

Математические методы машинного обучения

Кафедра Машинного Обучения и
Цифровой Гуманистике

Оглавление

	Page
Глава 1. Общие термины и обозначения	10
1.1. Линейные модели классификации и регрессии	14
1.2. Метод наименьших квадратов	15
1.3. Скользящий контроль	16
1.4. Определения и обозначения	16
1.4.1. Процедура скользящего контроля	17
1.4.2. Доверительное оценивание	17
1.4.3. Стратификация	18
1.5. Разновидности скользящего контроля	19
1.5.1. Полный скользящий контроль (complete CV)	19
1.5.2. Случайные разбиения	19
1.5.3. Контроль на отложенных данных (hold-out CV)	20
1.5.4. Контроль по отдельным объектам (leave-one-out CV)	20
1.5.5. Контроль по q блокам (q-fold CV)	21
1.5.6. Контроль по r×q блокам (r×q-fold CV)	21
1.6. Скользящий контроль в задачах прогнозирования	21
1.6.1. Контроль с нарастающей длиной обучения	22
1.6.2. Контроль с фиксированной длиной обучения	22
1.7. Недостатки скользящего контроля	22
1.8. Применение скользящего контроля	23
1.9. Вероятность переобучения	32
1.10. Статистические критерии в машинном обучении	38
1.10.1. Теория	38
1.10.2. Задачи	39
Глава 2. Линейные методы классификации и регрессии	40
2.1. О регуляризации	40
2.1.1. Гауссовский и лапласовский регуляризаторы	41
2.1.2. Вероятностная интерпретация регуляризации	42
2.1.3. Задачи	43
2.2. Метод наименьших квадратов (МНК) в общем случае	53
2.2.1. Про линейную регрессию и МНК	53
2.2.2. МНК в общем случае	53
2.2.3. Проблемы и ограничения МНК	53
2.2.4. Задачи	54
2.3. Вероятностные функции потерь	56
2.3.1. Принцип максимума правдоподобия	56

2.3.2. Связь правдоподобия и аппроксимации эмпирического риска	57
2.3.3. Вероятностный смысл регуляризации	57
2.3.4. Задачи	58
2.4. Методы оценки и проверки моделей	59
2.5. Многоклассовая классификация	62
2.5.1. Один против всех (one-versus-all)	62
2.5.2. Все против всех (all-versus-all)	64
2.5.3. Многоклассовая логистическая регрессия	66
 Глава 3. Основы нейросетевых моделей	 68
3.1. Полносвязная нейронная сеть (Персепtron).	68
3.1.1. Модель нейрона МакКаллока-Питтса	68
3.1.2. Реализация логических функций с помощью нейрона	69
3.1.3. Область применимости многослойных нейронных сетей	71
3.1.4. Полносвязная нейронная сеть	71
3.1.5. Градиентный спуск	72
3.1.6. Стохастический градиентный спуск	73
3.1.7. Метод обратного распространения ошибок (BackProp).	74
3.1.8. Алгоритм применения метода обратного распространения ошибки в стохастическом градиентном спуске.	77
3.2. Функции активации ReLU и PReLU. Проблема «паралича» сети.	78
3.2.1. Про функции активации	78
3.2.2. ReLU (Rectified Linear Unit), проблема «паралича» сети	79
3.2.3. PReLU (Parametric Rectified Linear Unit)	79
3.2.4. Задачи	79
3.3. Drop Out	81
3.3.1. Теоретические сведения	81
3.3.2. Реализация	82
3.3.3. Задачи	82
 Глава 4. Метрические методы классификации и регрессии	 84
4.1. Задачи	88
4.1.1. Задача 1	88
4.1.2. Ответ:	88
4.1.3. Задача 2	89
4.1.4. Ответ:	89
4.1.5. Задача 3	89
4.1.6. Ответ:	89
4.2. Формула Надарай-Ватсона.	90
4.3. Влияние выбора метрики на качество работы kNN	93
4.3.1. Сущность метода k -ближайших соседей	93

4.3.2. Популярные метрики расстояния	93
4.3.3. Влияние выбора метрики на качество работы модели	94
4.3.4. Примеры задач	95
4.3.5. Заключение	95
4.4. Более быстрые оптимизации kNN.	96
4.4.1. BallTree	96
4.4.2. KD-Tree	97
4.4.3. Примеры задач	98
4.5. Расстояние Махаланобиса в метрических методах классификации и регрессии	99
4.5.1. Определение:	100
4.5.2. Преимущества расстояния Махаланобиса:	100
4.5.3. Недостатки расстояния Махаланобиса:	100
4.5.4. Практические аспекты использования:	101
4.5.5. Применение в методах классификации и регрессии	101
4.5.6. Задачи	101
4.6. Профиль компактности и оценка обобщающей способности	105
4.6.1. CCV	105
4.6.2. Профиль компактности	105
4.6.3. CCV для метода 1NN	105
4.6.4. Пример профилей компактности	106
4.6.5. Задачи	107
4.6.6. Минимизация CCV	108
4.6.7. Random projection trees - <i>Annoy</i>	108
4.7. Отбор эталонных объектов	112
4.8. Компактность и профиль компактности	112
4.9. Метод окна Парзена.	116
4.10. Метод потенциальных функций	119
4.10.1. Подбор параметров	120
4.10.2. Преимущества и недостатки	120
4.10.3. Задачи для самопроверки	121
 Глава 5. Метод опорных векторов	123
5.1. SVM-классификация	123
5.1.1. Постановка задачи	123
5.1.2. Математическая формализация	123
5.1.3. Условия разделимости	123
5.1.4. Оптимационная задача	123
5.1.5. Двойственная задача	124
5.1.6. Ядра	124
5.1.7. Дискриминантная функция в ядовом пространстве	124

5.1.8. Мягкие граници	125
5.1.9. Задача 1	125
5.1.10. Задача 2	126
5.1.11. Задача 3	127
5.2. SVM-регрессия	128
5.2.1. Постановка задачи	128
5.2.2. Прямая задача	128
5.2.3. Преобразование задачи	129
5.2.4. Метод Лагранжа	129
5.2.5. Условия оптимальности	129
5.2.6. Двойственная задача	130
5.2.7. Построение финальной модели	130
5.2.8. Выбор ядра	131
5.2.9. Задача 1	131
5.2.10. Задача 2	133
5.2.11. Задача 3	133
1-norm SVM (LASSO SVM)	134
Сравнение L_2 и L_1 регуляризации	135
Doubly Regularized SVM (Elastic Net SVM)	135
Elastic Net Analysis	136
Support Features Machine (SFM)	137
Relevance Features Machine (RFM)	138
Задачи	138
Задача 1	138
Ответ:	138
Задача 2	139
Ответ:	139
Задача 3	139
Ответ:	140
 Глава 6. Метод главных компонент	141
Введение в метод главных компонент(PCA)	141
Задачи на использование метода главных компонент	142
Задача 1: Вклад признаков в главные компоненты	142
Задача 2: Ошибка "реконструкции"PCA	143
Задача 3: Построить критерий D-оптимальности для выбора лучших k-компонент	145
 Глава 7. Нелинейные модели машинного обучения	147
Backfitting	150
Алгоритм backfitting	151

Задача	151
Метод наименьших квадратов с итеративным пересчётом весов (IRLS)	152
Взвешенные метод наименьших квадратов	152
Алгоритм IRLS	153
Задачи	153
 Глава 8. Обобщенные линейные модели	155
 Глава 9. Нестандартные функции потерь	156
 Глава 10. Критерии выбора моделей	172
 Глава 11. Методы отбора признаков	178
Качество классификации	178
Методы отбора признаков: полный перебор, алгоритм Add	182
Генерация новых признаков	182
Полный перебор	183
Жадный алгоритм Add	183
Задачи	185
Методы преобразования категориальных признаков	186
 Глава 12. Логические методы классификации	191
Бинаризация признаков	191
Бинаризация количественных признаков	191
Разбиение диапазона значений признака на зоны	192
Жадный алгоритм слияния зон	192
Задачи	193
Взвешенное голосование правил	195
Принцип голосования	195
Алгоритм взвешенного голосования	195
Проблема диверсификации правил	196
Отказы от классификации	197
Задачи	198
Алгоритм ТЭМП	199
Алгоритм 1.9. Построение списка информативных конъюнкций методом поиска в ширину (алгоритм ТЭМП)	200
Алгоритм 1.9: Построение списка информативных конъюнкций методом поиска в ширину	200
Достоинства алгоритма ТЭМП	201
Недостатки алгоритма ТЭМП	201
Задачи	202

Глава 13. Поиск ассоциативных правил	203
Алгоритм FP-Growth	203
Построение FP-дерева	203
Построение условного FP-дерева	204
Построение списка ассоциативных правил	205
Задачи для практики	205
Алгоритм APriory	207
Свойство антимонотонности	207
Поиск частых наборов	208
Выделение ассоциативных правил.	208
Задачи для практики	209
Задача автоматического выделения терминов: алгоритм TopMine, UDPipe, модель PLSA.	211
Глава 14. Композиции классификаторов	217
Mixture of Experts (Смесь экспертов)	217
Задачи и решения	217
Глава 15. Методы бустинга	220
Градиентный бустинг	223
Что такое градиентный бустинг и его история	223
Объяснение с лекции К.В.Воронцова	224
Классическая аналогия с гольфом	225
Применимость градиентного бустинга	226
Задачи для практики	226
Полезные ссылки	228
CatBoost	228
LightGBM	235
Анализ сложности градиентного бустинга над решающими деревьями	235
Gradient-based One-Side Sampling (GOSS)	235
Exclusive Feature Bundling (EFB)	237
Задачи	237
Глава 16. Байесовская теория классификации	239
Глава 17. Методы кластеризации	249
Критерии качества кластеризации	249
Критерии, не требующие разметки выборки	249
Критерии, требующие разметки выборки	250
Различия и выбор метрик качества кластеризации	253
Задачи на понимание	253
DBSCAN	254

Примечание о HDBSCAN	256
Задачи	256
Простые эвристические методы частичного обучения	262
Постановка задачи	262
Self-training	264
Co-training	265
Co-learning	266
Задачи	266
Глава 18. Обучение на неразмеченных данных	269
Постановка	269
Суперпозиция	269
Постановка задачи SimCLR	269
Вопросы	271

Глава 1

Общие термины и обозначения

Переобучение. Борьба с переобучением

Переобучение (overfitting) - это явление, при котором модель машинного обучения показывает существенно лучшие результаты на обучающих данных по сравнению с тестовыми. Формально переобученность определяется как разность между ошибками на контрольной и обучающей выборках:

$$\delta(\mu, X^l, X^k) = \nu(\mu(X^l), X^k) - \nu(\mu(X^l), X^l)$$

где:

- μ - метод обучения
- X^l - обучающая выборка
- X^k - контрольная выборка
- ν - частота ошибок

О наличии переобучения говорят, когда $\delta > \varepsilon$ для некоторого порогового значения ε .

Признаки переобучения

1. Большая разница между качеством работы на обучающей и тестовой/валидационной выборках.
2. Продолжение улучшения качества на обучении при ухудшении на валидации в процессе обучения.
3. Слишком много параметров модели
4. Слишком большие значения весов
5. Избыточное количество признаков

Основные методы борьбы с переобучением

1. **Регуляризация** - добавление штрафа за сложность модели:

L1-регуляризация:

$$J(w) = L(w) + \lambda \sum_{i=1}^n |w_i|$$

L2-регуляризация:

$$J(w) = L(w) + \lambda \sum_{i=1}^n w_i^2$$

где $L(w)$ - исходная функция потерь, λ - коэффициент регуляризации.

2. Использование кросс-валидации Кросс-валидация - оценка качества и подбор гиперпараметров на разных разбиениях выборки:

- K-fold: Выборка разбивается на k равных частей. На каждой итерации одна часть используется для валидации, остальные k-1 частей - для обучения. Итоговая оценка усредняется по всем разбиениям:
- Leave-one-out (LOO): Частный случай k-fold CV при k = n, где n - размер выборки. На каждой итерации только один объект используется для валидации. Этот метод дает несмещенную оценку ошибки, но требует большого числа итераций обучения.

3. Уменьшение сложности модели:

- Сокращение числа параметров
- Отбор значимых признаков
- Упрощение архитектуры

4. Увеличение объема обучающей выборки

Оценка эффективности борьбы с переобучением

1. Мониторинг динамики ошибок на обучающей и валидационной выборках.
2. Расчет величины переобученности δ .
3. Оценка значимости признаков и параметров модели.
4. Анализ сложности модели относительно размера выборки.

Задача 1

В процессе обучения нейронной сети для задачи бинарной классификации получены следующие значения ошибок по эпохам обучения:

Эпоха	Ошибка на обучении	Ошибка на валидации
1	0.40	0.42
5	0.25	0.30
10	0.15	0.25
15	0.08	0.28
20	0.03	0.35
25	0.01	0.45

Определите оптимальное количество эпох обучения для минимизации эффекта переобучения и обоснуйте свой выбор. Рассчитайте величину переобученности δ для последней эпохи.

Решение: Для определения оптимального количества эпох проанализируем динамику ошибок:

1. До 10-й эпохи обе ошибки монотонно убывают, что говорит о нормальном процессе обучения.

2. После 10-й эпохи ошибка на обучении продолжает уменьшаться, но ошибка на валидации начинает расти, что является явным признаком переобучения.
3. Оптимальное количество эпох - 10, так как при дальнейшем обучении начинается переобучение

Величина переобученности на последней эпохе:

$$\delta = \nu(X^k) - \nu(X^l) = 0.45 - 0.01 = 0.44$$

Ответ: 10 эпох; $\delta = 0.44$

Задача 2

Дана выборка из 8 объектов с целевой переменной $y \in \{0, 1\}$ и одним признаком x :

$$(x_1 = 1, y_1 = 0), (x_2 = 2, y_2 = 0), (x_3 = 3, y_3 = 1), (x_4 = 4, y_4 = 1), \\ (x_5 = 5, y_5 = 1), (x_6 = 6, y_6 = 1), (x_7 = 7, y_7 = 0), (x_8 = 8, y_8 = 0)$$

Модель представляет собой пороговое правило вида:

$$a(x) = [x \geq t]$$

где t - порог, $[...]$ - индикатор.

Используя метод полного скользящего контроля (leave-one-out cross-validation), найдите оптимальное значение порога t , минимизирующее среднюю ошибку на контроле.

Решение: 1) При leave-one-out каждый объект по очереди становится контрольным, а остальные - обучающими.

2) Возможные значения порога t следует искать между соседними значениями признака x , принадлежащими разным классам. Таким образом, получаем множество потенциальных порогов: $t \in \{1.5, 2.5, 3.5, 4.5, 5.5, 6.5, 7.5\}$

3) Рассмотрим подробно процесс для порога $t = 2.5$. Запишем пороговое правило:

$$a(x) = [x \geq 2.5]$$

Для первого объекта ($x_1 = 1, y_1 = 0$):

$$a(1) = [1 \geq 2.5] = 0$$

Правильный ответ $y_1 = 0$, следовательно ошибка на первом объекте:

$$e_1 = |y_1 - a(x_1)| = |0 - 0| = 0$$

Для второго объекта ($x_2 = 2, y_2 = 0$):

$$a(2) = [2 \geq 2.5] = 0$$

$$e_2 = |y_2 - a(x_2)| = |0 - 0| = 0$$

Для третьего объекта ($x_3 = 3, y_3 = 1$):

$$a(3) = [3 \geq 2.5] = 1$$

$$e_3 = |y_3 - a(x_3)| = |1 - 1| = 0$$

Аналогично для остальных объектов получаем вектор ошибок:

$$e = (0, 0, 0, 0, 0, 0, 1, 1)$$

Средняя ошибка для порога $t = 3.5$ равна:

$$\text{err}(2.5) = \frac{1}{8} \sum_{i=1}^8 e_i = \frac{0 + 0 + 0 + 0 + 0 + 0 + 1 + 1}{8} = \frac{2}{8} = 0.25$$

Проведем аналогичные вычисления для остальных порогов и получим:

Значение порога t	Средняя ошибка $\text{err}(t)$	Количество ошибок
1.5	0.375	3
2.5	0.250	2
3.5	0.375	3
4.5	0.500	4
5.5	0.625	5
6.5	0.750	6
7.5	0.625	5

Ответ: Оптимальное значение порога $t = 2.5$, что дает среднюю ошибку 0.25.

Задача 3

При обучении модели классификации изображений исследователь записывал значения accuracy каждые 3 эпохи для двух различных архитектур нейронных сетей (A и B). Размер обучающей выборки: 500 изображений.

Эпоха	Модель А		Модель В	
	Обучение	Валидация	Обучение	Валидация
3	0.65	0.62	0.70	0.68
6	0.75	0.70	0.82	0.75
9	0.85	0.72	0.95	0.69
12	0.95	0.68	0.98	0.63

Требуется:

Определить, какая из моделей показывает более сильное переобучение. Обосновать ответ. Рассчитать величину переобученности δ для обеих моделей

на последней эпохе. Для модели с более сильным переобучением указать оптимальное количество эпох обучения.

Решение: Давайте проанализируем поведение каждой модели. Для модели А: На первых эпохах наблюдается нормальное обучение - точность растет как на обучающей, так и на валидационной выборке. После 6-й эпохи начинается расхождение: точность на обучении продолжает расти, а на валидации начинает падать. К 12-й эпохе разрыв становится существенным. Для модели В: Начальное обучение идет быстрее, чем у модели А. Однако уже после 6-й эпохи наблюдается резкое расхождение между обучающей и валидационной точностью. К 9-й эпохе точность на обучении достигает 0.95, но на валидации падает до 0.69. Рассчитаем величину переобученности δ на последней эпохе:

$$\text{Модель А: } \delta_A = 0.68 - 0.95 = -0.27 \quad \text{Модель В: } \delta_B = 0.63 - 0.98 = -0.35$$

Модель В показывает более сильное переобучение, потому что:

Больше абсолютная величина δ на последней эпохе (-0.35 против -0.27) Более быстрая скорость расхождения метрик после 6-й эпохи Более высокая точность на обучении при более низкой точности на валидации

Для модели В оптимальным является остановка на 6-й эпохе, так как:

Достигнута хорошая точность на валидации (0.75) Разрыв между обучением и валидацией еще не критический После этой эпохи начинается явное переобучение

Ответ: Модель В показывает более сильное переобучение ($\delta_B = -0.35$), в то время как ($\delta_A = -0.27$). Оптимальное количество эпох для модели В - 6 эпох, что обеспечивает наилучшее качество на валидации (0.75) без явного переобучения.

1.1. Линейные модели классификации и регрессии

Обе линейные модели описываются следующим образом. На пространстве X объектов заданы числовые признаки $f_1, \dots, f_n: X \rightarrow \mathbb{R}$. По заданному вектору $w \in \mathbb{R}^n$ весов определяются предсказания объектов

$$a(x, w) := \sum_{i=1}^n w_i f_i(x) = w^T f(x)$$

в задаче регрессии; в задаче классификации от этого выражения берётся знак. При фиксированной (но неизвестной) таргет-функции $y: X \rightarrow Y$ определяется функция потерь $L(x, w)$, которая в задаче классификации равна индикатору того, что $a(x, w) \neq y(x)$, а в задаче регрессии равна

$$L(x, w) = |a(x, w) - y(x)| \quad \text{или} \quad L(x, w) = |a(x, w) - y(x)|^2,$$

или любому другому симметричному неотрицательному функционалу. Далее в обеих моделях нужно решить задачу оптимизации

$$\sum_{i=1}^N L(x_i, w) \rightarrow \min_w,$$

где x_i суть элементы выборки размера N . То есть, неформально, нужно придумать линейную модель, которая по $x \in X$ восстанавливает $y(x) \in Y$.

1.2. Метод наименьших квадратов

В методе наименьших квадратов в задаче линейной регрессии необходимо решить задачу минимизации функционала

$$\|w^T f(X) - Y\|_2^2 \rightarrow \min_w,$$

где Y это вектор из таргетов $y_i = y(X_i)$, X это вектор наблюдений, а $f(X)$ это матрица признаков размера $n \times N$. Из линейной алгебры известно, что решением задачи минимизации является вектор

$$\hat{w} = (f(X)f(X)^T)^{-1}f(X)Y.$$

Известно, что он является несмешённой оценкой параметра w , и, кроме того, наилучшей оценкой этого параметра в классе всех линейных оценок вида $A(X)Y$ с квадратичной функцией потерь.

В гауссовской линейной модели дополнительно известно, что

$$Y \sim N(w^T f(X), \sigma^2 I_{N \times N}).$$

Тогда оценка \hat{w} является оптимальной оценкой вектора w в среднеквадратичном подходе в классе несмешённых оценок.

Задача 1

Дан набор точек в \mathbb{R}^3 , отмеченных ± 1 . Необходимо описать задачу проведения разделяющей плоскости, проходящей через начало координат, как задачу линейной регрессии.

Задача 2

В задаче линейной регрессии получить несмешённую оценку ошибки σ^2 .

Задача 3

В гауссовой линейной модели найдите оптимальную оценку параметра $w_1 + \dots + w_n$ и её распределение.

1.3. Скользящий контроль

Скользящий контроль (или кросс-проверка, кросс-валидация, англ. cross-validation, CV) — это метод эмпирической оценки обобщающей способности алгоритмов, когда они обучаются на примерах из данных.

Метод основан на использовании некоторого числа разбиений исходной выборки на два подмножества: обучающей и контрольной подвыборок. Для каждого разбиения выполняется обучение алгоритма на обучающей подвыборке, а затем рассчитывается средняя ошибка на контрольной подвыборке. Итоговой оценкой скользящего контроля является среднее значение ошибки по всем контрольным подвыборкам.

При условии независимости выборки средняя ошибка кросс-валидации даёт несмещённую оценку вероятности ошибки. Это преимущество выделяет её по сравнению со средней ошибкой на обучающей выборке, которая может быть смещена (занизена) из-за эффекта переобучения.

Скользящий контроль является стандартным методом для тестирования и сравнения алгоритмов классификации, регрессии и прогнозирования.

1.4. Определения и обозначения

Рассмотрим задачу обучения с учителем.

Пусть X — множество, содержащее описания объектов, а Y — множество возможных ответов.

Предположим, что имеется конечная выборка прецедентов $X^L = \{(x_i, y_i)\}_{i=1}^L \subset X \times Y$.

Задан алгоритм обучения — отображение μ , которое сопоставляет произвольной конечной выборке прецедентов X^m некоторую функцию (алгоритм) $a : X \rightarrow Y$.

Качество алгоритма a оценивается по произвольной выборке прецедентов X^m с использованием функционала качества $Q(a, X^m)$. Для процедуры скользящего контроля способ вычисления данного функционала может варьироваться, однако обычно он аддитивен по элементам выборки:

$$Q(a, X^m) = \frac{1}{m} \sum_{x_i \in X^m} \mathcal{L}(a(x_i), y_i),$$

где $\mathcal{L}(a(x_i), y_i)$ — неотрицательная функция потерь, показывающая величину ошибки алгоритма $a(x_i)$ при истинном ответе y_i .

1.4.1. Процедура скользящего контроля

Рассмотрим выборку X^L , которая разбивается на N различных способов на две непересекающиеся подвыборки: $X^L = X_n^m \cup X_n^k$, где X_n^m — обучающая подвыборка размера m , а X_n^k — контрольная подвыборка длины $k = L - m$, при этом $n = 1, \dots, N$ обозначает номер разбиения.

Для каждого разбиения n рассчитывается алгоритм $a_n = \mu(X_n^m)$ и вычисляется значение функционала качества $Q_n = Q(a_n, X_n^k)$. Среднее арифметическое значений Q_n по всем разбиениям называют оценкой по методу скользящего контроля:

$$CV(\mu, X^L) = \frac{1}{N} \sum_{n=1}^N Q(\mu(X_n^m), X_n^k).$$

Разные методы скользящего контроля отличаются как видами функционала качества, так и способами разбиения выборки.

1.4.2. Доверительное оценивание

Кроме среднего значения качества на контроле, строятся также доверительные интервалы.

Непараметрическая оценка доверительного интервала.

Строится вариационный ряд значений $Q_n = Q(a_n, X_n^k)$, где $n = 1, \dots, N$:

$$Q^{(1)} \leq Q^{(2)} \leq \dots \leq Q^{(N)}.$$

Утверждение 1. Если разбиения осуществлялись случайно, независимо и равновероятно, то с вероятностью $\eta = \frac{t}{N+1}$ значение случайной величины $Q(a(X^m), X^k)$ не превосходит $Q^{(N-t+1)}$.

Следствие 1. Значение случайной величины $Q(a(X^m), X^k)$ не превосходит $Q^{(N)}$ с вероятностью $\eta = \frac{1}{N+1}$.

В частности, для получения верхней оценки с надёжностью 95% достаточно взять $N = 20$ разбиений.

Утверждение 2. Если разбиения осуществлялись случайно, независимо и равновероятно, то с вероятностью $\eta = \frac{2t}{N+1}$ значение случайной величины $Q(a(X^m), X^k)$ не выходит за границы доверительного интервала $[Q^{(t)}, Q^{(N-t+1)}]$.

Следствие 2. Значение случайной величины $Q(a(X^m), X^k)$ не выходит за границы вариационного ряда $[Q^{(1)}, Q^{(N)}]$ с вероятностью $\eta = \frac{2}{N+1}$.

В частности, для получения двусторонней оценки с надёжностью 95% достаточно взять $N = 40$ разбиений.

Параметрические оценки доверительного интервала основываются на априорном предположении о виде распределения случайной величины $Q(a(X^m), X^k)$. Если априорные предположения не выполняются, доверительный интервал может оказаться сильно смещённым. В частности, если предположения о нормальности распределения не выполнены, то нельзя использовать стандартное «правило двух сигм» или «трёх сигм». Джон Лангфорд в своей диссертации [Langford2002] указывает на распространённую ошибку, когда правило двух сигм применяется к функционалу частоты ошибок, имеющему на самом деле биномиальное распределение. Однако биномиальным распределением в общем случае тоже нельзя пользоваться, поскольку в результате обучения по случайным подвыборкам X^m вероятность ошибки алгоритма $a(X^m)$ оказывается случайной величиной. Следовательно, случайная величина $Q(a(X^m), X^k)$ описывается не биномиальным распределением, а (неизвестной) смесью биномиальных распределений. Апроксимация смеси биномиальным распределением может приводить к ошибочному сужению доверительного интервала. Приведённые выше непараметрические оценки лишены этого недостатка.

1.4.3. Стратификация

Стратификация выборки представляет собой метод, направленный на уменьшение разброса (дисперсии) оценок, получаемых при скользящем контроле, что позволяет получить более узкие доверительные интервалы и более точные верхние оценки.

Процесс стратификации заключается в том, чтобы заранее разделить выборку на несколько частей (страты), и при разбиении на обучающую выборку длины m и контрольную выборку длины k обеспечить, чтобы каждая страта была представлена в обучении и контроле в одинаковых пропорциях $m : k$.

Стратификация классов в задачах классификации означает, что каждый класс делится между обучением и контролем в пропорции $m : k$.

Стратификация по вещественному признаку осуществляется следующим образом: объекты выборки сортируются по какому-либо критерию, например, по возрастанию значения одного из признаков. Затем выборка разбивается на k последовательных страт одинаковой длины (с точностью до 1). При формировании контрольных выборок из каждой страты выбирается по одному объекту — либо с заданным порядковым номером внутри страты, либо случайным образом.

1.5. Разновидности скользящего контроля

Существует несколько различных методов скользящего контроля, которые могут отличаться по способу разбиения выборки.

1.5.1. Полный скользящий контроль (complete CV)

Оценка скользящего контроля строится по всем возможным $N = C_L^k$ разбиениям. В зависимости от длины обучающей выборки k различают следующие варианты:

- **Частный случай при $k = 1$ — контроль по отдельным объектам (leave-one-out CV)**

Как было показано, контроль по отдельным объектам является асимптотически оптимальным при определённых условиях, а именно:

$$\frac{L_n(\hat{m})}{\inf_{m \in M_n} L_n(m)} \rightarrow 1 \text{ по вероятности,}$$

где:

- M_n — класс сравниваемых моделей;
- $L_n(m)$ — среднеквадратичная ошибка при выборе m -ой модели;
- $\hat{m} = \arg \min_{m \in M_n} \text{CV}(m)$.

- **Общий случай при $k > 2$**

В этом случае число разбиений $N = C_L^k$ становится очень большим, даже для сравнительно малых значений k , что делает практическое применение данного метода затруднительным. Для такого случая полный скользящий контроль используется либо в теоретических исследованиях [Voronov2004], либо в редких ситуациях, когда удаётся вывести эффективную вычислительную формулу. Например, для метода k ближайших соседей существует такая формула, которая позволяет эффективно выбирать параметр k .

На практике чаще применяются другие варианты скользящего контроля.

1.5.2. Случайные разбиения

Разбиения $n = 1, \dots, N$ выбираются случайным образом, независимо и с одинаковой вероятностью из множества всех возможных C_L^k разбиений. Именно для такого случая верны приведённые выше оценки доверительных интервалов. На практике эти оценки обычно применяются без изменений и к другим методам разбиения выборки.

1.5.3. Контроль на отложенных данных (hold-out CV)

Оценка модели с использованием метода скользящего контроля основана на одном случайному разбиении выборки, где $N = 1$.

Однако этот метод имеет несколько значительных недостатков:

1. Часто приходится оставлять слишком много объектов в контрольной подвыборке, что приводит к сокращению обучающей выборки. Это уменьшение объема обучающей выборки вызывает смещение оценки (пессимистически завышенную вероятность ошибки).
2. Оценка модели значительно зависит от конкретного разбиения данных, в то время как более предпочтительно, чтобы она характеризовала исключительно алгоритм обучения, а не случайность разбиения.
3. Дисперсия оценки может быть высокой. Она может быть снижена путём усреднения оценок по нескольким разбиениям.

Важно различать два типа контроля по отложенным данным и по тестовой выборке:

- **Контроль по отложенным данным (Hold-out)** — оценка вероятности ошибки производится для классификатора, построенного по всей выборке.
- **Контроль по тестовой выборке (Test-set)** — оценка вероятности ошибки вычисляется для классификатора, обученного на обучающей подвыборке.

1.5.4. Контроль по отдельным объектам (leave-one-out CV)

Метод leave-one-out (LOO) является частным случаем полной кросс-валидации с использованием скользящего контроля при $k = 1$, что означает $N = L$, где L — количество объектов в выборке. Это один из наиболее популярных вариантов скользящей кросс-валидации.

Основное преимущество LOO заключается в том, что каждый объект участвует в процессе контроля ровно один раз, при этом размер обучающих подвыборок лишь на единицу меньше общей выборки.

Однако метод LOO имеет и ряд недостатков, основным из которых является высокая вычислительная нагрузка, поскольку для каждого объекта выборки требуется провести обучение модели заново. В некоторых случаях, когда используемые алгоритмы обучения позволяют быстро адаптировать внутренние параметры при замене одного объекта на другой, процесс LOO можно значительно ускорить.

1.5.5. Контроль по q блокам (q -fold CV)

В методе q -fold кросс-валидации выборка случайнным образом делится на q непересекающихся блоков, каждый из которых имеет одинаковую (или почти одинаковую) длину k_1, \dots, k_q :

$$X^L = X_1^{k_1} \cup \dots \cup X_q^{k_q}, \quad k_1 + \dots + k_q = L.$$

Каждый блок поочередно используется в качестве контрольной подвыборки, а обучение модели проводится на оставшихся $q - 1$ блоках. Критерий ошибки определяется как среднее значение ошибки на контрольной подвыборке:

$$CV(\mu, X^L) = \frac{1}{q} \sum_{n=1}^q Q\left(\mu\left(X^L \setminus X_n^{k_n}\right), X_n^{k_n}\right),$$

где Q — функция потерь. Этот метод представляет собой компромисс между подходами LOO, hold-out и случайными разбиениями. С одной стороны, обучение проводится только q раз, а не L . С другой стороны, длина обучающих подвыборок, равная $L \frac{q-1}{q}$ с точностью до округления, практически не отличается от длины всей выборки L . Обычно выборку делят случайнным образом на 10 или 20 блоков.

1.5.6. Контроль по $r \times q$ блокам ($r \times q$ -fold CV)

Контроль с использованием $r \times q$ -кратного разбиения (или $r \times q$ -fold кросс-валидации) представляет собой метод, при котором процедура q -кратной кросс-валидации повторяется r раз. В каждом повторении данные случайнным образом делятся на q непересекающихся блоков, обеспечивая полное покрытие выборки. Этот метод сохраняет все преимущества стандартной q -кратной кросс-валидации, добавляя при этом гибкость в выборе количества разбиений.

Данный подход стратифицированного скользящего контроля считается одной из базовых методик для оценки и сравнения производительности алгоритмов классификации. Он широко используется в таких системах, как WEKA и «Полигон алгоритмов».

Метод скользящего контроля применяется также в задачах прогнозирования.

1.6. Скользящий контроль в задачах прогнозирования

В задачах прогнозирования, динамического обучения, обучения с подкреплением и активного обучения примеры часто линейно упорядочены по времени их появления. В таких случаях возможности применения различных вариантов скользящего контроля ограничены.

1.6.1. Контроль с нарастающей длиной обучения

Предполагается, что обучающая подвыборка включает все предыдущие наблюдения: $X_n^m = \{x_1, \dots, x_n\}$. Контрольная подвыборка состоит из последующих наблюдений: $X_n^k = \{x_{n+\delta+1}, \dots, x_{n+\delta+k}\}$, где $\delta \geq 0$ — величина задержки прогноза (как правило, $\delta = 0$). Момент «текущего времени» n перемещается по выборке:

$$CV(\mu, X^L) = \frac{1}{T_2 - T_1 + 1} \sum_{n=T_1}^{T_2} Q(\mu(X_n^m), X_n^k),$$

где T_1 — минимальная длина обучающей выборки, необходимая для корректной работы алгоритма обучения μ , $T_2 = L - \delta - k$.

Так как длина обучающей подвыборки $m = n$ увеличивается со временем, точность прогнозов алгоритма может постепенно возрастать. Этот эффект может быть нежелательным, если цель скользящего контроля — объективная оценка качества алгоритма обучения.

1.6.2. Контроль с фиксированной длиной обучения

Отличается от метода с нарастающей длиной тем, что размер обучающей подвыборки m остаётся постоянным, включающим только последние m примеров временного ряда: $X_n^m = \{x_{n-m+1}, \dots, x_n\}$. В этом случае минимальная длина обучающей выборки принимается равной $T_1 = m$.

1.7. Недостатки скользящего контроля

- При использовании скользящего контроля обучение выполняется N раз, что связано с высокими вычислительными затратами.
- Оценка скользящего контроля предполагает наличие заранее заданного алгоритма обучения μ , но она не раскрывает, какими свойствами должны обладать "хорошие" алгоритмы обучения и как их можно построить. В этом смысле теоретические оценки обобщающей способности могут давать полезные рекомендации.
- Попытка применить скользящий контроль как критерий оптимизации в процессе обучения приводит к утрате его несмешённости, что увеличивает риск переобучения.
- Скользящий контроль предоставляет несмешённую точечную оценку риска, но не интервальную. На данный момент нет методов, которые позволяли бы на основе скользящего контроля строить точные доверительные интервалы

для риска, то есть для математического ожидания потерь (включая вероятность ошибочной классификации).

1.8. Применение скользящего контроля

Скользящий контроль широко используется на практике для настройки некоторых ключевых параметров, которые, как правило, определяют структуру или сложность модели алгоритма и имеют ограниченное количество возможных значений.

Примеры применения:

- Выбор модели из ограниченного набора альтернативных алгоритмов.
- Оптимизация параметра регуляризации, включая:
 - параметр регуляризации в гребневой регрессии;
 - параметр весового затухания (weight decay) в нейронных сетях;
 - параметр C в методе опорных векторов.
- Настройка ширины окна в методах:
 - парзеновского окна;
 - ближайшего соседа;
 - ядерного сглаживания.
- Оптимизация количества нейронов в скрытом слое многослойной нейронной сети.
- Выбор информативного набора признаков.
- Сокращение решающего дерева.
- Минимизация структурного риска.

Список литературы

1. Воронцов К. В. Комбинаторный подход к оценке качества обучаемых алгоритмов. — Математические вопросы кибернетики. М.: Физматлит, 2004.
2. Эфрон Б. Нетрадиционные методы многомерного статистического анализа. — М: Финансы и статистика. — 1988.
3. Langford J. Quantitatively Tight Sample Complexity Bounds. — Carnegie Mellon Thesis. — 2002. — 124 с.
4. Kohavi R. A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. — 14th International Joint Conference on Artificial Intelligence, Palais de Congres Montreal, Quebec, Canada. — 1995. — С. 1137-1145.
5. Mullin M., Sukthankar R. Complete Cross-Validation for Nearest Neighbor Classifiers. — Proceedings of International Conference on Machine Learning. — 2000. — С. 1137-1145.

Задача: Cross-Validation (LOOCV)

У вас есть набор данных, состоящий из 6 объектов:

$$D = \{(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4), (x_5, y_5), (x_6, y_6)\},$$

где y — целевая переменная, которая может принимать значение 0 или 1.

Простая линейная модель для предсказания y задана как:

$$y = a \cdot x + b,$$

где параметры a и b нужно подобрать с помощью обучения на обучающей выборке.

Требуется:

1. Используйте **leave-one-out cross-validation (LOOCV)** для оценки качества модели.
2. Для каждой итерации используйте метод наименьших квадратов для подбора параметров a и b по формулам:

$$a = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2}, \quad b = \bar{y} - a \cdot \bar{x}.$$

3. Проведите LOOCV:

- На каждой итерации оставьте одно наблюдение для тестирования и обучайте модель на оставшихся данных.
- Рассчитайте параметры a и b для каждой подвыборки.
- Используя найденные параметры, предскажите y для отложенного объекта.

4. Рассчитайте среднеквадратическую ошибку (MSE) на тестовых объектах:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2.$$

Данные:

Таблица с наблюдениями:

Объект i	x_i	y_i
1	1	0
2	2	0
3	3	1
4	4	1
5	5	1
6	6	0

Решение задачи LOOCV

1. Процедура LOOCV:

- На каждой итерации исключаем одно наблюдение из выборки для тестирования.
- Обучаем модель на оставшихся наблюдениях.
- Рассчитываем параметры линейной модели по формулам:

$$a = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2}, \quad b = \bar{y} - a \cdot \bar{x},$$

где \bar{x} и \bar{y} — средние значения x и y на обучающей выборке.

- Предсказываем y для тестового объекта:

$$\hat{y} = a \cdot x + b.$$

2. Итерации:

Рассмотрим все 6 итераций:

- **Итерация 1:** Тестовый объект $(x_1 = 1, y_1 = 0)$
 - Обучающая выборка: $(x, y) = \{(2, 0), (3, 1), (4, 1), (5, 1), (6, 0)\}$.
 - Найденные параметры: a_1, b_1 .
 - Предсказание: \hat{y}_1 .
 - Ошибка: $(y_1 - \hat{y}_1)^2$.
- **Итерация 2:** Тестовый объект $(x_2 = 2, y_2 = 0)$
 - Обучающая выборка: $(x, y) = \{(1, 0), (3, 1), (4, 1), (5, 1), (6, 0)\}$.
 - Найденные параметры: a_2, b_2 .
 - Предсказание: \hat{y}_2 .
 - Ошибка: $(y_2 - \hat{y}_2)^2$.
- Аналогично повторяем для всех объектов $i = 3, 4, 5, 6$.

3. Среднеквадратическая ошибка:

Рассчитываем MSE:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2,$$

где $n = 6$.

4. Результат:

$$\text{MSE} \approx 0.653$$

Задача: Кросс-валидация для классификации (5-Fold)

Вам дан набор данных из 10 объектов:

$$D = \{(x_1, y_1), (x_2, y_2), \dots, (x_{10}, y_{10})\},$$

где x_i — признак (вещественное число), а y_i — метка класса (0 или 1).

Модель классификации представляет собой простое пороговое правило:

$$\hat{y} = \begin{cases} 1, & \text{если } x \geq t, \\ 0, & \text{если } x < t, \end{cases}$$

где t — порог, который нужно определить.

Требуется:

1. Разбейте данные на 5 фолдов для **5-Fold Cross-Validation**.
2. Для каждого фолда:
 - Используйте 4 фолда для обучения (подберите оптимальный порог t) и 1 фолд для тестирования.
 - Подберите t , чтобы минимизировать число ошибок классификации (ошибок предсказания).
3. Рассчитайте среднюю точность (accuracy) на тестовых фолдах:

$$\text{Accuracy} = \frac{\text{Число верных предсказаний}}{\text{Общее число объектов}}.$$

Данные:

Таблица с наблюдениями:

Объект i	x_i	y_i
1	1.2	0
2	2.3	0
3	2.8	0
4	3.4	1
5	4.1	1
6	4.8	1
7	5.5	1
8	6.0	1
9	6.7	0
10	7.5	0

Упрощение для решения:

- Разделите данные по порядку: первые 2 объекта в первый фолд, следующие 2 — во второй, и так далее.
- Для каждого обучающего фолда подберите порог t , перебрав средние значения между соседними объектами x_i .
- Для упрощения расчетов: оптимизация t и проверка классификации выполняются с помощью простых сравнений.

Решение задачи: Кросс-валидация для классификации (5-Fold)

Шаг 1: Разбиение данных на 5 фолдов

Каждый фолд состоит из двух объектов. Разбиение:

$$\begin{aligned}
 \text{Фолд 1: } & \{(1.2, 0), (2.3, 0)\}, \\
 \text{Фолд 2: } & \{(2.8, 0), (3.4, 1)\}, \\
 \text{Фолд 3: } & \{(4.1, 1), (4.8, 1)\}, \\
 \text{Фолд 4: } & \{(5.5, 1), (6.0, 1)\}, \\
 \text{Фолд 5: } & \{(6.7, 0), (7.5, 0)\}.
 \end{aligned}$$

Шаг 2: Выбор оптимального порога t для каждого фолда

Для обучения используется объединение 4 фолдов, а пятый фолд служит тестовым.

Фолд 1: Тестовый фолд $\{(1.2, 0), (2.3, 0)\}$ Обучающая выборка:

$$\{(2.8, 0), (3.4, 1), (4.1, 1), (4.8, 1), (5.5, 1), (6.0, 1), (6.7, 0), (7.5, 0)\}.$$

Проверяем пороги t (средние значения между x_i):

$$t_1 = \frac{2.8 + 3.4}{2} = 3.1, \quad t_2 = \frac{3.4 + 4.1}{2} = 3.75,$$

$$t_3 = \frac{4.1 + 4.8}{2} = 4.45, \quad \dots, \quad t_7 = \frac{6.7 + 7.5}{2} = 7.1.$$

Оптимальный порог $t = 3.1$ минимизирует ошибки. Тестируем: обе точки $(1.2, 0)$ и $(2.3, 0)$ классифицируются верно.

$$\text{Accuracy}_1 = 1.0.$$

Фолд 2: Тестовый фолд $\{(2.8, 0), (3.4, 1)\}$ Обучающая выборка:

$$\{(1.2, 0), (2.3, 0), (4.1, 1), (4.8, 1), (5.5, 1), (6.0, 1), (6.7, 0), (7.5, 0)\}.$$

Оптимальный порог $t = 3.75$. Тестируем:

$$\hat{y}(2.8) = 0 \quad (\text{верно}), \quad \hat{y}(3.4) = 1 \quad (\text{верно}).$$

$$\text{Accuracy}_2 = 1.0.$$

Фолд 3: Тестовый фолд $\{(4.1, 1), (4.8, 1)\}$ Обучающая выборка:

$$\{(1.2, 0), (2.3, 0), (2.8, 0), (3.4, 1), (5.5, 1), (6.0, 1), (6.7, 0), (7.5, 0)\}.$$

Оптимальный порог $t = 4.5$. Тестируем:

$$\hat{y}(4.1) = 1 \quad (\text{верно}), \quad \hat{y}(4.8) = 1 \quad (\text{верно}).$$

$$\text{Accuracy}_3 = 1.0.$$

Фолд 4: Тестовый фолд $\{(5.5, 1), (6.0, 1)\}$ Обучающая выборка:

$$\{(1.2, 0), (2.3, 0), (2.8, 0), (3.4, 1), (4.1, 1), (4.8, 1), (6.7, 0), (7.5, 0)\}.$$

Оптимальный порог $t = 5.75$. Тестируем:

$$\hat{y}(5.5) = 1 \quad (\text{верно}), \quad \hat{y}(6.0) = 1 \quad (\text{верно}).$$

$$\text{Accuracy}_4 = 1.0.$$

Фолд 5: Тестовый фолд $\{(6.7, 0), (7.5, 0)\}$ Обучающая выборка:
 $\{(1.2, 0), (2.3, 0), (2.8, 0), (3.4, 1), (4.1, 1), (4.8, 1), (5.5, 1), (6.0, 1)\}$.

Оптимальный порог $t = 6.35$. Тестируем:

$$\hat{y}(6.7) = 0 \quad (\text{верно}), \quad \hat{y}(7.5) = 0 \quad (\text{верно}).$$

Accuracy₅ = 1.0.

Шаг 3: Средняя точность

Средняя точность:

$$\text{Accuracy}_{\text{mean}} = \frac{\text{Accuracy}_1 + \text{Accuracy}_2 + \text{Accuracy}_3 + \text{Accuracy}_4 + \text{Accuracy}_5}{5} = 1.0.$$

Вывод

Модель с оптимальными порогами t классифицировала все объекты правильно на каждом из фолдов. Средняя точность составляет **100%**.

Задача: Кросс-валидация для оценки среднего значения

У вас есть набор данных, содержащий измеренные значения некоторой величины:

$$D = \{x_1, x_2, x_3, \dots, x_{12}\},$$

где x_i — вещественное число. Вам необходимо оценить среднее значение этой величины и одновременно оценить, насколько точно модель предсказывает новые значения, используя технику кросс-валидации.

Требуется:

1. Проведите разбиение данных на **4 фолда** для выполнения 4-Fold Cross-Validation.
2. Для каждого фолда:
 - Используйте 3 фолда для расчета среднего значения \bar{x} .
 - Используйте 1 фолд для тестирования, чтобы рассчитать абсолютное отклонение предсказанного среднего \bar{x} от истинных значений.
3. Рассчитайте среднее абсолютное отклонение (MAE) по всем тестовым фолдам:

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^n |x_i - \bar{x}|,$$

где n — количество объектов в тестовых фолдах.

Данные:

Таблица с измерениями:

Объект i	x_i
1	5.1
2	4.8
3	6.2
4	5.7
5	5.4
6	6.0
7	5.3
8	4.9
9	6.1
10	5.8
11	5.5
12	6.3

—

Упрощение для решения:

- Разделите данные на 4 фолда последовательно: первые 3 объекта в первый фолд, следующие 3 — во второй и так далее.
- Расчеты среднего \bar{x} и абсолютных отклонений выполняйте вручную для каждого фолда.
- Итоговая метрика MAE рассчитывается как среднее значение всех абсолютных отклонений.

Решение: Кросс-валидация для оценки среднего значения

Шаг 1. Разбиение данных на фолды

Разделим данные $D = \{5.1, 4.8, 6.2, 5.7, 5.4, 6.0, 5.3, 4.9, 6.1, 5.8, 5.5, 6.3\}$ на 4 фолда:

Фолд 1: $\{5.1, 4.8, 6.2\}$, Фолд 2: $\{5.7, 5.4, 6.0\}$,

Фолд 3: $\{5.3, 4.9, 6.1\}$, Фолд 4: $\{5.8, 5.5, 6.3\}$.

Шаг 2. Расчёт среднего значения и абсолютных отклонений для каждого фолда

Для каждого фолда используем три других фолда для расчёта среднего значения \bar{x} и тестируем на оставшемся фолде.

Фолд 1 (тест): {5.1, 4.8, 6.2} Среднее значение \bar{x} , рассчитанное по остальным фолдам:

$$\bar{x}_1 = \frac{5.7 + 5.4 + 6.0 + 5.3 + 4.9 + 6.1 + 5.8 + 5.5 + 6.3}{9} = 5.67.$$

Абсолютные отклонения:

$$|5.1 - 5.67| = 0.57, \quad |4.8 - 5.67| = 0.87, \quad |6.2 - 5.67| = 0.53.$$

Фолд 2 (тест): {5.7, 5.4, 6.0} Среднее значение \bar{x} , рассчитанное по остальным фолдам:

$$\bar{x}_2 = \frac{5.1 + 4.8 + 6.2 + 5.3 + 4.9 + 6.1 + 5.8 + 5.5 + 6.3}{9} = 5.55.$$

Абсолютные отклонения:

$$|5.7 - 5.55| = 0.15, \quad |5.4 - 5.55| = 0.15, \quad |6.0 - 5.55| = 0.45.$$

Фолд 3 (тест): {5.3, 4.9, 6.1} Среднее значение \bar{x} , рассчитанное по остальным фолдам:

$$\bar{x}_3 = \frac{5.1 + 4.8 + 6.2 + 5.7 + 5.4 + 6.0 + 5.8 + 5.5 + 6.3}{9} = 5.64.$$

Абсолютные отклонения:

$$|5.3 - 5.64| = 0.34, \quad |4.9 - 5.64| = 0.74, \quad |6.1 - 5.64| = 0.46.$$

Фолд 4 (тест): {5.8, 5.5, 6.3} Среднее значение \bar{x} , рассчитанное по остальным фолдам:

$$\bar{x}_4 = \frac{5.1 + 4.8 + 6.2 + 5.7 + 5.4 + 6.0 + 5.3 + 4.9 + 6.1}{9} = 5.49.$$

Абсолютные отклонения:

$$|5.8 - 5.49| = 0.31, \quad |5.5 - 5.49| = 0.01, \quad |6.3 - 5.49| = 0.81.$$

Шаг 3. Итоговая метрика MAE

Среднее абсолютное отклонение рассчитывается как:

$$\text{MAE} = \frac{\sum_{i=1}^{12} |x_i - \bar{x}|}{12}.$$

Подставляем рассчитанные значения:

$$\text{MAE} = \frac{0.57 + 0.87 + 0.53 + 0.15 + 0.15 + 0.45 + 0.34 + 0.74 + 0.46 + 0.31 + 0.01 + 0.81}{12}$$

Ответ:

Среднее абсолютное отклонение (MAE): 0.49

1.9. Вероятность переобучения

Рассмотрим объединенную выборку обучение + контроль:

$X^L = \{x_1, \dots, x_L\}$ – конечное генеральное множество объектов;

и

$A = \{a_1, \dots, a_D\}$ – конечное семейство алгоритмов;

т. е. те алгоритмы, среди которых мы выбираем (делаем Model Selection или делаем обучения по обучающей выборке). Пусть при этом функция потерь бинарна, т. е. у нас есть индикатор ошибки (условие того, что алгоритм a ошибается на объекте x):

$$\mathcal{L}(a, x) \equiv I(a, x) = [\text{алгоритм } a \text{ ошибается на объекте } x].$$

Тогда естественным образом возникает матрица ошибок (не путать с матрицей объект-признак) – у нее по строкам объекты, а по столбцам – наши алгоритмы (каждый столбец – бинарный вектор, на каких объектах ошибся наш алгоритм):

На столбце a_4 мы как раз видим переобучение – на обучающих объектах 0 ошибок, а на контрольных – все ошибки (идеально плохая ситуация). Фишка в том, что мы начнем разбивать генеральную выборку всеми возможными способами на обучение и контроль (их всего C_{l+k}^l). Обозначим

$$n(a, X) = \sum_{x \in X} I(a, x) – \text{число ошибок } a \in A \text{ на выборке } X \subset X^L;$$

$$\nu(a, X) = n(a, X)/|X| – \text{частота ошибок } a \text{ на выборке } X.$$

	a_1	a_2	a_3	a_4	a_5	a_6	\dots	a_D	
x_1	1	1	0	0	0	1	\dots	1	X^l — наблюдаемая
\dots	0	0	0	0	1	1	\dots	1	(обучающая) выборка
x_l	0	0	1	0	0	0	\dots	0	длины l
x_{l+1}	0	0	0	1	1	1	\dots	0	X^k — скрытая
\dots	0	0	0	1	0	0	\dots	1	(контрольная)
x_L	0	1	1	1	1	1	\dots	0	выборка длины

Таблица 1.1. $L \times D$ – матрица ошибок с попарно различными столбцами
 $k = L - l$

Нам придется сравнивать частоты ошибок на обучении и на контроле. Их разность – это и есть переобученность. Посмотрим на примере, откуда вообще берется матрица ошибок:

Пример 1.

Пусть у нас есть выборка из 10 объектов, 5 объектов одного класса и 5 другого. Строим линейные классификаторы. В матрице ошибок есть один нулевой вектор (линейный классификатор без ошибок см. рис. 1.1), 5 векторов с одной ошибкой (рис. 1.2 и т. д.). Вот откуда берется матрица ошибок. Максимально в ней может быть 2^l вектор-столбцов.

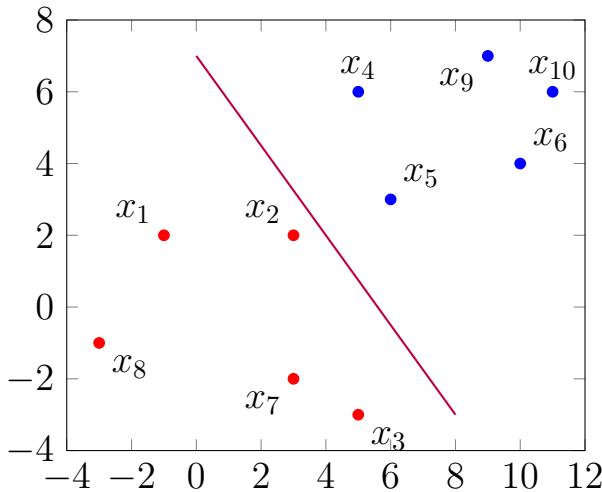


Рис. 1.1. линейный классификатор без ошибок

Наша задача будет оценивать вероятность переобучения. Для начала дадим несколько определений. Вероятностное пространство – это равновероятное разбиение выборки на 2 части (train X^l и test X^k). Интерпретация этому это гипотеза, что наша выборка IID (independent and identically distributed). Т. е. как бы мы эту выборку не упорядочивали, все ее упорядочивания равновероятны (как бы мы ее не разбивали на две части, все разбиения равновероятны). Это есть гипотеза простой выборки.

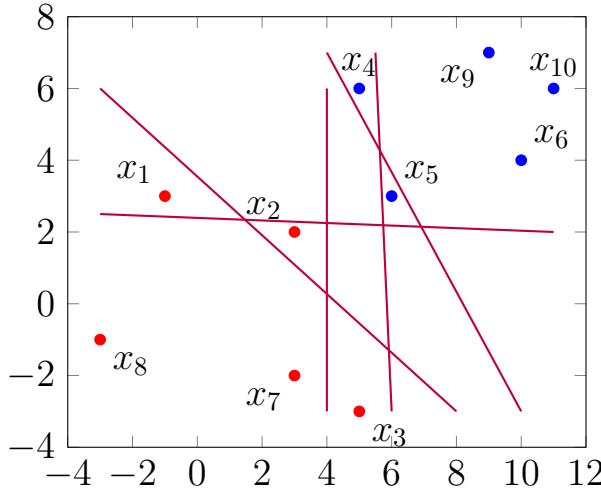


Рис. 1.2. линейный классификатор с 1 ошибкой

x_1	0	1	0	0	0	0	...
x_2	0	0	1	0	0	0	...
x_3	0	0	0	1	0	0	...
x_4	0	0	0	0	1	0	...
x_5	0	0	0	0	0	1	...
x_6	0	0	0	0	0	0	...
x_7	0	0	0	0	0	0	...
x_8	0	0	0	0	0	0	...
x_9	0	0	0	0	0	0	...
x_{10}	0	0	0	0	0	0	...

Таблица 1.2. матрица ошибок для примера 1

Замечание. С другой стороны на это можно смотреть, как на Complete Cross Validation (полный скользящий контроль). Всеми способами разбиваем выборку на train и test. Потом по всем этим способам вероятность любого события, зависящая от этого разбиения, усредняем по всем разбиениям. В этой вероятностной интерпретации очень удобно, что математическое ожидание есть просто усреднение по всем разбиениям выборки

$$P \equiv E \equiv \frac{1}{C_L^l} \sum_{X^l \subset X^L} .$$

Переобученность – это разность частот ошибок на X^k и на X^l (обучились на X^l , частоту ошибок посчитали на X^k , частота ошибок нормирована от 0 до 1):

$$\delta(\mu, X^l, X^k) = \nu(\mu(X^l), X^k) - \nu(\mu(X^l), X^l).$$

Переобучение — это событие $\delta(\mu, X^l, X^k) \geq \varepsilon$ для фиксированного ε .

Основная наша задача — оценить вероятность переобучения (вероятность в смысле доли разбиения выборки, на которых δ оказалась $\geq \varepsilon$ при заданном ε):

$$R_\varepsilon(\mu, X^L) = P[\delta(\mu, X^l, X^k) \geq \varepsilon].$$

Теперь наша задача — научиться сверху оценивать эту величину. Рассмотрим простейший, но важный частный случай.

Пусть у нас нет никакого выбора алгоритма, научимся оценивать вероятность переобучения хотя бы для одного отдельно взятого алгоритма, т. е. для одного вектор-столбца матрицы ошибок. Пусть $A = a$ — одноэлементарное множество, $m = n(a, X^L)$. Тогда вероятность переобучения есть вероятность большого отклонения частот ошибок в двух подвыборках:

$$R_\varepsilon(a, X^L) = P[\nu(a, X^k) - \nu(a, X^l) \geq \varepsilon].$$

Теорема Для любой выборки X^l , любого $\varepsilon \in [0, 1]$ вероятность большого отклонения частот ошибок в двух подвыборках для фиксированного вектора ошибок:

$$R_\varepsilon(a, X^L) = \mathcal{H}_L^{l,m}\left(\frac{l}{L}(m - \varepsilon k)\right),$$

где $\mathcal{H}_L^{l,m}(x) = \sum_{s=0}^{\lfloor x \rfloor} \frac{C_m^s C_{L-m}^{l-s}}{C_L^l}$ — функция гипергеометрического распределения (ее левый хвост).

Доказательство. \square Обозначим число ошибок на обучении как $s = n(a, X^l)$. Аналогия со 2 задачей из упражнения, где m — количество объектов с ошибками, l — обучающая выборка (гипергеометрическое распределение):

$$P[n(a, X^l) = s] = C_m^s C_{L-m}^{l-s} / C_L^l.$$

Распишем R_ε , подставив в него $\nu(a, X^k) = \frac{m-s}{k}$, $\nu(a, X^l) = \frac{s}{l}$ (частота ошибок на обучении $\frac{s}{l}$, а на контроле $\frac{m-s}{k}$) и учитывая тот факт, что $\frac{m-s}{k} - \frac{s}{l}$ есть переобученность:

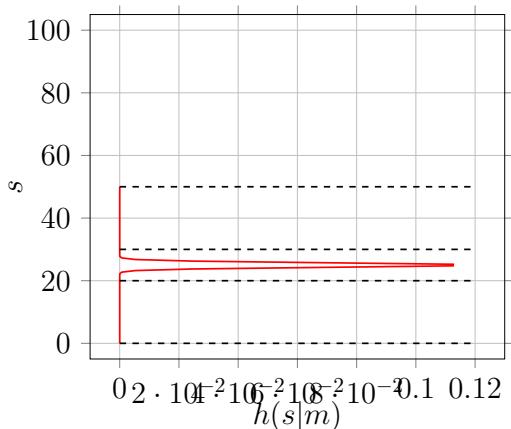
$$\begin{aligned} R_\varepsilon(a, X^L) &= P[\nu(a, X^k) - \nu(a, X^l) \geq \varepsilon] = \\ &= \sum_{s=0}^l \underbrace{\left[\frac{m-s}{k} - \frac{s}{l} \geq \varepsilon \right]}_{s \leq \frac{l}{L}(m - \varepsilon k)} \underbrace{P[n(a, X^l) = s]}_{C_m^s C_{L-m}^{l-s} / C_L^l} = \mathcal{H}_L^{l,m}\left(\frac{l}{L}(m - \varepsilon k)\right). \end{aligned}$$

При этом мы получили ограничение на s , которое является следствием того, что переобученность $\leq \varepsilon$. Наша оценка — левый хвост гипергеометрического распределения. ■

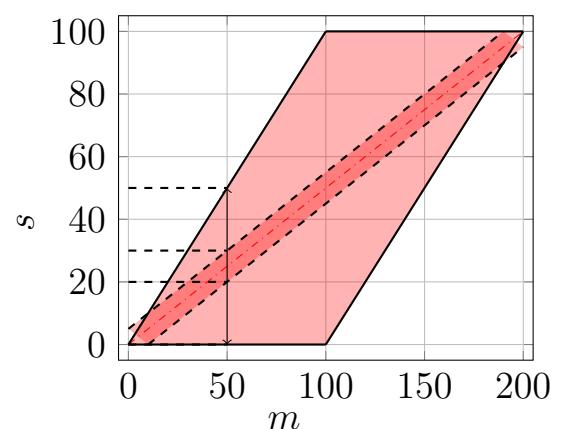
Посмотрим как это выглядит на картинке. На оси x отложено m — общее число ошибок на полной выборке, по оси y откладывается число ошибок на обучающей выборке. На графике 1.3а видна узкая полоска — явление концентрации вероятностной меры. Это очень важное явление и в теории вероятности, и в машинном обучении, и в Computational learning theory. Это основной теоретический момент, из которого могут быть получены различные оценки (красная полоска). При том чем больше будет длина выборки, тем уже будет относительная ширина этой полоски.

Предсказание числа $t = n(a, X^L)$ по числу $s = n(a, X^l)$ возможно благодаря узости гипергеометрического пика, причем при $l, k \rightarrow \infty$ он сужается, и $\nu(a, X^l) \rightarrow \nu(a, X^k)$ (явление концентрации вероятности, закон больших чисел).

Гипергеометрическое распределение нам говорит, что если мы знаем m (общее число ошибок на полной выборке), то мы можем предсказать диапазон изменений для s , т. е. сколько у нас ошибок попадет в обучение. Значит будем знать $m - s$, сколько ошибок попадет в контроль. Но в реальной ситуации нам приходится обращать это гипергеометрическое распределение, т. к. мы знаем сколько ошибок у нас попало в обучение. Мы знаем что у нас по y и можем провести горизонтальную линию и сказать, в каком у нас диапазоне могло бы быть m , т. е. в каком диапазоне могло бы лежать общее число ошибок на объединенной выборке. А $m - s$ это число ошибок на контроле.



(a) $h(s|m)$ при $m = 50$



(b) $h(s|m)$ при $L = 200, k = 100$

Рис. 1.3. Гипергеометрическое распределение $h(s|m)$

Задача 1. Сколько линейных классификаторов с двумя ошибками из примера 1?

Решение.

Для решения задачи необходимо посмотреть сколько существует различных способов разделить объекты на рисунке 1.1, чтобы объектов одного класса было на 2 больше, чем другого. Приведем ответ в виде таблицы:

x_1	1	0	0	0	0	1	1	0
x_2	1	1	0	0	0	0	0	0
x_3	0	1	1	0	0	0	0	1
x_4	0	0	1	1	0	0	0	0
x_5	0	0	0	1	1	1	0	0
x_6	0	0	0	0	1	0	1	0
x_7	0	0	0	0	0	0	0	1
x_8	0	0	0	0	0	0	0	0
x_9	0	0	0	0	0	0	0	0
x_{10}	0	0	0	0	0	0	0	0

Таблица 1.3. все возможные разбиения с 2 ошибками

Задача 2. В урне L шаров, m из них черные, остальные белые; извлекаем l шаров наугад. Какова вероятность, что ровно s из них черные?

Решение.

Обозначим: $N = L$ – общее количество шаров в урне, m – количество черных шаров, $N - m$ – количество белых шаров, l – общее количество извлеченных шаров, s – количество черных шаров среди извлеченных.

Чтобы найти вероятность, нам нужно рассмотреть, сколько способов существует для выбора s черных шаров и $l - s$ белых шаров из общего количества.

Выбрать s черных шаров из m черных можно C_m^s способами.

А выбрать $l - s$ белых шаров из $N - m$ белых – C_{N-m}^{l-s} способами.

Общее количество способов выбрать l шаров из N есть C_N^l .

Таким образом, вероятность того, что из l извлеченных шаров ровно s будут черными, будет равна отношению числа благоприятных исходов к общему числу исходов:

$$P(X = s) = \frac{C_m^s \cdot C_{N-m}^{l-s}}{C_N^l}$$

Задача 3. Как посчитать количество рыб в пруду?

Решение. Выловим некоторое случайное подмножество рыб, пометим их, а потом выпустим обратно в пруд. Через некоторое время рыбы в пруду перемешаются (важно, чтобы они успели перемешаться), опять вылавливаем некоторое количество рыб и смотрим, какая доля среди них – меченная. По этим данным, очевидно, можно найти общее количество рыб в пруду.

1.10. Статистические критерии в машинном обучении

1.10.1. Теория

Статистические инструменты используются для отбора, извлечения и преобразования наиболее релевантных признаков. Например, коэффициент корреляции Пирсона, ранговый коэффициент Спирмена, коэффициент корреляции ANOVA, ранговый коэффициент Кендалла и критерий Хи-квадрат

В данном обзоре рассмотрим приложения математической статистики в машинном обучении – точнее, в отборе признаков для обучения модели.

Мотивация: Преимущества хорошо подобранных признаков в датасете очевидны: это улучшает точность в обучении с учителем и без, уменьшает время и память, необходимые для корректной работы, помогает ослабить проклятие размерности (экспоненциальный рост необходимых данных).

Одним из основных методов, используемых для отбора признаков, является анализ корреляции. Корреляция позволяет определить, насколько сильно связаны между собой различные признаки и целевая переменная. Признаки с высокой корреляцией с целевой переменной могут быть отобраны для дальнейшего анализа, в то время как признаки, которые не имеют значительной связи, могут быть исключены. Это помогает избежать проблем многоколлинеарности, когда несколько признаков предоставляют одинаковую информацию.

Другим важным инструментом является тестирование гипотез. С помощью статистических тестов, таких как t-тест или ANOVA, можно оценить значимость отдельных признаков в контексте целевой переменной. Например, если мы имеем дело с задачей классификации, мы можем использовать тесты для определения того, какие признаки значительно различаются между классами. Это позволяет сосредоточиться на наиболее информативных признаках и исключить те, которые не вносят существенного вклада в модель.

Методы селекции на основе значимости также широко применяются в практике. Алгоритмы, такие как LASSO (Least Absolute Shrinkage and Selection Operator), используют регуляризацию для уменьшения коэффициентов менее значимых признаков до нуля, что автоматически исключает их из модели. Это не только упрощает модель, но и помогает избежать переобучения — проблемы, когда модель слишком точно подстраивается под обучающие данные и теряет способность обобщать на новых данных.

Методы оценки производительности модели также зависят от правильного отбора признаков. Кросс-валидация позволяет оценить, как изменения в наборе признаков влияют на общую производительность модели. Сравнение различных моделей с разными наборами признаков помогает выбрать оптимальный вариант, который обеспечивает наилучшие результаты.

Кроме того, современные подходы к машинному обучению, такие как ансамблевые методы (например, Random Forest), также используют статистические методы для оценки важности признаков. В таких методах важность каждого признака может быть оценена по тому, как сильно он влияет на уменьшение ошибки предсказания модели.

1.10.2. Задачи

1. Анализ корреляции

У вас есть набор данных с 5 признаками (X_1, X_2, X_3, X_4, X_5) и целевой переменной Y . Вы хотите выяснить, какие признаки имеют наибольшую корреляцию с Y .

Решение:

1. Рассчитайте коэффициенты корреляции Пирсона для каждого признака относительно Y .
2. Сравните полученные значения и выберите признаки с наибольшими абсолютными значениями коэффициента корреляции.

2. Тестирование гипотез

Вы хотите проверить, есть ли статистически значимая разница между двумя группами данных (группа А и группа В) по признаку X . У вас есть данные о значениях X для обеих групп.

Решение:

1. Проведите t -тест для независимых выборок.
2. Оцените p -значение для определения значимости.

3. Регуляризация LASSO

У вас есть набор данных с несколькими признаками и целевой переменной. Вы хотите использовать метод LASSO для отбора наиболее значимых признаков.

Решение:

1. Импортируйте библиотеку *Lasso* из *sklearn*.
2. Подготовьте данные и обучите модель LASSO.
3. Оцените важность признаков на основе полученных коэффициентов.

Глава 2

Линейные методы классификации и регрессии

2.1. О регуляризации

При решении задачи машинного обучения часто возникает проблема переобучения, при котором модель подстраивается под шум данных, что снижает ее обобщающую способность. Один из методов борьбы с этим — регуляризации, или же сокращение весов (англ.: weight decay). Данный метод добавляет штраф к функции потерь за сложность модели, в случае линейных моделей — штраф за большие веса коэффициентов. Регуляризация ограничивает пространство решений и делает модель более устойчивой к шуму, что увеличивает вероятность корректных предсказаний на новых данных. Пример, когда усложнение модели, путем добавления избыточных коэффициентов полиномиальной модели представлен на рис. 2.1.

