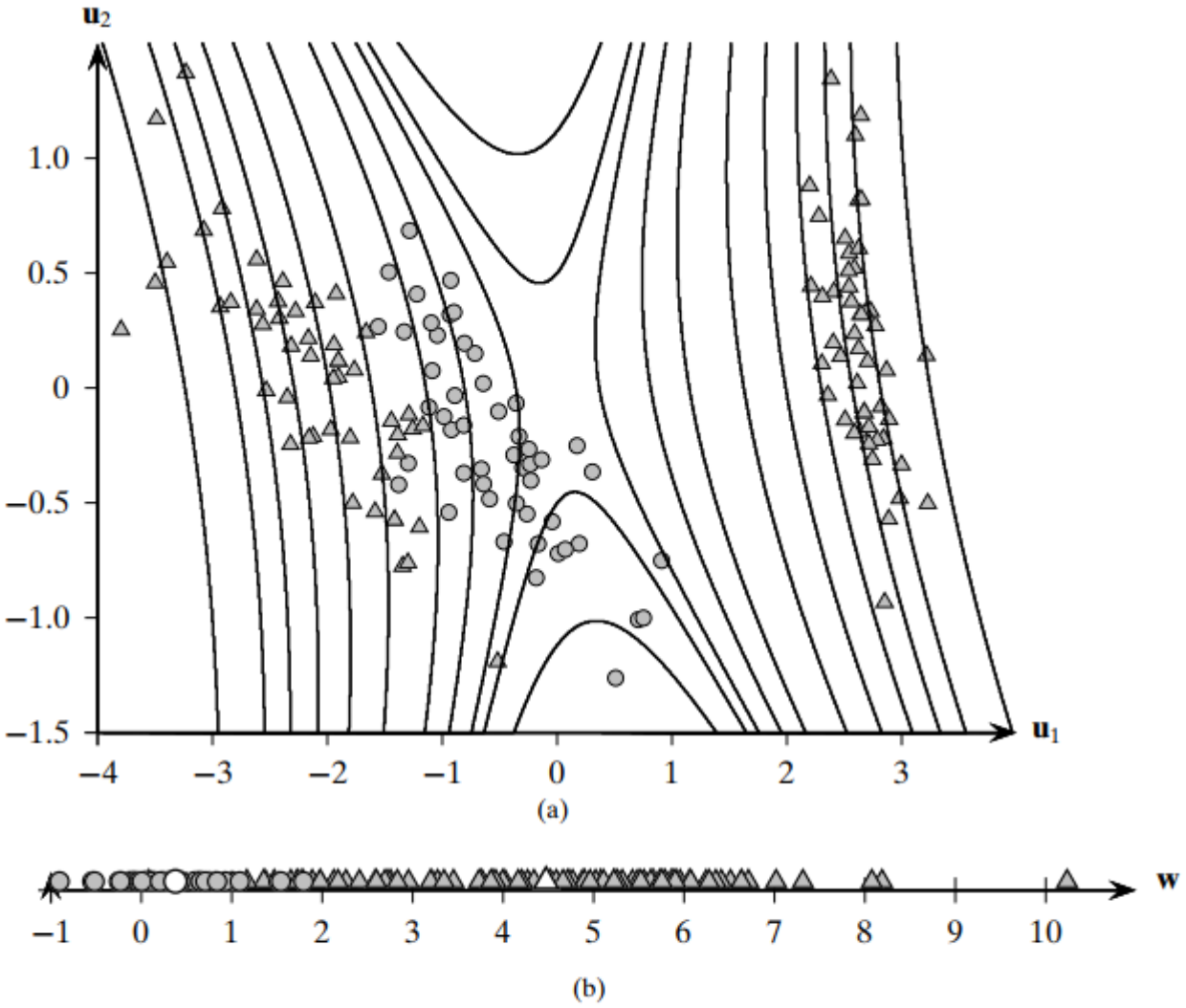


Рисунок 14.3. Дискриминантный анализ ядра: квадратичное однородное ядро

Лучшая картина вырисовывается, когда мы наносим на график координаты всех точек $\mathbf{x}_i \in \mathbf{D}$ при проецировании на $\mathbf{w}$, как показано на рисунке 14.3b. Можно заметить, что $\mathbf{w}$ может достаточно хорошо разделить два класса; все круги (c_1) сосредоточены слева, тогда как треугольники (c_2) расположены справа. Спроецированные средние показаны белым цветом. Рассеивание и средние для обоих классов после проецирования следующие:

$$m_1 = 0.338 \qquad m_2 = 4.476$$
$$s_1^2 = 13.862 \qquad s_2^2 = 320.934$$

Значение $J(\mathbf{w})$ вычисляется как

$$J(\mathbf{w}) = \frac{(m_1 - m_2)^2}{s_1^2 + s_2^2} = \frac{(0.338 - 4.476)^2}{13.862 + 320.934} = \frac{17.123}{334.796} = 0.0511$$

и ожидаемо соответствует $\lambda_1 = 0.0511$, приведенному выше.

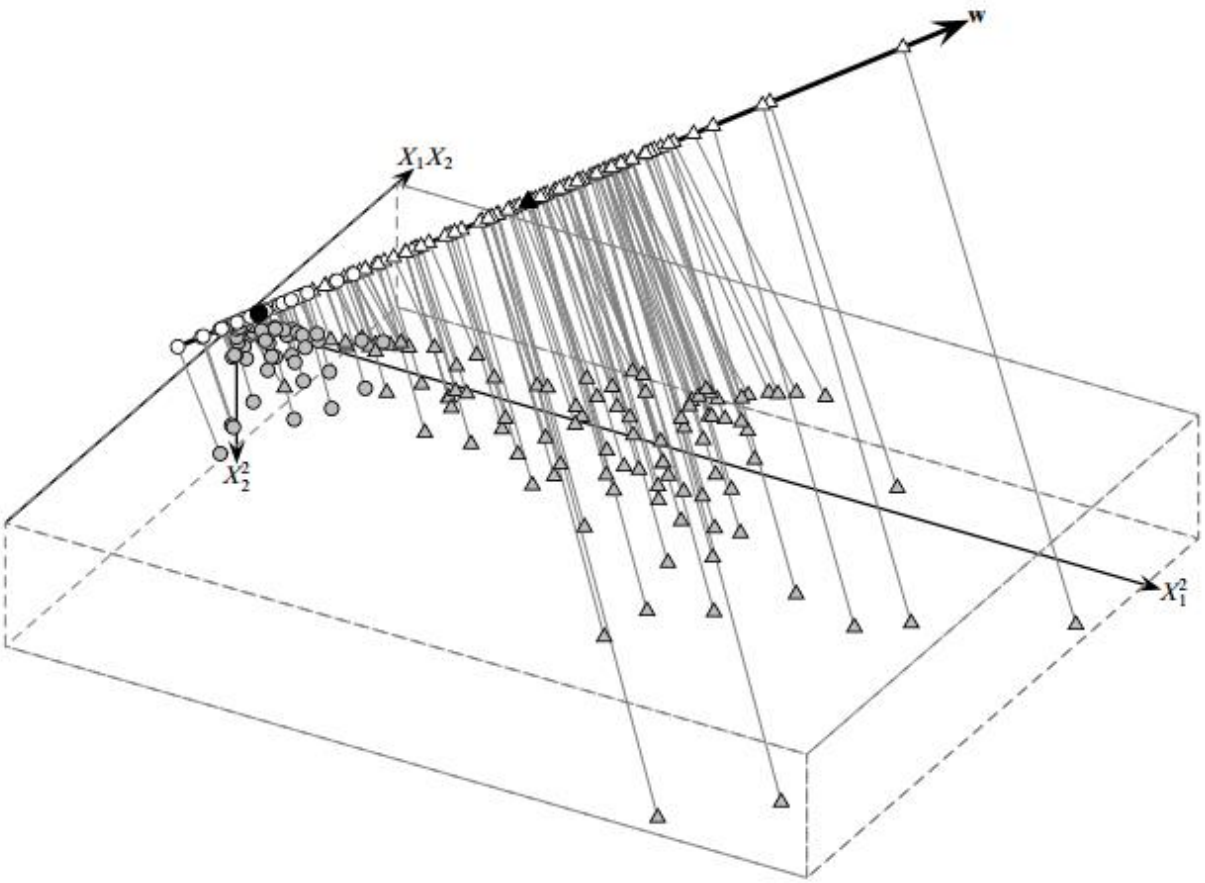


Рисунок 14.4. Пространство признаков однородного квадратичного ядра.

В общем, нежелательно или невозможно получить явный дискриминантный вектор **w**, поскольку он лежит в пространстве признаков. Однако, поскольку каждая точка $\mathbf{x} = (x_1, x_2)^T \in \mathbb{R}^2$ во входном пространстве отображается в точку $\phi(\mathbf{x}) = \left(\sqrt{2}x_1x_2, x_1^2, x_2^2\right)^T \in \mathbb{R}^3$ в пространстве признаков через однородное квадратичное ядро, для нашего примера возможно визуализировать пространство признаков, как показано на рисунке 14.4. Также показана проекция каждой точки $\phi(\mathbf{x}_i)$ на дискриминантный вектор **w**, где

$$\mathbf{w} = 0.511x_1x_2 + 0.761x_1^2 - 0.4x_2^2$$

Проекции на w идентичны показанным на рисунке 14.3b.

Глава 15. Метод опорных векторов

15.1 Опорные вектора и расстояния (зазоры)

Пусть $\mathbf{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ будет набором классифицируемых данных с n точками и размерностью d. Далее, предположим, что есть только две метки класса, то есть $y_i \in \{+1, -1\}$, обозначающих положительный и отрицательный классы.

Гиперплоскости

Гиперплоскость в d измерениях задается как множество всех точек $\mathbf{x} \in \mathbb{R}^d$, которые удовлетворяют уравнению $h(\mathbf{x}) = 0$, где $h(\mathbf{x})$ – функция гиперплоскости, определяемая следующим образом:

$$\begin{aligned} h(\mathbf{x}) &= \mathbf{w}^T\mathbf{x} + b \\ &= w_1x_1 + w_2x_2 + \cdots + w_dx_d + b \end{aligned} \qquad (15.1)$$

Здесь $\mathbf{w}$ – d-мерный вектор весов, а b – скаляр, называемый смещением. Для точек, лежащих на гиперплоскости, имеем

$$h(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b = 0 \qquad (15.2)$$

Таким образом, гиперплоскость определяется как множество всех таких точек, что $\mathbf{w}^T\mathbf{x} = -b$. Чтобы увидеть роль, которую играет b, предполагая, что $w_1 \neq 0$,и устанавливая $x_i = 0$ для всех $i > 1$, мы можем получить смещение, где гиперплоскость пересекает первую ось, так как по формуле. (15.2) имеем

$$w_1x_1 = -b \text{ или } x_1 = \frac{-b}{w_1}$$

Другими словами, точка $\left(\frac{-b}{w}, 0, \ldots, 0\right)$ лежит на гиперплоскости. Аналогичным образом мы можем получить смещение в месте пересечения гиперплоскости с каждой из осей, которое задается как $\frac{-b}{w_i}$ (при $w_i \neq 0$).

Разделяющая гиперплоскость

Гиперплоскость разбивает исходное d-мерное пространство на два полупространства. Набор данных называется линейно разделимым, если каждое полупространство имеет точки только из одного класса. Если входной набор данных является линейно разделимым, то мы можем найти разделяющую гиперплоскость $h(\mathbf{x}) = 0$, такую, что для всех точек с меткой $y_i = -1$ мы имеем $h(\mathbf{x}_i) < 0$, а для всех точек с меткой $y_i = +1$, мы имеем $h(\mathbf{x}_i) > 0$. Фактически, функция гиперплоскости $h(\mathbf{x})$ служит линейным классификатором или линейным дискриминантом, который предсказывает класс y для любой заданной точки $\mathbf{x}$ в соответствии с решающим правилом:

$$y = \begin{cases} +1 & \text{if } h(\mathbf{x}) > 0 \\ -1 & \text{if } h(\mathbf{x}) < 0 \end{cases} \qquad (15.3)$$

Пусть $\mathbf{a}_1$ и $\mathbf{a}_2$ – две произвольные точки, лежащие на гиперплоскости. Из уравнения. (15.2) имеем

$$\begin{aligned} h(\mathbf{a}_1) &= \mathbf{w}^T\mathbf{a}_1 + b = 0 \\ h(\mathbf{a}_2) &= \mathbf{w}^T\mathbf{a}_2 + b = 0 \end{aligned}$$

Вычитая одно из другого, получаем

$$\mathbf{w}^T(\mathbf{a}_1 - \mathbf{a}_2) = 0$$

Это означает, что вектор весов **w** ортогонален гиперплоскости, потому что он ортогонален любому произвольному вектору $(\mathbf{a}_1 - \mathbf{a}_2)$ на гиперплоскости. Другими словами, вектор весов **w** определяет направление, нормальное к гиперплоскости, которое фиксирует ориентацию гиперплоскости, тогда как смещение b фиксирует смещение гиперплоскости в d-мерном пространстве. Поскольку и **w**, и $-\mathbf{w}$ нормальны к гиперплоскости, мы устраняем эту двусмысленность, требуя, чтобы $h(\mathbf{x}_i) > 0$, когда $y_i = 1$, и $h(\mathbf{x}_i) < 0$, когда $y_i = -1$.

Расстояние от точки до гиперплоскости

Рассмотрим точку $\mathbf{x} \in \mathbb{R}^d$ такую, что **x** не лежит на гиперплоскости. Пусть $\mathbf{x}_p$ – ортогональная проекция **x** на гиперплоскость, и пусть $\mathbf{r} = \mathbf{x} - \mathbf{x}_p$, тогда, как показано на рисунке 15.1, мы можем записать **x** как

$$\mathbf{x} = \mathbf{x}_p + \mathbf{r}$$

$$\mathbf{x} = \mathbf{x}_p + r\frac{\mathbf{w}}{\|\mathbf{w}\|} \qquad (15.4)$$

где r – направленное расстояние точки **x** от $\mathbf{x}_p$, то есть r дает смещение **x** от $\mathbf{x}_p$ в терминах единичного вектора веса $\frac{\mathbf{w}}{\|\mathbf{w}\|}$. Смещение r положительно, если r находится в том же направлении что и **w**, и r отрицательно, если r находится в направлении, противоположном **w**.

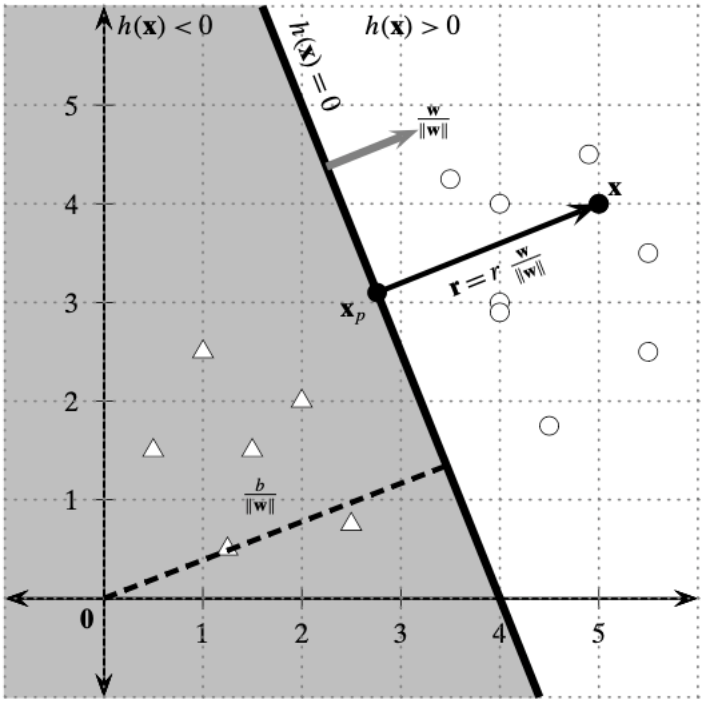


Рисунок 15.1 – Геометрия разделяющей гиперплоскости в 2D

Подставляя уравнение (15.4) в функцию гиперплоскости [Ур. (15.1)], получаем

$$
\begin{aligned}
h(\mathbf{x}) &= h\left(\mathbf{x}_p + r\frac{\mathbf{w}}{\|\mathbf{w}\|}\right) \\
&= \mathbf{w}^T\left(\mathbf{x}_p + r\frac{\mathbf{w}}{\|\mathbf{w}\|}\right) + b \\
&= \underbrace{\mathbf{w}^T\mathbf{x}_p + b}_{h(\mathbf{x}_p)} + r\frac{\mathbf{w}^T\mathbf{w}}{\|\mathbf{w}\|} \\
&= \underbrace{h(\mathbf{x}_p)}_{0} + r\|\mathbf{w}\| \\
&= r\|\mathbf{w}\|
\end{aligned}
$$

Последний шаг следует из того, что $h(\mathbf{x}_p) = 0$, поскольку $\mathbf{x}_p$ лежит на гиперплоскости. Используя результат выше, мы получаем выражение для направленного расстояния от точки до гиперплоскости:

$$r = \frac{h(\mathbf{x})}{\|\mathbf{w}\|}$$

Чтобы получить расстояние, которое должно быть неотрицательным, мы можем умножить r на метку класса y точки, потому что, когда $h(\mathbf{x}) < 0$, класс равен −1, а когда $h(\mathbf{x}) > 0$, класс равен +1. Таким образом, расстояние от точки $\mathbf{x}$ до гиперплоскости $h(\mathbf{x}) = 0$ задается как

$$\delta = yr = \frac{yh(\mathbf{x})}{\|\mathbf{w}\|} \qquad (15.5)$$

В частности, для начала координат $\mathbf{x} = 0$ направленное расстояние равно

$$r = \frac{h(\mathbf{0})}{\|\mathbf{w}\|} = \frac{\mathbf{w}^T\mathbf{0} + b}{\|\mathbf{w}\|} = \frac{b}{\|\mathbf{w}\|}$$

как показано на рисунке 15.1.

Расстояние (зазор) и опорные векторы гиперплоскости

Для обучающего набора размеченных точек, $\mathbf{D} = \{\mathbf{x}_i, y_i\}_{i=1}^n$ с $y_i \in \{+1, -1\}$, и для разделяющей гиперплоскости $h(\mathbf{x}) = 0$, для каждой точки $\mathbf{x}_i$ мы можем найти её расстояние до гиперплоскости по формуле (15.5):

$$\delta_i = \frac{y_i h(\mathbf{x}_i)}{\|\mathbf{w}\|} = \frac{y_i(\mathbf{w}^T\mathbf{x}_i + b)}{\|\mathbf{w}\|}$$

По всем n точкам мы определяем границу линейного классификатора как минимальное расстояние точки от разделяющей гиперплоскости, заданное как

$$\delta^* = \min_{\mathbf{x}_i}\left\{\frac{y_i(\mathbf{w}^T\mathbf{x}_i + b)}{\|\mathbf{w}\|}\right\} \qquad (15.6)$$

Заметим, что $\delta^* \neq 0$, так как $h(\mathbf{x})$ – разделяющая гиперплоскость, и должно выполняться уравнение (15.3).

Все точки (или векторы), которые достигают этого минимального расстояния, называются опорными векторами для гиперплоскости. Другими словами, опорный вектор $\mathbf{x}^*$ - точка, которая лежит точно на границе классификатора и, таким образом, удовлетворяет условию

$$\delta^* = \frac{y^*(\mathbf{w}^T\mathbf{x}^* + b)}{\| \mathbf{w} \|}$$

где y^* - метка класса для y^*. Числитель $y^*(\mathbf{w}^T\mathbf{x}^* + b)$ дает абсолютное расстояние опорного вектора к гиперплоскости, а знаменатель $\| \mathbf{w} \|$ делает его относительным расстоянием в терминах $\mathbf{w}$.

Каноническая гиперплоскость

Рассмотрим уравнение гиперплоскости [Ур. (15.2)]. Умножение с обеих сторон на некоторый скаляр s дает эквивалентную гиперплоскость:

$$sh(\mathbf{x}) = s\mathbf{w}^T\mathbf{x} + sb = (s\mathbf{w})^T\mathbf{x} + (sb) = 0$$

Чтобы получить единственную или каноническую гиперплоскость, мы выбираем скаляр s так, чтобы абсолютное расстояние вектора поддержки от гиперплоскости равнялось 1. То есть,

$$sy^*(\mathbf{w}^T\mathbf{x}^* + b) = 1$$

что подразумевает

$$s = \frac{1}{y^*(\mathbf{w}^T\mathbf{x}^* + b)} = \frac{1}{y^*h(\mathbf{x}^*)} \qquad (15.7)$$

В дальнейшем мы будем считать любую разделяющую гиперплоскость канонической. То есть, она уже соответствующим образом масштабирована так, что $y^*h(\mathbf{x}^*) = 1$ для вектора поддержки $\mathbf{x}^*$, а расстояние (зазор) задается как

$$\delta^* = \frac{y^*h(\mathbf{x}^*)}{\| \mathbf{w} \|} = \frac{1}{\| \mathbf{w} \|}$$

Для канонической гиперплоскости, для каждого вектора поддержки $\mathbf{x}_i^*$ (с меткой y_i^*), мы имеем $y_i^*h(\mathbf{x}_i^*) = 1$, и для любой точки, которая не является опорным вектором мы имеем $y_ih(\mathbf{x}_i) > 1$, потому что, по определению, он должен быть дальше от гиперплоскости, чем опорный вектор. Таким образом, по всем n точкам в наборе данных $\mathbf{D}$ мы получаем следующий набор неравенств:

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1, \text{ для всех точек } \mathbf{x}_i \in \mathbf{D} \qquad (15.8)$$

15.2 Случай линейного разбиения

Имея набор данных $\boldsymbol{D} = \{x_i, y_i\}_{\{i=1\}}^n$, где $\boldsymbol{x} \in \mathbb{R}^d, e \in \{+1, -1\}$, предположим, что точки можно линейно разбить, т.е. существует гиперплоскость, разбиение по которой безошибочно классифицирует каждую точку. Другими словами, все точки $y_i = +1$ лежат на одной стороне какой-либо гиперплоскости ($h(\boldsymbol{x}) > 0$), а все $y_i = -1$ лежат на другой стороне той же гиперплоскости ($h(\boldsymbol{x}) < 0$). Очевидно, что в этом случае количество подходящих гиперплоскостей бесконечно. Встает вопрос выбора гиперплоскости.

Гиперплоскость с максимальным зазором

Основная идея SVM – выбор канонической гиперплоскости, заданной вектором весом $\boldsymbol{w}$ и ошибкой b, которая дает наибольший зазор среди всех возможных разделяющих гиперплоскостей. Если δ_h^* - зазор гиперплоскости $h(\boldsymbol{x}) = 0$, то задача найти оптимальную гиперплоскость h^*:

$$h^* = \arg\max_h\{\delta_h^*\} = \arg\max_{\{w,b\}}\left\{\frac{1}{\|w\|}\right\}.$$

Задача SVM – поиск гиперплоскости, максимизирующей зазор $\frac{1}{\|w\|}$ при n ограничениях из (15.8), а именно $y_i\ (\boldsymbol{w}^T\boldsymbol{x_i} + b) \geq 1$ для всех точек $x \in D$. Заметьте, что максимизацию $\frac{1}{\|w\|}$ можно заменить минимизацией $\|w\|$. Эквивалентная формулировка задачи условной минимизации:

Целевая функция: $\min_{\{\mathbf{w},\mathrm{b}\}}\left\{\frac{\|\boldsymbol{w}\|^2}{2}\right\}$

Линейные ограничения: $y_i\ (\boldsymbol{w}^T\boldsymbol{x_i} + b) \geq 1, \forall \boldsymbol{x} \in \boldsymbol{D}$

Поставленную задачу *прямой* выпуклой минимизации с n линейными ограничениями можно решить стандартными алгоритмами оптимизации, как показано в главе 15.5. Однако, чаще решается *двойственная* задача, которую можно получить с помощью множителей Лагранжа. Основная идея – ввести множитель Лагранжа α_i для каждого ограничения, которое удовлетворяет условиями Каруша-Куна-Такера для оптимального решения:

$$\alpha_i(y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) - 1) = 0$$
$$\text{и } \alpha \geq 0$$

С учетом всех n ограничений, новая целевая функция – *Лагранжиан* – становится

$$\min_L L = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{n}\alpha_i(y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) - 1) \tag{15.9}$$

L минимизируется с учетом w и b, и взяв их нулевые значения, получаем

$$\frac{\partial}{\partial \mathbf{w}}L = \mathbf{w} - \sum_{i=1}^{n}\alpha_i y_i \mathbf{x}_i = \mathbf{0} \text{ или } \mathbf{w} = \sum_{i=1}^{n}\alpha_i y_i \mathbf{x}_i \tag{15.10}$$

$$\frac{\partial}{\partial b}L = \sum_{i=1}^{n}\alpha_i y_i = 0 \tag{15.11}$$

Вышеприведенные уравнения помогают получить важное предположение относительно оптимального веса вектора $\boldsymbol{w}$. В частности, (15.10) предполагает, что $\boldsymbol{w}$ можно выразить как линейную комбинацию точек данных $\boldsymbol{x}_i$ с коэффициентами в виде множителями Лагранжа,α, y_i. Затем, (15.11) предполагает, что сумма множителей Лагранжа, α_i, y_i должна быть равна 0.

Подставляя эти уравнения в (15.9), можно получить двойственную целевую функцию Лангранжа, заданную исключительно в терминах множителей Лагранжа.

$$L_{dual} = \frac{1}{2}\mathbf{w}^T\mathbf{w} - \mathbf{w}^T\left(\underbrace{\sum_{i=1}^{n}\alpha_i y_i \mathbf{x}_i}_{\mathbf{w}}\right) - b\underbrace{\sum_{i=1}^{n}\alpha_i y_i}_{0} + \sum_{i=1}^{n}\alpha_i$$

$$= -\frac{1}{2}\mathbf{w}^T\mathbf{w} + \sum_{i=1}^{n}\alpha_i$$

$$= \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j$$

Таким образом, двойственная задача:

Целевая функция: $\max_{\alpha} L_{dual} = \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j$ (15.12)

Линейные ограничения: $\alpha_i \geq 0, \forall i \in \mathbf{D}$, and $\sum_{i=1}^{n}\alpha_i y_i = 0$

где $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_n)^T$ – вектор из множителей Лагранжа. L_{dual} – задача выпуклого квадратичного программирования (обратите внимание на множители $\alpha_i\alpha_j$), которая может быть решена стандартными методами оптимизации. Решение задачи методом градиентного спуска представлено в разделе 15.5.

Вектор весов и ошибка

Получив значения α_i для $i = 1, \ldots, n,$ мы можем провести решение для вектора $\boldsymbol{w}$ и ошибки b. С учетом условия Каруша-Куна-Такера

$$\alpha_i(y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) - 1) = 0,$$

что дает две возможности:

(1) $\alpha_i = 0$, или

(2) $y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) - 1 = 0$, что предполагает $y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) = 1$

Это важный результат, поскольку если $\alpha_i > 0$, то $y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) = 1$, и точка $\boldsymbol{x_i}$ должна быть опорным вектором. С другой стороны, если $y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) > 1$, т.е. точка – не опорный вектор, то $\alpha_i = 0$.

Как только мы знаем α_i для всех точек, мы можем вычислить вектор весов $\boldsymbol{w}$ с помощью уравнения (15.10), но суммирую только опорные векторы:

$$\boldsymbol{w} = \sum_{i,\alpha_i>0}\alpha_i y_i \boldsymbol{x}_i \tag{15.13}$$

Другими словами, $\boldsymbol{w}$ получен как линейная комбинация опорных векторов, где $\alpha_i y_i$ представляют веса. Остальные точки (с $\alpha = 0$) – не опорные вектора и не важны при определении $\boldsymbol{w}$.

Чтобы вычислить ошибку b, сначала вычисляется одно решение b_i для одного опорного вектора:

$$\alpha_i(y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1) = 0$$
$$y_i(\mathbf{w}^T\mathbf{x}_i + b) = 1$$

$$b_i = \frac{1}{y_i} - \mathbf{w}^T \mathbf{x}_i = y_i - \mathbf{w}^T \mathbf{x}_i \tag{15.14}$$

b берется как средняя ошибка по опорным векторам:

$$b = avg_{\alpha_i > 0} \{b_i\} \tag{15.15}$$

Классификатор SVM

Имея оптимальную гиперплоскость $h(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + b$, для каждой новой точки $\boldsymbol{z}$ можно предсказать её класс как

$$\hat{y} = sign\big(h(\boldsymbol{z})\big) = sign(\boldsymbol{w}^T \boldsymbol{z} + b) \tag{15.16}$$

где $sign$ – функция, возвращающая $+1$ для положительного аргумента и -1 для отрицательного.

15.3 SVM с мягким зазором: случай линейной неразделимости

Пока мы предполагали, что набор данных можно безошибочно линейно разделить. Теперь рассмотрим случай, когда классы до некоторой степени пересекаются, так что безошибочное разделение невозможно, как показано на рис. 15.3

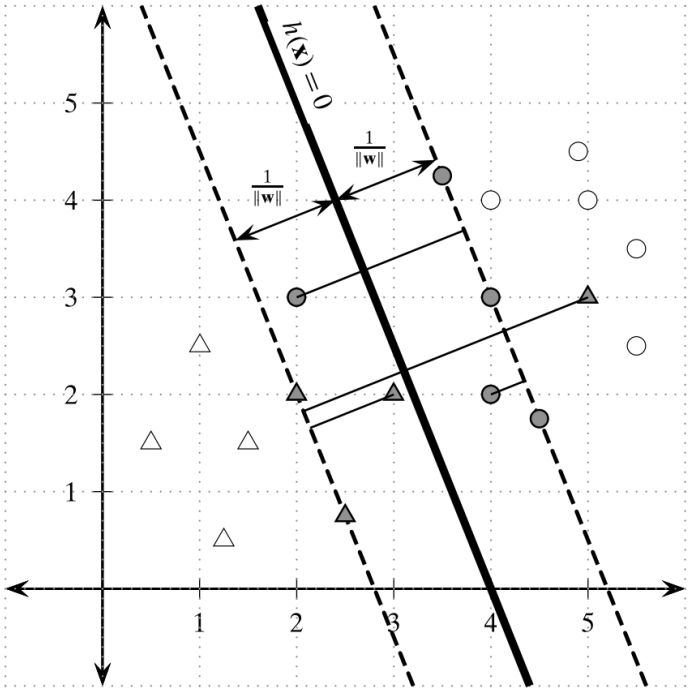


Рисунок 15.3. Гиперплоскость с мягким зазором: серые точки – опорные векторы. Показан зазор $\frac{1}{\|\boldsymbol{w}\|}$

Вспомним, что, когда точки линейно разделимы, можно найти разделяющую гиперплоскость, такую, что для всех точек $y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) \geq 1$. Обработать неразделимые точки можно, введя *слабые переменные* ξ_i в уравнение (15.8) следующим образом:

$$y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) \geq 1 - \xi_i,$$

где $\xi_i \geq 0 -$ слабая переменная для точки $\boldsymbol{x}_i$, показывающая, насколько точка нарушает условие разбиение, т.е. точка не обязаны быть на расстоянии как минимум $\frac{1}{\|\boldsymbol{w}\|}$ от гиперплоскости. Слабые переменные обозначают 3 типа точек. Если ξ_0, то точка на

расстоянии хотя бы $\frac{1}{\|w\|}$ от гиперплоскости. Если $0 < \xi_i < 1$, то точка внутри зазора и всё ещё корректно классифицируется, т.е. на правильной стороне гиперплоскости. Если $\xi_i \geq 1$, точка классифицируется неправильно и находится на неправильной стороне гиперплоскости.