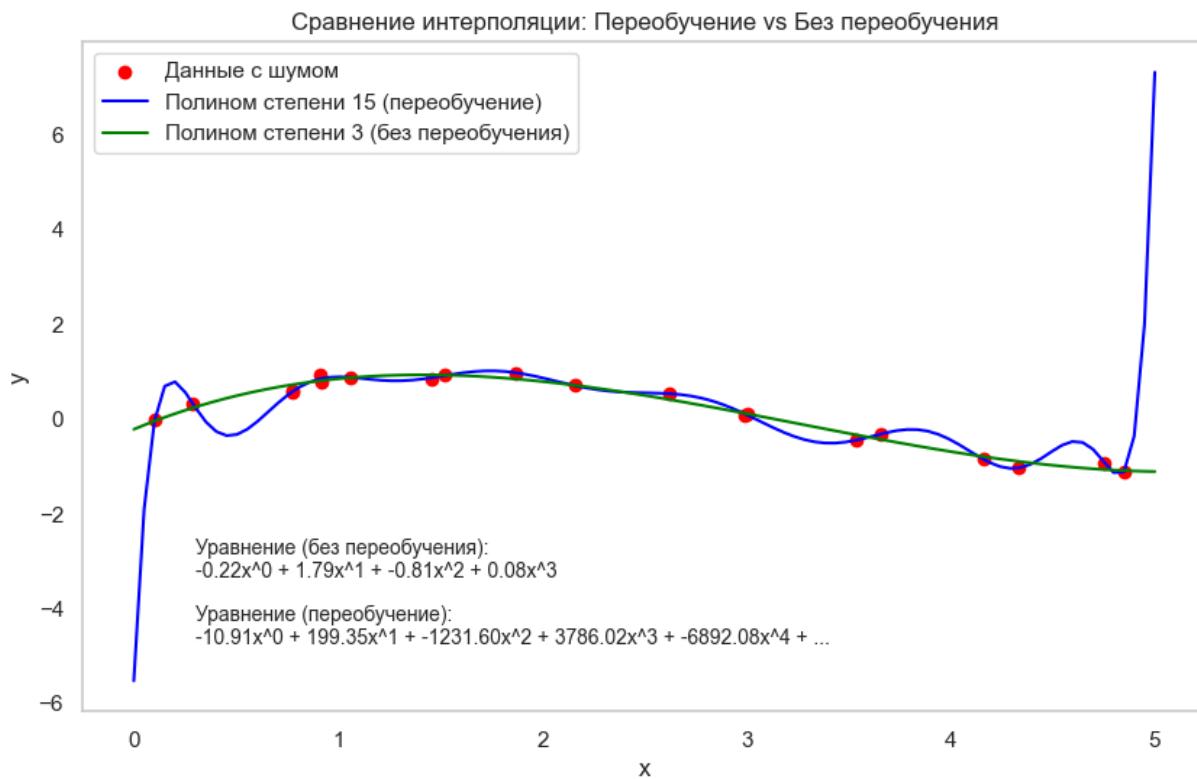


Рис. 2.1. Иллюстрация проблемы переобучения для решения линейной задачи

В разделе 2.1.1. рассмотрены основные методы регуляризации, применяемые в линейных моделях. Раздел 2.1.2. описывает вероятностную трактовку причин возникновения регуляризации. Завершается глава разделом 2.1.3., который содержит три теоретических задачи по теме регрессии.

2.1.1. Гауссовский и лапласовский регуляризаторы

Гауссовский и лапласовский регуляризаторы представляют собой две разные техники регуляризации, которые используются для управления сложностью моделей и предотвращения переобучения. Они основаны на различных подходах к штрафованию весов модели.

Лапласовский регуляризатор (L1 регуляризатор) Лапласовский (L1) регуляризатор использует сумму модулей значений весов модели в качестве штрафа. Формально новую функцию потерь можно записать следующим образом: $L1 = L_0 + \lambda \sum_{i=1}^n |w_i|$, где L_0 — исходная функция потерь, $|w_i|$ — абсолютные значения весов модели, λ — коэффициент регуляризации. Преимущества:

- Лапласовский регуляризатор может приводить к обнулению некоторых весов, что делает модель более интерпретируемой и позволяет выделять наиболее важные признаки.
- Он может быть особенно полезен в задачах с высокой размерностью, где много признаков могут быть неинформативными.

Недостатки:

- Оптимизация с использованием L1 регуляризации может быть более сложной и требовать специальных алгоритмов (например, координатного спуска или методов, основанных на субградиенте).

Гауссовский регуляризатор (L2 регуляризатор) Гауссовский (L2) регуляризатор использует штраф в виде суммы квадратов весов модели. Формально новую функцию потерь можно записать следующим образом: $L2 = L_0 + \lambda \sum_{i=1}^n w_i^2$, где L_0 — исходная функция потерь (например, среднеквадратичная ошибка для задачи регрессии), w_i — веса модели, λ — коэффициент регуляризации, который контролирует величину штрафа. Преимущества:

- Гауссовский регуляризатор помогает сгладить веса, что делает модель более устойчивой к шуму в данных.
- Он способствует распределению весов по всем признакам, уменьшая вероятность того, что некоторые признаки будут доминировать.

Недостатки:

- Может не приводить к полному обнулению весов, поэтому не всегда приводит к интерпретируемым моделям.

Сравнений L1 и L2 регуляризаторов Выбор между гауссовским и лапласовским регуляризаторами зависит от конкретной задачи и целей (сравнение см. в таблице 2.1). Если важна интерпретируемость модели и выделение значимых признаков, то стоит рассмотреть L1 регуляризацию. Если же цель — улучшить общее качество модели без сильного сокращения количества признаков, то L2 регуляризация может быть предпочтительнее. Это связано с тем, что в L2 регуляризации за счет возведения в квадрат значений весов вклад нулевого веса и просто малого веса неразличим на при наличии других выделенных признаков с весами порядка или более единицы. Таким образом, L2 регуляризация, в отличии от L1 не стремится обнулить коэффициенты, что снижает интерпретируемость модели. Однако квадратичная функция является гладкой, поэтому лучше поддается вычислительным методам оптимизации.

Таблица 2.1. Сравнение L1 и L2 регуляризаторов

Характеристика	Лапласовский (L1)	Гауссовский (L2)
Штраф	Сумма абсолютных значений весов	Сумма квадратов весов
Эффект на веса	Обнуление (спарсность)	Сглаживание
Интерпретируемость	Выше	Меньше
Оптимизация	Сложнее	Легче

В ситуациях используется комбинация обоих методов регуляризации. Для этого вводится дополнительный гиперпараметр α — доля первой нормы в штрафе за вес. Получается так называемая эластичная сеть (англ.: elastic net). В таком случае: $L = L_0 + \lambda[\alpha \sum_{i=1}^n |w_i| + (1 - \alpha) \sum_{i=1}^n w_i^2]$. Данный подход позволяет учесть особенности двух подходов, однако усложняет модель.

2.1.2. Вероятностная интерпретация регуляризации

В первом рассмотрении регуляризацию можно рассматривать как введение априорной информации о параметрах модели. Рассмотрим вводимые предположения:

- в случае L1 регуляризации мы предполагаем, что многие веса могут быть равны нулю, это соответствует идеи о том, что истинная модель простая и присутствуют лишние признаки;
- в случае L2 — веса распределены вблизи общего малого среднего значения, это отражает предположение о том, что все признаки вносят сопоставимый вклад в предсказание.

Описанным выше предположениям о весах соответствуют экспоненциальное и нормальное распределения (см. рис. 2.2). Рассмотрим их влияние на штраф и функцию потерь.

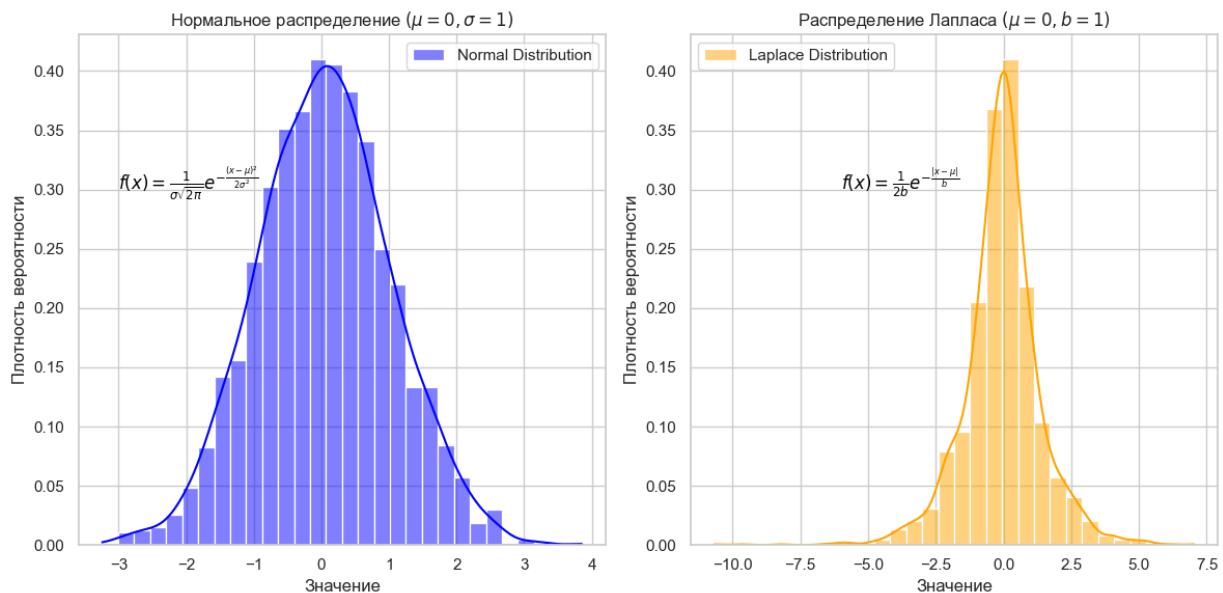


Рис. 2.2. Сравнение экспоненциального и нормального распределений

L1 регуляризация предполагает, что веса w_j распределены по лапласовскому закону (или двойному экспоненциальному распределению), $w_j \sim Laplace(0, b)$.

L2 регуляризация предполагает, что веса модели w_j имеют нормальное распределение с нулевым средним и некоторой дисперсией σ^2 . Это можно записать как $w_j \sim N(0, \sigma^2)$.

2.1.3. Задачи

Вопрос 1 Как изменение коэффициента λ влияет на величину весов w_j ?

L1 регуляризация

При увеличении λ происходит обнулению некоторых весов w_j . Это приводит к тому, что с увеличением λ количество ненулевых весов уменьшается, что может помочь в отборе признаков и упрощении модели.

L2 регуляризация

При увеличении λ происходит увеличение штрафа за большие значения весов. Это приводит к уменьшению величины весов w_j (все веса стремятся к нулю), что помогает избежать переобучения. В случае $\lambda = 0$ модель не имеет регуляризации, и веса могут принимать любые значения, что может привести к переобучению.

Вопрос 2 Какова геометрическая интерпретация регуляризации в пространстве весов?

L1 регуляризация

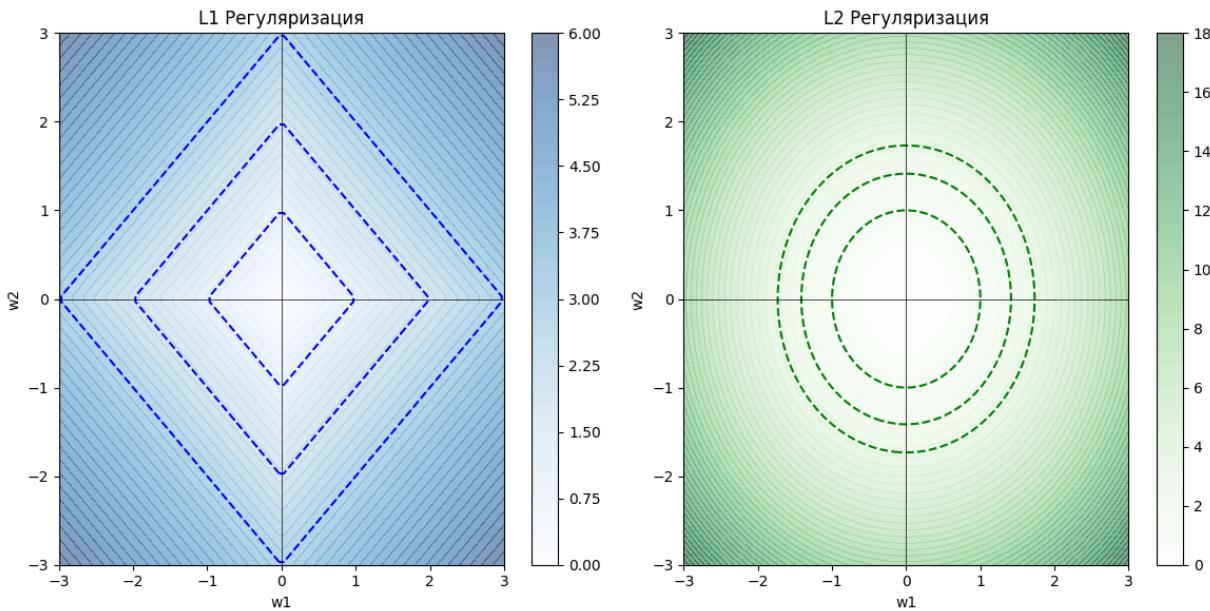


Рис. 2.3. Геометрическая интерпретация регуляризации в пространстве весов

Геометрически L1 регуляризация создает ромбовидные (или параллелепипедные) области в пространстве весов. Оптимальные веса находятся на вершинах этих ромбов, что приводит к обнулению некоторых весов и, следовательно, к отбору признаков.

$$L1(w_1, w_2) = \text{const} \Leftrightarrow |w_1| + |w_2| = \text{const}$$

L2 регуляризация

Геометрически L2 регуляризация создает сферу (или гиперсферу) в пространстве весов, внутри которой минимизируется функция потерь. Это означает, что оптимальные веса будут находиться на поверхности этой сферы, что приводит к сглаживанию и уменьшению значений весов.

$$L2(w_1, w_2) = \text{const} \Leftrightarrow w_1^2 + w_2^2 = \text{const}$$

Вопрос 3 Какие особенности признаков компенсируют регуляризации?

L1 регуляризация

Используя L1 регуляризацию, можно обнулить веса для менее значимых признаков. После обучения модели можно проанализировать ненулевые веса и оставить только те признаки, которые имеют значимые коэффициенты, тем самым осуществляя отбор признаков.

L2 регуляризация

L2 регуляризация помогает сгладить веса при наличии мультиколлинеарности, уменьшая их величину и тем самым снижая влияние коррелирующих признаков. Это позволяет избежать чрезмерного увеличения весов для сильно коррелирующих признаков.

Основная идея

Градиентные методы — это широкий класс оптимизационных алгоритмов, используемых не только в машинном обучении. Здесь градиентный подход будет рассмотрен в качестве способа подбора вектора синаптических весов w в линейном классификаторе. Пусть $y^* : X \rightarrow Y$ — целевая зависимость, известная только на объектах обучающей выборки: $X^l = (x_i, y_i)_{i=1}^l$, где $y_i = y^*(x_i)$.

Найдём алгоритм $a(x, w)$, аппроксимирующий зависимость y^* . В случае линейного классификатора искомый алгоритм имеет вид:

$$a(x, w) = \varphi \left(\sum_{j=1}^n w_j x^j - w_0 \right),$$

где $\varphi(z)$ играет роль функции активации (в простейшем случае можно положить $\varphi(z) = \text{sign}(z)$).

Согласно принципу минимизации эмпирического риска, для этого достаточно решить оптимизационную задачу:

$$Q(w) = \sum_{i=1}^l L(a(x_i, w), y_i) \rightarrow \min_w,$$

где $L(a, y)$ — заданная функция потерь.

Для минимизации применим метод градиентного спуска (gradient descent). Это пошаговый алгоритм, на каждой итерации которого вектор w изменяется в направлении наибольшего убывания функционала Q (то есть в направлении антиградиента):

$$w := w - \eta \nabla Q(w),$$

где η — положительный параметр, называемый темпом обучения (learning rate).

Основные подходы к реализации градиентного спуска

1. **Пакетный (batch):** на каждой итерации обучающая выборка просматривается целиком, и только после этого изменяется w . Этот подход требует больших вычислительных затрат.
2. **Стохастический (stochastic/online):** на каждой итерации из обучающей выборки случайным образом выбирается один объект. Таким образом, вектор w настраивается на каждый вновь выбираемый объект.

Алгоритм Stochastic Gradient (SG)

Вход:

- X^l — обучающая выборка;
- η — темп обучения;
- λ — параметр сглаживания функционала Q .

Выход: Вектор весов w

Тело алгоритма:

1. Инициализировать веса $w_j, j = 0, \dots, n$;
2. Инициализировать текущую оценку функционала: $Q := \sum_{i=1}^l L(a(x_i, w), y_i)$;
3. Повторять:
 - (a) выбрать объект x_i из X^l (например, случайным образом);
 - (b) вычислить выходное значение алгоритма $a(x_i, w)$ и ошибку: $\varepsilon_i := L(a(x_i, w), y_i)$;
 - (c) сделать шаг градиентного спуска:

$$w := w - \eta L'_a(a(x_i, w), y_i) \varphi'(\langle w, x_i \rangle) x_i;$$

- (d) оценить значение функционала:

$$Q := (1 - \lambda)Q + \lambda \varepsilon_i;$$

пока значение Q не стабилизируется и/или веса w не перестанут изменяться.

Порядок выбора объектов

В случае стохастического градиентного спуска объекты следует выбирать случайным образом, однако существуют эвристики, направленные на улучшение сходимости:

- Перемешивание (shuffling): случайно выбирать объекты, попеременно из разных классов. Идея в том, что объекты из разных классов менее "похожи" чем объекты одного класса, поэтому вектор w будет сильнее изменяться.
- Можно выбирать объект с вероятностью, обратно пропорциональной величине ошибки на объекте. Следует учитывать, что такая эвристика делает метод чувствительным к шумам.

Способы инициализации весов

1. Инициализация вектора w нулями.
2. $w_j := \text{rand}\left(-\frac{1}{n}, \frac{1}{n}\right)$, где n — размерность пространства признаков.
3. Решение исходной оптимизационной задачи при условии статистически независимых признаков, линейной функции активации (φ) и квадратичной функции потерь (L):

$$w_j := \frac{\langle y, f_j \rangle}{\langle f_j, f_j \rangle}.$$

Параметр сглаживания

Для оценки функционала Q на каждой итерации используется его приближённое значение по методу экспоненциального сглаживания, откуда λ лучше брать порядка $\frac{1}{l}$.

Известные частные случаи алгоритма

Метод SG (при соответствующем выборе функций активации и потерь) является обобщением следующих эвристик подбора w и алгоритмов классификации:

- Адаптивный линейный элемент (Adalines);
- Правило Хэбба;
- Алгоритм k -средних (K-Means);
- Learning Vector Quantization (LVQ).

Преимущества SG

- Метод подходит для динамического (online) обучения.
- Алгоритм способен обучаться на избыточно больших выборках.
- Различные стратегии обучения позволяют адаптировать алгоритм для задач с избыточной или небольшой выборкой.

Недостатки SG и способы их устранения

- Возможны проблемы сходимости. Для борьбы с этим применяют технику встряхивания коэффициентов.
- При высокой размерности пространства признаков n и/или малой длине выборки l возможно переобучение. Для борьбы с этим применяют метод сокращения весов:

$$Q_\tau(w) = Q(w) + \frac{\tau}{2} \|w\|^2.$$

Тогда правило обновления весов принимает вид:

$$w := w(1 - \eta\tau) - \eta\nabla Q(w).$$

- При больших значениях $\langle w, x_i \rangle$ значение φ' может становиться близким к нулю. Для предотвращения этого состояния вводят нормализацию признаков:

$$x^j := \frac{x^j - x_{\min}^j}{x_{\max}^j - x_{\min}^j}, \quad j = 1, \dots, n,$$

где x_{\min}^j, x_{\max}^j — минимальное и максимальное значения признака j -го признака. Регуляризация, такая как weight decay, также помогает избежать "паралича".

Сходимость алгоритма

Сходимость гарантируется при выпуклой функции $Q(w)$ и выполнении следующих условий:

$$\eta_t \xrightarrow{t \rightarrow \infty} 0, \quad \sum_{t=1}^{\infty} \eta_t = \infty, \quad \sum_{t=1}^{\infty} \eta_t^2 < \infty.$$

Например, можно положить $\eta_t = \frac{\eta_0}{t}$, хотя на практике это не всегда удачно.

Задачи

Задача 1: Доказать сходимость алгоритма при условиях выше

Решение:

- Выпуклость функции:** Поскольку $Q(w)$ выпукла, мы можем использовать свойства выпуклых функций. Для любого w и w^* (где w^* — точка минимума функции Q) выполняется неравенство:

$$Q(w) \geq Q(w^*) + \nabla Q(w^*)^T (w - w^*).$$

- Итерация метода стохастического градиента:** Обновление весов в SGD задается следующим образом:

$$w_{t+1} = w_t - \eta_t \nabla Q(w_t; \xi_t),$$

где ξ_t — случайная переменная, представляющая выборку данных на итерации t .

- Анализ изменения функции:** Мы можем оценить изменение функции Q на каждой итерации:

$$Q(w_{t+1}) \leq Q(w_t) + \nabla Q(w_t; \xi_t)^T (w_{t+1} - w_t) + \frac{L}{2} \|w_{t+1} - w_t\|^2,$$

где L — константа Липшица для градиента ∇Q .

Подставляя обновление:

$$Q(w_{t+1}) \leq Q(w_t) - \eta_t \nabla Q(w_t; \xi_t)^T \nabla Q(w_t) + \frac{L}{2} \eta_t^2 \|\nabla Q(w_t; \xi_t)\|^2.$$

4. **Суммирование изменений:** Суммируя по всем итерациям, мы получаем:

$$\sum_{t=1}^T Q(w_{t+1}) - Q(w_1) \leq - \sum_{t=1}^T \eta_t \nabla Q(w_t; \xi_t)^T \nabla Q(w_t) + \sum_{t=1}^T \frac{L}{2} \eta_t^2 \|\nabla Q(w_t; \xi_t)\|^2.$$

5. **Использование условий:** Условия $\sum_{t=1}^{\infty} \eta_t = \infty$ и $\sum_{t=1}^{\infty} \eta_t^2 < \infty$ позволяют нам сделать вывод о том, что:

- Сумма шагов обучения стремится к бесконечности, что означает, что веса w_t будут продолжать обновляться.
- Сумма квадратов шагов обучения конечна, что позволяет контролировать величину изменений на каждой итерации.

6. **Сходимость к минимуму:** В результате, при условии, что $Q(w)$ выпукла, и учитывая условия на шаги обучения, мы можем утверждать, что последовательность w_t будет сходиться к некоторой точке w^* , которая является минимумом функции $Q(w)$.

Таким образом, метод стохастического градиента сходится к минимуму выпуклой функции $Q(w)$ при выполнении заданных условий.

Задача 2: Оценка вариации градиента

Условие: Пусть $\xi_1, \xi_2, \dots, \xi_n$ — независимые и одинаково распределённые (i.i.d.) случайные переменные, представляющие собой выборки из обучающего набора. Рассмотрим стохастический градиент $\nabla L(\theta; \xi_t)$.

Задача: Доказать, что математическое ожидание стохастического градиента совпадает с истинным градиентом функции потерь:

$$\mathbb{E}[\nabla L(\theta; \xi_t)] = \nabla L(\theta)$$

и оценить дисперсию $\text{Var}(\nabla L(\theta; \xi_t))$ в зависимости от размера выборки n .

Доказательство

1. **Определение стохастического градиента:** Пусть $L(\theta)$ — функция потерь, зависящая от параметров θ и от выборки ξ . Мы можем записать функцию потерь как среднее значение по всем данным:

$$L(\theta) = \frac{1}{n} \sum_{i=1}^n l(\theta; x_i, y_i),$$

где $l(\theta; x_i, y_i)$ — функция потерь для примера (x_i, y_i) .

2. **Истинный градиент:** Тогда истинный градиент функции потерь можно выразить как:

$$\nabla L(\theta) = \frac{1}{n} \sum_{i=1}^n \nabla l(\theta; x_i, y_i).$$

3. **Математическое ожидание стохастического градиента:** Теперь рассмотрим стохастический градиент:

$$\nabla L(\theta; \xi_t) = \nabla l(\theta; \xi_t),$$

где ξ_t — случайная выборка. Поскольку ξ_t выбирается из одного из n примеров, математическое ожидание стохастического градиента будет:

$$\mathbb{E}[\nabla L(\theta; \xi_t)] = \mathbb{E}[\nabla l(\theta; \xi_t)] = \frac{1}{n} \sum_{i=1}^n \nabla l(\theta; x_i, y_i) = \nabla L(\theta).$$

Таким образом, мы доказали, что:

$$\mathbb{E}[\nabla L(\theta; \xi_t)] = \nabla L(\theta).$$

4. **Оценка дисперсии:** Теперь найдём дисперсию стохастического градиента:

$$\text{Var}(\nabla L(\theta; \xi_t)) = \mathbb{E}[(\nabla L(\theta; \xi_t) - \mathbb{E}[\nabla L(\theta; \xi_t)])^2].$$

Подставим выражение для стохастического градиента:

$$\text{Var}(\nabla L(\theta; \xi_t)) = \mathbb{E}[(\nabla l(\theta; \xi_t) - \nabla L(\theta))^2].$$

Поскольку ξ_t является случайным выбором, мы можем использовать свойства дисперсии. Для n независимых и одинаково распределённых (i.i.d.) выборок дисперсия стохастического градиента будет уменьшаться с увеличением размера выборки:

$$\text{Var}(\nabla L(\theta; \xi_t)) = \frac{1}{n} \text{Var}(l(\theta; x, y)),$$

где (x, y) — случайная выборка из обучающего набора. Это означает, что дисперсия стохастического градиента уменьшается с увеличением размера выборки n .

Задача 3: Регуляризация и стохастический градиент

Условие

Рассмотрим функцию потерь с L2-регуляризацией:

$$L(\theta) = \frac{1}{n} \sum_{i=1}^n l(\theta; x_i, y_i) + \frac{\lambda}{2} \|\theta\|^2$$

где $l(\theta; x_i, y_i)$ — функция потерь для примера (x_i, y_i) , а λ — коэффициент регуляризации.

Задача

Обосновать, как регуляризация влияет на сходимость метода стохастического градиента, и показать, что использование регуляризации может помочь избежать переобучения, уменьшая значение функции потерь на валидационном наборе.

Влияние регуляризации на сходимость метода стохастического градиента

1. Сглаживание функции потерь:

- Добавление L2-регуляризации к функции потерь делает её более гладкой и выпуклой. Это связано с тем, что регуляризационный член $\frac{\lambda}{2}\|\theta\|^2$ добавляет "наказание" за большие значения параметров, что предотвращает резкие изменения градиента.
- Гладкость функции потерь способствует более стабильному обновлению параметров при использовании стохастического градиента. Это означает, что обновления параметров будут более предсказуемыми и менее подвержены шуму, что улучшает сходимость алгоритма.

2. Уменьшение переобучения:

- Регуляризация способствует уменьшению значений параметров модели, что, в свою очередь, снижает сложность модели. Это позволяет избежать переобучения, когда модель слишком точно подстраивается под тренировочные данные, включая шум.
- В результате, при использовании регуляризации, модель будет лучше обобщаться на новых данных, что выражается в меньшем значении функции потерь на валидационном наборе.

Доказательство эффекта регуляризации на валидационном наборе

1. Функция потерь на валидационном наборе:

- Пусть $L_{val}(\theta)$ — функция потерь на валидационном наборе. При использовании регуляризации, мы можем записать:

$$L_{val}(\theta) = \frac{1}{m} \sum_{j=1}^m l(\theta; x_j, y_j) + \frac{\lambda}{2} \|\theta\|^2$$

где m — количество примеров в валидационном наборе.

2. Сравнение значений функции потерь:

- Без регуляризации, модель может иметь высокую функцию потерь на валидационном наборе из-за переобучения. При добавлении L2-регуляризации, даже если функция потерь на тренировочном наборе остаётся низкой, регуляризация помогает поддерживать значение функции потерь на валидационном наборе на более низком уровне.

3. Кросс-валидация для выбора λ :

- Оптимальное значение λ можно выбрать с помощью кросс-валидации. Это позволяет находить компромисс между сложностью модели и её обобщающей способностью, что в конечном итоге приводит к меньшему значению функции потерь на валидационном наборе.

Заключение по задаче 3

Регуляризация, особенно L2-регуляризация, играет ключевую роль в улучшении сходимости метода стохастического градиента и в предотвращении переобучения модели. Она помогает сделать функцию потерь более гладкой и выпуклой, что способствует стабильности обновлений параметров. В результате, использование регуляризации приводит к лучшему обобщению модели и снижению значения функции потерь на валидационном наборе, что является важным аспектом при разработке надёжных моделей в машинном обучении.

2.2. Метод наименьших квадратов (МНК) в общем случае

2.2.1. Про линейную регрессию и МНК

Линейная регрессия — это метод анализа данных, который используется для определения линейной зависимости между зависимой переменной y и одной или несколькими независимыми переменными x_1, x_2, \dots, x_n . Цель метода заключается в построении модели, которая минимизирует ошибку предсказания.

Метод наименьших квадратов (МНК) — это наиболее распространённый способ нахождения коэффициентов линейной регрессии. Он минимизирует сумму квадратов отклонений предсказанных значений от наблюдаемых. Таким образом, МНК позволяет определить такие коэффициенты $\beta_0, \beta_1, \dots, \beta_n$, которые обеспечивают наилучшее соответствие модели данным.

Основная идея МНК: минимизация ошибки предсказания, заданной формулой:

$$Q(\beta) = \sum_{i=1}^N (y_i - \hat{y}_i)^2,$$

где y_i — наблюдаемые значения, а \hat{y}_i — предсказанные моделью значения.

2.2.2. МНК в общем случае

Определение: В общем случае задача линейной регрессии может быть представлена в матричной форме:

$$\mathbf{y} = X\beta + \epsilon,$$

где:
- y — вектор целевых значений ($N \times 1$),
- X — матрица признаков ($N \times p$),
- β — вектор коэффициентов модели ($p \times 1$),
- ϵ — вектор ошибок ($N \times 1$).

Решение задачи МНК определяется как:

$$\hat{\beta} = (X^T X)^{-1} X^T \mathbf{y}.$$

Интерпретация: Этот результат минимизирует сумму квадратов остатков $\epsilon = \mathbf{y} - X\beta$. В случае, если матрица $X^T X$ вырожденная, решение может быть некорректным или недоступным, что требует применения регуляризации.

2.2.3. Проблемы и ограничения МНК

Несмотря на простоту и эффективность, метод МНК имеет ограничения:

1. Мультиколлинеарность

- **Определение:** мультиколлинеарность возникает, когда между независимыми переменными в матрице признаков X существует сильная линейная зависимость;
- **Почему это проблема:** при мультиколлинеарности матрица $X^T X$ становится плохо обусловленной (или даже вырожденной), что затрудняет нахождение её обратной матрицы. Это может привести к неустойчивым решениям, при которых малые изменения в данных существенно изменяют значения коэффициентов.

2. Чувствительность к выбросам

- **Определение:** МНК минимизирует сумму квадратов ошибок, что делает его очень чувствительным к выбросам (аномальным точкам);
- **Почему это проблема:** выбросы имеют большое влияние на значение целевой функции $Q(\beta)$, что может привести к сильному смещению коэффициентов регрессии.

3. Нарушение предположений:

МНК предполагает линейность модели, гомоскедастичность (постоянную дисперсию ошибок) и отсутствие автокорреляции.

4. Высокая вычислительная сложность

- **Определение:** МНК требует вычисления матрицы $X^T X$ и её обратной, что имеет временную сложность $O(Np^2 + p^3)O(Np^2 + p^3)$, где N — количество наблюдений, p — количество признаков;
- **Почему это проблема:** для больших наборов данных с большим количеством признаков вычислительная сложность становится значительной, что может сделать процесс обучения долгим.

2.2.4. Задачи

Задача 1: Докажите, что решение системы нормальных уравнений для метода наименьших квадратов существует и единственno, если матрица $X^T X$ положительно определённая.

Решение: Метод наименьших квадратов минимизирует квадратичную функцию:

$$Q(\beta) = \|y - X\beta\|^2 = (y - X\beta)^T(y - X\beta)$$

Рассмотрим стационарные точки функции $Q(\beta)$, задаваемые системой нормальных уравнений:

$$X^T X \beta = X^T y$$

- Условие существования решения:

Решение существует, если матрица $X^T X$ невырожденная, то есть её детерминант

$\det(X^T X) \neq 0$. Это выполняется, если признаки в X линейно независимы (нет мультиколлинеарности).

- Условие единственности решения:

Если $X^T X$ положительно определённая, то:

1. $(X^T X)$ симметрична.
2. Для любого ненулевого вектора v , $v^T (X^T X) v > 0$.

Положительная определённость гарантирует, что квадратичная форма $Q(\beta)$ строго выпуклая, а значит, имеет единственную точку минимума.

Задача 2: Опишите, как изменится целевая функция метода наименьших квадратов $Q(\beta) = \|y - X\beta\|^2$, если в данных присутствует выброс. Почему минимизация квадратичной ошибки делает метод чувствительным к таким точкам?

Решение: Выбросы увеличивают квадратичную ошибку, так как вклад отклонений от линии регрессии для таких точек пропорционален квадрату расстояния. Это означает, что несколько больших ошибок могут доминировать над множеством малых, и решение будет смещено в сторону выбросов. Например, если одна ошибка вдвое больше остальных, её вклад в целевую функцию будет в четыре раза больше. Это делает МНК очень чувствительным к выбросам и может привести к неверным коэффициентам модели.

Задача 3: Реализуйте простой случай МНК для одномерной линейной регрессии, где $X = [1, 2, 3]$, $y = [2, 4, 6]$. Найдите коэффициенты β_0 и β_1 .

Решение: Для одномерного случая формула нормальных уравнений имеет вид:

$$\hat{\beta} = (X^T X)^{-1} X^T y.$$

Подставляя значения:

$$X = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{bmatrix}, \quad y = \begin{bmatrix} 2 \\ 4 \\ 6 \end{bmatrix}.$$

Рассчитаем:

$$X^T X = \begin{bmatrix} 3 & 6 \\ 6 & 14 \end{bmatrix}, \quad X^T y = \begin{bmatrix} 12 \\ 28 \end{bmatrix}.$$

Получаем:

$$\hat{\beta} = \begin{bmatrix} 3 & 6 \\ 6 & 14 \end{bmatrix}^{-1} \begin{bmatrix} 12 \\ 28 \end{bmatrix}.$$

После вычислений $\hat{\beta} = \begin{bmatrix} 0 \\ 2 \end{bmatrix}$. Таким образом, уравнение линейной регрессии: $y = 2x$.

2.3. Вероятностные функции потерь

2.3.1. Принцип максимума правдоподобия

Предположим, что $X \times Y$ - **вероятностное пространство** с некоторой **плотностью совместного распределения** пары объект-ответ $p(x, y)$. Пусть X^l - *простая* (i.i.d., или independent identically distributed - независимо из одного и того же распределения) выборка: $(x_i, y_i)^l$, порожденная $p(x, y)$.

Задача состоит в том, чтобы оценить по выборке X^l плотность распределения $p(x, y)$. Получив оценку плотности, то с его помощью возможна классификация объекта x - мы сможем вычислять вероятность класса y для любого объекта x .

Далее введем **параметризацию** плотности, взяв за основу формулу условной вероятности: $p(x, y) = P(y|x, w)p(x)$, где $p(x, y)$ - искомая плотность; $P(y|x, w)$ - модель условной вероятности класса с параметром w ; $p(x)$ - непараметризуемое распределение в пространстве X . Возможны и другие способы введения параметризации, однако сейчас будет рассматриваться именно этот.

Так как выборка простая, то плотность порожденной выборки является произведением плотностей отдельных порожденных пар (x_i, y_i) . Таким образом, $\prod_{i=1}^l p(x_i, y_i)$ - *правдоподобие данных*.

В качестве критерия оптимизации возьмем один из фундаментальных методов в математической статистике - **принцип максимума правдоподобия**:

$$\prod_{i=1}^l p(x_i, y_i) = \prod_{i=1}^l P(y_i|x_i, w)p(x_i) \rightarrow \max_w$$

В силу того, что $p(x_i)$ - сомножитель, не зависящий от w , от него можно избавиться.

$$\prod_{i=1}^l P(y_i|x_i, w) \rightarrow \max_w$$

Поскольку стоит задача максимизации, критерий в виде произведения по всем объектам выборки неудобен. Чтобы избавиться от произведения, прологарифмируем критерий. Логарифм - монотонная функция, поэтому несущественно, что мы оптимизируем - функционал или его логарифм.

Логарифм правдоподобия (log-likelihood, log-loss):

$$L(w) = \sum_{i=1}^l \log P(y_i|x_i, w) \rightarrow \max_w$$

2.3.2. Связь правдоподобия и аппроксимации эмпирического риска

Посмотрим на одну и ту же задачу классификации на два класса $Y = \{+1; -1\}$ с двух сторон:

1. $P(y|x, w)$ - вероятностная модель классификации.
2. $g(x, w)$ - разделяющая (дискриминантная) функция - геометрический взгляд на задачу.

Критерии, возникающие в разных случаях:

1. *Максимизация правдоподобия* (Maximum Likelihood):

$$L(w) = \sum_{i=1}^l \log P(y_i|x_i, w) \rightarrow \max_w;$$

2. *Минимизация аппроксимированного эмпирического риска*:

$$Q(w) = \sum_{i=1}^l L(y_i g(x_i, w)) \rightarrow \min_w;$$

Здесь $L(M)$ - функция потерь; $y_i g(x_i, w)$ - значение отступа.

Оба критерия представляют собой оптимизацию некоторой величины, являющейся суммой по всем объектам выборки, по параметру. Каждое слагаемое суммы зависит только от одного объекта.

Эти два принципа **эквиваленты**, если положить:

$$-\log P(y_i|x_i, w) = L(y_i g(x_i, w))$$

2.3.3. Вероятностный смысл регуляризации

Рассматривается двухуровневая модель порождения данных:

1. $P(y|x, w)$ - вероятностная модель данных.
2. $p(w; \gamma)$ - априорное распределение параметров модели.
3. γ - вектор гиперпараметров.

Теперь не только появление выборки X^l , но и модели w полагается стохастическим.

Совместное правдоподобие данных и модели по формуле условной плотности:

$$p(X^l, w) = p(X^l|w)p(w; \gamma)$$

Принцип максимума апостериорной вероятности (Maximum a Posteriori Probability, MAP):

$$L(w) = \log p(X^l, w) = \sum_{i=1}^l \log P(y_i|x_i, w) + \log p(w; \gamma) \rightarrow \max_w$$

Таким образом, слагаемое $-\log p(w; \gamma)$ является **регуляризатором** с вероятностной точки зрения.

2.3.4. Задачи

Задача 1. Какому регуляризатору соответствует апостериорное распределение параметров модели, имеющее вид распределения Гаусса?

Решение:

Веса w_j независимы, $E[w_j] = 0$, $D[w_j] = C$.

$$p(w; C) = \frac{1}{(2\pi C)^{\frac{n}{2}}} \exp\left(-\frac{\|w^2\|}{2C}\right), \|w^2\| = \sum_{j=1}^n w_j^2$$

$$-\ln p(w; C) = \frac{1}{2C} \|w^2\| + const$$

От константы можно избавиться:

$$-\ln p(w; C) = \frac{1}{2C} \|w^2\|$$

Получаем квадратичный (L_2) регуляризатор. C является гиперпараметром, $\tau = \frac{1}{C}$ - коэффициент регуляризации.

Задача 2. Какому виду регуляризации соответствует апостериорное распределение параметров модели, имеющее вид распределения Лапласа?

Решение:

Веса w_j независимы, $E[w_j] = 0$, $D[w_j] = C$.

$$p(w; C) = \frac{1}{(2C)^n} \exp\left(-\frac{\|w\|}{C}\right), \|w\| = \sum_{j=1}^n |w_j|$$

$$-\ln p(w; C) = \frac{1}{C} \|w\| + const$$

От константы можно избавиться:

$$-\ln p(w; C) = \frac{1}{C} \|w\|$$

Получаем абсолютный (L_1) регуляризатор. C является гиперпараметром, $\tau = \frac{1}{C}$ - коэффициент регуляризации.

Задача 3. Найти вид апостериорного распределения параметров модели, соответствующий регуляризатору Elastic Net:

$$R(w; C_1; C_2) = \frac{1}{C_1} \sum_{i=1}^l |w_i| + \frac{1}{2C_2} \sum_{i=1}^l w_i^2$$

Решение:

$$-\ln p(w; C_1; C_2) = \frac{1}{C_1} \|w\| + \frac{1}{2C_2} \|w^2\|$$

С точностью до умножения на константу, получим:

$$p(w; C_1; C_2) = \exp\left(-\frac{\|w\|}{C_1}\right) \exp\left(-\frac{\|w^2\|}{2C_2}\right)$$

$$p(w; C_1; C_2) = \exp\left(-\frac{\|w\|}{C_1} - \frac{\|w^2\|}{2C_2}\right)$$

2.4. Методы оценки и проверки моделей

Оценка и проверка моделей являются важными этапами в построении машинного обучения. Эти методы позволяют определить, насколько хорошо модель обучается на данных и обобщает свои выводы на новых, ранее невиданных данных. Ниже представлены основные подходы к оценке и проверке моделей.

Кросс-валидация

Кросс-валидация — это метод проверки модели, который заключается в разбиении данных на несколько подвыборок (folds). Основные виды кросс-валидации:

- **K-блочная кросс-валидация** (K-fold cross-validation): данные делятся на K частей, и обучение проводится на $K - 1$ частях, а тестирование на оставшейся части. Процесс повторяется K раз, чтобы каждая часть данных использовалась для тестирования.
- **Leave-One-Out (LOO)**: частный случай кросс-валидации, где тестовая выборка состоит из одного наблюдения, а оставшиеся используются для обучения. Этот метод особенно полезен для небольших наборов данных, но может быть вычислительно затратным.
- **Stratified K-fold**: разновидность K-блочной кросс-валидации, где сохраняются пропорции классов в каждой из выборок, что особенно важно для несбалансированных данных.

Кросс-валидация помогает уменьшить риск переобучения и получить более надежные оценки качества модели.

Метрики качества

Для оценки модели применяются различные метрики, выбор которых зависит от задачи (регрессия или классификация):

- **Для регрессии:**

- Среднеквадратичная ошибка (MSE):

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2.$$

Эта метрика чувствительна к выбросам, так как большие ошибки квадратично увеличивают значение MSE.

- Средняя абсолютная ошибка (MAE):

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|.$$

MAE более устойчива к выбросам, так как ошибки учитываются линейно.

- Коэффициент детерминации (R^2):

$$R^2 = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2}.$$

Значение R^2 показывает, какую долю дисперсии целевой переменной объясняет модель.

- **Для классификации:**

- Точность (Accuracy):

$$\text{Accuracy} = \frac{\text{Количество верных предсказаний}}{\text{Общее количество наблюдений}}.$$

Используется для сбалансированных данных.

- Precision, Recall и F_1 -мера:

$$\text{Precision} = \frac{TP}{TP + FP}, \quad \text{Recall} = \frac{TP}{TP + FN}, \quad F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}.$$

Эти метрики особенно важны для несбалансированных классов.

- ROC-кривые и площадь под кривой (AUC): ROC-кривая показывает соотношение TPR (True Positive Rate) и FPR (False Positive Rate) при различных порогах классификации, а AUC характеризует общую способность модели различать классы.

Разделение данных

Одним из простейших методов проверки модели является разделение данных на обучающую и тестовую выборки. Чаще всего используется пропорция 70%/30% или 80%/20%. Однако этот метод имеет недостаток: если данные случайно разделены неудачно, это может привести к неверным оценкам качества модели.

Для улучшения оценки часто используется тройное разбиение данных:

- **Обучающая выборка** (Train set): используется для обучения модели.
- **Валидационная выборка** (Validation set): используется для подбора гиперпараметров и предотвращения переобучения.
- **Тестовая выборка** (Test set): применяется для окончательной оценки модели.

Проблемы и рекомендации

- **Переобучение:** Если модель слишком сложная, она может хорошо работать на обучающих данных, но плохо обобщаться на тестовые. Регуляризация и кросс-валидация помогают бороться с этой проблемой.
- **Недообучение:** Слишком простые модели могут не учитывать важные зависимости в данных. Следует выбирать более сложные модели или добавлять новые признаки.
- **Сбалансированность данных:** Для несбалансированных данных рекомендуется использовать метрики, такие как Precision, Recall или AUC.

Задачи

1. Для датасета (сгенерируйте сами) с 10 000 объектов примените 5-блочную кросс-валидацию. Опишите, как будут разделены данные на обучающие и тестовые выборки на каждом шаге. Рассчитайте среднее значение метрики Accuracy, если на каждом шаге она принимает значения: 0.85, 0.87, 0.86, 0.84, 0.88.
2. Для задачи регрессии выберите подходящую метрику качества из MSE, MAE или R^2 и объясните, почему вы сделали такой выбор. Рассчитайте её значение для предсказаний $\hat{y} = [3, 5, 2, 7]$ и истинных значений $y = [3, 5, 4, 6]$.
3. В задаче классификации модель предсказала 100 объектов, из которых 70 были классифицированы правильно, 20 — ложно положительными, а 10 — ложно отрицательными. Рассчитайте Precision, Recall и F_1 -меру.

2.5. Многоклассовая классификация

Задача многоклассовой классификации решает проблему нахождения принадлежности объекта к одному из K классов: $Y = \{1, 2, \dots, K\}$. В этом разделе будут разобраны некоторые из самых популярных подходов: one-vs-all, all-vs-all и многоклассовая логистическая регрессия. В качестве примера возьмем датасет из хендбука Яндекса, продемонстрированный на Рис. 2.4.

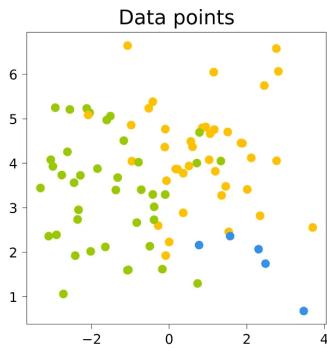


Рис. 2.4. Пример датасета задачи многоклассовой классификации

2.5.1. Один против всех (one-versus-all)

Обучим K линейных классификаторов $b_1(x), \dots, b_K(x)$, выдающих оценки принадлежности классам $1, \dots, K$ соответственно. В случае с линейными моделями эти классификаторы будут иметь вид $b_k(x) = \text{sign}(\langle w_k, x \rangle + w_{0k})$

Классификатор с номером k будем обучать по выборке $(x_i, 2\mathbb{I}[y_i = k] - 1)_{i=1}^\ell$; иными словами, мы учим классификатор отличать k -й класс от всех остальных.

Логично, чтобы итоговый классификатор выдавал класс, соответствующий самому уверенному из бинарных алгоритмов. Уверенность можно в каком-то смысле измерить с помощью значений линейных функций:

$$a(x) = \arg \max_{k \in Y} (\langle w_k, x \rangle + w_{0k})$$

Давайте посмотрим, что даст этот подход применительно к нашему датасету. Обучим три линейных модели, отличающих один класс от остальных (Рис. 2.5).

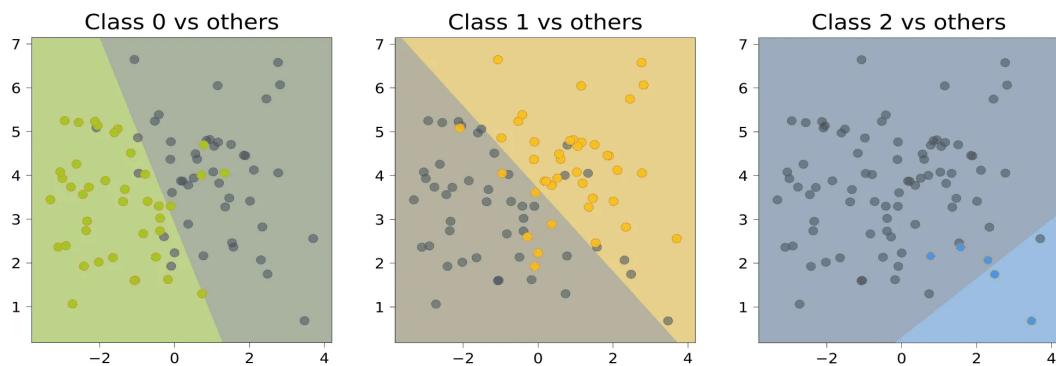


Рис. 2.5. Каждая из трех моделей, отделяющие свои классы

Теперь сравним значения линейных функций на Рис. 2.6.

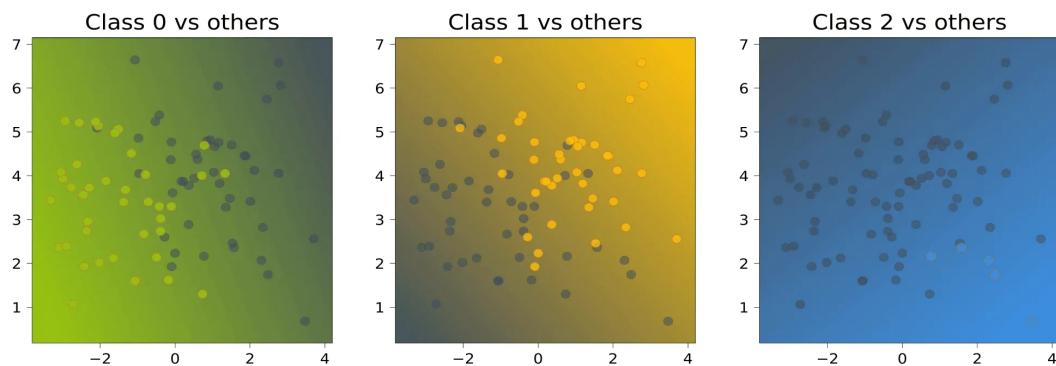


Рис. 2.6. Значения функций моделей

и для каждой точки выберем тот класс, которому соответствует большее значение, то есть самый «уверенный» классификатор, и изобразим это на Рис. 2.7

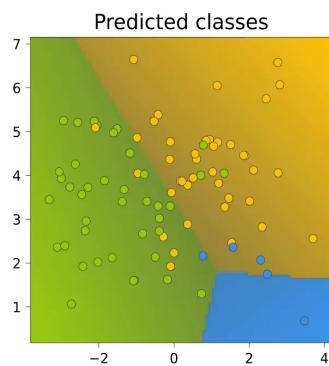


Рис. 2.7. Итог разделения пространства на классы

Хочется сказать, что самый маленький класс «обидели».

Проблема данного подхода заключается в том, что каждый из классификаторов $b_1(x), \dots, b_K(x)$ обучается на своей выборке, и значения линейных функций $\langle w_k, x \rangle + w_{0k}$ или, проще говоря, "выходы" классификаторов могут иметь разные масштабы. Из-за этого сравнивать их будет неправильно. Нормировать вектора весов, чтобы они выдавали ответы в одной и той же шкале, не всегда может быть разумным решением: так, в случае с SVM веса перестанут являться решением задачи, поскольку нормировка изменит норму весов.

Задача

Подумайте, какой простой двумерный датасет особенно плохо работает с данным подходом с линейным классификатором (что вызвано исключительно характером распределения классов, а не выбросами, количеством данных и т.п.).

Ответ

Примером такого датасета могут послужить три класса, такие что один находится между двумя другими, как на Рис. 2.8. В этом случае нельзя полагаться на значения разделяющей функции, так как, например, зеленый класс лежит сильно дальше границы желтого и синего классов, чем сам желтый класс. Поэтому, желто-синий классификатор будет относить зеленые точки к желтому цвету с большей уверенностью, чем желтые.

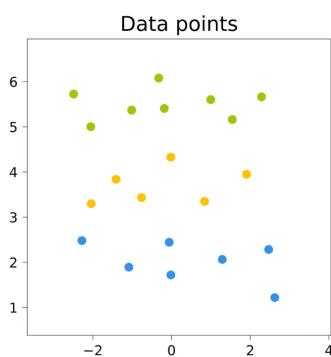


Рис. 2.8. Контрпример к one-vs-all

2.5.2. Все против всех (all-versus-all)

Обучим C_k^2 классификаторов $a_{ij}(x), i, j = 1, \dots, K, i \neq j$. Например, в случае с линейными моделями эти модели будут иметь вид

$$a_{ij}(x) = \text{sign}(\langle w_{ij}, x \rangle + w_{0,ij})$$

Классификатор $a_{ij}(x)$ будем настраивать по подвыборке $X_{ij} \subset X$, содержащей только объекты классов i и j . Соответственно, классификатор $a_{ij}(x)$ будет выдавать для любого объекта либо класс i , либо класс j . Проиллюстрируем это для нашей выборки на Рис. 2.9

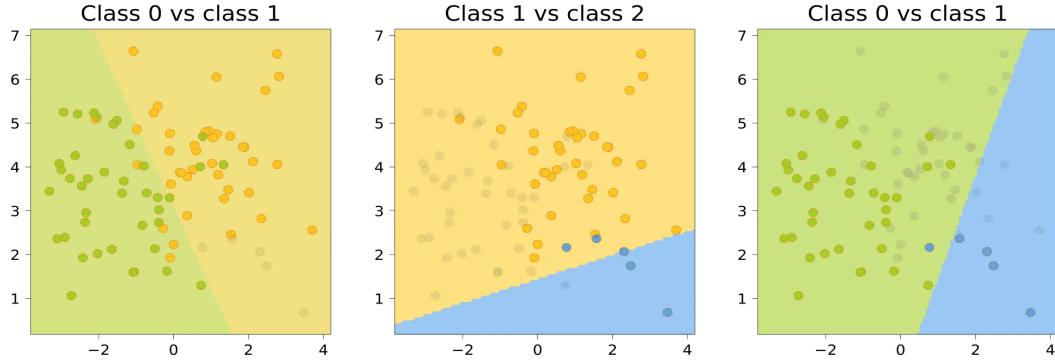


Рис. 2.9. Классификаторы, разделяющие по два класса

Чтобы классифицировать новый объект, подадим его на вход каждого из построенных бинарных классификаторов. Каждый из них проголосует за свой класс; в качестве ответа выберем тот класс, за который наберется больше всего голосов:

$$a(x) = \arg \max_{k \in Y} \sum_{i=1}^K \sum_{j \neq i} \mathbb{I}[a_{ij}(x) = k]$$

Для нашего датасета получается следующая картинка на Рис. 2.10

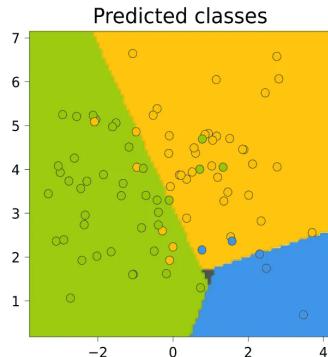


Рис. 2.10. Итоговый результат для all-vs-all

Обратите внимание на серый треугольник на стыке областей. Это точки, для которых голоса разделились (в данном случае каждый классификатор выдал какой-то свой класс, то есть у каждого класса было по одному голосу). Для этих точек нет явного способа выдать обоснованное предсказание.

2.5.3. Многоклассовая логистическая регрессия

Некоторые методы бинарной классификации можно напрямую обобщить на случай многих классов. Выясним, как это можно проделать с логистической регрессией.

В логистической регрессии для двух классов мы строили линейную модель

$$b(x) = \langle w, x \rangle + w_0$$

а затем переводили её прогноз в вероятность с помощью сигмоидной функции $\sigma(M) = \frac{1}{1+\exp(-M)}$. Допустим, что мы теперь решаем многоклассовую задачу и построили K линейных моделей

$$b_k(x) = \langle w_k, x \rangle + w_{0k}$$

каждая из которых даёт оценку принадлежности объекта одному из классов. Как преобразовать вектор оценок $(b_1(x), \dots, b_K(x))$ в вероятности? Для этого можно воспользоваться оператором $\text{softmax}(z_1, \dots, z_K)$,

Задача

Подумайте, как должен выглядеть этот оператор, зная, что в случае двух классов он совпадает с сигмоидой, принимает значения от нуля до единицы и равен вектору вероятностей принадлежности каждому из классов.

Ответ

$$\text{softmax}(z_1, \dots, z_K) = \left(\frac{\exp(z_1)}{\sum_{k=1}^K \exp(z_k)}, \dots, \frac{\exp(z_K)}{\sum_{k=1}^K \exp(z_k)} \right)$$

В этом случае вероятность k -го класса будет выражаться как

$$P(y = k|x, w) = \frac{\exp(\langle w_k, x \rangle + w_{0k})}{\sum_{j=1}^K \exp(\langle w_j, x \rangle + w_{0j})}$$

Обучать эти веса предлагается с помощью метода максимального правдоподобия: так же, как и в случае с двухклассовой логистической регрессией:

$$\sum_{i=1}^{\ell} \log P(y = y_i|x_i, w) \rightarrow \max_{w_1, \dots, w_K}$$

Задача

Подумайте, какой существует простой многоклассовый (нелинейный) классификатор, использующий только расстояние между объектами и не требующий обучения (у него отсутствуют параметры, есть только гиперпараметр).

Ответ

Таким классификатором является KNN, или метод k ближайших соседей. Этот алгоритм причисляет объект к самому популярному классу среди его k ближайших соседей.

Глава 3

Основы нейросетевых моделей

3.1. Полносвязная нейронная сеть (Персепtron).

3.1.1. Модель нейрона МакКаллока-Питтса

Рассмотрим модель нейрона МакКаллока-Питтса (1943) [2]. Множество объектов X (количество элементов множества M), множество ответов Y (количество элементов множества H_L). Признаки объектов задаются вектором функций $\vec{f}(x) = [f_1(x), f_2(x), \dots, f_n(x)]^T$ таких, что $f_j : X \rightarrow \mathbb{R}$. Для объекта $x_m \in X$ с помощью этих функций определяется вектор вещественных признаков $\vec{x}_m = [x_m^1, x_m^2, \dots, x_m^n]^T$, где $x_m^j = f_j(x_m)$. Нейрон вычисляет функцию активации σ от линейной комбинации вектора признаков \vec{x}_m с весами \vec{w} . То есть выход нейрона $a(\vec{x}_m, \vec{w})$ вычисляется по формуле:

$$a(\vec{x}_m, \vec{w}) = \sigma\left(\sum_{i=1}^n x_m^i w_i - w_0\right) = \sigma(\langle \vec{x}_m, \vec{w} \rangle)$$

Математическая модель нейрона изображена на рисунке 3.1. Функцией активации σ может быть любая непрерывная нелинейная функция. Нелинейность функции нужна, чтобы иметь возможность приблизить любую непрерывную функцию с желаемой точностью (теорема Горбаня, 1998). В формуле вектор признаков \vec{x}_m дополнен элементом -1 , который имеет вес w_0 , что соответствует физической интерпретации нейрона. Нейрон m получает на вход множество электрических сигналов x_m^i по дендритам i . Каждый дендрит i имеет свою толщину, и соответственно проводимость w_i . Чем выше проводимость w_i , тем больший вклад сигнала с данного дендрита в общую сумму. Нейрон суммирует сигналы с дендритов и если сумма выше порогового значения w_0 , то он передаёт информацию дальше по аксону. Сигнал на выходе нейрона, то есть аксоне, определяется функцией активации σ .

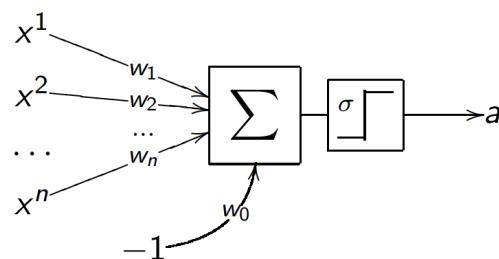


Рис. 3.1. Математическое описание модели нейрона МакКаллока-Питтса.

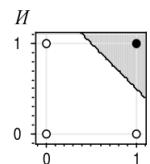
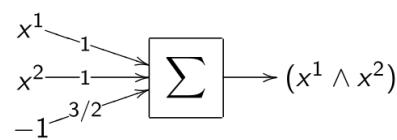
3.1.2. Реализация логических функций с помощью нейрона

Рассмотрим применение описанного выше нейрона для реализации простейших логических функций.

Логическая функция «и». Обозначение \wedge . На вход поступают признаки x^1, x^2 , результат функции $a = x^1 \wedge x^2$. Нейронной реализацией будет $a = [x^1 + x^2 - \frac{3}{2} > 0]$, где функция активации $[x > 0]$ равна 1, если $x > 0$, и равна 0 в противном случае.

x^1	x^2	$a = x^1 \wedge x^2$
0	0	0
0	1	0
1	0	0
1	1	1

(a) Таблица истинности функции «и»

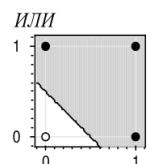
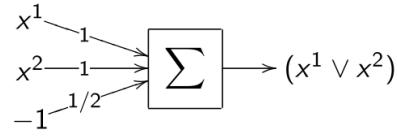


(b) Модель нейрона, описывающего функцию «и»

Логическая функция «или». Обозначение \vee . На вход поступают признаки x^1, x^2 , результат функции $a = x^1 \vee x^2$. Нейронной реализацией будет $a = [x^1 + x^2 - \frac{1}{2} > 0]$, где функция активации $[x > 0]$ равна 1, если $x > 0$, и равна 0 в противном случае.

x^1	x^2	$a = x^1 \vee x^2$
0	0	0
0	1	1
1	0	1
1	1	1

(a) Таблица истинности функции «или»



(b) Модель нейрона, описывающего функцию «или»

Задача 1. Привести модель нейрона, описывающего логическую функцию «не» $\neg x^1$.

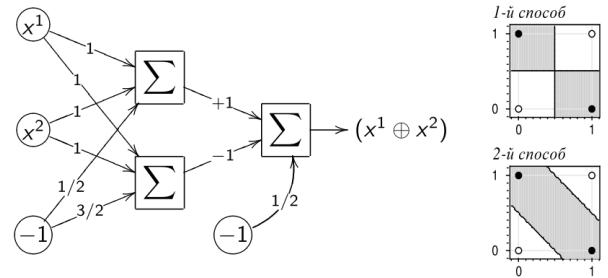
Решение:

$$a = [-x^1 + 1/2 > 0]$$

Логическая функция «исключающее или». Обозначение \oplus . Данная функция не реализуема с помощью одного нейрона. Но может быть реализована нейронной сетью $[n_1, n_2, n_3]$: $n_1 = (x^1 \vee x^2)$, $n_2 = (x^1 \wedge x^2)$, $n_3 = [n_1 - n_2 - \frac{1}{2} > 0]$.

x^1	x^2	$a = x^1 \oplus x^2$
0	0	0
0	1	1
1	0	1
1	1	0

(a) Таблица истинности функции «исключающее или»



(b) Модель нейрона, описывающего функцию «исключающее или»

Задача 2. Привести модель нейронной сети, описывающей логическую функцию $f(x^1, x^2, x^3)$, заданную таблицей истинности:

x^1	x^2	x^3	$f(x^1, x^2, x^3)$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

Решение:

Используем ранее описанные нейроны «и» \wedge , «или» \vee , «не» \neg . Перепишем функцию, используя эти операции: $f = ((\neg x^1) \wedge ((\neg x^2) \wedge x^3)) \vee (x^1 \wedge ((\neg x^2) \vee x^3))$. Конструируем нейроны: $n_1 = \neg x^1$, $n_2 = \neg x^2$, $n_3 = n_2 \wedge x^3$, $n_4 = n_1 \wedge n_3$, $n_5 = n_2 \vee x^3$, $n_6 = x^1 \wedge n_5$, $n_7 = n_4 \vee n_6$. Итоговая многослойная нейронная сеть состоит из нейронов n_1, n_2, \dots, n_7 . Результат сети находится на выходе нейрона n_7 .

3.1.3. Область применимости многослойных нейронных сетей

Согласно [1]

1. Двухслойная сеть в $0, 1^n$ позволяет реализовать произвольную булеву функцию.
2. Линейный нейрон в \mathbb{R}^n отделяет полупространство признаков гиперплоскостью. Тогда двухслойная сеть позволяет отделить многогранную область, не обязательно выпуклую и связную.
3. Согласно теоремы Горбаня (1998) с помощью линейных операций и одной нелинейной функции активации можно приблизить любую непрерывную функцию с любой желаемой точностью.

3.1.4. Полносвязная нейронная сеть

Обобщением рассмотренной выше модели является полносвязная нейронная сеть (рис. 3.5). Сеть состоит из входного слоя (жёлтый цвет), скрытых слоёв (зелёный цвет), и выходного слоя (оранжевый цвет). Выход одного слоя, поступает на вход другого. Обозначим количество нейронов в слое l за H_l , в каждом слое оно может быть разным, $l \in \{1, 2, \dots, L\}$. Всего L слоёв. Вектор x^l – это выход l -ого слоя и вход $l + 1$, если $l \neq L$, то есть x^l – не последний слой. x^0 – это вход нейронной сети. Матрицы коэффициентов перехода между слоями обозначим за W^l . W^l – это матрица перехода между слоями $l - 1$ и l .

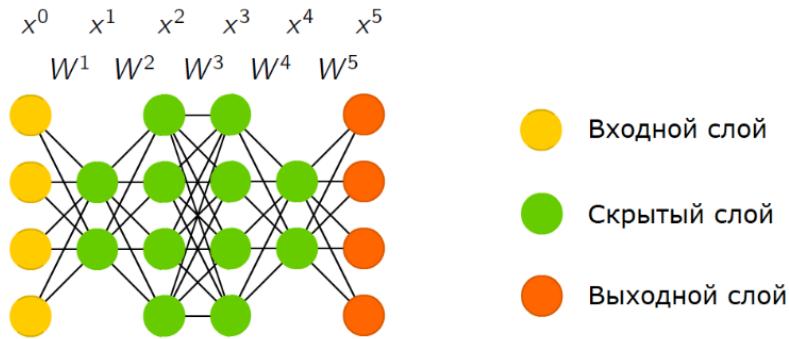


Рис. 3.5. Полносвязная нейронная сеть.

Задачей обучения нейронной сети является минимизация средних потерь на обучающей выборке X ($|X| = M$):

$$Q(\vec{W}, X) = \frac{1}{m} \sum_{m=1}^M \mathcal{L}(\vec{W}, x_m) \rightarrow \min_{\vec{W}}$$

Пусть объект x описывается вектором признаков $\vec{x} = [x_1, x_2, \dots, x_n]^T$. В задаче классификации множество ответов Y состоит из H_L элементов. Тогда последний слой нейронной сети x^L должен содержать H_L элементов. Нейронная сеть предсказывает ответ k , если на её выходе наблюдается вектор $x^L = [0, \dots, 0, 1, 0, \dots, 0]^T$, где единица стоит на k -ом месте.

Функция потерь может быть, например, квадратичной. Для объекта x функция потерь вычисляется по формуле $\mathcal{L}(\vec{W}^{(t)}, x) = \frac{1}{2} \sum_{i=1}^{H_L} (x_i^L((\vec{W}^{(t)}, x) - y_i)^2$, где \vec{y} – вектор из всех нулей кроме единицы на месте порядкового правильного ответа из множества ответов.

Рассмотрим различные методы обучения полносвязной нейронной сети.

3.1.5. Градиентный спуск

1. Выбираем какое-то начальное приближение вектора матриц перехода $\vec{W}^{(0)}$.
2. Итерационный процесс:

$$\vec{W}^{(t+1)} = \vec{W}^{(t)} - h \cdot \vec{\nabla} Q(\vec{W}^{(t)}, X)$$

где h – градиентный шаг или темп обучения.

Для рассмотренного ранее эмпирического риска:

$$\vec{W}^{(t+1)} = \vec{W}^{(t)} - \frac{h}{m} \cdot \vec{\nabla} \sum_{m=1}^M \mathcal{L}(\vec{W}^{(t)}, x_m)$$

3. Повторяем итерационный процесс, пока эмпирический риск $Q(\vec{W}^{(t)}, X)$ или вектор матриц перехода \vec{W} не сойдутся. Критерий сходимости может быть абсолютным, то есть когда модуль разности значений на последовательных шагах итерационного процесса меньше какого-то значения ε . Например, для эмпирического риска:

$$|Q^{(t+1)} - Q^{(t)}| < \varepsilon$$

Или может быть относительным, когда модуль отношения разности значений на последовательных шагах итерационного процесса к наименьшему, или наибольшему значению, или к первому меньше ε . Например, для эмпирического риска:

$$\left| \frac{Q^{(t+1)} - Q^{(t)}}{\min\{Q^{(t+1)}, Q^{(t)}\}} \right| < \varepsilon$$

При этом для эмпирического риска и для вектора матриц перехода значение ε может быть разным.

Для вектора матриц перехода можно предложить следующий способ:

$$\left\| \vec{W}^{(t+1)} - \vec{W}^{(t)} \right\|_1 = \sum_{l=1}^L \sum_{m=1}^{m_l} \sum_{n=1}^{n_l} |W_{mn}^{l(t+1)} - W_{mn}^{l(t)}|$$

где вектор матрица перехода рассматривается как вектор всех её элементов, и применяется первая норма. Размерность матрицы W^l равна $m_l \times n_l$. $W_{mn}^{l(t)}$ – это элемент в m -ой строке n -ого столбца матрицы перехода между слоями $l-1$ и l на шаге t итерационного процесса.

Недостатком данного метода является низкая скорость сходимости. Для ускорения применяется метод стохастического градиентного спуска.

3.1.6. Стохастический градиентный спуск

В отличие от обычного градиентного спуска, в котором вектор матриц перехода изменяется пропорционально градиенту эмпирического риска для всех объектов, в стохастическом градиенте вектор матриц перехода изменяется пропорционально функции потерь для одного объекта.

Алгоритм стохастического градиента:

1. Выбираем какое-то начальное приближение вектора матриц перехода $\vec{W}^{(0)}$. Вычисляем первое приближение эмпирического риска:

$$Q^{(0)}(\vec{W}^{(0)}, X) = \frac{1}{m} \sum_{m=1}^M \mathcal{L}(\vec{W}^{(0)}, x_m)$$

2. Итерационный процесс:

Выбираем какой-нибудь объект $x \in X$, например случайным образом или перебираем все элементы X по порядку. Корректируем вектор матриц перехода:

$$\vec{W}^{(t+1)} = \vec{W}^{(t)} - h \cdot \vec{\nabla} \mathcal{L}(\vec{W}^{(t)}, x)$$

где h – темп обучения.

Эмпирический риск оценивается по формуле:

$$Q^{(t+1)}(\vec{W}^{(t+1)}, X) = \lambda \mathcal{L}(\vec{W}^{(t)}, x) + (1 - \lambda) Q^{(t)}(\vec{W}^{(t)}, X)$$

где λ – темп забывания предыстории.

Рассмотрим, откуда взялась эта оценка $Q^{(t+1)}$.

Если $Q^{(t)}$ – среднее арифметическое объектов ε_i , $i = 1, 2, \dots, t$, то

$$Q^{(t)} = \frac{1}{t} \varepsilon_t + \frac{1}{t} \varepsilon_{t-1} + \frac{1}{t} \varepsilon_{t-2} + \dots + \frac{1}{t} \varepsilon_1$$

$$Q^{(t-1)} = \frac{1}{t-1} \varepsilon_{t-1} + \frac{1}{t-1} \varepsilon_{t-2} + \cdots + \frac{1}{t-1} \varepsilon_1$$

$Q^{(t)}$ можно выразить через $Q^{(t-1)}$:

$$Q^{(t)} = \frac{1}{t} \varepsilon_t + \frac{t-1}{t} Q^{(t-1)} = \frac{1}{t} \varepsilon_t + \left(1 - \frac{1}{t}\right) Q^{(t-1)}$$

Если $Q^{(t)}$ – экспоненциальное скользящее среднее с параметром λ , то

$$Q^{(t)} = \lambda \varepsilon_t + \lambda(1-\lambda) \varepsilon_{t-1} + \lambda(1-\lambda)^2 \varepsilon_{t-2} + \cdots + \lambda(1-\lambda)^{t-1} \varepsilon_1$$

$$Q^{(t-1)} = \lambda \varepsilon_{t-1} + \lambda(1-\lambda) \varepsilon_{t-2} + \lambda(1-\lambda)^2 \varepsilon_{t-3} + \cdots + \lambda(1-\lambda)^{t-2} \varepsilon_1$$

Или $Q^{(t)}$ через $Q^{(t-1)}$:

$$Q^{(t)} = \lambda \varepsilon_t + (1-\lambda) Q^{(t-1)}$$

Пусть теперь $\lambda \sim \frac{1}{t}$. Тогда $(1-\lambda)^{t-1} = (1-\frac{1}{t})^{t-1}$. $\lim_{t \rightarrow \infty} (1-\frac{1}{t})^{t-1} = \frac{1}{e}$. По аналогии из радиотехники, где для экспоненциально убывающего сигнала характерное время затухания измеряется по уменьшению сигнала в e раз, тогда характерное количество членов, когда происходит затухание / «забывание» предыдущей истории ряда равно t .

3. Повторяем итерационный процесс, пока эмпирический риск $Q(\vec{W}^{(t)}, X)$ или вектор матриц перехода \vec{W} не сойдутся.

3.1.7. Метод обратного распространения ошибок (BackProp).

Для вычисления градиента функции потерь применяется метод обратного распространения ошибок. Для упрощения формулы индекс $^{(t)}$ будем опускать.

$$\vec{\nabla} \mathcal{L}(\vec{W}, x) = \left[\frac{\partial}{\partial W^0}, \frac{\partial}{\partial W^1}, \dots, \frac{\partial}{\partial W^L} \right]^T \cdot \mathcal{L}(\vec{W}, x)$$

Для рассмотренной ранее квадратичной функции потерь:

$$\vec{\nabla} \mathcal{L}(\vec{W}, x) = \left[\frac{\partial}{\partial W^0}, \frac{\partial}{\partial W^1}, \dots, \frac{\partial}{\partial W^L} \right]^T \cdot \frac{1}{2} \sum_{i=1}^{H_L} (x_i^L((\vec{W}, x) - y_i)$$

То есть выход нейронной сети x^L является функцией от вектора матриц перехода \vec{W} .

Выведем формулу, по которой нейронная сеть вычисляет \vec{x}^L . На первом слое сети находится вектор \vec{x}^0 , который через матрицу перехода W^1 и вектор функций активации $\vec{\sigma}_1$ преобразуется во вход второго слоя x^1 :

$$\vec{x}^1 = \vec{\sigma}_1(W^1 \cdot \vec{x}^0)$$

Аналогично для последующих слоёв:

$$\vec{x}^2 = \vec{\sigma}_2(W^2 \cdot \vec{x}^1) = \vec{\sigma}_2(W^2 \cdot \vec{\sigma}_1(W^1 \cdot \vec{x}^0))$$

$$\vec{x}^L = \vec{\sigma}_L(W^L \cdot \vec{x}^{L-1}) = \vec{\sigma}_L(W^L \cdot \vec{\sigma}_{L-1}(W^{L-1} \cdot \vec{\sigma}_{L-1}(\dots (W^2 \cdot \vec{\sigma}_1(W^1 \cdot \vec{x}^0)) \dots)))$$

Чтобы получить формулы для обратного распространения ошибок необходимо найти $\vec{\nabla} \mathcal{L}$, рассматривая \vec{x}^L как функцию $\vec{x}^L(\vec{W})$.

Задача 3. Доказать рекуррентные формулы для метода обратного распространения ошибок в предположениях:

1. Функция потерь квадратичная $\mathcal{L}(\vec{W}, x) = \frac{1}{2} \sum_{i=1}^{H_L} (x_i^L((\vec{W}, x) - y_i)^2$.
2. Первый нейрон в слое нейронной сети всегда -1, то есть $\forall i : x_0^i = -1$. H_l – количество нейронов в слое l без учёта нейрона -1. Тогда индекс первого нейрона в слое, не всегда равного -1, равен 1, а индекс последнего нейрона равен H_l .
3. Вектор функций активации $\vec{\sigma}^l$ зависит от отступа \vec{M}^l : $\sigma_n^l = \sigma_n^l(M_n^l)$, где

$$M_n = \sum_{m=1}^{H_l} W_{nm}^l x_m^{l-1}. \quad \vec{M}^l = W^l \vec{x}^{l-1}.$$

$$\frac{\partial \mathcal{L}}{\partial x_n^L} = x_n^L - y_n,$$

$$\frac{\partial \mathcal{L}}{\partial W_{nm}^L} = \frac{\partial \mathcal{L}}{\partial x_n^L} \frac{\partial \sigma_n^L}{\partial M_n^L} x_m^{L-1},$$

$$\frac{\partial \mathcal{L}}{\partial x_n^l} = \sum_{m=1}^{H_l} \frac{\partial \mathcal{L}}{\partial x_m^{l+1}} \frac{\partial \sigma_m^{l+1}}{\partial M_m^{l+1}} W_{mn}^{l+1},$$

$$\frac{\partial \mathcal{L}}{\partial W_{nm}^l} = \frac{\partial \mathcal{L}}{\partial x_n^l} \frac{\partial \sigma_n^l}{\partial M_n^l} x_m^{l-1},$$

$$1 \leq l < L.$$

Решение:

Сначала докажем соотношения для $l = L$.

Рассматриваем функцию потерь как сложную функцию $\mathcal{L}(\vec{W}) = \mathcal{L}(\vec{x}^L(\vec{W}))$:

$$\frac{\partial \mathcal{L}}{\partial W_{nm}^L} = \sum_{i=1}^{H_L} \frac{\partial \mathcal{L}}{\partial x_i^L} \frac{\partial x_i^L}{\partial W_{nm}^L}$$

Матрица W^L используется в уравнении: $\vec{x}^L = \vec{\sigma}_L(W^L \vec{x}^{L-1})$. В данном уравнении предполагается, что вектор \vec{x}^L имеет размерность $H_L \times 1$ (является столбцом,

а не строкой) и не содержит -1, так как вес константы x_0^L не вычисляется по предыдущему слою, а определяется нулевым столбцом матрицы W^{L+1} , а вектор $\overrightarrow{x^{L-1}}$ уже имеет длину $1 + H_{L-1}$, и содержит -1. Тогда размерность матрицы W^L равна $H_L \times (1 + H_{L-1})$. Дифференцировать \mathcal{L} по x_0^L не имеет смысла, так как всегда $x_0^L = -1$.

Так как $\mathcal{L}(\overrightarrow{x^L}) = \frac{1}{2} \sum_{n=1}^{H_L} (x_n^L - y_n)^2$, то

$$\frac{\partial \mathcal{L}}{\partial x_i^L} = x_i^L - y_i$$

Так как $\overrightarrow{x^L} = \sigma^L(W^L \overrightarrow{x^{L-1}})$ или $x_i^L = \sigma_i^L \left(\sum_{k=0}^{H_{L-1}} W_{ik}^L x_k^{L-1} \right)$, тогда

$$\frac{\partial x_i^L}{\partial W_{nm}^L} = 0, \text{ если } n \neq i \text{ и}$$

$$\frac{\partial x_i^L}{\partial W_{nn}^L} = \frac{\partial \sigma_n^L}{\partial M_n^L} x_m^{L-1}, \text{ если } n = i.$$

То есть функция σ_i^L зависит от W_{nm}^l , если $n = i$. При дифференцииции суммы $\sum_{k=0}^{H_{L-1}} W_{ik}^L x_k^{L-1}$ остаётся только член с $k = m$, то есть x_m^{L-1} . Стоит отметить, что функция активации как функция от одного аргумента $\sigma_n^L = \sigma_n^L(M_n^L)$ известна при написании программы, аналогично будет известна и её производная $\frac{\partial \sigma_n^L}{\partial M_n^L} \Big|_{M_n^L}$.

Производная вычисляется в точке $M_n^L = \sum_{k=0}^{H_{L-1}} W_{nk}^L x_k^{L-1}$.

При использовании стохастического градиентного спуска отступ $\overrightarrow{M^l}$ будет вычислен при прямом ходе, при вычислении $\overrightarrow{x^l}$ от объекта $x \in X$. Поэтому для ускорения вычислений можно при прямом ходе вычислять и запоминать значения производных $\frac{\partial \sigma_n^l}{\partial M_n^l} \Big|_{M_n^l}$ в точке M_n^l , также на каждом слое нужно запоминать значения $\overrightarrow{x^l}$.

Теперь используя полученные соотношения запишем:

$$\frac{\partial \mathcal{L}}{\partial W_{nm}^L} = \sum_{i=1}^{H_L} \frac{\partial \mathcal{L}}{\partial x_i^L} \frac{\partial x_i^L}{\partial W_{nm}^L} = \frac{\partial \mathcal{L}}{\partial x_n^L} \frac{\partial x_n^L}{\partial W_{nm}^L} = \frac{\partial \mathcal{L}}{\partial x_n^L} \frac{\partial \sigma_n^L}{\partial M_n^L} x_m^{L-1}$$

где $\frac{\partial \mathcal{L}}{\partial x_n^L} = x_n^L - y_n$.

Теперь докажем формулы для $1 \leq l < L$.

$x_n^l = \sigma_n^l \left(\sum_{k=0}^{H_{l-1}} W_{nk}^l x_k^{l-1} \right)$ или $x_n^l = x_n^l(\overrightarrow{W^l}, \overrightarrow{x^{l-1}})$. Тогда

$$\frac{\partial \mathcal{L}}{\partial W_{nm}^l} = \sum_{i=1}^{H_l} \frac{\partial \mathcal{L}}{\partial x_i^l} \frac{\partial x_i^l}{\partial W_{nm}^l}$$

Найдём $\frac{\partial \mathcal{L}}{\partial x_i^l}$.

$$\frac{\partial \mathcal{L}}{\partial x_i^l} = \sum_{j=1}^{H_{l+1}} \frac{\partial \mathcal{L}}{\partial x_j^{l+1}} \frac{\partial x_j^{l+1}}{\partial x_i^l}$$

Производная $\frac{\partial \mathcal{L}}{\partial x_j^{l+1}}$ уже была вычислена на шаге для $l + 1$. Вычислим $\frac{\partial x_j^{l+1}}{\partial x_i^l}$. Так как $x_j^{l+1} = \sigma_j^{l+1} \left(\sum_{k=0}^{H_l} W_{jk}^{l+1} x_k^l \right)$, то

$$\frac{\partial x_j^{l+1}}{\partial x_i^l} = \frac{\partial \sigma_j^{l+1}}{\partial M_j^{l+1}} W_{ji}^{l+1}$$

Итого

$$\frac{\partial \mathcal{L}}{\partial x_i^l} = \sum_{j=1}^{H_{l+1}} \frac{\partial \mathcal{L}}{\partial x_j^{l+1}} \frac{\partial \sigma_j^{l+1}}{\partial M_j^{l+1}} W_{ji}^{l+1}$$

Производная $\frac{\partial x_i^l}{\partial W_{nm}^l}$ вычисляется аналогично, как и для $l = L$:

$$\frac{\partial x_i^l}{\partial W_{nm}^l} = 0, \text{ если } n \neq i \text{ и}$$

$$\frac{\partial x_i^l}{\partial W_{nm}^l} = \frac{\partial \sigma_n^l}{\partial M_n^l} x_m^{l-1}, \text{ если } n = i.$$

Окончательно

$$\frac{\partial \mathcal{L}}{\partial W_{nm}^l} = \frac{\partial \mathcal{L}}{\partial x_n^l} \frac{\partial \sigma_n^l}{\partial M_n^l} x_m^{l-1}$$

3.1.8. Алгоритм применения метода обратного распространения ошибки в стохастическом градиентном спуске.

Рассмотрим подробнее алгоритм применения метода обратного распространения ошибки в стохастическом градиентном спуске.

1. Выбираем какое-то начальное приближение вектора матриц перехода $\overrightarrow{W}^{(0)}$. Вычисляем первое приближение эмпирического риска:

$$Q^{(0)}(\overrightarrow{W}^{(0)}, X) = \frac{1}{m} \sum_{m=1}^M \mathcal{L}(\overrightarrow{W}^{(0)}, x_m)$$

2. Итерационный процесс:

Выбираем какой-нибудь объект $x \in X$, например случайным образом или перебираем все элементы X по порядку.

Прямой ход. Вычисляем отступ $\overrightarrow{M^l} = W^l \cdot \overrightarrow{x^{l-1}}$. Вычисляем и запоминаем значения выходов нейронов $\overrightarrow{x^l} = \sigma^l(\overrightarrow{M^l})$ и производные функций активации $\frac{\partial \sigma^l}{\partial \overrightarrow{M^l}} \Big|_{\overrightarrow{M^l}}$.

Вычисляем для последнего слоя производные от функции потерь:

$$\frac{\partial \mathcal{L}}{\partial x_n^L} = x_n^L - y_n$$

и

$$\frac{\partial \mathcal{L}}{\partial W_{nm}^L} = \frac{\partial \mathcal{L}}{\partial x_n^L} \frac{\partial \sigma_n^L}{\partial M_n^L} x_m^{L-1}$$

Обратный ход для всех слоёв $1 \leq l < L$:

$$\frac{\partial \mathcal{L}}{\partial x_n^l} = \sum_{m=1}^{H_l} \frac{\partial \mathcal{L}}{\partial x_m^{l+1}} \frac{\partial \sigma_m^{l+1}}{\partial M_m^{l+1}} W_{mn}^{l+1}$$

и

$$\frac{\partial \mathcal{L}}{\partial W_{nm}^l} = \frac{\partial \mathcal{L}}{\partial x_n^l} \frac{\partial \sigma_n^l}{\partial M_n^l} x_m^{l-1}$$

В итоге был вычислен градиент функции потерь: $\vec{\nabla} \mathcal{L}(\vec{W}^{(t)}, x)$.

Корректируем вектор матриц перехода:

$$\vec{W}^{(t+1)} = \vec{W}^{(t)} - h \cdot \vec{\nabla} \mathcal{L}(\vec{W}^{(t)}, x)$$

где h – темп обучения.

Эмпирический риск оценивается по формуле:

$$Q^{(t+1)}(\vec{W}^{(t+1)}, X) = \lambda \mathcal{L}(\vec{W}^{(t)}, x) + (1 - \lambda) Q^{(t)}(\vec{W}^{(t)}, X)$$

где λ – темп забывания предыстории.

3. Повторяем итерационный процесс, пока эмпирический риск $Q(\vec{W}^{(t)}, X)$ или вектор матриц перехода \vec{W} не сойдутся.

3.2. Функции активации ReLU и PReLU. Проблема «паралича» сети.

3.2.1. Про функции активации

Функция активации в контексте нейронных сетей определяет выходной сигнал нейрона на основе входного сигнала или набора входных сигналов. Она играет

важную роль в определении нелинейности модели и позволяет нейронной сети обучаться сложным зависимостям между входными данными и выходами, а также влиять на эффективность обучения модели.(грубо говоря определяет будет ли нейрон активирован - даст не нулевой выход)

В искусственных нейронных сетях используются различные типы функций активации, такие как тождественная функция, ступенчатые функции, сигмоидные функции и многие другие.

Важным требованием к функции активации является ее нелинейность, чтобы нейронная сеть могла эффективно справляться с нетривиальными задачами.

Примером широко используемой функции активации является ReLU.

3.2.2. ReLU (Rectified Linear Unit), проблема «паралича» сети

Определение: Функция активации ReLU определяется как: $f(x) = \max(0, x)$, где x — входное значение.

Не смотря на очевидную простоту функции активации ReLU, у неё есть одна существенная проблема: она может вызывать так называемый **«паралич» сети**. Если градиент весов становится слишком большим, то некоторые нейроны могут получить очень большие отрицательные веса, что приведёт к тому, что их выходные значения всегда будут равны нулю независимо от входного сигнала. Эти нейроны становятся неактивными («замороженными») и больше не участвуют в процессе обучения. Если это происходит слишком часто, то это может привести к ухудшению результата.

Для решения этой проблемы была предложена модификация ReLU.

3.2.3. PReLU (Parametric Rectified Linear Unit)

Определение: Функция PReLU является обобщением ReLU и определяется как:

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \leq 0 \end{cases}$$

В отличие от обычной ReLU, где все отрицательные значения обнуляются, PReLU вводит дополнительный параметр α , который управляет наклоном линии для отрицательных значений. Этот параметр α может быть обучен вместе с другими параметрами сети, что позволяет избежать полного отключения нейронов. То есть даже отрицательный входной сигнал будет вносить вклад в обучение модели, что уменьшает проблему паралича сети и улучшает получаемый результат.

3.2.4. Задачи

Задача 1

Представьте, что вы получили графики функций активации ReLU и PReLU (с заданным параметром α) для различных входных значений.

Опишите, как выглядит график функции активации ReLU и как он отличается от графика PReLU. Какова роль параметра α в PReLU?

Решение

График функции ReLU выглядит как угол, который начинается от нуля и идет вверх с углом 45 градусов для положительных значений.

График функции PReLU (Parametric ReLU) также имеет две части, но для отрицательных значений он не обрезается до нуля. Вместо этого, он наклонен под углом, определяемым параметром α .

Роль параметра α в PReLU:

Параметр α определяет наклон линии для отрицательных значений. Если α велико, то выход будет относительно большим даже для небольших отрицательных входов, что позволяет нейрону сохранять некоторую активность. Если α близко к нулю, то функция будет почти горизонтальной для отрицательных значений, что делает нейрон менее активным в этом диапазоне.

Задача 2.1

Рассмотрим простой случай использования функции активации ReLU в одном слое нейронной сети. Пусть входной сигнал равен -4 , а вес этого нейрона равен 10 . Каково будет значение на выходе этого нейрона после применения функции активации ReLU?

Решение:

Выходной сигнал до применения функции активации будет равен произведению входа и веса: $-4 \cdot 10 = -40$. После применения функции активации ReLU получаем: $f(-40) = \max(0, -40) = 0$. Таким образом, этот нейрон станет неактивным ("замороженным"), поскольку его выходной сигнал всегда будет нулевым независимо от изменения входного сигнала.

Задача 2.2

Пусть теперь используется функция активации PReLU с параметром $\alpha = 0.01$. Рассчитайте значение на выходе того же нейрона, что и в предыдущей задаче, но уже с использованием PReLU.

Решение:

После умножения входа и веса получаем тот же результат: $-4 \cdot 10 = -40$. Теперь применяем функцию активации PReLU:

$$f(-40) = \begin{cases} -40 & \text{if } (-40) > 0 \\ 0.01 \cdot (-40) & \text{if } (-40) \leq 0 \end{cases}$$

Так как $-40 \leq 0$, мы используем вторую часть определения функции: $f(-40) = 0.01 \cdot (-40) = -0.4$. В этом случае нейрон не становится полностью замороженным, так как его выходной сигнал остается отличным от нуля даже

при отрицательном входе.

Задача 3

Входной сигнал x распределён равномерно на интервале $[-10, 10]$, а вес w нейрона равен 2, параметр $\alpha = 0.1$.

Какова вероятность, что нейрон заморозится при использовании каждой из функций ReLU и PReLU?

Решение:

Для ReLU:

Нейрон «замораживается», если выходной сигнал после применения функции активации всегда равен нулю. То есть, если $wx \leq 0$.

Поскольку $w = 2$, условие $wx \leq 0$ эквивалентно $x \leq 0$.

Вероятность того, что x окажется меньше или равно нулю, равна вероятности того, что x попадет в интервал $-10, 0$. Это составляет половину всего диапазона распределения, поэтому вероятность равна 0.5.

При использовании PReLU нейрон никогда не «замерзнет», потому что даже при отрицательных значениях x выходной сигнал будет отличен от нуля благодаря параметру α . Таким образом, вероятность того, что нейрон «заморозится», равна 0.

3.3. Drop Out

3.3.1. Теоретические сведения

Drop Out (метод отключения случайных нейронов) - это метод, который представляет собой эффективный подход к борьбе с переобучением в полносвязных нейронных сетях.

Переобучение может происходить, когда нейроны в сети начинают "запоминать" шум и особенности обучающего набора, вместо того чтобы извлекать общие закономерности. Во время обучения нейронной сети нейроны взаимодействуют друг с другом, и иногда происходит так, что один нейрон начинает исправлять ошибки другого. Это может привести к ситуации, когда в одном слое нейронов образуются большие веса с разными знаками, что делает модель менее стабильной и затрудняет ее обобщение. В таких случаях, даже если модель демонстрирует высокую точность на обучающих данных, она может оказаться неэффективной на тестовых данных.

Идея Drop Out заключается в том, что при обучении случайные нейроны отключаются (то есть возвращают всегда нулевое значение) и не участвуют в данном шаге обучения. В таком случае большие значения весов разных знаков не всегда участвуют в обучении одновременно, и модель стабилизируется.

Обучение с Dropout можно интерпретировать как обучение одновременно 2^N моделей (где N — количество нейронов) с разными архитектурами связей.

При обучении выбирается модель, наиболее устойчивая к потере доли нейронов; разные части модели решают одну и ту же задачу вместо того, что бы компенсировать ошибки друг друга.

Недостатком Drop Out является более долгое обучение модели в связи со случайностью процесса обучения.

3.3.2. Реализация

При обучении выбирается параметр p — вероятность отключения. Модель обучается с отключением случайных нейронов. Можно отключать в каждом слое долю p нейронов, вместо независимого отключения, чтобы избежать полного отключения одного слоя.

$$x_i^l = \xi_i^l \sum_{j=1}^{K_{l-1}} w_{ij}^l x_j^{l-1}$$

l — номер текущего слоя, i — номер нейрона в этом слое, K_l — кол-во нейронов в слое l , $\xi \sim Be(1 - p)$ — случайная величина, отвечающая за отключение.

$[\xi_i^l] = 1 - p$, поэтому на этапе применения вводится нормировка:

$$x_i^l = (1 - p) \sum_{j=1}^{K_{l-1}} w_{ij}^l x_j^{l-1}$$

Чаще используется **Inverted Drop Out** для простоты применения:

$$x_i^l = \frac{\xi_i^l}{1 - p} \sum_{j=1}^{K_{l-1}} w_{ij}^l x_j^{l-1}$$

— при обучении;

$$x_i^l = \sum_{j=1}^{K_{l-1}} w_{ij}^l x_j^{l-1}$$

— при применении.

3.3.3. Задачи

Задача 1 Какие по порядку величины должны быть ограничения на значения p , чтобы избежать отключения одного слоя целиком?

Решение: если в сети L слоёв размерности K , то $p \sim (\frac{c}{L})^{1/K}$. Тогда вероятность отключения одного слоя $p^K = \frac{c}{L}$; вероятность функционирования всех слоёв: $\sim (1 - p^K)^L = (1 - \frac{c}{L})^L \sim e^{-c}$.

Если взять $c \sim 0.01$, получим вероятность работы сети $\sim 99\%$.

Если $L = 4, K = 10$, то $p \sim 54\%$.

Задача 2 Сколько шагов обучения нужно провести, чтобы не осталось нейронов, которые были бы выкинуты на каждом шаге (то есть совсем не обучались)?

Решение: если в сети N нейронов, то вероятность одному нейрону совсем не обучиться в течение T итераций равна p^T . Тогда среднее число необученных нейронов равно Np^T . Из условия $Np^T < 1$ получаем $T > -\log_p N = \frac{\ln N}{\ln \frac{1}{p}}$.

Глава 4

Метрические методы классификации и регрессии

Часто используемые ядра $K(r)$

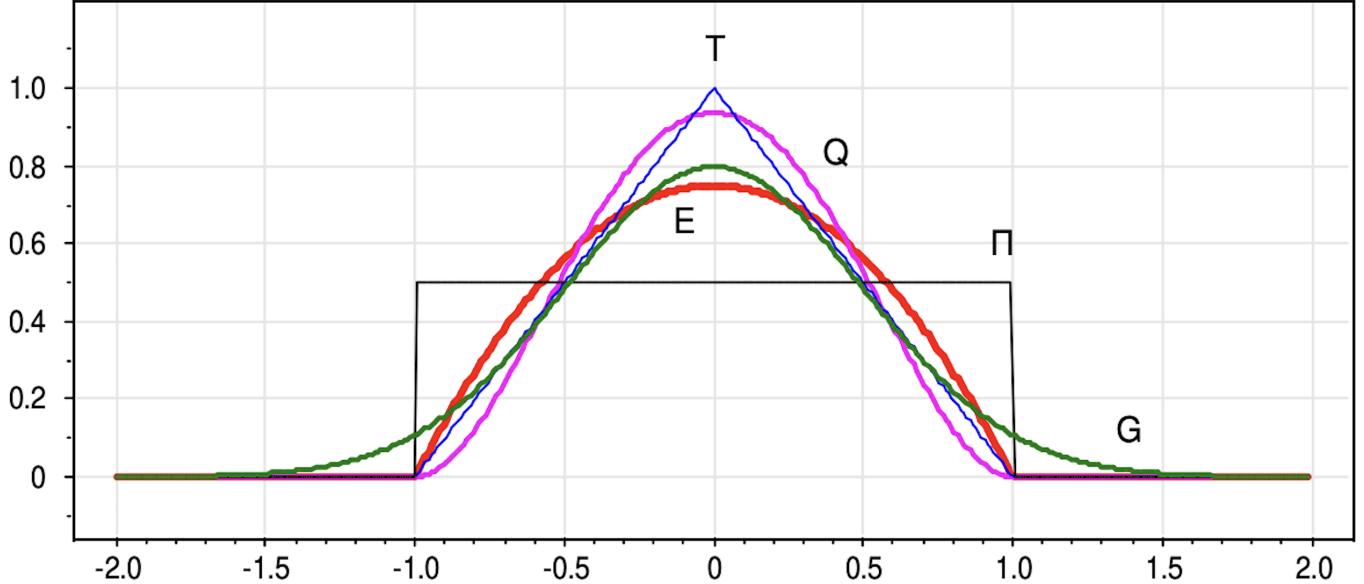


Рис. 4.1. Графики ядер

$\Pi(r) = [|r| \leq 1]$ — прямоугольное

$T(r) = (1 - |r|)[|r| \leq 1]$ — треугольное

$E(r) = (1 - r^2)[|r| \leq 1]$ — квадратичное (Епанечникова)

$Q(r) = (1 - r^2)^2[|r| \leq 1]$ — квартическое

$G(r) = \exp(-2r^2)$ — гауссовское

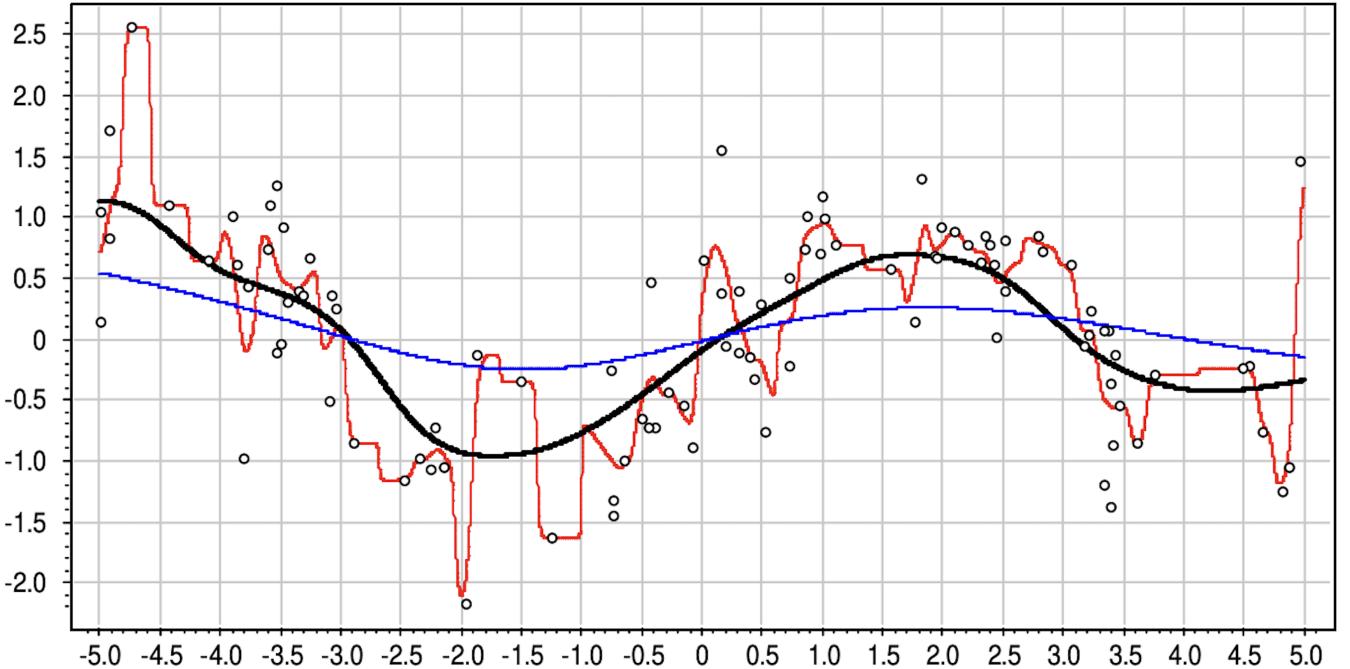
Выбор ядра K и ширины окна h

$h \in \{0.1, 1.0, 3.0\}$, гауссовское ядро $K(r) = \exp(-2r^2)$. Графики с различной шириной окна h :

- Гауссовское ядро \Rightarrow гладкая аппроксимация
- Ширина окна существенно влияет на точность аппроксимации

Выбор ядра K и ширины окна h

$h \in \{0.1, 1.0, 3.0\}$, треугольное ядро $K(r) = (1 - |r|)[|r| \leq 1]$. Графики с разными значениями h при треугольном ядре:



- Треугольное ядро \Rightarrow қусочно-линейная аппроксимация
- Аппроксимация не определена, если в окне нет точек выборки

Выбор ядра K и ширины окна h

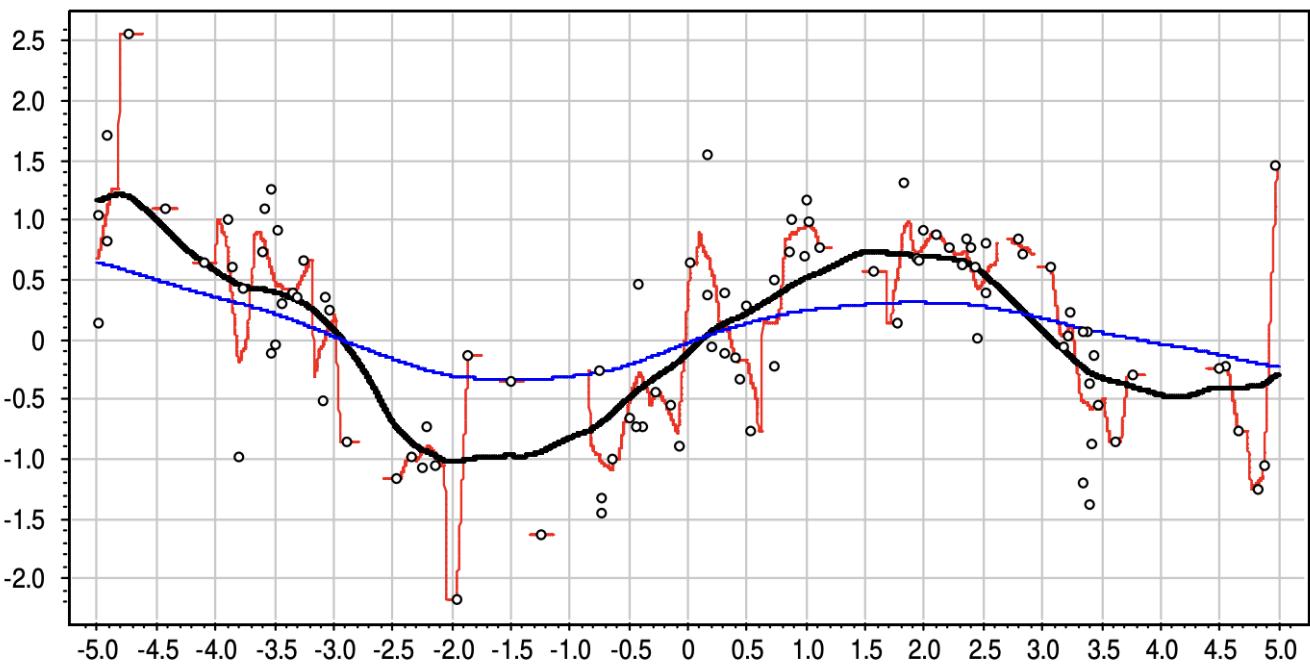
- Ядро $K(r)$
 - существенно влияет на гладкость функции $a_h(x)$,
 - слабо влияет на качество аппроксимации.
- Ширина окна h
 - существенно влияет на качество аппроксимации.
- Переменная ширина окна по k ближайшим соседям:

$$w_i(x) = K \left(\frac{\rho(x, x_i)}{h(x)} \right), \quad h(x) = \rho(x, x^{(k+1)})$$

где $x^{(k)}$ — k -й сосед объекта x .

- Оптимизация ширины окна по скользящему контролю:

$$\text{LOO}(h, X^\ell) = \sum_{i=1}^{\ell} (a_h(x_i; X^\ell \setminus \{x_i\}) - y_i)^2 \rightarrow \min_h$$

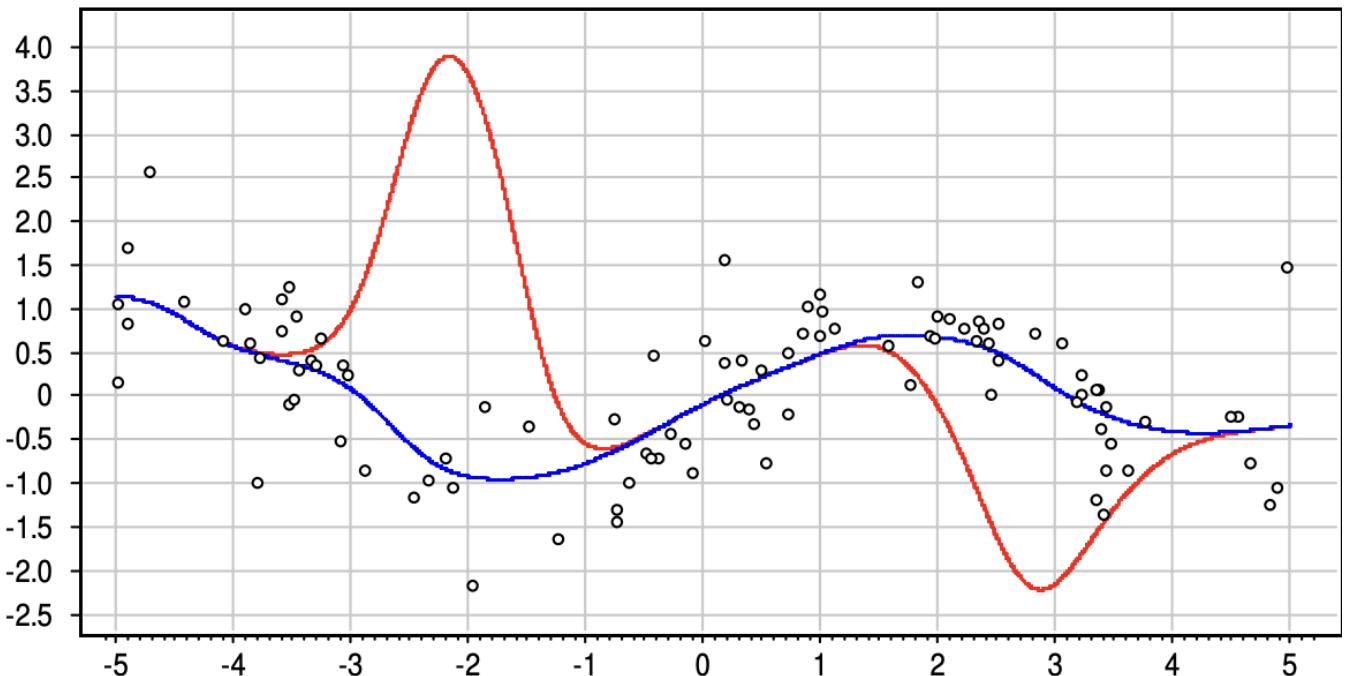


Проблема выбросов (эксперимент на синтетических данных)

$\ell = 100$, $h = 1.0$, гауссовское ядро $K(r) = \exp(-2r^2)$

Две из 100 точек — выбросы с ординатами $y_i = 40$ и -40

Синяя кривая — выбросов нет



Проблема выбросов и локально взвешенное сглаживание

Проблема выбросов: точки с большими случайными ошибками y_i сильно искажают функцию $a_h(x)$

Основная идея:

чем больше величина ошибки $\varepsilon_i = |a_h(x_i; X^\ell \setminus \{x_i\}) - y_i|$,
тем больше прецедент (x_i, y_i) похож на выброс,
тем меньше должен быть его вес $w_i(x)$.

Эвристика:

домножить веса $w_i(x)$ на коэффициенты $\gamma_i = \tilde{K}(\varepsilon_i)$,
где \tilde{K} — ещё одно ядро, вообще говоря, отличное от $K(r)$.

Рекомендация:

квадратичное ядро $\tilde{K}(\varepsilon) = K_Q\left(\frac{\varepsilon}{6 \operatorname{med}\{\varepsilon_i\}}\right)$,
где $\operatorname{med}\{\varepsilon_i\}$ — медиана вариационного ряда ошибок.

Алгоритм LOWESS (Locally Weighted Scatter plot Smoothing)

Вход: X^ℓ — обучающая выборка;

Выход: коэффициенты γ_i , $i = 1, \dots, \ell$;

инициализация: $\gamma_i := 1$, $i = 1, \dots, \ell$;

повторять

- оценки скользящего контроля в каждом объекте:

$$a_i := a_h(x_i; X^\ell \setminus \{x_i\}) = \frac{\sum_{j=1, j \neq i}^{\ell} y_j \gamma_j K\left(\frac{\rho(x_i, x_j)}{h(x_i)}\right)}{\sum_{j=1, j \neq i}^{\ell} \gamma_j K\left(\frac{\rho(x_i, x_j)}{h(x_i)}\right)}, \quad i = 1, \dots, \ell;$$

- $\gamma_i := \tilde{K}(|a_i - y_i|)$, $i = 1, \dots, \ell$;

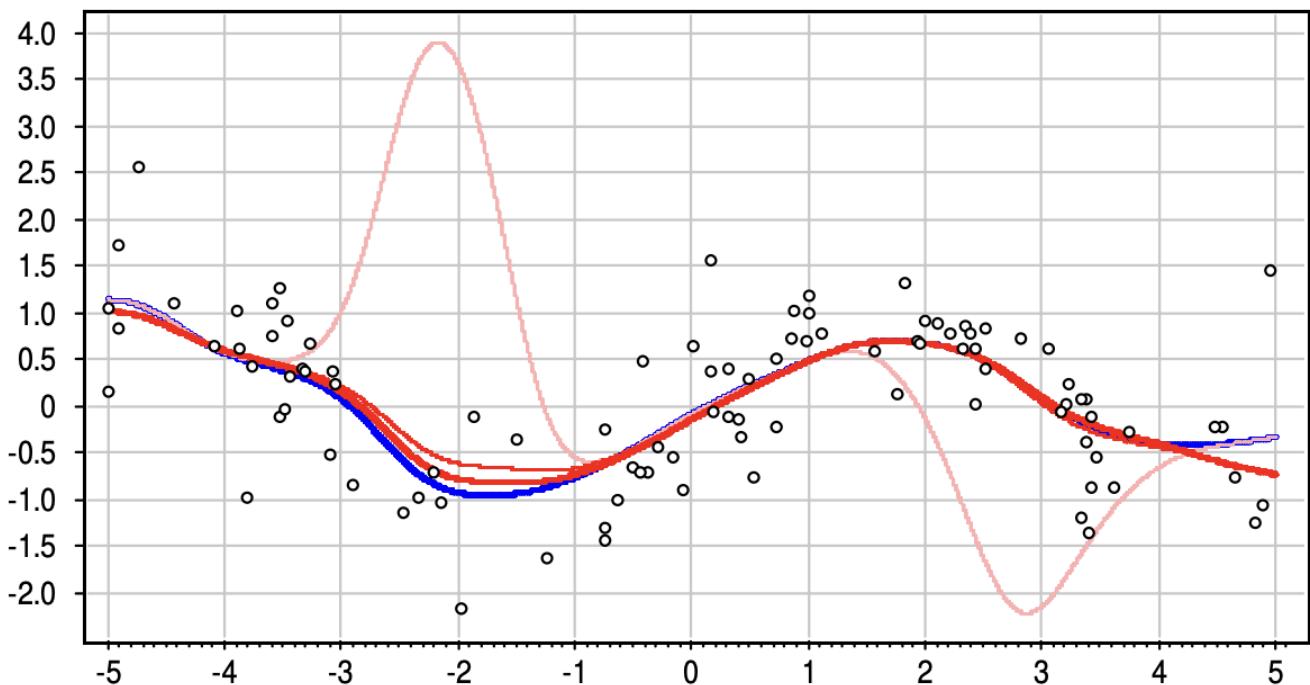
пока коэффициенты γ_i не стабилизируются;

Пример работы LOWESS на синтетических данных

$\ell = 100$, $h = 1.0$, гауссовское ядро $K(r) = \exp(-2r^2)$

Две из 100 точек — выбросы с ординатами $y_i = 40$ и -40

В данном случае LOWESS сходится за несколько итераций:



4.1. Задачи

4.1.1. Задача 1

Объясните, как выбор ядра $K(r)$ влияет на гладкость регрессионной функции $a_h(x)$. Приведите примеры различных ядер и их влияния.

4.1.2. Ответ:

Ядро $K(r)$ определяет форму и вес окрестности точки, используемой для оценки регрессионной функции. Гладкие ядра, такие как гауссовское, дают более плавные оценки, в то время как менее гладкие, такие как прямоугольное, приводят к менее сглаженным функциям. Например:

- Гауссовское ядро $K(r) = \exp(-r^2)$ — даёт плавные, гладкие оценки.
- Треугольное ядро $K(r) = 1 - |r|$ (при $|r| \leq 1$) — более резкое, но все ещё гладкое.
- Прямоугольное ядро $K(r) = 0.5$ (при $|r| \leq 1$) — приводит к кусочно-постоянной функции.

4.1.3. Задача 2

Приведите пример оптимального веса $w_i(x)$ в алгоритме LOWESS, если известно распределение ошибок в данных. Поясните, как это распределение должно влиять на выбор веса, и почему весовое ядро \tilde{K} должно учитывать распределение ошибок.

4.1.4. Ответ:

Если ошибка ε_i распределена согласно некоторому известному распределению, например нормальному, $\varepsilon_i \sim \mathcal{N}(0, \sigma^2)$, то весовой коэффициент должен минимизировать дисперсию предсказания. Весовая функция $\tilde{K}(\varepsilon_i)$ должна убывать, когда ε_i отклоняется от некоторого центрального значения (0 для нормального распределения), чтобы уменьшить влияние выбросов:

$$w_i(x) = \exp\left(-\frac{\varepsilon_i^2}{2\sigma^2}\right)$$

Этот вес сильнее подавляет ошибки, отклоняющиеся от средней, что уменьшает их влияние на итоговую модель. Выбор весового ядра \tilde{K} должен учитывать распределение ошибок, чтобы учесть типичные вариации данных.

4.1.5. Задача 3

На основе метода LOWESS предложите способ оценки доверительного интервала для предсказаний. Выведите формулу для доверительного интервала и объясните, как она может быть использована для оценки надежности модели.

4.1.6. Ответ:

1. Построение модели:

- Применим LOWESS для построения основной модели, получив предсказанные значения \hat{y}_i для каждого x_i .

2. Оценка остаточной дисперсии:

- Вычислим остаточные отклонения $e_i = y_i - \hat{y}_i$.

- Оценим дисперсию ошибок σ^2 как среднеквадратическое отклонение:

$$\sigma^2 = \frac{1}{n - k} \sum_{i=1}^n e_i^2$$

где n — число точек данных, k — число параметров (в случае LOWESS, это скорее степень полинома в локальных регрессиях).

3. Построение доверительного интервала:

- Для каждого предсказанного значения \hat{y}_i построим доверительный интервал, используя стандартное отклонение остаточных ошибок и критические значения из t-распределения:

$$\hat{y}_i \pm t_{\alpha/2, n-k} \cdot \sqrt{\frac{\sigma^2}{n_i}}$$

где $t_{\alpha/2, n-k}$ — квантиль t-распределения с уровнем значимости α , и n_i — эффективное число точек в окрестности x_i (окрестность, которая использовалась для регрессии, может быть выражена размером окна или числом соседей).

4. Использование доверительных интервалов:

- Доверительные интервалы позволяют пользователю оценить, насколько "надёжны" предсказанные значения. Узкие интервалы свидетельствуют о высокой уверенности.

- Визуализация доверительных интервалов на графиках помогает выявлять области, где модель может быть неопределённой или подверженной ошибкам.

4.2. Формула Надарадя-Ватсона.

X -объекты Y -ответы $X = (x_i, y_i)_{i=1}^l$ -обучающая выборка Будем обозначать $x_i = (x_i^1, \dots, x_i^n)$ - вектор признаков объекта x_i .

$a(x) = f(x, \theta)$ - параметрическая модель зависимости θ -вектор параметров модели

Метод наименьших квадратов (МНК):

$$Q(\theta; X^l) = \sum_{i=1}^l (f(x_i, \theta) - y_i)^2 w_i \longrightarrow \min_{\theta},$$

где $w(i, x)$ - некоторый вес, отражающий степень важности i-ого объекта. Вес неотрицателен и не возрастает по i .

Сложно определить, какую стоит взять параметрическую модель.

Идея: Будем считать, что $f(x_i, \theta) = \theta$, а w_i зависит от x - т. е. веса объектов задаются в зависимости от того, на каком объекте будет получен объект. Мы будем обучаться для каждого объекта, на котором хотим получить y .

Тогда:

$$Q(\theta; X^l) = \sum_{i=1}^l (\theta - y_i)^2 w(i, x) \longrightarrow \min_{\theta},$$

Наконец, положим

$$w_i(x) = K \left(\frac{\rho(x, x^{(i)})}{h} \right),$$

где $K(r)$ - невозрастающая функция, определённая на неотрицательных числах, называемая ядром, положительная на отрезке $[0, 1]$, h - ширина окна. То есть, чем больше расстояние до $x^{(i)}$, тем меньше оно влияет на x .

Продифференцируем $Q(\theta; X^l)$ по θ и приравняем получившееся значение 0, и отсюда получим формулу ядерного сглаживания Надаля-Ватсона:

$$\theta(x, X^l) = \frac{\sum_{i=1}^l y_i w_i(x)}{\sum_{i=1}^l w_i(x)} = \frac{\sum_{i=1}^l y_i K\left(\frac{\rho(x, x^{(i)})}{h}\right)}{\sum_{i=1}^l K\left(\frac{\rho(x, x^{(i)})}{h}\right)}$$

Средневзвешенное значение y на тех объектах, которые близки к x .

Мы видим, что такой метод для каждого объекта x рассматривает только объекты обучающей выборки, находящиеся на расстоянии не больше h от x . Причём, чем дальше объект от x , тем меньший вклад он даёт в оценку близости.

Обоснование формулы Надаля-Ватсона:

Теорема: пусть выполнены следующие условия:

1. Выборка X^l простая из совместного распределения $p(x, y)$
2. ядро $K(r)$ ограничено $\int_0^{+\infty} K(r)dr < \infty$, $\lim_{r \rightarrow \infty} rK(r) = 0$. (из этого условия следует, что ядро приводит большие расстояния в 0)
3. зависимость $E(y|x)$ не имеет вертикальных асимптот
4. последовательность h_l (ширина окна) убывает с ростом выборки, но не слишком быстро и не слишком медленно $\lim_{l \rightarrow \infty} h_l = 0$, $\lim_{l \rightarrow \infty} lh_l = 0$

Тогда имеет место сходимость по вероятности: $\theta(x, X^l) \xrightarrow{p} E(y|x)$

(здесь E -мат. ожидание y при условии x)

в любой точке $x \in X$, в которой $E(y|x)$, $p(x)$, $D(y|x)$ -непрерывны, $p(x) > 0$

По теореме даже если распределение $p(x, y)$ и $E(y|x)$ не известны, θ будет стремится к нему с ростом длины обучающей выборки

Замечание. Ширина окна отвечает за пере или недообучение, следовательно сильно влияет на качество аппроксимации.

Для подбора наилучшей модели оптимизируются параметры:

- ширина окна h ;
- ядро K .

Приведём примеры наиболее часто используемых ядер:

- $K_1(r) = (1 - r^2)[r \leq 1]$ - ядро Епанечникова;

- $K_2(r) = (1 - r^2)^2[r \leq 1]$ - квартическое ядро;
- $K_3(r) = (1 - |r|)[r \leq 1]$ - треугольное ядро;
- $K_4(r) = [r \leq 1]$ - прямоугольное ядро;
- $K_5(r) = e^{-2r^2}$ - гауссовское ядро.

Замечание. Существует проблема выбросов, при которой точки с большими случайными y_i сильно искажают итоговую функцию. С этим можно справиться, домножив веса w_i на коэффициенты $\gamma_i = K^*(\epsilon_i)$, где $\epsilon_i = |\theta(x_i, X^l) - y_i|$ подразумевается, что из X^l исключили x_i , K^* -другое ядро

Задача 1. Объяснить, почему не стоит брать финитное ядро на неизвестных данных при малой ширине окна h . Объяснить, почему нельзя просто взять большое h , чтобы решить проблему из предыдущего пункта.

Ответ: При малой ширине окна h и финитном ядре у некоторых точек может не оказаться соседей, попадающих в радиус ядра K . На случай такого события в формулу Надаля-Ватсона в знаменатель добавляют малый член, чтобы не делить на 0.

Если увеличить ширину окна h , то для всех точек с большей вероятностью смогут найтись соседи, но в таком случае точность аппроксимации ухудшится.

Задача 2. как ядро влияет на гладкость функции? Ответ обосновать.

Ответ: Будем считать, что ядро не обращается в 0. Вспомним, что функция, являющаяся результатом деления двух гладких функций, является гладкой. Сумма гладких функций, умноженных на число- также гладкая функция Пусть

ядро K - гладкое, тогда $\sum_{i=1}^l y_i K\left(\frac{\rho(x, x^{(i)})}{h}\right)$ -гладкое, и функция $\frac{\sum_{i=1}^l y_i K\left(\frac{\rho(x, x^{(i)})}{h}\right)}{\sum_{i=1}^l K\left(\frac{\rho(x, x^{(i)})}{h}\right)}$ будет гладкой

Следовательно, выбор ядра сильно влияет на гладкость.

Задача 3. Имеются следующие данные: $x = [1.2, 2.7, 3.1, 4.5, 5.3]$, $y = [3.5, 1.8, 5.2, 2.1, 4.7]$. Необходимо предсказать значение y для $x = 2.5$, используя формулу Надаля-Ватсона с квартическим ядром и $h = 1$.

Решение: Для каждой точки из набора данных вычисляем вес $K\left(\frac{x-x_i}{h}\right)$, где $x = 2.5$, $h = 1$, Суммируем все вычисленные веса и делим каждый вес на эту сумму, чтобы получить нормированные веса.

веса не равны 0 только у точек $x=2.7$ $x=3.1$, их веса равны соответственно 0.9216 и 0.4096.

При нормировке получим веса 0.6923 и 0.3077 Предсказание $\hat{y}(2.5)$ вычисляется как взвешенная сумма значений по уже известной формуле примерный $\hat{y}(2.5) = 2.85$

4.3. Влияние выбора метрики на качество работы kNN

Метод k -ближайших соседей (kNN) является одним из базовых алгоритмов машинного обучения и широко применяется в задачах классификации и регрессии. Этот алгоритм относится к метрическим методам, так как выбор ближайших соседей основывается на измерении расстояний между объектами. Главным фактором, определяющим качество модели, является выбор метрики расстояния, так как она определяет, какие объекты считаются «похожими».

4.3.1. Сущность метода k -ближайших соседей

Алгоритм k -ближайших соседей работает следующим образом:

1. Для нового объекта вычисляется расстояние до всех объектов обучающей выборки.
2. Выбираются k ближайших объектов в соответствии с выбранной метрикой.
3. Класс нового объекта определяется на основе классов выбранных соседей (например, по принципу большинства для задач классификации).
4. В задачах регрессии целевое значение нового объекта вычисляется как среднее (или взвешенное среднее) значений соседей.

Ключевым аспектом метода является расстояние, вычисляемое на основе метрики, которая влияет на работу алгоритма, в том числе на точность и устойчивость модели.

4.3.2. Популярные метрики расстояния

1. **Евклидова метрика (L_2 -норма):** Одна из самых популярных метрик, измеряющая геометрическое расстояние между точками в пространстве. Формула:

$$d(x, y) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$

Особенности:

- Подходит для данных, где все признаки одинаково масштабированы.
- Чувствительна к выбросам, так как квадрат отклонения значительно увеличивает вклад большого расхождения.

2. **Манхэттенская метрика (L_1 -норма):** Измеряет расстояние как сумму абсолютных разностей координат. Формула:

$$d(x, y) = \sum_{i=1}^n |x_i - y_i|$$

Особенности:

- Лучше подходит для разреженных данных.
- Менее чувствительна к выбросам по сравнению с Евклидовой метрикой.

3. Метрика Минковского: Обобщение Евклидовой и Манхэттенской метрик. Формула:

$$d(x, y) = \left(\sum_{i=1}^n |x_i - y_i|^p \right)^{\frac{1}{p}}$$

Особенности:

- При $p = 2$ совпадает с Евклидовой метрикой, при $p = 1$ — с Манхэттенской.
- Позволяет гибко настраивать степень влияния больших отклонений через параметр p .

4. Косинусное расстояние: Измеряет угол между векторами, игнорируя их длину. Формула:

$$d(x, y) = 1 - \frac{\langle x, y \rangle}{\|x\| \cdot \|y\|}$$

Особенности:

- Часто применяется для текстовых данных (например, векторов TF-IDF).
- Устойчиво к изменениям масштаба векторов.

5. Метрика Чебышёва: Учитывает максимальное различие по одной из координат. Формула:

$$d(x, y) = \max_i |x_i - y_i|$$

Особенности:

- Удобна для задач, где критически важно учитывать наибольшее расхождение.
- Формирует кубические границы ближайших соседей.

4.3.3. Влияние выбора метрики на качество работы модели

Выбор метрики может значительно изменить поведение алгоритма k -ближайших соседей. Основные факторы влияния:

1. Геометрия данных: Разные метрики формируют разные границы классов. Например, Евклидова метрика создает округлые границы, а Манхэттенская — прямоугольные. В задачах с нелинейными разделяющими гиперплоскостями может потребоваться комбинированный подход или нестандартные метрики.

2. **Масштаб признаков:** Метрики, такие как Евклидова, чувствительны к различиям в масштабе признаков. Например, если один признак имеет диапазон $[0, 1]$, а другой — $[0, 1000]$, последний будет доминировать. Для решения этой проблемы применяется нормализация или стандартизация данных.
3. **Шум и выбросы:** Метрики по-разному реагируют на шум. Евклидова метрика чувствительна к выбросам, так как квадратичное расстояние значительно увеличивается для больших расхождений. Метрики, такие как Манхэттенская или косинусное расстояние, менее чувствительны к шуму.
4. **Высокая размерность:** В задачах с большим количеством признаков (проблема «проклятия размерности») расстояния между всеми объектами становятся примерно одинаковыми. Это снижает различимость ближайших соседей, что делает выбор метрики критически важным.
5. **Специфика задачи:** Например, для задач обработки текстов косинусная метрика часто оказывается лучше, так как она учитывает только направление векторов, игнорируя их длину. Для задач с пространственными данными чаще применяются Евклидова или Манхэттенская метрика.

4.3.4. Примеры задач

1. **Теоретическая задача:** Рассмотрите два набора данных из двух классов, представленных точками на плоскости. Постройте границы разделения классов при использовании:
 - Евклидовой метрики,
 - Манхэттенской метрики.
 Объясните, как выбор метрики влияет на форму границ.
2. **Практическая задача:** Используя набор данных Iris, обучите модель k -ближайших соседей с Евклидовой, Манхэттенской и косинусной метриками. Сравните метрики качества (точность, F1-меру) и сделайте вывод о том, какая метрика работает лучше и почему.
3. **Исследовательская задача:** Для синтетических данных с перекрывающимися классами разработайте алгоритм выбора оптимальной метрики с использованием кросс-валидации. Постройте графики зависимости точности от параметра k для разных метрик.

4.3.5. Заключение

Выбор метрики расстояния является ключевым фактором, определяющим качество работы метода k -ближайших соседей. Оптимальная метрика зависит от природы данных, задачи и требований к точности и устойчивости

модели. Для улучшения результатов рекомендуется проводить предварительный анализ данных, нормализацию признаков и тестирование нескольких метрик с использованием кросс-валидации.

4.4. Более быстрые оптимизации kNN.

Описанный в пункте 4.3 тип KNN называется Brute-Force, поскольку в нём используется метод полного перебора для поиска ближайших соседей, что делает его простым в реализации, но слишком медленным при работе с большим объемом данных. Для решения данной проблемы в реализации scikit-learn предусмотрены более продвинутые методы, основанные на бинарных деревьях, что позволяет получить значительный прирост в производительности.

4.4.1. BallTree

BallTree — это древовидная структура, в основе которой лежит разбиение исходного пространства данных на вложенные гиперсфераe, что позволяет более эффективно отсекать большие области пространства, в которых отсутствуют ближайшие соседи для точек. В большинстве случаев такой алгоритм подходит для данных с произвольной метрикой расстояния.

Построение BallTree состоит из следующих шагов:

1. из множества точек выбирается одна случайным образом и для неё находится самая дальняя точка;
2. далее все точки разбиваются на гиперсфераe (узлы) по ближайшему расположению к двум точкам из шага 1;
3. затем данный процесс повторяется рекурсивно для каждой гиперсфераe, пока в ней не останется определённое количество точек или не будет достигнута заданная глубина дерева.

При поиске k-ближайших соседей для новой точки, алгоритм сравнивает расстояние от заданной точки до центра каждого дочернего узла и оставляет лишь те, в которых данное расстояние меньше радиуса узлов.

Для оценки качества полученной древовидной структуры и её дальнейшей оптимизации очень полезной будет информация о пересекающихся гиперсфераe (узлах) A и B в метрике M, расстояние между которыми можно определить следующим образом:

$$d_M(A, B) = \max(0, d_M(c_A, c_B) - r_A - r_B)$$

где c_A и c_B — центры сфер, а r_A и r_B — их радиусы.

В данном случае оптимизация BallTree с учётом пересекающихся гиперсфераe (узлов) может включать в себя следующие подходы:

балансировка дерева: поскольку пересечение гиперсфер может указывать на несбалансированность дерева, перебалансировка его узлов позволяет улучшить эффективность поиска, минимизируя количество посещаемых узлов;

выбор оптимального размера листа: в случае сильного пересечения гиперсфер, увеличение размера листа может уменьшить количество узлов, что также ускорит поиск;

слияние узлов: полезно при значительном пересечении, что также уменьшает их общее количество и используется в предыдущих пунктах;

выбор порядка обхода: информация о структуре пересечения может сделать более эффективным порядок посещаемых узлов, начиная проверку с наиболее вероятных кандидатов.

Стоит отметить, что описанные методы оптимизации плюс-минус похожим образом могут быть применимы и для алгоритма ниже.

4.4.2. KD-Tree

KD-Tree (k-dimensional tree) — ещё одна древовидная структура, отдалённо напоминающая BallTree, однако в данном случае используются гиперплоскости для разбиения точек вместо гиперсфер, что позволяет также эффективно оставлять лишь те области пространства данных, в которых могут присутствовать ближайшие соседи. Обычно KD-Tree больше подходит для данных с евклидовой или манхэттенской метрикой расстояния.

Построение KD-Tree состоит из следующих шагов:

1) из множества точек выбирается одна из координат (обычно поочередно для каждого уровня дерева, но можно и случайным образом) и по ней вычисляется медиана;

2) далее все точки разбиваются на два узла (подмножества) по отношению к медиане: на те, у которых значение выбранной координаты меньше либо равно медиане, и на те, у которых больше;

3) данный процесс повторяется рекурсивно для каждого узла, пока в нём не останется определённое количество точек или не будет достигнута заданная глубина дерева.

При поиске ближайших соседей для новой точки, алгоритм сравнивает значение заданной точки с медианой в каждом узле, выбирая таким образом ближайшее подпространство, которое будет листом с ближайшими соседями. Возвращаясь обратно к корню, алгоритм будет сравнивать точки в текущем узле с ближайшими соседями и обновлять их значения в случае нахождения более близких к заданной точке.

4.4.3. Примеры задач

Задача 1. У вас есть два набора данных:

Набор данных А: 10 миллионов точек в 5-мерном пространстве с использованием Евклидовой метрики.

Набор данных В: 1 миллион точек в 10-мерном пространстве с кастомной метрикой расстояния. Вопросы:

Какую структуру данных (KD-Tree или BallTree) вы выберете для каждого набора данных? Объясните ваш выбор. Как влияет размерность пространства на эффективность KD-Tree?

Ответ:

Для набора данных А оптимально использовать KD-Tree, так как он хорошо работает с Евклидовой метрикой, особенно в пространствах с небольшой размерностью. Для набора данных В лучше подходит BallTree, так как он поддерживает произвольные метрики и может быть эффективнее в высоких размерностях.

С увеличением размерности эффективность KD-Tree значительно снижается из-за проклятия размерности: гиперплоскости разбиения теряют свою полезность, и увеличивается количество узлов, которые нужно проверять.

Задача 2. Имеется 2D-набор данных с точками:

(2,3), (5,4), (9,6), (4,7), (8,1), (7,2)

1. Постройте KD-Tree для этих точек.

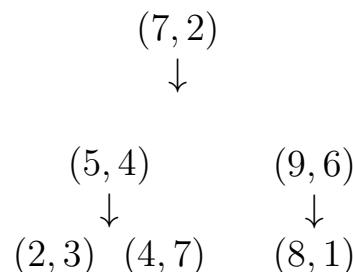
2. Какие оптимизации можно применить к этому дереву, если оно оказалось несбалансированным?

Ответ:

1. Построение дерева:

- Выбор корня: координата x , медиана по $x \rightarrow$ точка (7, 2).
- Левое поддерево ($x < 7$): медиана по $y \rightarrow$ точка (5, 4).
 - Левый потомок: (2, 3).
 - Правый потомок: (4, 7).
- Правое поддерево ($x > 7$): медиана по $y \rightarrow$ точка (9, 6).
 - Левый потомок: (8, 1).

Итоговое дерево:



2. Оптимизации:

- Если дерево несбалансировано, можно:
 - Перестроить дерево, используя медиану всех точек на каждом уровне.
 - Увеличить размер листов (количество точек в узле) для уменьшения глубины дерева.

Задача 3: Поиск ближайшего соседа с использованием BallTree

Дано: гиперсфера в 3D-пространстве, центры $(1, 1, 1)$, $(4, 4, 4)$, радиусы $r_1 = 2$, $r_2 = 3$. Найдите ближайшего соседа для точки $(2, 2, 2)$, используя алгоритм BallTree.

Решение:

1. Вычисляем расстояние от точки до центров гиперсфер:

$$d((2, 2, 2), (1, 1, 1)) = \sqrt{(2-1)^2 + (2-1)^2 + (2-1)^2} = \sqrt{3} \approx 1.73$$

$$d((2, 2, 2), (4, 4, 4)) = \sqrt{(2-4)^2 + (2-4)^2 + (2-4)^2} = \sqrt{12} \approx 3.46$$

2. Сравниваем с радиусами:

- Точка $(2, 2, 2)$ находится внутри гиперсферы с центром $(1, 1, 1)$, так как $1.73 < 2$.
- Точка не входит в гиперсферу с центром $(4, 4, 4)$, так как $3.46 > 3$.

Ответ: Ближайший сосед — точка в гиперсфере с центром $(1, 1, 1)$.

4.5. Расстояние Махalanобиса в метрических методах классификации и регрессии

Метрики являются фундаментальными инструментами в задачах классификации и регрессии, основанных на сходстве или расстоянии между объектами. Они определяют, как мы измеряем "близость" между парами объектов в пространстве признаков.

Расстояние Махalanобиса является обобщением Евклидова расстояния, которое учитывает не только разности значений признаков, но и статистические свойства данных, такие как дисперсии и корреляции между признаками. Это делает его особенно полезным в случае многомерных данных с неоднородной структурой.

4.5.1. Определение:

Расстояние Махalanобиса между вектором признаков объекта \mathbf{x} и средним вектором μ определяется как:

$$d(\mathbf{x}, \mu) = \sqrt{(\mathbf{x} - \mu)^\top \mathbf{S}^{-1} (\mathbf{x} - \mu)}$$

где:

- \mathbf{x} — вектор признаков объекта,
- μ — вектор средних значений признаков (например, центра кластера или класса),
- \mathbf{S} — ковариационная матрица признаков,
- \mathbf{S}^{-1} — обратная матрица к \mathbf{S} ,
- $(\mathbf{x} - \mu)^\top$ — транспонированный вектор разностей.

4.5.2. Преимущества расстояния Махalanобиса:

1. Расстояние Махalanобиса не зависит от масштаба признаков, что устраняет необходимость в предварительной нормализации или стандартизации данных.
2. Метрика учитывает корреляции между признаками, что позволяет более точно измерять расстояние в пространстве, где признаки взаимосвязаны.
3. Она адаптируется к распределению данных, отражая их внутреннюю структуру и дисперсию, что может улучшить результаты в задачах классификации и кластеризации.

4.5.3. Недостатки расстояния Махalanобиса:

1. Необходимо вычислять ковариационную матрицу \mathbf{S} и обратную матрицу \mathbf{S}^{-1} , что является дорогостоящей операцией, особенно при высокой размерности данных.
2. Для точной оценки ковариационной матрицы требуется достаточно большой объём данных, число наблюдений должно значительно превышать число признаков.
3. Если ковариационная матрица не обратима (например, из-за линейной зависимости между признаками или недостаточного числа наблюдений), то невозможно напрямую вычислить \mathbf{S}^{-1} .
4. Выбросы могут сильно искажать оценку ковариационной матрицы и средних значений, что приводит к неправильному вычислению расстояний.

5. Сложности в интерпретации: Поскольку расстояние учитывает сложные связи в данных, интерпретация значений может быть менее интуитивной по сравнению с Евклидовым расстоянием.

4.5.4. Практические аспекты использования:

1. Регуляризация ковариационной матрицы: Для решения проблем с вырожденностью часто применяется добавление небольшого значения к диагональным элементам ковариационной матрицы (тирихоновская регуляризация).
2. Понижение размерности: Можно использовать методы понижения размерности (например, анализ главных компонентов) для уменьшения вычислительной нагрузки и устранения мультиколлинеарности.
3. Отбор признаков: Исключение избыточных или некоррелированных признаков может улучшить оценку ковариационной матрицы и качество метрики.

4.5.5. Применение в методах классификации и регрессии

1. Классификация по ближайшим соседям (k-NN): Использование расстояния Махalanобиса вместо Евклидова может улучшить качество классификации в случаях, когда признаки имеют различные масштабы или коррелированы.
2. Дискриминантный анализ: В линейном дискриминантном анализе (LDA) расстояние между классами измеряется с помощью расстояния Махalanобиса, что позволяет учитывать разброс и ориентацию классов в пространстве признаков.
3. Обнаружение выбросов: Объекты с большим расстоянием Махalanобиса от центра распределения могут рассматриваться как выбросы.