В случае линейной неразделимости (иначе – случай с *мягким зазором*), цель SVM – найти гиперплоскость с максимальным зазором и минимальными слабыми переменными. Тогда новая цель:

$$\textbf{Целевая функция: } \min_{\mathbf{w},b,\xi_i} \left\{ \frac{\|\mathbf{w}\|^2}{2} + C \sum_{i=1}^{n} (\xi_i)^k \right\}$$

$$\textbf{Линейные ограничения: } \begin{matrix} y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i, \forall \mathbf{x}_i \in \mathbf{D} \\ \xi_i \geq 0 \forall \mathbf{x}_i \in \mathbf{D} \end{matrix} \quad (15.17)$$

Где C, k – ограничения, включающие в себя стоимость ошибочной классификации. Член $\sum_{i=1}^{n} (\xi_i)^k$ показывает *потерю*, т.е. отклонение от случая линейной разделимости. Скаляр C, выбирающийся эмпирически – *константа регуляризации*, контролирующая компромисс между максимизацией зазора (минимизацией $\frac{\|w\|^2}{2}$) или минимизацией потери (минимизацией суммы слабых членов $\sum_{i=1}^{n} (\xi_i)^k$). Так, если $C \to 0$, то компонент потери пропадает, и цель становится исключительно максимизацией зазора. Если же $C \to \infty$, то зазор перестает оказывать существенный эффект, и целевая функция минимизирует потерю. Константа k показывает вид потери. Обычно, k ставится в 1 или 2. Когда $k = 1$, это называется *петлевая функция потери* (hinge loss), и цель – минимизация суммы слабых переменных, когда $k = 2$, это *квадратичная функция потерь*, и цель – минимизация суммы квадратов слабых переменных

Петлевая функция потерь

Предположив $k = 1$, можно вычислить Лагранжиан для задачи (15.17), введя множители Лагранжа α_i, β_i, которые удовлетворяют следующим условиям Каруша-Куна-Такера для оптимального решения:

$$\begin{matrix} \alpha_i(y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1 + \xi_i) = 0 \text{ при } \alpha_i \geq 0 \\ \beta_i(\xi_i - 0) = 0 \text{ при } \beta_i \geq 0 \end{matrix} \quad (15.18)$$

И Лагранжиан задается как:

$$L = \frac{1}{2} \| \mathbf{w} \|^2 + C \sum_{i=1}^{n} \xi_i - \sum_{i=1}^{n} \alpha_i(y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1 + \xi_i) - \sum_{i=1}^{n} \beta_i \xi_i \quad (15.19)$$

Взятием частных производных по $\boldsymbol{w}, b, \xi$, и приравниванием их к 0, получаем:

$$\begin{matrix} \frac{\partial}{\partial \mathbf{w}} L = \mathbf{w} - \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i = \mathbf{0} \text{ or } \mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i \\ \frac{\partial}{\partial b} L = \sum_{i=1}^{n} \alpha_i y_i = 0 \\ \frac{\partial}{\partial \xi_i} L = C - \alpha_i - \beta_i = 0 \text{ or } \beta_i = C - \alpha_i \end{matrix} \quad (15.20)$$

Постановка этих значений в (15.19):

$$L_{\text{dual}} = \frac{1}{2}\mathbf{w}^T\mathbf{w} - \mathbf{w}^T\underbrace{\left(\sum_{i=1}^{n}\alpha_i y_i \mathbf{x}_i\right)}_{\mathbf{w}} - b\underbrace{\sum_{i=1}^{n}\alpha_i y_i}_{0} + \sum_{i=1}^{n}\alpha_i + \sum_{i=1}^{n}\underbrace{(C-\alpha_i-\beta_i)}_{0}\xi_i$$

$$= \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j$$

И двойственная задача:

Целевая функция: $\max_{\alpha} L_{dual} = \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j$

Линейные ограничения: $0 \le \alpha_i \le C, \forall i \in \mathbf{D}$ и $\sum_{i=1}^{n}\alpha_i y_i = 0$ (15.21)

Заметьте, что целевая функция – такая же, как и для случая линейной разделимости (15.12). Однако, ограничения на α_i теперь другие, т.к. мы требуем $\alpha_i + \beta_i = C$, где $\alpha_i \ge 0$ и $\beta_i \ge 0$, что предполагает $0 \le \alpha_i \le C$. Способ решения этой задачи методом градиентного спуска показан в разделе 15.5.

Вектор весов и потеря

После решения для α_i, мы оказываемся в той же ситуации, что и раньше – точки $\alpha_i = 0 -$ не опорные вектора, и $\alpha_i > 0$ только для опорных векторов, составляющих все $\boldsymbol{x}_i$, для которых

$$y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) = 1 - \xi \tag{15.22}$$

Заметим, что опорные вектора теперь включают как все точки вне зазора, для которых $\xi_i = 0$, так и точки $\xi_i > 0$.

Как и раньше, вектор весов можно получить из опорных векторов:

$$\mathbf{w} = \sum_{i,\alpha_i>0}\alpha_i y_i \mathbf{x}_i \tag{15.23}$$

Можно получить решение для β_i с помощью уравнения (15.20):

$$\beta_i = C - \alpha_i$$

Заменяя β_i в условиях Каруша-Куна-Такера (15.18), получаем:

$$(\mathrm{C} - \alpha_i)\xi_i = 0 \tag{15.14}$$

Т.е. для опорных векторов с $\alpha_i > 0$ возможны 2 случая:

$\xi_i > 0$, что предполагает $C - \alpha_i = 0$, т.е. $\alpha_i = C$, или

$C - \alpha_i > 0$, т.е. $\alpha_i < C$. В этом случае $\xi_i = 0$, и эти опорные вектора находятся вне зазора.

С использованием опорных векторов вне зазора, для которых $0 < \alpha_i < C$ и $\xi_i = 0$, можно решить для b_i:

$$\begin{aligned} &\alpha_i(y_i(\mathbf{w}^T\mathbf{x}_i + b_i) - 1) = 0 \\ &y_i(\mathbf{w}^T\mathbf{x}_i + b_i) = 1 \\ &b_i = \frac{1}{y_i} - \mathbf{w}^T\mathbf{x}_i = y_i - \mathbf{w}^T\mathbf{x}_i \end{aligned} \tag{15.25}$$

Чтобы получить итоговую ошибку b, нужно среднее по всем b_i. Из (15.23) и (15.25), и вектор весов $\boldsymbol{w}$, и ошибка b могут быть вычислены без явного вычисления членов ξ для каждой точки.

Когда оптимальная гиперплоскость определена, модель SVM предсказывает класс точки $\boldsymbol{z}$ следующим образом:

$$\hat{y} = \text{sign}\big(h(\mathbf{z})\big) = \text{sign}(\mathbf{w}^T\mathbf{z} + b)$$

Квадратичная потеря

Для квадратичной потери имеем $k = 2$ в целевой функции (15.17). В этом случае, ограничение на положительность $\xi_i \geq 0$ можно отбросить, т.к. (1) сумма слабых членов $\sum_{i=1}^{n} \xi^2$ всегда положительна, и (2) возможные отрицательные значения слабых членов исключаются во время оптимизации, потому что выбор $\xi_i = 0$ приводит к уменьшению значения основной цели, и всё ещё удовлетворяет ограничению $y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) \geq 1 - \xi_i$ когда $\xi_i < 0$. Другими словами, процесс оптимизации заменит отрицательные значения слабых членов нулями. Таким образом, задача SVM для квадратичной потери задана следующим образом:

Целевая функция: $\underset{\mathbf{w},b,\xi_i}{min} \left\{\frac{\|\mathbf{w}\|^2}{2} + C\sum_{i=1}^{n} (\xi_i)^2\right\}$

Линейные ограничения: $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i, \forall \mathbf{x}_i \in \mathbf{D}$

И Лагранжиан:

$$L = \frac{1}{2} \| \mathbf{w} \|^2 + C\sum_{i=1}^{n} \xi_i^2 - \sum_{i=1}^{n} \alpha_i(y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1 + \xi_i) \qquad (15.26)$$

Дифференцированием по $\boldsymbol{w}$, b и ξ_i и приравниванием соответствующих результатов к 0, получаем:

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0$$

$$\xi_i = \frac{1}{2C}\alpha_i$$

Подстановка результата обратно в (15.26) дает двойственную задачу:

$$L_{dual} = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j - \frac{1}{4C}\sum_{i=1}^{n} \alpha_i^2$$

$$= \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i\alpha_j y_i y_j \left(\mathbf{x}_i^T\mathbf{x}_j + \frac{1}{2C}\delta_{ij}\right)$$

где $\delta - дельта\text{-}функция\ Кронекера$, заданная как $\delta_{ij} = 1$, где $i = 1$, и $\delta_{ij} = 0$ во всех остальных случаях. Таким образом, двойственная задача:

$$\max_{\alpha} L_{dual} = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \left(\mathbf{x}_i^T \mathbf{x}_j + \frac{1}{2C}\delta_{ij}\right)$$
$$\text{с ограничениями } \alpha_i \geq 0, \forall i \in \mathbf{D}, \text{ и } \sum_{i=1}^{n} \alpha_i y_i = 0 \tag{15.27}$$

Решая для α_i методами из раздела 15.5, можно получить вектор весов и ошибку следующим образом:

$$\mathbf{w} = \sum_{i,\alpha_i>0} \alpha_i y_i \mathbf{x}_i$$
$$b = \text{avg}_{i,C>\alpha_i>0} \{y_i - \mathbf{w}^T \mathbf{x}_i\}$$

15.4 Ядерный SVM: нелинейный случай

Подход линейного SVM можно использовать для наборов данных с нелинейной границей решения с помощью трюка с ядром из главы 5. Концептуально идея состоит в том, чтобы отобразить исходные d-мерные точки $\mathbf{x}_i$ во входном пространстве в точки $\phi(\mathbf{x}_i)$в многомерном пространстве. пространство признаков с помощью некоторого нелинейного преобразования φ. Учитывая дополнительную гибкость, более вероятно, что точки $\phi(\mathbf{x}_i)$ могут быть линейно отделимы в пространстве признаков. Обратите внимание, однако, что линейная поверхность принятия решений в пространстве признаков фактически соответствует нелинейной поверхности принятия решений во входном пространстве. Кроме того, трюк с ядром позволяет нам выполнять все операции через функцию ядра, вычисляемую во входном пространстве вместо того, чтобы отображать точки в пространство признаков.

Чтобы применить трюк с ядром для нелинейной классификации SVM, мы должны показать, что для всех операций требуется только функция ядра:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

Пусть исходная база данных задана как $\mathbf{D} = \{\mathbf{x}_i, y_i\}_{i=1}^n$. Применяя ϕ к каждой точке, мы можем получить новый набор данных в пространстве признаков $\mathbf{D}_\phi = \{\phi(\mathbf{x}_i), y_i\}_{i=1}^n$.

Целевая функция SVM [уравнение (15.17)] в пространстве признаков задается как

Целевая функция:

$$\min_{\mathbf{w},b,\xi_i} \left\{ \frac{\| \mathbf{w} \|^2}{2} + C \sum_{i=1}^{n} (\xi_i)^k \right\} \tag{15.28}$$

Линейные ограничения:

$$y_i(\mathbf{w}^T \phi(\mathbf{x}_i) + b) \geq 1 - \xi_i, \text{ and } \xi_i \geq 0, \forall \mathbf{x}_i \in \mathbf{D}$$

где $\mathbf{w}$ – вектор весов, b – смещение, а ξ_i – переменные запаса (slack), все в пространстве признаков.

Функция потерь

Для функции потерь двойственная задача Лагранжа [Ур. (15.21)] в пространстве признаков задается как

$$\begin{aligned} \max_{\alpha} L_{dual} &= \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j) \\ &= \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \end{aligned} \tag{15.29}$$

Учитываются ограничения $0 \le \alpha_i \le C$ и $\sum_{i=1}^{n} \alpha_i y_i = 0$. Обратите внимание, что двойственный лагранжиан зависит только от скалярного произведения двух векторов в пространстве признаков $\phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j) = K(\mathbf{x}_i, \mathbf{x}_j)$, и, таким образом, мы можем решить задачу оптимизации, используя матрицу ядра $\mathbf{K} = \{K(\mathbf{x}_i, \mathbf{x}_j)\}_{i,j=1,\ldots,n}$. В разделе 15.5 описывается подход, основанный на стохастическом градиенте, для решения двойственной целевой функции.

Квадратичная функция потерь

Для квадратичных потерь двойственная задача Лагранжа [уравнение (15.27)] соответствует замене ядра. Определим новую функцию ядра K_q следующим образом:

$$K_q(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j + \frac{1}{2C}\delta_{ij} = K(\mathbf{x}_i, \mathbf{x}_j) + \frac{1}{2C}\delta_{ij}$$

которая влияет только на диагональные элементы матрицы ядра K, так как $\delta_{ij} = 1$ iff $i = j$, и ноль в противном случае. Таким образом, двойственная задача Лагранжа имеет вид

$$\max_{\alpha} L_{dual} = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K_q(\mathbf{x}_i, \mathbf{x}_j) \tag{15.30}$$

Учитываются ограничения $\alpha_i \ge 0$ и $\sum_{i=1}^{n} \alpha_i y_i = 0$. Вышеупомянутая оптимизация может быть решена с использованием того же подхода, что и для функции потерь, с простой заменой ядра.

Вектор веса и смещение

Мы можем решить для $\mathbf{w}$ в пространстве признаков следующим образом:

$$\mathbf{w} = \sum_{\alpha_i > 0} \alpha_i y_i \phi(\mathbf{x}_i) \tag{15.31}$$

Поскольку $\mathbf{w}$ использует $\phi(\mathbf{x}_i)$ напрямую, в общем случае мы не можем или не хотим вычислять $\mathbf{w}$ явно. Однако, как мы увидим далее, нет необходимости явно вычислять $\mathbf{w}$ для классификации точек.

Давайте теперь посмотрим, как вычислить смещение с помощью операций ядра. Используя уравнение (15.25), мы вычисляем b как среднее по опорным векторам, которые находятся на грани, то есть с $0 < \alpha_i < C$ и $\xi_i = 0$:

$$b = \text{avg}_{i, 0<\alpha_i<C}\{b_i\} = \text{avg}_{i, 0<\alpha_i<C}\{y_i - \boldsymbol{w}^T \phi(\boldsymbol{x}_i)\} \tag{15.32}$$

Подставляя $\mathbf{w}$ из уравнения. (15.31), получаем новое выражение для b_i:

$$
\begin{aligned}
b_i &= y_i - \sum_{\alpha_j > 0} \alpha_j y_j \boldsymbol{\phi}\left(\mathbf{x}_j\right)^T \boldsymbol{\phi}(\mathbf{x}_i) \\
&= y_i - \sum_{\alpha_j > 0} \alpha_j y_j K\left(\mathbf{x}_j, \mathbf{x}_i\right)
\end{aligned} \tag{15.33}
$$

Обратите внимание, что b_i является функцией скалярного произведения двух векторов в пространстве признаков, и поэтому его можно вычислить с помощью функции ядра во входном пространстве.

15.4 Ядерный SVM классификатор

Мы можем предсказать класс для новой точки $\mathbf{z}$ следующим образом:

$$
\begin{aligned}
\hat{y} &= \operatorname{sign}\left(\mathbf{w}^T \boldsymbol{\phi}(\mathbf{z}) + b\right) \\
&= \operatorname{sign}\left(\sum_{\alpha_i > 0} \alpha_i y_i \boldsymbol{\phi}(\mathbf{x}_i)^T \boldsymbol{\phi}(\mathbf{z}) + b\right) \\
&= \operatorname{sign}\left(\sum_{\alpha_i > 0} \alpha_i y_i K(\mathbf{x}_i, \mathbf{z}) + b\right)
\end{aligned}
$$

Мы снова видим, что $\hat{y}$ использует только скалярные произведения в пространстве признаков.

Основываясь на приведенных выше выводах, мы можем видеть, что для обучения и тестирования классификатора SVM сопоставленные точки $\boldsymbol{\phi}(\mathbf{x}_i)$ никогда не могут быть изолированными. Вместо этого все операции могут быть выполнены в терминах ядерной функции $K\left(\mathbf{x}_i, \mathbf{x}_j\right) = \boldsymbol{\phi}(\mathbf{x}_i)^T \boldsymbol{\phi}\left(\mathbf{x}_j\right)$. Таким образом, любая нелинейная функция ядра может использоваться для нелинейной классификации во входном пространстве. Примеры таких нелинейных ядер включают полиномиальное ядро, ядро Гаусса и другие.

15.5 Алгоритмы обучения SVM

Теперь обратим внимание на алгоритмы решения задач оптимизации SVM. Мы рассмотрим простые подходы к оптимизации как для решения двойственных, так и для простых формулировок. Важно отметить, что эти методы не самые эффективные. Однако, поскольку они относительно просты, они могут служить отправной точкой для более сложных методов.

Для алгоритмов SVM в этом разделе вместо того, чтобы явно иметь дело со смещением b, мы отображаем каждую точку $\mathbf{x}_i \in \mathbb{R}^d$ в точку $\mathbf{x}_i' \in \mathbb{R}^{d+1}$ следующим образом:

$$
\mathbf{x}_i' = (x_{i1}, \ldots, x_{id}, 1)^T \tag{15.34}
$$

Кроме того, мы также отображаем весовой вектор в $\mathbb{R}^{d+1}$, где $w_{d+1} = b$, так что

$$
\mathbf{w} = (w_1, \ldots, w_d, b)^T \tag{15.35}
$$

Уравнение гиперплоскости [уравнение (15.1)] тогда задается следующим образом:

$$h(\mathbf{x}'): \mathbf{w}^T\mathbf{x}' = 0$$

$$h(\mathbf{x}'): (w_1 \quad \cdots \quad w_d \quad b)\begin{pmatrix} x_{i1} \\ \vdots \\ x_{id} \\ 1 \end{pmatrix} = 0$$

$$h(\mathbf{x}'): w_1 x_{i1} + \cdots + w_d x_{id} + b = 0$$

В обсуждении ниже мы предполагаем, что член смещения был включен в **w**, и что каждая точка была отображена в $\mathbb{R}^{d+1}$ согласно уравнениям (15.34) и (15.35). Таким образом, последняя компонента **w** дает смещение b. Другим следствием отображения точек на $\mathbb{R}^{d+1}$ является то, что ограничение $\sum_{i=1}^{n} \alpha_i y_i = 0$ не применяется в двойственных формулировках SVM, приведенных в уравнениях (15.21), (15.27), (15.29) и (15.30), поскольку нет явного члена смещения b для линейных ограничений в целевой функции SVM, заданной в уравнении (15.17). Новый набор ограничений задается как

$$y_i \mathbf{w}^T \mathbf{x} \geq 1 - \xi_i$$

15.5.1 Двойственное решение: стохастический градиентный подъем

Мы рассматриваем только случай функции потерь, потому что квадратичные потери могут быть обработаны путем изменения ядра, как показано в уравнении (15.30). Двойственная целевая функция оптимизации для функции потерь [Ур. (15.29)] имеет вид

$$\max_{\alpha} J(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

при ограничениях $0 \leq \alpha_i \leq C \; for \; all \; i = 1, \ldots, n$. Тут $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \cdots, \alpha_n)^T \in \mathbb{R}^n$.

Рассмотрим слагаемые в $J(\boldsymbol{\alpha})$, в которые входит множитель Лагранжа α_k:

$$J(\alpha_k) = \alpha_k - \frac{1}{2}\alpha_k^2 y_k^2 K(\mathbf{x}_k, \mathbf{x}_k) - \alpha_k y_k \sum_{\substack{i=1 \\ i \neq k}}^{n} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}_k)$$

Градиент или скорость изменения целевой функции при $\boldsymbol{\alpha}$ задается как частная производная от $J(\boldsymbol{\alpha})$ по $\boldsymbol{\alpha}$, то есть по каждому α_k:

$$\nabla J(\boldsymbol{\alpha}) = \left(\frac{\partial J(\boldsymbol{\alpha})}{\partial \alpha_1}, \frac{\partial J(\boldsymbol{\alpha})}{\partial \alpha_2}, \ldots, \frac{\partial J(\boldsymbol{\alpha})}{\partial \alpha_n}\right)^T$$

где k-я компонента градиента получается дифференцированием $J(\alpha_k)$ по α_k:

$$\frac{\partial J(\boldsymbol{\alpha})}{\partial \alpha_k} = \frac{\partial J(\alpha_k)}{\partial \alpha_k} = 1 - y_k\left(\sum_{i=1}^{n} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}_k)\right) \tag{15.36}$$

Поскольку мы хотим максимизировать целевую функцию $J(\boldsymbol{\alpha})$, мы должны двигаться в направлении градиента $\nabla J(\boldsymbol{\alpha})$. Начиная с начального $\boldsymbol{\alpha}$, подход градиентного подъема последовательно обновляет его следующим образом:

$$\boldsymbol{\alpha}_{t+1} = \boldsymbol{\alpha}_t + \eta_t \nabla \boldsymbol{J}(\boldsymbol{\alpha}_t)$$

где $\boldsymbol{\alpha}_t$ – оценка на шаге t, η_t – размер шага.

Вместо обновления всего вектора $\boldsymbol{\alpha}$ на каждом шаге в подходе стохастического градиентного подъема мы обновляем каждый компонент α_k независимо и сразу же используем новое значение для обновления других компонентов. Это может привести к более быстрой сходимости. Правило обновления для k-го компонента задается как

$$\alpha_k = \alpha_k + \eta_k \frac{\partial J(\boldsymbol{\alpha})}{\partial \alpha_k} = \alpha_k + \eta_k \left(1 - y_k \sum_{i=1}^{n} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}_k)\right) \tag{15.37}$$

где η_k – размер шага. Мы также должны обеспечить выполнение ограничений $\alpha_k \in [0, C]$. Таким образом, на шаге обновления выше, если $\alpha_k < 0$, мы сбрасываем его до $\alpha_k = 0$, а если $\alpha_k > C$, мы сбрасываем его до $\alpha_k = C$. Псевдокод для стохастического градиентного подъема продемонстрирован в алгоритме 15.1.

Алгоритм 15.1: Dual SVM: Стохастический градиентный подъем

SVM-DUAL $(\mathbf{D}, K, C, \epsilon)$:

foreach $\mathbf{x}_i \in \mathbf{D}$ **do** $\mathbf{x}_i \leftarrow \begin{pmatrix} \mathbf{x}_i \\ 1 \end{pmatrix}$ // map to $\mathbb{R}^{d+1}$

if *loss* = `hinge` **then**

  $\mathbf{K} \leftarrow \{K(\mathbf{x}_i, \mathbf{x}_j)\}_{i,j=1,\ldots,n}$ // kernel matrix, hinge loss

else if *loss* = `quadratic` **then**

  $\mathbf{K} \leftarrow \{K(\mathbf{x}_i, \mathbf{x}_j) + \frac{1}{2C}\delta_{ij}\}_{i,j=1,\ldots,n}$ // kernel matrix, quadratic loss

for $k = 1, \ldots, n$ **do** $\eta_k \leftarrow \frac{1}{K(\mathbf{x}_k, \mathbf{x}_k)}$ // set step size

$t \leftarrow 0$

$\boldsymbol{\alpha}_0 \leftarrow (0, \ldots, 0)^T$

repeat

  $\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha}_t$

  for $k = 1$ *to* n **do**

    // update kth component of $\boldsymbol{\alpha}$

    $\alpha_k \leftarrow \alpha_k + \eta_k \left(1 - y_k \sum_{i=1}^{n} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}_k)\right)$

    if $\alpha_k < 0$ **then** $\alpha_k \leftarrow 0$

    if $\alpha_k > C$ **then** $\alpha_k \leftarrow C$

  $\boldsymbol{\alpha}_{t+1} \leftarrow \boldsymbol{\alpha}$

  $t \leftarrow t + 1$

until $\|\boldsymbol{\alpha}_t - \boldsymbol{\alpha}_{t-1}\| \leq \epsilon$

В идеале, чтобы определить размер шага η_k, мы хотели бы выбрать его так, чтобы градиент при α_k стремился к нулю, что происходит, когда

$$\eta_k = \frac{1}{K(\mathbf{x}_k, \mathbf{x}_k)} \tag{15.38}$$

Чтобы понять, почему, обратите внимание, что, когда обновляется только α_k, другие α_i не меняются. Таким образом, новый α имеет изменение только в α_k, и из уравнения (15.36) получаем

$$\frac{\partial J(\boldsymbol{\alpha})}{\partial \alpha_k} = \left(1 - y_k \sum_{i \neq k} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}_k)\right) - y_k \alpha_k y_k K(\mathbf{x}_k, \mathbf{x}_k)$$

Подставляя значение α_k из уравнения (15.37), имеем

$$\begin{aligned}\frac{\partial J(\boldsymbol{\alpha})}{\partial \alpha_k} &= \left(1 - y_k \sum_{i \neq k} \alpha_i y_i \boldsymbol{K}(\mathbf{x}_i, \mathbf{x}_k)\right) - \left(\alpha_k + \eta_k \left(1 - y_k \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{K}(\mathbf{x}_i, \mathbf{x}_k)\right)\right) \boldsymbol{K}(\mathbf{x}_k, \mathbf{x}_k) \\ &= \left(1 - y_k \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{K}(\mathbf{x}_i, \mathbf{x}_k)\right) - \eta_k \boldsymbol{K}(\mathbf{x}_k, \mathbf{x}_k) \left(1 - y_k \sum_{i=1}^{n} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}_k)\right) \\ &= \left(1 - \eta_k \boldsymbol{K}(\mathbf{x}_k, \mathbf{x}_k)\right) \left(1 - y_k \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{K}(\mathbf{x}_i, \mathbf{x}_k)\right)\end{aligned}$$

Подставляя η_k из уравнения (15.38), имеем

$$\frac{\partial J(\boldsymbol{\alpha})}{\partial a_k} = \left(1 - \frac{1}{K(\mathbf{x}_k, \mathbf{x}_k)} K(\mathbf{x}_k, \mathbf{x}_k)\right) \left(1 - y_k \sum_{i=1}^{n} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}_k)\right) = 0$$

Таким образом, в алгоритме 15.1 для лучшей сходимости мы выбираем η_k согласно формуле (15.38). Метод последовательно обновляет $\boldsymbol{\alpha}$ и останавливается, когда изменение падает ниже заданного порога ϵ. Поскольку приведенное выше описание предполагает общую функцию ядра между любыми двумя точками, мы можем восстановить линейный, неотделимый случай, просто установив $\boldsymbol{K}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i{}^T \mathbf{x}_j$. Вычислительная сложность метода составляет $O(n^2)$ на итерацию.

Обратите внимание, что после получения окончательного $\boldsymbol{\alpha}$ мы классифицируем новую точку $\mathbf{z} \in \mathbb{R}^{d+1}$ следующим образом:

$$\hat{y} = \text{sign}\left(h(\phi(\mathbf{z}))\right) = \text{sign}\left(\mathbf{w}^T \phi(\mathbf{z})\right) = \text{sign}\left(\sum_{\alpha_i > 0} \alpha_i y_i K(\mathbf{x}_i, \mathbf{z})\right)$$

15.5.2 Основное решение: оптимизация Ньютона

Двойственный подход является наиболее часто используемым для обучения SVM, но его также можно обучать, используя простую формулировку.

Рассмотрим функцию прямой оптимизации для линейного, но неотделимого случая [Ур. (15.17)]. При $\mathbf{w}, \mathbf{x}_i \in \mathbb{R}^{d+1}$, как обсуждалось ранее, мы должны минимизировать целевую функцию:

$$\min_{\mathbf{w}} J(\mathbf{w}) = \frac{1}{2} \| \mathbf{w} \|^2 + C \sum_{i=1}^{n} (\xi_i)^k \tag{15.39}$$

с учетом линейных ограничений

$$y_i(\mathbf{w}^T \mathbf{x}_i) \geq 1 - \xi_i \text{ and } \xi_i \geq 0 \text{ for all } i = 1, \ldots, n$$

Преобразуя написанное выше, получаем выражение для ξ_i

$\xi_i \geq 1 - y_i(\mathbf{w}^T\mathbf{x}_i)$ и $\xi_i \geq 0$ откуда следует, что

$$\xi_i = \max\{0, 1 - y_i(\mathbf{w}^T\mathbf{x}_i)\} \tag{15.40}$$

Подставляя уравнение (15.40) в целевую функцию [Ур. (15.39)], получаем

$$\begin{aligned} J(\mathbf{w}) &= \frac{1}{2} \parallel \mathbf{w} \parallel^2 + C \sum_{i=1}^{n} max\{0, 1 - y_i(\mathbf{w}^T\mathbf{x}_i)\}^k \\ &= \frac{1}{2} \parallel \mathbf{w} \parallel^2 + C \sum_{y_i(\mathbf{w}^T\mathbf{x}_i)<1} \left(1 - y_i(\mathbf{w}^T\mathbf{x}_i)\right)^k \end{aligned} \tag{15.41}$$

Последний шаг следует из уравнения (15.40), потому что $\xi_i > 0$ тогда и только тогда, когда $1 - y_i(\mathbf{w}^T\mathbf{x}_i) > 0$, то есть $y_i(\mathbf{w}^T\mathbf{x}_i) < 1$. К сожалению, формулировка функции потерь при $k = 1$ не дифференцируема. Можно было бы использовать дифференцируемую аппроксимацию функции потерь, но здесь мы опишем формулировку квадратичных потерь.

Квадратичная функция потерь

Для квадратичных потерь $k = 2$ целевую функцию [Ур. (15.41)] можно записать как

$$J(\mathbf{w}) = \frac{1}{2} \parallel \mathbf{w} \parallel^2 + C \sum_{y_i(\mathbf{w}^T\mathbf{x}_i)<1} \left(1 - y_i(\mathbf{w}^T\mathbf{x}_i)\right)^2$$

Градиент или скорость изменения целевой функции в $\mathbf{w}$ задается как частная производная $J(\mathbf{w})$ по отношению к $\mathbf{w}$:

$$\begin{aligned} \nabla_{\mathbf{w}} = \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} &= \mathbf{w} - 2C \left(\sum_{y_i(\mathbf{w}^T\mathbf{x}_i)<1} y_i\mathbf{x}_i\left(1 - y_i(\mathbf{w}^T\mathbf{x}_i)\right) \right) \\ &= \mathbf{w} - 2C \left(\sum_{y_i(\mathbf{w}^T\mathbf{x}_i)<1} \mathbf{y}_i \right) + 2C \left(\sum_{\mathbf{x}_i} \mathbf{x}_i\mathbf{x}_i^T \right)\mathbf{w} \\ &= \mathbf{w} - 2C\mathbf{v} + 2C\mathbf{S}\mathbf{w} \end{aligned}$$

где вектор $\mathbf{v}$ и матрица $\mathbf{S}$ заданы как

$$\mathbf{v} = \sum_{y_i(\mathbf{w}^T\mathbf{x}_i)<1} y_i\mathbf{x}_i \qquad \mathbf{S} = \sum_{y_i(\mathbf{w}^T\mathbf{x}_i)<1} \mathbf{x}_i\mathbf{x}_i^T$$

Обратите внимание, что матрица $\mathbf{S}$ является матрицей разброса, а вектор $\mathbf{v}$ в m раз больше среднего, скажем, m точек $y_i\mathbf{x}_i$, удовлетворяющих условию $y_i h(\mathbf{x}_i) < 1$.

Матрица Гессе определяется как матрица частных производных второго порядка $J(\mathbf{w})$ по $\mathbf{w}$, которая задается как

$$\mathbf{H}_{\mathrm{w}} = \frac{\partial \nabla_{\mathbf{w}}}{\partial \mathbf{w}} = \mathbf{I} + 2C\mathbf{S}$$

Поскольку мы хотим минимизировать целевую функцию $J(\mathbf{w})$, мы должны двигаться в направлении, противоположном градиенту. Правило обновления оптимизации Ньютона для $\mathbf{w}$ задается как

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \mathbf{H}_{\mathbf{w}_t}^{-1} \nabla_{\mathbf{w}_t} \tag{15.42}$$

где $\eta_t > 0$ – скалярное значение, обозначающее размер шага на итерации t. Обычно для нахождения оптимального размера шага η_t необходимо использовать метод линейного поиска, но значение по умолчанию $\eta_t = 1$ обычно работает для квадратичных потерь.

Алгоритм оптимизации Ньютона для обучения линейных неразделимых SVM приведен в алгоритме 15.2. Размер шага η_t по умолчанию равен 1. После вычисления градиента и гессиана в $\mathbf{w}_t$ (строки 6–9) правило обновления Ньютона используется для получения нового вектора весов $\mathbf{w}_{t+1}$ (строка 10). Итерации продолжаются до тех пор, пока весовой вектор не изменится очень незначительно. Вычисление $\mathbf{S}$ требует $O(nd^2)$ шагов; вычисление градиента ∇, матрицы Гессе $\mathbf{H}$ и обновление весового вектора $\mathbf{w}_{t+1}$ занимает время $O(d^2)$; и инвертирование гессиана требует $O(d^3)$ операций для общей вычислительной сложности $O(nd^2 + d^3)$ на итерацию в худшем случае.

Алгоритм 15.2: Primal SVM - Алгоритм оптимизации Ньютона

SVM-PRIMAL (D, C, ϵ):

1 **foreach** $\mathbf{x}_i \in \mathbf{D}$ **do**

2 $\quad \mathbf{x}_i \leftarrow \begin{pmatrix} \mathbf{x}_i \\ 1 \end{pmatrix}$ // map to $\mathbb{R}^{d+1}$

3 $t \leftarrow 0$

4 $\mathbf{w}_0 \leftarrow (0, \ldots, 0)^T$ // initialize $\mathbf{w}_t \in \mathbb{R}^{d+1}$

5 **repeat**

6 $\quad \mathbf{v} \leftarrow \sum_{y_i(\mathbf{w}_t^T \mathbf{x}_i) < 1} y_i \mathbf{x}_i$

7 $\quad \mathbf{S} \leftarrow \sum_{y_i(\mathbf{w}_t^T \mathbf{x}_i) < 1} \mathbf{x}_i \mathbf{x}_i^T$

8 $\quad \nabla \leftarrow (\mathbf{I} + 2C\mathbf{S})\mathbf{w}_t - 2C\mathbf{v}$ // gradient

9 $\quad \mathbf{H} \leftarrow \mathbf{I} + 2C\mathbf{S}$ // Hessian

10 $\quad \mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta_t \mathbf{H}^{-1} \nabla$ // Newton update rule [Eq. (21.42)]

11 $\quad t \leftarrow t + 1$

12 **until** $\|\mathbf{w}_t - \mathbf{w}_{t-1}\| \leq \epsilon$

Ядерные SVMs

В предыдущем обсуждении мы рассмотрели линейный неразделимый случай первичного обучения SVM. Теперь мы обобщаем первичный подход к изучению SVM на основе ядра, опять же для квадратичных потерь.

Пусть ϕ обозначает отображение из входного пространства в пространство признаков; каждая входная точка $\mathbf{x}_i$ отображается в характерную точку $\phi(\mathbf{x}_i)$. Пусть $K(\mathbf{x}_i, \mathbf{x}_j)$ обозначает ядерную функцию, а $\mathbf{w}$ обозначает весовой вектор в пространстве признаков. Гиперплоскость в пространстве признаков тогда задается как

$$h(\mathbf{x}): \mathbf{w}^T \phi(\mathbf{x}) = 0$$

Используя уравнения (15.28) и (15.40), основная целевая функция в пространстве признаков может быть записана как

$$\min_{\mathbf{w}} J(\mathbf{w}) = \frac{1}{2} \| \mathbf{w} \|^2 + C \sum_{i=1}^{n} L\big(y_i, h(\mathbf{x}_i)\big) \tag{15.43}$$

где $L\big(y_i, h(\mathbf{x}_i)\big) = max\{0, 1 - y_i h(\mathbf{x}_i)\}^k$ – функция потерь.

Градиент в $\mathbf{w}$ задается как

$$\nabla_{\mathbf{w}} = \mathbf{w} + C \sum_{i=1}^{n} \frac{\partial L\big(y_i, h(\mathbf{x}_i)\big)}{\partial h(\mathbf{x}_i)} \cdot \frac{\partial h(\mathbf{x}_i)}{\partial \mathbf{w}}$$

Где

$$\frac{\partial h(\mathbf{x}_i)}{\partial \mathbf{w}} = \frac{\partial \mathbf{w}^T \phi(\mathbf{x}_i)}{\partial \mathbf{w}} = \phi(\mathbf{x}_i)$$

При оптимальном решении градиент обращается в нуль, т. Е. $\nabla_{\mathbf{w}} = 0$, что дает

$$\begin{aligned} \mathbf{w} &= -C \sum_{i=1}^{n} \frac{\partial L\big(y_i, h(\mathbf{x}_i)\big)}{\partial h(\mathbf{x}_i)} \cdot \phi(\mathbf{x}_i) \\ &= \sum_{i=1}^{n} \beta_i \phi(\mathbf{x}_i) \end{aligned} \tag{15.44}$$

где β_i – коэффициент точки $\phi(\mathbf{x}_i)$ в пространстве признаков. Другими словами, оптимальный весовой вектор в пространстве признаков выражается как линейная комбинация точек $\phi(\mathbf{x}_i)$ в пространстве признаков.

Используя уравнение (15.44) расстояние до гиперплоскости в пространстве признаков можно выразить как

$$y_i h(\mathbf{x}_i) = y_i \mathbf{w}^T \phi(\mathbf{x}_i) = y_i \sum_{j=1}^{n} \beta_j K\big(\mathbf{x}_j, \mathbf{x}_i\big) = y_i \mathbf{K}_i^T \boldsymbol{\beta} \tag{15.45}$$

где $\mathbf{K} = \big\{K\big(\mathbf{x}_i, \mathbf{x}_j\big)\big\}_{i,j=1}^{n}$ – ядерная матрица $n \times n$, $\mathbf{K}_i$ – i-ая колонна $\mathbf{K}$ и $\boldsymbol{\beta} = (\beta_1, \dots, \beta_n)^T$ – вектор коэффициентов.

Подстановка уравнений (15.44) и (15.45) в уравнение (15.43) с квадратичными потерями (k = 2) дает формулировку SVM с ядром исключительно в терминах матрицы ядра.

$$\begin{aligned} \min_{\beta} J(\boldsymbol{\beta}) &= \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \beta_i \beta_j K\big(\mathbf{x}_i, \mathbf{x}_j\big) + C \sum_{i=1}^{n} max\{0, 1 - y_i \mathbf{K}_i^T \boldsymbol{\beta}\}^2 \\ &= \frac{1}{2} \boldsymbol{\beta}^T \mathbf{K} \boldsymbol{\beta} + C \sum_{y_i \mathbf{K}_i^T \beta < 1} (1 - y_i \mathbf{K}_i^T \boldsymbol{\beta})^2 \end{aligned}$$

Градиент $J(\boldsymbol{\beta})$ относительно $\boldsymbol{\beta}$ задается как

$$\nabla_\beta = \frac{\partial J(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = \mathbf{K}\boldsymbol{\beta} - 2C \sum_{y_i \mathbf{K}_i^T \boldsymbol{\beta} < 1} y_i \mathbf{K}_i (1 - y_i \mathbf{K}_i^T \boldsymbol{\beta})$$
$$= \mathbf{K}\boldsymbol{\beta} + 2C \sum_{y_i \mathbf{K}_i^T \boldsymbol{\beta} < 1} (\mathbf{K}_i \mathbf{K}_i^T)\boldsymbol{\beta} - 2C \sum_{y_i \mathbf{K}_i^T \boldsymbol{\beta} < 1} \mathbf{K}_i$$
$$= (\mathbf{K} + 2C\mathbf{S})\boldsymbol{\beta} - 2C\mathbf{v}$$

где вектор $\mathbf{v} \in \mathbb{R}^n$ и матрица $\mathbf{S} \in \mathbb{R}^{n \times n}$ заданы как

$$\mathbf{v} = \sum_{y_i \mathbf{K}_i^T \beta < 1} y_i \mathbf{K}_i$$

$$\mathbf{S} = \sum_{y_i \mathbf{K}_i^T \beta < 1} \mathbf{K}_i \mathbf{K}_i^T$$

Кроме того, матрица Гессе имеет вид

$$\mathbf{H}_\beta = \frac{\partial \nabla_\beta}{\partial \beta} = \mathbf{K} + 2C\mathbf{S}$$

Теперь мы можем минимизировать $J(\boldsymbol{\beta})$ с помощью оптимизации Ньютона, используя следующее правило обновления:

$$\boldsymbol{\beta}_{t+1} = \boldsymbol{\beta}_t - \eta_t \mathbf{H}_\beta^{-1} \nabla_{\boldsymbol{\beta}}$$

Заметим, что если $\mathbf{H}_\beta$ сингулярна, т.е. если у нее нет обратной, мы добавляем к диагонали небольшой гребень, чтобы регуляризовать ее. То есть сделаем $\mathbf{H}$ обратимым следующим образом:

$$\mathbf{H}_\beta = \mathbf{H}_\beta + \lambda \mathbf{I}$$

где $\lambda > 0$ – некоторое небольшое положительное значение гребня.

После определения $\boldsymbol{\beta}$ любую контрольную точку $\mathbf{z}$ легко классифицировать следующим образом:

$$\hat{y} = \text{sign}\,(\mathbf{w}^T \phi(\mathbf{z})) = \text{sign}\left(\sum_{i=1}^{n} \beta_i \phi(\mathbf{x}_i)^T \phi(\mathbf{z})\right) = \text{sign}\left(\sum_{i=1}^{n} \beta_i K(\mathbf{x}_i, \mathbf{z})\right)$$

Алгоритм оптимизации Ньютона для SVM ядра в первичном представлении приведен в алгоритме 15.3. Размер шага η_t по умолчанию установлен на 1, как и в линейном случае. На каждой итерации метод сначала вычисляет градиент и гессиан (строки 7–10). Затем используется правило обновления Ньютона для получения обновленного вектора коэффициентов $\boldsymbol{\beta}_{t+1}$ (строка 11). Итерации продолжаются до тех пор, пока $\boldsymbol{\beta}$ не изменится очень незначительно. Вычислительная сложность метода составляет $O(n^3)$ на итерацию в худшем случае.

Алгоритм 15.3: Primal Kernel SVM – Алгоритм оптимизации Ньютона для SVM ядра

SVM-PRIMAL-KERNEL ($\mathbf{D}, K, C, \epsilon$):

foreach $\mathbf{x}_i \in \mathbf{D}$ **do**

  $\mathbf{x}_i \leftarrow \begin{pmatrix} \mathbf{x}_i \\ 1 \end{pmatrix}$ // map to $\mathbb{R}^{d+1}$

$\mathbf{K} \leftarrow \{K(\mathbf{x}_i, \mathbf{x}_j)\}_{i,j=1,\ldots,n}$ // compute kernel matrix

$t \leftarrow 0$

$\boldsymbol{\beta}_0 \leftarrow (0, \ldots, 0)^T$ // initialize $\boldsymbol{\beta}_t \in \mathbb{R}^n$

repeat

  $\mathbf{v} \leftarrow \sum_{y_i(\mathbf{K}_i^T \boldsymbol{\beta}_t) < 1} y_i \mathbf{K}_i$

  $\mathbf{S} \leftarrow \sum_{y_i(\mathbf{K}_i^T \boldsymbol{\beta}_t) < 1} \mathbf{K}_i \mathbf{K}_i^T$

  $\nabla \leftarrow (\mathbf{K} + 2C\mathbf{S})\boldsymbol{\beta}_t - 2C\mathbf{v}$ // gradient

  $\mathbf{H} \leftarrow \mathbf{K} + 2C\mathbf{S}$ // Hessian

  $\boldsymbol{\beta}_{t+1} \leftarrow \boldsymbol{\beta}_t - \eta_t \mathbf{H}^{-1} \nabla$ // Newton update rule

  $t \leftarrow t + 1$

until $\left\| \boldsymbol{\beta}_t - \boldsymbol{\beta}_{t-1} \right\| \leq \epsilon$

Глава 16. Оценка классификации

В предыдущих главах мы рассмотрели различные классификаторы, такие как деревья решений, полные и наивные байесовские классификаторы, классификатор ближайших соседей, метод опорных векторов и т. д. В общем, мы можем думать о классификаторе как о модели или функции M, которая предсказывает метку класса $\hat{y}$ для данного входного примера $\mathbf{x}$:

$$\hat{y} = M(\mathbf{x})$$

где $\mathbf{x} = (x_1, x_2, \ldots, x_d)^T \in \mathbb{R}^d$ - точка в d-мерном пространстве, а $\hat{y} \in \{c_1, c_2, \ldots, c_k\}$ - ее предсказанный класс.

Чтобы построить модель классификации M, нам понадобится *обучающий набор* точек вместе с их известными классами. Различные классификаторы получаются в зависимости от допущений, используемых для построения модели M. Например, метод опорных векторов использует гиперплоскость с максимальным зазором для построения M. С другой стороны, байесовский классификатор напрямую вычисляет апостериорную вероятность $P(c_j \mid \mathbf{x})$ для каждого класса c_j, и предсказывает класс $\mathbf{x}$ как тот, у которого максимальная апостериорная вероятность, $\hat{y} = \arg max_{c_j} \{P(c_j \mid \mathbf{x})\}$. После обучения модели M, мы оцениваем ее производительность на отдельном *тестовом наборе* точек, для которых мы знаем истинные классы. Наконец, модель может быть развернута для прогнозирования класса будущих точек, класс которых мы обычно не знаем.

В этой главе мы рассмотрим методы оценки классификатора и сравнения нескольких классификаторов. Начнем с определения показателей точности классификатора. Затем мы обсудим, как определить границы ожидаемой ошибки. Наконец, мы обсудим, как оценивать эффективность классификаторов и сравнивать их.

16.1 Классификация мер эффективности

Пусть **D** - набор для тестирования, содержащий n точек в d-мерном пространстве, пусть $\{c1, c2, ..., c_k\}$ обозначает набор из k меток классов, а M - классификатор. Для $\mathbf{x}_i \in \mathbf{D}$ пусть y_i обозначает его истинный класс, а $\hat{y}_i = M(\mathbf{x}_i)$ обозначает его предсказанный класс.

Частота ошибок

Частота ошибок - это доля неверных прогнозов для классификатора на тестовом наборе, определяемая как

$$Error\ Rate = \frac{1}{n}\sum_{i=1}^{n} I(y_i \neq \hat{y}_i) \qquad (16.1)$$

где I - индикаторная функция, которая равна 1, когда ее аргумент истинен, и 0 в противном случае. Частота ошибок - это оценка вероятности ошибочной классификации. Чем ниже частота ошибок, тем лучше классификатор.

Доля правильных ответов

Доля правильных ответов классификатора - это доля правильных прогнозов на тестовом наборе:

$$Accuracy = \frac{1}{n}\sum_{i=1}^{n} I(y_i = \hat{y}_i) = 1 - Error\ Rate \qquad (16.2)$$

Данная оценка дает оценку вероятности правильного прогноза; таким образом, чем выше оценка, тем лучше классификатор.

Пример 16.1. На рисунке 16.1 показан двухмерный набор данных Iris с двумя атрибутами - длиной наружной доли околоцветника и шириной наружной доли околоцветника. Он имеет 150 точек и три класса одинакового размера: Iris-setosa (c_1; круги), Iris-versicolor (c_2; квадраты) и Iris-virginica (c_3; треугольники). Набор данных разделен на обучающий и тестовый наборы в соотношении 80:20. Таким образом, обучающая выборка имеет 120 точек (показано светло-серым), а тестовая выборка **D** имеет n = 30 точек (показано черным). Можно заметить, что, хотя c_1 хорошо отделен от других классов, c_2 и c_3 отделить непросто. Фактически, некоторые точки отмечены как c_2, так и c_3 (например, точка $(6, 2.2)^T$ появляется дважды, обозначенная как c_2 и c_3).

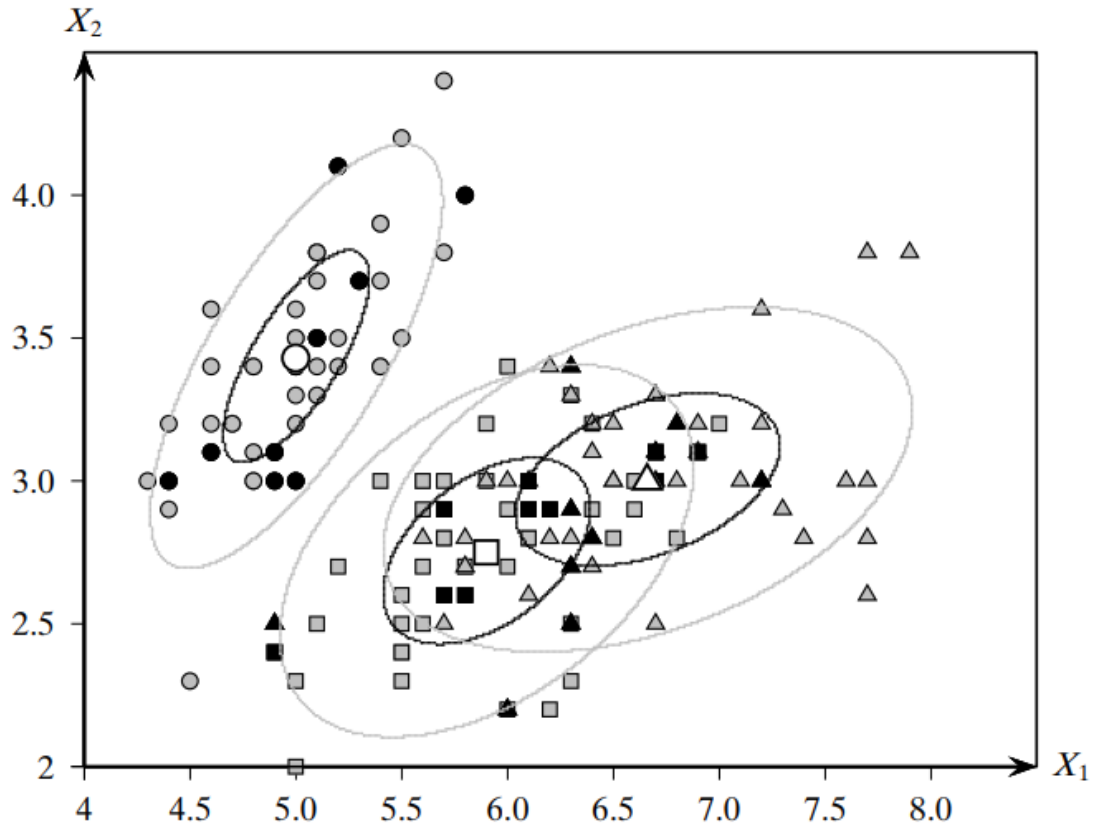


Рисунок 16.1. Набор данных Iris: три класса.

Мы классифицируем контрольные точки, используя полный байесовский классификатор. Каждый класс моделируется с использованием единственного нормального распределения, среднее значение которого (белым цветом) и изолинии плотности (соответствующие одному и двум стандартным отклонениям) также нанесены на рисунок 16.1. Классификатор неправильно классифицирует 8 из 30 тестовых случаев. Таким образом, мы имеем

Error Rate = 8/30 = 0.267

Accuracy = 22/30 = 0.733

16.1.1 Меры основанные на таблице сопряженности

Частота ошибок (и, следовательно, точность) является глобальной мерой, поскольку она явно не учитывает классы, которые оказывают влияние на ошибку. Более

информативные меры могут быть получены путем составления таблицы соответствия и несогласованности конкретных классов между истинными и прогнозируемыми метками по набору тестирования. Пусть $\mathcal{D} = \{\mathbf{D}_1, \mathbf{D}_2, \ldots, \mathbf{D}_k\}$ обозначает разбиение точек тестирования на основе их истинных меток классов, где

$$\mathbf{D}_j = \{\mathbf{x}_i \in \mathbf{D} \mid y_i = c_j\}$$

Пусть $n_i = |\mathbf{D}_i|$ обозначает размер истинного класса c_i.

Пусть $\mathcal{R} = \{\mathbf{R}_1, \mathbf{R}_2, \ldots, \mathbf{R}_k\}$ обозначает разбиение тестовых точек на основе предсказанных меток, то есть

$$\mathbf{R}_j = \{\mathbf{x}_i \in \mathbf{D} \mid \hat{y}_i = c_j\}$$

Пусть $m_j = |\mathbf{R}_j|$ обозначают размер предсказанного класса c_j.

$\mathcal{R}$ и $\mathcal{D}$ создают таблицу сопряженности $\mathbf{N}$ размером $k \times k$, также называемую *матрицей ошибок*, определяемую следующим образом:

$$\mathbf{N}(i,j) = n_{ij} = |\mathbf{R}_i \cap \mathbf{D}_j| = |\{\mathbf{x}_a \in \mathbf{D} \mid \hat{y}_a = c_i \text{ and } y_a = c_j\}|$$

где $1 \leq i, j \leq k$. Счетчик n_{ij} обозначает количество точек с предсказанным классом c_i, истинная метка которого - c_j. Таким образом, n_{ii} (для $1 \leq i \leq k$) обозначает количество случаев, когда классификатор соглашается на истинную метку c_i. Остальные подсчеты n_{ij}, где $i \neq j$ - это случаи, когда классификатор и истинные метки расходятся.

Доля правильных ответов / Точность

Зависящая от класса *доля правильных ответов* или *точность* классификатора M для класса c_i задается как доля правильных прогнозов по всем точкам, которые, как предполагается, находятся в классе c_i.

$$acc_i = \text{prec}_i = \frac{n_{ii}}{m_i}$$

где m_i - количество примеров, предсказанных как c_i классификатором M. Чем выше доля правильных ответов класса c_i, тем лучше классификатор.

Общая точность или доля правильных ответов классификатора - это средневзвешенное значение доли правильных ответов для конкретного класса:

$$\text{A}ccuracy = Precision = \sum_{i=1}^{k} \left(\frac{m_i}{n}\right) acc_i = \frac{1}{n}\sum_{i=1}^{k} n_{ii}$$

Это идентично выражению в формуле (16.2).

Покрытие/Полнота

Зависящее от класса *покрытие* или *полнота* M для класса c_i - это доля правильных прогнозов по всем точкам в классе c_i:

$$coverage_i = recall_i = \frac{n_{ii}}{n_i}$$

где n_i - количество точек в классе c_i. Чем выше покрытие, тем лучше классификатор.

F-мера

Часто существует компромисс между точностью и полнотой классификатора. Например, легко сделать $recall_i = 1$ (полноту), предсказав, что все точки тестирования будут в классе c_i. Однако в этом случае точность ($prec_i$) будет низкой. С другой стороны, мы можем сделать $prec_i$ очень высокой, предсказав только несколько точек как c_i, например, для тех прогнозов, где M имеет наибольшую уверенность, но в этом случае $recall_i$ будет низкой. В идеале мы хотели бы, чтобы точность и полнота были высокими.

Специфическая для класса F-мера пытается сбалансировать значения точности и полноты, вычисляя их среднее гармоническое значение для класса c_i:

$$F_i = \frac{2}{\frac{1}{prec_i} + \frac{1}{recall_i}} = \frac{2 \cdot \text{prec}_i \cdot \text{ recall }_i}{\text{prec }_i + \text{ recall }_i} = \frac{2n_{ii}}{n_i + m_i}$$

Чем выше значение F_i, тем лучше классификатор.

Общая *F-мера* для классификатора M - это среднее значение конкретных классов:

$$F = \frac{1}{k}\sum_{i=1}^{r} F_i$$

Для идеального классификатора максимальное значение F-меры равно 1.

Пример 16.2 Рассмотрим двумерный набор данных Iris, показанный на рисунке 16.1. В Примере 16.1 мы видели, что частота ошибок составила 26,7%. Однако измерение частоты ошибок не дает много информации о классах или экземплярах, которые труднее классифицировать. Из нормального распределения для конкретных классов на рисунке ясно, что классификатор Байеса должен хорошо работать для c_1, но, вероятно, возникнут проблемы с распознаванием некоторых тестовых примеров, которые лежат близко к границе решения между c_2 и c_3. Эта информация лучше отражается в матрице ошибок, полученной на тестовой выборке, как показано в таблице 16.1. Мы можем заметить, что все 10 точек в c_1 классифицированы правильно. Однако только 7 из 10 для c_2 и 5 из 10 для c_3 классифицируются правильно.

Таблица 16.1. Таблица сопряженности для набора данных Iris: тестовый набор

	True			
Predicted	Iris-setosa (c_1)	Iris-versicolor (c_2)	Iris-virginica (c_3)	
Iris-setosa (c_1)	10	0	0	m_1 = 10
Iris-versicolor (c_2)	0	7	5	m_2 = 12

Iris-virginica (c_3)	0	3	5	$m_3 = 8$
	$n_1 = 10$	$n_2 = 10$	$n_3 = 10$	$n = 30$

Из матрицы ошибок мы можем вычислить значения точности (или доли правильных ответов) для конкретного класса:

$$prec_1 = \frac{n_{11}}{m_1} = 10/10 = 1.0$$
$$prec_2 = \frac{n_{22}}{m_2} = 7/12 = 0.583$$
$$prec_3 = \frac{n_{33}}{m_3} = 5/8 = 0.625$$

Общая доля правильных ответов согласуется с данными из примера 16.1:

$$Accuracy = \frac{(n_{11} + n_{22} + n_{33})}{n} = \frac{(10 + 7 + 5)}{30} = 22/30 = 0.733$$

Значения полноты (или покрытия) для конкретного класса даются как

$$recall_1 = \frac{n_{11}}{n_1} = 10/10 = 1.0$$
$$recall_2 = \frac{n_{22}}{n_2} = 7/10 = 0.7$$
$$recall_3 = \frac{n_{33}}{n_3} = 5/10 = 0.5$$

Из них мы можем вычислить значения F-меры для конкретного класса:

$$F_1 = \frac{2 \cdot n_{11}}{(n_1 + m_1)} = 20/20 = 1.0$$
$$F_2 = \frac{2 \cdot n_{22}}{(n_2 + m_2)} = 14/22 = 0.636$$
$$F_3 = \frac{2 \cdot n_{33}}{(n_3 + m_3)} = 10/18 = 0.556$$

Таким образом, общая F-мера для классификатора равна

$$F = \frac{1}{3}(1.0 + 0.636 + 0.556) = \frac{2.192}{3} = 0.731$$

Таблица 16.2. Матрица ошибок для двух классов

	True Class	
Predicted Class	Positive (c_1)	Negative (c_2)
Positive (c_1)	True Positive (*TP*)	False Positive (*FP*)

Negative (c_2)	False Negative (FN)	True Negative (TN)

16.1.2 Бинарная классификация: положительный и отрицательный класс

Когда есть только $k = 2$ классов, мы называем класс c_1 положительным классом, а c_2 - отрицательным классом. Элементам результирующей матрицы ошибок 2 × 2, показанной в таблице 16.2, присвоены следующие специальные имена:

- *Истинно положительные результаты*: количество точек, которые классификатор правильно предсказывает как положительные:

$$TP = n_{11} = |\{\mathbf{x}_i \mid \hat{y}_i = y_i = c_1\}|$$

- *Ложно положительные*: количество точек, которые классификатор предсказывает как положительные, которые на самом деле относятся к отрицательному классу:

$$FP = n_{12} = |\{\mathbf{x}_i \mid \hat{y}_i = c_1 \text{ and } y_i = c_2\}$$

- *Ложно отрицательные*: количество точек, которые классификатор предсказывает как отрицательные, который на самом деле принадлежит к положительному классу:

$$FN = n_{21} = |\{\mathbf{x}_i \mid \hat{y}_i = c_2 \text{ and } y_i = c_1\}|$$

- *Истинно отрицательные*: количество точек, которые классификатор правильно считает отрицательными:

$$TN = n_{22} = |\{\mathbf{x}_i \mid \hat{y}_i = y_i = c_2\}|$$

Частота ошибок

Частота ошибок [уравнение (16.1)] для случая двоичной классификации дается как доля ошибок (или ложных предсказаний):

$$Error\ Rate = \frac{FP + FN}{n}$$

Доля правильных ответов

Доля правильных ответов [уравнение (16.2)] - доля верных прогнозов:

$$Accuracy = \frac{TP + TN}{n}$$

Выше приведены общие показатели эффективности классификатора. Мы можем получить меры, характерные для конкретных классов, следующим образом.