4.5.6. Задачи

Задача 1: Использование расстояния Махalanобиса в классификации k-NN

Вы применяете метод классификации k-ближайших соседей (k-NN) для разделения объектов на два класса: Класс А и Класс В. У каждого класса известны средние векторы признаков и ковариационные матрицы:

Класс А:

$$\mu_A = \begin{bmatrix} 2 \\ 2 \end{bmatrix}, \quad S_A = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

Класс В:

$$\mu_B = \begin{bmatrix} 5 \\ 5 \end{bmatrix}, \quad \mathbf{S}_B = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Новый объект имеет признаки $\mathbf{x} = \begin{bmatrix} 4 \\ 4 \end{bmatrix}$.

Вычислите расстояния Махalanобиса от объекта \mathbf{x} до каждого класса и определите, к какому классу следует отнести данный объект.

Решение:

1. Расстояние до Класса А:

- Вычисляем разность:

$$\mathbf{x} - \mu_A = \begin{bmatrix} 4 - 2 \\ 4 - 2 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$$

- Находим обратную ковариационную матрицу \mathbf{S}_A^{-1} :

$$\mathbf{S}_A^{-1} = \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{bmatrix}$$

- Вычисляем расстояние:

$$d_A = \sqrt{(\mathbf{x} - \mu_A)^\top \mathbf{S}_A^{-1} (\mathbf{x} - \mu_A)} = \sqrt{\begin{bmatrix} 2 & 2 \end{bmatrix} \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{bmatrix} \begin{bmatrix} 2 \\ 2 \end{bmatrix}} = \sqrt{(2)(1) + (2)(1)} = \sqrt{4} = 2$$

2. Расстояние до Класса В:

- Вычисляем разность:

$$\mathbf{x} - \mu_B = \begin{bmatrix} 4 - 5 \\ 4 - 5 \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \end{bmatrix}$$

- Обратная ковариационная матрица \mathbf{S}_B^{-1} уже известна, так как \mathbf{S}_B — единичная матрица:

$$\mathbf{S}_B^{-1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

- Вычисляем расстояние:

$$d_B = \sqrt{(\mathbf{x} - \mu_B)^\top \mathbf{S}_B^{-1} (\mathbf{x} - \mu_B)} = \sqrt{(-1)^2 + (-1)^2} = \sqrt{1 + 1} = \sqrt{2} \approx 1,414$$

3. В итоге:

Поскольку $d_B < d_A$, объект \mathbf{x} принадлежит Классу В.

Задача 2: Обнаружение аномалий с помощью расстояния Махalanобиса

В задаче обнаружения мошеннических транзакций используются два признака: сумма покупки (X_1) и количество товаров (X_2). Для нормальных транзакций известно:

- Средний вектор:

$$\mu = \begin{bmatrix} 100 \\ 10 \end{bmatrix}$$

- Ковариационная матрица:

$$\mathbf{S} = \begin{bmatrix} 400 & 0 \\ 0 & 4 \end{bmatrix}$$

Новая транзакция имеет значения $\mathbf{x} = \begin{bmatrix} 160 \\ 14 \end{bmatrix}$.

Вычислите расстояние Махalanобиса и определите, является ли эта транзакция аномальной, используя пороговое значение $d_{\text{порог}} = 3$.

Решение:

1. Вычисляем разность:

$$\mathbf{x} - \mu = \begin{bmatrix} 160 - 100 \\ 14 - 10 \end{bmatrix} = \begin{bmatrix} 60 \\ 4 \end{bmatrix}$$

2. Находим обратную ковариационную матрицу \mathbf{S}^{-1} :

$$\mathbf{S}^{-1} = \begin{bmatrix} \frac{1}{400} & 0 \\ 0 & \frac{1}{4} \end{bmatrix}$$

3. Вычисляем расстояние:

$$d = \sqrt{(\mathbf{x} - \mu)^\top \mathbf{S}^{-1} (\mathbf{x} - \mu)} = \sqrt{(60)^2 \times \frac{1}{400} + (4)^2 \times \frac{1}{4}} = \sqrt{\frac{3600}{400} + \frac{16}{4}} = \sqrt{9 + 4} = \sqrt{13} \approx$$

4. Поскольку $d = 3,606 > d_{\text{порог}} = 3$, транзакция считается аномальной.

Задача 3: Выбор метрики расстояния при наличии коррелированных признаков

Вы работаете с набором данных, содержащим два признака: X_1 и X_2 . При анализе данных вы обнаружили, что признаки X_1 и X_2 имеют сильную положительную корреляцию.

Вы планируете использовать метод k-ближайших соседей (k-NN) для классификации новых объектов. Возникает вопрос: какую метрику расстояния следует использовать — Евклидово расстояние или расстояние Махalanобиса? Объясните свой выбор.

Решение:

Предполагаемые рассуждения:

- Поскольку признаки X_1 и X_2 сильно коррелированы, это означает, что они содержат избыточную информацию. Изменения в одном признаке сопровождаются изменениями в другом.

- При использовании Евклидова расстояния каждый признак рассматривается независимо, без учета корреляции. Это может привести к тому, что влияние коррелированных признаков будет преувеличено, и объекты будут казаться более далекими друг от друга вдоль направления корреляции.

- Расстояние Махalanобиса учитывает ковариацию между признаками. За счет включения ковариационной матрицы оно корректирует влияние коррелированных признаков, "сжимая" пространство вдоль направления корреляции.

Вывод:

В данном случае целесообразнее использовать расстояние Махalanобиса, так как оно учитывает корреляцию между признаками и обеспечивает более точное измерение сходства между объектами. Это улучшит качество классификации методом k-NN, так как позволит корректно определять ближайших соседей основываясь на истинной структуре данных.

4.6. Профиль компактности и оценка обобщающей способности

Пусть стоит задача улучшения алгоритма 1NN. Этого можно добиться, оставив из выборки только нужные элементы, то есть произвести выбор эталонных элементов. Для такого выбора необходимо выполнить минимизацию CCV.

4.6.1. CCV

Полный скользящий контроль (complete cross-validation, CCV):

$$\text{CCV}(X^L) = \frac{1}{C_L^\ell} \sum_{X^\ell \sqcup X^k} \frac{1}{k} \sum_{x_i \in X^k} [a(x_i; X^\ell) \neq y_i]$$

- частота ошибок алгоритма на контрольной выборке X^k , усреднённая по всем C_L^ℓ разбиениям выборки $X^L = X^\ell \sqcup X^k$ на обучающую подвыборку X^ℓ и контрольную X^k .

Для дальнейших рассуждений введем понятие профиля компактности.

4.6.2. Профиль компактности

Профиль компактности выборки X^L - это функция доли объектов x_i , у которых m -й сосед $x_i^{(m)}$ лежит в другом классе:

$$\Pi(m) = \frac{1}{L} \sum_{i=1}^L [y_i \neq y_i^{(m)}]; \quad m = 1, \dots, L-1,$$

$x_i^{(m)}$ – m -й сосед объекта x_i среди X^L ;
 $y_i^{(m)}$ – ответ на m -м соседе объекта x_i .

4.6.3. CCV для метода 1NN

С учетом введенного $\Pi(m)$ справедлива следующая теорема о точном выражении CCV для метода 1NN:

$$\text{CCV}(X^L) = \sum_{m=1}^k \Pi(m) \frac{C_{L-1-m}^{\ell-1}}{C_{L-1}^\ell}.$$

Нетрудно заметить, что CCV при длине выборки равной 1 совпадает с LOO (leave-one-out).

4.6.4. Пример профилей компактности

Проанализируем графики.

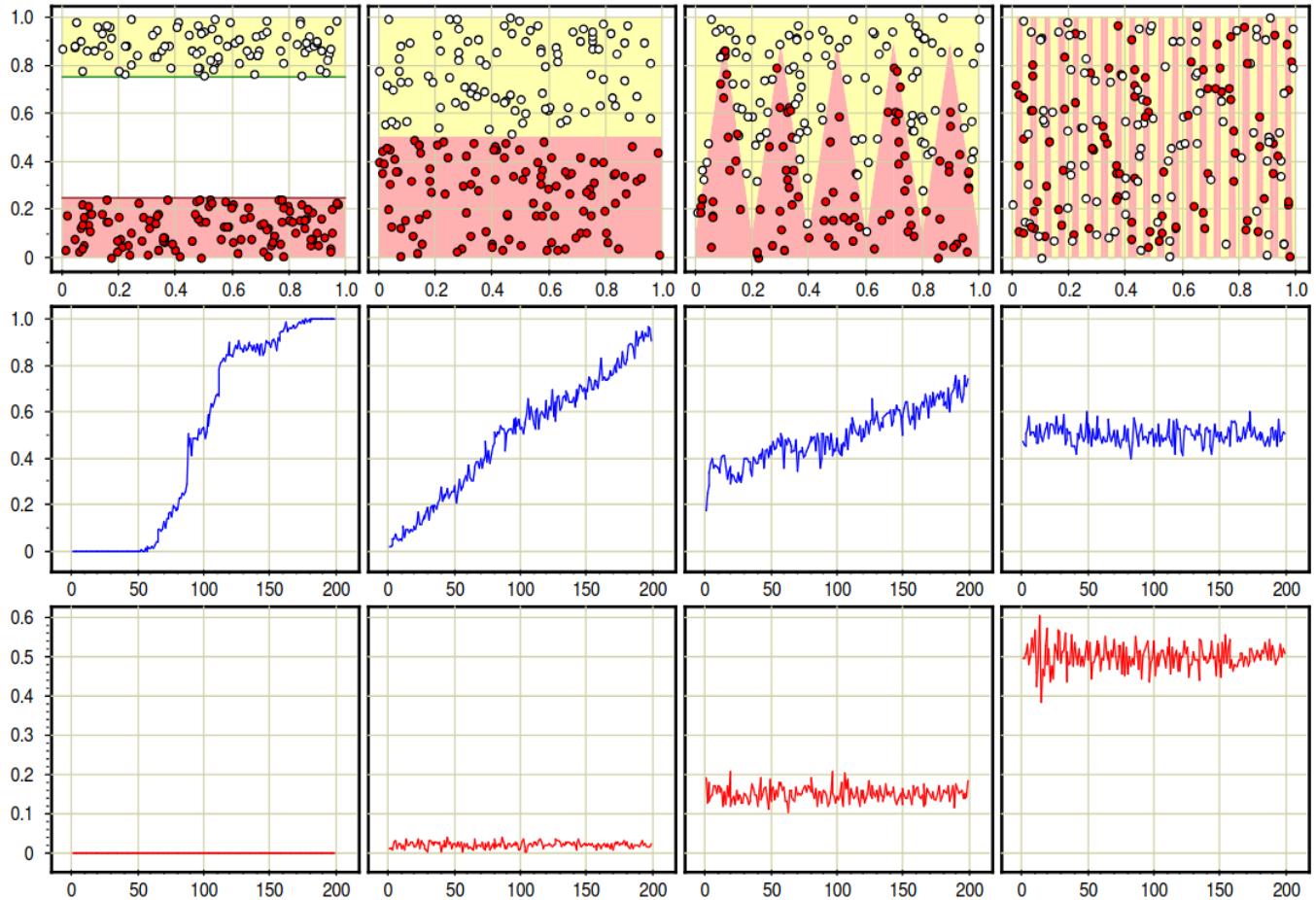


Рис. 4.2. Верхний ряд - два класса, средний ряд - профиль компактности $\Pi(m)$, нижний ряд - зависимость ССВ от длины контроля.

Первый набор: классы разнесены далеко друг от друга. Из профиля компактности видно, что у всех объектов 50 ближайших соседей лежат в своем классе. Значит, у этой задачи будет почти 100% качество при использовании kNN алгоритма.

Второй набор: классы расположены так, что между ними проходит граница. Профиль компактности линейно растет при увеличении числа соседей, причем для малого количества соседей имеем ненулевой $\Pi(m)$.

Третий набор: классы проникают друг в друга. Начальный участок профиля компактности выше нуля, значит, у алгоритма kNN будет больше ошибок.

Четвертый набор: классы равномерно распределены. Профиль компактности находится на уровне 0.5, и у kNN нет возможности адекватно решить данную задачу: имеем случайное гадание.

Замечание: из нижнего ряда имеем, что CCV почти не зависит от длины контроля, то есть нет причины выделять в контрольную выборку более одного объекта и достаточно использовать CCV(1).

4.6.5. Задачи

Задача 1: доказать точное выражение CCV для метода 1NN.

Решение:

$$\begin{aligned}
 \text{CCV} &= \frac{1}{C_L^\ell} \sum_{X^\ell \sqcup X^k} \frac{1}{k} \sum_{i=1}^L [x_i \in X^k] [a(x_i; X^\ell) \neq y_i] = \\
 &= \sum_{X^\ell \sqcup X^k} \sum_{i=1}^L \sum_{m=1}^k \frac{[y_i^{(m)} \neq y_i]}{k C_L^\ell} [x_i^{(m)} \in X^\ell] [x_i, x_i^{(1)}, \dots, x_i^{(m-1)} \in X^k] = \\
 &= \sum_{m=1}^k \sum_{i=1}^L \frac{[y_i^{(m)} \neq y_i]}{k C_L^\ell} \sum_{X^\ell \sqcup X^k} [x_i^{(m)} \in X^\ell] [x_i, x_i^{(1)}, \dots, x_i^{(m-1)} \in X^k] = \\
 &= \sum_{m=1}^k \sum_{i=1}^L \frac{[y_i^{(m)} \neq y_i]}{k C_L^\ell} C_{L-1-m}^{\ell-1} = \underbrace{\sum_{m=1}^k \frac{1}{\sum_{i=1}^L [y_i^{(m)} \neq y_i]} \frac{C_{L-1-m}^{\ell-1}}{C_{L-1}^\ell}}_{\Pi(m)}.
 \end{aligned}$$

Задача 2: получить выражения для CCV при малой длине контроля.

Решение:

$$\text{CCV}(1) = \Pi(1) = \text{LOO}$$

$$\text{CCV}(2) = \Pi(1) \frac{\ell}{\ell+1} + \Pi(2) \frac{1}{\ell+1}$$

$$\text{CCV}(3) = \Pi(1) \frac{\ell}{\ell+2} + \Pi(2) \frac{2\ell}{(\ell+1)(\ell+2)} + \Pi(3) \frac{2}{(\ell+1)(\ell+2)}$$

При достаточно гладком распределении $\Pi(m)$ CCV слабо зависит от длины контроля.

Задача 3: получить скорость спада множителя при $\Pi(m)$ в выражении для CCV.
Решение:

$$R(m) = \frac{C_{L-1-m}^{\ell-1}}{C_{L-1}^{\ell}};$$

$$\frac{R(m+1)}{R(m)} = 1 - \frac{\ell-1}{L-1-m} < \frac{k}{L-1}.$$

То есть $R(m)$ стремится к нулю быстрее геометрической прогрессии.

4.6.6. Минимизация CCV

Из рассмотренных графиков и приведенных задач получили, что CCV тем меньше, чем чаще близкие объекты лежат в одном классе. Значит, минимизируя CCV, можно добиться лучшего результата работы алгоритма k ближайших соседей. CCV почти не зависит от длины контроля для kNN, поэтому достаточно использовать CCV(1) равный LOO.

Приближенные методы поиска ближайших соседей (ANN)

Иногда методы точного поиска ближайших соседей могут быть достаточно времязатратными и вычислительно сложными. Но зачастую реальные задачи не требуют вычислять именно самых близких соседей, и достаточно найти всего несколько наиболее близких. Для этой цели нам подойдут приближенные методы поиска, рассмотренные ниже.

4.6.7. Random projection trees - Annoy

Ряд алгоритмов приближенного поиска основан на деревьях, которые часто применяются для поиска соседей. Одним из наиболее известных и зарекомендовавших себя методов из данного семейства является алгоритм **Annoy**. Опишем принцип его работы.

Пусть у нас есть обучающая выборка. Наша цель — построить структуру данных, которая позволит нам находить ближайшие точки к любой точке запроса. Для начала выберем из нее два объекта случайным образом. Между ними симметрично проводится разделяющая гиперплоскость. Далее в каждом из полученных полупространств снова выбирается два случайных элемента из набора данных, и уже между ними проводятся разделяющие гиперплоскости.

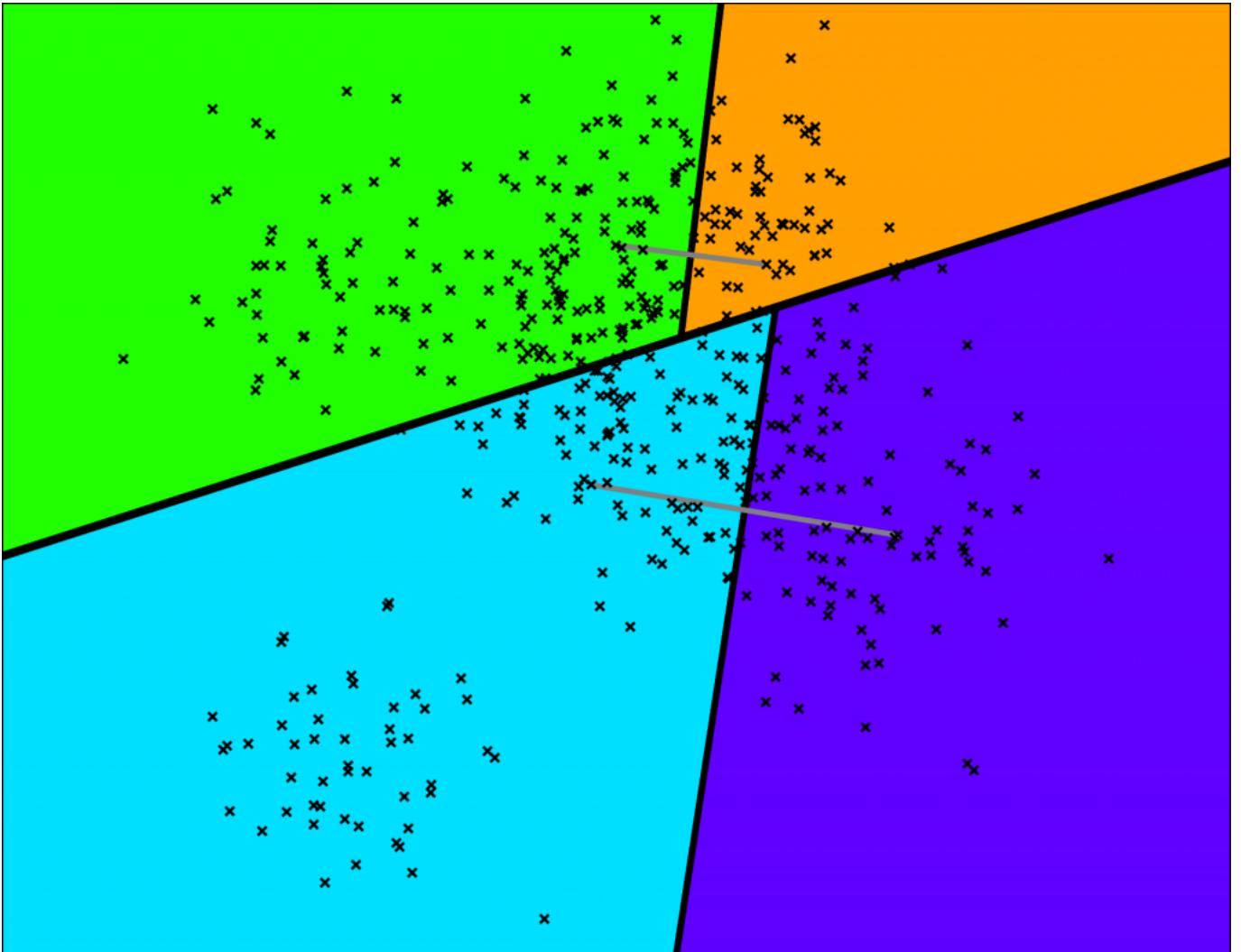


Рис. 4.3. Разделение пространства гиперплоскостями в алгоритме Annoy.

Данная процедура продолжается итеративно, пока в каждой области останется не более K объектов. Здесь K является подбираемым гиперпараметром. На самом деле, данная процедура чем-то похожа на то, как работают k-d-деревья.

В результате работы вышеописанного алгоритма мы получим бинарное дерево (рис. 3) (глубина порядка $O(\log N)$), спускаясь по которому, найдем область с целевым объектом и некоторым количеством близких к нему элементов. Давайте попробуем найти точку, обозначенную красным крестом на рис. 4.

То, как будет выглядеть путь вниз по бинарному дереву в данном случае, показано на рис. 5.

Таким образом, мы получаем 7 ближайших соседей. Уже хороший результат, но достаточно ли нам этого?

Задача 1

Мы проделали весь алгоритм, описанный выше, и получили для некоторой точки 7 ближайших соседей. На самом деле, нам этого недостаточно, попробуйте

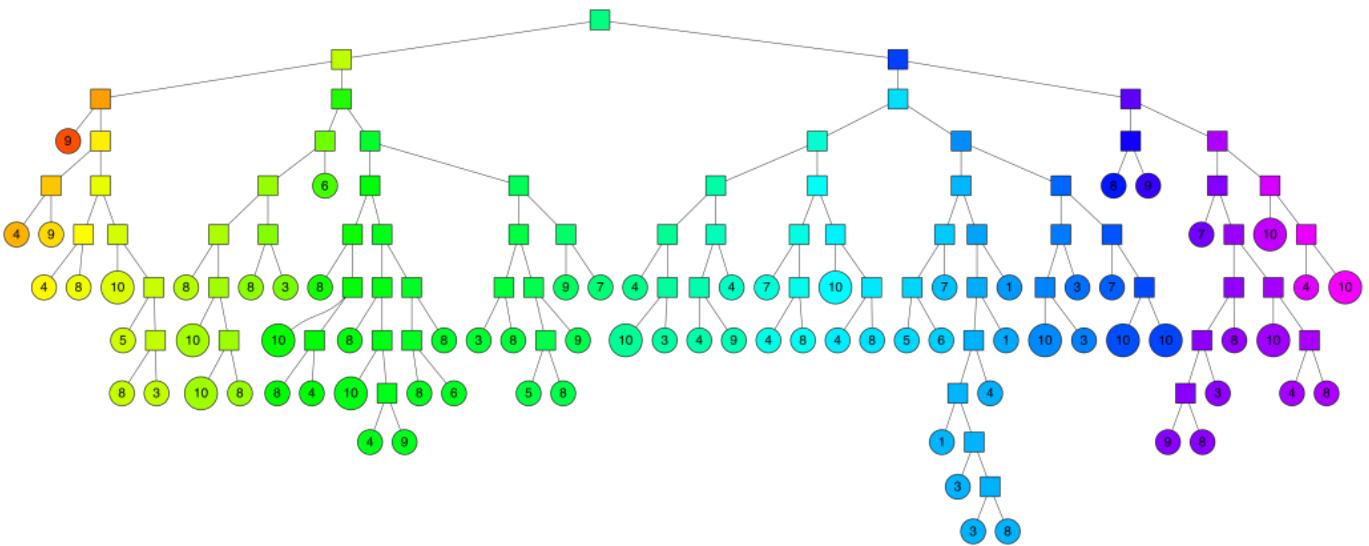


Рис. 4.4. В результате работы алгоритма получаем бинарное дерево.

подумать, почему.

Решение. Во-первых, может произойти ситуация, что нам нужно не 7 ближайших соседей, а больше. Например, методы приближенного поиска соседей используются для подбора рекомендаций фильмов, и в такой задаче нам необходимо найти около 10-15 наиболее похожих картин для пользователя. Во-вторых, некоторые из ближайших соседей на самом деле могут не попасть в итоговую область пространства и остаться за его пределами.

Необходимо увеличить точность алгоритма при помощи составления леса из таких деревьев. Мы можем выполнить поиск по всем деревьям одновременно, и взять объединение соответствующих целевому объекту областей. Именно так работает алгоритм Annoy.

Задача 2

Давайте подумаем, почему же методы приближенного поиска (ANN - approximate nearest neighbor) дают выигрыш по времени и являются менее ресурсозатратными по сравнению с точными методами?

Решение. Если объем анализируемых данных становится слишком большим, что нередко встречается в реальной жизни, то точные методы поиска ближайших соседей будут просматривать всю выборку данных, что займет огромное количество времени. А алгоритмы ANN не просматривают все объекты из датасета, следовательно, являются более быстрыми и эффективными. ANN — это алгоритм, который находит точку данных в наборе данных, которая очень близка к заданной точке запроса, но не обязательно является абсолютно ближайшей. Алгоритм точного поиска выполняет исчерпывающий поиск по всем данным, чтобы найти идеальное совпадение, тогда как алгоритм ANN остановится на совпадении, которое достаточно близко.

Задача 3

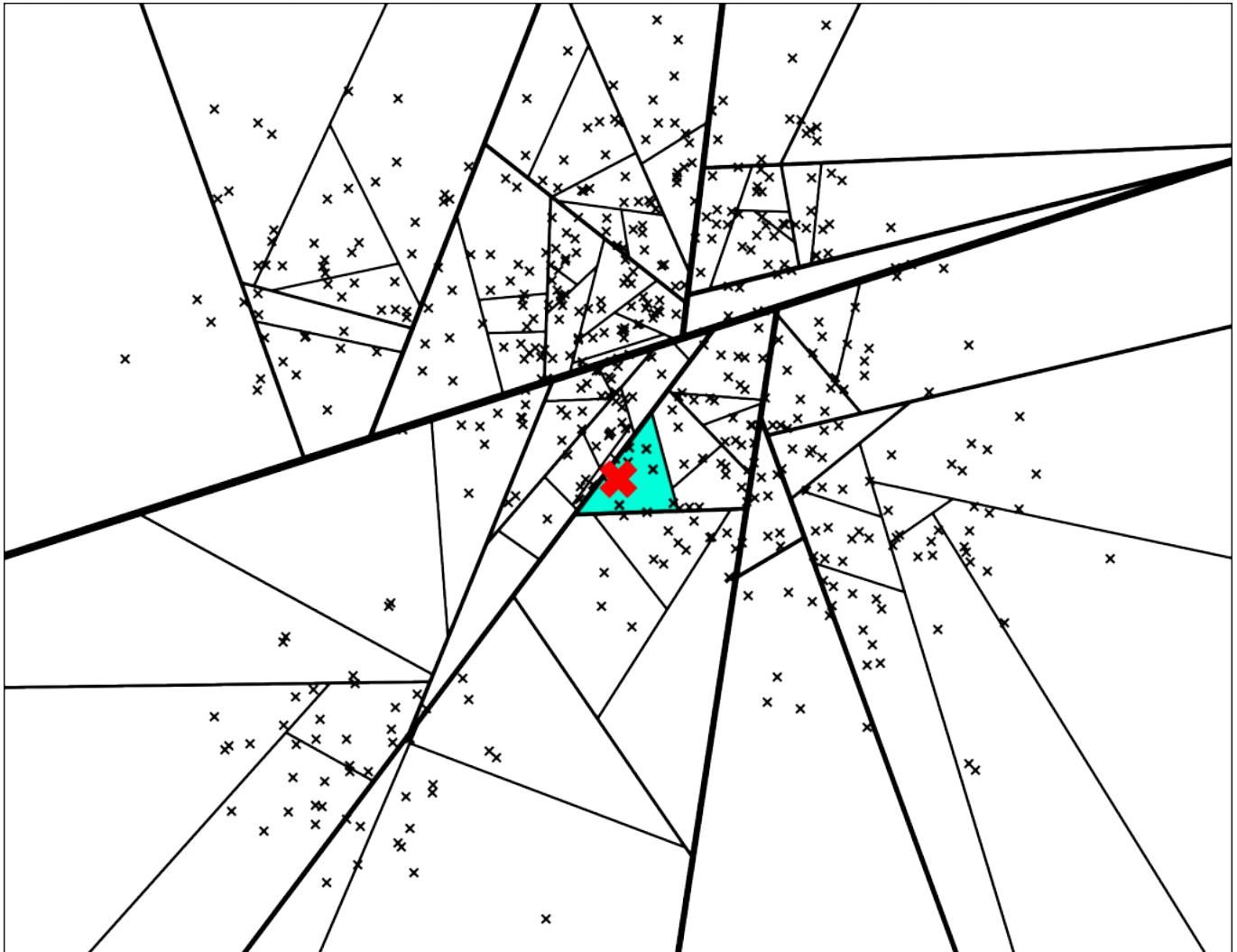


Рис. 4.5. Поиск для точки, обозначенной красным крестом.

Оцените время построения модели Annoy для базы данных из 500 объектов.

Решение. Время поиска ближайших соседей для одного запроса в модели Annoy — $O(\log N)$, где N — количество объектов в базе данных. А время построения модели Annoy для базы данных из 500 объектов составляет $O(N \cdot \log N)$. Для $N = 500$ будет :

$$T = O(500 \cdot \log 500)$$

$$\log 500 \approx 2.7$$

$$T = O(500 \cdot 2.7) = O(1350)$$

То есть время, необходимое для построения модели Annoy для 500 объектов, будет порядка 1350 операций.

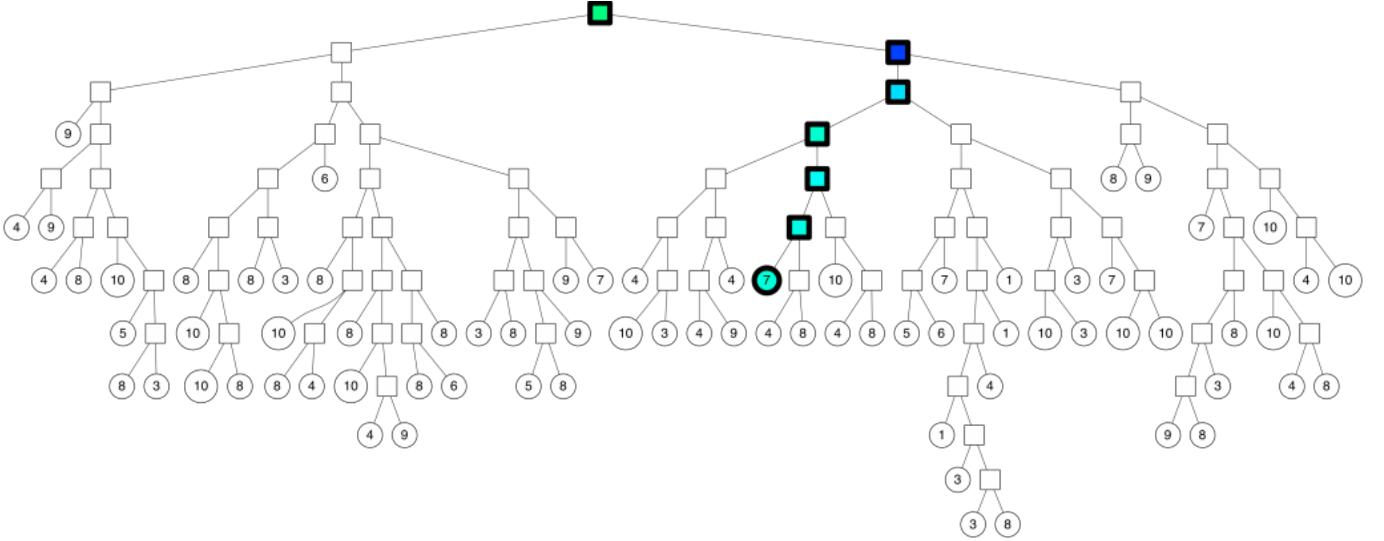


Рис. 4.6. Поиск точки.

4.7. Отбор эталонных объектов

Обучающие объекты делятся на три категории: эталоны (типичные представители классов), неинформативные (окружены объектами того же класса) и шумовые выбросы (расположены среди объектов чужого класса). Исключение шумовых и неинформативных объектов позволяет улучшить классификацию, сократить объём данных и ускорить поиск ближайших эталонов.

Алгоритм STOLP реализует эту идею, основываясь на весовой функции $w(i, u)$. Он строит метрический алгоритм $a(u; \Omega)$, где $\Omega \subset X_\ell$ — множество эталонов.

Отступ объекта x_i относительно алгоритма $a(x_i; \Omega)$, обозначенный как $M(x_i, \Omega)$, используется для классификации объектов:

- $M(x_i, \Omega) < 0$ — объект окружён чужими классами и считается выбросом;
- $M(x_i, \Omega) > 0$ — объект окружён своими классами и является либо эталоном, либо неинформативным.

4.8. Компактность и профиль компактности

Определение 1. Профиль компактности относительно множества эталонов $\Omega \subseteq X^L$ определяется как:

$$\Pi(m, \Omega) = \frac{1}{L} \sum_{i=1}^L \left[y_i^{(m|\Omega)} \neq y_i \right],$$

где $x_i^{(m|\Omega)}$ — m -й сосед объекта x_i из множества Ω , а y_i — истинная метка класса объекта x_i .

Теорема 1. Компактность множества эталонов Ω вычисляется как:

$$CCV(\Omega) = \frac{1}{L} \sum_{i=1}^L \sum_{m=1}^k \left[y_i^{(m|\Omega)} \neq y_i \right] \cdot \frac{C_{L-1}^{L-1-m}}{C_L^{L-1}},$$

где $T(x_i, \Omega)$ — вклад объекта x_i в CCV .

Жадный отбор эталонов по критерию $CCV(\Omega) \rightarrow \min$

Жадная стратегия удаления не-эталонов:

1. Инициализация: $\Omega := X^L$.
 2. Повторять:
 - (a) Найти $x \in \Omega$, при котором $CCV(\Omega \setminus \{x\}) \rightarrow \min$.
 - (b) Удалить x : $\Omega := \Omega \setminus \{x\}$.
 - (c) Обновить $T(x_i, \Omega)$ для всех x_i , где $x \in kNN(x_i)$.
- пока CCV уменьшается или практически не увеличивается.

Жадная стратегия добавления эталонов:

1. Инициализация: $\Omega := \{\text{по одному объекту от каждого класса}\}$.
 2. Повторять:
 - (a) Найти $x \in X^L \setminus \Omega$, при котором $CCV(\Omega \cup \{x\}) \rightarrow \min$.
 - (b) Добавить x : $\Omega := \Omega \cup \{x\}$.
 - (c) Обновить $T(x_i, \Omega)$ для всех x_i , где $x \in kNN(x_i)$.
- пока CCV уменьшается.

Алгоритм: Отбор эталонных объектов STOLP

Вход:

- X_ℓ — обучающая выборка;
- δ — порог фильтрации выбросов;
- ℓ_0 — допустимая доля ошибок.

Выход:

- Множество опорных объектов $\Omega \subset X_\ell$.

Алгоритм:

1. Для всех $x_i \in X_\ell$ проверить, является ли x_i выбросом:

если $M(x_i, X_\ell) < \delta$, то $X_{\ell-1} := X_\ell \setminus \{x_i\}$; $\ell := \ell - 1$;

2. Инициализация: взять по одному эталону от каждого класса:

$$\Omega := \arg \max_{x_i \in X_\ell, y \in Y} M(x_i, X_\ell);$$

3. Пока $\Omega \neq X_\ell$:

- Выделить множество объектов, на которых алгоритм $a(u; \Omega)$ ошибается:

$$E := \{x_i \in X_\ell \setminus \Omega : M(x_i, \Omega) < 0\};$$

- Если $|E| < \ell_0$, то выход.
- Присоединить к Ω объект с наименьшим отступом:

$$x_i := \arg \min_{x \in E} M(x, \Omega); \quad \Omega := \Omega \cup \{x_i\};$$

Результаты работы алгоритма

Алгоритм делит обучающие объекты на три категории:

- Шумовые выбросы, которые удаляются.
- Эталонные объекты, которые формируют подмножество Ω .
- Неинформативные объекты, которые также удаляются.

Если гипотеза компактности верна, то большая часть обучающих объектов окажется неинформативной и будет отброшена, обеспечивая сжатие данных.

Оценка эффективности алгоритма STOLP

Алгоритм STOLP имеет относительно низкую эффективность: добавление каждого эталона требует перебора объектов $X_\ell \setminus \Omega$ и вычисления отступов относительно Ω , что приводит к сложности $O(|\Omega|^2 \ell)$. Для ускорения можно добавлять несколько эталонов одновременно, выбирая их на большом расстоянии друг от друга, чтобы минимизировать влияние на отступы.

На этапе отсея выбросов допустимо вычислить отступы один раз и отбросить объекты с $M(x_i, \Omega) < \delta$. Эффективная реализация включает процедуру обновления отступов $M_i = M(x_i, \Omega)$, что позволяет гибко управлять вычислениями.

Задачи

Задача 1 Докажите, что при использовании алгоритма STOLP отступ объекта $M(x_i, \Omega)$ не зависит от порядка перебора эталонных объектов в Ω .

Решение. Рассмотрим отступ объекта x_i относительно множества эталонов Ω :

$$M(x_i, \Omega) = \sum_{x_j \in \Omega} y_j w(\rho(x_i, x_j)).$$

Порядок перебора объектов $x_j \in \Omega$ не влияет на результат вычисления, так как операция суммирования коммутативна:

$$\sum_{x_j \in \Omega} y_j w(\rho(x_i, x_j)) = \sum_{x_j \in \text{перестановке } \Omega} y_j w(\rho(x_i, x_j)).$$

Таким образом, $M(x_i, \Omega)$ остается неизменным независимо от порядка объектов в Ω . Это доказывает инвариантность отступа относительно перестановки эталонов.

Задача 2 В выборке X_ℓ используется алгоритм STOLP для классификации объектов по двум классам. Выбросы имеют отступы $M(x_i, \Omega) < \delta$, где $\delta = 0$. Рассмотрим набор из трёх объектов:

- $x_1 : M(x_1, \Omega) = -0.5$,
- $x_2 : M(x_2, \Omega) = 0.8$,
- $x_3 : M(x_3, \Omega) = -0.2$.

Определите, какие объекты будут исключены из обучающей выборки X_ℓ на первом этапе алгоритма STOLP.

Решение. На этапе отсея выбросов STOLP удаляет объекты, для которых отступ $M(x_i, \Omega) < \delta$. Учитывая $\delta = 0$, удаляются объекты с отрицательными отступами.

Проверяем отступы:

- $M(x_1, \Omega) = -0.5 < 0$: x_1 будет удалён,
- $M(x_2, \Omega) = 0.8 > 0$: x_2 останется в выборке,
- $M(x_3, \Omega) = -0.2 < 0$: x_3 будет удалён.

Ответ: из обучающей выборки будут удалены объекты x_1 и x_3 .

Задача 3 Докажите, что алгоритм STOLP допускает использование любых метрических функций $\rho(x, x')$, если они удовлетворяют свойствам метрики.

Решение. Метрическая функция $\rho(x, x')$ используется для вычисления отступов $M(x_i, \Omega)$ и определяется через весовую функцию $w(\rho)$. Для корректной работы алгоритма требуется, чтобы $\rho(x, x')$ удовлетворяла следующим свойствам метрики:

1. **Неотрицательность:** $\rho(x, x') \geq 0$,
2. **Равенство нулю только при совпадении точек:** $\rho(x, x') = 0 \iff x = x'$,
3. **Симметричность:** $\rho(x, x') = \rho(x', x)$,
4. **Неравенство треугольника:** $\rho(x, z) \leq \rho(x, y) + \rho(y, z)$.

Эти свойства гарантируют, что отступы $M(x_i, \Omega)$ корректно отражают относительное положение объекта x_i относительно эталонов.

Кроме того, функция $w(\rho)$ должна быть убывающей, чтобы больший вклад в отступ вносили ближайшие эталоны, что не зависит от конкретной метрики, а лишь от её свойств.

Вывод: любой выбор функции $\rho(x, x')$, удовлетворяющей свойствам метрики, допустим в алгоритме STOLP.

4.9. Метод окна Парзена.

Напомним идею метрического классификатора. Будем обозначать $x = (x^1, \dots, x^n)$ - вектор признаков объекта x , $x_i = (x_i^1, \dots, x_i^n)$ - вектор признаков объекта x_i . Пусть на пространстве признаков задана метрика ρ . Для произвольного объекта x отранжируем объекты обучающей выборки x_1, x_2, \dots, x_l :

$$\rho(x, x^{(1)}) \leq \rho(x, x^{(2)}) \leq \dots \leq \rho(x, x^{(l)}).$$

Таким образом, $x^{(i)}$ - i -й ближайший сосед объекта x среди обучающей выборки, обозначим через $y^{(i)}$ ответ на нём.

Метрический классификатор предлагает следующую модель зависимости:

$$a(x; X^l) = \arg \max_{y \in Y} \sum_{i=1}^l [y^{(i)} = y] w(i, x),$$

где $w(i, x)$ - некоторый вес, отражающий степень близости к объекту x его i -го соседа. Вес неотрицателен и не возрастает по i .

Кроме того, введём обозначение:

$$\Gamma_y(x) = \sum_{i=1}^l [y^{(i)} = y] w(i, x) - \text{оценка близости объекта } x \text{ к классу } y.$$

Напомним также, что метод k ближайших соседей заключается в выборе в качестве весовой функции $w(i, x) = [i \leq k]$. Среди недостатков этого метода выделим следующие:

- в силу того, что функция близости дискретнозначная (принимает не более k значений), часто попадаем в ситуацию неоднозначности классификации, когда $\Gamma_y(x) = \Gamma_z(x)$, $y \neq z$;
- метод не учитывает значение расстояний от объекта до ближайших соседей. Естественным кажется использование меньшего веса для далёкого объекта, даже если он попадает в k ближайших.

Для борьбы с этими недостатками модифицируем веса следующим образом:

$$w(i, x) = [i \leq k] w_i, \text{ где } w_i \text{ зависит только от номера соседа.}$$

Тем самым, получим метод k взвешенных ближайших соседей. В качестве w_i можно брать, например, линейно убывающие веса $w_i = \frac{k+1-i}{k}$ или экспоненциально убывающие веса $w_i = q^i$, $0 < q < 1$.

Однако линейно убывающие веса всё ещё допускают неоднозначность классификации. Кроме того, метод по-прежнему не учитывает расстояния между объектами.

Наконец, положим

$$w(i, x) = K \left(\frac{\rho(x, x^{(i)})}{h} \right),$$

где $K(r)$ - невозрастающая функция, определённая на неотрицательных числах, называемая ядром, положительная на отрезке $[0, 1]$ и равная нулю вне его, h - ширина окна.

Таким образом, получим метод окна Парзена фиксированной ширины:

$$a(x; X^l, h, K) = \arg \max_{y \in Y} \sum_{i=1}^l [y^{(i)} = y] K \left(\frac{\rho(x, x^{(i)})}{h} \right).$$

Мы видим, что такой метод для каждого объекта x рассматривает только объекты обучающей выборки, находящиеся на расстоянии не больше h от x . Причём, чем дальше объект от x , тем меньший вклад он даёт в оценку близости.

Замечание. Вообще говоря, в качестве ядра можно брать функции, которые принимают ненулевые значения вне отрезка $[0, 1]$, однако тогда они подбираются быстро убывающими (см. далее гауссовское ядро).

Заметим, что если расстояние от объекта x до всех объектов обучающей выборки больше h , то построенный метод не может классифицировать x , так как в таком случае $\sum_{i=1}^l [y^{(i)} = y] K \left(\frac{\rho(x, x^{(i)})}{h} \right) \equiv 0$. Это соображение наталкивает на использование метода окна Парзена переменной ширины:

$$a(x; X^l, h, K) = \arg \max_{y \in Y} \sum_{i=1}^l [y^{(i)} = y] K \left(\frac{\rho(x, x^{(i)})}{\rho(x, x^{(k+1)})} \right).$$

Замечание. В знаменателе в ядре стоит расстояние до $k+1$ -го соседа, так как на практике зачастую берутся ядра, которые в точке 1 равны 0. Таким образом, метод окна Парзена переменной ширины учитывает именно k ближайших соседей объекта.

Для подбора наилучшей модели оптимизируются параметры:

- ширина окна h или количество соседей k ;
- ядро K .

Выше мы считали, что ядро определено на неотрицательной полуоси. Эквивалентно, можно считать, что ядро определено на всей действительной оси, является чётной функцией, невозрастающей на отрезке $[0, 1]$ (и, как правило, равна 0 вне отрезка $[-1, 1]$). В таком случае расстояние между объектами можно понимать как ориентированное. Однако, нетрудно видеть, что ориентация ни на что не влияет. Приведём примеры наиболее часто используемых ядер:

- $K_1(r) = \frac{3}{4}(1 - r^2)[r \leq 1]$ - ядро Епанечникова;

- $K_2(r) = \frac{15}{16}(1 - r^2)^2[r \leq 1]$ - квартическое ядро;
- $K_3(r) = (1 - |r|)[r \leq 1]$ - треугольное ядро;
- $K_4(r) = \frac{1}{2}[r \leq 1]$ - прямоугольное ядро;
- $K_5(r) = \frac{1}{\sqrt{2\pi}}e^{-r^2/2}$ - гауссовское ядро.

Числовые коэффициенты выбираются из соображений нормировки: $\int_{-\infty}^{+\infty} K_i(r)dr = 1$.

Задача 1. Докажите, что метод окна Парзена фиксированной ширины можно переписать следующим образом:

$$a(x; X^l, h, K) = \arg \max_{y \in Y} \sum_{i=1}^l [y_i = y] K\left(\frac{\rho(x, x_i)}{h}\right),$$

т.е. объекты обучающей выборки не обязательно ранжировать по расстоянию до объекта.

Доказательство. Заметим, что при любом фиксированном $y \in Y$ справедливо равенство $\sum_{i=1}^l [y_i = y] K\left(\frac{\rho(x, x_i)}{h}\right) = \sum_{i=1}^l [y^{(i)} = y] K\left(\frac{\rho(x, x^{(i)})}{h}\right)$, поскольку каждый объект обучающей выборки в обеих суммах участвует лишь единожды и значение соответствующего слагаемого не зависит от номера объекта. Отметим, что метод окна Парзена переменной ширины также можно переписать аналогичным образом.

Задача 2. Вася решает методом окна Парзена фиксированной ширины задачу классификации объектов на 8 классов по 3 вещественным признакам. Все признаки объектов в обучающей выборке принимают значения, по модулю не меньшие 1, и объекты i -го класса попадают в i -й октант пространства признаков. Вася использует евклидову метрику на пространстве признаков и берёт ширину окна $h = 1$. К какому из классов может его классификатор отнести точку 0?

Ответ. Ни к какому.

Решение. По условию расстояние между 0 и объектом обучающей выборки $\rho(x_i, 0) \geq \sqrt{(1-0)^2 + (1-0)^2 + (1-0)^2} = \sqrt{3} > 1 = h$. Таким образом, ни одна точка обучающей выборки не попадает в нужное окно с центром в нуле.

Задача 3. Докажите, что в качестве ядер в методе окна Парзена фиксированной ширины также можно брать выпуклую комбинацию ядер K_1, K_2, K_3, K_4 из примеров выше.

Доказательство. Заметим, что выпуклая комбинация также будет положительна при $|r| \leq 1$ и нулевая вне этого отрезка, чётна и не убывает на отрезке $[0, 1]$. Более того, также сохраняется нормировка:

$$\int_{-1}^1 [\alpha_1 K_1(r) + \alpha_2 K_2(r) + \alpha_3 K_3(r) + \alpha_4 K_4(r)] dr = \alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 = 1.$$

4.10. Метод потенциальных функций

Метод потенциальных функций - метрический классификатор, частный случай метода ближайших соседей. Позволяет с помощью простого алгоритма оценивать вес («важность») объектов обучающей выборки при решении задачи классификации.

В общем виде, алгоритм ближайших соседей есть:

$$a(x; X^l) = \arg \max_{y \in Y} \sum_{i=1}^l [y^{(i)} = y] w(i, x),$$

где $w(i, x)$ — вес, степень близости к объекту x его i -го соседа.

Метод потенциальных функций заключается в выборе в качестве веса $w(i, x)$, функции следующего вида:

$$w(i, x) = \gamma^{(i)} K\left(\frac{\rho(x, x^{(i)})}{h^{(i)}}\right),$$

где:

- $K(r)$ — ядро, не возрастает и положительно на $[0, 1]$,
- $\gamma^{(i)} \geq 0$ — «заряд» объекта $x^{(i)}$,
- $h^{(i)} > 0$ — параметр, задающий «ширину потенциала» объекта $x^{(i)}$.

Основная идея метода была навеяна электростатическим взаимодействием элементарных частиц. Известно, что потенциал («мера воздействия») электрического поля элементарной заряженной частицы в некоторой точке пространства пропорционален отношению заряда частицы (Q) к расстоянию до частицы (r): $\varphi(r) \sim \frac{Q}{r}$. Из-за этого в качестве $K(r)$ часто выступают функции: $\frac{1}{r}$ или $\frac{1}{r+a}$.

Метод потенциальных функций реализует полную аналогию указанного выше примера. При классификации объект проверяется на близость к объектам из обучающей выборки. Считается, что объекты из обучающей выборки «заряжены» своим классом, а мера «важности» каждого из них при классификации зависит от его «заряда» и расстояния до классифицируемого объекта.

4.10.1. Подбор параметров

Отметим, что параметрическую модель зависимости метода потенциальных функций можно существенно упростить (см. задачу ниже):

$$a(x; X^l) = \arg \max_{y \in Y} \sum_{i=1}^l [y_i = y] \gamma_i K\left(\frac{\rho(x, x_i)}{h_i}\right).$$

Из выражения выше следует, что в методе потенциальных функций используются две группы параметров: $\{h_i\}$ и $\{\gamma_i\}$.

«Ширина окна потенциала» h_i выбирается для каждого объекта из эмпирических соображений. «Заряд» γ_i объектов выборки можно подобрать, исходя из информации, содержащейся в выборке. Ниже приведен алгоритм, который позволяет «обучать» параметры $\gamma_1, \dots, \gamma_l$, то есть подбирать их значения по обучающей выборке X^l .

Алгоритм подбора параметров $\{\gamma_i\}$

Вход: обучающая выборка из l пар «объект-ответ» – $X^l = ((x_1, y_1), \dots, (x_l, y_l))$.

Выход: значения параметров γ_i для всех $i = \overline{1, l}$.

Описание:

1. Инициализация: $\gamma_i := 0$ для всех $i = \overline{1, l}$;
2. Повторять пункты 3 – 4, пока эмпирический риск $Q(a, X^l) > \varepsilon$ (то есть пока процесс не стабилизируется):
3. Выбрать очередной объект x_i из выборки X^l ;
4. Если $a(x_i) \neq y_i$, то $\gamma_i := \gamma_i + 1$;
5. Вернуть значения γ_i для всех $i = \overline{1, l}$.

4.10.2. Преимущества и недостатки

Преимущества метода потенциальных функций:

- Метод прост для понимания и алгоритмической реализации;
- Порождает потоковый алгоритм;
- Хранит лишь часть выборки, следовательно, экономит память.

Недостатки метода:

- Порождаемый алгоритм медленно сходится;
- Параметры $\{\gamma_i\}$ и $\{h_i\}$ настраиваются слишком грубо;
- Значения параметров $\gamma_1, \dots, \gamma_l$ зависят от порядка выбора объектов из выборки X^l .

4.10.3. Задачи для самопроверки

Задача 1.

Покажите, что параметрическую модель зависимости метода потенциальных функций можно переписать в следующем виде:

$$a(x; X^l) = \arg \max_{y \in Y} \sum_{i=1}^l [y_i = y] \gamma_i K \left(\frac{\rho(x, x_i)}{h_i} \right).$$

Решение.

Ответ следует из независимости суммы от перестановки слагаемых. $\forall y \in Y$:

$$\sum_{i=1}^l [y^{(i)} = y] \gamma^{(i)} K \left(\frac{\rho(x, x^{(i)})}{h^{(i)}} \right) = \sum_{i=1}^l [y_i = y] \gamma_i K \left(\frac{\rho(x, x_i)}{h_i} \right).$$

Задача 2.

Покажите, что метод потенциальных функций можно свести к линейному классификатору в случае $Y = \{-1, +1\}$ (бинарная классификация).

Решение.

Обозначим $\Gamma_y(x) = \sum_{i=1}^l [y^{(i)} = y] w(i, x)$ - оценка близости объекта x к классу y .

Тогда получим:

$$a(x; X^l) = \arg \max_{y \in Y} \Gamma_y(x) = \text{sign} (\Gamma_{+1}(x) - \Gamma_{-1}(x)) = \text{sign} \sum_{i=1}^l \gamma_i y_i K \left(\frac{\rho(x, x_i)}{h_i} \right),$$

что совпадает с линейной моделью классификации

$$a(x; X^l) = \text{sign} \sum_{j=1}^n \gamma_j f_j(x),$$

где

- $f_j(x) = y_j K \left(\frac{\rho(x, x_j)}{h_j} \right)$ – новые признаки объекта x ,
- γ_j – веса линейного классификатора,
- $n = l$ – число признаков равно числу объектов обучения.

Задача 3.

Привести пример выборки $(x_1, y_1), \dots, (x_l, y_l)$, такой, что метод потенциальных функций построит модель с эмпирическим риском $Q(a, X^l) = \frac{1}{l} \sum_{i=1}^l [a(x_i) \neq y_i] = \frac{1}{l}$.

Решение.

Рассмотрим l -мерное пространство признаков $X = R^l$. В качестве выборки можно взять набор объектов, геометрически находящихся в вершинах l -симплекса. Тогда $\rho(x_i, x_j) = \text{const}$ (для простоты положим $\rho(x_i, x_j) = 1$). Остается лишь определить множество ответов: $y_1 = +1, y_2 = +1, \dots, y_{l-1} = +1, y_l = -1$. Используя результат задачи 2 (положив $h_i = 1$), получим:

$$a(x; X^l) = \text{sign}\left(\sum_{i=1}^{l-1} \gamma_i - \gamma_l\right).$$

Алгоритм подбора параметров $\{\gamma_i\}$ приведет к следующему набору (уже на второй итерации): $\gamma_1 = 1, \dots, \gamma_{l-1} = 0, \gamma_l = 1$. Т.е. $\forall x \in X^l$

$$a(x; X^l) = 1.$$

Воспользовавшись формулой для эмпирического риска получаем требуемое.

Глава 5

Метод опорных векторов

5.1. SVM-классификация

5.1.1. Постановка задачи

Метод опорных векторов (Support Vector Machine, SVM) решает задачу бинарной классификации, где требуется найти оптимальную гиперплоскость, разделяющую два класса в пространстве признаков. Оптимальность понимается как максимизация ширины разделяющей полосы между классами.

5.1.2. Математическая формализация

Пусть дана обучающая выборка $\{(x_i, y_i)\}_{i=1}^{\ell}$, где $x_i \in \mathbb{R}^n$ - векторы признаков, $y_i \in \{-1, +1\}$ - метки классов. Разделяющая гиперплоскость описывается уравнением:

$$\langle w, x \rangle - w_0 = 0,$$

где $w \in \mathbb{R}^n$ - вектор весов, $w_0 \in \mathbb{R}$ - порог.

5.1.3. Условия разделимости

Для корректной классификации должны выполняться условия:

$$\begin{cases} \langle w, x_i \rangle - w_0 \geq +1, & \text{если } y_i = +1 \\ \langle w, x_i \rangle - w_0 \leq -1, & \text{если } y_i = -1 \end{cases}$$

Эти условия можно объединить:

$$y_i(\langle w, x_i \rangle - w_0) \geq 1, \quad i = 1, \dots, \ell$$

5.1.4. Оптимизационная задача

Ширина разделяющей полосы равна $\frac{2}{\|w\|}$. Задача максимизации ширины эквивалентна задаче минимизации:

$$\frac{1}{2} \|w\|^2 \rightarrow \min_{w, w_0}$$

при ограничениях $y_i(\langle w, x_i \rangle - w_0) \geq 1$.

5.1.5. Двойственная задача

Применяя метод множителей Лагранжа, получаем двойственную задачу:

$$L(w, w_0, \lambda) = \frac{1}{2} \|w\|^2 - \sum_{i=1}^{\ell} \lambda_i (y_i (\langle w, x_i \rangle - w_0) - 1)$$

Условия оптимальности:

$$\begin{cases} w = \sum_{i=1}^{\ell} \lambda_i y_i x_i \\ \sum_{i=1}^{\ell} \lambda_i y_i = 0 \end{cases}$$

5.1.6. Ядра

Для нелинейной классификации используется переход в пространство признаков большей размерности через отображение $\phi(x)$. Скалярное произведение заменяется на ядро:

$$K(x, z) = \langle \phi(x), \phi(z) \rangle$$

Популярные ядра:

- Линейное: $K(x, z) = \langle x, z \rangle$
- Полиномиальное: $K(x, z) = (\langle x, z \rangle + 1)^d$
- RBF: $K(x, z) = \exp(-\gamma \|x - z\|^2)$

5.1.7. Дискриминантная функция в ядовом пространстве

После применения ядрового преобразования классификация новых точек осуществляется с помощью дискриминантной функции, которая принимает вид:

$$f(x) = \sum_{i=1}^{\ell} \lambda_i y_i K(x_i, x) - w_0$$

где:

- x_i - опорные векторы из обучающей выборки
- λ_i - множители Лагранжа (двойственные переменные)
- y_i - метки классов опорных векторов
- $K(x_i, x)$ - значение ядровой функции между опорным вектором и классифицируемой точкой
- w_0 - порог, определяющий сдвиг разделяющей гиперплоскости

Важно отметить, что в этой формуле суммирование происходит только по опорным векторам, так как для остальных точек обучающей выборки $\lambda_i = 0$. Это свойство обеспечивает эффективность вычислений при классификации новых точек.

Решающее правило для определения класса новой точки:

$$\text{class}(x) = \text{sign}(f(x)) = \begin{cases} +1, & \text{если } f(x) > 0 \\ -1, & \text{если } f(x) < 0 \end{cases}$$

5.1.8. Мягкие границы

Для случая линейно неразделимой выборки вводятся переменные ослабления $\xi_i \geq 0$:

$$y_i(\langle w, x_i \rangle - w_0) \geq 1 - \xi_i$$

Целевая функция модифицируется:

$$\frac{1}{2}\|w\|^2 + C \sum_{i=1}^{\ell} \xi_i \rightarrow \min_{w, w_0, \xi}$$

где $C > 0$ - параметр регуляризации.

5.1.9. Задача 1

Условие:

Дан набор точек в двумерном пространстве с метками классов:

$$\begin{aligned} x_1 &= (0, 0), \quad y_1 = -1 \\ x_2 &= (2, 0), \quad y_2 = +1 \\ x_3 &= (0, 2), \quad y_3 = +1 \\ x_4 &= (2, 2), \quad y_4 = +1 \end{aligned}$$

В результате обучения SVM с линейным ядром получены следующие значения двойственных переменных:

$$\lambda_1 = 0.5, \quad \lambda_2 = 0, \quad \lambda_3 = 0.5, \quad \lambda_4 = 0$$

Найдите вектор весов w и определите, является ли точка $x_{\text{new}} = (1, 1)$ опорным вектором, если известно, что она лежит точно на разделяющей гиперплоскости.

Решение:

1. Найдем вектор весов w через опорные векторы:

$$w = \sum_{i=1}^4 \lambda_i y_i x_i$$

2. Подставляем известные значения:

$$\begin{aligned} w &= 0.5 \cdot (-1) \cdot (0, 0) + 0 \cdot (+1) \cdot (2, 0) + \\ &\quad + 0.5 \cdot (+1) \cdot (0, 2) + 0 \cdot (+1) \cdot (2, 2) \\ &= (0, 0) + (0, 0) + (0, 0.6) + (0, 0) \\ &= (0, 1) \end{aligned}$$

3. Для точек на разделяющей гиперплоскости выполняется:

$$\langle w, x \rangle - w_0 = 0$$

4. Подставляя координаты $x_{\text{new}} = (1, 1)$:

$$\langle (0, 1), (1, 1) \rangle - w_0 = 0$$

$$w_0 = 1$$

5. Чтобы точка была опорным вектором, она должна лежать на границе разделяющей полосы, то есть:

$$y_{\text{new}}(\langle w, x_{\text{new}} \rangle - w_0) = \pm 1$$

6. Проверяем это условие:

$$\langle (0, 1), (1, 1) \rangle - 1 = 0$$

Получаем 0, что не равно ± 1 .

Ответ: $w = (0, 1)$. Точка x_{new} не является опорным вектором.

5.1.10. Задача 2

Условие:

Дано множество точек в двумерном пространстве:

$$\begin{aligned} x_1 &= (1, 1), & y_1 &= +1 \\ x_2 &= (2, 2), & y_2 &= +1 \\ x_3 &= (0, 0), & y_3 &= -1 \\ x_4 &= (-1, 1), & y_4 &= -1 \end{aligned}$$

После обучения SVM получена разделяющая гиперплоскость $2/3 \cdot x_{i1} + 1/3 \cdot x_{i2} - 1 = 0$. Определите, какие из точек являются опорными векторами.

Решение:

Опорными векторами являются точки, лежащие на границе разделяющей полосы. Для их определения нужно:

1. Запишем вектор весов из уравнения гиперплоскости:

$$w = (2/3, 1/3) \quad w_0 = 1$$

2. Вычислим отступ для каждой точки по формуле:

$$M(x) = y(\langle w, x \rangle - w_0) = y(2x_{i1} + x_{i2} - 3)$$

3. Проверяем каждую точку:

$$\begin{aligned} M(x_1) &= (+1)(2/3 \cdot 1 + 1/3 \cdot 1 - 1) = 0 \\ M(x_2) &= (+1)(2/3 \cdot 2 + 1/3 \cdot 2 - 1) = 1 \\ M(x_3) &= (-1)(2/3 \cdot 0 + 1/3 \cdot 0 - 1) = 1 \\ M(x_4) &= (-1)(2/3 \cdot (-1) + 1/3 \cdot 1 - 1) = 4/3 \end{aligned}$$

4. Опорными векторами являются точки с отступом, равным единице, а также точки, лежащие на разделяющей гиперплоскости (отступ равен нулю).

Ответ: Точки x_2 и x_3 являются опорными векторами.

5.1.11. Задача 3

Условие:

При обучении SVM с полиномиальным ядром второй степени $K(x, z) = (\langle x, z \rangle + 1)^2$ получены следующие опорные векторы:

$$\begin{aligned} x_1 &= (2, 0), \quad \lambda_1 = 0.2, \quad y_1 = +1 \\ x_2 &= (0, 1), \quad \lambda_2 = 0.3, \quad y_2 = -1 \\ x_3 &= (1, 1), \quad \lambda_3 = 0.1, \quad y_3 = +1 \end{aligned}$$

К какому классу будет отнесена точка $x_{\text{new}} = (1, 0)$, если $w_0 = 0.5$?

Решение:

Для определения класса точки нужно найти знак дискриминантной функции:

$$f(x) = \sum_{i=1}^3 \lambda_i y_i K(x_i, x) - w_0$$

Вычислим значения ядра для x_{new} и каждого опорного вектора:

$$\begin{aligned} K(x_1, x_{\text{new}}) &= (2 \cdot 1 + 0 \cdot 0 + 1)^2 = 9 \\ K(x_2, x_{\text{new}}) &= (0 \cdot 1 + 1 \cdot 0 + 1)^2 = 1 \\ K(x_3, x_{\text{new}}) &= (1 \cdot 1 + 1 \cdot 0 + 1)^2 = 4 \end{aligned}$$

Подставляем в дискриминантную функцию:

$$\begin{aligned} f(x_{\text{new}}) &= 0.2 \cdot (+1) \cdot 9 + 0.3 \cdot (-1) \cdot 1 + 0.1 \cdot (+1) \cdot 4 - 0.5 \\ &= 1.8 - 0.3 + 0.4 - 0.5 = 1.4 \end{aligned}$$

Так как $f(x_{\text{new}}) > 0$, точка относится к классу +1.

Ответ: Точка x_{new} будет отнесена к классу +1.

5.2. SVM-регрессия

5.2.1. Постановка задачи

В задаче регрессии требуется найти функцию $f(x) = w^T \phi(x) + b$, которая аппроксимирует целевые значения y на основе входных данных x , минимизируя ошибки предсказания.

В SVM-регрессии вводится допустимая область погрешностей — ϵ -окрестность. Это означает, что отклонения $|f(x_i) - y_i|$ в пределах ϵ считаются несущественными, и модель игнорирует их. Цель — минимизировать сложность модели, связанную с $\|w\|$, штрафуя при этом за отклонения, выходящие за пределы ϵ .

5.2.2. Прямая задача

Функция потерь. Для задачи SVM-регрессии используется ϵ -чувствительная функция потерь:

$$L_\epsilon(f(x), y) = \begin{cases} 0, & \text{если } |f(x) - y| \leq \epsilon, \\ |f(x) - y| - \epsilon, & \text{иначе.} \end{cases}$$

Прямая постановка задачи:

$$\min_{w,b} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n L_\epsilon(f(x_i), y_i),$$

где:

- $\|w\|^2$ — регуляризационный член, минимизирующий сложность модели,
- C — коэффициент, регулирующий баланс между штрафами за ошибки и сложностью модели,
- $L_\epsilon(f(x_i), y_i)$ — штраф за выход за пределы ϵ -окрестности.

5.2.3. Преобразование задачи

Для учёта отклонений выше ϵ вводятся штрафные переменные ξ_i и ξ_i^* :

- ξ_i — превышение сверху ($y_i > f(x_i) + \epsilon$),
- ξ_i^* — превышение снизу ($y_i < f(x_i) - \epsilon$).

Задача минимизации принимает вид:

$$\min_{w,b,\xi,\xi^*} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n (\xi_i + \xi_i^*),$$

при ограничениях:

$$\begin{aligned} y_i - (w^T \phi(x_i) + b) &\leq \epsilon + \xi_i, \\ (w^T \phi(x_i) + b) - y_i &\leq \epsilon + \xi_i^*, \\ \xi_i, \xi_i^* &\geq 0. \end{aligned}$$

5.2.4. Метод Лагранжа

Для решения задачи вводится лагранжиан, который включает:

- Целевую функцию,
- Ограничения через множители Лагранжа ($\alpha, \alpha^*, \eta, \eta^*$).

Лагранжиан записывается как:

$$\begin{aligned} L(w, b, \xi, \xi^*, \alpha, \alpha^*, \eta, \eta^*) = & \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n (\xi_i + \xi_i^*) - \\ & - \sum_{i=1}^n \alpha_i [\epsilon + \xi_i - y_i + w^T \phi(x_i) + b] - \\ & - \sum_{i=1}^n \alpha_i^* [\epsilon + \xi_i^* + y_i - w^T \phi(x_i) - b] - \\ & - \sum_{i=1}^n (\eta_i \xi_i + \eta_i^* \xi_i^*). \end{aligned}$$

Для нахождения двойственной задачи необходимо минимизировать L по w, b, ξ, ξ^* и максимизировать по множителям Лагранжа.

5.2.5. Условия оптимальности

1. Производная по w :

$$\frac{\partial L}{\partial w} = w - \sum_{i=1}^n (\alpha_i - \alpha_i^*) \phi(x_i) = 0 \implies w = \sum_{i=1}^n (\alpha_i - \alpha_i^*) \phi(x_i).$$

2. Производная по b :

$$\frac{\partial L}{\partial b} = \sum_{i=1}^n (\alpha_i - \alpha_i^*) = 0.$$

3. Производные по ξ_i и ξ_i^* :

$$\alpha_i + \eta_i = C, \quad \alpha_i^* + \eta_i^* = C, \quad 0 \leq \alpha_i, \alpha_i^* \leq C.$$

5.2.6. Двойственная задача

Подставляя условия оптимальности в лагранжиан, исключаем w , b , ξ_i , ξ_i^* . Получаем двойственную задачу:

$$\max_{\alpha, \alpha^*} -\frac{1}{2} \sum_{i,j=1}^n (\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*) K(x_i, x_j) - \epsilon \sum_{i=1}^n (\alpha_i + \alpha_i^*) + \sum_{i=1}^n y_i (\alpha_i - \alpha_i^*),$$

где $K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$ — ядровая функция.

Ограничения:

$$\sum_{i=1}^n (\alpha_i - \alpha_i^*) = 0, \quad 0 \leq \alpha_i, \alpha_i^* \leq C.$$

Для решения двойственной задачи используется метод квадратичного программирования.

5.2.7. Построение финальной модели

После решения двойственной задачи оптимальные α_i и α_i^* определяют параметры модели:

$$f(x) = \sum_{i=1}^n (\alpha_i - \alpha_i^*) K(x_i, x) + b.$$

Смещение b вычисляется через опорные векторы — точки, где выполняется одно из условий:

$$y_i - (w^T \phi(x_i) + b) = \epsilon, \quad \text{или} \quad y_i - (w^T \phi(x_i) + b) = -\epsilon.$$

Опорные векторы ($\alpha_i > 0$ или $\alpha_i^* > 0$) определяют форму модели.

5.2.8. Выбор ядра

Выбор ядра играет ключевую роль в качестве работы модели SVM-регрессии. Различные ядра по-разному преобразуют входные данные, что может существенно повлиять на точность предсказаний и обобщающую способность модели.

Выбор ядра зависит от особенностей данных, структуры зависимости и доступных вычислительных ресурсов. Экспериментальная проверка нескольких типов ядер и последующая оценка метрик качества модели — это ключевой этап в процессе выбора оптимального ядра.

На рисунке ниже представлена SVM-регрессия с тремя типами ядер: RBF, линейным и полиномиальным.