Точность, зависящая от класса

Точность для положительного и отрицательного классов задается как

$$\text{prec}_P = \frac{TP}{TP + FP} = \frac{TP}{m_1}$$
$$\text{prec}_N = \frac{TN}{TN + FN} = \frac{TN}{m_2}$$

где $m_i = |\mathbf{R}_i|$ - количество точек, прогнозируемых M как имеющих класс c_i.

Чувствительность: истинно положительный результат

Истинно положительный показатель, также называемый *чувствительностью*, это доля правильных прогнозов по всем точкам в положительном классе, то есть это просто полнота для положительного класса

$$TPR = recall_P = \frac{TP}{TP + FN} = \frac{TP}{n_1}$$

где n_1 - размер положительного класса.

Специфичность: истинно отрицательный результат

Истинный отрицательный результат, также называемый *специфичностью* - это просто полнота для отрицательного класса:

$$TNR = \ specificity \ = recall_N = \frac{TN}{FP + TN} = \frac{TN}{n_2}$$

где n_2 - размер отрицательного класса.

Ложно отрицательный результат

Ложно отрицательный результат определяется как

$$FPR = \frac{FP}{FP + TN} = \frac{FP}{n_2} = 1 - \ specificity$$

Пример 16.3. Рассмотрим набор данных Iris, спроецированный на его первые две главные компоненты, как показано на рисунке 16.2. Задача состоит в том, чтобы отделить Ирис-разноцветный (класс c_1; кружками) от двух других Ирисов (класс c_2; в треугольниках). Точки из класса c_1 лежат между точками из класса c_2, что затрудняет (линейную) классификацию. Набор данных был случайным образом разделен на 80% точек обучения (серым цветом) и 20% точек тестирования (черным цветом). Таким образом, в обучающей выборке 120 точек, а в тестовой $n = 30$ точек.

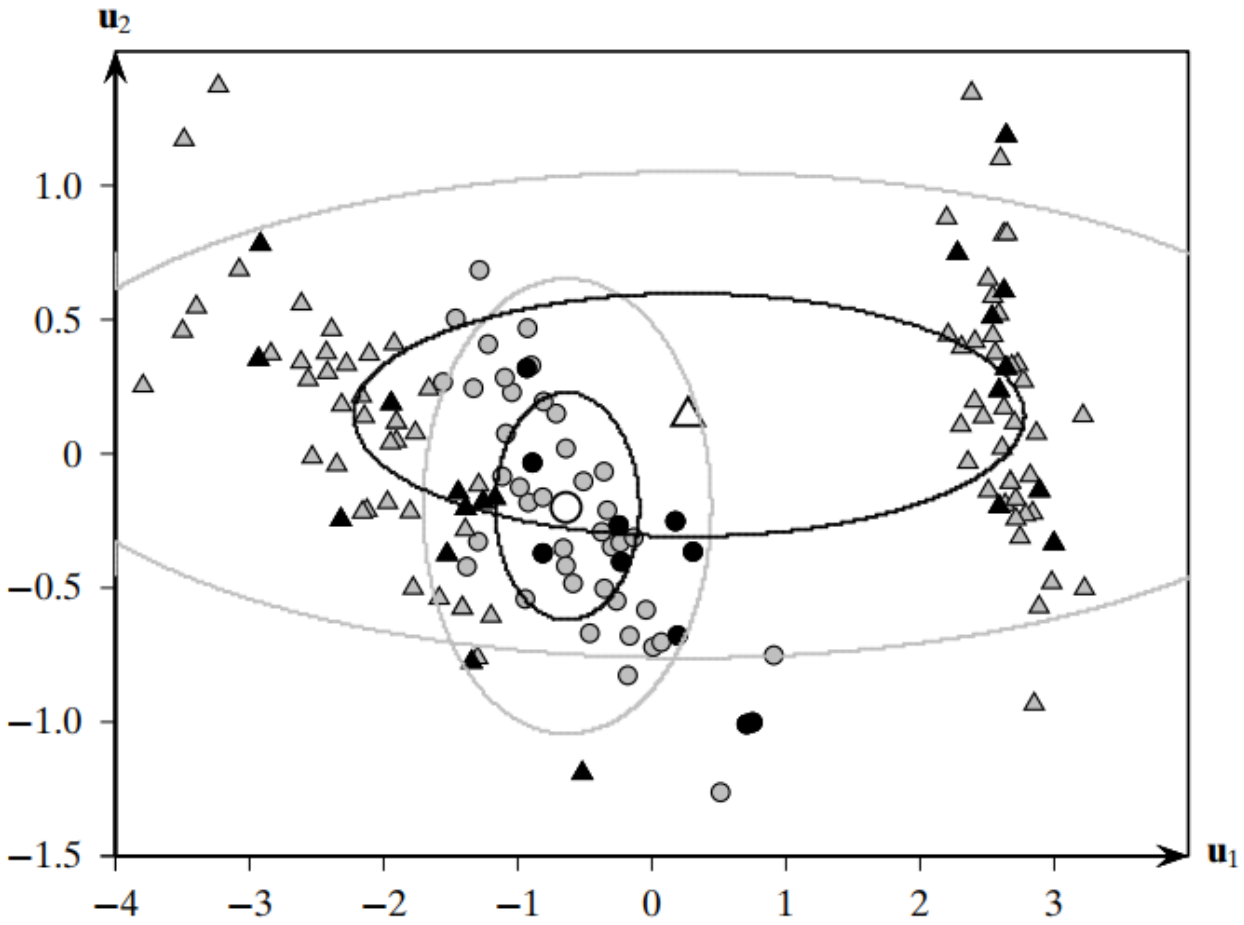


Рисунок 16.2. Главные компоненты набора данных Iris: тренировочный и тестировочный наборы

Применение наивного байесовского классификатора (с одной нормалью на класс) к обучающей выборке дает следующие оценки для среднего, ковариационной матрицы и априорной вероятности для каждого класса:

$$\hat{P}(c_1) = 40/120 = 0.33 \qquad \hat{P}(c_2) = 80/120 = 0.67$$
$$\hat{\mu}_1 = (-0.641 \quad -0.204)^T \qquad \hat{\mu}_2 = (0.27 \quad 0.14)^T$$
$$\hat{\boldsymbol{\Sigma}}_1 = \begin{pmatrix} 0.29 & 0 \\ 0 & 0.18 \end{pmatrix} \qquad \hat{\boldsymbol{\Sigma}}_2 = \begin{pmatrix} 6.14 & 0 \\ 0 & 0.206 \end{pmatrix}$$

Среднее (белым цветом) и контурный график нормального распределения для каждого класса также показаны на рисунке; контуры показаны для одного и двух стандартных отклонений по каждой оси.

Для каждой из 30 тестовых точек мы классифицируем их, используя приведенные выше оценки параметров (см. главу 18). Наивный байесовский классификатор неправильно классифицировал 10 из 30 тестовых примеров, в результате частота ошибок и доля правильных ответов

$$\textit{Error Rate} = 10/30 = 0.33$$
$$\textit{Accuracy} = 20/30 = 0.67$$

Матрица ошибок для этой задачи двоичной классификации показана в таблице 16.3. Из этой таблицы мы можем вычислить различные показатели производительности:

$$prec_P = \frac{TP}{TP + FP} = \frac{7}{14} = 0.5$$

Таблица 16.3. Главные компоненты набора данных Iris: таблица сопряженности для двоичной классификации

	True		
Predicted	Positive (c_1)	Negative (c_2)	
Positive (c_1)	$TP = 7$	$FP = 7$	$m_1 = 14$
Negative (c_2)	$FN = 3$	$TN = 13$	$m_2 = 16$
	$n_1 = 10$	$n_2 = 20$	$n = 30$

$$prec_N = \frac{TN}{TN + FN} = \frac{13}{16} = 0.8125$$

$$recall_P = sensitivity = TPR = \frac{TP}{TP + FN} = \frac{7}{10} = 0.7$$

$$recall_N = specificity = TNR = \frac{TN}{TN + FP} = \frac{13}{20} = 0.65$$

$$FNR = 1 - sensitivity = 1 - 0.7 = 0.3$$

$$FPR = 1 - specificity = 1 - 0.65 = 0.35$$

Мы можем заметить, что точность для положительного класса довольно низкая. Истинно положительный результат также невелик, ложно положительный уровень относительно высок. Таким образом, наивный байесовский классификатор не особенно эффективен для этого тестового набора данных.

16.1.3 ROC анализ

Рабочая характеристика приемника (англ. receiver operating characteristic - ROC) - это популярная стратегия для оценки производительности классификаторов, в случаях с двумя классами. ROC-анализ требует, чтобы классификатор выводил значение оценки для положительного класса для каждой точки в тестовом наборе. Затем эти оценки можно использовать для сортировки точек в порядке убывания. Мы можем использовать апостериорную вероятность $P(c_1|\mathbf{x}_i)$ в качестве оценки, например, для байесовских классификаторов. Для классификаторов SVM в качестве оценки мы можем использовать расстояние со знаком от гиперплоскости, потому что большие положительные расстояния являются предсказаниями с высокой степенью достоверности для c_1, а большие отрицательные расстояния - предсказаниями с очень низкой степенью достоверности для c_1 (на самом деле они являются предсказаниями с высокой степенью достоверности для отрицательного класса c_2).

Как правило, бинарный классификатор выбирает некоторый порог положительной оценки ρ и классифицирует все точки с оценкой выше ρ как положительные, а остальные баллы классифицируются как отрицательные. Однако такой порог, вероятно, будет несколько произвольным. Вместо этого ROC-анализ строит график производительности классификатора по всем возможным значениям порогового параметра ρ. В частности, для каждого значения ρ он отображает частоту ложно положительных результатов (специфичность 1) на оси абсцисс в сравнении с частотой истинных положительных результатов (чувствительность) на оси ординат. Полученный график называется *кривой ROC* или *графиком ROC* для классификатора.

Пусть $S(\mathbf{x}_i)$ обозначает действительную оценку положительного результата класса, полученного классификатором M для точки $\mathbf{x}_i$. Пусть максимальный и минимальный пороги оценки, полученные в тестовом наборе данных **D**, будут следующими:

$$\rho^{min} = \min_i\{S(\mathbf{x}_i)\} \qquad\qquad \rho^{max} = \max_i\{S(\mathbf{x}_i)\}$$

Таблица 16.4. Различные случаи для матрица ошибок размером 2x2.

	True	
Predicted	Pos	Neg
Pos	0	0
Neg	*FN*	*TN*

(a) Initial: all negative

	True	
Predicted	Pos	Neg
Pos	*TP*	*FP*
Neg	0	0

(b) Final: all positive

	True	
Predicted	Pos	Neg
Pos	*TP*	0
Neg	0	*TN*

(c) Ideal classifier

(a) Исходная: все отрицательные

(b) Конечная: все положительные

(c) Идельный классификатор

Изначально мы классифицируем все точки как отрицательные. Таким образом, как *TP*, так и *FP* изначально равны нулю (как показано в таблице 16.4a), в результате чего коэффициенты *TPR* и *FPR* равны нулю, что соответствует точке (0,0) в нижнем левом углу графика ROC. Затем для каждого отдельного значения ρ в диапазоне $[\rho^{\min}, \rho^{\max}]$ заносим в таблицу набор положительных точек:

$$\mathbf{R}_1(\rho) = \{\mathbf{x}_i \in \mathbf{D}: S(\mathbf{x}_i) > \rho\}$$

и вычисляем соответствующие истинные и ложные положительные результаты, чтобы получить новую точку на графике ROC. Наконец, на последнем шаге мы классифицируем все точки как положительные. Таким образом, *FN* и *TN* равны нулю (как показано в таблице 16.4b), в результате чего значения *TPR* и *FPR* равны 1. Получаем точку (1,1) в правом верхнем углу графика ROC. Идеальный классификатор соответствует левой верхней точке (0,1), что соответствует случаю $FPR = 0$ и $TPR = 1$, то есть классификатор не имеет ложных положительных результатов и идентифицирует все истинные результаты (как следствие, он также правильно предсказывает все точки в отрицательном классе). Этот случай рассмотрен в таблице 16.4c. Таким образом, кривая ROC указывает уровень, по которому классификатор ранжирует положительные экземпляры выше, чем отрицательные экземпляры. Идеальный классификатор должен считать все положительные точки выше, любой отрицательной. Таким образом, классификатор с кривой расположенной ближе к идеальной, то есть ближе к левому верхнему углу, является лучшим.

Площадь под кривой ROC

Площадь под кривой ROC, сокращенно обозначается AUC (англ. area under ROC curve), может использоваться как мера производительности классификатора. Поскольку общая площадь графика равна 1, AUC лежит в интервале [0,1] - чем выше, тем лучше. Значение AUC - это, по сути, вероятность того, что классификатор оценит случайный положительный тестовый случай выше, чем случайный отрицательный тестовый случай.

Алгоритм ROC/AUC

Алгоритм 16.1 показывает этапы построения кривой ROC и вычисления площади под кривой. Он принимает в качестве входных данных тестовый набор **D** и классификатор *M*. Первый шаг - спрогнозировать оценку $S(\mathbf{x}_i)$ для положительного класса (c_1) для каждой тестовой точки $\mathbf{x}_i \in \mathbf{D}$. Затем мы сортируем ($S(\mathbf{x}_i)$, y_i) пары, то есть пары оценки и истинных классов, в порядке убывания оценок (строка 3). Первоначально мы устанавливаем положительный порог оценки $\rho = \infty$ (строка 7). Цикл for (строка 8) проверяет каждую пару ($S(\mathbf{x}_i)$, y_i) в отсортированном порядке, и для каждого отдельного значения оценки устанавливает $\rho = S(\mathbf{x}_i)$ и строит точку

$$(FPR, TPR) = \left(\frac{FP}{n_2}, \frac{TP}{n_1}\right)$$

По мере проверки каждой контрольной точки истинные и ложные положительные значения корректируются на основе истинного класса y_i для контрольной точки $\mathbf{x}_i$. Если $y_1 = c_1$, мы увеличиваем истинно положительный результат, в противном случае мы увеличиваем ложно положительный результат (строки 15-16). В конце цикла for мы наносим конечную точку кривой ROC (строка 17).

Алгоритм 16.1: Алгоритм ROC/AUC

ROC-CURVE(D, M):

$n_1 \leftarrow \left|\{\mathbf{x}_i \in \mathbf{D} \,|\, y_i = c_1\}\right|$ // size of positive class

$n_2 \leftarrow \left|\{\mathbf{x}_i \in \mathbf{D} \,|\, y_i = c_2\}\right|$ // size of negative class

// classify, score, and sort all test points

$L \leftarrow$ sort the set $\{(S(\mathbf{x}_i), y_i) : \mathbf{x}_i \in \mathbf{D}\}$ by decreasing scores

$FP \leftarrow TP \leftarrow 0$

$FP_{prev} \leftarrow TP_{prev} \leftarrow 0$

$AUC \leftarrow 0$

$\rho \leftarrow \infty$

foreach $(S(\mathbf{x}_i), y_i) \in L$ **do**

  if $\rho > S(\mathbf{x}_i)$ **then**

    plot point $\left(\frac{FP}{n_2}, \frac{TP}{n_1}\right)$

    $AUC \leftarrow AUC + \text{TRAPEZOID-AREA}\left(\left(\frac{FP_{prev}}{n_2}, \frac{TP_{prev}}{n_1}\right), \left(\frac{FP}{n_2}, \frac{TP}{n_1}\right)\right)$

    $\rho \leftarrow S(\mathbf{x}_i)$

    $FP_{prev} \leftarrow FP$

    $TP_{prev} \leftarrow TP$

  if $y_i = c_1$ **then** $TP \leftarrow TP + 1$

  else $FP \leftarrow FP + 1$

plot point $\left(\frac{FP}{n_2}, \frac{TP}{n_1}\right)$

$AUC \leftarrow AUC + \text{TRAPEZOID-AREA}\left(\left(\frac{FP_{prev}}{n_2}, \frac{TP_{prev}}{n_1}\right), \left(\frac{FP}{n_2}, \frac{TP}{n_1}\right)\right)$

TRAPEZOID-AREA$((x_1, y_1), (x_2, y_2))$:

$b \leftarrow |x_2 - x_1|$ // base of trapezoid

$h \leftarrow \frac{1}{2}(y_2 + y_1)$ // average height of trapezoid

return $(b \cdot h)$

Значение AUC вычисляется по мере добавления каждой новой точки на график ROC. Алгоритм сохраняет предыдущие значения ложно и истинно положительных результатов, FP_{prev} и TP_{prev}, для предыдущего порога оценки ρ. Учитывая текущие значения FP и TP, мы вычисляем площадь под кривой, определяемой четырьмя точками

$$(x_1, y_1) = \left(\frac{FP_{\text{prev}}}{n_2}, \frac{TP_{\text{prev}}}{n_1}\right) \qquad (x_2, y_2) = \left(\frac{FP}{n_2}, \frac{TP}{n_1}\right)$$

$$(x_1, 0) = \left(\frac{FP_{\text{prev}}}{n_2}, 0\right) \qquad (x_2, 0) = \left(\frac{FP}{n_2}, 0\right)$$

Эти четыре точки определяют трапецию, если $x_2 > x_1$ и $y_2 > y_1$, в противном случае они определяют прямоугольник (который может быть вырожденным с нулевой площадью). Функция TRAPEZOID-AREA вычисляет площадь под трапецией, которая задается как $b \cdot h$,

Таблица 16.5. Отсортированные оценки и истинные классы

$S(\mathbf{x}_i)$	0.93	0.82	0.80	0.77	0.74	0.71	0.69	0.67	0.66	0.61
y_i	c_2	c_1	c_2	c_1	c_1	c_1	c_2	c_1	c_2	c_2

$S(\mathbf{x}_i)$	0.59	0.55	0.55	0.53	0.47	0.30	0.26	0.11	0.04	2.97e-03
y_i	c_2	c_2	c_1	c_1	c_1	c_1	c_1	c_2	c_2	c_2

$S(\mathbf{x}_i)$	1.28e-03	2.55e-07	6.99e-08	3.11e-08	3.109e-08
y_i	c_2	c_2	c_2	c_2	c_2

$S(\mathbf{x}_i)$	1.53e-08	9.76e-09	2.08e-09	1.95e-09	7.83e-10
y_i	c_2	c_2	c_2	c_2	c_2

где $b = |x_2 - x_1|$ - длина основания трапеции, а $h = \frac{1}{2}(y_1 + y_2)$ - средняя высота трапеции.

Пример 16.4. Рассмотрим задачу двоичной классификации из примера 16.3 для главных компонент набора данных Iris. Тестовый набор данных **D** содержит $n = 30$ точек, из которых $n_1 = 10$ точек в положительном классе и $n_2 = 20$ точек в отрицательном классе.

Для вычисления вероятности того, что каждая контрольная точка принадлежит положительному классу (c_1; iris-versicolor) используется наивный байесовский классификатор. Таким образом, оценка классификатора для контрольной точки $\mathbf{x}_i$ равна $S(\mathbf{x}_i) = P(c_1|\mathbf{x}_i)$. Отсортированные оценки (в порядке убывания) вместе с истинными обозначениями классов показаны в таблице 16.5.

Кривая ROC для тестового набора данных показана на рисунке 16.3. Рассмотрим положительный порог оценки $\rho = 0{,}71$. Если мы классифицируем все точки с оценкой выше этого значения как положительные, то у нас будет следующее количество истинно и ложно положительных результатов:

$TP = 3$ $\qquad\qquad$ $FP = 2$

Следовательно, частота ложно положительных результатов равна $\frac{FP}{n_2} = \frac{2}{20} = 0.1$, а частота истинно положительных результатов равна $\frac{TP}{n_1} = 3/10 = 0.3$. Это соответствует точке (0.1, 0.3) на кривой ROC. Другие точки на кривой ROC получаются аналогично тому, как показано на рисунке 16.3. Общая площадь под кривой равна 0.775.

Пример 16.5 (AUC). Чтобы понять, почему нам нужно рассматривать трапеции при вычислении AUC, рассмотрим следующие отсортированные оценки вместе с истинным классом для некоторого тестового набора данных с $n = 5$, $n_1 = 3$ и $n_2 = 2$.

$(0.9,c_1),(0.8,c_2),(0.8,c_1),(0.8,c_1),(0.1,c_2)$

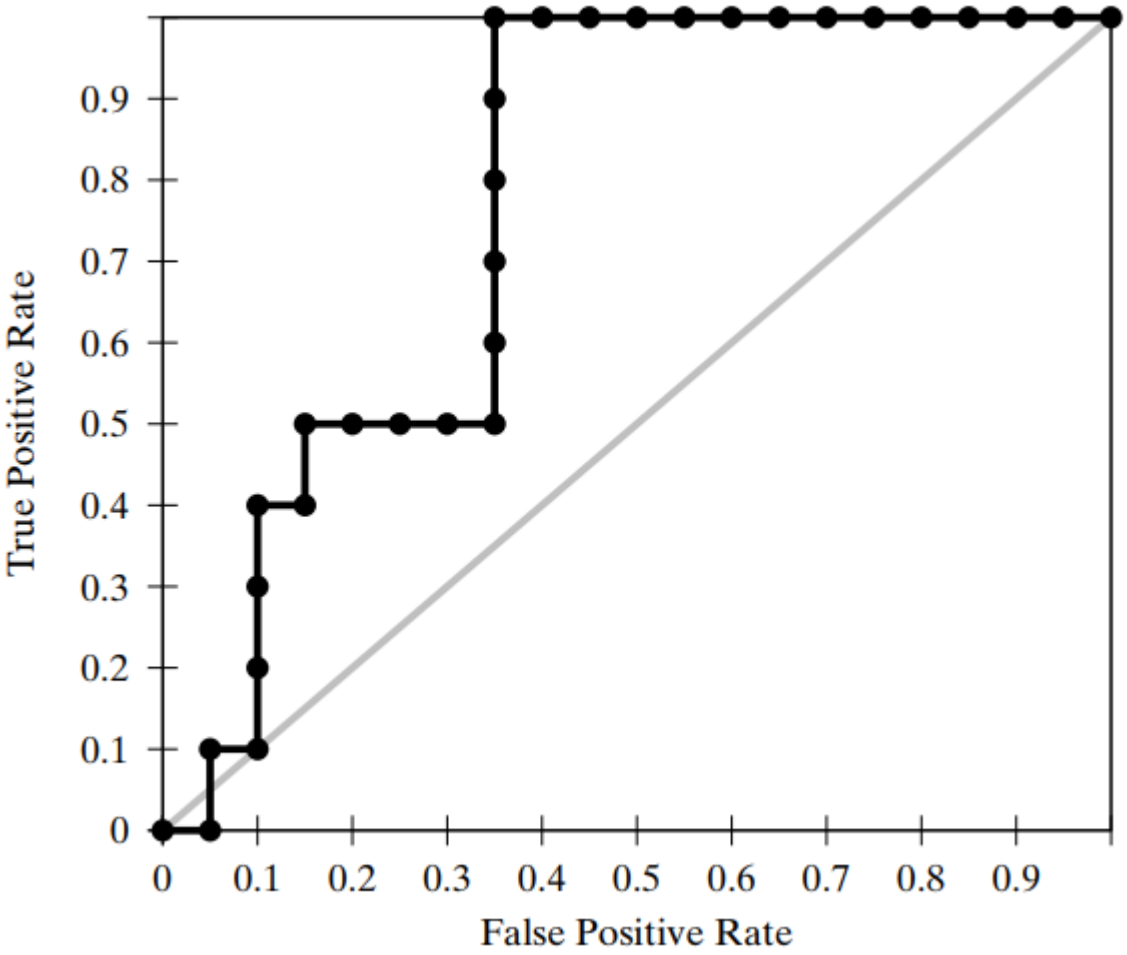


Рисунок 16.3. График ROC для главных компонент набора данных Iris. Показаны кривые ROC для наивного байесовского (черный) и случайного (серый) классификаторов.

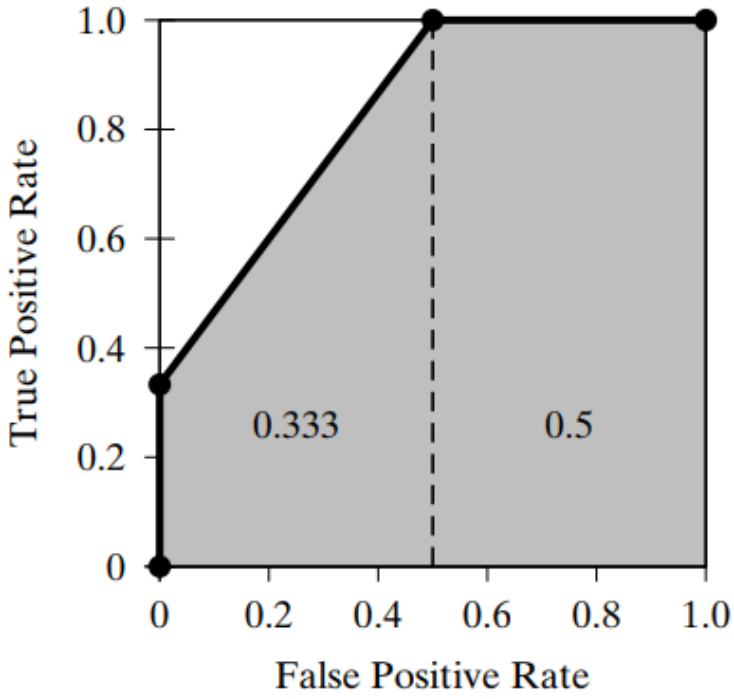


Рисунок 16.4. График ROC и AUC: область трапеции.

Алгоритм 16.1 дает следующие точки, которые добавляются к графику ROC вместе с текущим AUC:

ρ	FP	TP	(FPR, TPR)	AUC
∞	0	0	(0, 0)	0
0.9	0	1	(0, 0.333)	0
0.8	1	3	(0.5, 1)	0.333
0.1	2	3	(1, 1)	0.833

На рисунке 16.4 показан график ROC с заштрихованной областью, представляющей AUC. Можно заметить, что трапеция получается, когда есть хотя бы одна положительная и одна отрицательная точка с одинаковой оценкой. Общая AUC равна 0,833, и вычисляется как сумма трапециевидной области слева (0.333) и прямоугольной области справа (0.5).

Случайный классификатор

Интересно отметить, что случайный классификатор соответствует диагональной линии на графике ROC. Чтобы увидеть это, подумайте о классификаторе, который случайным образом угадывает класс точки как положительный в половине случаев и отрицательный в другой половине. Затем мы ожидаем, что половина истинно положительных и истинно отрицательных результатов будет определена правильно, определяя точку (TPR, FPR) = (0.5,0.5) для графика ROC. С другой стороны, если классификатор угадывает класс точки как положительный в 90% случаев и как отрицательный в 10% случаев, то мы ожидаем, что 90% истинно положительных результатов и 10% истинно отрицательных результатов будут помечены. правильно, в результате чего $TPR = 0.9$ и $FPR = 1 - TNR = 1 - 0.1 = = 0.9$, то есть мы получаем точку (0.9,0.9) на графике ROC. В общем, любая фиксированная вероятность предсказания, скажем r, для положительного класса, дает точку (r, r) в пространстве ROC. Диагональная линия, таким образом, представляет производительность случайного классификатора по всем возможным порогам прогнозирования положительного класса r. Из этого следует, что если кривая ROC для любого классификатора ниже диагонали, это указывает на худшую производительность, чем у случайного предположения. В таких случаях инвертирование присвоения классов дает лучший классификатор. Как следствие диагональной кривой ROC, значение AUC для случайного классификатора составляет 0.5. Таким образом, если какой-либо классификатор имеет значение AUC менее 0.5, это указывает на производительность хуже случайной.

Пример 16.6. В дополнение к кривой ROC для наивного байесовского классификатора на рис. 16.3 также показан график ROC для случайного классификатора (диагональная линия серого цвета). Мы видим, что кривая ROC для наивного байесовского классификатора намного лучше случайного. Его значение AUC равно 0.775, что намного лучше, чем значение AUC равное 0.5 для случайного классификатора. Однако в самом начале наивный байесовский классификатор работает хуже, чем случайный классификатор, потому что точка с самой высокой оценкой приходится на отрицательный класс. Таким образом, кривую ROC следует рассматривать как дискретную аппроксимацию гладкой

кривой, которая была бы получена для очень большого (бесконечного) тестового набора данных.

Дисбаланс классов

Стоит отметить, что кривые ROC нечувствительны к несбалансированным классам. Это связано с тем, что TPR, интерпретируемый как вероятность предсказания положительной точки как положительной, и FPR, интерпретируемый как вероятность предсказания отрицательной точки как положительной, не зависят от соотношения размеров положительного и отрицательного классов. Это желательное свойство, поскольку кривая ROC по существу останется той же самой, независимо от того, сбалансированы ли классы (имеют относительно одинаковое количество точек) или несбалансированны (когда один класс имеет намного больше точек, чем другой).

16.2 Оценка классификаторов

В этом разделе мы обсудим, как оценить классификатор M с помощью некоторой меры производительности θ. Как правило, входной набор данных $\mathbf{D}$ случайным образом разбивается на непересекающиеся обучающий набор и набор для тестирования. Обучающий набор используется для изучения модели M, а тестовый набор используется для оценки меры θ. Однако насколько мы можем быть уверены в эффективности классификации? Результаты могут быть связаны с артефактом случайного разбиения, например, случайным образом, в наборе тестирования может быть особенно легко (или сложно) классифицировать точки, что приводит к хорошей (или плохой) производительности классификатора. Таким образом, фиксированное заранее определенное разделение набора данных не является хорошей стратегией для оценки классификаторов. Также обратите внимание, что, в общем, $\mathbf{D}$ сама по себе является d-мерной многомерной случайной выборкой, взятой из истинной (неизвестной) совместной функции плотности вероятности $f(\mathbf{x})$, которая представляет интересующую совокупность. В идеале мы хотели бы знать ожидаемое значение $E[\theta]$ показателя производительности для всех возможных наборов тестирования, взятых из f. Однако, поскольку f неизвестно, мы должны оценить $E[\theta]$ из $\mathbf{D}$. Перекрестная проверка и повторная выборка – два распространенных подхода к вычислению ожидаемого значения и дисперсии данного показателя производительности; мы обсудим эти методы в следующих разделах.