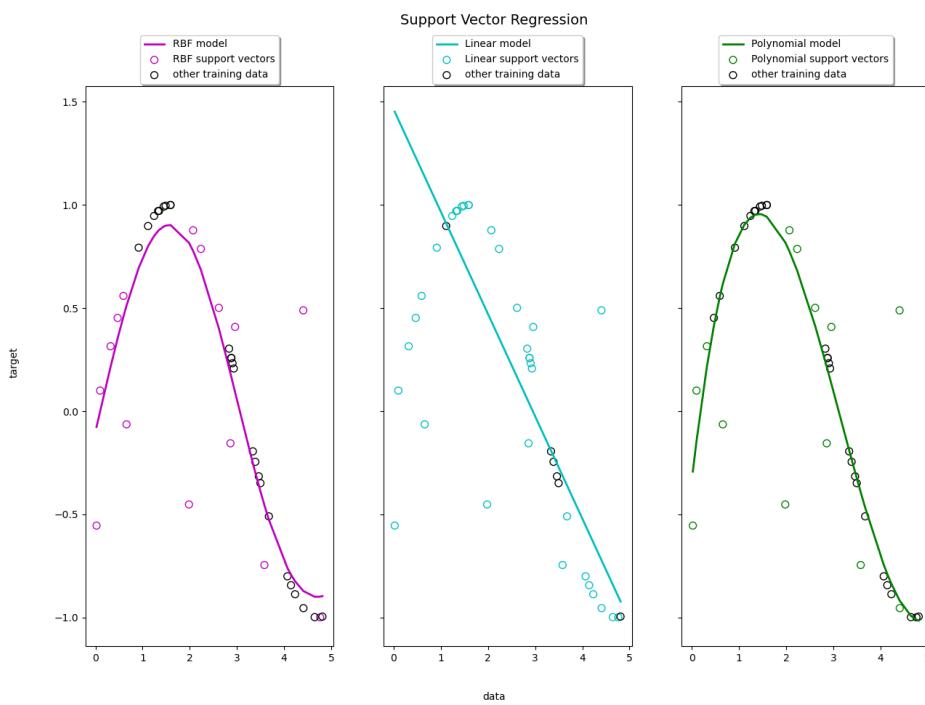


Рис. 5.1. Сравнение SVM-регрессия с разными типами ядер

5.2.9. Задача 1

Условие:

Рассмотрим следующий набор точек, лежащих на границе ϵ -окрестности для SVM-регрессии с линейным ядром:

$$\{(1, 2), (2, 3), (3, 5), (4, 6)\}$$

Постройте регрессионную модель и найдите смещение b , если $w = 2$, $\epsilon = 1$.

Решение:

Для нахождения смещения b необходимо использовать точки, которые лежат на границах ϵ -окрестности. Мы знаем, что для таких точек выполняется равенство:

$$y_i - (wx_i + b) = \epsilon \quad \text{или} \quad y_i - (wx_i + b) = -\epsilon$$

Найдем для каждой точки b , подставив их координаты в эти уравнения:

- Для точки $(1, 2)$:

$$2 - (2 \cdot 1 + b) = 1 \Rightarrow 2 - (2 + b) = 1 \Rightarrow -b = 1 \Rightarrow b = -1$$

- Для точки $(2, 3)$:

$$3 - (2 \cdot 2 + b) = 1 \Rightarrow 3 - (4 + b) = 1 \Rightarrow -b = 2 \Rightarrow b = -2$$

- Для точки $(3, 5)$:

$$5 - (2 \cdot 3 + b) = 1 \Rightarrow 5 - (6 + b) = 1 \Rightarrow -b = 2 \Rightarrow b = -2$$

- Для точки $(4, 6)$:

$$6 - (2 \cdot 4 + b) = 1 \Rightarrow 6 - (8 + b) = 1 \Rightarrow -b = 3 \Rightarrow b = -3$$

Для вычисления окончательного значения смещения b , усредняем найденные значения:

$$b_{\text{avg}} = \frac{-1 + (-2) + (-2) + (-3)}{4} = \frac{-8}{4} = -2$$

Таким образом, смещение $b = -2$.

Регрессионная модель для SVM с линейным ядром имеет следующий вид:

$$f(x) = wx + b$$

Подставляем данное в условии значения $w = 2$ и найденное значение $b = -2$:

$$f(x) = 2x - 2$$

Это и есть наша линейная регрессионная модель.

Ответ: $f(x) = 2x - 2$.

5.2.10. Задача 2

Условие:

У нас есть два набора данных для задачи регрессии:

- Набор 1: $\{(1, 2), (2, 3), (3, 4), (4, 5)\}$
- Набор 2: $\{(1, 1), (2, 4), (3, 9), (4, 16)\}$

Предположим, что мы используем SVM-регрессию с различными типами ядер (линейное, полиномиальное, RBF). Определите, какое ядро будет оптимальным для каждого набора данных.

Решение:

- Набор 1: данные имеют линейную зависимость, следовательно, линейное ядро будет лучшим выбором.
- Набор 2: данные имеют квадратичную зависимость, следовательно, оптимально будет использовать полиномиальное ядро второй степени.

Ответ: для первого набора данных оптимальным ядром будет линейное, для второго набора данных - полиномиальное второй степени.

5.2.11. Задача 3

Условие:

Дан набор данных для SVM-регрессии с линейным ядром:

$$\{(1, 2), (2, 2.8), (3, 5.2), (4, 8)\}.$$

Параметры модели: $w = 1.5$, $b = 0.5$, $\epsilon = 0.5$.

1. Определите, какие из точек набора данных находятся вне ϵ -окрестности (требуют штрафных переменных ξ или ξ^*).
2. Вычислите значения штрафных переменных для этих точек.

Решение:

1. Определение границ ϵ -окрестности: Уравнение модели SVM-регрессии с линейным ядром:

$$f(x) = wx + b.$$

Подставляем данные в условии значения:

$$f(x) = 1.5x + 0.5.$$

Границы ϵ -окрестности:

$$f(x) - \epsilon \leq y \leq f(x) + \epsilon.$$

2. Проверка точек:

- Для точки $(1, 2)$:

$$f(1) = 1.5 \cdot 1 + 0.5 = 2.0, \quad 2 - 0.5 \leq 2 \leq 2 + 0.5 \quad (\text{в окрестности}).$$

- Для точки $(2, 2.8)$:

$$f(2) = 1.5 \cdot 2 + 0.5 = 3.5, \quad 3.5 - 0.5 \not\leq 2.8 \leq 3.5 + 0.5 \quad (\text{вне окрестности}).$$

- Для точки $(3, 5.2)$:

$$f(3) = 1.5 \cdot 3 + 0.5 = 5.0, \quad 5.0 - 0.5 \leq 5.2 \leq 5.0 + 0.5 \quad (\text{в окрестности}).$$

- Для точки $(4, 8)$:

$$f(4) = 1.5 \cdot 4 + 0.5 = 6.5, \quad 6.5 - 0.5 \not\leq 8.0 \leq 6.5 + 0.5 \quad (\text{вне окрестности}).$$

3. Штрафные переменные:

- Для точки $(2, 2.8)$:

$$\xi_i = f(2) - y - \epsilon = 3.5 - 2.8 - 0.5 = 0.2.$$

- Для точки $(4, 8)$:

$$\xi_i^* = y - f(4) - \epsilon = 8 - 6.5 - 0.5 = 1.0.$$

Таким образом, штрафные переменные:

$$\xi_2^* = 0.2, \quad \xi_4^* = 1.0.$$

Ответ: $\xi_2^* = 0.2, \quad \xi_4^* = 1.0.$

1-norm SVM (LASSO SVM)

Аппроксимация эмпирического риска с L_1 -регуляризацией

$$\sum_{i=1}^{\ell} (1 - M_i(w, w_0))_+ + \mu \sum_{j=1}^n |w_j| \rightarrow \min_{w, w_0}$$

Плюс: отбор признаков с параметром селективности μ

- чем больше μ , тем меньше признаков останется

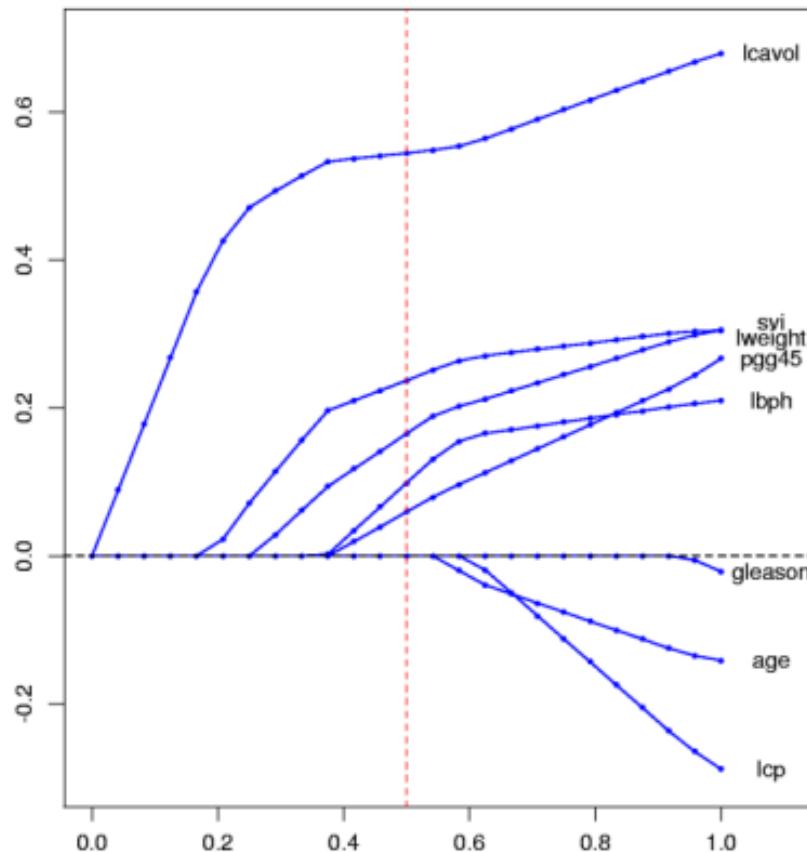
Минус: слишком агрессивный отбор признаков

- по мере увеличения μ признак может быть отброшен, хотя y существенно зависит от него (даже когда ещё не все шумовые признаки отброшены)

Сравнение L_2 и L_1 регуляризации

Зависимость весов w_j от коэффициента $\frac{1}{\mu}$:

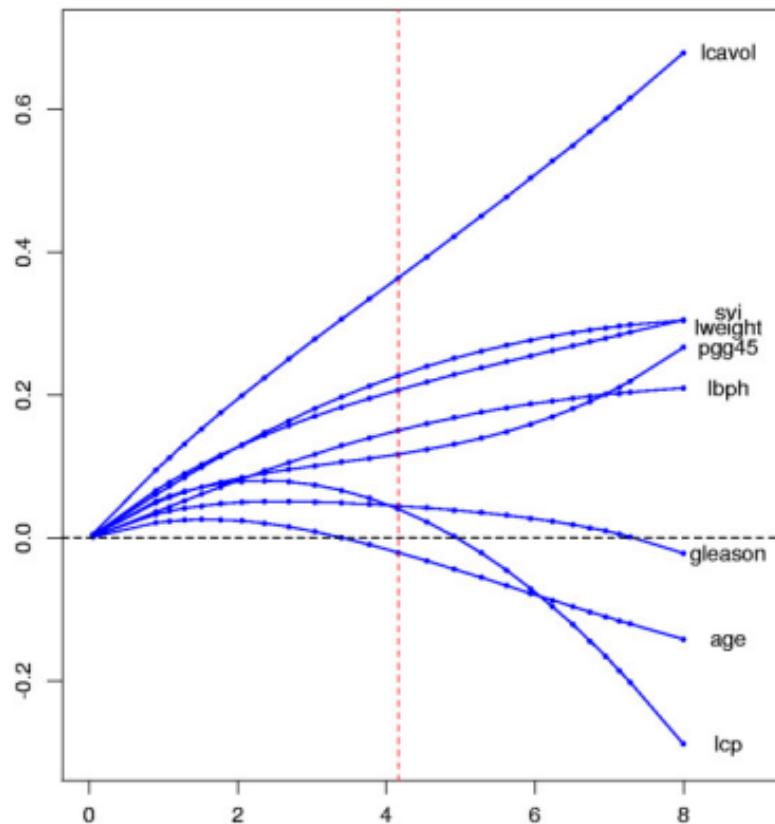
- L_1 регуляризатор: $\mu \sum_j |w_j|$



- L_2 регуляризатор: $\mu \sum_j w_j^2$

Doubly Regularized SVM (Elastic Net SVM)

$$C \sum_{i=1}^{\ell} (1 - M_i(w, w_0))_+ + \mu \sum_{j=1}^n |w_j| + \frac{1}{2} \sum_{j=1}^n w_j^2 \rightarrow \min_{w, w_0}$$



Плюсы:

- Параметр селективности μ управляет отбором признаков: чем больше μ , тем меньше признаков останется
- Есть эффект группировки (grouping effect): значимые зависимые признаки отбираются вместе

Минусы:

- Шумовые признаки также группируются и могут вместе оставаться в модели
- Приходится подбирать два параметра регуляризации μ, τ (есть специальные методы, например, regularization path)

Elastic Net Analysis

Elastic Net менее жёстко отбирает признаки.

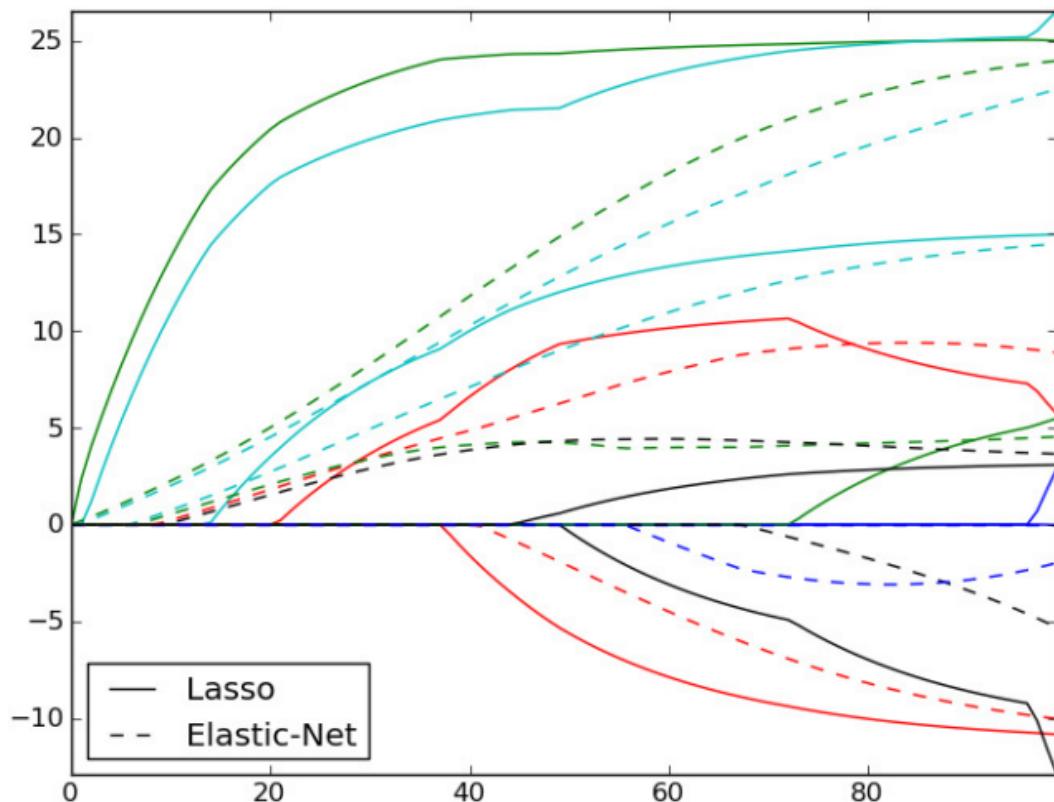


Рис. 5.2. Зависимости весов w_j от коэффициента $\log \frac{1}{\mu}$

Support Features Machine (SFM)

$$C \sum_{i=1}^{\ell} (1 - M_i(w, w_0))_+ + \sum_{j=1}^n R_\mu(w_j) \rightarrow \min_{w, w_0}$$

$$R_\mu(w_j) = \begin{cases} 2\mu|w_j|, & |w_j| \leq \mu \\ \mu^2 + w_j^2, & |w_j| \geq \mu \end{cases}$$

Плюсы

- Только один параметр регуляризации μ
- Отбор признаков с параметром селективности μ
- Эффект группировки: значимые зависимые признаки ($|w_j| > \mu$) входят в решение совместно (как в *Elastic Net*)
- Шумовые признаки ($|w_j| < \mu$) не группируются и подавляются независимо друг от друга (как в *LASSO*)

Relevance Features Machine (RFM)

$$C \sum_{i=1}^{\ell} (1 - M_i(w, w_0))_+ + \sum_{j=1}^n \ln(w_j^2 + \frac{1}{\mu}) \rightarrow \min_{w, w_0}$$

$$R(w) = \ln(w^2 + \frac{1}{\mu}), \quad \mu = 0.1, 1, 100$$

Плюсы

- Только один параметр регуляризации μ
- Отбор признаков с параметром селективности μ
- Есть эффект группировки
- Лучше отбирает набор значимых признаков, когда они только совместно обеспечивают хорошее решение

Задачи

Задача 1

Качественно объяснить, почему L_1 -регуляризатор приводит к отбору признаков

Ответ:

Аппроксимация эмпирического риска с L_1 -регуляризацией:

$$\sum_{i=1}^{\ell} (1 - M_i(w, w_0))_+ + \mu \sum_{j=1}^n |w_j| \rightarrow \min_{w, w_0}$$

Почему L_1 -регуляризатор приводит к отбору признаков?

Замена переменных:

$$u_j = \frac{1}{2}(|w_j| + w_j), \quad v_j = \frac{1}{2}(|w_j| - w_j).$$

Тогда

$$w_j = u_j - v_j \quad |w_j| = u_j + v_j.$$

$$\sum_{i=1}^{\ell} (1 - M_i(u - v, w_0))_+ + \mu \sum_{j=1}^n (u_j + v_j) \rightarrow \min_{u,v},$$

$$u_j \geq 0, \quad v_j \geq 0, \quad j = 1, \dots, n.$$

чем больше μ , тем больше индексов j таких, что $u_j = v_j = 0$, но тогда $w_j = 0$, значит, **признак не учитывается**.

Задача 2

Привести пример нежелательного эффекта в процессе обучения, с которым поможет справиться регуляризация

Ответ:

Регуляризация помогает в случае линейной зависимости (мультиколлинеарности) признаков:

Пусть построен классификатор: $a(x, w) = \text{sign}\langle w, x \rangle$

Мультиколлинеарность: $\exists u \in \mathbb{R}^n: \forall x \in X \langle u, x \rangle = 0$

Неединственность решения и рост нормы вектора весов: $\forall \gamma \in \mathbb{R}$
 $a(x, w) = \text{sign}\langle w, x \rangle = \text{sign}\langle w + \gamma u, x \rangle$

Проявления переобучения:

- слишком большие веса $|w_j|$ разных знаков
- неустойчивость дискриминантной функции $\langle w, x \rangle$
- $Q(X^\ell) \ll Q(X^k)$

Способ уменьшить переобучение:

регуляризация $\|w\| \rightarrow \min$ (сокращение весов, *weight decay*)

Задача 3

Дана задача оптимизации:

$$\frac{1}{2}(wx - b)^2 + \lambda|w| \rightarrow \min_w,$$

где $x, b \in \mathbb{R}; \lambda \geq 0$

При каких λ данная задача имеет решение $w_0 \neq 0$?

Ответ:

Находим правую и левую односторонние производные в нуле и рассматриваем, когда они больше и меньше 0 соответственно:

$$\begin{cases} -xb + \lambda > 0 \\ -xb - \lambda < 0 \end{cases} \Leftrightarrow \lambda > |xb|$$
$$\lambda \geq 0$$

Это условие на λ , при котором задача имеет решение $w_0 = 0$, поэтому нам подходит $\lambda \in [0; |xb|]$.

Глава 6

Метод главных компонент

Введение в метод главных компонент(PCA)

Метод главных компонент (Principal Component Analysis, PCA) – это статистический метод, используемый для снижения размерности данных с сохранением наиболее значимой информации. PCA находит новые признаки (главные компоненты), которые представляют собой линейные комбинации исходных признаков, причем эти компоненты ортогональны и ранжированы по величине объясняемой дисперсии. **Основные этапы метода:**

1. Центрирование данных:

Данные центрируются так, чтобы среднее значение каждой переменной было равно нулю:

$$X_c = X - \bar{X},$$

где X - исходная матрица данных (размер $n \times p$), \bar{X} - вектор средних значений по столбцам.

2. Построение ковариационной матрицы:

Вычисляется ковариационная матрица:

$$\Sigma = \frac{1}{n-1} X_c^T X_c$$

где Σ - симметричная матрица размером $p \times p$.

3. Собственные значения и собственные векторы:

Решается задача нахождения собственных значений и собственных векторов ковариационной матрицы:

$$\Sigma \mathbf{v}_i = \lambda_i \mathbf{v}_i$$

где λ_i - собственные значения, \mathbf{v}_i - соответствующие им собственные векторы.

4. Ранжирование главных компонент:

Собственные значения упорядочиваются по убыванию:

$$\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_p$$

Первые несколько компонент, соответствующие самым большим собственным значениям, объясняют большую часть дисперсии данных.

5. Проекция данных:

Данные проецируются на главные компоненты:

$$Z = X_c V_k,$$

где V_k – матрица k собственных векторов, соответствующих k наибольшим собственным значениям, Z – матрица данных в пространстве главных компонент.

Свойства метода:

- Главные компоненты ортогональны:

$$\mathbf{v}_i^T \mathbf{v}_j = 0, \quad i \neq j$$

- Дисперсия объясняется последовательностью собственных значений:

$$\text{Объясненная дисперсия} = \frac{\sum_{i=1}^k \lambda_i}{\sum_{i=1}^p \lambda_i}.$$

Задачи на использование метода главных компонент

Задача 1: Вклад признаков в главные компоненты

Пусть $X \in R_{n \times p}$ набор данных с n образцами (строками) и p признаками (столбцами). PCA стремится найти набор собственных векторов (главных компонент), которые максимизируют дисперсию данных при проецировании на эти векторы.

Задача состоит в том, чтобы математически оценить, какой вклад вносит каждый признак в главные компоненты, и проранжировать признаки в зависимости от их вклада.

Решение: Метод PCA ищет собственные векторы \mathbf{v}_i и собственные значения λ_i удовлетворяющие:

$$\Sigma \mathbf{v}_i = \lambda_i \mathbf{v}_i,$$

где λ_i - величина дисперсии данных вдоль \mathbf{v}_i . Собственные векторы \mathbf{v}_i формируют матрицу $V = [\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_p]$, где каждый столбец \mathbf{v}_i указывает направления главных компонент.

Вклад признака в главные компоненты:

Каждый признак в X вносит вклад в главные компоненты через веса собственных векторов \mathbf{v}_i . Элементы v_{ij} (где v_{ij} – j -й элемент i -го собственного вектора) определяют значимость j -го признака для i -й главной компоненты.

Вклад j -го признака в i -ю главную компоненту оценивается как квадрат соответствующего элемента v_{ij}^2 .

Общий вклад j -го признака во все главные компоненты можно найти, суммируя его взвешенные вклады с учётом дисперсий (λ_i):

$$j = \sum_{i=1}^p \lambda_i v_{ij}^2.$$

Этот показатель учитывает как значимость признака для каждой компоненты

(v_{ij}^2) , так и долю дисперсии, объясняемую компонентой (λ_i).

На конкретном примере: Пусть собственные вектора образуют матрицу V:

$$V = \begin{bmatrix} 0.5 & 0.6 & 0.3 & 0.1 \\ 0.4 & -0.7 & 0.2 & 0.5 \\ -0.6 & 0.2 & 0.7 & -0.4 \\ 0.5 & 0.3 & -0.6 & -0.6 \end{bmatrix}$$

Собственные значения:

$$\Lambda = \text{diag}(4.0, 2.5, 1.2, 0.3)$$

Посчитаем вклад признака 1($j = 1$): Для этого берём первую строчку V :

$$v_{1\cdot} = [0.5, 0.6, 0.3, 0.1]$$

Считаем:

$$\begin{aligned} \text{Contribution}_1 &= (0.5^2 \cdot 4.0) + (0.6^2 \cdot 2.5) + (0.3^2 \cdot 1.2) + (0.1^2 \cdot 0.3) \\ &= 1.0 + 0.9 + 0.108 + 0.003 = 2.011 \end{aligned}$$

То же самое повторяем для остальных строк и находим максимальное значение.

Задача 2: Ошибка "реконструкции" РСА

Пусть $X \in R_{n \times p}$ набор данных с n образцами (строками) и p признаками (столбцами), с помощью метода главных компонент нужно:

- Спроецируйте данные в более низкоразмерное пространство, определяемое k главными компонентами.
- Реконструируйте исходные данные из пространства пониженной размерности.
- Вычислите ошибку реконструкции и оцените, как она меняется в зависимости от количества сохраняемых компонент k .

Решение: Для реконструкции данных из k -мерного подпространства используется обратная проекция:

$$\hat{X} = ZV_k^T + \bar{X}$$

Здесь: - ZV_k^T возвращает проекцию данных в исходное p -мерное пространство.
- Добавление \bar{X} восстанавливает исходное смещение данных.

Ошибка реконструкции должна показывать какую часть информации мы

потеряли при использовании только k компонент при репрезентации данных.

1. Определим ошибку реконструкции для одного объекта x_i :

$$E_i = \|x_i - \hat{x}_i\|^2 = \|(x_i - \bar{X}) - (z_i V_k^\top)\|^2$$

2. Обобщим на весь набор данных:

$$E = \frac{1}{n \times p} \sum_{i=1}^n \|x_i - \hat{x}_i\|^2$$

3. Заменим на выражение для \hat{x}_i :

$$E = \frac{1}{n \times p} \sum_{i=1}^n \|x_i - \bar{X} - ZV_k^\top\|^2$$

Мы получили выражение для ошибки реконструкции. Теперь докажем, что ошибка реконструкции E уменьшается монотонно с k , и когда $k = p$, $E = 0$.

1. Общая дисперсия данных - это след ковариационной матрицы, которая представляет собой сумму всех собственных значений:

$$\text{Total Variance} = \sum_{j=1}^p \lambda_j$$

2. Дисперсия, которую уловили k компонент это:

$$\text{Captured Variance} = \sum_{j=1}^k \lambda_j$$

3. Ошибка реконструкции по сути является дисперсией, которую не удалось уловить, то есть просто:

$$E = \text{Total Variance} - \text{Captured Variance} = \sum_{j=k+1}^p \lambda_j$$

4. Так как $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_p \geq 0$, добавление большего числа компонент ($k \rightarrow k + 1$) уменьшает E :

$$\sum_{j=k+1}^p \lambda_j > \sum_{j=k+2}^p \lambda_j$$

5. В тот момент, когда $k = p$, $\sum_{j=k+1}^p \lambda_j = 0 \Rightarrow E = 0$.

Задача 3: Построить критерий D-оптимальности для выбора лучших k-компонент

Набор данных представляет собой матрицу $n \times p$, где n - число образцов (строк), а p - число признаков (столбцов). Введём критерий D-оптимальности, используемый для выбора подмножества точек из набора данных, которое максимизирует детерминант информационной матрицы.

$$D_{opt} : \max \det(X^T X)$$

Ключевым свойством критерия D-оптимальности является то, что он максимизирует объём многомерной фигуры, которая получается из рассматриваемых признаков.

Нужно построить критерий D-оптимальности для выбора лучших k главных компонент, которые максимизируют детерминант объясненной дисперсии (или объём фигуры) в k -мерном подпространстве PCA.

Решение:

Критерий D-оптимальности для подпространства k задаётся максимизацией детерминанта информационной матрицы Λ_k :

$$D_{opt}(k) = \max \det(\Lambda_k)$$

Так как Λ_k является диагональной матрицей, её детерминант равен произведению собственных значений:

$$\det(\Lambda_k) = \prod_{i=1}^k \lambda_i$$

Итак, наша задача сводится к выбору k -мерного подпространства (т.е. первых k главных компонент), которые максимизируют произведение $\lambda_1 \cdot \lambda_2 \cdots \lambda_k$, что эквивалентно решению следующей задачи:

$$\max_{V_k} \prod_{i=1}^k \lambda_i$$

где λ_i - собственные значения матрицы ковариации Σ . Для вычисления $D_{opt}(k)$:

1. Центрируем данные:

$$X_c = X - \bar{X},$$

где \bar{X} - матрица средних значений.

2. Вычисляем ковариационную матрицу:

$$\Sigma = \frac{1}{n-1} X_c^T X_c$$

3. Находим собственные значения $\lambda_1, \lambda_2, \dots, \lambda_p$ и соответствующие собственные векторы $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_p$.

4. Выбираем первые k собственных значений $\lambda_1, \lambda_2, \dots, \lambda_k$, которые максимизируют:

$$\prod_{i=1}^k \lambda_i$$

Максимизация $\prod_{i=1}^k \lambda_i$ эквивалентна максимизации объёма **k -мерного эллипсоида**, описывающего данные в пространстве первых k главных компонент. Это позволяет отобрать k измерений, которые сохраняют максимальную дисперсию данных.

Глава 7

Нелинейные модели машинного обучения

Методы Ньютона-Рафсона, Ньютона-Гаусса

Мы уже познакомились с задачами линейной регрессии и обсудили несколько методов их решения. Но что делать если задача нелинейна? Оказывается, что идея локальной линейности гладкой функций позволяет свести задачу к более простому. На этой идее основаны методы второго порядка - Ньютона-Рафсона и Ньютона-Гаусса.

Метод Ньютона-Рафсона

Начнем немного сдалека, а именно рассмотрим задачу поиска нуля a функций $f(x)$. Пусть мы находимся в точке a^0 и хотим найти такое приращение h , чтобы приблизиться к точке a : $a^0 + h \approx a$. Применим разложение в ряд:

$$f(a^0 + h) = f(a^0) + f'(a^0)h + o(h)$$

$$f(a^0 + h) \approx f(a) = 0 \Rightarrow f(a^0) + f'(a^0)h \approx 0$$

Откуда

$$h \approx -\frac{f(a^0)}{f'(a^0)}$$

Значит, для поиска нуля выпуклой функций f можно применить следующий итеративный метод:

$$a^{k+1} = a^k - \frac{f(a^k)}{f'(a^k)}$$

Вернемся к начальной задаче:

- обучающая выборка $X^l = (x_i, y_i)$, где x_i - вектор признаков i -го объекта
- $y_i = y(x_i)$, $y : X \rightarrow Y$ - неизвестная регрессионная зависимость
- $f(x, \alpha)$ - нелинейная модель регрессии, где α - вектор параметров

Хотим решить задачу оптимизации методом наименьших квадратов:

$$Q(\alpha, X^l) = \sum_{i=1}^l (f(x_i, \alpha) - y_i)^2 \rightarrow \min_{\alpha}$$

Функция потерь $Q(\alpha, X^l)$ выпукла и гладкая в предположении гладкости $f(\alpha, x)$, поэтому для ее минимизации достаточно найти нуль производной (градиента). Применяя рассуждения выше и опустив технические детали, получим следующий итерационный процесс:

$$\alpha^{k+1} = \alpha^k - h_k (Q''(\alpha^k))^{-1} Q'(\alpha^k)$$

где $Q'(\alpha^k)$, $Q''(\alpha^k)$ - градиент и гессиан Q в точке α^k соответственно, h_k - величина шага.

Метод Ньютона-Гаусса

Подсчет обратного гессиана на каждой итерации может дорого обходиться, поэтому посмотрим на полученный выше результат с другой стороны. Для этого запишем несколько формул:

Компоненты градиента $Q(\alpha, X^l)$:

$$\frac{\partial Q(\alpha, X^l)}{\partial \alpha_j} = 2 \sum_{i=1}^l (f(x_i, \alpha) - y_i) \frac{\partial f(x_i, \alpha)}{\partial \alpha_j}$$

Компоненты гессиана:

$$\frac{\partial Q(\alpha, X^l)}{\partial \alpha_j \partial \alpha_k} = 2 \sum_{i=1}^l \frac{\partial f(x_i, \alpha)}{\partial \alpha_j} \frac{\partial f(x_i, \alpha)}{\partial \alpha_k} - 2 \sum_{i=1}^l (f(x_i, \alpha) - y_i) \frac{\partial^2 f(x_i, \alpha)}{\partial \alpha_j \partial \alpha_k}$$

Второе слагаемое в формуле выше полагается равным нулю, исходя из линейной аппроксимации функций f . Введем следующие обозначения и перепишем формулу итерации метода Ньютона-Рафсона:

$$F_k = \left(\frac{\partial f}{\partial \alpha_j}(x_i, \alpha^k) \right)_{i,j}$$

$$f_k = (f(x_i, \alpha^k))_i$$

Получим:

$$\alpha^{k+1} = \alpha^k - h_k (F_k^T F_k)^{-1} F_k^T (f_k - y)$$

Положив $\theta = (F_k^T F_k)^{-1} F_k^T (f_k - y)$, получим решение задачи линейной регрессии, где новые ответы обучающей выборки – $(f_k - y)$, с новой матрицей признаков – F_k . Таким образом, каждый шаг метода Ньютона-Гаусса сводится к задаче линейной регрессии.

Задачи

Задача 1: Локальная сходимость

Найдите допустимые значения начального приближения для поиска нуля функции $f(x) = \frac{x}{\sqrt{1+x^2}}$.

Решение:

Нуль функции f достигается в точке $a = 0$. Посчитаем производную f :

$$f'(x) = \left(\frac{x}{\sqrt{1+x^2}} \right)' = \frac{1}{\sqrt{1+x^2}} - x \cdot \frac{2x}{2(1+x^2)^{3/2}} = \frac{1}{(1+x^2)^{3/2}}$$

Распишем формулу итерации метода Ньютона:

$$a^{k+1} = a^k - \frac{f(a^k)}{f'(a^k)} = -(a^k)^3$$

Отсюда видно, что сходимость есть при $|a^0| < 1$. Делаем вывод о важности выбора начального приближения в данном методе.

Задача 2: Квадратичная задача

Как сработает метод Ньютона-Рафсона для поиска минимума задачи $f(x) = x^T Ax + bx + c$, где $x, b \in \mathbb{R}^n$, A - симметричная, положительно определенная матрица.

Решение:

Для поиска минимума нужно найти нуль градиента. Это и будет точкой минимума, так как задача выпукла. Посчитаем градиент и гессиан:

$$\nabla f(x) = Ax + b$$

$$\nabla^2 f(x) = A$$

Пусть x^0 начальная точка. Применяя формулу из метода Ньютона-Рафсона получим:

$$x^1 = x^0 - (\nabla^2 f(x^0))^{-1} \nabla f(x^0) = x^0 - A^{-1} (Ax^0 + b) = -A^{-1}b$$

но с другой стороны, градиент обращается в нуль в этой точке:

$$\nabla f(x) = 0 = Ax + b \Rightarrow x = -A^{-1}b$$

Значит для квадратичной задачи данный метод дает ответ за 1 шаг. Вообще говоря, если функция μ -выпукла и имеет M -липшицевый гессиан, то скорость сходимости локально квадратична.

Задача 3: Система уравнений

Составьте алгоритм решения следующей системы с помощью методов второго порядка:

$$\begin{cases} x^2 + y^2 = 4 \\ y = e^x \end{cases}$$

Решение:

Нужно найти нули следующей функции от двух переменных

$$F(x, y) = \begin{pmatrix} x^2 + y^2 - 4 \\ y - e^x \end{pmatrix}$$

Запишем итерацию метода Ньютона-Рафсона:

$$x^{k+1} = x^k - J_F(x^k)^{-1} F(x^k)$$

Его можно записать как:

$$J_F(x^k)(x^{k+1} - x^k) = -F(x^k)$$

Посчитаем матрицу якоби функции F :

$$J_F(x) = \begin{pmatrix} 2x & 2y \\ -e^x & 1 \end{pmatrix}$$

Тогда на каждом шаге нужно решить следующую систему:

$$\begin{pmatrix} 2x^k & 2(y^k)^2 \\ -e^{x^k} & 1 \end{pmatrix} \begin{pmatrix} c_1^{k+1} \\ c_2^{k+1} \end{pmatrix} = \begin{pmatrix} (x^k)^2 + (y^k)^2 - 4 \\ (y^k) - e^{x^k} \end{pmatrix}$$

где $c^{k+1} = -(x^{k+1} - x^k)$. Тогда для решения данной задачи нужно взять начальную точку и проделать несколько итераций описанных уравнениями выше.

Backfitting

На практике встречаются задачи, в которых использование линейных моделей необосновано, но и не удается предложить явную нелинейную модель. В таком случае строится модель вида

$$y(x) = f(x, \theta) = \sum_{j=1}^m \theta_j \phi_j(x_j),$$

где x — объект, x_j — признаки объекта, $\phi_j : \mathbb{R} \rightarrow \mathbb{R}$ — нелинейные в общем случае преобразования.

Задача состоит в том, чтобы одновременно подбирать коэффициенты модели θ_j и неизвестные преобразования ϕ_j .

Суть метода backfitting (метод настройки с возвращением) заключается в чередовании оптимизации коэффициентов θ_j при постоянных ϕ_j методами линейной регрессии, и оптимизации преобразований ϕ_j при постоянных коэффициентах θ_j .

Для минимизации используется сумма квадратов ошибок на всех объектах:

$$Q(\theta, \phi) = \sum_{i=1}^l (y(x_i) - y_i)^2 = \sum_{i=1}^l \left(\sum_{j=1}^m \theta_j \phi_j(X_{ij}) - y_i \right)^2$$

Здесь X — матрица признаков.

Алгоритм backfitting

$\phi_j(t) \equiv t$ — изначальное приближение линейными функциями
while (Q уменьшается) **do**

$\theta \leftarrow \operatorname{argmin}_{\theta} Q(\theta, \phi_j)$ — при фиксированных ϕ_1, \dots, ϕ_k

for $j = 1 \dots m$ **do**

$$r_i = y_i - \sum_{k \neq j} \theta_k \phi_k(X_{ik})$$

r_i — ошибка модели на i объекте, без учёта j признака

$$\phi_j \leftarrow \operatorname{argmin}_{\phi} \sum_{i=1}^l (\theta_j \phi(X_{ij}) - r_i)^2 \text{ — при } \theta_j = \text{const}$$

end for

end while

Оптимизация по коэффициентам $\theta \leftarrow \operatorname{argmin}_{\theta} Q(\theta, \phi_j)$ выполняется методами линейной регрессии, такими как стохастический градиентный спуск.

Оптимизация по функции ϕ_j выполняется одномерными методами, такими как ядерное сглаживание.

Варианты дальнейшего развития метода:

1. Во внутреннем цикле выбирать индексы j не в фиксированном порядке, а в первую очередь оптимизировать дающие наибольший вклад в Q .
2. Регуляризация Q по параметрам θ и по сложности функций ϕ .

Задача

Если матрица признаков X разреженная, с какой проблемой можно столкнуться, применяя алгоритм backfitting, и как её решить?

Решение: если каждая из функций ϕ_j вызывается лишь на небольшом наборе аргументов, то её значения на них могут быть почти независимы (если модель ϕ достаточно сложна, например, полином высокого порядка), и случится переобучение. Варианты решения: большая константа регуляризации по сложности функций ϕ ; ограничить ϕ более простым классом функций (например, только квадратичные функции вместо полиномов); расширить матрицу признаков несколькими простыми нелинейными преобразованиями ($\sin x, x^2 \dots$) и использовать методы линейной регрессии.

Метод наименьших квадратов с итеративным пересчётом весов (IRLS)

Метод наименьших квадратов с итеративным пересчётом весов (Iteratively Reweighted Least Squares) применяется для решения задач оптимизации. В частности, применение метода Ньютона-Рафсона к задаче *Логистической регрессии* сводится к **IRLS**.

Напомним постановку задачи линейной регрессии. Будем искать приближенное решение системы:

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1N} \\ \vdots & \vdots & \ddots & \\ a_{M1} & \dots & \dots & a_{MN} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_M \end{bmatrix} = \begin{bmatrix} y_1 \\ \vdots \\ y_M \end{bmatrix}$$

То же самое в матричных обозначениях:

$$\mathbf{A}\mathbf{x} = \mathbf{y}$$

Поставим задачу минимизировать норму вектора ошибки (невязки):

$$\mathbf{e} = \mathbf{A}\mathbf{x} - \mathbf{y}$$

В качестве нормы возьмем p -норму: $\|\mathbf{e}\|_p = (\sum_i |e_i|^p)^{1/p}$. Как известно, если методом наименьших квадратов можно аналитически найти решение, минимизирующее Евклидову норму вектора невязки (root-mean-squared error) $\sqrt{\mathbf{e}^T \mathbf{e}}$. Покажем как можно построить итеративный подход, использующий результаты для взвешенного метода наименьших квадратов, для нахождения оптимального решения для l_p нормы.

Взвешенные метод наименьших квадратов

Для построения IRLS нужно сперва вспомнить основные результаты взвешенной модификации МНК. Добавим в Евклидову норму веса для каждой компоненты вектора \mathbf{x} и будем минимизировать норму:

$$\|\mathbf{W}\mathbf{e}\|_2^2 = \sum_i w_i^2 e_i^2 = \mathbf{e}^T \mathbf{W}^T \mathbf{W} \mathbf{e}$$

$$\mathbf{W} = \text{diag}(w_1, w_2, \dots, w_M)$$

\mathbf{W} - диагональная матрица ненулевых весов. Легко видеть, что такое взвешивание соответствует линейному преобразованию растяжения с коэффициентами w_1, w_2, \dots, w_M . Для переопределенной системы решение с минимальной взвешенной нормой оказывается равным:

$$\mathbf{x} = [\mathbf{A}^T \mathbf{W}^T \mathbf{W} \mathbf{A}]^{-1} \mathbf{A}^T \mathbf{W}^T \mathbf{W} \mathbf{y}$$

Алгоритм IRLS

Поиск минимума $\|\mathbf{e}\|_p = (\sum_i |e_i|^p)^{1/p}$ сводится к итеративному алгоритму, где на каждом шаге применяется взвешенный МНК. Набор весов w_1, w_2, \dots, w_M пересчитывается на каждой итерации n .

$$\begin{aligned} \|e(n+1)\|_p &= \left(\sum_i w_i^2(n) |e_i(n)|^2 \right)^{1/2} \\ w_i(n) &= e_i(n)^{\frac{p-2}{2}} \end{aligned}$$

Начальные веса w_1, w_2, \dots, w_M берутся единичными, т.е. для первой итерации используется в качестве приближения используется стандартный МНК.

Задачи

Задача 1: WLS Получить оптимальное решение для переопределенной системы с взвешенной нормой.

Решение:

Вспомним результат обычного МНК:

$$\mathbf{Ax} = \mathbf{y}$$

$$\mathbf{x} = [\mathbf{X}^T \mathbf{X}]^{-1} \mathbf{X} \mathbf{y}$$

Домножая слева на \mathbf{W} получим: $\mathbf{W}\mathbf{Ax} = \mathbf{W}\mathbf{y}$. Получаем исходную задачу МНК с матрицей $\mathbf{W}\mathbf{A}$ и правой частью $\mathbf{W}\mathbf{y}$. Отсюда сразу получается ответ:

$$\mathbf{x} = [\mathbf{A}^T \mathbf{W}^T \mathbf{W} \mathbf{A}]^{-1} \mathbf{A}^T \mathbf{W}^T \mathbf{W} \mathbf{y}$$

Задача 2: Пример IRLS для l_1 Реализовать IRLS и применить его для нахождения линейной модели по набору точек $x, y: (1, 2), (2, 3), (4, 6)$. Применить реализованный метод и убедиться, чтобы итеративный метод сходится к $3/2$ для $p = 1$ (норма l_1) и к $32/21$ для $p = 2$.

Задача 3: Зависимость весов для l_p Обосновать выбор $w_i(n) = e_i(n)^{\frac{p-2}{2}}$, считая что итеративный метод сходится.

Задача 4: Логистическая регрессия как IRLS Показать, как применение метода Ньютона-Рафсона к логистической регрессии приводит к IRLS.

Решение:

см. раздел про логистическую регрессию

Глава 8
Обобщенные линейные модели

Глава 9

Нестандартные функции потерь

Нестандартные функции потерь. Метод наименьших модулей. Квантильная регрессия.

Квадратичная функция потерь обычно дает хорошие результаты, является удобной в использовании, а потому и применяется чаще всего. Но в некоторых ситуациях она все же неприменима, и приходится использовать нестандартные функции потерь.

Метод наименьших модулей.

Здесь и далее будем использовать следующие обозначения: ℓ - количество объектов в тренировочной выборке, x_i ($i \in \{1, \dots, \ell\}$) - вектор признаков, y_i ($i \in \{1, \dots, \ell\}$) - таргет, a - модель, $\varepsilon_i := a(x_i) - y_i$ ($i \in \{1, \dots, \ell\}$).

Для стандартной квадратичной функции потерь $\mathcal{L}(\varepsilon) = \varepsilon^2$ задача будет выглядеть так:

$$\frac{1}{\ell} \sum_{i=1}^{\ell} (a(x_i) - y_i)^2 \longrightarrow \min_a .$$

Заменим функцию потерь на $\mathcal{L}(\varepsilon) = |\varepsilon|$. Задача станет выглядеть так:

$$\frac{1}{\ell} \sum_{i=1}^{\ell} |a(x_i) - y_i| \longrightarrow \min_a .$$

Данный метод называется *методом наименьших модулей*.

В какой ситуации такой подход может оказаться полезным? Рассмотрим ситуацию, когда наша модель - это константа, то есть она вообще не зависит от признаков. Для квадратичной функции потерь получим:

$$\frac{1}{\ell} \sum_{i=1}^{\ell} (a - y_i)^2 \longrightarrow \min_a .$$

Здесь можно посчитать ответ аналитически, оптимальной константой a будет $a = \frac{1}{\ell} \sum_{i=1}^{\ell} y_i$, то есть среднее арифметическое таргетов. Это плохая оценка, если, например, в нашей выборке присутствуют выбросы, или распределение ошибок имеет тяжёлые «хвосты».

А в случае использования метода наименьших модулей задача будет выглядеть так:

$$\frac{1}{\ell} \sum_{i=1}^{\ell} |a - y_i| \longrightarrow \min_a .$$

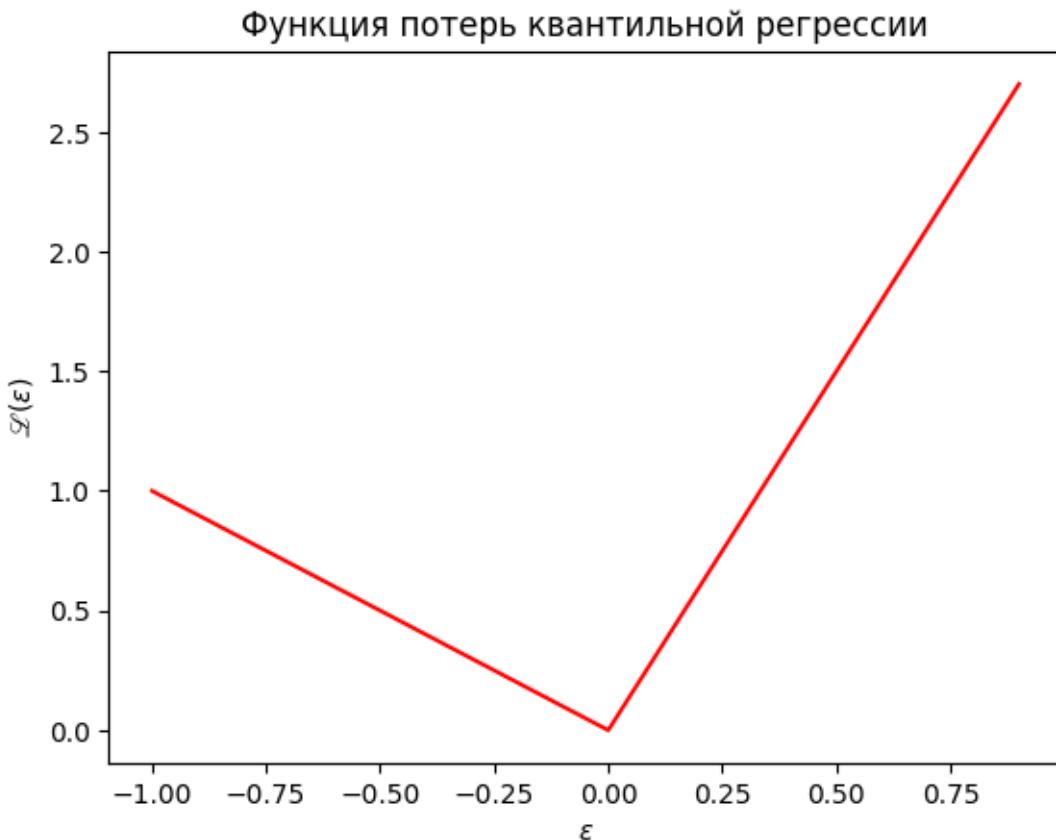
Здесь опять же можно посчитать ответ аналитически, оптимальным a будет $a = \text{median}\{y_1, \dots, y_\ell\} = y_{(\ell/2)}$ (серединный член вариационного ряда). Эта оценка хороша тем, что устойчива к выбросам, хорошо работает для распределений ошибок с тяжелыми «хвостами».

Таким образом, использование метода наименьших модулей в некоторых ситуациях помогает бороться с выбросами.

Квантильная регрессия.

Метод наименьших модулей можно обобщить. Давайте по-разному штрафовать отрицательные и положительные ошибки. Т.е. будем рассматривать функцию потерь вида

$$\mathcal{L}(\varepsilon) = \begin{cases} C_+ |\varepsilon|, & \varepsilon \geq 0, \\ C_- |\varepsilon|, & \varepsilon < 0. \end{cases}$$



Опять же рассмотрим случай, когда модель не зависит от признаков, то есть является константой:

$$\frac{1}{\ell} \sum_{i=1}^{\ell} \mathcal{L}(a - y_i) \longrightarrow \min_a.$$

Решение данной задачи опять же можно получить аналитически, оптимальным a будет $a = y_{(q)}$ (q -тый член вариационного ряда), где $q = \frac{\ell C_-}{C_- + C_+}$. Именно поэтому используемый метод называется *методом квантильной регрессии*.

Рассмотрим также один случай, когда квантильная регрессия оказывается хорошим вариантом с точки зрения решения задачи оптимизации, а именно случай линейной модели: $a(x) = \langle x, w \rangle$, где w - вектор весов.

Сделаем замену переменных $\varepsilon_i^+ = (a(x_i) - y_i)_+$, $\varepsilon_i^- = (a(x_i) - y_i)_-$. Тогда наша задача будет иметь вид

$$\begin{cases} \frac{1}{\ell} \sum_{i=1}^{\ell} C_+ \varepsilon_i^+ + C_- \varepsilon_i^- \longrightarrow \min_w, \\ \langle w, x_i \rangle - y_i = \varepsilon_i^+ - \varepsilon_i^-, i \in \{1, \dots, \ell\}. \end{cases}$$

Это задача линейного программирования, для решения которой существует масса способов.

Задачи.

Задача 1. Предположим, у Вас есть датасет с данными о жителях некоторой страны Южной или Центральной Африки, а также данные об их среднем ежедневном доходе, и Вы хотите научиться предсказывать этот доход по значениям рассматриваемых признаков. Полученная модель в последующем будет использоваться для составления плана по оказанию гуманитарной помощи населению. Что использовать предпочтительнее: квадратичную функцию потерь или функцию потерь квантильной регрессии? Почему? Если использовать предпочтительнее функцию потерь квантильной регрессии, то каким должно быть соотношение параметров C_+ и C_- и почему?

Решение. В условии задачи не зря указано, из какого региона у нас страна. Большая часть населения в ней, скорее всего, крайне бедна, при этом есть очень незначительное количество сверхбогатых людей, т.е., в нашей терминологии, выбросов. Поэтому функция потерь квантильной регрессии более предпочтительна, чем квадратичная функция потерь.

Теперь обратим внимание на то, как будет использована модель, чтобы понять, какая ошибка для нас наиболее страшна: положительная или отрицательная. Заметим, что если мы предсказали доход человека больше реального, т.е. ошибка положительна, то, скорее всего, на него будет выделено меньше помощи. Это кажется более страшным, чем ситуация, в которой мы предсказали доход человека меньше реального, и дали ему чуть больше помощи. Поэтому штрафовать положительные ошибки стоит сильнее, чем отрицательные, то есть стоит выбрать C_+ и C_- так, чтобы $C_+/C_- > 1$.

Задача 2. Предположим, что у в природе некоторый таргет действительно линейно зависит от вектора признаков, но вектор таргетов, который у нас есть, зашумлен ошибками, причем ошибки эти приходят из симметричного распределения Коши. Почему использование квадратичной функции потерь в данном случае будет крайне нежелательным? А если ошибки приходит из распределения $\mathcal{N}(a, \sigma^2)$, где $a \neq 0$?

Решение. Из теории вероятностей и математической статистики известно, что у распределения Коши тяжелые «хвосты», из-за чего у него даже нет матожидания. Именно поэтому квадратичная функция потерь, очень сильно штрафующая за большие ошибки, здесь не подходит. А вот метод наименьших модулей подходит лучше.

Также можно заметить, что квадратичная функция потерь одинаково штрафует положительные и отрицательные ошибки, то есть неявно подразумевается симметричность распределения. В общем случае это не так. Квантильная регрессия позволяет более гибко работать с несимметричными распределениями.

Задача 3. Предположим, что Вы тестируете новое лекарство на мышах. Вы хотите понять, какая доза лекарства оптимальна: уже вылечивает мышь, но все еще не дает негативных побочных эффектов. Известно, что каждый новый побочный эффект проявляется при превышении некоторого примерно одинакового для всех мышей порога передозировки и действует все сильнее и сильнее при дальнейшем превышении этого порога. Также известно, что оптимум не единственен, а достигается на некотором отрезке, длина которого вам тоже заранее известна. Наконец, ниже некоторого порога лекарство действует все хуже и хуже. У Вас есть некоторые данные о мышах, а также таргеты - максимальные оптимальные дозы лекарств. Предложите модификацию функции потерь квантильной регрессии, оптимальную для данной задачи.

Решение. Основная идея квантильной регрессии по сравнению с методом наименьших модулей состоит в том, что мы по-разному штрафуем положительные и отрицательные ошибки. Давайте разовьем эту идею, и будем по-разному штрафовать ошибки в зависимости от того, в какую часть числовой прямой мы попали.

Так, для нашей задачи, пусть новые побочные эффекты проявляются при превышении дозировки на a_1, \dots, a_n у.е., где $0 = a_1 < \dots < a_n$. А длина оптимального отрезка дозировки равна b . Тогда можно взять такую функцию потерь:

$$\mathcal{L}(\varepsilon) = \begin{cases} -v_b \varepsilon, & \varepsilon < -b, \\ 0, & \varepsilon \in [-b; a_1], \\ v_{a_1} \varepsilon, & \varepsilon \in (a_1; a_2], \\ (v_{a_1} + v_{a_2}) \varepsilon, & \varepsilon \in (a_2; a_3], \\ \vdots \\ (v_{a_1} + \dots + v_{a_{n-1}}) \varepsilon, & \varepsilon \in (a_{n-1}; a_n], \\ (v_{a_1} + \dots + v_{a_n}) \varepsilon, & \varepsilon > a_n, \end{cases}$$

где $v_b, v_{a_1}, \dots, v_{a_n}$ - скорости роста соответствующих проблем.

SVM-Regression

Регрессионные модели на основе метода опорных векторов (SVM-regression) зачастую используют особую функцию потерь, отличающуюся от стандартной квадратичной ошибки (MSE). В частности, классический SVM для регрессии использует ε -insensitive функцию потерь (ε -insensitive loss), которая игнорирует ошибки, величина которых меньше заранее заданного порога ε . Однако существуют и нестандартные функции потерь, которые могут быть полезны в специфических задачах, например, когда необходимо учитывать асимметрию ошибок, различные веса отдельных наблюдений или же более сложные, ориентированные на распределение отклонений, формулы.

Классическая ε -insensitive функция потерь: Для модели вида $f(x) = \langle w, x \rangle + b$ функция потерь определяется следующим образом:

$$L(\varepsilon) = \max(0, |y - f(x)| - \varepsilon).$$

Если $|y - f(x)| \leq \varepsilon$, то штраф равен нулю, что делает модель нечувствительной к небольшим отклонениям и позволяет контролировать сложность аппроксимации.

Нестандартные функции потерь могут иметь следующие особенности:

1. Асимметричные функции потерь: например, более сильное штрафование положительных отклонений, чем отрицательных, или наоборот. Это может быть полезно, если цена переоценки или недооценки целевой переменной различна.
2. Частично-кусочные функции: использование различных режимов штрафования в зависимости от величины отклонения, например, линейная область штрафа для малых ошибок и квадратичная — для больших.
3. Функции с весами для разных точек: если некоторые точки обучающей выборки важнее, можно ввести веса и штрафовать ошибки по более значимым точкам сильнее.
4. Нелинейные преобразования ошибки: например, логарифмическая или экспоненциальная функция потерь, которая меняет чувствительность модели к большим отклонениям.

Пример асимметричной ε -insensitive функции потерь:

$$L(\varepsilon, \alpha) = \begin{cases} 0, & |y - f(x)| \leq \varepsilon \\ \alpha(y - f(x)) - \varepsilon, & y - f(x) > \varepsilon \\ \frac{(y-f(x))}{\alpha} - \varepsilon, & f(x) - y > \varepsilon \end{cases}$$

где $\alpha > 1$ задаёт степень асимметрии штрафования. Такая функция потерь позволит модели сильнее реагировать на недостаточную оценку и слабее — на переоценку.

Задачи

Задача 1. Пусть у нас есть набор точек для регрессии $\{(x_i, y_i)\}_{i=1}^m$, и используется ε -insensitive функция потерь. Как будет влиять на итоговую модель выбор ε ? Что произойдёт с моделью при увеличении и при уменьшении ε ?

Решение При увеличении ε расширяется область, внутри которой ошибка не штрафуется. Это означает, что модель меньше старается подогнать точно каждую точку. Итоговая функция при этом может стать более гладкой и с меньшей чувствительностью к выбросам, но при этом с большой вероятностью систематической ошибки внутри расширенной области.

При уменьшении ε модель стремится к более точному описанию данных. При этом модель станет более чувствительной к шуму и выбросам.

Задача 2. Предположим, что мы хотим использовать асимметричную ε -нечувствительную функцию потерь. Запишите целевую функцию и ограничения для такой SVM-регрессии (линейный случай), если функция потерь равна:

$$L(\varepsilon) = \begin{cases} 0, & |y_i - f(x_i)| \leq \varepsilon \\ 2(y_i - f(x_i) - \varepsilon), & y_i - f(x_i) > \varepsilon \\ f(x_i) - y_i - \varepsilon, & f(x_i) - y_i > \varepsilon \end{cases}$$

Решение Введём неотрицательные переменные ξ_i^+ и ξ_i^- для верхних и нижних ошибок. Тогда ограничения на ошибки с учётом ε :

$$\begin{aligned} y_i - (w^T x_i + b) &\leq \varepsilon + \xi_i^+ \\ (w^T x_i + b) - y_i &\leq \varepsilon + \xi_i^- \end{aligned}$$

Целевая функция с учётом асимметрии штрафов:

$$\min_{w, b, \xi_i^+, \xi_i^-} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n (2\xi_i^+ + \xi_i^-)$$

Таким образом, мы увеличили штраф в два раза для положительных отклонений (когда модель переоценивает значение, $f(x_i) > y_i + \varepsilon$) по сравнению с отрицательными отклонениями.

Задача 3. Пусть у нас есть три варианта функций потерь для SVM-регрессии над одним и тем же набором данных $\{(x_i, y_i)\}_{i=1}^m$:

1. Стандартная ε -insensitive функция потерь.
2. Модифицированная ε -insensitive функция потерь, где штраф начинается не с 0, а с небольшой линейной части:

$$L(\varepsilon) = \max(0, |y - f(x)| - \varepsilon) + \delta(y - f(x)),$$

где $\delta > 0$ малый коэффициент.

3. Huber-функция потерь с параметром δ .

Предположим, что мы обучили три SVM-регрессии, каждый с одной из этих функций потерь, при одинаковых параметрах C . Укажите, в каких случаях может быть предпочтительна Huber-функция по сравнению с ε -insensitive, и для чего может пригодиться модификация с добавлением линейной части $\delta(y - f(x))$.

Решение

1. Huber-функция потерь предпочтительна в ситуациях, когда в данных присутствуют выбросы или редкие, но крупные отклонения. Она сочетает в себе квадратичную форму для небольших ошибок (что способствует более мягкой подгонке и «дружелюбно» относится к шуму) и линейную для больших отклонений (не позволяя слишком сильно штрафовать крупные ошибки и тем самым уменьшая чувствительность к выбросам).
2. Модифицированная ε -insensitive функция потерь с дополнительной линейной частью может быть полезна, когда нам важно учесть даже малые ошибки, но мы хотим сохранить идею ε -зоны. Добавление $\delta(y - f(x))$ означает, что даже если ошибка меньше ε , мы всё же учитываем некий штраф (хотя и небольшой). Это может помочь, когда не хочется полностью игнорировать малые отклонения, но при этом сохранять определённую «зону толерантности» к маленьким ошибкам, чтобы не переобучаться на шумах.

Энтропийные функции потерь. Перекрёстная энтропия.

В задачах классификации (бинарной или многоклассовой) очень распространено использование энтропийных функций потерь, в первую очередь перекрёстной энтропии. Эти функции потерь тесно связаны с понятиями теории вероятностей и информационной теории, в частности с понятием дивергенции Кульбака–Лейблера и энтропией Шеннона.

Основная идея: мы хотим не просто предсказать «класс», но и оценить распределение вероятностей классов. Пусть у нас есть:

- Набор объектов: $(x_1, y_1), \dots, (x_\ell, y_\ell)$, где x_i – вектор признаков, а y_i – соответствующий истинный класс объекта (для простоты – номер класса или унитарный вектор-«one-hot»).
- Модель a , выдающая оценку распределения вероятностей по классам для входа x_i . Обозначим предсказанную моделью вероятность класса k как $p_{ik} = a_k(x_i)$, где $\sum_k p_{ik} = 1$.

Бинарная классификация

В случае двух классов, обозначим целевой класс как $y_i \in \{0, 1\}$ и предсказанную моделью вероятность класса 1 для объекта i как $p_i = a_1(x_i)$. Тогда функция потерь на одном объекте может быть записана как (перекрёстная энтропия для бинарного случая):

$$\mathcal{L}(y_i, p_i) = -[y_i \log(p_i) + (1 - y_i) \log(1 - p_i)].$$

Эта функция потерь равна отрицательному логарифму правдоподобия при условии, что y_i берётся из Бернулиевского распределения с параметром p_i . Минимизация этой функции потерь равносильна максимизации правдоподобия.

Многоклассовая классификация

Пусть класс y_i закодирован в one-hot формате как вектор $Y_i = (y_{i1}, \dots, y_{iK})$, где $y_{ik} = 1$, если объект относится к классу k , и 0 в противном случае. Предсказание модели есть вероятностный вектор $P_i = (p_{i1}, \dots, p_{iK})$. Тогда перекрёстная энтропия:

$$\mathcal{L}(Y_i, P_i) = - \sum_{k=1}^K y_{ik} \log(p_{ik}).$$

Оптимизируя по параметрам модели, мы стремимся сделать предсказанное распределение вероятностей P_i как можно более «острым» и совпадающим с истинным распределением мишени Y_i (которое в обучающих данных детерминировано и задаётся one-hot вектором).

Связь с KL-дивергенцией

Перекрёстная энтропия между истинным распределением Y_i и предсказанным P_i связана с дивергенцией Кульбака–Лейблера:

$$H(Y_i, P_i) = H(Y_i) + D_{\text{KL}}(Y_i \| P_i),$$

где $H(Y_i)$ – энтропия истинного распределения (константа в рамках оптимизации), а $D_{\text{KL}}(Y_i \| P_i)$ – дивергенция Кульбака–Лейблера, которая всегда неотрицательна. Минимизируя перекрёстную энтропию, мы минимизируем D_{KL} , что приводит к более точному приближению истинного распределения предсказанным.

Модификации

- **Взвешенная перекрёстная энтропия:** для решения проблем несбалансированных классов можно вводить веса w_k для каждого класса:

$$\mathcal{L}(Y_i, P_i) = - \sum_{k=1}^K w_k y_{ik} \log(p_{ik}).$$

- **Label smoothing:** позволяет сгладить «жесткие» one-hot лейблы, заменяя значение 1 на $1 - \alpha$ и распределяя оставшуюся массу α равномерно по остальным классам. Это снижает перенакрой и делает модель более устойчивой.

Задачи

Задача 1 (Бинарная классификация и правдоподобие) Пусть у нас есть задача бинарной классификации: $y_i \in \{0, 1\}$, и модель $a(x_i)$, дающая оценку вероятности класса 1 для x_i . Предположим, что истинное распределение метки y_i для данного x_i – это бернуллиевское распределение с параметром $p_i = a(x_i)$. Покажите, что минимизация средней перекрёстной энтропии

$$\frac{1}{\ell} \sum_{i=1}^{\ell} [-y_i \log(p_i) - (1 - y_i) \log(1 - p_i)]$$

эквивалентна максимизации правдоподобия выборки. Объясните, почему это даёт статистически обоснованную функцию потерь.

Решение. Правдоподобие выборки при условии независимости объектов есть:

$$L = \prod_{i=1}^{\ell} p_i^{y_i} (1 - p_i)^{1 - y_i}.$$

Взяв логарифм, получим:

$$\log L = \sum_{i=1}^{\ell} [y_i \log(p_i) + (1 - y_i) \log(1 - p_i)].$$

Максимизация $\log L$ по параметрам модели эквивалентна минимизации

$$-\log L = -\sum_{i=1}^{\ell} [y_i \log(p_i) + (1 - y_i) \log(1 - p_i)],$$

что есть сумма (или среднее) перекрёстной энтропии. Таким образом, минимизация перекрёстной энтропии совпадает с максимизацией правдоподобия. Это придаёт функции потерь статистическое обоснование: мы получаем состоятельную оценку параметров модели при верном специфицировании вероятностной модели.

Задача 2 (Небаланс классов) Предположим, что у вас есть задача многоклассовой классификации с сильно несбалансированными классами. Один из классов встречается существенно реже других. Объясните, как можно модифицировать функцию перекрёстной энтропии, чтобы придать более высокий штраф за неправильную классификацию редкого класса. Предложите конкретную формулу и аргументируйте её использование.

Решение. Стандартная перекрёстная энтропия для многоклассовой задачи:

$$\mathcal{L}(Y_i, P_i) = - \sum_{k=1}^K y_{ik} \log(p_{ik}).$$

Если класс r встречается редко, мы можем ввести для него повышающий вес $w_r > 1$. Тогда функция потерь модифицируется так:

$$\mathcal{L}(Y_i, P_i) = - \sum_{k=1}^K w_k y_{ik} \log(p_{ik}),$$

где $w_k = 1$ для всех «частых» классов, а $w_r > 1$ для редкого класса. Это увеличивает штраф за ошибку на редком классе и стимулирует модель уделять ему больше внимания. В итоге модель будет «стараться» точнее предсказывать редкий класс, жертвуя немного точностью на других, более частых классах, что зачастую улучшает общую полезность модели при решении практических задач с несбалансированными данными.

Задача 3 (Label Smoothing) Предположим, что у вас есть задача многоклассовой классификации с K классами. Истинные метки заданы в виде one-hot векторов. Вы подозреваете, что модель может слишком «уверенно» переобучаться, пытаясь подогнать вероятности к очень жёсткому распределению (где истинный класс имеет вероятность 1, а остальные 0). Предложите модификацию перекрёстной энтропии (label smoothing), объясните, в чём она заключается и как именно изменится функция потерь.

Решение. Идея label smoothing состоит в том, чтобы заменить истинный one-hot вектор Y_i на более «сглаженный» вектор Y'_i . Пусть α – небольшой положительный параметр сглаживания (например, 0.1). Тогда:

- Для истинного класса c_i объекта i мы присваиваем $y'_{ic_i} = 1 - \alpha$.
- Для всех остальных классов $k \neq c_i$ мы присваиваем $y'_{ik} = \frac{\alpha}{K-1}$.

Таким образом, истинный вектор $(0, \dots, 0, 1, 0, \dots, 0)$ превращается в вектор, где истинный класс имеет вероятность чуть меньше 1, а остальные классы получают небольшую ненулевую вероятность.

Новая функция потерь будет выглядеть так:

$$\mathcal{L}(Y'_i, P_i) = - \sum_{k=1}^K y'_{ik} \log(p_{ik}).$$

Поскольку Y'_i теперь не является «жёстким» one-hot вектором, модель не будет излишне стремиться предсказывать для истинного класса вероятность ровно 1, что снижает риск переобучения и делает распределение прогнозов более гладким и устойчивым.

Функции потерь для задач несбалансированной классификации

Введение

В задачах классификации, где данные имеют несбалансированное распределение классов, выбор функции потерь играет критическую роль в обучении моделей. Несбалансированные классы могут привести к тому, что стандартные функции потерь, такие как кросс-энтропия, будут недостаточно чувствительны к меньшинственным классам, что может негативно сказаться на качестве предсказаний. В этом параграфе будут рассмотрены подходы к созданию функций потерь, которые учитывают диспропорцию между классами и помогают модели лучше справляться с трудными для классификации примерами.

Введем следующие обозначения:

X – пространство признаков;

$x \in X$ – объект;

w – параметры модели;

C – количество классов, которые могут принимать значения от 1 до C ;

$p = p(x, w) \in [0, 1]^C$ – вектор предсказанных моделью вероятностей;

$y = y(x) \in \{1, \dots, C\}$ – истинный класс объекта x ;

$\mathcal{L}(p, y) = \mathcal{L}(p(x, w), y(x))$ – функция потерь.