16.2.1 K-кратная перекрестная проверка

Перекрестная проверка делит набор данных $\mathbf{D}$ на K частей равного размера, называемых свертками, а именно $\mathbf{D}_1, \mathbf{D}_2, \ldots, \mathbf{D}_K$. Каждый сверток $\mathbf{D}_i$, в свою очередь, рассматривается как набор для тестирования, а остальные свертки составляют набор для обучения $\mathbf{D} \setminus \mathbf{D}_i = \bigcup_{j \neq i} \mathbf{D}_j$. После обучения модели M_i на $\mathbf{D} \setminus \mathbf{D}_i$ мы оцениваем ее эффективность на множестве тестирования $\mathbf{D}_i$, чтобы получить i-ю оценку θ_i. Ожидаемое значение показателя эффективности затем можно оценить как

$$\hat{\mu}_\theta = E[\theta] = \frac{1}{K} \sum_{i=1}^{K} \theta_i, \tag{16.3}$$

и его дисперсия как

$$\hat{\sigma}_\theta^2 = \frac{1}{K} \sum_{i=1}^{K} (\theta_i - \hat{\mu}_\theta)^2, \tag{16.4}$$

Алгоритм 16.2 показывает псевдокод для K-кратной перекрестной проверки. После случайного перемешивания набора данных $\mathbf{D}$ мы разделяем его на K равных сверток (кроме, возможно, последней). Затем каждая сверток $\mathbf{D}_i$ используется в качестве набора для тестирования, на котором мы оцениваем производительность θ_i классификатора M_i, обученного на $\mathbf{D} \setminus \mathbf{D}_i$. Затем можно оценить среднее значение и дисперсию θ. Обратите внимание, что K-кратная перекрестная проверка может повторяться несколько раз; начальная случайная перетасовка гарантирует, что свертки каждый раз будут разными.

Обычно K выбирается равным 5 или 10. Особый случай, когда $K = n$, называется перекрестной проверкой с исключением по одному, когда набор тестирования состоит из одной точки, а остальные данные используются для целей обучения.

Алгоритм 16.2: К-кратная перекрестная проверка

CROSS-VALIDATION(K, $\mathbf{D}$):
1 $\mathbf{D} \leftarrow$ randomly shuffle $\mathbf{D}$
2 $\{\mathbf{D}_1, \mathbf{D}_2, \ldots, \mathbf{D}_K\} \leftarrow$ partition $\mathbf{D}$ in K equal parts
3 **foreach** $i \in [1, K]$ **do**
4 $\quad M_i \leftarrow$ train classifier on $\mathbf{D} \setminus \mathbf{D}_i$
5 $\quad \theta_i \leftarrow$ assess M_i on $\mathbf{D}_i$
6 $\hat{\mu}_\theta = \frac{1}{K} \sum_{i=1}^{K} \theta_i$
7 $\hat{\sigma}_\theta^2 = \frac{1}{K} \sum_{i=1}^{K} (\theta_i - \hat{\mu}_\theta)^2$
8 **return** $\hat{\mu}_\theta, \hat{\sigma}_\theta^2$

16.2.2 Повторная выборка начальной загрузки

Другой подход к оценке ожидаемой производительности классификатора - использование метода повторной выборки начальной загрузки. Вместо того, чтобы разбивать входной набор данных $\mathbf{D}$ на непересекающиеся свертки, метод начальной загрузки рисует K случайных выборок размера n с заменой из $\mathbf{D}$. Таким образом, каждая выборка $\mathbf{D}_i$ имеет тот же размер, что и $\mathbf{D}$, и имеет несколько повторяющихся точек. Рассмотрим вероятность того, что точка $\mathbf{x}_j \in \mathbf{D}$ не выбрана для i-го образца $\mathbf{D}_i$. Из-за выборки с заменой вероятность того, что данная точка выбрана, дается как $p = \frac{1}{n}$, и, таким образом, вероятность того, что она не выбрана, равна

$$q = 1 - p = 1 - \frac{1}{n}$$

Поскольку $\mathbf{D}_i$ имеет n точек, вероятность того, что $\mathbf{x}_j$ не будет выбрана даже после n попыток, задается как

$$P(\mathbf{x}_j \notin \mathbf{D}_i) = q^n = \left(1 - \frac{1}{n}\right)^n \simeq e^{-1} = 0.368$$

Алгоритм 16.3: Повторная выборка начальной загрузки

BOOTSTRAP-RESAMPLING(K, D):

1 **for** $i \in [1, K]$ **do**
2 $\quad \mathbf{D}_i \leftarrow$ sample of size n with replacement from $\mathbf{D}$
3 $\quad M_i \leftarrow$ train classifier on $\mathbf{D}_i$
4 $\quad \theta_i \leftarrow$ assess M_i on $\mathbf{D}$
5 $\hat{\mu}_\theta = \frac{1}{K}\sum_{i=1}^{K} \theta_i$
6 $\hat{\sigma}_\theta^2 = \frac{1}{K}\sum_{i=1}^{K} (\theta_i - \hat{\mu}_\theta)^2$
7 **return** $\hat{\mu}_\theta, \hat{\sigma}_\theta^2$

С другой стороны, вероятность того, что $\mathbf{x}_j \in \mathbf{D}$, задается как

$$P(\mathbf{x}_j \in \mathbf{D}_i) = 1 - P(\mathbf{x}_j \notin \mathbf{D}_i) = 1 - 0.368 = 0.632$$

Это означает, что каждая выборка начальной загрузки содержит примерно 63,2% точек из **D**.

Образцы начальной загрузки можно использовать для оценки классификатора путем обучения его на каждой из выборок $\mathbf{D}_i$ и последующего использования полного набора входных данных **D** в качестве набора для тестирования, как показано в алгоритме 16.3. Ожидаемое значение и дисперсия показателя эффективности θ могут быть получены с помощью формул (16.3) и (16.4). Однако следует иметь в виду, что оценки будут несколько оптимистичными из-за довольно большого совпадения наборов данных для обучения и тестирования (63,2%). Подход с перекрестной проверкой не страдает от этого ограничения, потому что он сохраняет несовпадающие наборы для обучения и тестирования.

16.2.3 Доверительные интервалы

Оценив ожидаемое значение и дисперсию для выбранного показателя производительности, мы хотели бы получить доверительные границы того, насколько оценка может отклоняться от истинного значения.

Чтобы ответить на этот вопрос, мы используем центральную предельную теорему, которая утверждает, что сумма большого числа независимых и одинаково распределенных (IID) случайных величин имеет приблизительно нормальное распределение, независимо от распределения отдельных случайных величин. Более формально, пусть $\theta_1, \theta_2, \ldots, \theta_K$ будет последовательностью случайных величин IID, представляющих, например, частоту ошибок или некоторую другую меру производительности по K-кратным значениям в выборках перекрестной проверки или K выборок загрузки. Предположим, что каждое θ_i имеет конечное среднее $E[\theta_i] = \mu$ и конечную дисперсию $var(\theta_i) = \sigma^2$.

Пусть $\hat{\mu}$ обозначает среднее выборки:

$$\hat{\mu} = \frac{1}{K}(\theta_1 + \theta_2 + \cdots + \theta_K)$$

По линейности ожидания имеем

$$E[\hat{\mu}] = E\left[\frac{1}{K}(\theta_1 + \theta_2 + \cdots + \theta_K)\right] = \frac{1}{K}\sum_{i=1}^{K} E[\theta_i] = \frac{1}{K}(K\mu) = \mu$$

Используя линейность дисперсии для независимых случайных величин и отмечая, что $\text{var}(aX) = a^2$ для $a \in \mathbb{R}$, дисперсия $\hat{\mu}$ задается как

$$\mathrm{var}(\hat{\mu}) = \mathrm{var}\left(\frac{1}{K}(\theta_1 + \theta_2 + \cdots + \theta_K)\right) = \frac{1}{K^2}\sum_{i=1}^{K}\mathrm{var}(\theta_i) = \frac{1}{K^2}(K\sigma^2) = \frac{\sigma^2}{K}$$

Таким образом, стандартное отклонение $\hat{\mu}$ определяется как

$$\mathrm{std}(\hat{\mu}) = \sqrt{\mathrm{var}(\hat{\mu})} = \frac{\sigma}{\sqrt{K}}$$

Нас интересует распределение z-показателя $\hat{\mu}$, которое само по себе является случайной величиной.

$$Z_K = \frac{\hat{\mu} - E[\hat{\mu}]}{\mathrm{std}(\hat{\mu})} = \frac{\hat{\mu} - \mu}{\frac{\sigma}{\sqrt{K}}} = \sqrt{K}\left(\frac{\hat{\mu} - \mu}{\sigma}\right)$$

Z_K определяет отклонение оценочного среднего от истинного среднего в терминах его стандартного отклонения. Центральная предельная теорема утверждает, что по мере увеличения размера выборки случайная величина Z_K сходится по распределению к стандартному нормальному распределению (которое имеет среднее значение 0 и дисперсию 1). То есть при $K \to \infty$ для любого x $\in \mathbb{R}$ имеем

$$\lim_{K\to\infty} P(Z_K \le x) = \Phi(x)$$

где $\Phi(x)$ – кумулятивная функция распределения для стандартной нормальной функции плотности f (x | 0,1). Пусть $z_{\alpha/2}$ обозначает значение z-показателя, которое включает $\alpha/2$ вероятностной массы для стандартного нормального распределения, то есть

$$P\left(0 \le Z_K \le z_{\alpha/2}\right) = \Phi\left(z_{\alpha/2}\right) - \Phi(0) = \alpha/2$$

тогда, поскольку нормальное распределение симметрично относительно среднего, мы имеем

$$\lim_{K\to\infty} P\left(-z_{\alpha/2} \le Z_K \le z_{\alpha/2}\right) = 2 \cdot P\left(0 \le Z_K \le z_{\alpha/2}\right) = \alpha, \qquad (16.5)$$

Обратите внимание, что

$$\begin{aligned} -z_{\frac{\alpha}{2}} \le Z_K \le z_{\frac{\alpha}{2}} &\Longrightarrow -z_{\frac{\alpha}{2}} \le \sqrt{K}\left(\frac{\hat{\mu} - \mu}{\sigma}\right) \le z_{\frac{\alpha}{2}} \\ &\Longrightarrow -z_{\frac{\alpha}{2}}\frac{\sigma}{\sqrt{K}} \le \hat{\mu} - \mu \le z_{\frac{\alpha}{2}}\frac{\sigma}{\sqrt{K}} \\ &\Longrightarrow \left(\hat{\mu} - z_{\frac{\alpha}{2}}\frac{\sigma}{\sqrt{K}}\right) \le \mu \le \left(\hat{\mu} + z_{\frac{\alpha}{2}}\frac{\sigma}{\sqrt{K}}\right) \end{aligned}$$

Подставляя вышеуказанное в формулу (16.5), получаем оценки истинного среднего μ через оценочное значение $\hat{\mu}$, т. е.

$$\lim_{K\to\infty} P\left(\hat{\mu} - z_{\alpha/2}\frac{\sigma}{\sqrt{K}} \le \mu \le \hat{\mu} + z_{\alpha/2}\frac{\sigma}{\sqrt{K}}\right) = \alpha \qquad (16.6)$$

Таким образом, для любого заданного уровня достоверности α мы можем вычислить вероятность того, что истинное среднее значение μ лежит в α% доверительном интервале $\left(\hat{\mu} - z_{\frac{\alpha}{2}}\frac{\sigma}{\sqrt{K}}, \hat{\mu} + z_{\alpha/2}\frac{\sigma}{\sqrt{K}}\right)$. Другими словами, даже если мы не знаем истинного среднего значения μ, мы можем получить уверенную оценку интервала, в котором оно должно находиться (например, установив α = 0,95 или α = 0,99).

Неизвестная дисперсия

Приведенный выше анализ предполагает, что нам известна истинная дисперсия σ^2, что, как правило, не так. Однако мы можем заменить σ^2 выборочной дисперсией

$$\hat{\sigma}^2 = \frac{1}{K}\sum_{i=1}^{K} (\theta_i - \hat{\mu})^2 \tag{16.7}$$

поскольку $\hat{\sigma}^2$ является последовательной оценкой для σ^2, то есть при $K \to \infty$ $\hat{\sigma}^2$ сходится с вероятностью 1, *сходится почти наверняка*, к σ^2. Центральная предельная теорема утверждает, что определенная ниже случайная величина Z_K^* сходится по распределению к стандартному нормальному распределению:

$$Z_K^* = \sqrt{K}\left(\frac{\hat{\mu} - \mu}{\hat{\sigma}}\right) \tag{16.8}$$

и, таким образом, мы имеем

$$\lim_{K\to\infty} P\left(\hat{\mu} - z_{\alpha/2}\frac{\hat{\sigma}}{\sqrt{K}} \le \mu \le \hat{\mu} + z_{\alpha/2}\frac{\hat{\sigma}}{\sqrt{K}}\right) = \alpha \tag{16.9}$$

Другими словами, мы говорим, что $\left(\hat{\mu} - z_{\frac{\alpha}{2}}\frac{\sigma}{\sqrt{K}}, \hat{\mu} + z_{\alpha/2}\frac{\sigma}{\sqrt{K}}\right)$ – доверительный интервал α % для μ.

Небольшой размер выборки

Доверительный интервал в формуле (16.9) применяется только при размере выборки K → ∞. Мы хотели бы получить более точные доверительные интервалы для небольших выборок. Рассмотрим случайные величины V_i для $i = 1, \ldots, K$, определенные как

$$V_i = \frac{\theta_i - \hat{\mu}}{\sigma}$$

Далее рассмотрим сумму их квадратов:

$$S = \sum_{i=1}^{K} V_i^2 = \sum_{i=1}^{K}\left(\frac{\theta_i - \hat{\mu}}{\sigma}\right)^2 = \frac{1}{\sigma^2}\sum_{i=1}^{K} (\theta_i - \hat{\mu})^2 = \frac{K\hat{\sigma}}{\sigma^2} \tag{16.10}$$

Последний шаг следует из определения выборочной дисперсии в формуле (16.7).

Если мы предположим, что V_i являются IID со стандартным нормальным распределением, тогда сумма S соответствует распределению хи-квадрат с $K - 1$ степенями свободы, обозначенным $\chi^2(K-1)$, поскольку S – сумма квадратов K случайных величин V_i. Есть только $K - 1$ степеней свободы, потому что каждая V_i зависит от $\hat{\mu}$ и, таким образом, сумма θ_i фиксирована.

Рассмотрим случайную величину Z_K^* в уравнении (16.8). У нас есть,

$$Z_K^* = \sqrt{K}\left(\frac{\hat{\mu} - \mu}{\hat{\sigma}}\right) = \left(\frac{\hat{\mu} - \mu}{\hat{\sigma}/\sqrt{K}}\right)$$

Разделив числитель и знаменатель в приведенном выше выражении на $\hat{\sigma}/\sqrt{K}$, получим

$$Z_K^* = \left(\frac{\hat{\mu} - \mu}{\sigma/\sqrt{K}} \Big/ \frac{\hat{\sigma}/\sqrt{K}}{\sigma/\sqrt{K}}\right) = \left(\frac{\frac{\hat{\mu} - \mu}{\sigma/\sqrt{K}}}{\hat{\sigma}/\sigma}\right) = \frac{Z_K}{\sqrt{S/K}} \tag{16.11}$$

Последний шаг следует из уравнения (16.10), потому что

$$S = \frac{K\hat{\sigma}^2}{\sigma^2} \text{ implies that } \frac{\hat{\sigma}}{\sigma} = \sqrt{S/K}$$

Предполагая, что Z_K соответствует стандартному нормальному распределению, и отмечая, что S соответствует распределению хи-квадрат с $K-1$ степенями свободы, тогда распределение Z_K^* является в точности распределением Стьюдента с $K-1$ степенями свободы. Таким образом, в случае небольшой выборки вместо использования стандартной нормальной плотности для получения доверительного интервала мы используем t распределение. В частности, мы выбираем значение $t_{\alpha/2,K-1}$ так, чтобы кумулятивная функция распределения t с $K-1$ степенями свободы охватывала $\alpha/2$ вероятностной массы, т. е.

$$P\left(0 \le Z_K^* \le t_{\alpha/2,K-1}\right) = T_{K-1}\left(t_{\alpha/2}\right) - T_{K-1}(0) = \alpha/2$$

где T_{K-1} – кумулятивная функция распределения для t распределения Стьюдента с $K-1$ степенями свободы. Поскольку t распределение симметрично относительно среднего, мы имеем

$$P\left(\hat{\mu} - t_{\alpha/2,K-1}\frac{\hat{\sigma}}{\sqrt{K}} \le \mu \le \hat{\mu} + t_{\alpha/2,K-1}\frac{\hat{\sigma}}{\sqrt{K}}\right) = \alpha \tag{16.12}$$

Таким образом, доверительный интервал α% для истинного среднего μ равен

$$\left(\hat{\mu} - t_{\alpha/2,K-1}\frac{\hat{\sigma}}{\sqrt{K}} \le \mu \le \hat{\mu} + t_{\alpha/2,K-1}\frac{\hat{\sigma}}{\sqrt{K}}\right)$$

Обратите внимание на зависимость интервала как от α, так и от размера выборки K.

На рисунке 16.6 показана функция плотности распределения t для различных значений K. На нем также показана стандартная функция нормальной плотности. Мы можем заметить, что у t распределения больше вероятностей сосредоточено в его хвостах, чем у стандартного нормального распределения. Кроме того, по мере увеличения K t распределение очень быстро сходится по распределению к стандартному нормальному распределению, что соответствует случаю с большой выборкой. Таким образом, для больших выборок можно использовать обычный порог $z_{\alpha/2}$.

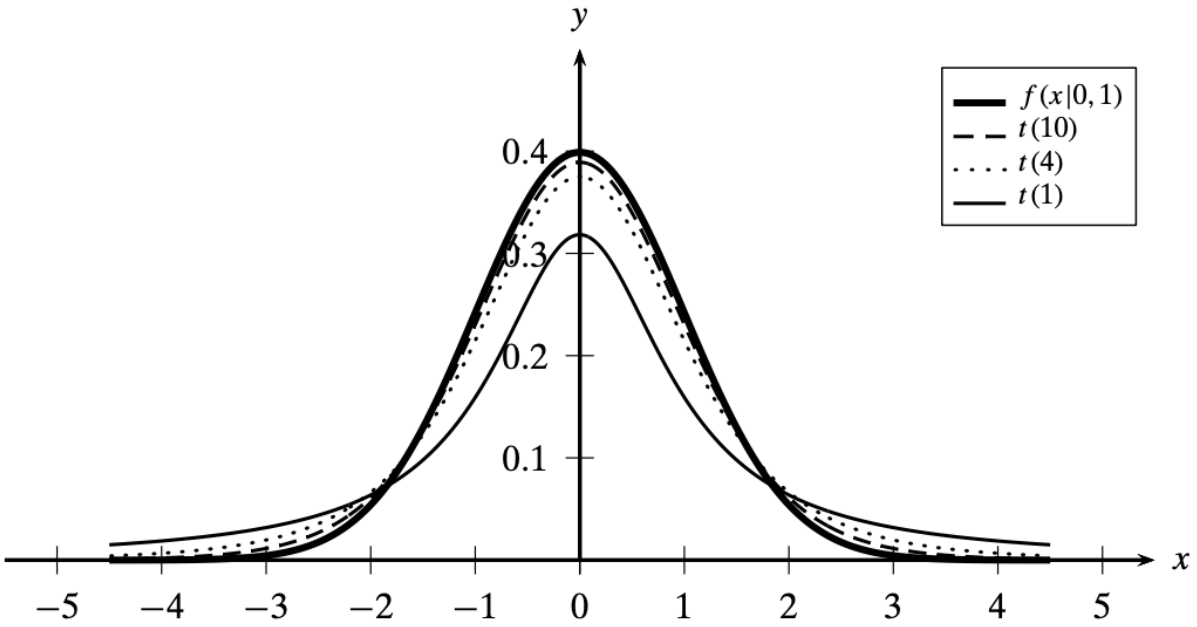


Рисунок 16.6 t-распределение студентов. K градусов свободы. Толстая линия – нормальное распределение.

16.2.4 Сравнение классификаторов: парный t-тест

В этом разделе мы рассмотрим метод, который позволяет нам проверить значительную разницу в производительности классификации двух альтернативных классификаторов, M^A и M^B. Мы хотим оценить, какие из них имеют лучшие характеристики классификации в данном наборе данных **D**.

Следуя методологии оценки, описанной выше, мы можем применить перекрестную K-кратную проверку (или повторную выборку начальной загрузки) и свести в таблицу их производительность по каждому из K сверток с идентичными свертками для обоих классификаторов. То есть мы проводим парный тест, когда оба классификатора обучены и протестированы на одних и тех же данных. Пусть $\theta_1^A, \theta_2^A, \ldots, \theta_K^A$ и $\theta_1^B, \theta_2^B, \ldots, \theta_K^B$ обозначают значения производительности для M^A и M^B соответственно. Чтобы определить, имеют ли два классификатора разную или одинаковую производительность, определим случайную величину δ_i как разницу в их производительности на i-м наборе данных:

$$\delta_i = \theta_i^A - \theta_i^B$$

Теперь рассмотрим оценки ожидаемой разницы и дисперсии разностей:

$$\hat{\mu}_\delta = \frac{1}{K}\sum_{i=1}^{K} \delta_i \qquad \hat{\sigma}_\delta^2 = \frac{1}{K}\sum_{i=1}^{K} (\delta_i - \hat{\mu}_\delta)$$

Мы можем настроить структуру проверки гипотез, чтобы определить, есть ли статистически значимая разница между производительностью M^A и M^B. Нулевая гипотеза H_0 состоит в том, что их производительность одинакова, то есть истинная ожидаемая разница равна нулю, тогда как альтернативная гипотеза H_a заключается в том, что они не одинаковы, то есть истинная ожидаемая разница μ_δ не равна нулю:

$$H_0: \quad \mu_\delta = 0 \qquad\qquad H_a: \quad \mu_\delta \neq 0$$

Определим случайную величину z-оценки для ожидаемой разницы как

$$Z_\delta^* = \sqrt{K}\left(\frac{\hat{\mu}_\delta - \mu_\delta}{\hat{\sigma}_\delta}\right)$$

Следуя аналогичным аргументам, как в формуле (16.11) Z_δ^* соответствует t распределению с $K-1$ степенями свободы. Однако при нулевой гипотезе $\mu_\delta = 0$, и, следовательно,

$$Z_\delta^* = \sqrt{K}\left(\frac{\hat{\mu}_\delta}{\hat{\sigma}_\delta}\right) \sim t_{K-1}$$

где обозначение $Z_\delta^* \sim t_{K-1}$ означает, что Z_δ^* следует распределению t с $K-1$ степенями свободы.

Учитывая желаемый уровень достоверности α, мы заключаем, что

$$P\left(-t_{\frac{\alpha}{2},K-1} \leq Z_\delta^* \leq t_{\frac{\alpha}{2},K-1}\right) = \alpha$$

Другими словами, если $Z_\delta^* \notin \left(-t_{\frac{\alpha}{2},K-1}, t_{\frac{\alpha}{2},K-1}\right)$, то мы можем отклонить нулевую гипотезу с достоверностью α%. В этом случае мы приходим к выводу, что есть существенная разница между производительностью M^A и M^B. С другой стороны, если Z_δ^* действительно лежит в указанном выше доверительном интервале, мы принимаем нулевую гипотезу о том, что и M^A, и M^B имеют, по существу, одинаковую производительность. Псевдокод для парного t-критерия показан в алгоритме 16.4.

Алгоритм 16.4: Сравнение классификаторов: парный t-тест

PAIRED t-TEST(α, K, D):

D ← randomly shuffle **D**
$\{\mathbf{D}_1, \mathbf{D}_2, \ldots, \mathbf{D}_K\}$ ← partition **D** in K equal parts
foreach $i \in [1, K]$ **do**
  M_i^A, M_i^B ← train the two different classifiers on $\mathbf{D} \setminus \mathbf{D}_i$
  θ_i^A, θ_i^B ← assess M_i^A and M_i^B on $\mathbf{D}_i$
  $\delta_i = \theta_i^A - \theta_i^B$
$\hat{\mu}_\delta = \frac{1}{K}\sum_{i=1}^K \delta_i$
$\hat{\sigma}_\delta^2 = \frac{1}{K}\sum_{i=1}^K (\delta_i - \hat{\mu}_\delta)^2$
$Z_\delta^* = \frac{\sqrt{K}\hat{\mu}_\delta}{\hat{\sigma}_\delta}$
if $Z_\delta^* \in \left(-t_{\alpha/2,K-1}, t_{\alpha/2,K-1}\right)$ **then**
  Accept H_0; both classifiers have similar performance
else
  Reject H_0; classifiers have significantly different performance

16.3 Разложение на смещение и разброс (Bias-variance decomposition)

Для обучающего набора $\mathbf{D} = \{\mathbf{x}_i, y_i\}_{i=1}^n$, содержащего n точек $\mathbf{x}_i \in \mathbb{R}^d$, с соответствующими им классами y_i, изученная модель классификации M предсказывает класс для данной контрольной точки $\mathbf{x}$. Различные меры производительности, описанные выше, в основном сосредоточены на минимизации ошибки прогнозирования путем табулирования доли ошибочно классифицированных точек. Однако во многих приложениях могут быть накладные расходы, вызванные неправильными прогнозами. *Функция потерь* определяет стоимость или штраф за предсказание того, что класс будет

$\hat{y} = M(\boldsymbol{x})$, когда истинным классом является y. Обычно используемой функцией потерь для классификации является коэффициент неправильной классификации (zero-one loss), определяемый как

$$L(y, M(\mathbf{x})) = I(M(\mathbf{x}) \neq y) = \begin{cases} 0 & \text{if } M(\mathbf{x}) = y \\ 1 & \text{if } M(\mathbf{x}) \neq y \end{cases}$$

Таким образом, коэффициенту неправильной классификации присваивается нулевая стоимость, если прогноз верен, и единичная в противном случае.

Другим часто используемой функцией потерь является *квадрат потерь*, определяемый как

$$L(y, M(\mathbf{x})) = (y - M(\mathbf{x}))^2$$

где мы предполагаем, что классы дискретны, а не качественны.

Ожидаемые потери

Идеальный или оптимальный классификатор - это тот, который минимизирует функцию потерь. Поскольку истинный класс для контрольного примера x неизвестен, цель изучения модели классификации может быть сформулирована как минимизация ожидаемых потерь:

$$E_y[L(y, M(\mathbf{x})) \mid \mathbf{x}] = \sum_y L(y, M(\mathbf{x})) \cdot P(y \mid \mathbf{x}) \qquad (16.13)$$

где $P(y \mid \mathbf{x})$ - это условная вероятность класса y для данной контрольной точки $\mathbf{x}$, а E_y означает, что математическое ожидание берется для различных значений класса y.

Минимизация ожидаемого коэффициента неправильной классификации соответствует минимизации коэффициента ошибок. Это можно увидеть, расширив формулу (16.13) с коэффициентом неправильной классификации. Пусть $M(\mathbf{x}) = c_i$, тогда имеем

$$\begin{aligned} E_y[L(y, M(\mathbf{x})) \mid \mathbf{x}] &= E_y[I(y \neq M(\mathbf{x})) \mid \mathbf{x}] \\ &= \sum_y I(y \neq c_i) \cdot P(y \mid \mathbf{x}) \\ &= \sum_{y \neq c_i} P(y \mid \mathbf{x}) \\ &= 1 - P(c_i \mid \mathbf{x}) \end{aligned}$$

Таким образом, чтобы минимизировать ожидаемые потери, мы должны выбрать c_i как класс, который максимизирует апостериорную вероятность, то есть $c_i = \text{argmax}_y P(y|\mathbf{x})$. Поскольку по определению [уравнение (16.1)], коэффициент ошибок - это просто оценка ожидаемого коэффициента неправильной классификации, этот выбор также минимизирует коэффициент ошибок.

Смещение и дисперсия

Ожидаемые потери для функции квадрата потерь предлагают важное понимание проблемы классификации, поскольку ее можно разложить на смещение и дисперсию. Интуитивно *смещение* классификатора относится к систематическому отклонению его прогнозируемой границы решения от истинной границы решения, тогда как *дисперсия* классификатора относится к отклонению между изученными границами принятия решения

по разным обучающим выборкам. Более формально, поскольку M зависит от обучающей выборки, для данной тестовой точки $\mathbf{x}$ мы обозначаем ее предсказанное значение как $M(\mathbf{x}, \mathbf{D})$. Рассмотрим ожидаемый квадрат потерь:

$$
\begin{aligned}
& E_y[L(y, M(\mathbf{x}, \mathbf{D})) \mid \mathbf{x}, \mathbf{D}] \\
& \quad = E_y[(y - M(\mathbf{x}, \mathbf{D}))^2 \mid \mathbf{x}, \mathbf{D}] \\
& \quad = E_y\left[(y \underbrace{-E_y[y \mid \mathbf{x}] + E_y[y \mid \mathbf{x}]}_{\text{add and subtract same term}} - M(\mathbf{x}, \mathbf{D}))^2 \mid \mathbf{x}, \mathbf{D}\right] \\
& \quad = E_y\left[\left(y - E_y[y \mid \mathbf{x}]\right)^2 \mid \mathbf{x}, \mathbf{D}\right] + E_y\left[\left(M(\mathbf{x}, \mathbf{D}) - E_y[y \mid \mathbf{x}]\right)^2 \mid \mathbf{x}, \mathbf{D}\right] \\
& \qquad + E_y\left[2\left(y - E_y[y \mid \mathbf{x}]\right) \cdot \left(E_y[y \mid \mathbf{x}] - M(\mathbf{x}, \mathbf{D})\right) \mid \mathbf{x}, \mathbf{D}\right] \\
& \quad = E_y\left[\left(y - E_y[y \mid \mathbf{x}]\right)^2 \mid \mathbf{x}, \mathbf{D}\right] + \left(M(\mathbf{x}, \mathbf{D}) - E_y[y \mid \mathbf{x}]\right)^2 \\
& \qquad + 2\left(E_y[y \mid \mathbf{x}] - M(\mathbf{x}, \mathbf{D})\right) \cdot \underbrace{\left(E_y[y \mid \mathbf{x}] - E_y[y \mid \mathbf{x}]\right)}_{0} \\
& \quad = \underbrace{E_y\left[\left(y - E_y[y \mid \mathbf{x}]\right)^2 \mid \mathbf{x}, \mathbf{D}\right]}_{\text{var } (y|\mathbf{x})} + \underbrace{\left(M(\mathbf{x}, \mathbf{D}) - E_y[y \mid \mathbf{x}]\right)^2}_{\text{squared-error}}
\end{aligned}
\tag{16.14}
$$

Выше мы использовали тот факт, что для любых случайных величин X и Y и для любой константы a мы имеем $E[X + Y] = E[X] + E[Y]$, $E[aX] = aE[X]$, и $E[a]=a$. Первый член в уравнении (16.14) - это просто дисперсия y при заданном $\mathbf{x}$. Второй член - это квадрат ошибки между предсказанным значением $M(\mathbf{x}, \mathbf{D})$ и ожидаемым значением $E_y[y|\mathbf{x}]$. Поскольку этот член зависит от обучающей выборки, мы можем устранить эту зависимость, усреднив все возможные обучающие тесты размера n. Средняя или ожидаемая квадратичная ошибка для данной контрольной точки $\mathbf{x}$ по всем обучающим выборкам тогда задается как

$$
\begin{aligned}
& E_{\mathbf{D}}\left[\left(M(\mathbf{x}, \mathbf{D}) - E_y[y \mid \mathbf{x}]\right)^2\right] \\
& \quad = E_{\mathbf{D}}\left[(M(\mathbf{x}, \mathbf{D}) \underbrace{-E_{\mathbf{D}}[M(\mathbf{x}, \mathbf{D})] + E_{\mathbf{D}}[M(\mathbf{x}, \mathbf{D})]}_{\text{add and subtract same term}} - E_y[y \mid \mathbf{x}])^2\right] \\
& \quad = E_{\mathbf{D}}[(M(\mathbf{x}, \mathbf{D}) - E_{\mathbf{D}}[M(\mathbf{x}, \mathbf{D})])^2] + E_{\mathbf{D}}\left[\left(E_{\mathbf{D}}[M(\mathbf{x}, \mathbf{D})] - E_y[y \mid \mathbf{x}]\right)^2\right] \\
& \qquad + 2\left(E_{\mathbf{D}}[M(\mathbf{x}, \mathbf{D})] - E_y[y \mid \mathbf{x}]\right) \cdot \underbrace{\left(E_{\mathbf{D}}[M(\mathbf{x}, \mathbf{D})] - E_{\mathbf{D}}[M(\mathbf{x}, \mathbf{D})]\right)}_{0} \\
& \quad = \underbrace{E_{\mathbf{D}}[(M(\mathbf{x}, \mathbf{D}) - E_{\mathbf{D}}[M(\mathbf{x}, \mathbf{D})])^2]}_{\text{variance}} + \underbrace{\left(E_{\mathbf{D}}[M(\mathbf{x}, \mathbf{D})] - E_y[y \mid \mathbf{x}]\right)^2}_{\text{bias}}
\end{aligned}
\tag{16.15}
$$

Это означает, что ожидаемая квадратичная ошибка для данной контрольной точки может быть разложена на смещение и дисперсию. Комбинируя уравнения (16.14) и (16.15) ожидаемый квадрат потерь по всем тестовым точкам **x** и по всем обучающим наборам **D** размера **n** дает следующее разложение на шум, дисперсию и смещение:

$$
\begin{aligned}
&E_{\mathbf{x},\mathbf{D},y}[(y - M(\mathbf{x},\mathbf{D}))^2] \\
&\quad = E_{\mathbf{x},\mathbf{D},y}\left[\left(y - E_y[y \mid \mathbf{x}]\right)^2 \mid \mathbf{x},\mathbf{D}\right] + E_{\mathbf{x},\mathbf{D}}\left[\left(M(\mathbf{x},\mathbf{D}) - E_y[y \mid \mathbf{x}]\right)^2\right] \\
&\quad = \underbrace{E_{\mathbf{x},y}\left[\left(y - E_y[y \mid \mathbf{x}]\right)^2\right]}_{\text{noise}} + \underbrace{E_{\mathbf{x},\mathbf{D}}[(M(\mathbf{x},\mathbf{D}) - E_{\mathbf{D}}[M(\mathbf{x},\mathbf{D})])^2]}_{\text{average variance}} \\
&\qquad\qquad + \underbrace{E_{\mathbf{x}}\left[\left(E_{\mathbf{D}}[M(\mathbf{x},\mathbf{D})] - E_y[y \mid \mathbf{x}]\right)^2\right]}_{\text{average bias}}
\end{aligned}
\tag{16.16}
$$

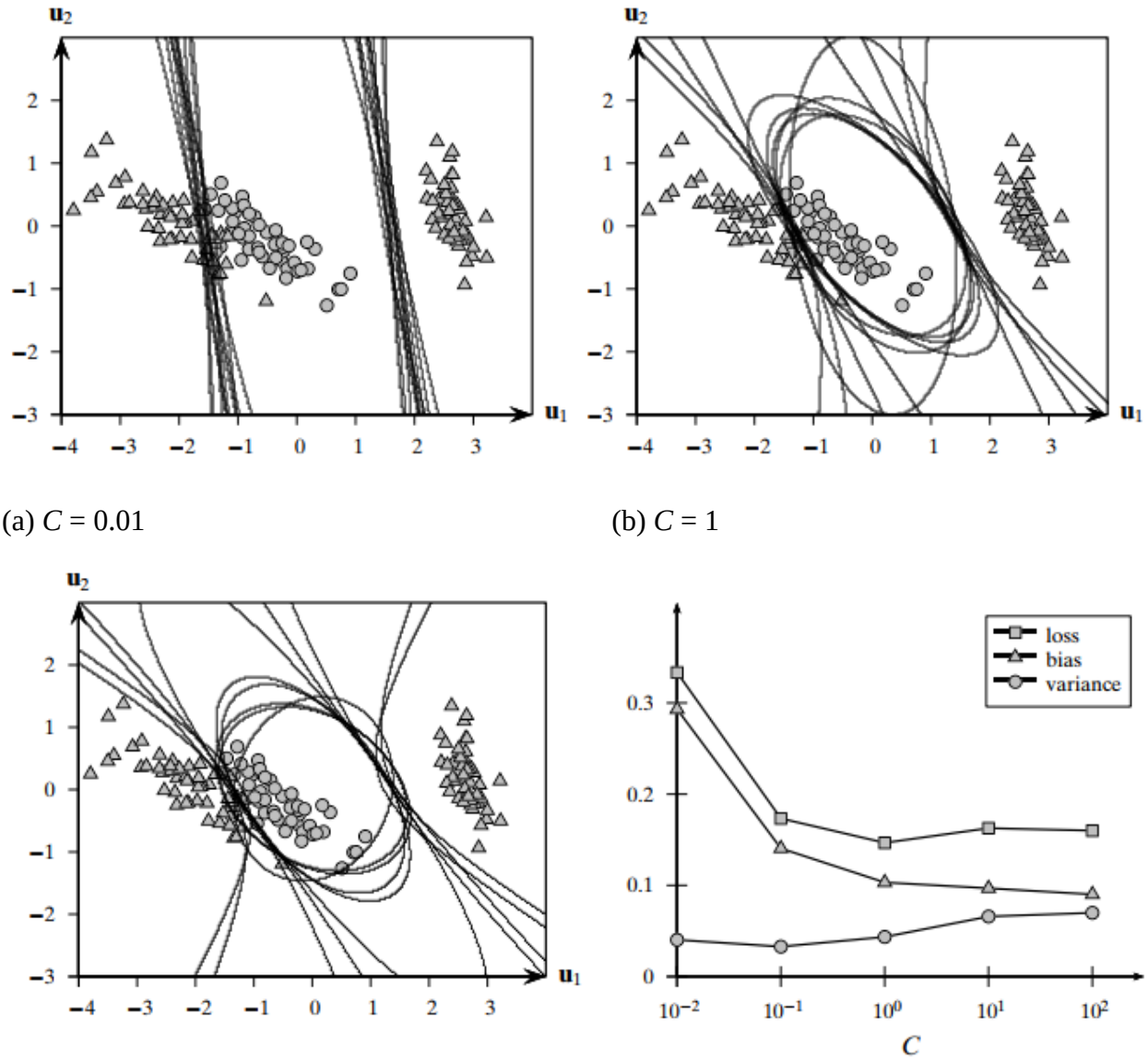
Таким образом, ожидаемый квадрат потерь по всем контрольным точкам и обучающим наборам можно разложить на три состовляющие: шум, среднее смещение и среднее отклонение. Шумовая составляющая - это средняя дисперсия *var* ($y \mid \mathbf{x}$) по всем контрольным точкам **x**. Он вносит фиксированный вклад в потери независимо от модели и, таким образом, может быть проигнорирован при сравнении различных классификаторов. Затем потери, характерные для классификатора, могут быть отнесены к условиям дисперсии и смещения. В общем, смещение указывает на то, верна модель *M* или нет. Оно также отражает наши предположения о предметной области с точки зрения границы принятия решения. Например, если граница принятия решения является нелинейной, и мы используем линейный классификатор, то он, вероятно, будет иметь большое смещение, то есть будет постоянно неверным для разных обучающих наборов. С другой стороны, нелинейный (или более сложный) классификатор с большей вероятностью уловит правильную границу принятия решения и, следовательно, будет иметь небольшое смещение. Тем не менее, это не обязательно означает, что сложный классификатор будет лучше, поскольку мы также должны учитывать составляющую дисперсии, которая измеряет несогласованность решений классификатора. Сложный классификатор порождает более сложную границу принятия решений и, следовательно, может быть подвержен *переобучению*, то есть он может пытаться смоделировать все мелкие нюансы в обучающих данных и, таким образом, может быть восприимчив к небольшим изменениям в обучающем наборе, что может привести к большой дисперсии.

В общем, ожидаемые потери могут быть отнесены на счет большого смещения или большой дисперсии, обычно с компромиссом между этими двумя терминами. В идеале мы ищем баланс между этими противоположными тенденциями, то есть мы предпочитаем классификатор с приемлемым смещением (отражающим предположения, относящиеся к предметной области или набору данных) и с минимально возможной дисперсией.

Пример 16.7 На рисунке 16.7 показан компромисс между смещением и дисперсией с использованием основных компонентов набора данных Iris, который имеет $n = 150$ точек и $k = 2$ класса ($c_1 = +1$ и $c_2 = -1$). Мы создаем K=10 обучающих наборов данных с помощью бутстреп выборки и используем их для обучения классификаторов SVM с использованием квадратичного (однородного) ядра, изменяя константу регуляризации C от 10^{-2} до 10^2.

Напомним, что C контролирует вес, придаваемый резервным переменным в отличие от края гиперплоскости. Небольшое значение C подчеркивает запас, тогда как большое значение C пытается минимизировать условия резервирования. Рисунки 16.7a, 16.7b и 16.7c

показывают, что дисперсия модели SVM увеличивается по мере увеличения C, как видно из изменяющихся границ решения. На рисунке 16.7d показаны средняя дисперсия и среднее смещение для разных значений C, а также ожидаемые потери. Компромисс смещения и дисперсии четко виден, поскольку по мере уменьшения смещения дисперсия увеличивается. Наименьшие ожидаемые потери получаем при $C = 1$.



(a) $C = 0.01$ (b) $C = 1$

(c) $C = 100$ (d) Смещение и дисперсия

Рисунок 16.7. Разложение на смещение и дисперсию: квадратичные ядра SVM. Границы решения построены для $K = 10$ бутстреп выборок.

16.3.1 Классификаторы ансамбля

Классификатор называется *неустойчивым*, если небольшие возмущения в обучающей выборке приводят к большим изменениям границы прогноза или решения. Классификаторы с высокой дисперсией по своей природе неустойчивым, так как они имеют тенденцию чрезмерно соответствовать данным. С другой стороны, методы с высоким смещением обычно не соответствуют данным и обычно имеют низкую дисперсию. В любом случае цель обучения - уменьшить ошибку классификации за счет уменьшения

дисперсии или смещения, в идеале и того, и другого. Методы ансамбля создают *комбинированный классификатор*, используя выходные данные нескольких *базовых классификаторов*, которые обучаются на разных подмножествах данных. В зависимости от того, как выбраны обучающие наборы, и от устойчивости базовых классификаторов, ансамблевые классификаторы могут помочь уменьшить дисперсию и смещение, что приведет к повышению общей производительности.

Бэггинг

Бэггинг, сокращенно от бутстреп-агрегирования, представляет собой метод классификации ансамбля, который использует несколько бутстреп выборок (с заменой) из входных обучающих данных **D** для создания немного разных обучающих наборов $\mathbf{D}_i$, i = 1,2, ..., K. Изучены различные базовые классификаторы M_i, причем M_i обучен на $\mathbf{D}_i$. Для любой контрольной точки **x** она сначала классифицируется с использованием каждого из K базовых классификаторов M_i. Пусть количество классификаторов, которые предсказывают класс **x** как c_j, задано как

$$v_j(\mathbf{x}) = \left|\{M_i(\mathbf{x}) = c_j \mid i = 1, \ldots, K\}\right|$$

Комбинированный классификатор, обозначенный $\boldsymbol{M}^K$, предсказывает класс контрольной точки **x** *большинством голосов* среди k классов:

$$\mathbf{M}^K(\mathbf{x}) = \arg\max_{c_j} \{v_j(\mathbf{x}) \mid j = 1, \ldots, k\}$$

Для двоичной классификации, предполагая, что классы заданы как {+1, −1}, объединенный классификатор $\mathbf{M}^K$ может быть выражен проще

$$\mathbf{M}^K(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^{K} M_i(\mathbf{x})\right)$$

Бэггинг может помочь уменьшить дисперсию, особенно если базовые классификаторы неустойчивы, из-за эффекта усреднения при голосовании большинством. В целом это не оказывает большого влияния на смещение.

Пример 16.8. На рисунке 16.8a показан эффект усреднения бэггинга для основных компонентов набора данных Iris из примера 16.12. На рисунке показаны границы решения SVM для квадратичного ядра при C = 1. Базовые классификаторы SVM обучаются на K = 10 бутстреп выборках. Комбинированный (средний) классификатор выделен жирным шрифтом.

На рисунке 16.8b показаны комбинированные классификаторы, полученные для различных значений K, при сохранении C = 1. Коэффициент неправильной классификации и квадрат потерь для выбранных значений K показаны ниже.

K	Коэффициент неправильной классификации	Квадрат потерь
3	0.047	0.187
5	0.04	0.16

8	0.02	0.10
10	0.027	0.113
15	0.027	0.107

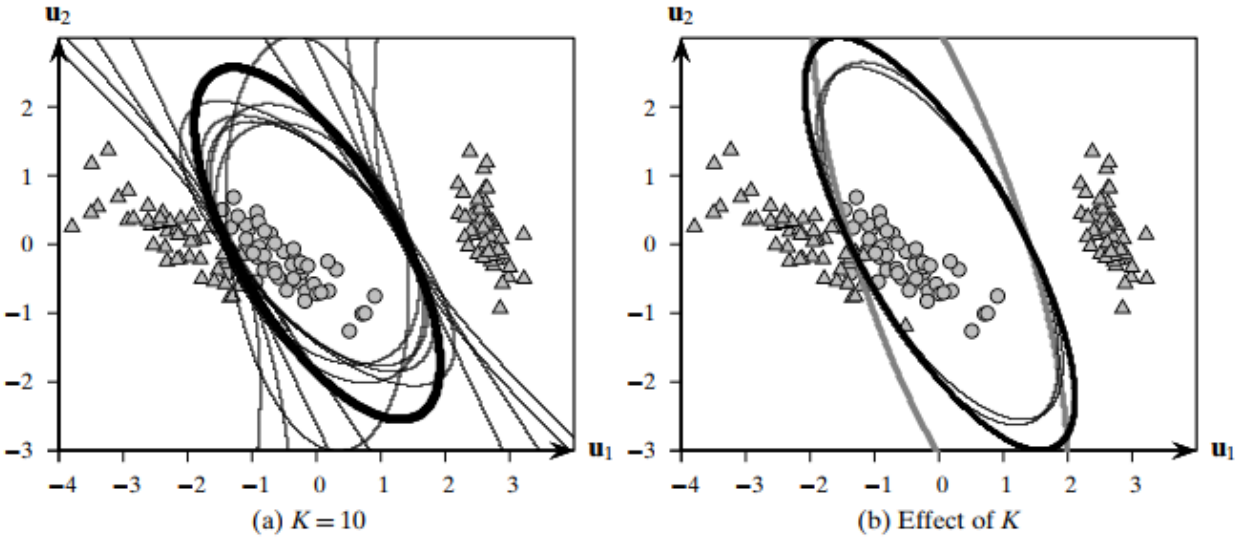


Рисунок 16.8. Бэггинг: комбинированные классификаторы. (а) использует K = 10 бутстреп выборки. (б) показана средняя граница решения для различных значений K.

Наихудшие результаты обучения получены для K = 3 (жирный серый цвет), а лучшие - для K = 8 (толстый черный цвет).

Бустинг

Бустинг - это еще один метод ансамбля, который тренирует базовые классификаторы на разных выборках. Однако основная идея состоит в том, чтобы тщательно отбирать выборки, чтобы *повысить* производительность трудно классифицируемых экземпляров. Начиная с начальной обучающей выборки $\mathbf{D}_1$, мы обучаем базовый классификатор M_1 и получаем его частоту ошибок обучения. Чтобы построить следующую выборку $\mathbf{D}_2$, мы выбираем неверно классифицированные экземпляры с большей вероятностью, и после обучения M_2 мы получаем его частоту ошибок обучения. Для построения $\mathbf{D}_3$ те экземпляры, которые сложно классифицировать по M_1 или M_2, будут выбраны с большей вероятностью. Этот процесс повторяется K итераций. Таким образом, в отличие от бэггинга, в котором используются независимые случайные выборки из входного набора данных, бустинг использует взвешенные или смещенные выборки для построения различных обучающих наборов, при этом текущая выборка зависит от предыдущих. Наконец, объединенный классификатор получается путем взвешенного голосования по выходным данным K базовых классификаторов M_1, M_2, ..., M_K.

Бустинг наиболее выгоден, когда базовые классификаторы *слабые*, то есть имеют коэффициент ошибок, который немного меньше, чем у случайного классификатора. Идея состоит в том, что, хотя M_1 может быть не особенно хорош во всех тестовых экземплярах, по замыслу M_2 может помочь классифицировать некоторые случаи, когда M_1 не работает, а M_3 может помочь классифицировать экземпляры, где M_1 и M_2 не работают, и так далее. Таким образом, бустинг имеет больший эффект уменьшения смещения. Каждый из слабых учеников, вероятно, будет иметь высокое смещение (это лишь немного лучше, чем

случайное предположение), но окончательный комбинированный классификатор может иметь гораздо меньшее смещение, поскольку разные слабые ученики учатся классифицировать экземпляры в разных областях входного пространства. Несколько вариантов бустинга могут быть получены на основе того, как вычисляются веса экземпляров для выборки, как комбинируются базовые классификаторы и так далее. Мы обсудим Adaptive Boosting (AdaBoost), который является одним из самых популярных вариантов.

Алгоритм 16.5: Адаптивный алгоритм бустинга: AdaBoost

ADABOOST(K, D):

$\mathbf{w}^0 \leftarrow \left(\frac{1}{n}\right) \cdot \mathbf{1} \in \mathbb{R}^n$

$t \leftarrow 1$

while $t \leq K$ **do**

  $\mathbf{D}_t \leftarrow$ weighted resampling with replacement from $\mathbf{D}$ using $\mathbf{w}^{t-1}$

  $M_t \leftarrow$ train classifier on $\mathbf{D}_t$

  $\epsilon_t \leftarrow \sum_{i=1}^{n} w_i^{t-1} \cdot I\left(M_t(\mathbf{x}_i) \neq y_i\right)$ `// weighted error rate on` **D**

  if $\epsilon_t = 0$ **then break**

  else if $\epsilon_t < 0.5$ **then**

    $\alpha_t = \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$ `// classifier weight`

    foreach $i \in [1, n]$ **do**

      `// update point weights`

      $$w_i^t = \begin{cases} w_i^{t-1} & \text{if } M_t(\mathbf{x}_i) = y_i \\ w_i^{t-1}\left(\frac{1-\epsilon_t}{\epsilon_t}\right) & \text{if } M_t(\mathbf{x}_i) \neq y_i \end{cases}$$

    $\mathbf{w}^t = \frac{\mathbf{w}^t}{\mathbf{1}^T\mathbf{w}^t}$ `// normalize weights`

    $t \leftarrow t + 1$

return $\{M_1, M_2, \ldots, M_K\}$

Adaptive Boosting: AdaBoost Пусть **D** - входной обучающий набор, состоящий из n точек $\mathbf{x}_i \in \mathbb{R}^d$. Процесс бустинга будет повторен K раз. Пусть t обозначает итерацию, а α_t обозначает вес для t-го классификатора M_t. Пусть ω_i^t обозначает вес для $\mathbf{x}_i$, причем $\mathbf{w}^t = (\omega_1^t, \omega_2^t, \ldots, \omega_n^t)^T$ - вектор веса по всем точкам для t-й итерации. Фактически, **w** - это вектор вероятности, сумма элементов которого равна единице. Изначально все точки имеют одинаковый вес, то есть

$$\mathbf{w}^0 = \left(\frac{1}{n}, \frac{1}{n}, \ldots, \frac{1}{n}\right)^T = \frac{1}{n}\mathbf{1}$$

где $1 \in \mathbb{R}^n$ n-мерный вектор всех единиц.

Псевдокод для AdaBoost показан в алгоритме 16.5. Во время итерации t обучающая выборка $\mathbf{D}_t$ получается посредством взвешенной передискретизации с использованием распределения $\mathbf{w}^{t-1}$, то есть мы рисуем выборку размера n с заменой, так что i-я точка выбирается в соответствии с ее вероятностью ω_i^{t-1}. Затем мы обучаем классификатор M_t, используя $\mathbf{D}_t$, и вычисляем его взвешенную частоту ошибок ϵ_t для всего входного набора данных **D**:

$$\epsilon_t = \sum_{i=1}^{n} w_i^{t-1} \cdot I(M_t(\mathbf{x}_i) \neq y_i)$$

где I - индикаторная функция, которая равна 1, когда ее аргумент истинен, то есть когда M_t неверно классифицирует $\mathbf{x}_i$, и равен 0 в противном случае.

Затем вес для t-го классификатора устанавливается как

$$\alpha_t = \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$$

и вес для каждой точки $\mathbf{x}_i \in \mathbf{D}$ обновляется в зависимости от того, была ли точка классифицирована неправильно или нет

$$w_i^t = w_i^{t-1} \cdot \exp\{\alpha_t \cdot I(M_t(\mathbf{x}_i) \neq y_i)\}$$

Таким образом, если предсказанный класс соответствует истинному классу, то есть, если M_t ($\mathbf{x}_i$) = y_i, то $I(M_t$ ($\mathbf{x}_i$) ≠ y_i) = 0, и вес для точки $\mathbf{x}_i$ остается неизменным. С другой стороны, если точка классифицирована неправильно, то есть M_t ($\mathbf{x}_i$) ≠ y_i, то мы имеем $I(M_t$ ($\mathbf{x}_i$) ≠ y_i) = 1, и

$$w_i^t = w_i^{t-1} \cdot \exp\{\alpha_t\} = w_i^{t-1}\exp\left\{\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)\right\} = w_i^{t-1}\left(\frac{1}{\epsilon_t} - 1\right)$$

Мы можем заметить, что если частота ошибок $Є_t$ мала, то для $\mathbf{x}_i$ есть большее приращение веса. Интуиция подсказывает, что точка, неправильно классифицированная хорошим классификатором (с низким уровнем ошибок), с большей вероятностью будет выбрана для следующего набора обучающих данных. С другой стороны, если частота ошибок базового классификатора близка к 0,5, то изменение веса будет незначительным, поскольку ожидается, что плохой классификатор (с высокой частотой ошибок) неправильно классифицирует многие экземпляры. Обратите внимание, что для задачи двоичного класса коэффициент ошибок 0,5 соответствует случайному классификатору, то есть тому, который делает случайное предположение. Таким образом, мы требуем, чтобы базовый классификатор имел частоту ошибок, по крайней мере, немного лучше, чем случайное предположение, то есть $Є_t$ <0,5. Если частота ошибок $Є_t \geq 0{,}5$, то алгоритм бустинга отбрасывает классификатор и возвращается к строке 5, чтобы попробовать другую выборку данных. В качестве альтернативы можно просто инвертировать прогнозы для двоичной классификации. Стоит подчеркнуть, что для задачи с несколькими классами (с k > 2) требование, чтобы $Є_t$ <0,5 является значительно более строгим, чем для задачи двоичного класса (k = 2), потому что в случае с несколькими классами ожидается, что частота ошибок для случайного классификатора будет составлять $\frac{k-1}{k}$. Также обратите внимание, что если частота ошибок базового классификатора $Є_t$ = 0, то мы можем остановить итерации бустинга.

После обновления весов точек мы повторно нормализуем веса так, чтобы $\mathbf{w}_t$ стало вектором вероятности (строка 14):

$$\mathbf{w}^t = \frac{\mathbf{w}^t}{\mathbf{1}^T\mathbf{w}^t} = \frac{1}{\sum_{j=1}^{n} w_j^t}(w_1^t, w_2^t, \ldots, w_n^t)^T$$

Комбинированный классификатор Учитывая набор классификаторов после бустинга M_1, M_2, ..., M_K, вместе с их весами α_1, α_2, ..., α_K, класс для контрольного примера **x**

получается посредством голосования взвешенного большинства. Пусть v_j (**x**) обозначает взвешенный голос за класс c_j над K классификаторами, заданный как

$$v_j(\mathbf{x}) = \sum_{t=1}^{K} \alpha_t \cdot I\big(M_t(\mathbf{x}) = c_j\big)$$

Поскольку $I(M_t(\mathbf{x}) = c_j)$ равно 1, только когда $M_t(\mathbf{x}) = c_j$, переменная $v_j(\mathbf{x})$ просто получает результат для класса c_j среди K базовых классификаторов с учетом весов классификатора. Комбинированный классификатор, обозначенный $\mathbf{M}^K$, затем прогнозирует класс для **x** следующим образом:

$$\mathbf{M}^K(\mathbf{x}) = \arg\max_{c_j} \{v_j(\mathbf{x}) \mid j = 1, \ldots, k\}$$

В случае двоичной классификации с классами {+1, −1} комбинированный классификатор $\mathbf{M}^K$ можно выразить проще как

$$\mathbf{M}^K(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{K} \alpha_t M_t(\mathbf{x})\right)$$

Пример 16.9. На рисунке 16.9а показан подход к бустингу на основных компонентах набора данных Iris с использованием линейных SVM в качестве базовых классификаторов. Константа регуляризации была задана как $C = 1$. Гиперплоскость, изученная на итерации t, обозначается h_t, таким образом, модель классификатора задается как $M_t(\text{x}) = \text{sign}(h_t(\mathbf{x}))$. Таким образом, никакая отдельная линейная гиперплоскость не может хорошо различать классы, как видно из их коэффициентов ошибок на обучающем наборе:

M_t	h_1	h_2	h_3	h_4
ϵ_t	0.280	0.305	0.174	0.282
α_t	0.944	0.826	1.559	0.935

Однако, когда мы объединяем решения из последовательных гиперплоскостей, взвешенных по α_t, мы наблюдаем заметное падение частоты ошибок для комбинированного классификатора $\mathbf{M}^K$ (**x**) по мере увеличения K:

combined model	$\mathbf{M}^1$	$\mathbf{M}^2$	$\mathbf{M}^3$	$\mathbf{M}^4$
training error rate	0.280	0.253	0.073	0.047

Можно заметить, например, что комбинированный классификатор $\mathbf{M}^3$, включающий h_1, h_2 и h_3, уже уловил существенные особенности границы нелинейного решения между

двумя классами, что дает частоту ошибок 7,3%. Дальнейшее снижение ошибки обучения достигается за счет увеличения количества шагов алгоритма бустинга.

Чтобы оценить эффективность комбинированного классификатора на независимых тестовых данных, мы используем 5-кратную перекрестную проверку и построим график зависимости средней частоты ошибок тестирования и обучения от K на рис. 16.9b. Можно заметить, что по мере увеличения числа базовых классификаторов K частота ошибок как обучения, так и тестирования уменьшается. Однако, хотя ошибка обучения по существу становится равной 0, ошибка тестирования не уменьшается более 0,02, что происходит при $K = 110$. Этот пример иллюстрирует эффективность бустинга в уменьшении смещения.

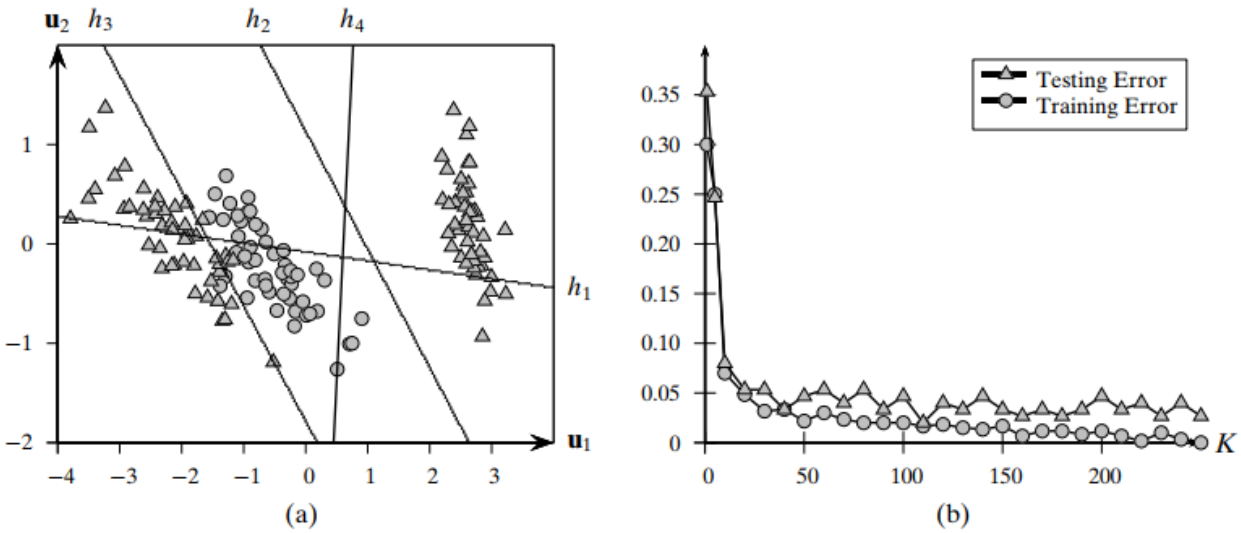


Рисунок 16.9. (a) Бустинг SVM с линейным ядром. (b) Средняя ошибка тестирования и обучения: 5-кратная перекрестная проверка.