Focal Loss

Focal Loss является динамически масштабируемой модификацией Cross Entropy, которая снижает вклад хорошо классифицированных примеров в задачах с сильно несбалансированными классами, сосредоточивая обучение на трудных примерах.

Cross Entropy Введем Focal Loss (FL) с рассмотрения Cross Entropy (CE) для бинарной классификации

$$\text{CE}(p, y) = \begin{cases} -\log(p) & \text{if } y = 1 \\ -\log(1 - p) & \text{otherwise,} \end{cases}$$

где p – предсказанная вероятность класса 1, а y – метка истины (0 или 1).

Для удобства обозначим

$$p_t = \begin{cases} p & \text{if } y = 1 \\ 1 - p & \text{otherwise,} \end{cases}$$

тогда перепишем

$$\text{CE}(p, y) = \text{CE}(p_t) = -\log(p_t).$$

Balanced CE с весами $\alpha \in [0, 1]$ для класса 1 и $1 - \alpha$ для класса 0 будет записываться как

$$\text{CE}(p_t) = -\alpha_t \log(p_t).$$

Эксперименты показывают, что большой дисбаланс классов подавляет функцию потерь CE. Легко классифицируемые объекты составляют большую часть потери и доминируют в градиенте.

Математическая формулировка Предлагается изменить форму функции потерь, чтобы уменьшить влияние легких примеров и, тем самым, сосредоточить обучение на трудных. Для этого добавим модулирующий фактор $(1 - p_t)^\gamma$ к CE с настраиваемым параметром фокусировки γ . Таким образом, определим фокусную потерю как:

$$\text{FL}(p_t) = -\alpha_t(1 - p_t)^\gamma \log(p_t).$$

Перепишем, раскрыв обозначения:

$$\text{FL}(p, y) = -\alpha y(1 - p)^\gamma \log(p) - (1 - \alpha)(1 - y)p^\gamma \log(1 - p)$$

Аналогично можно определить Focal Loss для категориальной классификации

$$\text{FL}(p, y) = -\alpha_y(1 - p_y)^\gamma \log(p_y),$$

где

$\alpha_i \geq 0$ – вес класса i , компенсирующий несбалансированность классов;

$\gamma \geq 0$ – фокусирующий параметр, который контролирует степень влияния трудных примеров.

Class Balanced Loss

Выборка данных как случайное покрытие Пусть S – множество всех возможных данных в пространстве признаков заданного класса. Будем предполагать, что S имеет объем, равный $N \geq 1$, а каждый объект представляет собой подмножество S с объемом 1. Выборку можно рассматривать как случайное покрытие этими объектами. Ожидаемый объем выбранных точек растет с их увеличением числа и ограничен N .

Определение. Эффективное число выборки E_n – это ожидаемый объем выбранных n точек.

Вычисление этого объема сложно и зависит от формы выборки и размерности пространства. Для упрощения не будем учитывать частичное перекрытие: новая точка может быть либо внутри ранее выбранных данных (с вероятностью p), либо снаружи (с вероятностью $1 - p$).

Предложение. $E_n = (1 - \beta^n)/(1 - \beta)$, где $\beta = (N - 1)/N$

Математическая формулировка Class Balanced Loss предлагает решить проблему несбалансированных классов, добавив в функцию потерь вес, обратно пропорциональный эффективному числу объектов соответствующего класса.

Пусть для объекта x с классом y предсказаны вероятности $p \in [0, 1]^C$. Обозначим функцию потерь $\mathcal{L}(p, y)$. Предложенное эффективное число для класса $i \in \{1, \dots, C\}$ есть $E_{n_i} = (1 - \beta_i^{n_i})/(1 - \beta_i)$, где n_i — число объектов класса i из выборки, $\beta_i = (N_i - 1)/N_i$.

Без дополнительной информации о данных для каждого класса трудно эмпирически найти набор хороших гиперпараметров N_i для всех классов. Поэтому на практике мы предполагаем, что N_i зависит только от набора данных, и положим $N_i = N$, $\beta_i = \beta = (N - 1)/N$ для всех i .

Тогда Class Balanced Loss определяется следующим образом:

$$\mathcal{L}_{CB}(p, y) = \frac{1}{E_{n_y}} \mathcal{L}(p, y) = \frac{1 - \beta}{1 - \beta^{n_y}} \mathcal{L}(p, y),$$

где $\beta \in [0, 1)$ — гиперпараметр, который контролирует степень балансировки.

Замечание. Class Balanced Loss можно использовать в сочетании с различными функциями потерь, например, с Focal Loss

$$FL_{CB}(p, y) = -\alpha_y \frac{1 - \beta}{1 - \beta^{n_y}} (1 - p_y)^\gamma \log(p_y),$$

Задачи

Задача 1. Покажите, что $FL(p, y)$ является выпуклой относительно первого (прогностического) аргумента p для всех $\gamma \geq 0$ как для задачи бинарной классификации, так и категориальной.

Решение. Начнем с бинарной классификации. Рассмотрим отдельно особые случаи $\gamma = 0$ и $\gamma = 1$. При $\gamma = 0$ FL совпадает с CE, которая является выпуклой. При $\gamma = 1$:

$$\frac{\partial^2(-(1-p)\log(p))}{\partial p^2} = \frac{1+p}{p^2} > 0 \quad \forall p \in (0, 1).$$

Теперь рассматриваем случаи, когда $\gamma \notin \{0, 1\}$:

$$\frac{\partial^2(-(1-p)^{\gamma-1}\log(p))}{\partial p^2} = \frac{\gamma(1-p)^{\gamma-1}}{p} - \gamma(\gamma-1)(1-p)^{\gamma-2}\log(p) - \frac{-\gamma(1-p)^{\gamma-1}p - (1-p)^\gamma}{p^2}.$$

Перепишем в виде квадратного трехчлена относительно γ :

$$\gamma^2(-\log(p)(1-p)^{\gamma-2}) + \gamma \left(\frac{2(1-p)^{\gamma-1}}{p} + (1-p)^{\gamma-2}\log(p) \right) + \frac{(1-p)^\gamma}{p^2}.$$

$\forall p \in (0, 1)$ все коэффициенты этого квадратного трехчлена положительны, поэтому этот трехчлен тоже положителен $\forall \gamma > 0$.

Для категориальной классификации покажем положительную определенность гессиана. Его диагональные элементы положительны, поскольку совпадают с производными выше, недиагональные равны 0, так как FL зависит от предсказанной вероятности только истинного класса:

$$\frac{\partial^2 \text{FL}}{\partial p_i \partial p_j} = 0 \quad \forall i \neq j.$$

Матрицы с такой структурой всегда положительны определены, Q.E.D.

Задача 2. В пункте «Выборка данных как случайное покрытие» доказать предложение

$$E_n = \frac{1 - \beta^n}{1 - \beta}.$$

Решение. Доказательство по индукции. Очевидно, что при $E_1 = 1$, так как при выборе одного объекта нет перекрытий, поэтому база индукции выполняется:

$$E_1 = \frac{1 - \beta^1}{1 - \beta} = 1.$$

Пусть ожидаемый объем выборки на шаге $n - 1$ равным E_{n-1} , тогда вероятность новой точки быть покрытой $p = E_{n-1}/N$. Таким образом, ожидаемый объем выборки на n -м шаге

$$E_n = pE_{n-1} + (1 - p)(E_{n-1} + 1) = 1 + \frac{N - 1}{N}E_{n-1}.$$

Пользуясь предположением индукции для $n - 1$ шага, запишем

$$E_n = 1 + \beta \frac{1 - \beta^{n-1}}{1 - \beta} = \frac{1 - \beta + \beta + \beta^n}{1 - \beta} = \frac{1 - \beta^n}{1 - \beta},$$

переход доказан, Q.E.D.

Задача 3. В пункте «Выборка данных как случайное покрытие» покажите, что если N велико, то можно считать, что каждый объект уникален, а при $N = 1$ – класс представим единственным объектом.

Решение. Заметим, что $\beta = 0$ при $N = 1$ и $\beta \rightarrow 1$ при $N \rightarrow \infty$. Тогда

$$E_n = \begin{cases} \frac{1 - 0^n}{1 - 0} = 1 & \text{if } N = 1 \\ \lim_{\beta \rightarrow 1} \frac{1 - \beta^n}{1 - \beta} = \lim_{\beta \rightarrow 1} \frac{-n\beta^{n-1}}{-1} = n & \text{if } N \rightarrow \infty \end{cases}.$$

При большом N эффективное число выборки равно ее размеру. Это значит, что нет перекрытия данных, следовательно каждый объект уникален. С другой стороны, если $N = 1$, $E_n = 1$, что означает существования единственного прототипа, так что все данные в этом классе могут быть представлены им посредством аугментации данных, преобразований и т.п., Q.E.D.

Глава 10

Критерии выбора моделей

Качество классификации: Precision, Recall

В задаче классификации **precision** (точность) и **recall** (полнота) являются ключевыми метриками для оценки качества предсказания, особенно в задачах с несбалансированными классами.

Определения

Рассмотрим бинарную классификацию, то есть объект может быть либо положительным (positive), либо отрицательным (negative), и определим:

- TP (*True Positives*) — количество объектов, правильно классифицированных как положительные.
- FP (*False Positives*) — количество объектов, ошибочно классифицированных как положительные.
- FN (*False Negatives*) — количество объектов, ошибочно классифицированных как отрицательные.
- TN (*True Negatives*) — количество объектов, правильно классифицированных как отрицательные.

На основе этих величин вычисляются:

1. Precision:

$$\text{Precision} = \frac{TP}{TP + FP}.$$

Precision показывает долю истинно положительных объектов среди всех объектов, классифицированных как положительные.

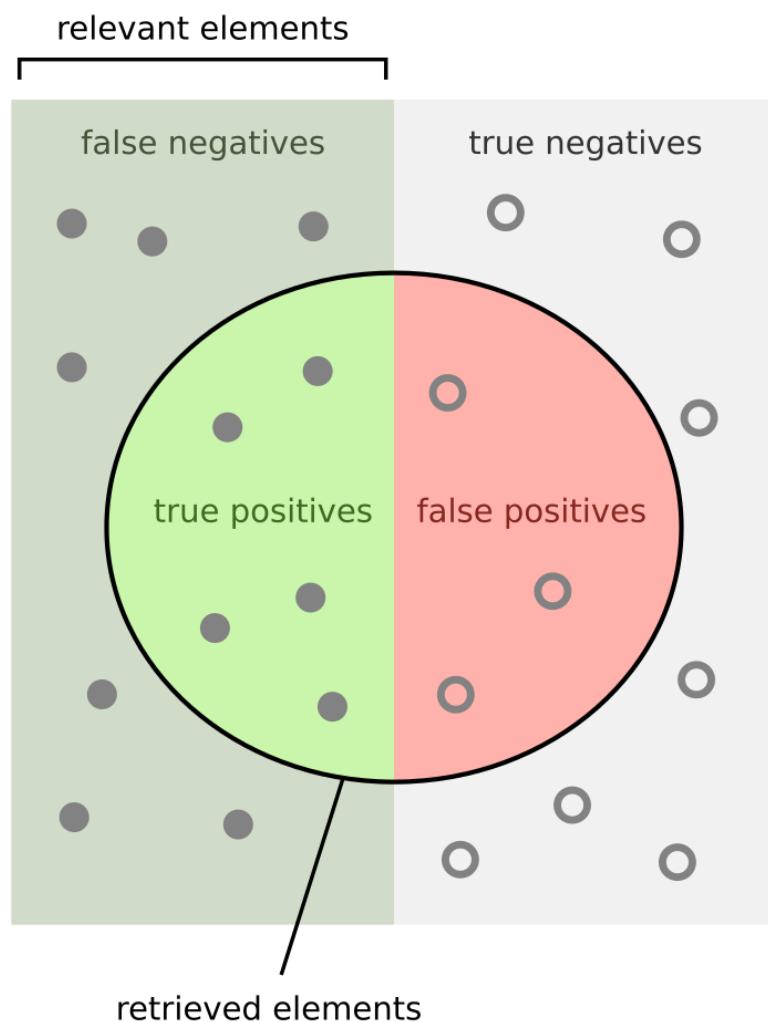
2. Recall:

$$\text{Recall} = \frac{TP}{TP + FN}.$$

Recall показывает долю истинно положительных объектов среди всех реально положительных объектов.

Интуитивное объяснение

- **Precision:** Насколько «точен» алгоритм, когда он говорит, что объект положительный? Если precision высокое, значит, ложные срабатывания (FP) минимальны.
- **Recall:** Насколько хорошо алгоритм находит все положительные объекты? Если recall высокое, значит, пропущенные положительные объекты (FN) минимальны.



How many retrieved items are relevant?

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

How many relevant items are retrieved?

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

Пример

Классический пример использования метрик precision и recall - задача поиска спама на почте. В этом случае спам - положительная категория. Пусть у нас есть

100 писем, из которых 40 писем — спам, 60 писем — не спам.

Алгоритм классифицировал 50 писем как спам, из которых 30 писем действительно оказались спамом, а остальные 20 писем были ошибочно классифицированы как спам. Вычислим в этом случае precision и recall:

- Precision:

$$\text{Precision} = \frac{TP}{TP + FP} = \frac{30}{30 + 20} = 0.6.$$

- Recall:

$$\text{Recall} = \frac{TP}{TP + FN} = \frac{30}{30 + 10} = 0.75.$$

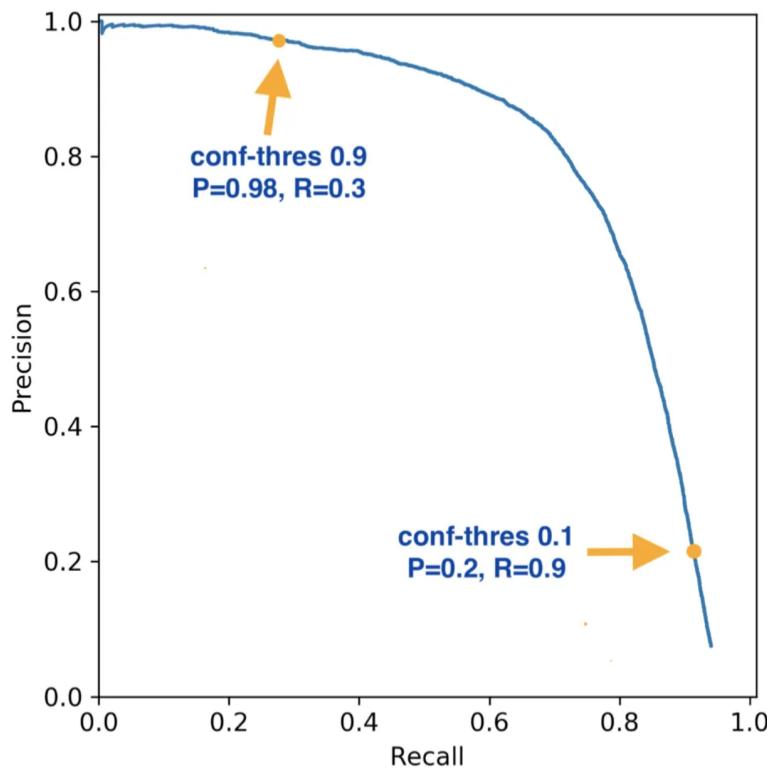
Баланс между precision и recall

Модель для каждого объекта на входе генерирует какое-то число на выходе. В простейшем варианте объект классифицируется как положительный, если это число больше некого выставленного порога, и как отрицательный в обратном случае. Увеличивая порог классификации, мы снижает количество False Positive объектов, потому precision увеличивается. При этом количество "незамеченных" моделью положительных объектов тоже вырастет, поэтому recall снизится. При уменьшении порога будет наблюдаться обратный эффект. Обычно стараются добиться компромиссного значения, при котором precision и recall оба принимают удовлетворительные значения. В некоторых случаях одна из метрик важнее другой:

- Детекция спам-рассылок. В этом случае мы чаще всего не хотим помечать важные письма, как спам. Поэтому нужно снизить False Positive - важнее precision.
- Первичного выявление заболевания. Мы не хотим пропустить пациентов, которые на самом деле больны, только потому что модель сказала обратное. Поэтому важно снизить False Negative - в этом случае важнее recall.

Интуитивное объяснение

Если при получении положительного ответа от модели мы предпринимаем какое-либо действие, то precision важнее, когда действие обходится дорого, а recall важнее, когда бездействие обходится дорого. В примерах выше: помещение важного письма в папку "спам"(действие) может привести к финансовым потерям, а пропуск реального спама во "входящие"(бездействие) лишь заставит человека сделать это вручную. С другой стороны, при ложноположительном диагнозе человек пройдёт дополнительные анализы (действие), а ложноотрицательный может стоить ему жизни (бездействие).



Для того, чтобы изобразить баланс между двумя метриками, строят precision-recall кривую. Точки на ней соответствуют разным значениям порога.

F-метрика (дополнительно)

Часто для оценки общего качества модели используется метрика F -мера, которая является гармоническим средним между precision и recall с параметром β :

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{Precision} \cdot \text{Recall}}{\beta^2 \cdot \text{Precision} + \text{Recall}}.$$

Из формулы понятно, что β определяет, насколько recall важнее по сравнению с precision. Наиболее часто используется F_1 -мера:

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}.$$

Задачи

Задача 1: Простейшая модель

Как будет выглядеть precision-recall кривая у простейшей модели, которая для любого объекта делает positive предсказание с вероятностью $1 - t$ (при $t = 1$ все предсказания отрицательные, при $t = 0.5$ - половина)? Как в этом случае и с

использованием метрик precision и recall подбирать параметр t , чтобы добиться лучшей работы модели?

Ответ:

В координатах (*recall*, *precision*) - отрезок с концами в $(0, \alpha)$ и $(1, \alpha)$, где α - доля положительных объектов в выборке (горизонтальный отрезок). Объяснение состоит в том, что доля TP равна $\alpha(1-t)$, доля $TP+FN$ равна α , а доля $TP+FP$ равна $(1-t)$. Таким образом, значения precision и recall не зависят от параметра t , то есть его изменение не изменит качество модели в этих метриках.

Задача 2: Дисбаланс классов

Какая из метрик – precision или recall – будет больше в случае сильного дисбаланса классов на тестовой выборке (рассмотреть оба случая), если известно, что модель обучалась на сбалансированном датасете?

Ответ:

Если положительных объектов значительно меньше, чем отрицательных, то recall будет больше. Это связано с тем, что сбалансированная модель в этом случае будет допускать много False Positive ошибок. При обратном дисбалансе precision будет больше из-за большого количества False Negative ошибок.

Задача 3: Мошеннические транзакции

Финансовая компания использует алгоритм для выявления мошеннических транзакций. Из 10,000 проверенных транзакций 500 транзакций являются мошенническими, 9,500 транзакций являются легитимными.

Алгоритм определил 600 транзакций как мошеннические, из которых 400 действительно оказались мошенническими.

1. Вычислите F_1 -меру.
2. Насколько может измениться F_1 -мера, если алгоритм пометит ещё 50 транзакций как мошеннические, в зависимости от того, являются они на самом деле мошенническими или нет?

Решение: Найдём precision и recall.

-

$$\text{precision} = \frac{400}{400 + 200} = \frac{400}{600} \approx 0.667.$$

-

$$\text{recall} = \frac{400}{400 + 100} = \frac{400}{500} = 0.8.$$

-

$$F_1\text{-score} = 2 \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} = 2 \frac{0.667 \cdot 0.8}{0.667 + 0.8} \approx 0.722$$

Обозначим за α долю тех новых помеченных транзакций, которые на самом деле являются мошенническими ($0 \leq \alpha \leq 1$). Понятно, что TP увеличится на 50α , FN уменьшится на 50α , а $TP + FP$ станет равным 650. Посчитаем precision, recall, изменение F_1 -score в общем случае:

- $$precision = \frac{400 + 50\alpha}{650}.$$
- $$recall = \frac{400 + 50\alpha}{500}.$$
- $$F_1\text{-score} = \frac{16 + 2\alpha}{23} \approx 0.696 + 0.087\alpha$$
- $$\Delta F_1\text{-score} \approx 0.696 + 0.087\alpha - 0.722 = -0.026 + 0.087\alpha$$

Таким образом, F_1 -мера уменьшится на -0.026 если все новые помеченные транзакции на самом деле легитимные, увеличится на 0.061, если они все на самом деле мошеннические. Заметим, что F_1 -мера останется неизменной, если $\alpha = \frac{1}{3}$, то есть изначальная доля ложноположительных среди всех помеченных.

Глава 11

Методы отбора признаков

Качество классификации

Рассмотрим задачу бинарной классификации с обучающей выборкой $D = \{(x_i, y_i)\}_{i=1}^n$, где $x_i \in \mathbb{R}^d$ — вектор признаков, $y_i \in \{0, 1\}$ — бинарная переменная. Мы хотим построить модель $f : \mathbb{R}^d \rightarrow \{0, 1\}$, которая принимает на вход вектор признаков и выдает предсказание класса. Есть следующие способы измерять качество модели:

Accuracy

Precision

Recall

F_β score

ROC AUC

Для каждого объекта из выборки мы имеем 4 варианта развития событий:

TP — True Positive, классификатор предсказал 1, верное значение тоже 1

FP — False Positive, классификатор предсказал 1, верное значение 0

TN — True Negative, классификатор предсказал 0, верное значение тоже 0

FN — False Negative, классификатор предсказал 0, верное значение 1

Ясно, что мы хотим видеть как можно больше TP и TN и как можно меньше FP и FN.

Accuracy — это доля правильных ответов классификатора.

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN} = \frac{1}{n} \sum_{i=1}^n \mathbb{I}(y_i = f(x_i))$$

Достаточно банальная метрика, которая не учитывает дисбаланса классов, но в качестве базового варианта подходит отлично.

Precision — это доля объектов, которые классификатор предсказал как 1, среди всех объектов, которые он предсказал как 1.

$$\text{Precision} = \frac{TP}{TP + FP}$$

Recall — это доля объектов, которые классификатор предсказал как 1, среди всех объектов, которые действительно равны 1.

$$\text{Recall} = \frac{TP}{TP + FN}$$

Precision и Recall — базовые метрики, но по отдельности их использовать нельзя, поэтому придумали F_β score.

F_β score — это взвешенное среднее гармоническое precision и recall.

$$F_\beta = (1 + \beta^2) \frac{\text{precision} \cdot \text{recall}}{\beta^2 \cdot \text{precision} + \text{recall}}$$

TPR — это доля объектов, которые классификатор предсказал как 1, среди всех объектов, которые действительно равны 1.

$$TPR = \frac{TP}{TP + FN}$$

FPR — это доля объектов, которые классификатор предсказал как 1, среди всех объектов, которые действительно равны 0.

$$FPR = \frac{FP}{FP + TN}$$

Обычно, когда мы решаем задачу бинарной классификации, то мы не предсказываем явный класс, а предсказываем какое-то значение, и чем больше это значение, тем больше вероятность того, что объект принадлежит к классу 1. Следовательно, мы можем устанавливать пороговое значение, которое будет определять, к какому классу отнести объект. При увеличении порога отсечения мы увеличиваем количество объектов, которые отнесены к классу 0, и уменьшаем количество объектов, которые отнесены к классу 1. Заметим, что также при увеличении порога отсечения TPR и FPR будут уменьшаться.

ROC кривая — это кривая, которая показывает зависимость TPR от FPR при изменении порога отсечения. То есть по оси x откладывается FPR , а по оси y — TPR .

Строго говоря, ROC кривая — это множество точек вида $(FPR(t), TPR(t))$, где t — пороговое значение вероятности.

ROC-AUC — это площадь под ROC кривой.

Задача 1

Мы хотим построить модель бинарной классификации, которая будет по некоторому описанию пациента предсказывать, болен ли он раком. В случае, если наша модель скажет, что пациент болен раком, то мы отправим его на дополнительное обследование, что обойдется пациенту в некоторую неприятную, но не критичную сумму денег. Если же наша модель скажет, что пациент здоров, то мы ничего не будем предпринимать. Мы хотим выбрать метрику качества для нашей модели из следующего списка: Accuracy, Precision, Recall, F_β score, что нам лучше всего подойдет? Если мы хотим выбрать F_β score, то какие значения β нам лучше всего подойдут?

Решение

Accuracy — не лучший вариант, так как ошибка FP и FN в нашем случае не равнозначны. Не отправить больного на дополнительное обследование значительно хуже, чем отправить здорового пациента на обследование. Precision и Recall — достаточно плохие метрики, ведь есть очень простая модель, которая даст 100% Precision — просто предсказывать, что все пациенты больны, аналогичная проблема и с Recall. F_β score — отличный вариант, ведь с помощью β мы можем выбирать, какое предпочтение мы отдаём Precision, а какое Recall. В нашем случае, мы хотим, чтобы Precision был предпочтительнее Recall, поэтому β должно быть меньше 1. Если мы для себя решили, что смерть человека стоит как 10 дообследований, то стоит брать $\beta = \frac{1}{10}$.

Задача 2

Предположим, что мы измеряем качество модели с помощью ROC-AUC. Допустим, что у нас есть две модели, чьи ROC кривые выглядят следующим образом: Как эти модели упорядочить по качеству?

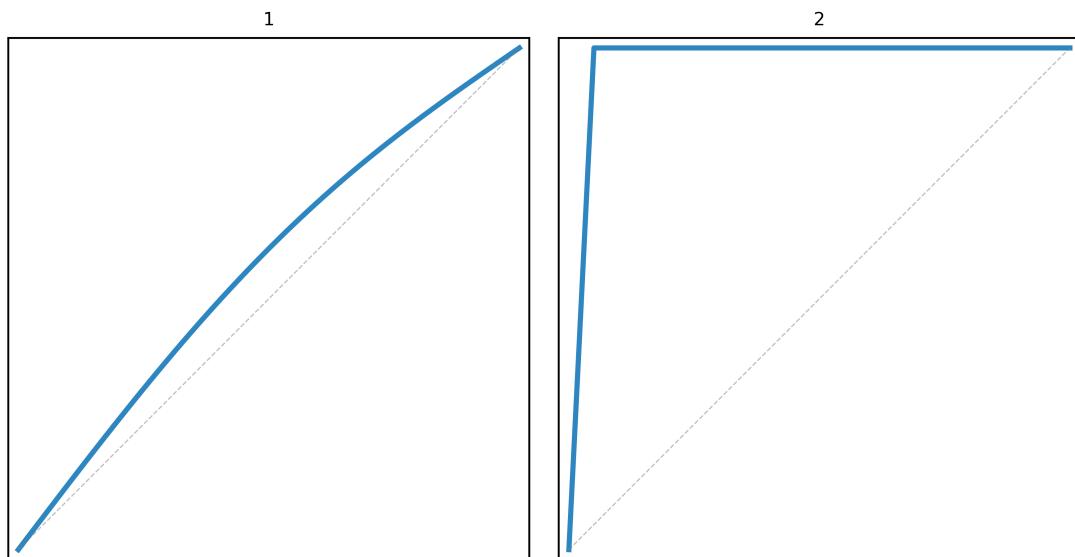


Рис. 11.1. Две ROC кривые

Решение

Первая модель самая худшая, ведь ее ROC кривая максимально близка к диагонали, значит она имеет минимальную ROC-AUC. Это плохо, но что по смыслу означает близкая к диагонали ROC кривая?

Это означает, что для любого порога отсечения t имеем примерно $TPR(t) = FPR(t)$. Что означает $TPR(t) = FPR(t)$? Это означает, что модель просто случайным образом с вероятностью $TPR(t) = FPR(t)$ предсказывает 1 класс. Вторая модель лучшая, но почему?

А что означает график, который близок к уголку, как на рисунке 2? Это означает, что мы можем взять такой порог отсечения t , что почти выполнено $TPR(t) = 1$ и $FPR(t) = 0$. А это идеальная модель, которая никогда не ошибается.

Заметим, что чем больше площадь под ROC кривой, тем ближе эта кривая к уголку, а значит тем лучше модель.

Задача 3

Продолжим измерять качество моделей с помощью ROC-AUC. Допустим, что у нас есть две модели, чьи ROC кривые выглядят следующим образом: Как

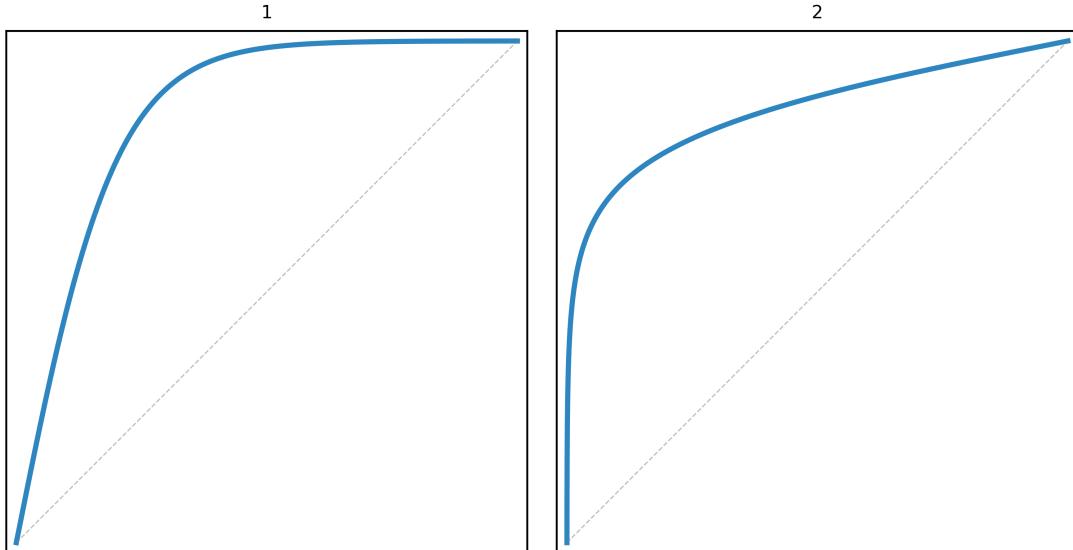


Рис. 11.2. Две ROC кривые

эти модели упорядочить по качеству? А как выбрать порог отсечения? В чем преимущество ROC перед F_β score?

Решение

В данном случае, обе ROC кривые имеют одинаковую площадь под графиком, а значит одинаковую ROC-AUC, поэтому просто посмотреть на метрику ROC-AUC нельзя. На самом деле, однозначного ответа нет, всё зависит от конкретной задачи.

Первая кривая быстро достигает высокого значения TPR , при не самом высоком значении FPR .

Это означает, что выбрав порог отсечения t , при котором $TPR(t)$ будет близок к 1, а $FPR(t)$ будет близок к 0.5, мы получим модель, которая угадывает почти всех, кто попадает в 1 класс, но при этом не сильно много ошибается на классе 0. Такая модель отлично подойдет в первой задаче, где мы хотим отгадать почти всех больных, но при этом не сильно много отправлять на дополнительное обследование здоровых людей.

На втором графике ситуация обратная, мы можем выбрать порог отсечения t , при котором $TPR(t)$ будет близок к 0.5, а $FPR(t)$ будет близок к 1. Такая модель будет полезна в случае, когда мы хотим минимизировать количество ошибок на классе 0, но при этом не сильно много ошибаться на классе 1. В контексте первой задачи, это означает, что мы не хотим тратить лишние деньги на дополнительное обследование здоровых людей даже ценой смерти больных, которых мы не отправляем на дополнительное обследование.

Исходя из этих примеров, становится понятно, как стоит выбирать порог отсечения. По сути его выбор означает, насколько мы готовы пожертвовать ошибками на предсказаниях одного класса, чтобы минимизировать ошибки на предсказаниях другого класса. Это такой аналог β в F_β score, только β мы выбирали до измерения метрики, а в случае ROC кривой мы выбираем порог отсечения глядя на график и имея какую-то дополнительную информацию о том, как работает модель при разных порогах. Это может быть очень полезно, ведь далеко не всегда мы готовы определиться с балансом ошибок на классах перед тем, как начать решать задачу.

Методы отбора признаков: полный перебор, алгоритм Add

Работа с признаками включает два основных подхода:

1. Генерация новых признаков;
2. Отбор существующих признаков.

Генерация новых признаков

Генерация новых признаков подразумевает создание таких признаков, которые представляют собой преобразованные версии исходных данных или полностью новые параметры, полученные из имеющихся данных. Примеры методов генерации признаков включают:

- Построение статистик на основе существующих признаков (например, среднее, стандартное отклонение, медиана и т.д.);
- Использование методов понижения размерности, таких как анализ главных компонент (PCA);

- Применение нейросетей, где скрытые слои (за исключением последнего) могут рассматриваться как новое, более информативное признаковое пространство.

В этой главе мы сосредоточимся на отборе существующих признаков, рассмотрев два наиболее популярных метода: полный перебор и жадный алгоритм Add.

Полный перебор

Метод полного перебора представляет собой один из самых простых и технически точных подходов к отбору признаков. Суть метода заключается в оценке качества модели на всех возможных комбинациях признаков и выборе той комбинации, которая обеспечивает наилучшую метрику. Таким образом, полный перебор **гарантирует нахождение оптимального набора признаков**, так как проверяет все возможные варианты.

Основное преимущество полного перебора заключается в его точности: при достаточных вычислительных ресурсах метод всегда найдёт лучшее решение. Тем не менее, с увеличением числа признаков n , количество возможных подмножеств растёт экспоненциально (2^n). Это делает полный перебор крайне ресурсоёмким для наборов данных с большим n . Например, уже при $n = 20$ потребуется рассчитать $2^{20} = 1,048,576$ комбинаций.

Метод полного перебора чаще применяется:

- Для небольших наборов признаков, где перебор возможен за приемлемое время;
- В задачах, где цена ошибки высока (например, в медицинских исследованиях);
- Когда требуется хорошая интерпретируемость и точность подмножеств признаков.

На больших данных иногда применяются модификации полного перебора, такие как частичный перебор, где анализируется только часть комбинаций (например, сначала отбираются наиболее значимые признаки с помощью другого метода, а затем производится перебор подмножеств только из этих признаков).

Жадный алгоритм Add

Алгоритм Add (жадный подход) используется гораздо чаще, чем полный перебор, благодаря лучшей производительности на больших данных и более простой интерпретации. Суть метода заключается в пошаговом добавлении признаков в модель. На каждом шаге из оставшихся признаков выбирается тот, который обеспечивает наибольшее улучшение метрики модели. Таким образом, итоговый

набор строится итеративно, начиная с пустого множества и поочередно добавляя наиболее значимые признаки.

Жадный алгоритм не гарантирует нахождения глобального оптимума, так как выбирает локально лучшие решения на каждом из шагов. Однако его высокая скорость и способность работать с большими наборами признаков делают его востребованным для многих задач.

Очевидно, что из общего числа признаков n должен оставаться некоторый k -подмножество наиболее значимых признаков. Критерии остановки алгоритма Add могут быть следующими:

- **Фиксированное количество признаков k :** Алгоритм останавливается, как только выбрано заданное количество признаков k . *Плюсы*: простота реализации и возможность заранее контролировать размерность данных. *Минусы*: фиксированное значение k может оказаться избыточным в одной задаче или недостаточным в другой. Чтобы минимизировать неопределенность, значения параметра k можно подбирать по сетке с помощью кросс-валидации.
- **Остановка при отсутствии улучшения метрики:** На каждом шаге проверяется, насколько новая комбинация признаков улучшает метрику модели (например, R^2 , MAPE, RMSE). Если добавление нового признака не приводит к значимому улучшению (или ухудшает метрику), алгоритм прекращает выполнение. *Плюсы*: гибкость метода, так как он адаптируется к конкретной задаче. *Минусы*: возможно преждевременное завершение, если данные содержат шум.

Метод Add применяется, когда требуется построить интерпретируемую модель с допустимым уровнем качества за ограниченное время. Например, алгоритм широко используется в финансах для выделения ключевых факторов прогнозирования прибыли или моделирования рисков, а также в обработке медицинских данных.

Преимущества метода Add:

- Высокая скорость работы и масштабируемость для большого числа признаков;
- Возможность построить достаточно простую и интерпретируемую модель.

Недостатки метода Add:

- Чувствительность к входным данным: выбросы и избыточные признаки могут искажать отбор на ранних этапах;
- Отсутствие глобальной оптимальности: алгоритм может игнорировать комбинации признаков, т.к. учитывает только влияние признаков по отдельности.

Задачи

Задача 1. Есть набор данных с $N = 10000$ признаков, и задача состоит в отборе важных признаков. Отбор происходит с помощью метода Add, но на каждом шаге анализ всех оставшихся признаков (выбор из 10000, затем из 9999 и т.д.) занимает слишком много времени.

Как можно было бы улучшить этот метод?

Ответ 1. Можно разделить все признаки случайным образом на блоки (например, по 100 признаков). В таком случае хорошей идеей будет выбрать лучший признак из каждого блока на предварительном этапе (т.е. жадный Add выполняется на подмножествах сначала локально в блоках), а затем продолжить работу с меньшим числом представителей от каждого блоков. Если блоки и так большого размера, то можно воспользоваться этой идеей рекурсивно для каждого подблока.

Задача 2. Рассмотрим использование жадного алгоритма Add для выбора признаков из набора данных с 5 признаками (A, B, C, D, E) в задаче классификации.

График перезаписи последовательного роста метрики R^2 при добавлении выбранных признаков показан ниже:

Итерация	Добавленный признак	R^2
1	A	0.45
2	B	0.53
3	C	0.58

Между тем, известна модель из полного перебора, где признаковая комбинация (A, D, E) даёт $R^2 = 0.67$.

Почему метод Add мог не найти эту комбинацию? Как можно было бы исправить эту ситуацию и повысить вероятность нахождения глобального оптимума?

Ответ 2. Жадный алгоритм Add выбирает признак на каждом шаге, который локально улучшает метрику максимально. Однако комбинация (A, D, E) могла быть значимой только в сочетании (то есть признаки дают прирост метрики только в комбинации), а по отдельности эти фичи дают не такой хороший прирост к метрике, как B и C . Add не проверяет взаимодействие признаков, поэтому выбор локально наилучших признаков (A, B, C) мог игнорировать комбинацию, которая глобально лучше.

Как с этим можно бороться?

- Проверять метрику при добавлении не одного признака, а пары/тройки/произвольного числа m признаков.
- Использовать модификацию жадного алгоритма с дополнительными "прыжками то есть, например, после 3 итераций Add можно сделать перезапуск среди лучших комбинаций 2 признаков.

Вопрос 3. Есть набор данных с 20 признаками. Известно, что 10 из них абсолютно не связана с целевой переменной, т.е. являются шумовыми. Однако не известно, какие именно из них являются шумовыми. Для отбора признаков был применен жадный алгоритм Add.

Предположим, что на первых итерациях жадный алгоритм случайно выбрал несколько шумовых признаков, так как они хорошо улучшили метрику на обучающей выборке.

Как это повлияет на работу алгоритма Add в следующих итерациях? Какие подходы вы бы предложили, чтобы минимизировать вероятность включения шумовых признаков на ранних этапах?

Ответ 3.

1. Как это повлияет на дальнейшую работу алгоритма?

Жадный алгоритм фиксирует признаки и рассматривает следующие шаги на основе текущего набора. Если шумовые признаки остаются в модели, они могут "вытеснять" более значимые признаки в дальнейших итерациях. Итоговый результат может быть хуже, так как модель становится менее интерпретируемой и более склонной к переобучению на шум.

2. Как можно минимизировать влияние шума на обучение?

Для этого стоит перед алгоритмом применить другие фильтрационные методы для удаления явно нерелевантных признаков, например, убрать признаки, плохо коррелирующие с целевой переменной.

Также для этого можно попробовать применить кросс-валидацию при оценке каждого нового признака, чтобы убедиться, что он действительно улучшает модель (а не только улучшает метрику на обучающей выборке).

Методы преобразования категориальных признаков

Категориальные признаки (или категориальные переменные) — это тип данных, который представляет собой категории или группы. В отличие от количественных признаков, которые принимают числовые значения, категориальные признаки описывают качественные характеристики.

- Номинальные признаки – эти признаки представляют собой категории, которые не имеют никакого порядка (город, где проживает человек; цвет машины).

- Порядковые признаки – эти признаки также представляют категории, но в отличие от номинальных, они имеют естественный порядок (уровень образования; степень удовлетворенности).

Такие признаки невозможно напрямую использовать в моделях машинного обучения, поскольку они работают с числами. Для решения этой проблемы есть различные методы.

One-hot (dummy) encoding

One-hot encoding — это метод кодирования категориальных переменных, который преобразует каждую категорию в бинарный вектор, где только один элемент равен 1 (представляет категорию), а все остальные элементы равны 0.

Преимущества:

- Он легко реализуется, а данные легко интерпретируются.
- Хорошо совместим почти со всеми алгоритмами.
- Подходит для изначально неупорядоченных данных.

Недостатки:

- В случае большого количества категорий сильно увеличивает размерность, может привести к переобучению.
- Полученные признаки сильно разрежены, неэффективно по памяти и скорости вычислений.
- Не подходит для упорядоченных или взаимосвязанных данных.

Effect (treatment, deviation) encoding

Effect encoding — это метод кодирования категориальных переменных, который представляет каждую категорию в виде разности между средним значением целевой переменной и средними значениями для каждой категории. Этот метод позволяет учитывать информацию о влиянии каждой категории на целевую переменную.

Преимущества:

- Сохраняет информацию о влиянии категории на целевую переменную.
- Не увеличивает размерность пространства признаков.
- Создает порядок между категориями, основываясь на значении целевой переменной.

Недостатки:

- Не так прост для интерпретации, как one-hot encoding.
- Может создать многократную коллинеарность между некоторыми категориальными признаками.

Label (ordinal) encoding

Label encoding — это метод кодирования категориальных переменных, при котором каждой категории присваивается уникальное целочисленное значение.

Преимущества:

- Прост в реализации.
- Не увеличивает размерность пространства признаков.
- Сохраняет информацию порядковых признаков.
- Хорошо совместим с алгоритмами, не зависящими от расстояния между значениями (деревья решений).

Недостатки:

- Создает ложный порядок в номинальных признаках.
- Плохо совместим с линейными моделями.
- Создает ложные расстояния между категориями.

Count encoding

Count encoding — это метод кодирования категориальных переменных, при котором каждой категории присваивается количество её появлений в наборе данных.

Преимущества:

- Прост в реализации.
- Сохраняет информацию о частоте.
- Не увеличивает размерность пространства признаков.

Недостатки:

- Не сохраняет информацию порядковых признаков.
- Может создать коллинеарность для разных категорий с близкими частотами.
- Создает ложные расстояния между категориями.

Задача 1

В случае использования one-hot encoding для категориального признака с N уникальными значениями, мы получим N новых признаков. Однако любой признак выражается через остальные, поскольку единица всегда стоит только в одном из них. То есть мы получили мультиколлинеарность, которая может ухудшить качества модели! Каким образом можно решить эту проблему?

Решение

Самым простым решением будет просто удалить любой из полученных признаков. Мы избавимся от мультиколлинеарности и сохраним всю информацию о выборке.

Задача 2

Проведите преобразование категориальных признаков по методу effective encoding:

Пробег, тыс. км	Цвет	Год выпуска	Тип кузова	Стоимость, у.е.
100	Красный	2015	Хэтчбэк	10
10	Зеленый	2019	Седан	17
17	Синий	2022	Седан	18
150	Красный	2019	Универсал	15
30	Красный	2023	Хэтчбэк	20
174	Синий	2017	Хэтчбэк	12
89	Зеленый	2017	Хэтчбэк	14

Решение

Для каждого цвета вычислим средние значения стоимости:

- Красный: $r = \frac{10+15+20}{3} = 15$.
- Зеленый: $g = \frac{17+14}{2} = 15.5$.
- Синий: $b = \frac{18+9}{2} = 13.5$.

Тогда среднее для всех признаков:

$$avg = \frac{15 + 15.5 + 13.5}{3} = 14.66.$$

Значение нового признака, например, для красного цвета:

$$r - avg = 0.33.$$

Пробег, тыс. км	Цвет	Год выпуска	Тип кузова	Цвет (eff)	Тип кузова (eff)	Стоимость, у.е.
100	Красный	2015	Хэтчбэк	0,33	-2,00	10
10	Зеленый	2019	Седан	0,83	2,25	17
17	Синий	2022	Седан	-1,16	2,25	18
150	Красный	2019	Универсал	0,33	-0,25	15
30	Красный	2023	Хэтчбэк	0,33	-2,00	20
174	Синий	2014	Хэтчбэк	-1,16	-2,00	9
89	Зеленый	2017	Хэтчбэк	0,83	-2,00	14

Задача 3

Предложите методы обработки пропущенных значений признаков. Подсказка: можно основываться на некоторых методах преобразования категориальных признаков.

Решение

Некоторые из возможных вариантов:

- Добавление отдельного бинарного признака: есть/нет значение у исходного признака.
- Вычисление среднего по всем объектам с пропущенным значением исходного признака.
- Задание произвольных, не совпадающих с остальными, значений для каждого пропущенного значения исходного признака.
- Вычисление частоты пропущенных признаков.

Глава 12

Логические методы классификации

Бинаризация признаков

Бинаризация признаков – это процесс преобразования исходных признаков в бинарные переменные, которые принимают значения 0 или 1.

Бинаризация количественных признаков

Для признака $f : X \rightarrow D_f$, где D_f – множество возможных значений признака, бинаризация заключается в создании предикатов, проверяющих выполнение определённых условий. Эти предикаты позволяют разбить множество значений признака на подмножества, которые можно использовать в логических моделях.

В зависимости от типа признака, бинаризация осуществляется следующим образом:

- **Номинальный признак** (f принимает конечное множество значений, без упорядоченности):

$$\begin{aligned}\beta(x) &= [f(x) = d], \quad d \in D_f; \\ \beta(x) &:= [f(x) \in D'], \quad D' \subset D_f.\end{aligned}$$

- **Порядковый или количественный признак** (f принимает значения, между которыми можно определить порядок):

$$\beta(x) = [f(x) \leq d], \quad d \in D_f;$$

$$\beta(x) = [d \leq f(x) \leq d'], \quad d, d' \in D_f, d < d'.$$

Для количественных признаков ($f : X \rightarrow \mathbb{R}$) важно выбирать такие пороговые значения d , которые разделяют выборку на значимые группы. Например,

$$d_i = \frac{f^{(i)} + f^{(i+1)}}{2}, \quad f^{(i)} \neq f^{(i+1)}, i = 1, \dots, \ell - 1,$$

где $f^{(1)} \leq f^{(2)} \leq \dots \leq f^{(\ell)}$ – упорядоченные значения признака. (См. рис)

Такими способами можно получить много разных предикатов. Мы хотим выбрать из них самые "лучшие" (в каком-либо смысле). Для этого разобьем диапазон значений признака на зоны.

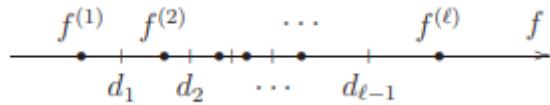


Рис. 12.1. Вариационный ряд значений признака $f(x)$ и пороги d_i

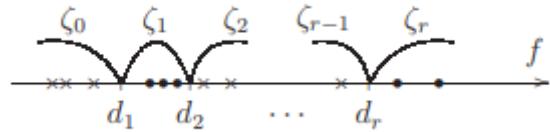


Рис. 12.2. Начальное разбиение на зоны

Разбиение диапазона значений признака на зоны

Каждая зона определяется бинарным предикатом:

$$\begin{aligned}\zeta_0(x) &= [f(x) < d_1], \\ \zeta_s(x) &= [d_s \leq f(x) < d_{s+1}], \quad s = 1, \dots, r-1, \\ \zeta_r(x) &= [d_r \leq f(x)].\end{aligned}$$

Способы разбиения:

- Жадная максимизация информативности путем слияний
- Разбиение на равномощные подвыборки
- Разбиение по равномерной сетке "удобных" значений (например, с минимальным числом значащих цифр)
- Объединение нескольких разбиений

Жадный алгоритм слияния зон

Алгоритм начинает с разбиения на "мелкие" зоны. Пороги проходят между всеми соседними парами точек, принадлежащих *разным* классам, т. к. расстановка порогов между точками одного класса приведет только к уменьшению информативности зон. Далее зоны укрупняются путём слияния *троек* соседних зон. Зоны сливаются до тех пор, пока информативность некоторой слитой зоны превышает информативность исходных зон, либо пока не будет получено заданное количество зон r . Каждый раз сливается тройка, дающая наибольший выигрыш в информативности.

Вход:

- $f(x)$ — признак;
- $c \in Y$ — выделенный класс;
- $X^\ell = \{(x_i, y_i)\}_{i=1}^\ell$ — выборка, упорядоченная по возрастанию $f(x_i)$;
- r — желаемое количество зон;
- δ_0 — порог слияния зон (по умолчанию $\delta_0 = 0$).

Выход:

$D = \{d_1, \dots, d_n\}$ — строго возрастающая последовательность порогов;

1. $D := \emptyset$;
2. **для всех** $i = 2, \dots, \ell$:
 - **если** $f(x_{i-1}) \neq f(x_i)$ и $[y_{i-1} = c] \neq [y_i = c]$ **то**
 - добавить новый порог $d := \frac{f(x_{i-1}) + f(x_i)}{2}$ в конец последовательности D ;
3. **повторять**
 - (a) **для всех** $d_i \in D, i = 1, \dots, |D| - 1$:
 - вычислить выигрыш от слияния тройки соседних зон $\zeta_{i-1}, \zeta_i, \zeta_{i+1}$:
$$\delta_i := I_c(\zeta_{i-1} \cup \zeta_i \cup \zeta_{i+1}) - \max\{I_c(\zeta_{i-1}), I_c(\zeta_i), I_c(\zeta_{i+1})\};$$
 - (b) найти тройку зон, для которой слияние наиболее выгодно:

$$i := \arg \max \delta_i;$$
 - (c) **если** $\delta_i > \delta_0$ **то**
 - слить зоны $\zeta_{i-1}, \zeta_i, \zeta_{i+1}$, удалить пороги d_i и d_{i+1} из последовательности D ;
4. **пока** $|D| > r + 1$.

Задачи

Задача 1 Предположим, вы владелец интернет-магазина, и у вас есть данные о стоимости товаров и их популярности (популярен — 1, непопулярен — 0). Для анализа спроса вы хотите разбить товары на ценовые зоны, чтобы лучше понять поведение покупателей.

Данные представлены в таблице:

№ товара	Цена товара (\$)	Популярность y
1	10	1
2	12	1
3	15	0
4	17	1
5	20	0
6	23	0
7	25	1

Как алгоритм слияния зон первично разбьёт выборку на ценовые зоны?

Решение Рассчитаем пороги:

$$\begin{aligned}d_1 &= \frac{12 + 15}{2} = 13.5, \\d_2 &= \frac{15 + 17}{2} = 16.0, \\d_3 &= \frac{17 + 20}{2} = 18.5, \\d_4 &= \frac{23 + 25}{2} = 24.0\end{aligned}$$

На основе рассчитанных порогов получаем зоны:

$$\begin{aligned}\zeta_0(x) &= [\text{Цена} < 13.5], \\ \zeta_1(x) &= [13.5 \leq \text{Цена} < 16.0], \\ \zeta_2(x) &= [16.0 \leq \text{Цена} < 18.5], \\ \zeta_3(x) &= [18.5 \leq \text{Цена} < 24.0], \\ \zeta_4(x) &= [\text{Цена} \geq 24.0].\end{aligned}$$

Задача 2 Будут ли разбиения диапазона меняться в зависимости от класса, относительно которого они производятся? Как изменить алгоритм для получения "универсального" разбиения, учитывающего сразу все классы?

Решение Да, будут, т. к. информативность зависит от класса. Нужно заменить критерий информативности многоклассовым критерием.

Задача 3 Какую сложность имеет алгоритм слияния зон? Как можно его ускорить?

Решение Этот алгоритм имеет трудоёмкость $O(l^2)$. Его можно заметно ускорить, если на каждой итерации сливать не одну тройку зон, а τl троек с достаточно большим выигрышем δI_i , при условии, что они не перекрываются. В этом случае трудоёмкость составляет $O(l/\sqrt{\tau})$.

Взвешенное голосование правил

Допустим, имеется консилиум экспертов, каждый член которого может допустить ошибку. Процедура голосования — это способ повышения качества принимаемых решений, при котором ошибки отдельных экспертов компенсируют друг друга.

Ранее принцип голосования применялся для построения композиций из произвольных алгоритмов классификации. Теперь рассмотрим композиции, состоящие из логических закономерностей.

Принцип голосования

Пусть для каждого класса $c \in Y$ построено множество логических закономерностей (правил), специализирующихся на различении объектов данного класса:

$$R_c = \{\varphi_{tc} : X \rightarrow \{0, 1\} \mid t = 1, \dots, T_c\}$$

Считается, что если $\varphi_{tc}(x) = 1$, то правило φ_{tc} относит объект $x \in X$ к классу c . Если же $\varphi_{tc}(x) = 0$, то правило воздерживается от классификации объекта x .

Алгоритм простого голосования (simple voting) подсчитывает долю правил в наборах R_c , относящих объект x к каждому из классов:

$$\Gamma_c(x) = \frac{1}{T_c} \sum_{t=1}^{T_c} \varphi_{tc}(x), \quad c \in Y,$$

и относит объект x к тому классу, за который подана наибольшая доля голосов:

$$a(x) = \arg \max_{c \in Y} \Gamma_c(x).$$

Если максимум достигается одновременно на нескольких классах, выбирается тот, для которого цена ошибки меньше.

Нормирующий множитель $\frac{1}{T_c}$ вводится для того, чтобы наборы с большим числом правил не перетягивали объекты в свой класс.

Алгоритм взвешенного голосования

Алгоритм взвешенного голосования (weighted voting, WV) действует более тонко, учитывая, что правила могут иметь различную ценность. Каждому правилу φ_{tc} приписывается вес $\alpha_{tc} \geq 0$, и при голосовании берется взвешенная сумма голосов:

$$\Gamma_c(x) = \sum_{t=1}^{T_c} \alpha_{tc} \varphi_{tc}(x), \quad \alpha_{tc} > 0.$$

Веса нормируются на единицу:

$$\sum_{t=1}^{T_c} \alpha_{tc} = 1, \quad \forall c \in Y.$$

Поэтому функцию $\Gamma_c(x)$ называют также выпуклой комбинацией правил $\varphi_1, \dots, \varphi_{T_c}$. Очевидно, простое голосование является частным случаем взвешенного, когда веса одинаковы и равны $\frac{1}{T_c}$.

На первый взгляд, вес правила должен определяться его информативностью. Однако, важно также учитывать, насколько данное правило уникально. Если имеется 10 хороших, но одинаковых (или почти одинаковых) правил, их суммарный вес должен быть сравним с весом столь же хорошего правила, не похожего на все остальные. Таким образом, веса должны учитывать не только ценность правил, но и их различность.

Простой общий подход к настройке весов заключается в том, чтобы сначала найти набор правил $\{\varphi_{tc}(x)\}$, затем принять их за новые (бинарные) признаки и построить в этом новом признаковом пространстве линейную разделяющую поверхность (кусочно-линейную, если $|Y| > 2$). Для этого можно использовать логистическую регрессию, однослойный персептрон или метод опорных векторов. Существуют и другие подходы. Например, в разделе 1.5.4 будет рассмотрен метод бустинга, в котором правила настраиваются последовательно, и для каждого правила сразу вычисляется его вес.

Проблема диверсификации правил

Голосующие правила должны быть существенно различны, иначе они будут бесполезны для классификации. Продолжая аналогию с консилиумом, заметим, что нет никакого смысла держать в консилиуме эксперта А, если он регулярно подсматривает решения у эксперта В.

Приведем простое теоретико-вероятностное обоснование принципа диверсификации, или повышения различности (diversity) правил [14]. Пусть X — вероятностное пространство, множество ответов Y конечно. Введем случайную величину $M(x)$, равную перевесу голосов в пользу правильного класса; её называют также отступом (margin) объекта x от границы классов:

$$M(x) = \Gamma_c(x) - \Gamma_{\bar{c}}(x), \quad \Gamma_{\bar{c}}(x) = \max_{y \in Y \setminus \{c\}} \Gamma_y(x), \quad c = y^*(x).$$

Если отступ положителен ($M(x) > 0$), то алгоритм голосования правильно классифицирует объект x . Предположим, что в среднем наш алгоритм классифицирует хотя бы немного лучше, чем наугад: $E[M] > 0$. Тогда можно оценить вероятность ошибки по неравенству Чебышева:

$$P\{M < 0\} \leq P\{|E[M] - M| > E[M]\} \leq \frac{D_M}{(E[M])^2}.$$

Отсюда вывод: для уменьшения вероятности ошибки необходимо максимизировать ожидание перевеса голосов $E[M]$ и минимизировать его дисперсию D_M . Для выполнения этих условий каждый объект должен выделяться примерно одинаковым числом правил. Обычно ни одно из правил не выделяет класс целиком, поэтому правила должны быть существенно различны, то есть выделять существенно различные подмножества объектов.

Неплохая эвристика, усиливающая различия между правилами и позволяющая равномернее выделять объекты обучения, используется в алгоритме CORAL [12]. Сначала для фиксированного класса $c \in Y$ строится покрывающий набор правил точно так, как это делалось для решающих списков. Затем строится второй покрывающий набор, но при этом запрещается использовать признаки, часто входившие в закономерности первого набора. Поэтому второй набор неминуемо окажется отличным от первого. Затем запрещаются признаки, часто входившие в оба набора, и строится третий набор. И так далее, для каждого класса $c \in Y$.

Отказы от классификации

Возможны ситуации, когда ни одно из правил не выделяет классифицируемый объект x . Тогда алгоритм должен либо отказываться от классификации, либо относить объект к классу, имеющему наименьшую цену ошибки. Отказ алгоритма означает, что данный объект является нетипичным, не подпадающим ни под одну из ранее обнаруженных закономерностей. Вообще, обнаружение нетипичности (novelty detection) принято считать отдельным видом задач обучения по прецедентам, наряду с классификацией и кластеризацией. Способность алгоритмов отказываться от классификации нетипичных объектов во многих приложениях является скорее преимуществом, чем недостатком. В то же время, число отказов не должно быть слишком большим.

Итак, при построении алгоритмов взвешенного голосования правил возникает четыре основных вопроса:

- Как построить много правил по одной и той же выборке?
- Как избежать повторов и построения почти одинаковых правил?
- Как избежать появления непокрытых объектов и обеспечить равномерное покрытие всей выборки правилами?
- Как определять веса правил при взвешенном голосовании?

Рассмотрим, как эти проблемы решаются в известных алгоритмах в следующих параграфах.

Задачи

Задача 1: Алгоритм простого голосования

Допустим, для класса $c \in Y$ существуют 10 правил, которые используют данные для классификации. Каждое правило возвращает 1 или 0 для объекта x . Если для объекта x правила 3 и 7 верно классифицируют объект как класс c , а остальные возвращают 0, как будет рассчитана доля голосов для класса c ?

Решение: Для класса c доля голосов рассчитывается как сумма всех правил, которые классифицируют объект как класс c , делённая на общее количество правил. Если 10 правил, то:

$$\Gamma_c(x) = \frac{1}{10} (1 + 0 + 0 + 0 + 0 + 0 + 1 + 0 + 0 + 0) = \frac{2}{10} = 0.2$$

Таким образом, для объекта x доля голосов для класса c составит 0.2.

Задача 2: Проблема с весами в алгоритме взвешенного голосования

В алгоритме взвешенного голосования веса для каждого правила нормируются на единицу. Если для класса c у нас есть 3 правила с весами $\alpha_1 = 0.5$, $\alpha_2 = 0.3$, и $\alpha_3 = 0.2$, как будет выглядеть итоговая сумма голосов $\Gamma_c(x)$, если объект x классифицируется всеми тремя правилами как класс c ?

Решение: Итоговая сумма голосов для класса c рассчитывается по формуле:

$$\Gamma_c(x) = \sum_{t=1}^{T_c} \alpha_{tc} \varphi_{tc}(x)$$

где $\varphi_{tc}(x) = 1$, если правило классифицирует объект как класс c , и 0 в противном случае. Если все 3 правила классифицируют объект как c , то:

$$\Gamma_c(x) = 0.5 + 0.3 + 0.2 = 1.0$$

Задача 3: Проблема диверсификации правил

Какова вероятность ошибки при использовании алгоритма голосования, если все правила сильно похожи друг на друга (например, классифицируют одинаковые подмножества объектов)?

Решение: Если правила сильно похожи, то вероятность ошибки возрастает. В таких случаях, возможно, правило не будет существенно различать объекты, и алгоритм может ошибаться при классификации новых объектов. Чтобы уменьшить вероятность ошибки, правила должны быть разнообразными, то есть они должны выделять разные подмножества объектов. Для максимизации различий между правилами можно использовать метод, как в алгоритме CORAL, который строит покрывающие наборы правил, постепенно исключая часто встречающиеся признаки.

Задача 4: Принцип диверсификации правил

Пусть для объекта x имеется 10 правил, из которых 8 классифицируют объект как класс c , а остальные 2 — как класс d . Какой отступ $M(x)$ будет при расчете вероятности правильной классификации?

Решение: Отступ для объекта x определяется как разница между голосами для правильного класса и максимальным голосом для всех остальных классов:

$$M(x) = \Gamma_c(x) - \Gamma_{\bar{c}}(x)$$

Если из 10 правил 8 голосуют за класс c (доля голосов 0.8) и 2 — за класс d (доля голосов 0.2), то:

$$M(x) = 0.8 - 0.2 = 0.6$$

Если отступ положителен ($M(x) > 0$), то классификация будет правильной.

Задача 5: Отказы от классификации

Если для объекта x не существует правила, которое его классифицирует, что должен делать алгоритм голосования? Как можно обработать такой случай?

Решение: В таком случае алгоритм может либо отказаться от классификации, либо отнести объект к классу с наименьшей ценой ошибки. Такой отказ от классификации часто называют обнаружением нетипичности (novelty detection), что является отдельной задачей в машинном обучении. Важно, чтобы количество отказов не было слишком большим, так как это может снизить эффективность алгоритма.

Алгоритм ТЭМП

Полный перебор всех конъюнкций ранга не более K требует экспоненциального по K числа операций. В реальных задачах объём вычислений становится огромным уже при $K > 3$, и от идеи полного перебора приходится отказаться.

Существует две стандартные стратегии перебора конъюнкций: поиск в глубину (depth-first search) и поиск в ширину (breadth-first search). Первая применяется в алгоритме КОРА, вторая — в алгоритме ТЭМП, предложенным Г. С. Лбовым в 1976 году. Поиск в ширину работает немного быстрее, и в него легче встраивать различные эвристики, сокращающие перебор.

В исходном варианте алгоритм ТЭМП выполнял полный перебор всех конъюнкций ранга не более K . Ниже описан слегка модифицированный вариант, позволяющий ограничить перебор и увеличить максимальный ранг конъюнкций K .

Алгоритм 1.9 начинает процесс поиска закономерностей с построения конъюнкций ранга 1. Для этого отбираются не более T_1 самых информативных предикатов из базового множества \mathcal{B} . Затем к каждому из отобранных предикатов добавляется по одному терму из \mathcal{B} всеми возможными способами. Получается не более $T_1|\mathcal{B}|$ конъюнкций ранга 2, из которых снова отбираются T_1 самых

информационных. И так далее. На каждом шаге процесса делается попытка добавить один терм к каждой из имеющихся конъюнкций. Наращивание конъюнкций прекращается либо при достижении максимального ранга K , либо когда ни одну из конъюнкций не удаётся улучшить путём добавления терма.

Лучшие конъюнкции, собранные со всех шагов, заносятся в списки R_c . Таким образом, списки R_c могут содержать конъюнкции различного ранга.

Параметр T_1 позволяет найти компромисс между качеством и скоростью работы алгоритма. При $T_1 = 1$ алгоритм ТЭМП работает исключительно быстро и строит единственную конъюнкцию, добавляя термы по очереди. Фактически, он совпадает с жадным Алгоритмом 1.2. При увеличении T_1 пространство поиска расширяется, алгоритм начинает работать медленнее, но находит больше информативных конъюнкций. На практике выбирают максимальное значение параметра T_1 , при котором поиск занимает приемлемое время. Однако стратегия поиска всё равно остаётся жадной — термы оптимизируются по-отдельности, и при подборе каждого терма учитываются только предыдущие, но не последующие термы.

Для улучшения конъюнкций к ним применяют эвристические методы «финальной шлифовки» — стабилизацию и редукцию.

В результате стабилизации конъюнкции становятся локально неулучшаемыми. Алгоритм в целом становится более устойчивым — при незначительных изменениях в составе обучающей выборки он чаще находит одни и те же закономерности, а значит, улучшается его способность обобщать эмпирические факты.

В результате стабилизации некоторые конъюнкции могут совпасть, и в списке появятся дубликаты. Их удаление предусмотрено на шаге 13. Если список R_c поддерживается отсортированным по информативности, то удаление дубликатов является недорогой операцией, так как достаточно проверять на совпадение только соседние конъюнкции с одинаковой информативностью.

Если задать $T_1 = \infty$, то алгоритм выполнит полный перебор, как в исходном варианте ТЭМП. «Финальная шлифовка» в этом случае не нужна.

Алгоритм 1.9. Построение списка информативных конъюнкций методом поиска в ширину (алгоритм ТЭМП)

Алгоритм 1.9: Построение списка информативных конъюнкций методом поиска в ширину

Вход:

- X_ℓ — обучающая выборка;
- \mathcal{B} — семейство элементарных предикатов;
- $c \in Y$ — класс, для которого строится список конъюнкций;

- K — максимальный ранг конъюнкций;
- T_1 — число лучших конъюнкций, отбираемых на каждом шаге;
- T_0 — число лучших конъюнкций, отбираемых на последнем шаге, $T_0 \leq T_1$;
- I_{\min} — порог информативности;
- E_{\max} — порог допустимой доли ошибок;
- X_k — контрольная выборка для проведения редукции.

Выход: Список конъюнкций $R_c = \{\varphi_t^c(x) : t = 1, \dots, T_c\}$.

1. $R_c \leftarrow \emptyset$;
2. Для всех $\beta \in \mathcal{B}$: Добавить_в_список(R_c , β , T_1);
3. Для всех $k = 2, \dots, K$:
 - (а) Для всех конъюнкций $\varphi \in R_c$ ранга ($k - 1$):
 - и. Для всех предикатов $\beta \in \mathcal{B}$, которых ещё нет в конъюнкции φ :
 - Добавить терм β к конъюнкции φ : $\varphi' = \varphi \wedge \beta$;
 - Если $I_c(\varphi') > I_{\min}$, $E_c(\varphi') \leq E_{\max}$ и конъюнкция φ' нет в R_c , то Добавить_в_список(R_c , φ' , T_1).
 4. Для всех конъюнкций $\varphi \in R_c$:
 - (а) Стабилизация(φ);
 - (б) Редукция(φ , X^k);
 5. Удалить из списка R_c дублирующие конъюнкции;
 6. Оставить в списке R_c не более T_0 лучших конъюнкций.

Достоинства алгоритма ТЭМП

- ТЭМП существенно более эффективен, чем КОРА, особенно при поиске конъюнкций ранга больше 3. Он решает поставленную задачу за $O(KT_1|B|ell)$ операций, тогда как КОРА имеет трудоёмкость $O(|B|Kell)$.
- Параметр T_1 позволяет управлять жадностью алгоритма и находить компромисс между качеством конъюнкций и скоростью работы алгоритма.
- Благодаря простоте и эффективности алгоритм ТЭМП можно использовать в составе других алгоритмов как генератор конъюнкций, достаточно близких к оптимальным.

Недостатки алгоритма ТЭМП

- Нет гарантии, что будут найдены самые лучшие конъюнкции, особенно при малых значениях параметра T_1 .
- Алгоритм не стремится увеличивать различность конъюнкций, добиваясь равномерного покрытия объектов выборки. Стабилизация и редукция лишь отчасти компенсируют этот недостаток.
- Нет настройки коэффициентов α_{tc} ; предполагается простое голосование.

Задачи

Задача 1.

Рассмотрите граф $G = (V, E)$, где $V = \{A, B, C, D, E\}$ и $E = \{(A, B), (A, C), (B, D), (C, D), (D, E)\}$. Как алгоритм ТЭМП выполняет поиск в ширину для этого графа? Предложите, как можно выполнить поиск в глубину.

Решение:

1. Алгоритм ТЭМП начинает с вершины A , добавляя её соседей B и C в очередь. Затем из очереди выбирается вершина B , её сосед D добавляется в очередь, и так далее.
2. Для поиска в глубину из вершины A алгоритм выбирает один путь (например, $A \rightarrow B \rightarrow D \rightarrow E$), пока не дойдёт до конца, затем возвращается и ищет другие пути.

Задача 2.

Пусть у нас есть множество предикатов $B = \beta_1, \beta_2, \beta_3$ и максимальный ранг $K = 2$. Какое количество конъюнкций будет проверено алгоритмом ТЭМП, если для каждого шага выбирается $T_1 = 2$? Если мы увеличим T_1 до 3, как это повлияет на количество проверяемых конъюнкций?

Решение:

- При $T_1 = 2$ для ранга 1 будет 2 конъюнкции, для ранга 2 — $2 \times 3 = 6$ конъюнкций. Всего будет проверено $2 + 6 = 8$ конъюнкций.
- Если $T_1 = 3$, то для ранга 1 будет 3 конъюнкции, для ранга 2 — $3 \times 3 = 9$ конъюнкций. Всего будет проверено $3 + 9 = 12$ конъюнкций.

Задача 3.

Представьте, что вы используете алгоритм ТЭМП для обучения модели с большим количеством предикатов. Какой будет влияние на время вычислений, если вы увеличите T_1 с 1 до 5? Объясните это с точки зрения расширения пространства поиска.