Бэггинг как частный случай AdaBoost: Бэггинг можно рассматривать как частный случай AdaBoost, где $\omega^t = \frac{1}{n}\mathbf{1}$, а $\alpha_t = 1$ для всех K итераций. В этом случае для взвешенной повторной выборки по умолчанию используется обычная повторная выборка с заменой, а для прогнозируемого класса для тестового примера также по умолчанию используется простое правило большинства.

Глава 17. Лабораторные работы

17.1 Лабораторная работа № 1

Предобработка данных

Цель:

Ознакомиться с методами предобработки данных из библиотеки Scikit Learn

Выполнение:

Загрузка данных

1. Загрузить датасет по ссылке: https://www.kaggle.com/andrewmvd/heart-failure-clinical-dat
   a. Данные представлены в виде csv таблицы.
2. Создать Python скрипт. Загрузить датасет в датафрейм, и исключить бинарные признаки и признак времени

```
import pandas as pd
import numpy as np
df = pd.read_csv('heart_failure_clinical_records_dataset.csv')
df = df.drop(columns =
['anaemia','diabetes','high_blood_pressure','sex','smoking','time','DEATH_EV
ENT'])
print(df) #Вывод датафрейма с данными для лаб. работы. Должно быть 299
наблюдений и 6 признаков
```

3. Построить гистограммы признаков

```
import matplotlib.pyplot as plt
n_bins = 20
fig, axs = plt.subplots(2,3)
axs[0, 0].hist(df['age'].values, bins = n_bins)
axs[0, 0].set_title('age')
axs[0, 1].hist(df['creatinine_phosphokinase'].values, bins = n_bins)
axs[0, 1].set_title('creatinine_phosphokinase')
axs[0, 2].hist(df['ejection_fraction'].values, bins = n_bins)
axs[0, 2].set_title('ejection_fraction')
axs[1, 0].hist(df['platelets'].values, bins = n_bins)
```

```
axs[1, 0].set_title('platelets')

axs[1, 1].hist(df['serum_creatinine'].values, bins = n_bins)

axs[1, 1].set_title('serum_creatinine')

axs[1, 2].hist(df['serum_sodium'].values, bins = n_bins)

axs[1, 2].set_title('serum_sodium')

plt.show()
```

4. На основании гистограмм определите диапазоны значений для каждого из признаков, а также возле какого значения лежит наибольшее количество наблюдений.

5. Так как библиотека Sklearn работает с NumPy массива, то преобразуйте датафрейм к двумерному массиву NumPy, где строка соответствует наблюдению, а столбец признаку

```
data = df.to_numpy(dtype='float')
```

Стандартизация данных

1. Подключите модуль Sklearn. Настройте стандартизацию на основе первых 150 наблюдений используя StandardScaler

```
from sklearn import preprocessing

scaler = preprocessing.StandardScaler().fit(data[:150,:])
```

2. Стандартизуйте все данные

```
data_scaled = scaler.transform(data)
```

3. Постройте гистограммы стандартизированных данных

```
fig, axs = plt.subplots(2,3)

axs[0, 0].hist(data_scaled[:,0], bins = n_bins)

axs[0, 0].set_title('age')

axs[0, 1].hist(data_scaled[:,1], bins = n_bins)

axs[0, 1].set_title('creatinine_phosphokinase')

axs[0, 2].hist(data_scaled[:,2], bins = n_bins)

axs[0, 2].set_title('ejection_fraction')

axs[1, 0].hist(data_scaled[:,3], bins = n_bins)

axs[1, 0].set_title('platelets')
```

```
axs[1, 1].hist(data_scaled[:,4], bins = n_bins)

axs[1, 1].set_title('serum_creatinine')

axs[1, 2].hist(data_scaled[:,5], bins = n_bins)

axs[1, 2].set_title('serum_sodium')

plt.show()
```

4. Сравните данные до и после стандартизации. Опишите, что изменилось и почему.
5. Рассчитайте мат. ожидание и СКО до и после стандартизации. На основании этих значений выведите для каждого признака формулы по которым они стандартизировались.
6. Сравните значений из формул с полями mean_ и var_ объекта scaler
7. Проведите настройку стандартизации на всех данных и сравните с результатами настройки на основании 150 наблюдений.

Примечание: вместо двух методов fit и transform можно использовать метод fit_transfrom, чтобы сразу настроить параметры и преобразовать данные.

Приведение к диапазону

1. Приведите данные к диапазону используя MinMaxScaler

```
min_max_scaler = preprocessing.MinMaxScaler().fit(data)

data_min_max_scaled = min_max_scaler.transform(data)
```

2. Постройте гистограммы для признаков и сравните с исходными данными
3. Через параметры MinMaxScaler определите минимальное и максимальное значение в данных для каждого признака
4. Аналогично трансформируйте данные используя MaxAbsScaler и RobustScaler. Постройте гистограммы. Определите к какому диапазону приводятся данные.
5. Напишите функцию, которая приводит все данные к диапазону [-5 10]

Нелинейные преобразования

1. Приведите данные к равномерному распределению используя QuantileTransformer

```
quantile_transformer = preprocessing.QuantileTransformer(n_quantiles = 100,

random_state=0).fit(data)

data_quantile_scaled = quantile_transformer.transform(data)
```

2. Постройте гистограммы и сравните с исходными данными
3. Определите, как и на что влияет значение параметра n_quantiles
4. Приведите данные к нормальному распределению передав в QuantileTransformer параметр output_distribution=‘normal’

5. Постройте гистограммы и сравните с исходными данными
6. Самостоятельно приведите данные к нормальному распределению используя PowerTransformer

Дискретизация признаков

1. Проведите дискретизацию признаков, используя KBinsDiscretizer, на следующее количество диапазонов:
   - age - 3
   - creatinine_phosphokinase - 4
   - ejection_fraction - 3
   - platelets - 10
   - serum_creatinine - 2
   - serum_sodium - 4
2. Постройте гистограммы. Объясните полученные результаты
3. Через параметр bin_edges_ выведите диапазоны каждого интервала для каждого признака

17.2 Лабораторная работа № 2

Понижение размерности пространства признаков

Цель:

Ознакомиться с методами понижения размерности данных из библиотеки Scikit Learn

Выполнение:

Загрузка данных

1. Загрузить датасет по ссылке: https://www.kaggle.com/uciml/glass . Данные представлены в виде csv таблицы.
2. Создать Python скрипт. Загрузить датасет в датафрейм, и разделить данные на описательные признаки и признак отображающий класс.

```
import pandas as pd
import numpy as np
df = pd.read_csv('glass.csv')
var_names = list(df.columns) #получение имен признаков
labels = df.to_numpy('int')[:,-1] #метки классов
data = df.to_numpy('float')[:,:-1] #описательные признаки
```

3. Провести нормировку данных к интервалу [0 1]

```
from sklearn import preprocessing
data = preprocessing.minmax_scale(data)
```

4. Построить диаграммы рассеяния для пар признаков. Самостоятельно определите соответствие цвета на диаграмме и класса в датасете

```
import matplotlib.pyplot as plt
fig, axs = plt.subplots(2,4)
for i in range(data.shape[1]-1):
axs[i // 4, i % 4].scatter(data[:,i],data[:,(i+1)],c=labels,cmap='hsv')
axs[i // 4, i % 4].set_xlabel(var_names[i])
axs[i // 4, i % 4].set_ylabel(var_names[i+1])
plt.show()
```

Метод главных компонент

1. Используя метод главных компонент (PCA). Проведите понижение размерности пространства до размерности 2

```
from sklearn.decomposition import PCA
pca = PCA(n_components = 2)
pca_data = pca.fit(data).transform(data)
```

2. Выведите значение объясненной дисперсии в процентах и собственные числа соответствующие компонентам

```
print(pca.explained_variance_ratio_)
print(pca.singular_values_)
```

3. Постройте диаграмму рассеяния после метода главных компонент

```
plt.scatter(pca_data[:,0],pca_data[:,1],c=labels,cmap='hsv')
plt.show()
```

4. Проанализируйте и обоснуйте полученные результаты
5. Изменяя количество компонент, определите количество при котором компоненты объясняют не менее 85% дисперсии данных
6. Используя метод inverse_transform восстановите данные, сравните с исходными
7. Исследуйте метод главных компонент при различных параметрах svd_solver

Модификации метода главных компонент

1. По аналогии с PCA исследуйте KernelPCA для различных параметров kernel и различных параметрах для ядра
2. Определите, при каких параметрах KernelPCA работает также как PCA

3. Аналогично исследуйте SparcePCA
4. Проанализируйте и обоснуйте полученные результаты

Факторный анализ

1. Проведите понижении размерности используя факторный анализ FactorAnalysis
2. Сравните полученные результаты с PCA
3. Объясните в чем разница между методом главных компонент и факторным анализом

17.3 Лабораторная работа № 3

Частотный анализ

Цель:

Ознакомиться с методами частотного анализа из библиотеки MLxtend

Выполнение:

Для выполнения лаб. работы необходимо установить библиотеку MLxtend

Для этого в терминале введите:

```
pip install mlxtend
```

Загрузка данных

1. Загрузить датасет по ссылке: https://www.kaggle.com/acostasg/random-shopping-cart. Данные представлены в виде csv таблицы. Данные представляют собой информацию о том, какой покупатель что и когда покупал. В данной лаб. работе не будем использовать данные о дате покупки.
2. Создать Python скрипт. Загрузить данные в датафрейм.

```
import pandas as pd
import numpy as np
all_data = pd.read_csv('dataset_group.csv',header=None)
#В файле нет строки с названием столбцов, поэтому параметр header равен
None.
#Интерес представляет информация об id покупателя - столбец с названием 1
#Название купленного товара хранится в столбце с названием 2
```

3. Получим список всех id покупателей, которые есть в файле.

```
unique_id = list(set(all_data[1]))
print(len(unique_id)) #Выведем количество id
```

4. Получим список всех товаров, которые есть в файле.

```
items = list(set(all_data[2]))
print(len(items)) #Выведем количество товаров
```

5. Далее необходимо сформировать датасет подходящий для частотного анализа. Для этого надо слить все товары одного покупателя в один список. Для дальнейшего частотного анализа id покупателя будет не нужен

```
dataset = [[elem for elem in all_data[all_data[1] == id][2] if elem in
items] for id in unique_id]
```

Подготовка данных

1. Так как полученные датасет не пригоден для анализа напрямую, так как каждый список пользователя может содержать разное количество товаров. Поэтому данные надо закодировать так, чтобы их можно было представить в виде матрицы. Для кодированния данных используем TransactionEncoder

```
from mlxtend.preprocessing import TransactionEncoder
te = TransactionEncoder()
te_ary = te.fit(dataset).transform(dataset)
df = pd.DataFrame(te_ary, columns=te.columns_)
```

2. Выведите полученный dataframe и объясните, как стали представляться данные

```
print(df)
```

Ассоциативный анализ с использованием алгоритма Apriori

1. Применим алгоритм apriori с минимальным уровнем поддержки 0.3

```
from mlxtend.frequent_patterns import apriori
results = apriori(df, min_support=0.3, use_colnames=True)
results['length'] = results['itemsets'].apply(lambda x: len(x)) #добавление
размера набора
print(results)
```

Объясните полученный результат

2. Применим алгоритм apriori с тем же уровнем поддержки, но ограничим максимальный размер набора единицей

```
results = apriori(df, min_support=0.3, use_colnames=True, max_len=1)
```

```
print(results)
```

3. Применим алгоритм apriori и выведем только те наборы, которые имеют размер 2, а также количество таких наборов

```
results = apriori(df, min_support=0.3, use_colnames=True)
results['length'] = results['itemsets'].apply(lambda x: len(x))
results = results[results['length'] == 2]
print(results)
print('\nCount of result itemstes = ',len(results))
```

4. Посчитайте количество наборов при различных уровнях поддержки. Начальное значение поддержки 0.05, шаг 0.01. Постройте график зависимости количества наборов от уровня поддержки
5. Определите значение уровня поддержки при котором перестают генерироваться наборы размера 1,2,3, и.т.д. Отметьте полученные уровне поддержки на графике построенном в пункте 4
6. Построим датасет только из тех элементов, которые попадают в наборы размером 1 при уровне поддержки 0.38

```
results = apriori(df, min_support=0.38, use_colnames=True, max_len=1)
new_items = [ list(elem)[0] for elem in results['itemsets']]
new_dataset = [[elem for elem in all_data[all_data[1] == id][2] if elem in
new_items] for id in unique_id]
```

7. Приведите полученный датасет к формату, который можно обработать
8. Проведите ассоциативный анализ при уровне поддержки 0.3 для нового датасета. Опишите в чем сходства и различия
9. Проведите ассоциативный анализ при уровне поддержки 0.15 для нового датасета. Выведите все наборы размер которых больше 1 и в котором есть 'yogurt' или 'waffles'
10. Постройте датасет, из тех элементов, которые не попали в датасет в п. 6 и приведите его к удобному для анализа виду
11. Проведите анализ aprioti для полученного датасета
12. Напишите правило, для вывода всех наборов, в которых хотя бы два элемента начинаются на 's'
13. Напишите правило, для вывода всех наборов, для которых уровень поддержки изменяется от 0.1 до 0.25

17.4 Лабораторная работа № 4

Ассоциативный анализ

Цель:

Ознакомиться с методами ассоциативного анализа из библиотеки MLxtend

Выполнение:

Для выполнения лаб. работы необходимо установить библиотеку MLxtend

Для этого в терминале введите:

```
pip install mlxtend
```

Загрузка данных:

1. Загрузить датасет по ссылке: https://www.kaggle.com/irfanasrullah/groceries. Данные представлены в виде csv таблицы. Данные представляют собой информацию о купленных вместе товарах
2. Создать Python скрипт. Загрузить данные в датафрейм

```
all_data = pd.read_csv('groceries - groceries.csv')
print(all_data) #Видно, что датафрейм содержит NaN значения
```

3. Переформируем данные удалив все значения NaN

```
np_data = all_data.to_numpy()
np_data = [[elem for elem in row[1:] if isinstance(elem,str)] for row in
np_data]
```

4. Получим список всех уникальных товаров

```
unique_items = set()
for row in np_data:
for elem in row:
unique_items.add(elem)
```

5. Выведите список товаров, а также их количество

FPGrowth и FPMax

1. Преобразуем данные к виду, удобному для анализа

```
from mlxtend.preprocessing import TransactionEncoder
te = TransactionEncoder()
te_ary = te.fit(np_data).transform(np_data)
data = pd.DataFrame(te_ary, columns=te.columns_)
```

2. Проведем ассоциативный анализ используя алгоритм FPGrowth при уровне поддержки 0.03

```
from mlxtend.frequent_patterns import fpgrowth

result = fpgrowth(data, min_support=0.03, use_colnames = True)

print(result)
```

3. Проанализируйте получившиеся варианты. Определите минимальное и максимальное значения для уровня поддержки для набора из 1,2, и.т.д. объектов.
4. Проведите аналогичный анализ используя алгоритм FPMax
5. Сравните полученные результаты для FPGrowth и FPMax. Объясните в чем разница работы алгоритмов
6. Постройте гистограмму для каждого товара. Столбцы на гистограмме должны быть упорядочены по уменьшению частоты. Отобразите результат только для 10 самых встречаемых товаров. Как данная гистограмма коррелируют с результатами?
7. Преобразуем набор данных, чтобы он содержал ограниченный набор товаров

```
items = ['whole milk', 'yogurt', 'soda', 'tropical fruit', 'shopping bags',

'sausage',

'whipped/sour cream', 'rolls/buns', 'other vegetables', 'root

vegetables',

'pork', 'bottled water', 'pastry', 'citrus fruit', 'canned beer',

'bottled beer']

np_data = all_data.to_numpy()

np_data = [[elem for elem in row[1:] if isinstance(elem,str) and elem in

items] for row in np_data]
```

8. Проведите анализ FPGrowth и FPMax для нового набора данных. Проанализируйте, что изменилось
9. Постройте график изменения количества получаемых правил от уровня поддержки. На графике отдельно отобразите кривые для набора товаров 1, 2, и.т.д. Какие выводы можно сделать по данному графику?

Ассоциативные правила

1. Сформируем набор данных из определенных товаров и так, чтобы размер транзакции был 2 и более

```
np_data = all_data.to_numpy()

np_data = [[elem for elem in row[1:] if isinstance(elem,str) and elem in

items] for row in np_data]

np_data = [row for row in np_data if len(row) > 1]
```

2. Получим частоты наборов используя алгоритм FPGrowth

```
result = fpgrowth(data, min_support=0.05, use_colnames = True)
```

3. Проведем ассоциативный анализ

```
rules = association_rules(result, min_threshold = 0.3)

print(rules)
```

Объясните, что означает каждая колонка в полученных результатах.

4. Определите, на основе какой метрики проводится расчет
5. Проведите построение ассоциативных правил для различных метрик (значение min_threshold выберите такое, чтобы выводилось не менее 10 правил). Какой смысл несет каждая метрика?
6. Рассчитайте среднее значение, медиану и СКО для каждой из метрик.
7. Постройте граф для следующего анализа

```
rules = association_rules(result, min_threshold = 0.4, metric='confidence')
```

Каждая вершина графа должна отображать набор товаров. Граф должен быть ориентирован от антецедента к консеквенту. Ширина ребра должна отображать уровень

support, а подпись на ребре отображать confidence.

Для отрисовка графа можно использовать библиотеку NetworkX

8. Проанализируйте полученный граф, какую информацию можно из него извлечь?
9. Предложите свои способы визуализации полученных правил

17.5 Лабораторная работа № 5

Кластеризация (k-средних, иерархическая)

Цель:

Ознакомиться с методами кластеризации модуля Sklearn

Выполнение:

Загрузка данных:

1. Загрузить датасет по ссылке: https://archive.ics.uci.edu/ml/datasets/iris. Данные представлены в виде data файла. Данные представляют собой информацию о трех классах цветов
2. Создать Python скрипт. Загрузить данные в датафрейм

```
import pandas as pd

import numpy as np

data = pd.read_csv('iris.data',header=None)
```

K-means

1. Проведем кластеризацию методов k-средних

```
from sklearn.cluster import KMeans
k_means = KMeans(init='k-means++', n_clusters=3, n_init=15)
k_means.fit(no_labeled_data)
```

2. Получим центры кластеров и определим какие наблюдения в какой кластер попали

```
from sklearn.metrics.pairwise import pairwise_distances_argmin
k_means_cluster_centers = k_means.cluster_centers_
k_means_labels = pairwise_distances_argmin(no_labeled_data,
k_means_cluster_centers)
```

3. Построим результаты классификации для признаков попарно (1 и 2, 2 и 3, 3 и 4)

```
import matplotlib.pyplot as plt
f, ax = plt.subplots(1, 3)
colors = ['#4EACC5', '#FF9C34', '#4E9A06']
print(ax)
for i in range(3):
my_members = k_means_labels == i
cluster_center = k_means_cluster_centers[i]
for j in range(3):
ax[j].plot(no_labeled_data[my_members, j],
no_labeled_data[my_members, j+1], 'w',
markerfacecolor=colors[i], marker='o', markersize=4)
ax[j].plot(cluster_center[j], cluster_center[j+1], 'o',
markerfacecolor=colors[i],
markeredgecolor='k', markersize=8)
plt.show()
```

Опишите полученные результаты. По каким из признаков произошло наилучшее

разделение. Как влияет значение параметра n_init

4. Уменьшите размерность данных до 2 используя метод главных компонент и нарисуйте карту для всей области значений, на которой каждый кластер занимает определенную область со своим цветом (как это делать)

5. Исследуйте работу алгоритма k-средних при различных параметрах init. Сначала надо выполнить несколько раз с параметров 'random', затем для вручную выбранных точек

6. Определите наилучшее количество методом локтя

7. Проведите кластеризацию используя пакетную кластеризацию k-средних . В чем отличие от обычного метода k-средних. Постройте диаграмму рассеяния, на которой будут выделено точки, которые для разных методов попали в разные кластеры

Иерархическая кластеризация

1. Проведем иерархическую кластеризацию на тех же данных

```
from sklearn.cluster import AgglomerativeClustering

hier = AgglomerativeClustering(n_clusters=3, linkage='average')

hier = hier.fit(no_labeled_data)

hier_labels = hier.labels_
```

2. Отобразим результаты кластеризации

```
f, ax = plt.subplots(1, 3)

colors = ['#4EACC5', '#FF9C34', '#4E9A06']

for i in range(3):

my_members = hier_labels == i

for j in range(3):

ax[j].plot(no_labeled_data[my_members, j],

no_labeled_data[my_members, j+1], 'w',

markerfacecolor=colors[i], marker='o', markersize=4)

plt.show()
```

В чем отличия от метода k-средних

3. Проведите исследование для различного размера кластеров (от 2 до 5). Приведите полученные результаты

4. Нарисуйте дендограмму до уровня 6

5. Сгенерируйте случайные данные в виде двух колец

```
import random

import math

data1 = np.zeros([250,2])

for i in range(250):
```

```
r = random.uniform(1, 3)
a = random.uniform(0, 2 * math.pi)
data1[i,0] = r * math.sin(a)
data1[i,1] = r * math.cos(a)
data2 = np.zeros([500,2])
for i in range(500):
r = random.uniform(5, 9)
a = random.uniform(0, 2 * math.pi)
data2[i,0] = r * math.sin(a)
data2[i,1] = r * math.cos(a)
data = np.vstack((data1, data2))
```

6. Проведите иерархическую кластеризацию

```
hier = AgglomerativeClustering(n_clusters=2, linkage='ward')
hier = hier.fit(data)
hier_labels = hier.labels_
```

7. Выведите полученные результаты

```
my_members = hier_labels == 0
plt.plot(data[my_members, 0], data[my_members, 1], 'w', marker='o',
markersize=4,
color='red',linestyle='None')
my_members = hier_labels == 1
plt.plot(data[my_members, 0], data[my_members, 1], 'w', marker='o',
markersize=4,
color='blue',linestyle='None')
plt.show()
```

8. Исследуйте кластеризацию при всех параметрах linkage. Отобразите и обоснуйте полученные результаты. Для каких случаев, какой тип связи работает лучше всего.

17.6 Лабораторная работа № 6

Кластеризация (DBSCAN, OPTICS)

Цель:

Ознакомиться с методами кластеризации модуля Sklearn

Выполнение:

Загрузка данных:

1. Загрузить датасет по ссылке: https://www.kaggle.com/arjunbhasin2013/ccdata. Данные представлены в виде csv файла. Датасет содержит пропущенные значения
2. Создать Python скрипт. Загрузить данные в датафрейм, убрав столбец с метками и откинув наблюдения с пропущенными значениями

```
import pandas as pd
import numpy as np
data = pd.read_csv('CC GENERAL.csv').iloc[:,1:].dropna()
```

DBSCAN

1. Проведем кластеризацию методов k-средних

```
from sklearn.cluster import KMeans

k_means = KMeans(init='k-means++', n_clusters=3, n_init=15)k_means.fit(no_labeled_data)
```

2. Так как разные признаки лежат в разных шкалах, то стандартизируем данные

```
from sklearn import preprocessing

data = np.array(data, dtype='float')

min_max_scaler = preprocessing.StandardScaler()
scaled_data = min_max_scaler.fit_transform(data)
```

3. Проведем кластеризацию методов DBSCAN при параметрах по умолчанию. Выведем метки кластеров, количество кластеров, а также процент наблюдений, которые кластеризовать не удалось

```
    clustering = DBSCAN().fit(scaled_data)


    print(set(clustering.labels_))
    print(len(set(clustering.labels_)) - 1)
    print(list(clustering.labels_).count(-1) /
len(list(clustering.labels_)))
```

Опишите все параметры, которые принимает DBSCAN

4. Постройте график количества кластеров и процента не кластеризованных наблюдений в зависимости от максимальной рассматриваемой дистанции между наблюдениями. Минимальное значение количества точек образующих, кластер оставить по умолчанию
5. Постройте график количества кластеров и процента не кластеризованных наблюдений в зависимости от минимального значения количества точек, образующих кластер. Максимальную рассматриваемую дистанцию между наблюдениями оставьте по умолчанию
6. Определите значения параметров, при котором количество кластеров получается от 5 до 7, и процент не кластеризованных наблюдений не превышает 12%.
7. Понизьте размерность данных до 2 при используя метод главных компонент. Визуализируйте результаты кластеризации полученные в пункте 6 (метки должны быть получены на данных до уменьшения размерности).

OPTICS

1. Опишите параметры метода OPTICS , а также какими атрибутами он обладает
2. Найдите такие параметры метода OPTICS (*max_eps *и min_samples) при которых, чтобы получить результаты близкие к результатам DBSCAN из пункта 6 В чем отличия от метода OPTICS от метода DBSCAN
3. Визуализируйте полученный результат, а также постройге график достижимости (reachable plot)
4. Исследуйте работу метода OPTICS с использованием различных метрик (выберите не менее 5 метрик)

17.7 Лабораторная работа № 7

Классификация (Байесовскиеметоды, деревья)

Цель:

Ознакомиться с методами классификации модуля Sklearn

Выполнение:

Загрузка данных:

1. Загрузить датасет по ссылке: https://archive.ics.uci.edu/ml/datasets/iris. Данные представлены в виде data файла. Данные представляют собой информацию о трех классах цветов
2. Создать Python скрипт. Загрузить данные в датафрейм

```
import pandas as pd
import numpy as np

data = pd.read_csv('iris.data',header=None)
```

3. Выделим данные и их метки

```
X = data.iloc[:,:4].to_numpy()
labels = data.iloc[:,4].to_numpy()
```

4. Преобразуем тексты меток к числам

```
le = preprocessing.LabelEncoder()
Y = le.fit_transform(labels)
```

5. Разобьём выборку на обучающую и тестовую

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =
train_test_split(X, Y, test_size=0.5)
```

Байесовские методы

1. Проведем классификацию наблюдений наивным байесовским методом

```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
```

```
y_pred = gnb.fit(X_train, y_train).predict(X_test)

print((y_test != y_pred).sum()) #количество
наблюдений, который были неправильно определены
```

Опишите атрибуты данного классификатора

2. Используя функцию score() выведите точность классификации
3. Постройте график зависимости неправильно классифицированных наблюдений и точности классификации от размера тестовой выборки. Размер тестовой выборки изменяйте от 0.05 до 0.95 с шагом 0.05. Параметр random_state сделайте равным номеру своей зачетной книжки. Обоснуйте полученные результаты.
4. Проведите классификацию используя MultinomialNB, ComplementNB, BernoulliNB. Опишите особенности методов.

Классифицирующие деревья

1. Классификацию при помощи деревьях на тех же данных

```
from sklearn import tree

clf = tree.DecisionTreeClassifier()

y_pred = clf.fit(X_train, y_train).predict(X_test)
print((y_test != y_pred).sum())
```

2. Используя функцию score() выведите точность классификации
3. Выведите характеристики дерева, количество листьев и глубину, используя функции get_n_leaves и get_depth
4. Выведите изображение полученного дерева

```
import matplotlib.pyplot as plt

plt.subplots(1,1,figsize = (10,10))
tree.plot_tree(clf, filled = True)
plt.show()
```

Опишите полученный рисунок

5. Постройте график зависимости неправильно классифицированных наблюдений и точности классификации от размера тестовой выборки. Размер тестовой выборки изменяйте от 0.05 до 0.95 с шагом 0.05. Параметр random_state сделайте равным номеру своей зачетной книжки. Обоснуйте полученные результаты.

6. Исследуйте работу классифицирующего дерева при различных параметрах criterion, **splitter, max_depth, min_samples_split, min_samples_leaf**

17.8 Лабораторная работа № 8

Классификация (линейныйдискриминантный анализ, методопорных векторов)

Цель:

Ознакомиться с методами классификации модуля Sklearn

Выполнение:

Загрузка данных:

1. Загрузить датасет по ссылке: https://archive.ics.uci.edu/ml/datasets/iris. Данные представлены в виде data файла. Данные представляют собой информацию о трех классах цветов

2. Создать Python скрипт. Загрузить данные в датафрейм

```
import pandas as pd
import numpy as np

data = pd.read_csv('iris.data',header=None)
```

3. Выделим данные и их метки

```
X = data.iloc[:,:4].to_numpy()
labels = data.iloc[:,4].to_numpy()
```

4. Преобразуем тексты меток к числам

```
le = preprocessing.LabelEncoder()
Y = le.fit_transform(labels)
```

```
```

5. Разобьём выборку на обучающую и тестовую

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =
train_test_split(X, Y, test_size=0.5)
```

Линейный дискриминантный анализ

1. Проведем классификацию наблюдений используя LDA

```
X_train, X_test, y_train, y_test =
train_test_split(X, Y, test_size=0.5, random_state=0)

from sklearn.discriminant_analysis import
LinearDiscriminantAnalysi

sclf = LinearDiscriminantAnalysis()

y_pred = clf.fit(X_train, y_train).predict(X_test)
print((y_test != y_pred).sum()) #количество
наблюдений, который были неправильно определены
```

Опишите атрибуты и параметры данного классификатора

2. Используя функцию score() выведите точность классификации
3. Постройте график зависимости неправильно классифицированных наблюдений и точности классификации от размера тестовой выборки. Размер тестовой выборки изменяйте от 0.05 до 0.95 с шагом 0.05. Параметр random_state сделайте равным номеру своей зачетной книжки. Обоснуйте полученные результаты.
4. Опишите для чего нужна функция transform? Примените ее, и визуализируйте результаты.
5. Исследуйте работу классификатор при различных параметрах **solver, shrinkage.**
6. Задайте априорную вероятность классу с номером 1 равную 0.7, остальным классам задайте равные априорные вероятности. Как это сказалось на результате?