Решение:

- При $T_1 = 1$ алгоритм будет проверять только по одному терму для каждой конъюнкции на каждом шаге, что будет довольно быстрым, но может не давать лучшие результаты.
- При увеличении T_1 до 5 пространство поиска расширяется, так как на каждом шаге будет добавляться больше термов к конъюнкциям, что увеличивает количество проверяемых комбинаций и тем самым замедляет процесс.
- В результате увеличение T_1 может существенно замедлить алгоритм, но при этом улучшится качество найденных решений.

Глава 13

Поиск ассоциативных правил

Алгоритм FP-Growth

Алгоритм FP-Growth был опубликован в 2005 году Кристианом Боргельтом, и до сих пор является передовым в поиске ассоциативных правил. Алгоритм APrivity, работает за $O(nl \cdot 2^k)$, где n - это число признаков, l - число записей в базе, а k - максимальный размер ассоциативного правила. В то же время FP-Growth работает за $O(nl + f(k, n))$.

Идея этого алгоритма лежит в создании бора всех наборов φ , который будет хранить всю информацию о частотах встречаемости наборов, благодаря чему нам не придётся каждый раз пробегать всю базу с целью подсчёта $\nu(\varphi)$

Алгоритм состоит из двух этапов:

1. Построение FP-дерева
2. Построение списка ассоциативных правил

Построение FP-дерева

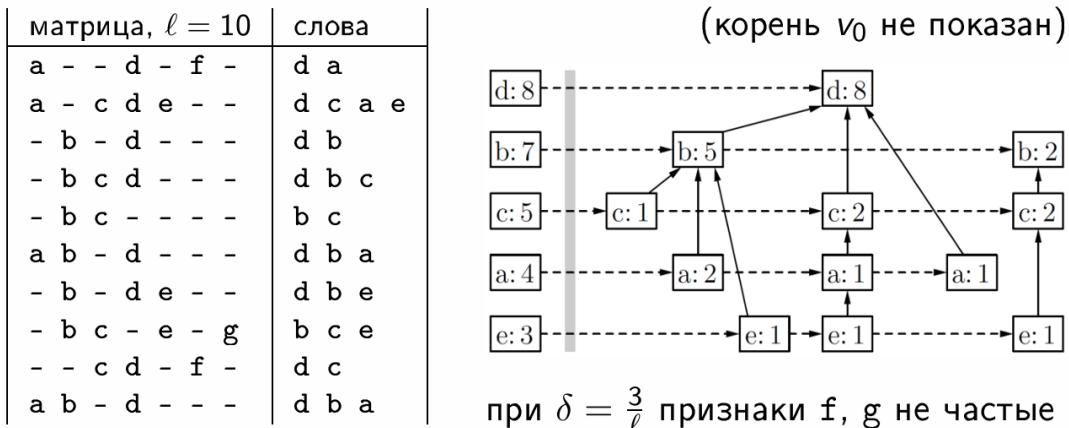


Рис. 13.1. Пример FP-дерева

Первым делом упорядочим все признаки $f \in \mathcal{F} : \nu(f) \geq \delta$ по убыванию $\nu(f)$, их порядок будет соответствовать уровням вершин дерева (это не глубины!). Будем последовательно перебирать транзакции из базы. Будем хранить бор (FP-дерево), в каждой вершине v которого будут храниться:

1. Признак $f_v \in \mathcal{F}$
2. Множество дочерних вершин S_v

3. Счетчик $c_v = l\nu(\phi_v)$, где ϕ_v - набор состоящий из признаков соответствующих пути от корня до данной вершины. По сути этот счетчик равен количеству строк из базы, распознаваемых данной вершиной.

Пусть для текущей транзакции i положительны признаки из множества $F_i = \{f \in \mathcal{F} : f(x_i) = 1\}$, упорядочим их по убыванию $\nu(f)$, получим строку R_i , начнем распознавать её бором. Если её нет в боре, создадим недостающие вершины, проинициализировав счетчики $c_{v_i} = 0$ в новых вершинах. Добавим 1 к счетчикам c_v вершин, которые лежат на пути от корня, соответствующем R_i .

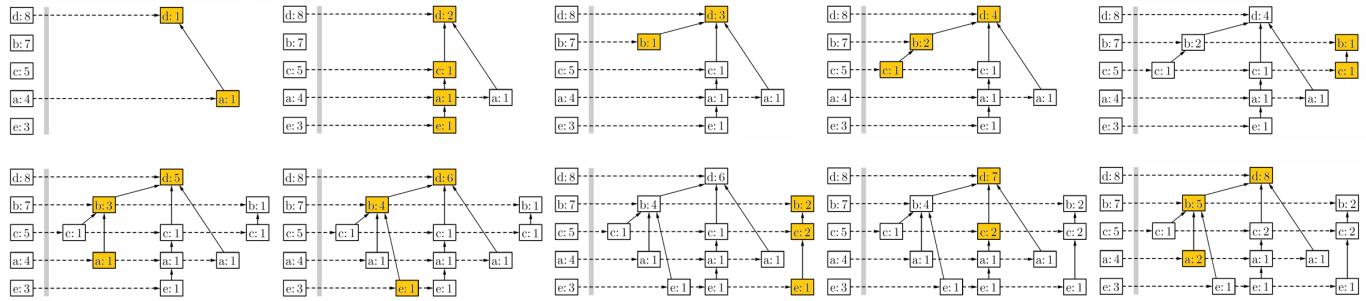


Рис. 13.2. Иллюстрация этапов построения FP-дерева

Введём обозначения:

- $V(T, f) = v \in T : f_v = f$ - все вершины, соответствующие признаку f . Соответственно, у них равные глубины.
- $C(T, f) = \sum_{v \in V(T, f)} c_v$ - Сумма счетчиков вершин, соответствующих признаку f . Несложно понять, что эта сумма равна числу объектов $x_i \in U$, для которых $f(x_i) = 1$. Другими словами $C(T, f) = l\nu(f)$

Построение условного FP-дерева

Прежде чем приступить к второму этапу алгоритма, разберёмся в одной важной для него процедуре.

Определение. Пусть FP-дерево T построено по подвыборке U . Условное FP-дерево $T' = T|\varphi$ - это FP-дерево, порождаемое подвыборкой $\{x_i \in U | \varphi(x_i) = 1\}$ из которого удалены все вершины признака f и ниже.

Рассмотрим алгоритм построения $T|f$ по T , где f - признак из \mathcal{F} :

1. Оставим в дереве только те вершины, которые принадлежат путям, идущим от вершин признака f до корня.
2. Обновим счетчики c_i : для вершин признака f ничего не меняем, а для остальных рекурсивно снизу вверх обновляем по формуле $c_u = \sum_{w \in S_u} c_w$

3. Удаляем вершины признака f из T' .

Корректность следует из построения.

Построение списка ассоциативных правил

Для начала поймем, как эффективно находить $\nu(\varphi)$ используя FP-дерево. Для этого мы можем просто найти $f = \operatorname{argmin}_{f \in \varphi} (\nu(f))$ и просуммировать c_v для всех вершин v признака f , таких что $\varphi \subset [f_{v_0}, f_v]$, где $[f_{v_0}, f_v]$ - набор признаков f_i вершин из пути от корня до v .

Рассмотрим функцию $\text{FP-find}(T, \varphi, R)$, которая находит по FP-дереву T , все частные наборы, содержащие только частный набор φ , и признаки, которые в FP-дереве стоят выше всех признаков из φ , и добавляет их в список R . Она играет ключевую роль в работе данного алгоритма. Вот как её можно реализовать:

Псевдокод:

1. **ПРОЦЕДУРА** $\text{FP-find}(T, \varphi, R)$;

Вход: T – FP-дерево, частный набор φ , список наборов R .

Выход: Добавить все частные наборы, содержащие φ в R .

2. для всех $f \in \mathcal{F} : V(T, f) \neq \emptyset$ по уровням снизу вверх:

3. если $C(T, f) \geq l\delta$ то:

4. набор $\varphi \cup \{f\}$ - частый, добавляем его в R ;

5. $T' := T|f$ - условное FP-дерево;

6. $\text{FP-find}(T', \varphi \cup \{f\}, R)$; - рекурсивно найти все частые наборы, содержащие $\varphi \cup \{f\}$

Для получения списка всех частых наборов, мы можем просто запустить $\text{FP-find}(T, \emptyset, \emptyset)$. Алгоритм просмотрит все наборы в обратном лексикографическом порядке. Для выделения ассоциативных правил теперь можем запустить функцию `AssocRules` из алгоритма APrivity.

Задачи для практики

Задача 1. Как используя FP-дерево, построенное по выборке U найти а) самый частый набор, б) самый редкий набор?

Решение.

а) По свойству антимонотонности, один из наборов размера 1 будет самым часто встречаемым. Так как признаки в FP-дереве уже отсортированы по частоте, самым частым будет самый верхний. Сложность алгоритма получается равной $O(1)$.

б) Подмножество каждого пути из корня в FP-дереве соответствует какому-то набору. Также каждый набор, соответствующий строке в базе, соответствует какому-то пути в дереве. По свойству антимонотонности, одним из наборов, соответствующих путям из корня до листьев, должен быть самым редким. Количество упоминаний одного из таких наборов - это значение счетчика c_v в соответствующем листе. То есть наша задача сводится к поиску листа с минимальным значением этого счетчика. Это можно сделать, пройдя обходом в глубину по всему дереву, за $O(2^n)$, где n - глубина дерева в худшем случае.

Задача 2. Как по уже построенному FP-дереву для выборки U и фиксированных параметров поддержки δ и значимости χ , найти все ассоциативные правила, удовлетворяющие δ, χ вида $a \rightarrow b$, где $a, b \in \mathcal{F}$ за $O(n2^n)$ в худшем случае?

Решение.

Заметим, что всего таких правил порядка n^2 , где n - число признаков. То есть для того, чтобы дать ответ, нам достаточно найти $\nu(f), \nu(f \cup g)$ для всех признаков f, g из дерева. Первые величины уже известны, так как были вычислены на первом этапе построения FP-дерева. Для нахождения всех $\nu(f \cup g)$, достаточно создать двумерный массив C , такой что C_{fg} равно числу $x \in U : (f \wedge g)(x_i) = 1$. Для его заполнения мы можем пройти обходом в глубину по дереву, и находясь в вершине v , прибавлять c_v к $C_{f_u f_v} = C_{f_v f_u}$ для всех $u \in [v_0, v]$, где $[v_0, v]$ - путь от корня до v . Этот обход потребует порядка $O(n2^n)$ действий. То есть суммарно у нас уйдет $O(n2^n + n^2) = O(n2^n)$ действий.

Задача 3. Оцените время работы алгоритма FP-Growth для нахождения ассоциативных правил, удовлетворяющих поддержке δ и значимости χ в худшем случае.

Решение.

Если с временем построения FP-дерева все понятно - оно совпадает с временем построения бора по массиву строк и равно $O(nl)$. То асимптотика генерации всех частых наборов и выделение ассоциативных правил значительно труднее для вычисления. Построение условного FP-дерева $T|f$ по самому нижнему признаку f выполняется одним обходом в глубину - асимптотически $O(2^n)$. Пусть $\text{FP-find}(T, \emptyset, \emptyset)$ выполняется за время $k(n)$, тогда $k(n) = O\left(\sum_{i=0}^{n-1} (2^i + k(i))\right)$, где 2^i действий уходит на построение условного дерева по каждому из признаков.

Сделаем замену $t(n) := 2^n + k(n)$, тогда $t(n) - 2^n = O\left(\sum_{i=0}^{n-1} t(i)\right)$, заметим, что для $t(n) = n2^n$ верно: $n2^n = O\left(\sum_{i=0}^{n-1} i2^i\right) = O((n-2)2^n)$. Значит $k(n) = O(n2^n)$. Теперь найдем время выделения ассоциативных правил. Операция нахождения $\nu(\varphi)$ по данному φ может быть оптимизирована с использованием дерева поиска для хранения множества φ , что даст суммарное время работы $O(n \log n + 2^d \cdot \log(n)) = O((2^d + n) \cdot \log(n))$, где d - глубина самого нижнего признака из φ . Для выделение ассоциативных правил из набора φ длины k , время упирается в скорость вычисления $\nu(\varphi')$ для проверки $\nu(y'|\varphi') \geq \chi$, поэтому оно будет не хуже, чем $O(2^k \cdot (2^d + n) \cdot \log(n))$, и не лучше $O(2^{k-1} \cdot (2^d + n) \cdot \log(n))$ - так как есть 2^{k-1} набор содержащий признак глубины d , поддержку которого надо вычислить. Наконец выделение всех ассоциативных правил для всех частых наборов в худшем случае будет порядка

$$O\left(\sum_{d=1}^n \sum_{k=1}^{n-d} C_{n-d}^k \cdot 2^{k-d} \cdot \log(n)\right) = O\left(\sum_{d=1}^n \frac{3^{n-d}}{2^d} \cdot \log(n)\right) = O(3^n \log(n))$$

Что и будет решающим слагаемым в асимптотике работы всего алгоритма.

Алгоритм APrIory

Свойство антимонотонности

Алгоритм **APriory** основывается на свойстве **антимонотонности** функции $\varphi(x) = \bigwedge_{f \in \varphi} f(x)$:

Для любых $\psi, \varphi \subset \mathcal{F}$, из $\varphi \subset \psi$ следует $\nu(\varphi) \geq \nu(\psi)$.

Следствия:

1. Если ψ частый, то все его подмножества $\varphi \subset \psi$ частые.
2. Если φ не частый, то все наборы $\psi \supset \varphi$ также не частые.
3. $\nu(\varphi \cup \psi) \leq \nu(\varphi)$ для любых φ, ψ .

Основываясь на этих свойствах, мы можем построить итеративный алгоритм нахождения ассоциативных правил (**APriory**). Алгоритм состоит из двух этапов:

1. Поиск частых наборов
2. Выделение ассоциативных правил

Вторая часть алгоритма считается полностью *решенной*, так как существует простая и эффективная процедура, реализующая её. Первая же часть представляет наибольший интерес, так как требует многократное обращение к базе данных, поэтому стоит важный вопрос оптимизации этого этапа.

Поиск частых наборов

Идея алгоритма: на j этапе храним все *частые наборы* мощности j : G_j . Тогда при переходе от G_j к G_{j+1} пытаемся увеличить каждое из множеств $\varphi \in G_j$, чтобы оно по-прежнему оставалось *частым набором*.

Псевдокод:

Вход: X^ℓ — обучающая выборка; минимальная поддержка δ ; минимальная значимость χ .

Выход: $R = \{(\varphi, \psi)\}$ — список ассоциативных правил.

1. Множество **всех частых** исходных признаков:

$$G_1 := \{f \in \mathcal{F} \mid \nu(f) \geq \delta\};$$

2. Для всех $j = 2, \dots, n$:

3. Множество **всех частых** наборов мощности j :

$$G_j := \{\varphi \cup \{f\} \mid \varphi \in G_{j-1}, f \in G_1, \nu(\varphi \cup \{f\}) \geq \delta\};$$

4. Если $G_j = \emptyset$, то:

5. **выход** из цикла по j ;

6. $R := \emptyset$;

7. Для всех $\psi \in G_j$, $j = 2, \dots, n$:

8. **AssocRules(** R, ψ, \emptyset **)**;
-

Заметим, что за лаконичной математической формулой $\nu(\varphi \cup \{f\}) \geq \delta$ на самом деле скрывается просмотр базы данных, что делает эту проверку очень тяжелой.

Выделение ассоциативных правил.

Идея алгоритма: отщипываем по одному признаку из φ , проверяем ассоциативность получившегося правила.

Псевдокод:

Вход: R — список ассоциативных правил;

Выход: (φ, y) — ассоциативное правило;

1. **ПРОЦЕДУРА** $\text{AssocRules}(R, \varphi, y)$;

2. для всех $f \in \varphi$

3. $\varphi' := \varphi \setminus \{f\}$; $y' := y \cup \{f\}$;

-
4. **если** $\nu(y'|\varphi') \geq \chi$ **то**
 5. добавить ассоциативное правило (φ', y') в список R ;
 6. **если** $|\varphi'| > 1$ **то**
 7. AssocRules (φ', y') ;
-

Задачи для практики

Задача 1: Простота алгоритма APRIORI: Чтобы лучше прочувствовать алгоритм APRIORI (а также убедиться в его простоте) проделайте его руками для следующей таблицы транзакций (числа - номера транзакции, буквы - признаки), а именно найдите все частые наборы признаков (G_1, G_2, \dots) при $\text{minsup} = 0.3$.

Таблица 13.1. Таблица транзакций

Транзакция	A	B	C	D	E	F
1	1	1	1	0	0	0
2	1	0	1	1	0	0
3	0	1	1	1	1	0
4	1	1	0	1	0	1
5	1	0	1	1	1	0
6	0	1	1	0	0	0
7	1	1	1	0	1	0
8	1	0	0	1	1	1
9	0	1	1	1	0	0
10	1	1	1	0	1	0

Решение:

Шаг 1: Частые одноэлементные наборы (G_1)

Считаем поддержку для каждого элемента:

$$\{A\} : 7, \quad \{B\} : 6, \quad \{C\} : 8, \quad \{D\} : 6, \quad \{E\} : 5, \quad \{F\} : 2.$$

Частые одноэлементные наборы ($support \geq 3$):

$$G_1 = \{\{A\}, \{B\}, \{C\}, \{D\}, \{E\}\}.$$

Шаг 2: Частые пары (G_2)

Формируем все возможные пары элементов (φ_2) и считаем их поддержку:

$$\begin{aligned} \{A, B\} : 4, & \quad \{A, C\} : 4, \quad \{A, D\} : 4, \quad \{A, E\} : 4, \\ \{B, C\} : 6, & \quad \{B, D\} : 3, \quad \{B, E\} : 3, \\ \{C, D\} : 4, & \quad \{C, E\} : 4, \quad \{D, E\} : 3. \end{aligned}$$

Частые пары ($support \geq 3$):

$$G_2 = \{\{A, B\}, \{A, C\}, \{A, D\}, \{A, E\}, \{B, C\}, \{B, D\}, \{B, E\}, \{C, D\}, \{C, E\}, \{D, E\}\}.$$

Шаг 3: Частые тройки (G_3)

Формируем тройки (φ_3) и считаем их поддержку:

$$\begin{aligned} \{A, B, C\} : 3, & \quad \{A, B, D\} : 1, \quad \{A, B, E\} : 1, \\ \{A, C, D\} : 2, & \quad \{A, C, E\} : 3, \quad \{A, D, E\} : 1, \\ \{B, C, D\} : 2, & \quad \{B, C, E\} : 3, \quad \{B, D, E\} : 1, \\ \{C, D, E\} : 1. & \end{aligned}$$

Частые тройки ($support \geq 3$):

$$G_3 = \{\{A, B, C\}, \{A, C, E\}, \{B, C, E\}\}.$$

Шаг 4: Частые четверки (G_4)

Формируем четверки (φ_4):

$$\{A, B, C, E\} : 1.$$

Поддержка меньше минимальной ($support < 3$), поэтому завершаем алгоритм.

Задача 2: Сложность алгоритма APrivity: К.В. Воронцов так отзывался об алгоритме APrivity: "Простой, классический, везде описанный, но пользоваться им не надо". Мы уже упоминали о неэффективности этого алгоритма, теперь убедитесь в этом сами: посчитайте временную асимптотику и асимптотику используемой памяти (обозначим $|D|$ - количество признаков)

Решение: Если размер максимального по включению частого множества $|S|$, то мы переберем также все его подмножества, то есть пройдет $\mathcal{O}(2^{|S|})$ итераций. Итого асимптотика $\mathcal{O}(2^{|D|})$

Пусть снова S - максимальное частое множество. Значит все его подмножества - также частые множества, значит они также будут сохранены, значит памяти будет занято $\mathcal{O}(2^{|S|})$. Итоговая асимптотика памяти $\mathcal{O}(2^{|D|})$.

Основная проблема алгоритма - мы наивно ищем частые наборы. Следующий алгоритм *FP-Growth* признан решить эту проблему, используя эффективные структуры данных для быстрого поиска.

Задача 3: Недостатки алгоритма APriry на практике: В прошлой задаче мы убедились в неэффективности алгоритма с точки зрения его временной сложности. Подумайте теперь о недостатках алгоритма при его использовании на практике (не считая временной сложности). Для ответа вспомните примеры использования ассоциативных правил в жизни.

Решение: Вспомнил классический пример применения поиска ассоциативных правил — анализ рыночной корзины. При непосредственном использовании алгоритма, где товары рассматриваются как признаки, а чеки — как объекты, возникает важная проблема: нам необходимо не просто анализировать отдельные товары, а учитывать сгруппированные категории. Например, нас интересует не просто покупка конкретных видов хлеба, а сам факт приобретения хлебобулочных изделий (или, например, приобретения черного хлеба). Таким образом, нужно учитывать иерархию признаков.

Другая проблема - мы никак не учитываем информацию о самих клиентах (например, семейное положение, возраст...). На практике при построении ассоциативных правил эта информация сильно влияет на правила, поэтому их нельзя игнорировать.

Задача автоматического выделения терминов: алгоритм TopMine, UD-Pipe, модель PLSA.

Цель для такой задачи является выделение составных терминов в текстовых коллекциях. Термин - фраза (n-грамма) со следующим набором свойств:

- Высокая частотность - много раз встречается в коллекции.
- Контактная сочетаемость слов - состоит из слов, неслучайно часто встречающихся вместе.
- Полнота - является максимальной по включению цепочкой слов.
- Синтаксическая связность - является грамматически корректным словосочетанием.
- Тематичность - часто встречается в узком подмножестве тем.

Для проверки каждой из свойств используются

- Статистический анализ: для проверки первых трёх свойств терминов. Он позволяет находить высокочастотные токены, выделять слова, неслучайно стоящие рядом и штрафовать неполные последовательности слов, входящие как подмножество в другую высокочастотную последовательность неслучайно стоящих слов.
- Синтаксический анализ: для проверки синтаксической связности. Данный анализ позволяет выделять синтактически связанные словосочетания в предложениях.

вход: коллекция D , пороги ε_k ;

выход: хэш-таблица частот $C(a_1, \dots, a_k)$, $k = 1, \dots, k_{\max}$;

$C(w) := n_w$ для всех $w \in W$;

$A_{d,0} := \{1, \dots, n_d\}$;

для $k := 1, \dots, k_{\max}$

для **всех** $d \in D$

$A_{d,k} := \{i \in A_{d,k-1} \mid C(w_{d,i}, \dots, w_{d,i+k-1}) \geq \varepsilon_k\}$;

для **всех** $i \in A_{d,k}$

если $i+1 \in A_{d,k}$ **то** $\text{++}C(w_{d,i}, \dots, w_{d,i+k})$;

 оставить только частые k -граммы: $C(a_1, \dots, a_k) \geq \varepsilon_k$;

Рис. 13.3. Алгоритм быстрого поиска

- Тематическая модель: для проверки тематичности. Она позволяет сопоставить каждому токену распределение тем.

Для каждого анализа мы будем использовать модели TopMine, UDPipe и PLSA для семантического, синтаксического анализа и тематической модели соответственно. Рассмотрим в подробности каждую модель

Статистический анализ: TopMine

Этот алгоритм итеративно сливает слова и фразы в предложении, рассчитывая для каждого слияния оценку значимости $SignificanceScore$ и останавливается, когда для всех возможных слияний значимость меньше заданного порога.

В начале мы должны найти все частые k -грамм и алгоритм быстрого поиска приведён на рисунке 1. где $C(a_1, \dots, a_k)$ - хэш-таблица частот k -грамм, $a_i \in W$, $C(w) = n_w$ для всех униграмм $w \in W : n_w \geq \varepsilon_1$, где W - множество слов или фраз. ε_1 - пороговое значение частоты частых k -грамм $A_{d,k}$ - множество позиций i в документе d , с которых начинаются все частые k -граммы $C(w_{d,i}, \dots, w_{d,i+k-1}) \geq \varepsilon_k$. Свойство антимонотонности: $C(a_1, \dots, a_k) \geq C(a_1, \dots, a_k + 1)$

Затем должны провести итеративное слияние фраз с понижением значимости до α . $SignificanceScore = \frac{p_{uv} - p_u p_v}{\sqrt{p_{uv}}}$ где p_u - оценка вероятности встретить фразу u , p_{uv} = оценка вероятности встретить фразу uv .

В начале у нас есть кортеж исходных фраз и первой итерацией считается $SignificanceScore$ для всех соседних пар фраз. И затем из кортежа удаляются все пары с $SignificanceScore$ больше α . Оставшиеся элементы в кортеже и являются термами. На рисунке 2 приведен пример работы алгоритма TopMine.

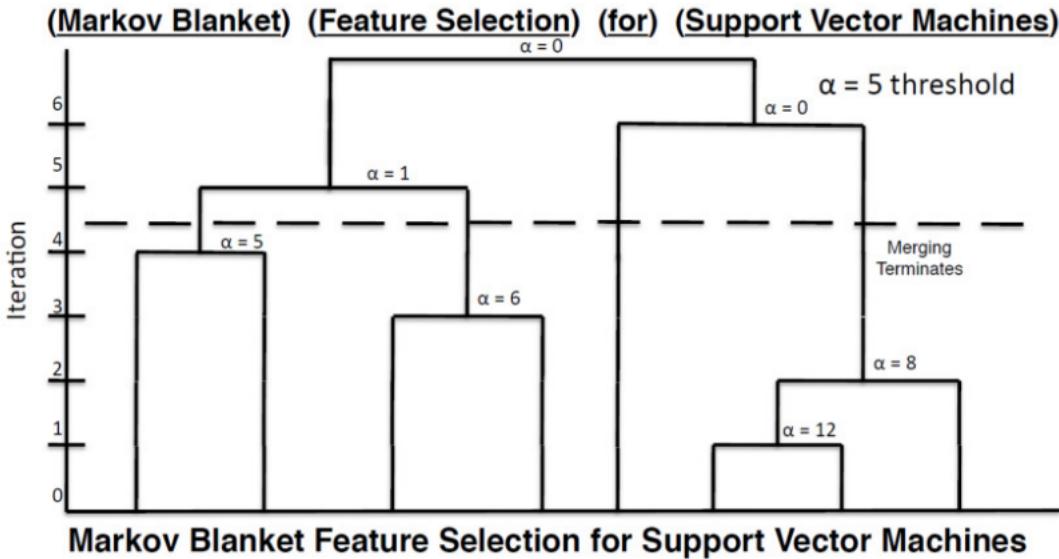


Рис. 13.4. Пример работы алгоритма TopMine

Синтаксический анализ: UDPipe

UDPipe - предобученная модель, которая может распознавать синтаксические связи в предложениях и разметка частей речи слов в предложениях. На вход подаётся список предложений, и для каждого слова в предложениях вычисляется:

- Часть речи слова.
- Член предложения.
- ID родительского слова.

Модель для каждого предложения создаёт синтаксическое дерево, и при помощи неё проверяет словосочетаний-кандидаты. Для них подсчитывается оценка его синтаксической связанности по формуле $SyntaxScore(W) = \max SyntaxDistance(w_i, w_j)$, где W - N-грамма, $1 \leq i \leq N$, $1 \leq j \leq N$ и $i \neq j$. Таким образом, если одно из слов в словосочетании-кандидате синтактически не связано с остальным, $SyntaxScore$ это выявит.

Тематический анализ: PLSA

У нас есть коллекция текстовых документов D и словарь токенов W , из которых состоят документы. Каждый документ $d \in D$ представляет собой последовательность входящих в него токенов из словаря W . Если предположить, что местоположение токенов в документе не влияет на определение тематики документа, то документ это подмножество $d \subset W$, в котором каждому токену $w \in d$ поставлено в соответствие число n_{dw} вхождений в документ d .

Модель описывает вероятности появления токенов w в документах d при предположении условной независимости:

$$p(w|d) = \sum_{t \in T} p(w|t)p(t|d) = \sum_{t \in T} \phi_{wt}\theta_{td}$$

где $\phi_{wt} = p(w|t)$, $\theta_{td} = p(t|d)$ являются обучаемыми параметрами модели. Для обучения параметров модели, представленные в виде матрицы $\Phi = (\phi_{wt})_{W \times T}$ и $\Theta = (\theta_{td})_{T \times D}$ по коллекции документов D максимизируется логарифм правдоподобия

$$L(\Phi, \Theta) = \sum_{d \in D} \sum_{w \in d} n_{dw} \ln \sum_{t \in T} \phi_{wt}\theta_{td} \rightarrow \max_{\Phi, \Theta}$$

при ограничениях неотрицательности и нормировки

$$\sum_{w \in W} \phi_{wt} = 1, \phi_{wt} \geq 0, \sum_{t \in T} \theta_{td} = 1, \theta_{td} \geq 0$$

На основе полученных параметров Φ и Θ для каждого словосочетания-кандидата считается распределение тем $p(t|w) = \phi_{wt} \frac{p(t)}{p(w)}$. Отбор происходит распределением тематик $p(t|w)$ имеет высокую вероятность только для некоторых тем из малого подмножества всех тем T . Это можно оценить через отдаленность $p = p(t|w)$ от равномерного $p_{unif} = 1/T$, где T -мощность множества T . Расстояние между двумя распределениями считается несколькими способами:

- Дивергенция Кульбака-Лейблера

$$KL(pp_{unif}) = \sum_{t \in T} \frac{1}{|T|} \ln \frac{\frac{1}{|T|}}{p(t|w)}$$

- Дивергенция Йесена-Шеннона

$$JS(pp_{unif}) = \frac{1}{2}KL(p_{unif}p_1) + \frac{1}{2}KL(pp_1)$$

$$p_1(t|w) = \frac{1}{2}(p(t|w) + \frac{1}{|T|})$$

- Сумма степенных функций, $\gamma > 1$

$$Deg(p) = \sum_{t \in T} p(t|w)^\gamma$$

Чем больше значение данных метрик для w , тем тематичнее данная N-грамма.

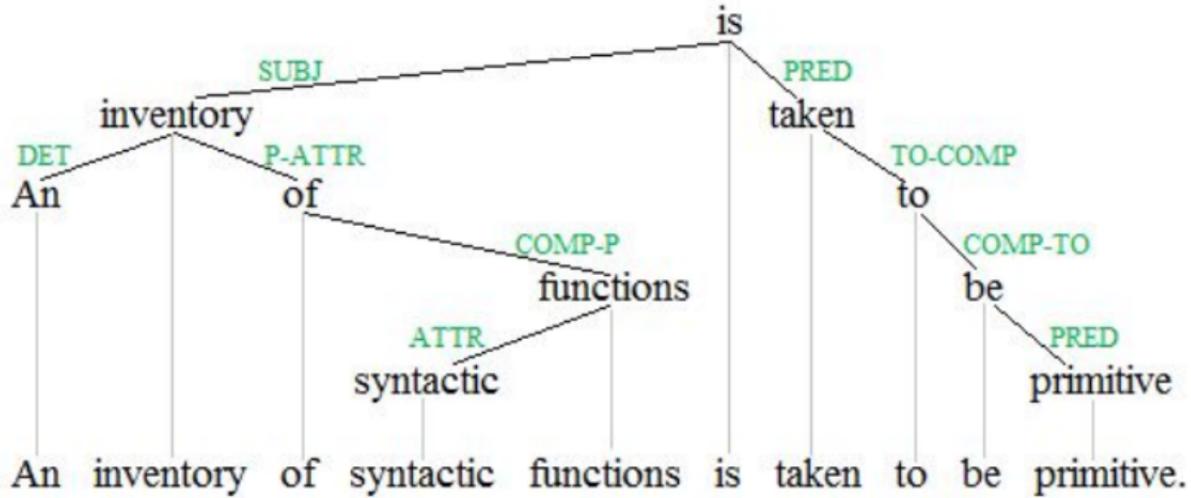


Рис. 13.5. Задача 2

Задачи

Задача 1. Нам дана фраза $a \ b$, было оценено вероятность встречи фраз $p_a = 0,1$, $p_b = 0,2$, $p_{ab} = 0,05$. Пороговая значимость $\alpha = 1$. Является ли фраза ab термином?

Ответ: Нет.

Решение.

$$\frac{p_{ab} - p_a p_b}{\sqrt{p_{ab}}} \approx 0,01 < \alpha$$

Задача 2. Дано синтаксическое дерево (рис. 3) для предложения "An inventory of syntactic functions is taken to be primitive". Найдите *SyntaxScore* для фразы "syntactic functions is".

Ответ: 4.

Задача 3. Привести аналитическое решение максимизации задачи

$$L(\Phi, \Theta) = \sum_{d \in D} \sum_{w \in d} n_{dw} \ln \sum_{t \in T} \phi_{wt} \theta_{td} \rightarrow \max_{\Phi, \Theta}$$

и выразить элементы матрицы Φ и Θ через n_{dw} и $p(t|d, w)$.

Решение.

Запишем лангранжиан задачи, учитывая ограничение нормировки:

$$L(\Phi, \Theta) = \sum_{d \in D} \sum_{w \in d} n_{dw} \ln \sum_{t \in T} \phi_{wt} \theta_{td} - \sum_{t \in T} \lambda_t \left(\sum_{w \in d} \phi_{wt} - 1 \right) - \sum_{d \in D} \mu_d \left(\sum_{t \in T} \theta_{td} - 1 \right)$$

Продифференцируем лангранжиан по ϕ_{wt} и приравняем к нулю производную, получим:

$$\lambda_t = \sum_{d \in D} n_{ds} \frac{\theta_{td}}{p(w|d)}$$

Домножив обе части на ϕ_{wt} , получим:

$$\lambda_t \phi_{wt} = \sum_{d \in D} n_{ds} \frac{\phi_{wt} \theta_{td}}{p(w|d)}$$

По формуле Байеса:

$$p(t|d, w) = \frac{p(w|t)p(t|d)}{p(w|d)} = \frac{\phi_{wt} \theta_{td}}{p(w|d)}$$

Сделав замену получим:

$$\lambda_t \phi_{wt} = \sum_{d \in D} n_{ds} p(t|d, w)$$

Просуммируем по $w \in W$ и получим:

$$\lambda_t \sum_{w \in W} \phi_{wt} = \sum_{w \in W} \sum_{d \in D} n_{ds} p(t|d, w)$$

В соответствии с условием нормировки $\sum_{w \in W} \phi_{wt} = 1$:

$$\lambda_t = \sum_{w \in W} \sum_{d \in D} n_{ds} p(t|d, w)$$

Выразив из $\lambda_t \phi_{wt} = \sum_{d \in D} n_{dw} p(t|d, w)$ ϕ_{wt} и подставив λ_t получаем:

$$\phi_{wt} = \frac{\sum_{d \in D} n_{dw} p(t|d, w)}{\sum_{w_1 \in W} \sum_{d \in D} n_{dw_1} p(t|d, w_1)}$$

Для θ_{td} поступаем аналогично. После приравнивания производной по θ_{td} к 0, домнажения на θ_{td} , суммирования по всем $t \in T$ находим μ_d :

$$\mu_d = \sum_{t \in T} \sum_{w \in d} n_{dw} p(t|d, w)$$

Выразим θ_{td} :

$$\theta_{td} = \frac{\sum_{w \in d} n_{dw} p(t|d, w)}{\sum_{w \in d} n_{dw} \sum_{t_1 \in T} p(t_1|d, w)}$$

Глава 14

Композиции классификаторов

Mixture of Experts (Смесь экспертов)

Модель *Смеси экспертов* (Квазилинейный ансамбль, Mixture of Experts, MoE) является архитектурой машинного обучения, которая сочетает в себе несколько моделей (экспертов) для решения сложных задач. Основная идея заключается в том, чтобы разделить пространство входных данных на части, в каждом из которых определенный эксперт специализируется. Общая модель обучается так, чтобы комбинировать выходы экспертов с учетом их специализации.

Математически, выход модели MoE для признакового описания объекта \mathbf{x} может быть представлен как:

$$y = \sum_{k=1}^K g_k(\mathbf{x}) f_k(\mathbf{x}),$$

где:

- K — количество экспертов,
- $f_k(\mathbf{x})$ — локальная модель, функция k -го эксперта, производящая прогноз,
- $g_k(\mathbf{x})$ — шлюзовая функция или функция компетентности, определяющая вес вклада k -го эксперта, причём $\sum_{k=1}^K g_k(\mathbf{x}) = 1$ и $g_k(\mathbf{x}) \geq 0$ для всех k .

Шлюзовая (Gate) функция обычно моделируется с помощью функции softmax

$$g_k(\mathbf{x}) = \frac{\exp(h_k(\mathbf{x}))}{\sum_{j=1}^K \exp(h_j(\mathbf{x}))},$$

где $h_k(\mathbf{x})$ — функция компетентности для k -го эксперта. Выбираются из каких-либо содержательных соображений.

Преимущество MoE заключается в способности моделировать сложные зависимости путем разделения задачи между специализированными экспертами, что может улучшить обобщающую способность и эффективность обучения.

Задачи и решения

Задача 1 Пусть имеется модель MoE с двумя экспертами, функции которых заданы как $f_1(x) = 2x$ и $f_2(x) = x^2$. Функции $h_k(x) : h_1(x) = -x$, $h_2(x) = x$.

Требуется найти выражение для общего выхода модели y в зависимости от x .

Решение:

Шлюзовая функция моделируется как:

$$g_1(x) = \frac{\exp(-x)}{\exp(-x) + \exp(x)}, \quad g_2(x) = \frac{\exp(x)}{\exp(-x) + \exp(x)}.$$

Суммарный выход модели:

$$y = g_1(x)f_1(x) + g_2(x)f_2(x) = g_1(x) \cdot 2x + g_2(x) \cdot x^2.$$

Подставим выражения для $g_1(x)$ и $g_2(x)$:

$$y = \frac{\exp(-x)}{\exp(-x) + \exp(x)} \cdot 2x + \frac{\exp(x)}{\exp(-x) + \exp(x)} \cdot x^2.$$

Поэтому итоговое выражение для y :

$$y = \frac{\exp(-x) \cdot 2x + \exp(x) \cdot x^2}{\exp(-x) + \exp(x)}.$$

Задача 2 Рассмотрим модель MoE, где шлюзовая функция выбирает только одного эксперта в зависимости от знака x :

$$g_1(x) = \begin{cases} 1, & \text{если } x \geq 0, \\ 0, & \text{если } x < 0, \end{cases}$$

$$g_2(x) = \begin{cases} 0, & \text{если } x \geq 0, \\ 1, & \text{если } x < 0. \end{cases}$$

Функции экспертов заданы как $f_1(x) = x^2$ и $f_2(x) = -x$. Найдите общий выход модели y для любого x .

Решение:

Поскольку в каждый момент времени активен только один эксперт, общий выход модели определяется функцией активного эксперта.

Для $x \geq 0$:

$$g_1(x) = 1, \quad g_2(x) = 0, \quad y = g_1(x)f_1(x) = x^2.$$

Для $x < 0$:

$$g_1(x) = 0, \quad g_2(x) = 1, \quad y = g_2(x)f_2(x) = -x.$$

Таким образом, общий выход модели равен:

$$y = \begin{cases} x^2, & \text{если } x \geq 0, \\ -x, & \text{если } x < 0. \end{cases}$$

Это означает, что модель ведет себя по-разному на положительных и отрицательных значениях x , отражая специализацию экспертов на разных областях входных данных.

Задача 3 Рассмотрим модель Mixture of Experts, состоящую из двух экспертных моделей f_1 и f_2 , а также шлюзовой функции g . Пусть на вход подаётся одно признаковое значение x . Выражения для выходов моделей заданы следующим образом:

$$f_1(x) = w_1x + b_1, \quad f_2(x) = w_2x + b_2, \quad g(x) = \sigma(vx + c)$$

где $\sigma(z) = \frac{1}{1+e^{-z}}$ — сигмоидальная функция активации, а w_1, w_2, b_1, b_2, v, c — параметры модели.

Выход всей модели MoE определяется как:

$$y(x) = g(x)f_1(x) + (1 - g(x))f_2(x)$$

Вопрос: Предположим, что при $x = 0$ мы наблюдаем, что $y(0) = 1$, $f_1(0) = 1$, $f_2(0) = 3$. Найдите значение $g(0)$.

Решение:

Из условия задачи при $x = 0$ имеем:

$$y(0) = g(0) \cdot f_1(0) + (1 - g(0)) \cdot f_2(0) = g(0) \cdot 1 + (1 - g(0)) \cdot 3$$

Подставляем известное значение $y(0) = 1$:

$$\begin{aligned} g(0) \cdot 1 + (1 - g(0)) \cdot 3 &= 1 \\ g(0) + 3 - 3g(0) &= 1 \\ -2g(0) + 3 &= 1 \\ -2g(0) &= -2 \\ g(0) &= 1 \end{aligned}$$

Таким образом, $g(0) = 1$.

Глава 15

Методы бустинга

Гармонический бустинг (Harmonic Boosting)

Гармонический бустинг (Harmonic Boosting) представляет собой ансамблевый метод, целью которого является минимизация вклада моделей с высокой ошибкой путём взвешивания их предсказаний на основе гармонического среднего. Этот подход улучшает устойчивость ансамбля и снижает риск ухудшения качества из-за наличия "шумных" моделей.

Основная идея

Гармоническое среднее даёт больший вес точным моделям и минимизирует влияние слабых или ошибочных предсказаний. Для n моделей ансамбля итоговое предсказание классификации вычисляется как:

$$\hat{y} = \arg \max_k \left(\frac{n}{\sum_{i=1}^n \frac{1}{P_k(y_i)}} \right),$$

где $P_k(y_i)$ — вероятность принадлежности к классу k , предсказанная i -й моделью.

Для регрессии итоговое предсказание:

$$\hat{y} = \frac{n}{\sum_{i=1}^n \frac{1}{y_i}},$$

где y_i — результат i -й модели.

Математическое обоснование

Рассмотрим задачу регрессии. Пусть y_i — предсказание i -й модели, а e_i — её ошибка. Общая ошибка ансамбля определяется как:

$$E = \frac{n}{\sum_{i=1}^n \frac{1}{e_i}}.$$

Гармоническое среднее минимизирует E , поскольку больший вес получают модели с меньшей ошибкой. Это свойство обеспечивает устойчивость ансамбля и снижает влияние "шумных" моделей.

Для классификации гармоническое среднее вероятностей используется в логарифмическом масштабе:

$$\log P(C_k) = - \sum_{i=1}^n \frac{1}{\log P_k(y_i)}.$$

Такой подход позволяет смягчить влияние моделей с крайне низкой вероятностью.

Алгоритм гармонического бустинга

Вход:

- Набор данных $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^m$,
- Количество слабых моделей T ,
- Функция потерь $L(y, \hat{y})$.

Шаги алгоритма:

1. Инициализация ансамбля с первой моделью h_1 , обученной на \mathcal{D} .
2. Вычисление весов для объектов в \mathcal{D} на основе обратной ошибки:

$$w_j^{(t)} = \frac{1}{e_j^{(t)}},$$

где $e_j^{(t)} = L(y_j, h_t(x_j))$.

3. Построение новой модели h_{t+1} с учётом весов $w_j^{(t)}$.
4. Итоговое предсказание ансамбля:

$$\hat{y} = \frac{n}{\sum_{i=1}^T \frac{1}{h_i(x)}}.$$

Сравнение с другими методами бустинга

В отличие от AdaBoost и градиентного бустинга, где обновление весов основывается на увеличении влияния сложных для классификации объектов, гармонический бустинг нацелен на минимизацию влияния моделей с высокой ошибкой. Это делает метод более устойчивым к выбросам и шуму.

Пример применения

Рассмотрим задачу классификации на двух классах C_1 и C_2 . Пусть ансамбль состоит из трёх моделей с вероятностями:

$$P(C_1|h_1) = 0.8, \quad P(C_1|h_2) = 0.6, \quad P(C_1|h_3) = 0.2.$$

Итоговое предсказание для класса C_1 будет:

$$P(C_1) = \frac{3}{\frac{1}{0.8} + \frac{1}{0.6} + \frac{1}{0.2}} \approx 0.44.$$

Преимущества и ограничения

Преимущества:

- Устойчивость к шумным данным и выбросам.
- Минимизация вклада моделей с высокой ошибкой.

Ограничения:

- Высокая вычислительная сложность.
- Чувствительность к выбору базовых моделей.

Задачи

Задача 1: Теоретическое доказательство свойства гармонического среднего

Докажите, что гармоническое среднее минимизирует влияние на общую ошибку моделей с большими значениями ошибки e_i .

Решение:

1. Запишем общую ошибку ансамбля:

$$E = \frac{n}{\sum_{i=1}^n \frac{1}{e_i}}.$$

2. Рассмотрим случай, когда одна из ошибок e_i существенно больше других.
Тогда:

$$\sum_{i=1}^n \frac{1}{e_i} \approx \frac{1}{e_1} + \frac{1}{e_2} + \dots + \frac{1}{e_k} + \frac{1}{e_{\max}},$$

где $e_{\max} \gg e_k$.

3. Гармоническое среднее делает вклад $\frac{1}{e_{\max}}$ минимальным, сохраняя значительное влияние $\frac{1}{e_k}$, что уменьшает общий эффект высоких ошибок на ансамбль.

Задача 2: Алгоритм на данных

Даны предсказания трёх моделей на выборке:

$$P(C_1|h_1) = [0.9, 0.3, 0.6], \quad P(C_1|h_2) = [0.8, 0.4, 0.7], \quad P(C_1|h_3) = [0.7, 0.2, 0.5].$$

Используя гармонический бустинг, вычислите итоговое предсказание ансамбля.

Решение:

1. Рассчитаем гармоническое среднее для каждого объекта:

$$P(C_1|x_1) = \frac{3}{\frac{1}{0.9} + \frac{1}{0.8} + \frac{1}{0.7}} \approx 0.8,$$

$$P(C_1|x_2) = \frac{3}{\frac{1}{0.3} + \frac{1}{0.4} + \frac{1}{0.2}} \approx 0.26,$$

$$P(C_1|x_3) = \frac{3}{\frac{1}{0.6} + \frac{1}{0.7} + \frac{1}{0.5}} \approx 0.57.$$

2. Итоговое предсказание: $[0.8, 0.26, 0.57]$.

Задача 3: Сравнение с арифметическим средним

Сравните результаты гармонического и арифметического средних для трёх моделей с предсказаниями:

$$P(C_1|h_1) = 0.9, \quad P(C_1|h_2) = 0.1, \quad P(C_1|h_3) = 0.8.$$

Решение:

1. Арифметическое среднее:

$$P(C_1) = \frac{0.9 + 0.1 + 0.8}{3} = 0.6.$$

2. Гармоническое среднее:

$$P(C_1) = \frac{3}{\frac{1}{0.9} + \frac{1}{0.1} + \frac{1}{0.8}} \approx 0.27.$$

3. Гармоническое среднее уменьшает вклад модели h_2 с высокой ошибкой ($P(C_1) = 0.1$), что приводит к более устойчивому предсказанию.

Градиентный бустинг

Что такое градиентный бустинг и его история

Градиентный бустинг — это метод машинного обучения, основанный на идее последовательного обучения ансамбля слабых моделей (чаще всего деревьев решений) с использованием градиентного спуска для минимизации заданной функции потерь. Основная цель этого метода — построить сильную модель, которая последовательно улучшает свои предсказания, исправляя ошибки предыдущих шагов.

Идея градиентного бустинга была предложена Джереми Фридманом в 1999 году в его работе "Greedy Function Approximation: A Gradient Boosting Machine". Этот метод стал обобщением алгоритма AdaBoost, который также строит ансамбль из слабых моделей, но оптимизирует экспоненциальную функцию потерь. Градиентный бустинг же позволил использовать произвольные дифференцируемые функции потерь, такие как квадратичная ошибка для регрессии или логистическая функция для классификации. Эта гибкость сделала градиентный бустинг одним из самых мощных методов для решения задач прогнозирования.

Объяснение с лекции К.В.Воронцова

Рассмотрим уже пройденный линейный ансамбль базовых алгоритмов b_t из семейства \mathcal{B} :

$$a_T(x) = \sum_{t=1}^T \alpha_t b_t(x), \quad x \in \mathbf{X}, \quad b_t : \mathbf{X} \rightarrow \mathbb{R}, \quad \alpha_t \in \mathbb{R}^+.$$

Эвристика: обучаем a_T , b_T при фиксированных предыдущих.

Критерий качества с заданной **гладкой** функцией потерь $L(b, y)$:

$$Q(\alpha, b; X^\ell) = \sum_{i=1}^{\ell} L \left(\sum_{t=1}^{T-1} \alpha_t b_t(x_i) + \alpha b(x_i), y_i \right) \rightarrow \min_{\alpha, b}.$$

Цель построить следующий алгоритм $b(x_i)$ и подобрать к нему α , на основе предыдущих t алгоритмов $\alpha_t b_t(x_i)$. Будем называть $\alpha_t b_t(x_i)$ - текущее приближение, и $\alpha b(x_i)$ - следующее приближение.

Можем рассматривать это как минимизацию в другом пространстве: $Q(f) \rightarrow \min$, $f \in \mathbb{R}^\ell$. Предположим, что в этом пространстве мы можем производить градиентный спуск, тогда выбор приближения будет выглядеть так:

$$\begin{aligned} a_{0,i} &:= \text{начальное приближение,} \\ a_{T,i} &:= a_{T-1,i} - \alpha g_i, \quad i = 1, \dots, \ell, \\ g_i &= L'_f(a_{T-1,i}, y_i) \quad (\text{компоненты вектора градиента}), \\ \alpha &= \text{градиентный шаг.} \end{aligned}$$

Это эквивалентно добавлению одного базового алгоритма:

$$a_{T,i} := a_{T-1,i} + \alpha b(x_i), \quad i = 1, \dots, \ell.$$

Идея: найти такой базовый алгоритм $b_T \in \mathcal{B}$, чтобы вектор $(b_T(x_i))_{i=1}^\ell$ аппроксимировал вектор антиградиента $(-g_i)_{i=1}^\ell$:

$$b_T := \arg \min_{b \in \mathcal{B}} \sum_{i=1}^{\ell} (b(x_i) + g_i)^2.$$

Алгоритм градиентного бустинга

Вход: обучающая выборка X^ℓ ; параметр T .

Выход: базовые алгоритмы и их веса $\alpha_t b_t$, $t = 1, \dots, T$.

Инициализация:

$$a_{0,i} := 0, \quad i = 1, \dots, \ell.$$

Для всех $t = 1, \dots, T$:

- Базовый алгоритм, приближающий антиградиент:

$$b_t := \arg \min_{b \in \mathcal{B}} \sum_{i=1}^{\ell} (b(x_i) + L'(a_{t-1,i}, y_i))^2.$$

- Задача одномерной минимизации:

$$\alpha_t := \arg \min_{\alpha > 0} \sum_{i=1}^{\ell} L(a_{t-1,i} + \alpha b_t(x_i), y_i).$$

- Обновление вектора значений на объектах выборки:

$$a_{t,i} := a_{t-1,i} + \alpha_t b_t(x_i), \quad i = 1, \dots, \ell.$$

Каждый следующий базовый алгоритм обучается так, чтобы по возможности исправить ошибки предыдущих алгоритмов.

Классическая аналогия с гольфом

Представьте себе игру в гольф: игрок должен довести мяч до лунки. Первый удар обычно самый сильный и приблизительно доставляет мяч в район цели. Однако этого недостаточно, чтобы загнать мяч в лунку. Следующие удары — более точные и тонкие — корректируют траекторию, пока мяч не окажется в нужной точке.

Градиентный бустинг работает по аналогичному принципу. Изначально модель делает "грубое" предсказание (аналог первого удара в гольфе). Затем на каждом последующем шаге слабые модели "подталкивают" результат в сторону оптимума, исправляя ошибки предыдущих шагов. Эти корректировки можно рассматривать как попытки уменьшить расстояние между текущим предсказанием и истинным ответом (целевой меткой). Эта аналогия помогает интуитивно понять, как небольшие шаги градиентного спуска приближают модель к минимизации функции потерь.

Применимость градиентного бустинга

Градиентный бустинг применяется в широком спектре задач машинного обучения благодаря своей гибкости и высокой эффективности. Среди основных областей применения можно выделить:

- **Классификация:** задачи, где нужно распределить объекты по классам, например, выявление мошеннических транзакций, прогнозирование оттока клиентов или медицинская диагностика.
- **Регрессия:** задачи, где нужно предсказать численное значение, такие как прогнозирование спроса, цены на рынке или уровня загрязнения воздуха.
- **Ранжирование:** задачи, где важно упорядочить объекты по степени их важности, например, поисковые системы или рекомендательные системы.
- **Обработка категориальных данных:** современные реализации, такие как CatBoost, значительно упростили работу с категориальными признаками, что делает градиентный бустинг полезным для анализа данных с такими особенностями.

Благодаря своей способности работать с разнородными признаками, поддержке пользовательских функций потерь и встроенным механизмам борьбы с переобучением, градиентный бустинг стал основным инструментом для построения производительных моделей в индустрии.

Задачи для практики

Задача 1: Оптимизация базового алгоритма На шаге t градиентного бустинга известны антиградиенты $-g_i$ для объектов x_i . Пусть $g_i = \hat{y}_{t-1}(x_i) - y_i$. Требуется построить базовый алгоритм $b_t(x)$, который минимизирует выражение:

$$\sum_{i=1}^{\ell} (b_t(x_i) + g_i)^2.$$

Найдите оптимальное значение $b_t(x_i)$ для каждого объекта x_i .

Решение:

Развернем выражение:

$$\sum_{i=1}^{\ell} (b_t(x_i) + g_i)^2 = \sum_{i=1}^{\ell} b_t^2(x_i) + 2b_t(x_i)g_i + g_i^2.$$

Оптимизируем по $b_t(x_i)$, приравняв производную к нулю:

$$\frac{\partial}{\partial b_t(x_i)} (b_t^2(x_i) + 2b_t(x_i)g_i) = 2b_t(x_i) + 2g_i = 0.$$

Отсюда:

$$b_t(x_i) = -g_i.$$

Ответ: $b_t(x_i) = -g_i$.

Задача 2: Влияние параметра α В градиентном бустинге используется коэффициент α_t , определяющий, насколько сильно предсказания обновляются на каждом шаге:

$$\hat{y}_t(x_i) = \hat{y}_{t-1}(x_i) + \alpha_t b_t(x_i).$$

Объясните, как α_t влияет на процесс обучения и какие последствия имеет слишком маленькое или слишком большое значение α_t .

Решение:

Параметр α_t регулирует величину обновлений предсказаний:

- Если α_t слишком **маленькое**, обновления будут незначительными. Это может привести к медленной сходимости модели, так как процесс обучения станет более долгим.
- Если α_t слишком **большое**, модель будет стремиться к переобучению. Это связано с тем, что большие обновления могут чрезмерно акцентировать внимание на текущих ошибках, ухудшая обобщающую способность модели.

На практике α_t подбирается либо эмпирически, либо используется небольшой фиксированный шаг (например, 0.1), чтобы контролировать обучение и избежать резких изменений.

Ответ: α_t определяет баланс между скоростью обучения и устойчивостью модели. Малые значения α_t обеспечивают стабильность, большие — ускоряют обучение, но могут привести к переобучению.

Задача 3: Отрицательные значения на тесте Может ли модель градиентного бустинга, обученная на выборке с исключительно положительными значениями признаков и таргетов (как в обучении, так и на валидации), предсказать отрицательные значения на тестовых данных?

Решение:

Да, такая ситуация возможна. Градиентный бустинг на каждом шаге обучает базовые алгоритмы $b_t(x)$, которые могут принимать как положительные, так и отрицательные значения. Если сумма весов $\alpha_t b_t(x)$ на каком-то объекте приводит к уменьшению значения \hat{y} ниже нуля, результатом будет отрицательное предсказание.

Пример: если текущие предсказания $\hat{y}_{t-1}(x)$ высоки, а таргет y ниже текущих предсказаний, градиентный спуск может добавить отрицательные значения, чтобы компенсировать разницу.

Вывод: Даже если данные для обучения содержат только положительные значения, итоговые предсказания могут быть отрицательными. Это связано с тем, что градиентный бустинг не ограничивает диапазон выходных значений модели.

Полезные ссылки

- Параграф из учебника ШАДа про градиентный бустинг — ШАД.
- Популярный источник с несколькими вариациями объяснения — explained.ai.

CatBoost

CatBoost (сокращение от Categorical Boosting) представляет собой современный алгоритм машинного обучения, разработанный компанией Яндекс. Он создан для эффективного анализа табличных данных, особенно когда в них присутствуют значимые категориальные признаки. Основной механизм работы CatBoost — градиентный бустинг на деревьях решений.

Основные мотивации CatBoost

CatBoost стремится решить две основные проблемы, возникающие в задачах машинного обучения:

1. **Обработка категориальных признаков** с большим числом редких значений, таких как пользователь, регион, город, рекламодатель, магазин и другие. Эти типы данных часто встречаются в реальных приложениях и требуют специального подхода для эффективной обработки.
2. **Избегание переобучения в градиентах**, также известного как смещенность или *target leakage*, то есть: $g_i = \mathcal{L}'(a_{t-1}(x_i), y_i)$ вычисляются в тех же точках x_i , по которым ансамбль $a_{t-1}(x)$ обучался аппроксимировать y_i ;

Идея: для получения несмещенных оценок на каждом объекте x_i будем хранить и дообучать ансамбль бех этого объекта. Но ведь тогда имеем $O(n)$ выборок? На самом деле можно реализовать алгоритм с помощью $O(\log(n))$ выборок.

Упорядоченный бустинг

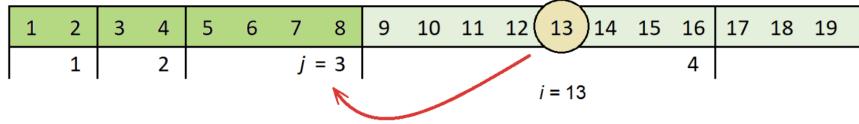
Ordered boosting решает проблему смещенности, прием взят с онлайн-алгоритмов.

Идеи:

- вычислять g_i по модели a_{t-1} , которая не обучалась на x_i ;
- строить обучающие подвыборки удваивающейся длины;
- построить много таких случайно перемешанных выборок.

Обозначения:

- $\sigma_1, \dots, \sigma_s$ — случайные перестановки выборки X^ℓ ;
- X^{rj} — подвыборка первых 2^j объектов из $\sigma_r(X^\ell)$;
- $a_t^{rj}(x)$ — ансамбль-полуфабрикат, обученный по X^{rj} ;
- $g_{ti}^r = -\mathcal{L}'(a_{t-1}^{rj}(x_i), y_i)$ — антиградиент в точке (x_i, y_i) для ансамбля a_{t-1}^{rj} , который по ней не обучался, $j = \lfloor \log_2(i-1) \rfloor$.



Модифицированный алгоритм градиентного бустинга в CatBoost

1. Сгенерировать случайные перестановки $\sigma_0, \sigma_1, \dots, \sigma_s$;
2. для всех $t = 1, \dots, T$:
 - (a) выбрать перестановку σ_r случайно из $\sigma_1, \dots, \sigma_s$;

$$g_{ti}^r := -\mathcal{L}'(a_{t-1}^{rj}(x_i), y_i) \quad \text{— несмешённый антиградиент;}$$

$$b_t := \arg \min_b \sum_{i=1}^{\ell} (b(x_i) - g_{ti}^r)^2;$$

для всех деревьев b_t^{rj} , $r = 1, \dots, s$, $2^j \leq \ell$:

- скопировать общую для них структуру дерева из b_t ;
 - вычислить в листах b_t^{rj} средние по $\{g_{ti}^r : x_i \in X^{rj}\}$;
- (b) вычислить в листах b_t средние по $\{g_{ti}^0 : x_i \in X^{0j}\}$;
 - (c) **GB part:** вычислить α_t и обновить $a_{t,i} := a_{t-1,i} + \alpha_t b_t(x_i)$;

Обработка категориальных фичей

Пусть V — множество значений признака $f(x)$.

Стандартные методы либо громоздкие, либо переобучаются:

- бинаризация (one-hot encoding): $b_v(x) = [f(x) = v]$;
- группирование (кластеризация) значений (LightGBM);
- статистика по целевому признаку - Target Statistics (TS):

$$\tilde{f}(x_i) = \frac{\sum_{k=1}^{\ell} [f(x_k) = f(x_i)] y_k + \gamma p}{\sum_{k=1}^{\ell} [f(x_k) = f(x_i)] + \gamma}$$

CatBoost:

- статистика TS вычисляется по перестановкам X^{rj} :

$$\tilde{f}(x_i) = \frac{\sum_{x_k \in X^{rj}} [f(x_k) = f(x_i)] y_k + \gamma p}{\sum_{x_k \in X^{rj}} [f(x_k) = f(x_i)] + \gamma}, \quad j = \lfloor \log_2(i - 1) \rfloor$$

- конъюнкции категориальных признаков создаются «налёту» в процессе построения деревьев.

Особенности построения деревьев в CatBoost (Oblivious Decision Tree)

Одной из ключевых особенностей алгоритма CatBoost является инновационный подход к построению деревьев решений. Деревья в CatBoost формируются по слоям, придерживаясь принципа: *все вершины одного уровня имеют одинаковый предикат*. Это означает, что на каждом уровне дерева используется идентичный сплит для всех его вершин. Данный метод позволяет избавиться от сложных ветвлений в коде инференса модели и вместо этого использовать более эффективные битовые операции. Такое усовершенствование существенно ускоряет применение модели, особенно при работе с батчами данных.

Дерево глубины H , $D_v = \{0, 1\}$, для всех узлов уровня h условие ветвления $f_h(x)$ одинаково; на уровне h ровно 2^{h-1} вершин, X делится на 2^H ячеек.

Классификатор задаётся *таблицей решений* $B : \{0, 1\}^H \rightarrow Y$:

$$a(x) = B(f_1(x), \dots, f_H(x)).$$

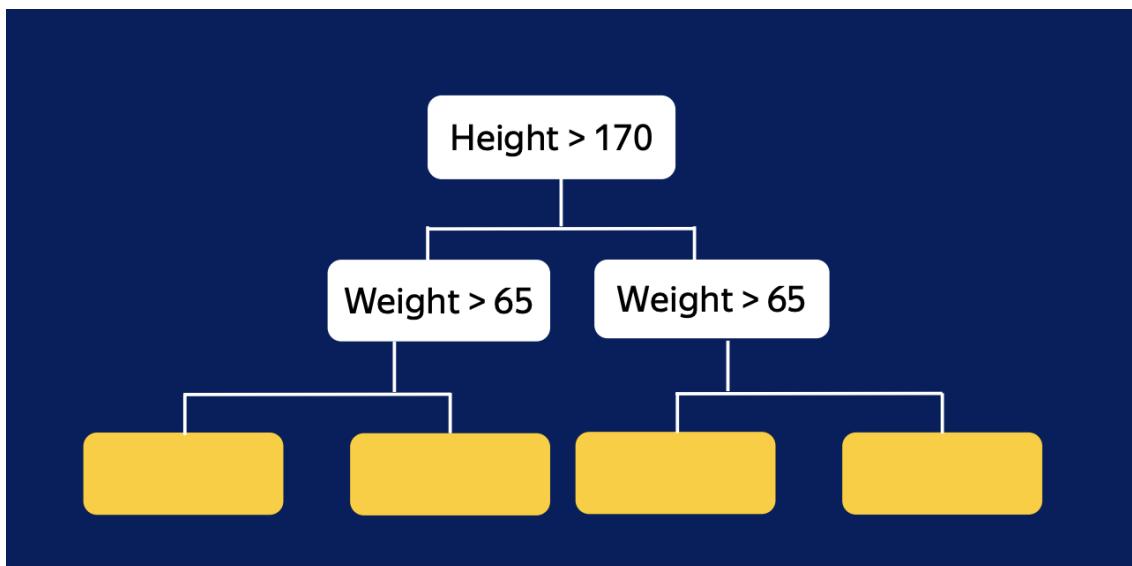


Рис. 15.1. Пример сплита в слоях дерева в CatBoost

Кроме того, этот способ служит мощным средством регуляризации, обеспечивающим устойчивость модели к переобучению. Также данный подход позволяет всегда строить полные бинарные деревья (аналогично XGBoost, с тем различием, что CatBoost создаёт их даже в тех случаях, когда на некоторые поддеревья не попадает ни один объект из обучающей выборки). Основным критерием остановки процесса является ограничение на глубину деревьев.

Алгоритм обучения Oblivious Decision Tree

Вход: выборка X^ℓ ; множество признаков F ; глубина дерева H ;

Выход: признаки f_h , $h = 1, \dots, H$ таблица $B : \{0, 1\}^H \rightarrow Y$;

Для всех $h = 1, \dots, H$:

предикат с максимальным выигрышем определённости:

$$f_h := \arg \max_{f \in \text{bin}\{F\}} \text{Gain}(f_1, \dots, f_{h-1}, f);$$

$$\mathcal{B}(\beta) := \Phi(U_{H\beta}), \text{ где } \Phi(U) = \text{avg}\{g_{ti}^r : x_i \in U\};$$

$U_{h\beta} = \{x_i \in X^\ell : f_s(x_i) = \beta_s, s = 1 \dots h\}$ — выборка объектов x_i , дошедших до вершины

Выигрыш от ветвления на уровне h по всей выборке X^ℓ :

$$\text{Gain}(f_1, \dots, f_h) = \Phi(X^\ell) - \sum_{\beta \in \{0, 1\}^h} \frac{|U_{h\beta}|}{\ell} \Phi(U_{h\beta})$$

Основные преимущества CatBoost:

- **Эффективная работа с категориальными признаками.** CatBoost способен автоматически обрабатывать категориальные данные без необходимости их предварительного кодирования, такого как one-hot encoding.
- **Интегрированная обработка пропусков.** Алгоритм самостоятельно справляется с отсутствующими значениями.
- **Противодействие переобучению.** CatBoost применяет различные техники для предотвращения переобучения модели, включая мощные механизмы регуляризации и усреднение.
- **Высокая скорость и производительность.** В CatBoost реализованы множественные оптимизации, ускоряющие процесс обучения и

предсказания. Он поддерживает многопоточную обработку и эффективно использует память.

- **Стабильность и воспроизводимость результатов.** Алгоритм гарантирует стабильные результаты даже при изменении порядка поступления входных данных, что важно для практического применения в реальных задачах.

Задача 1

Предположим, что у вас есть обучающая выборка из 8 объектов. Покажите, как будут сформированы подвыборки X^{rj} для объекта x_5 в одной из случайных перестановок, и какая модель будет использоваться для вычисления градиента на этом объекте.

Решение:

Используем одну случайную перестановку σ , которая для простоты совпадает с исходным порядком объектов: $\sigma = (x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8)$.

1. Определение значения j . Позиция объекта x_5 в перестановке: $i = 5$
Вычисляем значение j :

$$j = \lfloor \log_2(i - 1) \rfloor = \lfloor \log_2(5 - 1) \rfloor = \lfloor \log_2(4) \rfloor = 2$$

2. Формирование подвыборки X^{rj} :

Вычисляем размер подвыборки: $2^j = 2^2 = 4$

Подвыборка X^{rj} содержит первые $2^j = 4$ объекта из перестановки σ :

$$X^{rj} = \{x_1, x_2, x_3, x_4\}$$

Заметим, что объект x_5 не входит в эту подвыборку.

3. Выбор модели для вычисления градиента на x_5 :

Модель a_{t-1}^{rj} была обучена на подвыборке $X^{rj} = \{x_1, x_2, x_3, x_4\}$

Поскольку $x_5 \notin X^{rj}$, модель a_{t-1}^{rj} не была обучена на объекте x_5 .

Антиградиент на объекте x_5 вычисляется как:

$$g_{t5}^r = -\mathcal{L}' \left(a_{t-1}^{rj}(x_5), y_5 \right)$$

где a_{t-1}^{rj} — модель, обученная на подвыборке $\{x_1, x_2, x_3, x_4\}$.

Объект x_i	$f(x_i)$	y_i
x_1	A	1
x_2	B	0
x_3	A	1
x_4	C	0
x_5	B	1
x_6	A	0

Задача 2

Опишите, как CatBoost вычисляет статистики по целевой переменной для категориальных признаков на примере:

Дано:

- Перестановка объектов: $x_1, x_2, x_3, x_4, x_5, x_6$
- Значения признака $f(x_i)$ и целевой переменной y_i
- Параметры: $\gamma = 1, p = 0,5$.

Найти: $\tilde{f}(x_5)$

Решение:

1. Определяем параметр j :

$$i = 5 \rightarrow j = \lfloor \log_2(i - 1) \rfloor = \lfloor \log_2(4) \rfloor = 2$$

2. Формируем подвыборку X^{rj} :

Количество объектов в подвыборке: $2^j = 2^2 = 4$. Значит, $X^{rj} = \{x_1, x_2, x_3, x_4\}$.

3. Выбираем объекты с $f(x_k) = f(x_5) = B$:

Совпадающие объекты $= x_2$, так как $f(x_2) = B$.

4. Получаем значения y_k для совпадающих объектов: $y_2 = 0$.

5. Вычисляем сумму в числителе и знаменателе:

- Числитель: $\sum_{x_k \in X^{rj}} [f(x_k) = f(x_5)] y_k = y_2 = 0$
- Знаменатель: $\sum_{x_k \in X^{rj}} [f(x_k) = f(x_5)] = 1$

6. Вычисляем $\tilde{f}(x_5)$:

$$\tilde{f}(x_5) = \frac{\sum_{x_k \in X^{rj}} [f(x_k) = f(x_5)] y_k + \gamma p}{\sum_{x_k \in X^{rj}} [f(x_k) = f(x_5)] + \gamma} = \frac{0 + 1 \times 0,5}{1 + 1} = \frac{0,5}{2} = 0,25$$

Ответ: $\tilde{f}(x_5) = 0,25$

Задача 3

CatBoost использует особый тип деревьев решений, называемых небрежными деревьями (Oblivious Decision Trees), где на каждом уровне все узлы используют одинаковый предикат для разделения данных. Опишите основные преимущества использования обливиозных деревьев в CatBoost.