Метод опорных векторов

1. Классификацию при SVM на тех же данных

```
clf = svm.SVC()
y_pred = clf.fit(X_train, y_train).predict(X_test)
print((y_test != y_pred).sum())
print(clf.score(X, Y))
```

2. Используя функцию score() выведите точность классификации
3. Выведите следующую информацию

```
print(clf.support_vectors_)
print(clf.support_)
print(clf.n_support_)
```

Объясните, что отображают данные параметры, от чего зависят.

4. Постройте график зависимости неправильно классифицированных наблюдений и точности классификации от размера тестовой выборки. Размер тестовой выборки изменяйте от 0.05 до 0.95 с шагом 0.05. Параметр random_state сделайте равным номеру своей зачетной книжки. Обоснуйте полученные результаты.

5. Исследуйте работу метода опорных векторов при различных значениях **kernel, degree, max_iter**

6. Проведите исследование для методов NuSVC и LinearSVC. В чем их отличие от SVC?

Глава 18. Практические занятия

18.1 Практическое занятие № 1

Задание 1

Предположим X и Y две случайные переменные отражающие возраст и вес, соответственно. Рассмотрим случайную выборку из 20 наблюдений

X = (69, 74, 68, 70, 72, 67, 66, 70, 76, 68, 72, 79, 74, 67, 66, 71, 74, 75, 75, 76)

Y = (153, 175, 155, 135, 172, 150, 115, 137, 200, 130, 140, 265, 185, 112, 140, 150, 165, 185, 210, 220)

Необходимо:

Найти среднее, медиану и моду величины X

Найти дисперсию Y

Построить график нормального распределения для X

Найти вероятность того, что возраст больше 80

Найти двумерное мат. ожидания и ковариационную матрицу для этих двух величин

Определять корреляцию между X и Y

Построить диаграмму рассеяния, отображающая зависимость между возрастом и весом

Задание 2

Для следующего набора данных

	X_1	X_2	X_3
a	17	17	12
b	11	9	13
c	11	8	19

Рассчитайте ковариационную матрицу и обобщенную дисперсию

Задание 3

Даны два одномерных нормальных распределения N_a и N_b с мат. ожиданиями 4, 8 и СКО 1, 2 соответственно.

Для каждого из значения {5,6,7} определите какое из распределений сгенерировало значение с большей вероятностью.

Найди значение, которой могло быть сгенерировано обеими распределениями с равной вероятностью

18.2 Практическое занятие № 2

Задание 1

Рассмотрим данные

i	x_i
x_1	(4, 2.9)
x_2	(2.5, 1)
x_3	(3.5, 4)
x_4	(2, 2.1)

Есть ядро (функция сходства):

$$\boldsymbol{K}(\boldsymbol{x_i}, \boldsymbol{x_j}) = \|\boldsymbol{x_i} - \boldsymbol{x_j}\|^{\mathbf{2}}$$

Рассчитайте ядерную матрицу

Задание 2

Рассмотрим данные в виде матрицы D:

X_1	X_2
8	-20
0	-1
10	-19
10	-20
2	0

- Рассчитайте среднее **μ** и ковариационную матрицу **Σ** для матрицы D
- Рассчитайте собственные числа для матрицы **Σ**
- Какой “внутренний” размер данного набора данных?
- Рассчитай первый главный компонент

Если **μ** и **Σ** сверху характеризуют нормальное распределение, из которого были сгенерированы точки, нарисуйте ориентацию / протяженность 2-мерной функции нормальной плотности.

Задание 3

Для данных и ядра из первого задания найдите первую главную компоненту при нелинейном преобразовании для заданного ядра

18.3 Практическое занятие № 3

Задание №1

Дан набор данных:

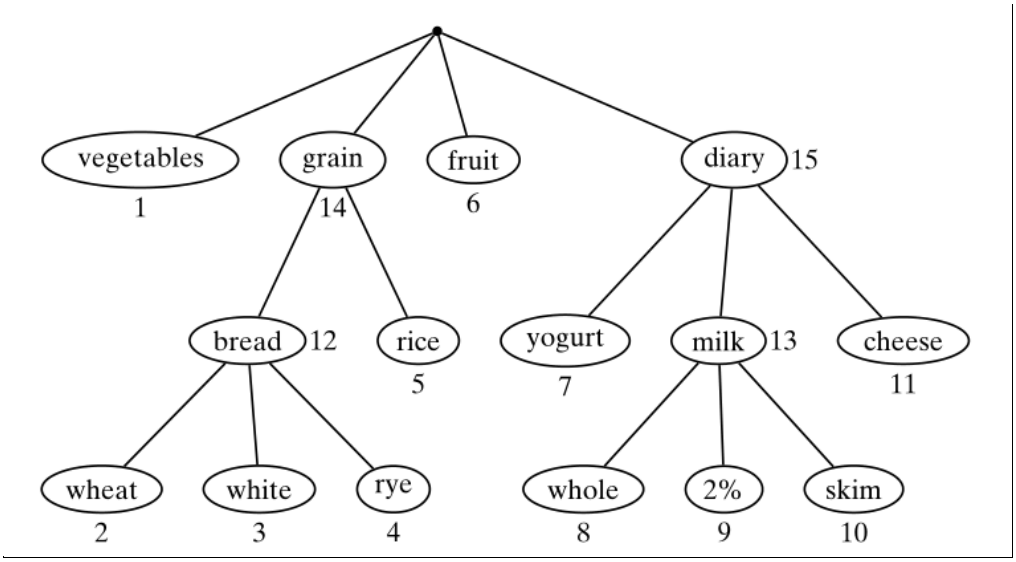
tid	itemset
t_1	$ABCD$
t_2	$ACDF$
t_3	$ACDEG$
t_4	$ABDF$
t_5	BCG
t_6	DFG
t_7	ABG
t_8	$CDFG$

Предположив, что минимальный уровень поддержки равен 3 / 8. Продемонстрируйте, как алгоритм Apriori перебирает данный набор данных.

Предположив, что минимальный уровень поддержки равен 2 / 8. Продемонстрируйте, как алгоритм FPGRowth перебирает данный набор данных.

Задание №2

На рисунке представлена классификация различных продуктов. Каждый лист дерева это конкретный продукт, внутренний узел дерева представляет категорию продукта более верхнего уровня.



Был получен следующий набор данных:

tid	itemset
1	2 3 6 7
2	1 3 4 8 11
3	3 9 11
4	1 5 6 7
5	1 3 8 10 11
6	3 5 7 9 11
7	4 6 8 10 11
8	1 3 5 8 11

Выполните следующие задание:

1. Каков размер области поиска наборов элементов, если ограничиваться только наборами, состоящими из простых элементов?

2. Предположив, что минимальный уровень поддержки = 7/8. Найдите все часто встречающиеся наборы элементов, состоящие только из элементов высокого уровня в таксономии. Имейте в виду, что если в транзакции появляется простой элемент, предполагается, что все его предки высокого уровня также присутствуют в транзакции.

18.4 Практическое занятие № 4

Задание №1

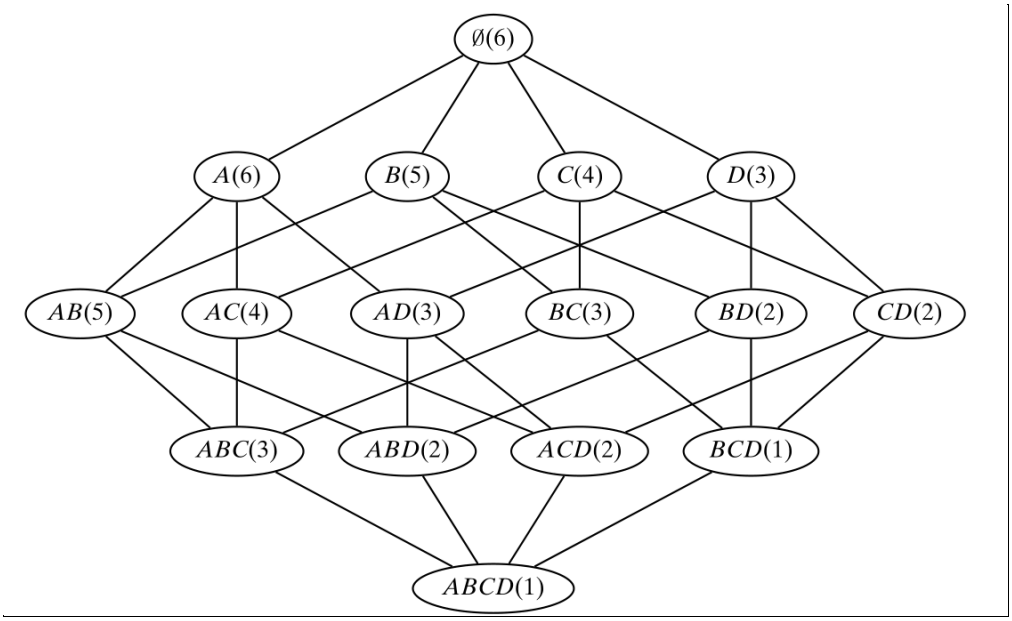
Дан набор данных:

Tid	Itemset
1	ACD
2	BCD
3	AC
4	ABD
5	$ABCD$
6	BCD

Найдите все минимальные генераторы для минимального уровня поддержки = 1.

Задание №2

Дана решетка наборов и их частоты.



Выполните следующие задание:

1. Выпишите список всех закрытых наборов (closed itemsets)
2. Является ли набор BCD выводимым? Является ли набор ABCD выводимым? Какие границы их поддержки?.

Задание №3

Даны последовательности

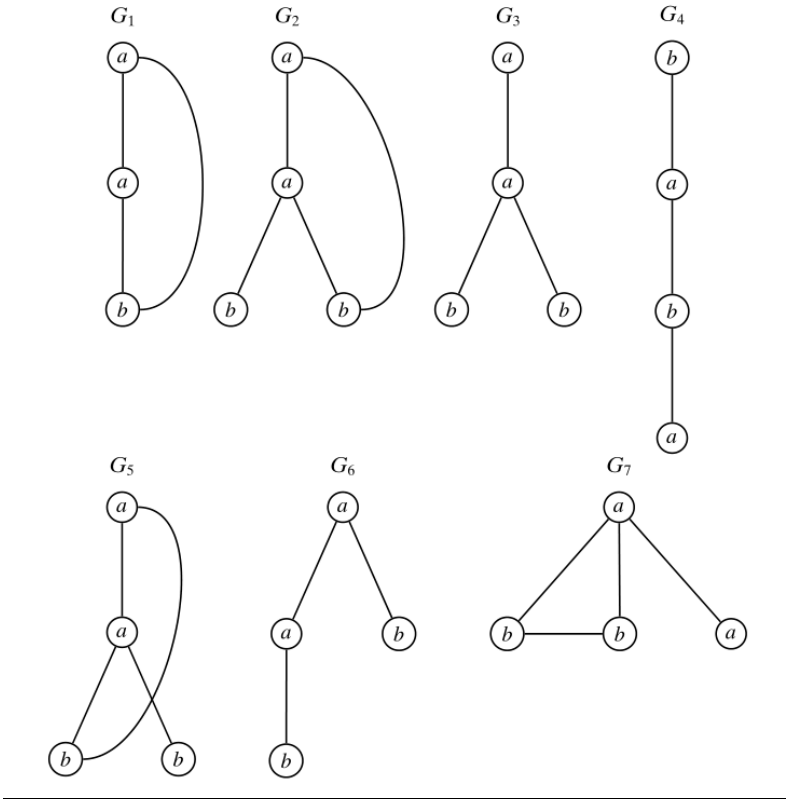
Id	Sequence
s_1	*AATACAAGAAC*
s_2	*GTATGGTGAT*
s_3	*AACATGGCCAA*
s_4	*AAGCGTGGTCAA*

Найдите все подпоследовательности в минимальным уровнем поддержки = 4

Для алфавита {A,C,G,T} посчитайте, сколько всего может быть разных последовательностей длины k

Задание №4

Дан набор графов. Разделите их на изоморфные группы



18.5 Практическое занятие № 5

Задание №1

Дан набор значений 2,4,10,12,3,20,30,11,25. Предположим количество кластеров k = 3, и выбраны начальные средние значения $m_1 = 2$, $m_2 = 4$, $m_3 = 6$. Покажите, какие кластеры будут после первой итерации алгоритма k-средних, и рассчитайте новые значения центров кластеров для следующей итерации.

Задание №2

Дан набор точек *x* и вероятности из принадлежности к кластерам C_1 и C_2.

x	$P(C_1\|x)$	$P(C_2\|x)$
2	0.9	0.1
3	0.8	0.1
7	0.3	0.7
9	0.1	0.9
2	0.9	0.1
1	0.8	0.2

Выполните следующие задание:

1. Найдите оценку максимального правдоподобия для средних μ1 и μ2
2. Предположим, что μ1 = 2, μ2 = 7 и σ1 = σ2 = 1. Найдите вероятности принадлежности точки x = 5 к кластерам C_1 и C_2 . Априорные вероятности каждого кластера P(C_1) = P(C_2) = 0.5 и P(x = 5) = 0.029

Задание №3

Даны категориальные данные размерности 5

Point	X_1	X_2	X_3	X_4	X_5
$\mathbf{x}_1$	1	0	1	1	0
$\mathbf{x}_2$	1	1	0	1	0
$\mathbf{x}_3$	0	0	1	1	0
$\mathbf{x}_4$	0	1	0	1	0
$\mathbf{x}_5$	1	0	1	0	1
$\mathbf{x}_6$	0	1	1	0	0

Близость двух наблюдений определяется через количество совпадений и несовпадений значений признаков. Допустим, что n_{11} количество признаков одновременной равных 1 для наблюдений x_i и x_j, и n_{10} количество признаков равных 1 для наблюдения x_i и в то же время равных 0 для наблюдения x_j. По аналогии определяются значения n_{01} and n_{00}:

	$\mathbf{x}_j$		
$\mathbf{x}_i$		1	0
	1	n_{11}	n_{10}
	0	n_{01}	n_{00}

Определим следующие метрики:

- Коэффициент простого совпадения

$$SMC(\mathbf{x}_i, \mathbf{x}_j) = \frac{n_{11}+n_{00}}{n_{11}+n_{10}+n_{01}+n_{00}}$$

- Коэффициент Жаккара

- Коэффициент Рассела и Рао

$$RC(\mathbf{x}_i, \mathbf{x}_j) = \frac{n_{11}}{n_{11}+n_{10}+n_{01}+n_{00}}$$

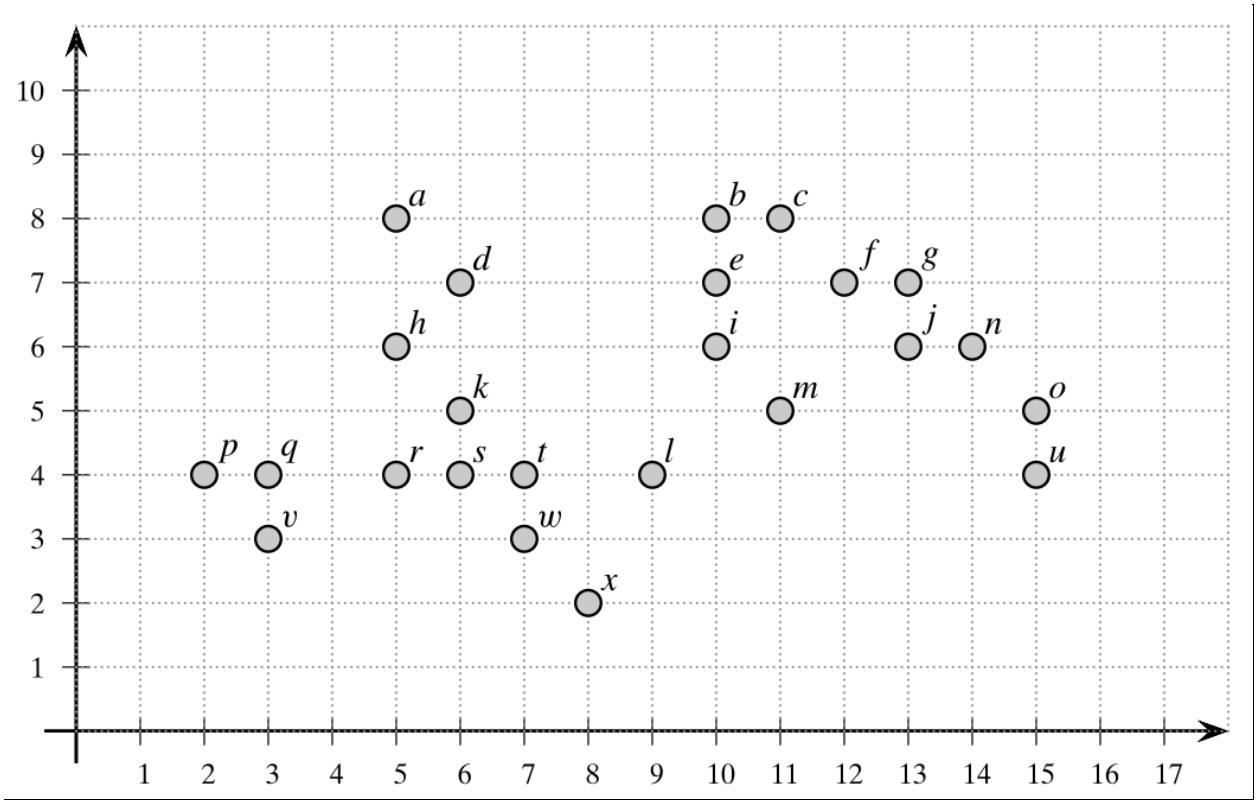
Постройте дендограммы полученные после иерархической кластеризации при следующих параметрах:

- Метод одиночной связи с метрикой RC
- Метод полной связи с метрикой SMC
- Невзвешенный центроидный метод с метрикой JC

18.6 Практическое занятие № 6

Задание №1

Дан рисунок



Допустим, что используется Евклидово расстояние, $\epsilon = 2$ и minPts = 3. Выполните следующие задачи:

1. Выпишите список всех основных точек
2. Покажите, является ли точка *a* прямо достижимой из точки *d*

3. Покажите, является ли точка *o* достижимой по плотности из точки *i*. Если нет, то покажите на какой точке цепочка построения пути оборвалась.

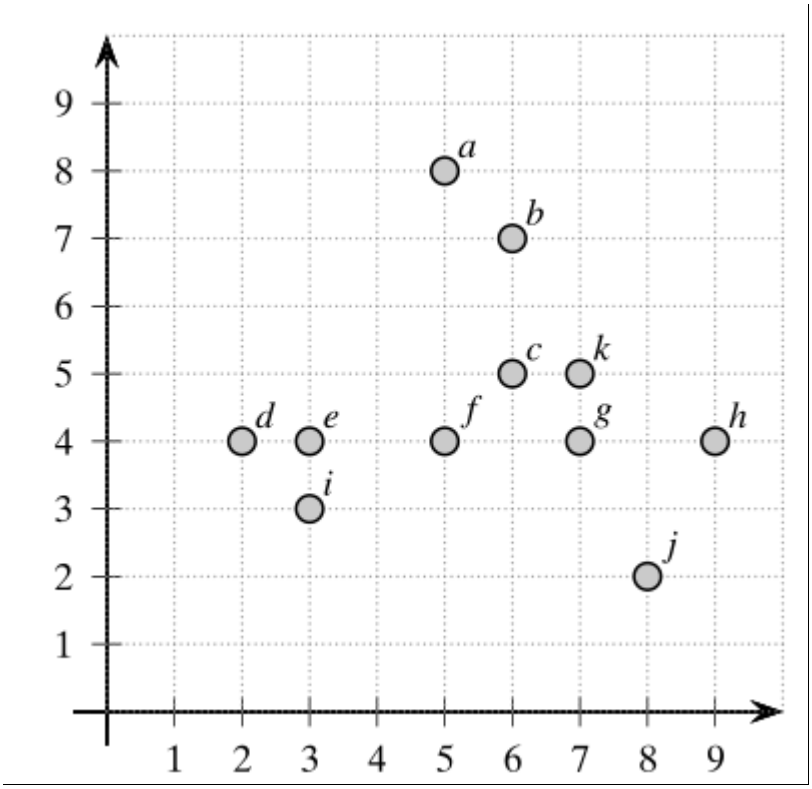
4. Покажите кластеры полученные алгоритмом DBSCAN и выпавшие точки.

Задание №2

Даны следующие метрики:

1. $$L_{\infty}(\mathbf{x},\mathbf{y}) = \max_{i=1}^{d}\{|x_i - y_i|\}$$
2. $$L_{\frac{1}{2}}(\mathbf{x},\mathbf{y}) = \left(\sum_{i=1}^{a}|x_i - y_i|^{\frac{1}{2}}\right)^2$$
3. $$L_{\min}(\mathbf{x},\mathbf{y}) = \min_{i=1}^{d}\{|x_i - y_i|\}$$
4. $$L_{pow}(\mathbf{x},\mathbf{y}) = \left(\sum_{i=1}^{d}2^{i-1}(x_i - y_i)^2\right)^{1/2}$$

Для данных представленных следующим рисунком



Используя метод DBSCAN проведите кластеризацию при следующих параметрах:

- $\epsilon = 2$ и minPts = 5 и метрика (1)
- $\epsilon = 4$ и minPts = 3 и метрика (2)
- $\epsilon = 1$ и minPts = 6 и метрика (3)
- $\epsilon = 4$ и minPts = 6 и метрика (3)

Для всех случаев построить кластеры и отобразить основные точки, достижимые по плотности точки и выпавшие точки

18.7 Практическое занятие № 7

Задание №1

Даны следующие данные

$\mathbf{x}_i$	a_1	a_2	a_3	Class
$\mathbf{x}_1$	T	T	5.0	Y
$\mathbf{x}_2$	T	T	7.0	Y
$\mathbf{x}_3$	T	F	8.0	N
$\mathbf{x}_4$	F	F	3.0	Y
$\mathbf{x}_5$	F	T	7.0	N
$\mathbf{x}_6$	F	T	4.0	N
$\mathbf{x}_7$	F	F	5.0	N
$\mathbf{x}_8$	T	F	6.0	Y
$\mathbf{x}_9$	F	T	1.0	N

Используя наивный байесовский классификатор определите класс точки (T,F,1.0)

Задание №2

Даны два класса c_1 and c_2 со следующими мат. ожиданиями и матрицами ковариации:

$$\boldsymbol{\mu}_1 = (1, 3) \qquad \boldsymbol{\mu}_2 = (5, 5)$$

$$\boldsymbol{\Sigma}_1 = \begin{pmatrix} 5 & 3 \\ 3 & 2 \end{pmatrix} \qquad \boldsymbol{\Sigma}_2 = \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix}$$

Классифицируйте точку (3,4) используя Байесовский вывод, предположив, что классы распределены по нормальному закону, и $P(c1) = P(c2) = 0.5$

Задание №3

Даны следующие данные

Point	Age	Car	**Risk**
$\mathbf{x}_1$	25	Sports	*L*
$\mathbf{x}_2$	20	Vintage	*H*
$\mathbf{x}_3$	25	Sports	*L*
$\mathbf{x}_4$	45	SUV	*H*
$\mathbf{x}_5$	20	Sports	*H*
$\mathbf{x}_6$	25	SUV	*H*

Постройте решающее дерево используя порог для чистоты (purity threshold) равным 100%.

В качестве критерия для разделения используйте энтропию. Классифицируйте наблюдение (Age=27,Car=Vintage)

18.8 Практическое занятие № 8

Задание №1

Даны следующие данные

i	$\mathbf{x}_i$	y_i
$\mathbf{x}_1$	(4,2.9)	1
$\mathbf{x}_2$	(3.5,4)	1
$\mathbf{x}_3$	(2.5,1)	−1
$\mathbf{x}_4$	(2,2.1)	−1

1. Рассчитайте μ_{+1} и μ_{-1} ,а так матрицу B - матрица межклассового разброса
2. Рассчитайте S_{+1} и S_{-1} , а также матрица S - матрица внутриклассового разброса
3. Найдите направление w, которое лучше всего дискриминирует классы
4. На направлении w, найдите точку, которая лучше всего делит классы

Задание №2

Даны следующие данные, принадлежащие к двум классам (y) и множители Лагранжа (α)

i	x_{i1}	x_{i2}	y_i	α_i
$\mathbf{x}_1$	4	2.9	1	0.414
$\mathbf{x}_2$	4	4	1	0
$\mathbf{x}_3$	1	2.5	−1	0
$\mathbf{x}_4$	2.5	1	−1	0.018
$\mathbf{x}_5$	4.9	4.5	1	0
$\mathbf{x}_6$	1.9	1.9	−1	0
$\mathbf{x}_7$	3.5	4	1	0.018
$\mathbf{x}_8$	0.5	1.5	−1	0
$\mathbf{x}_9$	2	2.1	−1	0.414
$\mathbf{x}_{10}$	4.5	2.5	1	0

- Рассчитайте уравнение гиперплоскости h(x) для метода опорных векторов
- Какое расстояние от точки x_6 до гиперплоскости h(x). Находится ли эта точка в границах опорных векторов?
- Классифицируйте точку (3, 3) используя гиперплоскость h(x)

Глава 19. Индивидуальное домашнее задание

В рамках выполнения ИДЗ необходимо на предложенном датасете решить связанную с ним задачу. При выполнении ИДЗ надо самостоятельно провести подготовку данных и выбрать подходящую модель МО.

Требования

Требования к разрабатываемой модели

- Модель должна быть разработана на языке Python. Разрешается использовать существующие библиотеки (например, SKLearn или mlxtend)
- Исходный код проекта должен быть в формате PEP8
- В исходном коде должны быть поясняющие комментарии
- Для апробации качества модели необходимо разбивать датасет на обучающее и тестовое подмножество. Разбиение необходимо повторить несколько раз, тем самым показав, что выбранная модель дает одинаковый результат вне зависимости от обучающего подмножества
- *Плюсом будет сравнение нескольких моделей МО*

Требования к отчету

- В отчете должно быть описание датасета, а также описание решаемой задачи
- В отчете должен быть проведен начальный анализ данных. Проведение статистического анализа, анализа на необходимость нормировки данных, обоснование применения определенного вида нормировки, и.т.д.
- В отчете должно быть обоснование выбора модели
- В отчете должно быть математическое описание выбранной модели
- В отчете необходимо отразить весь процесс разработки
- В конце отчета должен быть приведен анализ результирующей модели, а также перечислены возникшие проблемы (и как они были решены) и проблемы, которые решить не удалось. Плюсом будет предложение по улучшению модели.
- В отчете должно быть указано, кто в бригаде за что отвечал (написание отчета не является зоной ответственности)
- В приложении должен быть исходный код

Варианты

После названия датасета в скобках указана сложность задачи. Сложность задачи влияет на строгость оценивания ИДЗ. Для более сложных задач допускается получение не самого лучшего результата.

Также для каждого датасета указано количество задач.

1. **Zoo** (1) - 1 задача (http://archive.ics.uci.edu/ml/datasets/Zoo)

   Датасет булевых значений. Включает в себя информацию о 101 животном из зоопарка, которые относятся к 7 классам.

   Задача заключается в классификации животного по его описанию.

2. **Wine Quality** (1) - 1 задача (http://archive.ics.uci.edu/ml/datasets/Wine+Quality)

   Числовой датасет. Включает себя информацию о химических характеристик вина и его оценку по шкале от 0 до 10. Состоит из 4898 наблюдений.

   Задача заключается в предсказании качества вина на основе его химических характеристик. (Задача регрессии)

3. **Groceries dataset** (1) - 1 задача (https://www.kaggle.com/heeraldedhia/groceries-dataset)

   Датасет содержит id покупателя, дату покупки, и наименование купленного товара. Датасет содержит 38.8 тысяч записей.

   Задача заключается в построении ассоциативных правил и анализе полученных правил

4. **Lonosphere** (2) - 1 задача (https://www.kaggle.com/prashant111/ionosphere?select=ionosphere_data.csv)

   Датасет содержит информацию о сигнале и том, отразился ли он от ионосферы. Датасет состоит из 351 наблюдения.

   Задача кластеризации. Составить кластеры сигналов и провести анализ полученных кластеров и того, как они связаны с тем, что сигнал отразился или нет.

5. **Chocolate Bar Ratings** (2) - 1 задача (https://www.kaggle.com/rtatman/chocolate-bar-ratings)

   Датасет содержит информацию о шоколаде. Датасет содержит 1795 наблюдений. Один из признаков отвечает за рейтинг шоколада.

   Задача заключается в выделение значимых признаков и построении регрессионной модели для предсказания рейтинга.

6. **Mobile Price Classification** (2) - 1 задача (https://www.kaggle.com/iabhishekofficial/mobile-price-classification?select=test.csv)

   Датасет содержит информацию о характеристиках телефона и диапазоне цены (от 0 до 3). Предварительно датасет разбит на обучающую и тестовую выборку. Обучающая выборка содержит 2000 наблюдений, а тестовая 1000 наблюдений.

   Задача заключается в предсказании ценового диапазона телефона по его характеристикам (Задача классификации)

7. **Student Grade Prediction** (2) - 3 задачи (https://www.kaggle.com/dipam7/student-grade-prediction)

   Датасет содержит информацию о студентах и их оценках за 3 семестра. Датасет состоит из 395 наблюдений.

   Задача регрессии. Предсказать оценки студента на основе информации о нем

   Задача кластеризации. Провести кластеризацию студентов и анализ полученных кластеров. При проведении кластеризации информацию об оценках использовать не надо, но использовать ее при оценке кластеров.

   Задача ассоциативного анализа. На основе ассоциативного анализа найти зависимости между признаками и провести анализ полученных правил. Информацию об оценках использовать не надо.

8. **Gas Turbine CO and NOx Emission** (2) - 1 задача (http://archive.ics.uci.edu/ml/datasets/Gas+Turbine+CO+and+NOx+Emission+Data+Set)

   Датасет содержит измерения 11 показателей турбины и выбросов газа CO и NOx. Датасет состоит из 36733 наблюдений

   Задача регрессии. Предсказание значений выбросов на основе показателей турбины

9. **The Blog Authorship Corpus** (3) - 2 задачи (https://u.cs.biu.ac.il/~koppel/BlogCorpus.htm)

   Датасет текста. Включает в себя 681288 записей в блоге от 19320 людей, а также информацию о людях.

   Задача заключается в определении пола и возраста человека по тексту (Задача классификации)

   Задача ассоциативного анализа. Построение ассоциативных правил для текста с выделением того, какие слова в каких случаях и кем используются. Требует тщательной подготовки исходных данных.