Решение:

1. Ускорение инференса:

- Благодаря симметричной структуре дерева и одинаковым предикатам на каждом уровне, вычисления при предсказании можно оптимизировать.
- Использование битовых векторов и операций позволяет быстро определять путь по дереву и получать предсказание из таблицы решений.

2. Оптимизация кода:

- Упрощённая структура дерева снижает сложность кода и количество условных операторов.
- Это приводит к более эффективному использованию процессорного конвейера и кэшей.

3. Регуляризация и устойчивость к переобучению:

- Ограничение модели фиксированными предикатами на уровне уменьшает её сложность.
- Это служит формой регуляризации, помогая предотвратить переобучение на обучающей выборке.

4. Предсказуемость и воспроизведимость:

- Симметричная структура обеспечивает стабильность модели при различных изменениях в данных.
- Лёгкость воспроизведения результатов в различных средах и на различных устройствах.

5. Совместимость с обработкой категориальных признаков:

- Обливиозные деревья хорошо интегрируются с методами обработки категориальных признаков в CatBoost.
- Позволяют эффективно использовать конъюнкции (сочетания) признаков для повышения качества модели.

LightGBM

Анализ сложности градиентного бустинга над решающими деревьями

Самая вычислительно сложная часть градиентного бустинга над решающими деревьями (GBDT) является поиск наилучшей точки разбиения. В LightGBM используется **histogram-based** подход. Он заменяет истинные значения непрерывных признаков на группы ("бины") и для каждого признака строит гистограмму, где подсчитываются суммы весов для каждого бина. Вычислительная сложность в таком случае оказывается равной $\mathcal{O}(\#bin \cdot \#features)$, что оказывается гораздо выгоднее алгоритма предварительной сортировки ($\mathcal{O}(\#data \cdot \#features)$), поскольку обычно $\#bin \ll \#data$.

Gradient-based One-Side Sampling (GOSS)

Идея заключается в том, чтобы придумать способ, с помощью которого мы смогли бы находить наилучшее разбиение, не просматривая всю выборку. Аналогично алгоритму AdaBoost мы хотим присвоить каждому элементу выборки "вес чтобы акцентировать внимание модели на них. В качестве этого веса может выступать градиент элемента. Так, например, данные с маленькими (по абсолютному значению) градиентами вносят малый вклад в ошибку, поэтому необязательно использовать их все.

GOSS предлагает следующий алгоритм сэмплирования: GOSS сначала сортирует данные в убывающем порядке по абсолютному значению градиентов и выбирает топ $a \times 100\%$ элементов. Далее он рандомно сэмплирует $b \times 100\%$ элементов. Далее GOSS умножает часть с маленькими градиентами на константу $\frac{1-a}{b}$ при вычислении прироста информации.

Более строго GBDT пытается выучить отображение из исходного пространства \mathcal{X}^s в пространство градиентов \mathcal{G} . Предположим, мы имеем n одинаковых независимо распределенных элементов $\{x_1, x_2, \dots, x_n\}$, $x_i \in \mathcal{X}^s \forall i = \overline{1, n}$. Для каждой итерации градиентного бустинга определим $\{g_1, g_2, \dots, g_n\}$ - антиградиенты loss-функции по отношению к текущему предсказанию модели.

Определение. Пусть \mathcal{X} - обучающая выборка в текущей ноде решающего дерева. Тогда прирост информации при разделении признака j по значению d для этой ноды определяется как

$$V_{j|\mathcal{X}}(d) = \frac{1}{|\mathcal{X}|} \left(\frac{\left(\sum_{x_i \in \mathcal{X}: x_{ij} \leq d} g_i \right)^2}{n_l^j(d)} + \frac{\left(\sum_{x_i \in \mathcal{X}: x_{ij} > d} g_i \right)^2}{n_r^j(d)} \right)$$

где $n_l^j(d) = |x_i \in \mathcal{X} : x_{ij} \leq d|$, $n_r^j(d) = |x_i \in \mathcal{X} : x_{ij} > d|$. Для каждого признака j алгоритм выбирает $d_j^* = \arg \max_d V_j(d)$ и вычисляет наибольший прирост информации $V_j(d_j^*)$. Далее данные делятся по признаку j^* и по значению d_j^* на правое и левое поддервья. В предложенном GOSS алгоритме выберем топ $a \times 100\%$ элементов с наибольшим градиентом (по абсолютному значению) и обозначим это множество A . Далее из \bar{A} случайно выберем $b \times |\bar{A}|$ элементов и обозначим B . Тогда оценка прироста информации имеет вид

$$\tilde{V}_j(d) = \frac{1}{|\mathcal{X}|} \left(\frac{\left(\sum_{x_i \in A_l} g_i + \frac{1-a}{b} \sum_{x_i \in B_l} g_i \right)^2}{n_l^j(d)} + \frac{\left(\sum_{x_i \in A_r} g_i + \frac{1-a}{b} \sum_{x_i \in B_r} g_i \right)^2}{n_r^j(d)} \right),$$

где $A_l = \{x_i \in A : x_{ij} < d\}$, $A_r = \{x_i \in A : x_{ij} > d\}$, $B_l = \{x_i \in B : x_{ij} \leq d\}$, $B_r = \{x_i \in B : x_{ij} > d\}$.

То есть в GOSS мы используем оценку прироста информации $\tilde{V}_j(d)$, рассчитанную по меньшему датасету, нежели точную $V_j(d)$. При этом согласно утверждению следующей теоремы, используя оценку $\tilde{V}_j(d)$ вместо $V_j(d)$ мы не сильно проигрывает в точности:

Теорема. Обозначим ошибку аппроксимации GOSS как $\varepsilon(d) = |\tilde{V}_j(d) - V_j(d)|$ и $\bar{g}_l^j(d) = \frac{\sum_{x_i \in \mathcal{X}_l} |g_i|}{n_l^j(d)}$, $\bar{g}_r^j(d) = \frac{\sum_{x_i \in \mathcal{X}_r} |g_i|}{n_r^j(d)}$. Тогда с вероятностью хотя бы $1 - \delta$ имеет место оценка

$$\varepsilon(d) \leq C_{a,b}^2 \ln \frac{1}{\delta} \cdot \max \left\{ \frac{1}{n_l^j(d)}, \frac{1}{n_r^j(d)} \right\} + 2DC_{a,b} \sqrt{\frac{\ln \frac{1}{\delta}}{n}},$$

где $C_{a,b} = \frac{1-a}{\sqrt{b}} \max_{x_i \in \bar{A}} |g_i|$ и $D = \max \left\{ \bar{g}_l^j(d), \bar{g}_r^j(d) \right\}$.

Из данной теоремы следует асимптотическая оценка GOSS $\mathcal{O} \left(\frac{1}{n_l^j(d)} + \frac{1}{n_r^j(d)} + \frac{1}{\sqrt{n}} \right)$. Поэтому если разбиения будут не слишком несбалансированными (то есть $n_l^j(d) \geq \mathcal{O}(\sqrt{n})$ и $n_r^j(d) \geq \mathcal{O}(\sqrt{n})$), то общая асимптотика будет составлять $\mathcal{O} \left(\frac{1}{\sqrt{n}} \right) \xrightarrow{n \rightarrow \infty} 0$. То есть при большом количестве данных приближение будет достаточно точным.

Обобщая способность GOSS будет также близка к алгоритму с "честным" приростом информации. $\varepsilon_{\text{gen}}^{\text{GOSS}}(d) = |\tilde{V}_j(d) - V_*(d)| \leq |\tilde{V}_j(d) - V_j(d)| + |V_j(d) - V_*(d)| = \varepsilon_{\text{GOSS}}(d) + \varepsilon_{\text{gen}}(d)$.

Exclusive Feature Bundling (EFB)

Зачастую, датасеты большой размерности являются сильно разреженными, возникает мысль, что возможно уменьшить количество признаков, не сильно теряя при этом в точности. В частности, в разреженном пространстве зачастую встречаются взаимоисключающие признаки, то есть признаки, которые не могут быть ненулевыми одновременно. Такие признаки можно объединить в один (exclusive feature bundle). Соответственно, гистограмма из подпункта 1.1, построенная на этих объединенных признаках имеет асимптотику $\mathcal{O}(\#data \times \#bundle)$, вместо $\mathcal{O}(\#data \times \#feature)$, что значительно быстрее при $\#bundle \ll \#feature$.

Сведем задачу разделения на такие bundle к задаче раскраски графа (NP-Hard), признаки будем считать вершинами и ребро будем проводить между признаками, которые не являются взаимоисключающими. Точного полиномиального решения на текущий момент не существует, тем не менее задачу можно достаточно хорошо решить с помощью жадного алгоритма. Более того, обычно существует довольно много признаков, которые хотя и не являются на 100% взаимоисключающими, но редко принимают ненулевые значения одновременно. Если наш алгоритм сможет разрешить небольшую долю конфликтов, мы сможем получить еще меньшее количество bundles признаков и еще больше повысить эффективность вычислений. Такая процедура повлияет на точность при обучении не более чем $\mathcal{O}\left([(1 - \gamma)n]^{\frac{2}{3}}\right)$, где γ - максимальная частота конфликтов в каждом bundle.

Сформулируем итоговый алгоритм. Строим граф признаков с весами на ребрах, соответствующими общему числу конфликтов между признаками. Далее сортируем признаки по количеству соседей, с которыми есть конфликты в убывающем порядке, наконец, для каждого признака из упорядоченного списка проверяется, можно ли добавить признак в существующие bundles с минимальным конфликтом (контролируется параметром γ), иначе создается новый bundle. Общая сложность алгоритма - $\mathcal{O}(\#feature^2)$.

Далее нам надо понять, как правильно объединять признаки в один bundle, то есть по значению признака в bundle мы должны уметь определить исходное значение объединенных признаков. Это может быть сделано с помощью добавления смещения к признакам. Например, признак A принимает значения из диапазона $[0, 10)$, а признак B - из $[0, 20)$. Добавим смещение признаку B , теперь он принимает значения из диапазона $[10, 30)$. После объединения признаков A и B полученный признак будет принимать значения в диапазоне $[0, 30)$.

Задачи

1. Обычно в методах градиентного бустинга стараются строить неглубокие деревья, чтобы не переобучиться на обучающей выборке и ускорить время

обучения. Сохраняется ли это в LightGBM? Почему?

Ответ: Нет, это нарушается в LightGBM реализации градиентного бустинга. Дело в другой стратегии роста - leaf-wise growth. Глядя на функционал прироста информации видно, что каждый раз выборка делится на одну часть с большими градиентами и вторую часть, с маленькими градиентами. Данные с маленькими градиентами уже вносят малый вклад в ошибку и дальше разбивать их не так информативно. LightGBM на каждом шаге выбирает наиболее оптимальное разбиение, не обращая внимание на сбалансированность получающегося дерева, из-за чего зачастую получаются глубокие и несбалансированные деревья.

2. В GOSS при подсчете оценки прироста информации мы домножаем на множитель $\frac{1-a}{b}$. Зачем это нужно и что будет, если это домножение не производить?

Ответ: Если мы просто уменьшим количество данных с малыми градиентами, это исказит относительную важность этих объектов в функции потерь. Масштабирование градиентов увеличивает вклад выбранных объектов с малыми градиентами так, чтобы их совокупный эффект оставался пропорциональным их изначальной доле в данных. Если множитель не использовать, объекты с большими градиентами будут еще сильнее доминировать при построении дерева. Модель будет чрезмерно подстраиваться под сложные примеры, пренебрегая общей точностью и, скорее всего, ухудшая обобщающую способность.

3. В чем важность параметра `min_data_in_leaf` параметра в LightGBM, к чему могут привести его высокие и низкие значения? А параметра `num_leaves`?

Ответ:

- (a) `min_data_in_leaf` - отвечает за минимальное количество элементов выборки в каждом листе. Чем выше этот параметр, тем более модель устойчива к зашумленным данным. Слишком большие значения, однако, могут приводить к снижению чувствительности модели сложных зависимостей, ухудшая качество предсказания. Слишком малые значения параметра, напротив, позволяют выучить достаточно сложные зависимости, однако возрастает вероятность переобучения.
- (b) `num_leaves` - отвечает за максимальное количество листьев дерева. Аналогично предыдущему признаку регулирует сложность модели. Чем больше листьев - тем более сложные зависимости между признаками и целевой переменной может улавливать дерево, однако высока вероятность переобучиться. И напротив, чем меньше листьев, тем модель более устойчива к переобучению и к шума и тем хуже она справляется со сложными зависимостями.

Глава 16

Байесовская теория классификации

Задача 1

Теория

Допустим, у нас есть некоторая выборка, на которой линейные методы работают лучше решающих деревьев с точки зрения ошибки на контроле. Почему это так? Чем можно объяснить превосходство определённого метода обучения? Оказывается, ошибка любой модели складывается из трёх факторов: сложности самой выборки, схожести модели с истинной зависимостью ответов от объектов в выборке, и богатства семейства, из которого выбирается конкретная модель. Между этими факторами существует некоторый баланс, и уменьшение одного из них приводит к увеличению другого. Такое разложение ошибки носит название разложения на смещение и разброс, и его формальным выводом мы сейчас займёмся.

Пусть задана выборка $X = (x_i, y_i)_{i=1}^n$ с вещественными ответами $y_i \in \mathbb{R}$ (рассматриваем задачу регрессии). Будем считать, что на пространстве всех объектов и ответов $X \times Y$ существует распределение $p(x, y)$, из которого сгенерирована выборка X и ответы на ней.

Рассмотрим квадратичную функцию потерь

$$L(y, a) = (y - a(x))^2$$

и соответствующий ей среднеквадратичный риск

$$R(a) = \mathbb{E}_{x,y} [(y - a(x))^2] = \int_X \int_Y p(x, y)(y - a(x))^2 dx dy.$$

Данный функционал усредняет ошибку модели в каждой точке пространства x и для каждого возможного ответа y , причём вклад пары (x, y) , по сути, пропорционален вероятности получить её в выборке $p(x, y)$. Разумеется, на практике мы не можем вычислить данный функционал, поскольку распределение $p(x, y)$ неизвестно. Тем не менее, в теории он позволяет измерить качество модели на всех возможных объектах, а не только на наблюдённой выборке.

Задание

Покажите, что минимум среднеквадратичного риска достигается на функции, возвращающей условное математическое ожидание ответа при фиксированном объекте.

$$a_*(x) = \mathbb{E}[y \mid x] = \int_Y yp(y \mid x) dy = \arg \min_a R(a).$$

Иными словами покажите, что мы должны провести «взвешенное голосование» по всем возможным ответам, при этом веса ответа равны апостериорной вероятности.

Решение

Преобразуем функцию потерь:

$$\begin{aligned} L(y, a(x)) &= (y - a(x))^2 = (y - \mathbb{E}(y \mid x) + \mathbb{E}(y \mid x) - a(x))^2 = \\ &= (y - \mathbb{E}(y \mid x))^2 + 2(y - \mathbb{E}(y \mid x))(\mathbb{E}(y \mid x) - a(x)) + (\mathbb{E}(y \mid x) - a(x))^2. \end{aligned}$$

Подставляя её в функционал среднеквадратичного риска, получаем:

$$\begin{aligned} R(a) &= \mathbb{E}_{x,y}[L(y, a(x))] = \\ &= \mathbb{E}_{x,y}[(y - \mathbb{E}(y \mid x))^2] + \mathbb{E}_{x,y}[(\mathbb{E}(y \mid x) - a(x))^2] + 2\mathbb{E}_{x,y}[(y - \mathbb{E}(y \mid x))(\mathbb{E}(y \mid x) - a(x))]. \end{aligned}$$

Разберёмся сначала с последним слагаемым. Перейдём от матожидания $\mathbb{E}_{x,y}[f(x, y)]$ к цепочке матожиданий:

$$\mathbb{E}_x \mathbb{E}_y[f(x, y) \mid x] = \int_X \left(\int_Y f(x, y) p(y \mid x) dy \right) p(x) dx$$

и заметим, что величина $(\mathbb{E}(y \mid x) - a(x))$ не зависит от y , и поэтому её можно вынести за матожидание по y :

$$\begin{aligned} &\mathbb{E}_x \mathbb{E}_y [(y - \mathbb{E}(y \mid x))(\mathbb{E}(y \mid x) - a(x)) \mid x] = \\ &= \mathbb{E}_x ((\mathbb{E}(y \mid x) - a(x)) \mathbb{E}_y [(y - \mathbb{E}(y \mid x)) \mid x]) = \\ &= \mathbb{E}_x ((\mathbb{E}(y \mid x) - a(x)) (\mathbb{E}_y [y \mid x] - \mathbb{E}_y \mathbb{E}(y \mid x))) = \\ &= 0. \end{aligned}$$

Получаем, что функционал среднеквадратичного риска имеет вид:

$$R(a) = \mathbb{E}_{x,y}(y - \mathbb{E}(y \mid x))^2 + \mathbb{E}_{x,y}((\mathbb{E}(y \mid x) - a(x))^2).$$

От алгоритма $a(x)$ зависит только второе слагаемое, и оно достигает своего минимума, если $a(x) = \mathbb{E}(y \mid x)$. Таким образом, оптимальная модель регрессии для квадратичной функции потерь имеет вид:

$$a_*(x) = \mathbb{E}(y \mid x) = \int_Y yp(y \mid x) dy.$$

Что и требовалось показать.

Задача 2

Теория

Для того, чтобы построить идеальную функцию регрессии, необходимо знать распределение на объектах и ответах $p(x, y)$, что, как правило, невозможно. На практике вместо этого выбирается некоторый *метод обучения* $\mu : (\mathbb{X} \times \mathbb{Y})^\ell \rightarrow A$, который произвольной обучающей выборке ставит в соответствие некоторый алгоритм из семейства A . В качестве меры качества метода обучения можно взять усредненный по всем выборкам среднеквадратичный риск алгоритма, выбранного методом μ по выборке:

$$\begin{aligned}
L(\mu) &= \mathbb{E}_X \left[\mathbb{E}_{x,y} \left[(y - \mu(X)(x))^2 \right] \right] = \\
&= \int_{(\mathbb{X} \times \mathbb{Y})^\ell} \int_{\mathbb{X} \times \mathbb{Y}} (y - \mu(X)(x))^2 p(x, y) \prod_{i=1}^{\ell} p(x_i, y_i) dx dy dx_1 dy_1 \dots dx_\ell dy_\ell.
\end{aligned} \tag{1}$$

Здесь матожидание $\mathbb{E}_X[\cdot]$ берется по всем возможным выборкам $\{(x_1, y_1), \dots, (x_\ell, y_\ell)\}$ из распределения $\prod_{i=1}^{\ell} p(x_i, y_i)$.

Обратим внимание, что результатом применения метода обучения $\mu(X)$ к выборке X является модель, поэтому правильно писать $\mu(X)(x)$. Но это довольно громоздкая запись, поэтому будем везде дальше писать просто $\mu(X)$, но не будем забывать, что это функция, зависящая от объекта x .

Среднеквадратичный риск на фиксированной выборке X можно расписать как:

$$\mathbb{E}_{x,y} \left[(y - \mu(X))^2 \right] = \mathbb{E}_{x,y} \left[(y - \mathbb{E}[y | x])^2 \right] + \mathbb{E}_{x,y} \left[(\mathbb{E}[y | x] - \mu(X))^2 \right].$$

Задание

Подставим это представление в (1):

$$\begin{aligned}
L(\mu) &= \mathbb{E}_X \left[\mathbb{E}_{x,y} \left[(y - \mathbb{E}[y | x])^2 \right] + \mathbb{E}_{x,y} \left[(\mathbb{E}[y | x] - \mu(X))^2 \right] \right] = \\
&= \mathbb{E}_{x,y} \left[(y - \mathbb{E}[y | x])^2 \right] + \mathbb{E}_{x,y} \left[\mathbb{E}_X \left[(\mathbb{E}[y | x] - \mu(X))^2 \right] \right].
\end{aligned} \tag{2}$$

Преобразуем второе слагаемое:

$$\begin{aligned}
&\mathbb{E}_{x,y} \left[\mathbb{E}_X \left[(\mathbb{E}[y | x] - \mu(X))^2 \right] \right] = \\
&= \mathbb{E}_{x,y} \left[\mathbb{E}_X \left[(\mathbb{E}[y | x] - \mathbb{E}_X[\mu(X)] + \mathbb{E}_X[\mu(X)] - \mu(X))^2 \right] \right] = \\
&= \mathbb{E}_{x,y} \left[\mathbb{E}_X \left[(\mathbb{E}[y | x] - \mathbb{E}_X[\mu(X)])^2 \right] \right] + \mathbb{E}_{x,y} \left[\mathbb{E}_X \left[(\mathbb{E}_X[\mu(X)] - \mu(X))^2 \right] \right] + \\
&\quad + 2\mathbb{E}_{x,y} [\mathbb{E}_X [(\mathbb{E}[y | x] - \mathbb{E}_X[\mu(X)]) (\mathbb{E}_X[\mu(X)] - \mu(X))]].
\end{aligned} \tag{3}$$

Покажите, что последнее слагаемое обращается в нуль.

Решение

Покажем, что последнее слагаемое обращается в нуль:

$$\begin{aligned} \mathbb{E}_X [(\mathbb{E}[y | x] - \mathbb{E}_X [\mu(X)]) (\mathbb{E}_X [\mu(X)] - \mu(X))] &= \\ &= (\mathbb{E}[y | x] - \mathbb{E}_X [\mu(X)]) \mathbb{E}_X [\mathbb{E}_X [\mu(X)] - \mu(X)] = \\ &= (\mathbb{E}[y | x] - \mathbb{E}_X [\mu(X)]) [\mathbb{E}_X \mu(X) - \mathbb{E}_X \mu(X)] = \\ &= 0. \end{aligned}$$

Задача 3**Задание**

Используя результаты предыдущих задач и подставляя (3) в (2) получите выражение для $L(\mu)$, укажите слагаемые, отвечающие за *смещение, шум и разброс*.

Решение

Подставим выражение (3) в (2), учитывая результаты предыдущих задач:

$$L(\mu) = \underbrace{\mathbb{E}_{x,y} [(y - \mathbb{E}[y | x])^2]}_{\text{шум}} + \underbrace{\mathbb{E}_x [(\mathbb{E}_X[\mu(X)] - \mathbb{E}[y | x])^2]}_{\text{смещение}} + \underbrace{\mathbb{E}_x [\mathbb{E}_X [(\mu(X) - \mathbb{E}_X[\mu(X)])^2]]}_{\text{разброс}}$$

Рассмотрим подробнее компоненты полученного разложения ошибки. Первая компонента характеризует *шум (noise)* в данных и равна ошибке идеального алгоритма. Невозможно построить алгоритм, имеющий меньшую среднеквадратичную ошибку. Вторая компонента характеризует *смещение (bias)* метода обучения, то есть отклонение среднего ответа обученного алгоритма от ответа идеального алгоритма. Третья компонента характеризует *дисперсию (variance)*, то есть разброс ответов обученных алгоритмов относительно среднего ответа.

Задача 3. Проверьте, что если все признаки бинарные, то наивный байесовский классификатор с 2 классами эквивалентен логистической регрессии с фиксированными весами и найдите эти веса.

Линейный дискриминант Фишера

Линейный дискриминант Фишера в первоначальном значении — метод, определяющий расстояние между распределениями двух разных классов объектов или событий. Он может использоваться в задачах машинного обучения при статистическом (байесовском) подходе к решению задач классификации.

Предположим, что обучающая выборка удовлетворяет помимо базовых гипотез байесовского классификатора также следующим гипотезам:

- Классы распределены по нормальному закону.
- Матрицы ковариаций классов равны.

Такой случай соответствует наилучшему разделению классов по дискриминанту Фишера (в первоначальном значении). Тогда статистический подход приводит к линейному дискриминанту, и именно этот алгоритм классификации в настоящее время часто понимается под термином линейный дискриминант Фишера.

Введение

При некоторых общих предположениях байесовский классификатор сводится к формуле:

$$a(x) = \arg \max_{y \in Y} \lambda_y P_y p_y(x),$$

где Y — множество ответов (классов), x принадлежит множеству объектов X , P_y — априорная вероятность класса y , $p_y(x)$ — функция правдоподобия класса y , λ_y — весовой коэффициент (цена ошибки на объекте класса y).

При выдвижении всех указанных выше гипотез, кроме гипотезы о равенстве матриц ковариаций, данная формула принимает вид:

$$a(x) = \arg \max_{y \in Y} \left(\ln(\lambda_y P_y) - \frac{1}{2}(x - \mu_y)^T \Sigma_y^{-1} (x - \mu_y) - \frac{1}{2} \ln(\det \Sigma_y^{-1}) - \frac{n}{2} \ln(2\pi) \right),$$

где

$$\mu_y = \frac{1}{l_y} \sum_{\substack{i=1 \\ y_i=y}}^l x_i, \quad \Sigma_y = \frac{1}{l_y} \sum_{\substack{i=1 \\ y_i=y}}^l (x_i - \mu_y)(x_i - \mu_y)^T$$

— приближения вектора математического ожидания и матрицы ковариации класса y , полученные как оценки максимума правдоподобия, l — длина обучающей выборки, l_y — количество объектов класса y в обучающей выборке, $x \in \mathbb{R}^n$.

Данный алгоритм классификации является квадратичным дискриминантом. Он имеет ряд недостатков, одним из самых существенных из которых является плохая обусловленность или вырожденность матрицы ковариаций Σ_y при малом количестве обучающих элементов класса y , вследствие чего при обращении данной матрицы Σ_y^{-1} может получиться сильно искаженный результат, и весь алгоритм классификации окажется неустойчивым, будет работать плохо (возможна также ситуация, при которой обратная матрица Σ_y^{-1} вообще не будет существовать). Линейный дискриминант Фишера решает данную проблему.

Задача 1. Каковы преимущества и недостатки использования квадратичного дискриминантного анализа (QDA) по сравнению с линейным дискриминантным анализом (LDA) в задачах классификации?

Основная идея алгоритма

При принятии гипотезы о равенстве между собой ковариационных матриц алгоритм классификации принимает вид:

$$a(x) = \arg \max_{y \in Y} \left(\ln(\lambda_y P_y) - \frac{1}{2} \mu_y^T \Sigma^{-1} \mu_y + x^T \Sigma^{-1} \mu_y \right),$$

или

$$a(x) = \arg \max_{y \in Y} (\beta_y + x^T \alpha_y).$$

Простота классификации линейным дискриминантом Фишера — одно из достоинств алгоритма: в случае с двумя классами в двумерном признаковом пространстве разделяющей поверхностью будет прямая. Если классов больше

двух, то разделяющая поверхность будет кусочно-линейной. Но главным преимуществом алгоритма по сравнению с квадратичным дискриминантом является уменьшение эффекта плохой обусловленности ковариационной матрицы при недостаточных данных.

При малых l_y приближения

$$\Sigma_y = \frac{1}{l_y} \sum_{\substack{i=1 \\ y_i=y}}^l (x_i - \mu_y)(x_i - \mu_y)^T$$

дадут плохой результат, поэтому даже в тех задачах, где заведомо известно, что классы имеют различные формы, иногда бывает выгодно воспользоваться эвристикой дискриминанта Фишера и считать матрицы ковариаций всех классов одинаковыми. Это позволит вычислить некоторую "среднюю" матрицу ковариаций, используя всю выборку:

$$\Sigma = \frac{1}{l} \sum_{i=1}^l (x_i - \mu_{y_i})(x_i - \mu_{y_i})^T,$$

использование которой в большинстве случаев сделает алгоритм классификации более устойчивым.

Задача 2. Каковы основные предпосылки и ограничения линейного дискриминанта Фишера, и в каких случаях его применение может быть предпочтительнее по сравнению с квадратичным дискриминантом?

Выводы

Эвристика линейного дискриминанта Фишера является в некотором роде упрощением квадратичного дискриминанта. Она используется с целью получить более устойчивый алгоритм классификации. Наиболее целесообразно пользоваться линейным дискриминантом Фишера, когда данных для обучения недостаточно. Вследствие основной гипотезы, на которой базируется алгоритм, наиболее удачно им решаются простые задачи классификации, в которых по формам классы "похожи" друг на друга.

Процесс классификации линейным дискриминантом Фишера можно описать следующей схемой:

1. Обучение

- Оценивание математических ожиданий μ_y
- Вычисление общей ковариационной матрицы Σ и ее обращение

2. Классификация

- Использование формулы

$$a(x) = \arg \max_{y \in Y} \left(\ln(\lambda_y P_y) - \frac{1}{2} \mu_y^T \Sigma^{-1} \mu_y + x^T \Sigma^{-1} \mu_y \right)$$

Задача 3. Даны два класса объектов, представленные следующими данными:

- Класс 1: $X_1 = (2, 3), (3, 3), (2, 4)$
- Класс 2: $X_2 = (5, 6), (6, 5), (5, 7)$

Найдите линейный дискриминант Фишера

Ответ к задаче 1

- QDA лучше подходит для задач, где классы имеют разные дисперсии и формы, и когда доступно достаточно данных для надежной оценки параметров.
- LDA может быть предпочтительнее в случаях с ограниченным количеством данных или когда классы можно считать линейно разделимыми.

Ответ к задаче 2 Основные предпосылки линейного дискриминанта Фишера:

- Нормальность: Предполагается, что данные в каждом классе распределены нормально.
- Однородность дисперсий: Линейный дискриминант предполагает одинаковые матрицы ковариаций для всех классов.
- Линейная разделимость: Предполагается, что классы можно разделить линейной границей.

Ограничения:

- Если данные не удовлетворяют предпосылкам нормальности или однородности дисперсий, производительность линейного дискриминанта может значительно ухудшиться.
- Линейный дискриминант не может захватить сложные нелинейные зависимости между классами.

Когда предпочтительнее:

- Линейный дискриминант может быть предпочтительнее квадратичного в случаях, когда:
- Данные имеют высокую размерность и при этом имеют достаточно малое количество образцов (линейный подход менее подвержен переобучению).
- Классы действительно линейно разделимы или близки к этому.

Пример: В задачах распознавания лиц с использованием признаков (например, цветовые компоненты пикселей) линейный дискриминант может быть эффективным из-за высокой размерности данных и необходимости в простоте модели.

Ответ к задаче 3

1. Найдите средние векторы для каждого класса.

Средние векторы:

$$\mu_1 = \left(\frac{2+3+2}{3}, \frac{3+3+4}{3} \right) = (2.33, 3.33)$$

$$\mu_2 = \left(\frac{5+6+5}{3}, \frac{6+5+7}{3} \right) = (5.33, 6.00)$$

2. Вычислите матрицы дисперсии для каждого класса.

Для класса 1:

$$S_1 = \frac{1}{n_1 - 1} \sum_{i=1}^{n_1} (x_i - \mu_1)(x_i - \mu_1)^T$$

После вычислений получаем:

$$S_1 = (0.33 \ 0.33 \ 0.33 \ 0.67)$$

Для класса 2:

$$S_2 = \frac{1}{n_2 - 1} \sum_{i=1}^{n_2} (x_i - \mu_2)(x_i - \mu_2)^T$$

После вычислений получаем:

$$S_2 = (0.67 \ -0.33 \ -0.33 \ 0.67)$$

3. Найдите линейный дискриминант Фишера.

Сначала находим объединённую матрицу дисперсии:

$$S_W = S_1 + S_2 = (0.33 + 0.67 \ 0.33 - 0.33 \ 0.33 - 0.33 \ 0.67 + 0.67) = (1 \ 0 \ 0 \ 1.34)$$

Теперь находим весовой вектор w :

$$w = S_W^{-1}(\mu_1 - \mu_2) = (1 \ 0 \ 0 \ 1.34)^{-1} (2.33 - 5.33 \ 3.33 - 6) = (-3 \ -2.67)$$

Глава 17

Методы кластеризации

Критерии качества кластеризации

Давайте детально разберем основные метрики, используемые для оценки качества кластеризации данных.

Выбор подходящей метрики напрямую зависит от наличия или отсутствия предварительной разметки данных и от того, задано ли количество кластеров априори или оно подбирается в процессе кластеризации.

Критерии, не требующие разметки выборки

Среднее внутрикластерное расстояние

Название этой метрики говорит само за себя: она отражает среднее расстояние между всеми парами точек внутри одного кластера. Иными словами, мы подсчитываем сумму расстояний между всеми парами точек в каждом кластере и делим на общее количество таких пар.

Формула метрики выглядит так:

$$F_0 = \frac{\sum_{i=1}^n \sum_{j=i}^n \rho(x_i, x_j) I[a(x_i) = a(x_j)]}{\sum_{i=1}^n \sum_{j=i}^n I[a(x_i) = a(x_j)]}.$$

В формуле учитываются и пары вида (x_i, x_i) , что позволяет избежать неопределенности $\frac{0}{0}$ в случае, если кластер состоит всего из одной точки. Однако, иногда для упрощения вычислений суммирование ведется только по парам (x_i, x_j) , где $i < j$, при этом для случая одноточечных кластеров значение метрики полагается равным нулю.

Вычисление этого критерия достаточно трудоёмко, поэтому можно также ввести средний квадрат внутрикластерного расстояния, если нам известны представители, или центры масс, кластеров μ_k :

$$\Phi_0 = \frac{1}{nK} \sum_{k=1}^K \sum_{i=1}^n \rho(\mu_k, x_i)^2 I[a(x_i) = k].$$

Наша цель при кластеризации – получить максимально компактные кластеры, поэтому мы стремимся минимизировать значение этой метрики. Чем меньше среднее внутрикластерное расстояние, тем лучше.

Среднее межклластерное расстояние

В отличие от предыдущей метрики, среднее межклластерное расстояние оценивает среднее расстояние между точками из разных кластеров.

Формула выглядит следующим образом:

$$F_1 = \frac{\sum_{i=1}^n \sum_{j=i}^n \rho(x_i, x_j) I[a(x_i) \neq a(x_j)]}{\sum_{i=1}^n \sum_{j=i}^n I[a(x_i) \neq a(x_j)]}.$$

Здесь, наоборот, мы стремимся к максимизации этого значения. Чем больше расстояние между кластерами, тем лучше разделение.

Критерии, требующие разметки выборки

Следующие метрики требуют, чтобы мы заранее знали, к какому классу принадлежит каждый объект в наборе данных. Это позволяет сравнить результаты кластеризации с истинным распределением данных.

Мы будем обозначать кластеры, полученные в результате кластеризации, как $k \in \{1, \dots, K\}$, а истинные классы – как $c \in \{1, \dots, C\}$.

Гомогенность

Если у нас есть разметка, то можно свести задачу кластеризации к использованию методов классификации. Если размеченных данных достаточно много, то обучение классификатора – более подходящий подход. Однако часто возникает ситуация, когда данных достаточно для оценки качества кластеризации, но всё ещё не хватает для использования методов обучения с учителем.

Пусть n – общее количество объектов в выборке, n_k – количество объектов в кластере номер k , m_c – количество объектов в классе номер c , а $n_{c,k}$ количество

объектов из класса c в кластере k . Рассмотрим следующие величины:

$$\begin{aligned} H_{class} &= - \sum_{c=1}^C \frac{m_c}{n} \log \frac{m_c}{n}, \\ H_{clust} &= - \sum_{k=1}^K \frac{n_k}{n} \log \frac{n_k}{n}, \\ H_{class|clust} &= - \sum_{c=1}^C \sum_{k=1}^K \frac{n_{c,k}}{n} \log \frac{n_{c,k}}{n_k}. \end{aligned}$$

Несложно заметить, что эти величины соответствуют формуле энтропии и условной энтропии для мультиномиальных распределений $\frac{m_c}{n}, \frac{n_k}{n}, \frac{n_{c,k}}{n_k}$ соответственно.

Гомогенность кластеризации определяется такой формулой:

$$Homogeneity = 1 - \frac{H_{class|clust}}{H_{class}}.$$

Отношение $\frac{H_{class|clust}}{H_{class}}$ показывает, насколько уменьшается неопределенность в распределении классов (измеряемая энтропией), если мы знаем, к какому кластеру относится каждый объект.

Худший сценарий – когда отношение равно единице, то есть знание о кластерной принадлежности никак не помогает определить истинный класс объекта (энтропия не изменилась).

Лучший случай – когда каждый кластер содержит только объекты одного класса, и, следовательно, зная номер кластера, мы точно знаем истинный класс (гомогенность равна 1). Заметим, что тривиальный (и бессмысленный) способ добиться максимальной гомогенности – это выделить каждый объект в отдельный кластер.

Полнота

Метрика полноты аналогична гомогенности, но использует условную энтропию $H_{clust|class}$, которая симметрична $H_{class|clust}$:

$$Completeness = 1 - \frac{H_{clust|class}}{H_{clust}}.$$

Полнота равна единице, когда все объекты, принадлежащие одному и тому же истинному классу, находятся в одном и том же кластере.

Тривиальный, но непрактичный способ получить максимальную полноту – это объединить все объекты выборки в один большой кластер.

V-мера

Метрики гомогенности и полноты в кластеризации аналогичны точности и полноте в классификации. V-мера, в свою очередь, аналогична F-мере и представляет собой гармоническое среднее гомогенности и полноты. Пусть β – весовой параметр, тогда формула выглядит следующим образом:

$$V_\beta = \frac{(1 + \beta) \cdot \text{Homogeneity} \cdot \text{Completeness}}{\beta \cdot \text{Homogeneity} + \text{Completeness}}.$$

В случае $\beta = 1$ получаем, что V_1 -мера является просто средним гармоническим гомогенности и полноты:

$$V_\beta = \frac{2 \cdot \text{Homogeneity} \cdot \text{Completeness}}{\text{Homogeneity} + \text{Completeness}}.$$

Использование V-меры позволяет избежать тривиальных решений, таких как присвоение каждого объекта в отдельный кластер (максимальная гомогенность) или объединение всех объектов в один кластер (максимальная полнота). V-мера обеспечивает сбалансированную оценку качества кластеризации, учитывая как гомогенность, так и полноту.

Коэффициент силуэта

Коэффициент силуэта — метрика качества кластеризации, не требующая наличия меток классов. Сначала он вычисляется для каждого объекта, а затем итоговая метрика для всей выборки определяется как среднее значение коэффициентов силуэта всех объектов.

Для вычисления коэффициента силуэта $S(x_i)$ нужны две величины:

- $A(x_i)$ — среднее расстояние от объекта x_i до всех других объектов в том же кластере.
- $B(x_i)$ — среднее расстояние от объекта x_i до объектов ближайшего соседнего кластера.

Сам коэффициент силуэта вычисляется по формуле:

$$S(x_i) = \frac{B(x_i) - A(x_i)}{\max(B(x_i), A(x_i))}.$$

В идеале, точки внутри кластера должны быть ближе друг к другу, чем к точкам ближайшего соседнего кластера $A(x_i) < B(x_i)$. Однако это не всегда

так. Например, если кластер сильно вытянут или большой, а рядом находится небольшой кластер, то среднее расстояние до точек своего кластера ($A(x_i)$) может оказаться больше, чем до точек соседнего ($B(x_i)$). Поэтому разность $B(x_i) - A(x_i)$ может быть как положительной, так и отрицательной, хотя в идеале ожидается положительное значение. Коэффициент силуэта $S(x_i)$, изменяющийся от -1 до +1, максимизируется, когда кластеры компактны и хорошо разделены.

Главное преимущество коэффициента силуэта — он не требует меток классов и позволяет оценивать качество кластеризации при разных количествах кластеров.

Различия и выбор метрик качества кластеризации

Выбор метрики качества кластеризации зависит от нескольких факторов. Если число кластеров известно и разметка данных отсутствует, то целесообразно использовать среднее внутрикластерное или среднее межкластерное расстояние для оптимизации качества кластеризации.

Если же доступна разметка данных, то для оценки качества можно использовать гомогенность и полноту, а V-мера, сочетающая эти две метрики, позволяет также подбирать оптимальное число кластеров.

В случае отсутствия разметки и неизвестного числа кластеров, коэффициент силуэта является наиболее подходящей метрикой на практике. Исключение составляет ситуация, когда результаты кластеризации используются как промежуточный этап в более сложной задаче. В таких случаях качество кластеризации оценивается косвенно, по качеству решения конечной задачи, и выбор алгоритма кластеризации и его параметров подчиняется этой цели.

Задачи на понимание

Задача 1 Представьте, что у вас есть два набора данных с одинаковым средним внутрикластерным расстоянием. Может ли это означать, что качество кластеризации в обоих наборах одинаково? Объясните, почему да или нет.

Ответ Нет, одинаковое среднее внутрикластерное расстояние не гарантирует одинаковое качество кластеризации. Эта метрика отражает только компактность кластеров, игнорируя другие важные аспекты, такие как разделение между кластерами, форма кластеров и наличие выбросов. Например, в одном наборе данных кластеры могут быть компактными и хорошо разделены, а в другом — компактными, но сильно перекрывающимися. Среднее внутрикластерное расстояние будет одинаковым, но качество кластеризации — разным.

Задача 2 У вас есть данные, где границы между кластерами размыты, и некоторые точки могут принадлежать нескольким кластерам одновременно.

Какая метрика качества кластеризации наименее подходит для оценки результатов в этом случае, и почему?

Ответ Метрики, основанные на жестком распределении точек по кластерам (например, среднее внутрикластерное расстояние, среднее межкластерное расстояние), наименее подходят. Это связано с тем, что они предполагают, что каждая точка строго принадлежит только одному кластеру. В случае нечетких кластеров более подходящими могут быть метрики, учитывающие степень принадлежности точки к каждому кластеру.

Задача 3 Опишите сценарий, в котором высокая гомогенность, но низкая полнота. И наоборот.

Ответ Высокая гомогенность, низкая полнота. Представим, что у нас есть два истинных класса А и В. Алгоритм кластеризации создал много маленьких кластеров, каждый из которых содержит преимущественно объекты одного класса (высокая гомогенность). Однако объекты одного и того же класса (например, класса А) разбросаны по множеству разных кластеров. В этом случае полнота будет низкой, так как объекты одного класса не собраны вместе.

Высокая полнота, низкая гомогенность. В этом случае объекты одного класса сгруппированы в одном кластере (высокая полнота). Однако этот кластер содержит значительное количество объектов из других классов, что приводит к низкой гомогенности. Например, один большой кластер может содержать значительное количество объектов класса А и меньшее – класса В. Полнота для класса А высокая, а гомогенность низкая, потому что кластер "загрязнен" объектами класса В.

DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) - алгоритм кластеризации, решающий проблему с О сферичностью кластеров, он не делает никаких предположений о форме кластеров. Также он довольно быстрый и подходит для кластеризации больших данных.

Он основан на понятии *окрестности*.

Определение 1. Задан объект $x \in U$, его ε -окрестность $U_\varepsilon(x) = \{ u \in U : \rho(x, u) \leq \varepsilon \}$ - это множество объектов, которые находятся на расстоянии не больше ε от заданного объекта x .

Тогда каждый объект может быть отнесен к одному из трёх типов:

- *корневой*: имеющий плотную окрестность, $|U_\varepsilon(x)| \geq m$, т.е. ε содержит $\geq m$ объектов.

- *границный*: не корневой, но в окрестности корневого.
- *шумовой (выброс)*: не корневой и не граничный.

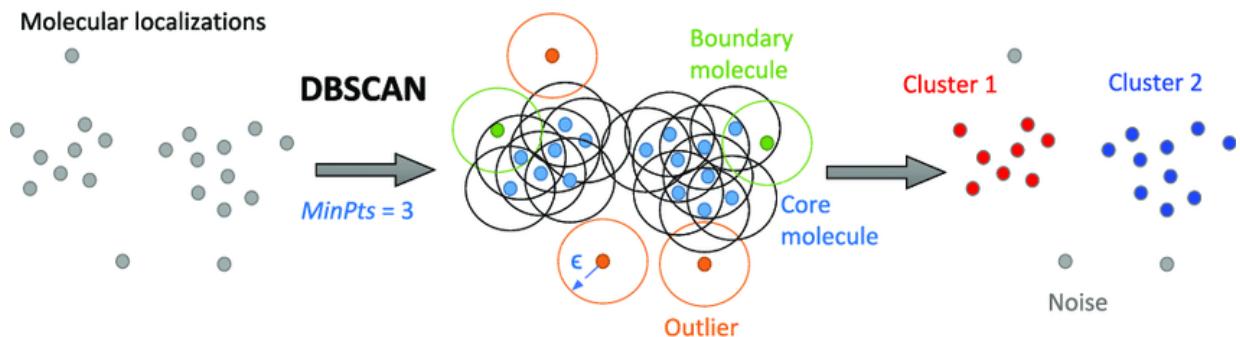


Рис. 17.1. An Example Illustrating the Density-Based DBSCAN Clustering Method Applied to SMLM Data

Возникает 2 параметра: ε и m . Других параметров не будет. От этих параметров и будет зависеть то, какой картина кластеризации получится. Также к преимуществам этого метода относится то, что он не задает заранее количество кластеров, в отличие, например, от k-means, причём количество кластеров будет зависеть от ε и m .

Как работает алгоритм: берётся произвольная точка, если она имеет плотную окрестность, то дальше рассматривается каждая точка этой плотной окрестности, и вокруг неё также строится ε -окрестность, и так пока не будет достигнута граница некоторого множества объектов.

Хорошей аналогией может служить лес: один лес - это один кластер, через опушку, второй лес, - другой кластер, мы находимся в лесу. Смотрим, в нашей окрестности деревьев много, это значит, что мы в корневой точке находимся, и дальше мы идём, пока не выйдем на опушку леса, там мы окажемся в граничной точке - она уже не корневая, вокруг деревьев меньше. А где-то могут расти отдельно стоящие деревья - это шумовые выбросы. И вот так ходим по лесу, пока его весь не обойдём, и как только мы обошли весь лес, назовем его кластером. После чего случайно выбираем новое дерево и начинаем строить другой кластер.

Формализуем алгоритм в виде псевдокода:

вход: выборка $X^l = \{x_1, \dots, x_l\}$; параметры ε и m

выход: разбиение выборки на кластеры и шумовые выбросы;

$U := X^l$ - не помеченные точки, $a := 0$

пока в выборке есть непомеченные точки, $U \neq \emptyset$:

взять случайную точку $x \in U$;

если $|U_\varepsilon(x)| < m$ **то**

пометить x как, возможно, шумовой;

иначе

создать новый кластер: $K := U_\varepsilon(x)$; $a := a + 1$;

для всех $x' \in K$, не помеченных или шумовых

если $|U_\varepsilon(x')| \geq m$, **то** $K := K \cup U_\varepsilon(x')$;

иначе пометить x' как граничный кластера K ;

$a_j := a$ для всех $x_i \in K$;

$U := U \setminus K$;

В таком виде алгоритм обладает следующими **свойствами**:

- быстрая кластеризация больших данных:
 $O(l^2)$ в худшем случае,
 $O(l \ln l)$ при эффективной реализации $U_\varepsilon(x)$;
- кластеры произвольной формы
- деление объектов на корневые, граничные, шумовые.

При этом важно понимать, что граничные объекты не выстраивают в точности границу каждого кластера. Практически это означает, что не стоит всерьез рассматривать граничные объекты, в отличие от шумовых, которые действительно можно в дальнейшем анализировать.

Примечание о HDBSCAN

От гиперпараметра ε можно избавиться, используя дивизивную кластеризацию. Такая модификация называется HDBSCAN. Его суть проста: необходимо построить дендрограмму, где по Oy будет отложен ε (на рис.17.2 снизу distance). Так мы сможем явно видеть вложенные кластеры. Алгоритм затем сам вычисляет оптимальное количество кластеров на основе метрики "стабильности кластеров".

Задачи

Задача 1.

Условие. Применить DBSCAN для выборки из таблицы с $m = 4$, $\varepsilon = 1.9$. Метрика евклидова.

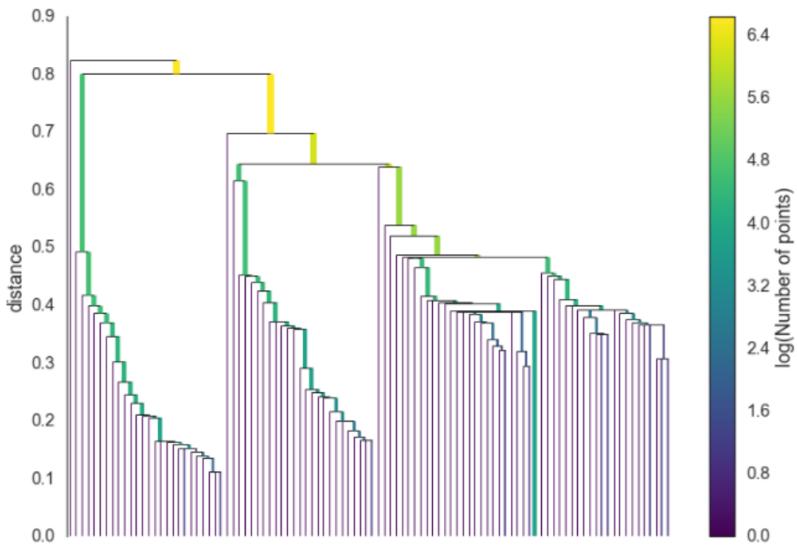


Рис. 17.2. К примечанию о HDBSCAN

P1(3,7)	P5(7,3)	P9(3,3)
P2(4,6)	P6(6,2)	P10(2,6)
P3(5,5)	P7(7,2)	P11(3,5)
P4(6,4)	P8(8,4)	P12(2,4)

Решение. Запишем матрицу, составленную из соответственных расстояний между точками выборки:

dot	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12
P1	0											
P2	1.41	0										
P3	2.83	1.41	0									
P4	4.24	2.83	1.41	0								
P5	5.66	4.24	2.83	1.41	0							
P6	5.83	4.47	3.16	2.00	1.41	0						
P7	6.40	5.00	3.61	2.24	1.00	1.00	0					
P8	5.83	4.47	3.16	2.00	1.41	2.83	2.24	0				
P9	4.00	3.16	2.83	3.16	4.00	3.16	4.12	5.10	0			
P10	1.41	2.00	3.16	4.47	5.83	5.83	5.66	6.40	6.32	0		
P11	2.00	1.41	2.00	3.16	4.47	4.24	5.00	5.10	2.00	1.41	0	
P12	2.83	3.16	4.00	5.10	4.47	5.39	6.00	1.41	2.00	2.00	1.41	0

Сравнивая значения в каждом столбце матрицы с ε и отбирая те, что меньше этого значения, находим окрестности каждой точки.

точка	окрестность
P1	P2, P10
P2	P1, P3, P11
P3	P2, P4
P4	P3, P5
P5	P4, P6, P7, P8
P6	P5, P7
P7	P5, P6
P8	P5
P9	P12
P10	P1, P11
P11	P2, P10, P12
P12	P9, P11

Если в окрестности больше $m = 4$ точек (включая ее саму), то отнесем эту точку к корневой, иначе - к шумовой.

точка	тип
P1	шум
P2	корневая
P3	шум
P4	шум
P5	корневая
P6	шум
P7	шум
P8	шум
P9	шум
P10	шум
P11	корневая
P12	шум

Уточним классификацию, учтя граничные точки, т.е. точки, лежащие в окрестности корневых, но при этом не являющимися корневыми:

точка	тип
P1	границная
P2	корневая
P3	границная
P4	границная
P5	корневая
P6	границная
P7	границная
P8	границная
P9	шум
P10	границная
P11	корневая
P12	границная

К первому кластеру отнесем окрестность корневой точки 2, причем в ее окрестности находится еще одна корневая точка 11, так что отнесем и ее окрестность к первому кластеру. Ко второму кластеру отнесем корневую точку 5 и ее окрестность. Осталась лишь одна точка P9, которая не относится ни к какому кластеру и является шумовой.

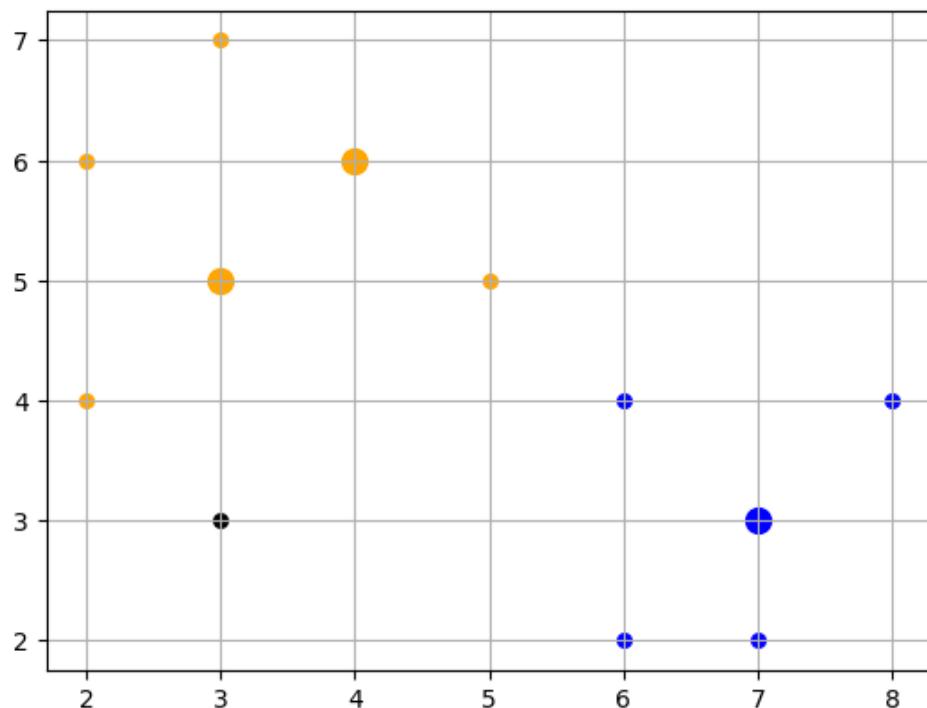


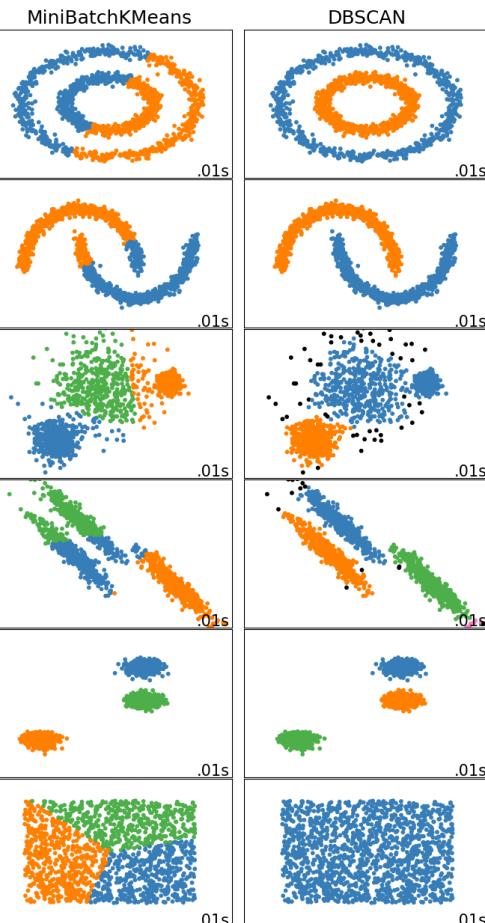
Рис. 17.3. Кластеризация в задаче 1

Задача 2.

Условие. Сравните результаты кластеризации с помощью k-means и с помощью DBSCAN и объясните их.

Решение. Объяснение различий:

- *Форма кластера:* K-средние: стремится найти сферические или выпуклые кластеры. Предполагается, что кластеры изотропны (однородны во всех направлениях) и имеют схожий размер. DBSCAN: может обнаруживать кластеры произвольной формы и размера. Не делает предположений о форме кластеров.
- *Обработка шума:* K-средние: плохо справляется с шумом. Точки шума могут быть назначены кластерам, что может повлиять на центры кластеров. DBSCAN: может идентифицировать и маркировать точки шума, которые не назначены ни одному кластеру.
- *Плотность кластера:* K-средние: не учитывает плотность точек. Каждый кластер представлен центроидом. DBSCAN: учитывает плотность точек. Кластеры формируются на основе плотности точек в окрестности.
- *Чувствительность параметров:* K-средние: требует предварительного указания количества кластеров (K), так что, если если заранее указать 3 кластера, то алгоритм и найдет три кластера, даже если он всего один, как на последней паре картинок.



Задача 3.

Предисловие. При решении задачи 1 использовалась матрица, состоящая из расстояний между парами точек (*матрица симметричности*). Понятием, противоположным расстоянию, является понятие сходства между объектами. Неотрицательная вещественная функция $S(x_i, x_j) = S_{ij}$ называется *мерой сходства*, если:

- $0 \leq S(x_i, x_j) < 1$, для $x_i \neq x_j$
- $S(x_i, x_j) = 1$
- $S(x_i, x_j) = S(x_j, x_i)$

Пары значений мер сходства можно объединить в *матрицу сходства* S , симметричную и единичной диагональю. **Условие.** Применить DBSCAN с пороговым значением *меры сходства* 0.8 и $m = 2$ и заданной матрицей сходства между точками выборки:

dot	P1	P2	P3	P4	P5
P1	1.0				
P2	0.10	1.0			
P3	0.41	0.64	1.0		
P4	0.55	0.47	0.44	1.0	
P5	0.35	0.98	0.85	0.76	1.0

Сравнивая значения в каждом столбце матрицы с ε и выбирая те точки, для которых значение сходства выше, чем порог, формируем окрестности всех точек.

точка	окрестность
P1	-
P2	P5
P3	P5
P4	-
P5	P2, P3

Если в окрестности больше $m = 2$ точек (включая ее саму), то отнесем эту точку к корневой, иначе - к шумовой.

точка	тип
P1	шум
P2	корневая
P3	корневая
P4	шум
P5	корневая

Уточнение классификации, путем учитывания граничных точек, т.е. точек, лежащие в окрестности корневых, но при этом не являющимися корневыми, ничего не дает, т.к. в окрестности точек, определенных как шумовые вообще нет других точек, так что они действительно являются шумом.

К первому кластеру отнесем окрестность корневой точки P2, причем в ее окрестности находятся еще краевая точка P5, так что отнесем ее к этому же кластеру. В окрестности точки P5 помимо уже классифицированной P2 находится еще корневая точка P3, которую также отнесем к первому кластеру. Остальные точки классифицированы как шумовые. Таким образом в данной задаче всего один кластер, состоящий из точек P2, P3, P5.

Простые эвристические методы частичного обучения

Постановка задачи

Существует привычная нам задача классификации. Мы имеем X — множество объектов с известными признаками и Y — пространство классов. Есть неизвестная функция $X \rightarrow Y$, сопоставляющая каждому объекту его класс. Имеется обучающая выборка $\{x_1, x_2, \dots\} \subset X$ и соответствующие им известные классы $\{y_1, y_2, \dots\} \subset Y$. Задача классификации сводится к построению классификатора — некой аппроксимации неизвестной функции $X \rightarrow Y$.

С другой стороны существует задача кластеризации. Мы все также имеем X — множество объектов. И хотим аппроксимировать функцию $X \rightarrow Y$. Но на этот раз у нас нет обучающей выборки, зато есть функция расстояния между объектами $\rho : X \times X \rightarrow \mathbb{R}$. И в этом случае кластеризующая функция строится не на основе обучающей выборки, а так чтобы расстояние между объектами одного кластера было мало, а расстояние между объектами разных кластеров было велико.

Первый случай называется обучение с учителем, а кластеризация называется обучением без учителя. Где-то по середине между этими задачами находится задача частичного обучения. В этом случае у нас все также есть множество объектов из X и (возможно) функция ρ . При этом только для некоторой доли имеющихся объектов известна классовая принадлежность. Иначе говоря обучающая выборка размечена **частично**.

Приведем пример (Рис. 1). Допустим известные объекты в пространстве признаков имеют вид в виде двух бананов. И нам известна принадлежность только двух точек. В таком случае чистая задача кластеризации обучит классификатор только по двум точкам, который будет иметь сомнительное качество на всех остальных данных. При этом частичное обучение могло бы учсть явную кластеризацию данных и дать значительно лучшую классификацию.

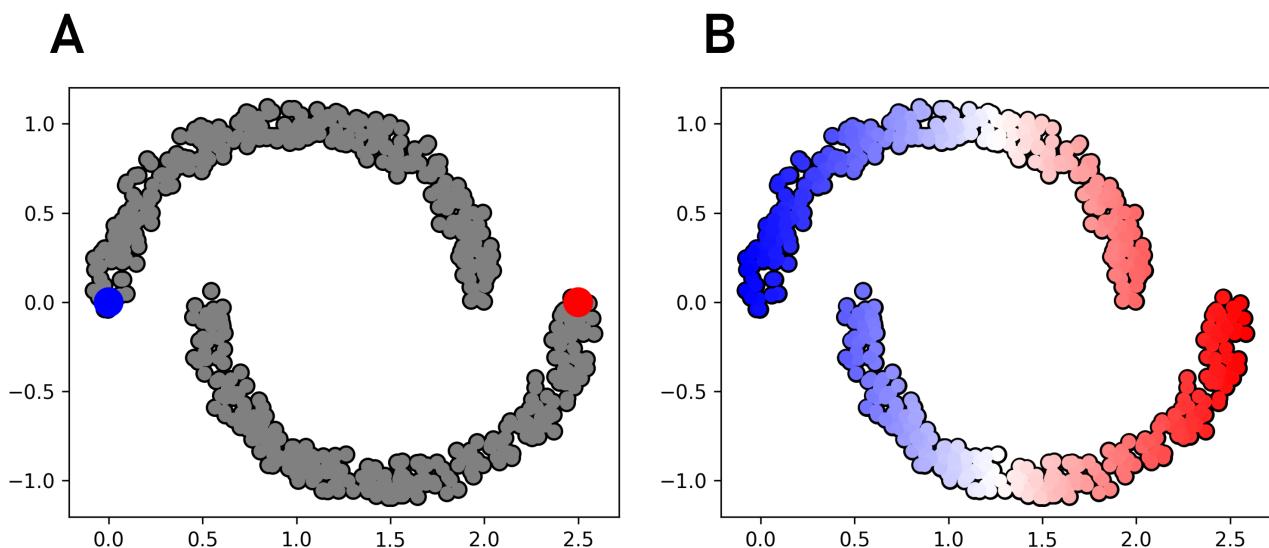


Рисунок 1. (A) Пример частично размеченных данных. (B) Классификация обученная на размеченных данных не учитывает кластерную структуру неразмеченных.

Однако частичное обучение не может сводиться только к кластеризации, представьте теперь, что мы знаем классы уже трех точек на выборке из двух бананов (Рис. 2). В таком случае кластеризация дала бы предсказание которое не может соотноситься с известными классами всех трех точек. Таким образом задача частичного обучения действительно находится посередине между классификацией и кластеризацией, но не является ни тем, ни другим.

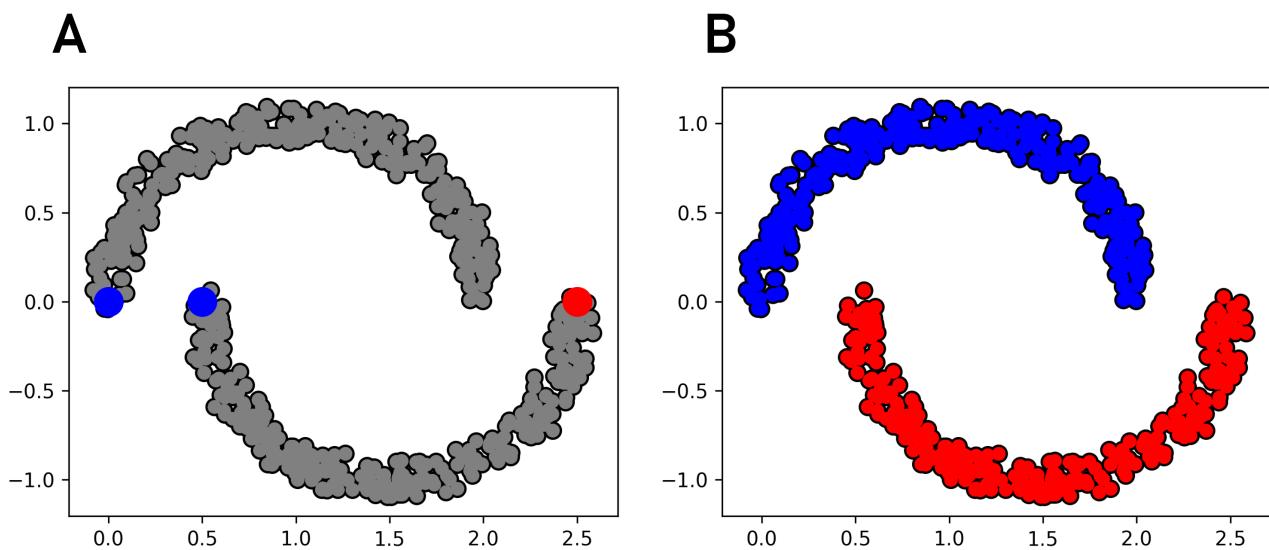


Рисунок 2. (A) Пример частично размеченных данных. (B) Кластеризация не учитывает классовую принадлежность размеченных данных.

Приведем пример: программист решил сделать классификатор для фотографий котиков и собачек, для этого он скачал по миллиону фотографий и

тех и других. Однако, ему хватило сил подписать котик это или собачка только для 1000 фотографий. Классификация не может быть построена только по 1000 фотографиям — мало данных, а кластеризация может разбить фотографии неизвестным образом — по цвету фона, по размеру животного. И только частичное обучение может помочь ленивому программисту.

Self-training

Каким же образом можно реализовать частичное обучение? Рассмотрим подход, который называется self-training (само-обучение). Вернемся к датасету с бананами в котором известны классы двух точек. Как было сказано ранее построение классификатора по двум точкам тут не поможет. Однако допустим, что мы все же построили классификатор по двум точкам, к примеру логистическую регрессию. Вспомним что многие методы классификации могут оценивать свою уверенность в предсказании. Тогда для некоторых точек с наибольшей уверенностью классификации (скажем 5% самых уверенных) предсказание классификации может оказаться вполне верным. Давайте же дополним обучающую выборку классификатора этими точками. На второй итерации нашего подхода мы обучим классификатор уже по дополненной обучающей выборке. Выполнив предсказание для остальных данных, выберем еще раз 5% ранее неразмеченных точек с наибольшей уверенностью и пополним ими обучающую выборку. Будем повторять такие итерации пока не классифицируем все точки в нашем датасете. Как мы видим, результат такого подхода заметно лучше, чем у классификации по двум исходным точкам.

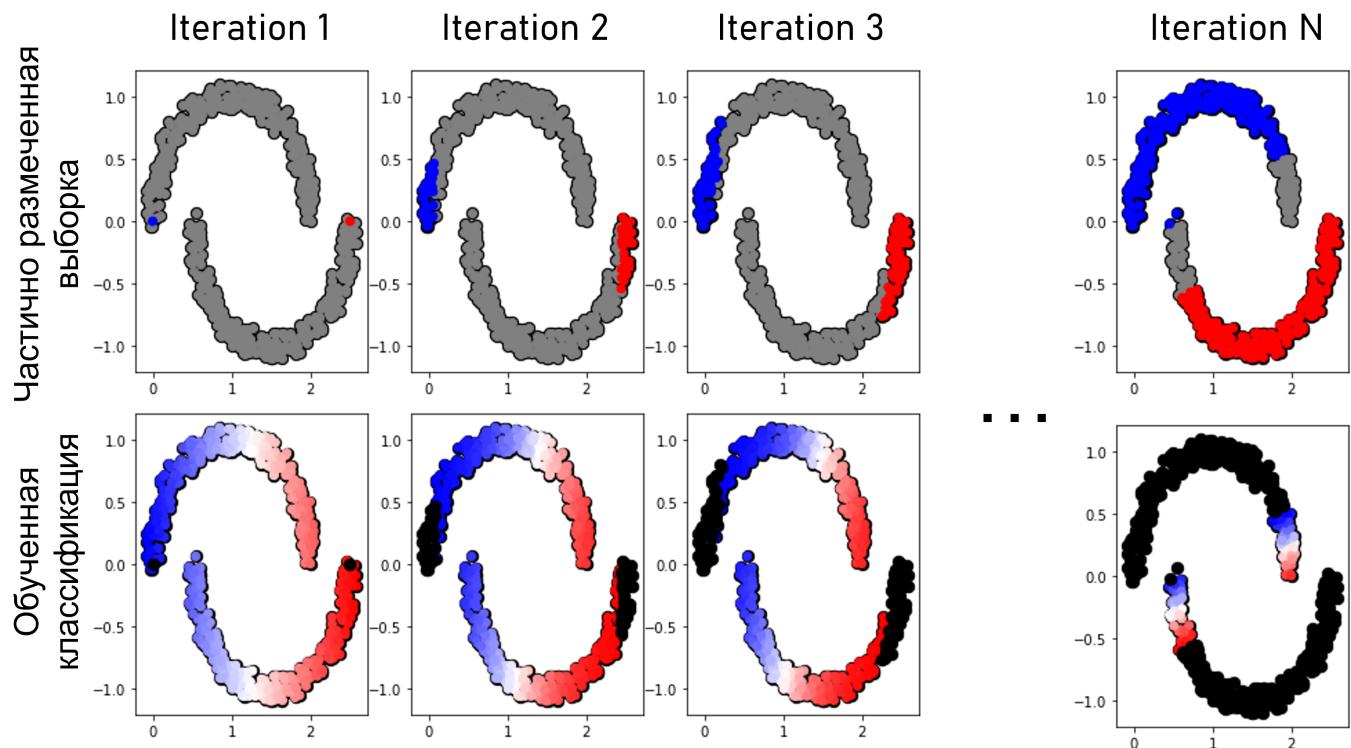


Рисунок 3. Пример работы итераций self-training.

Оформим описанный процесс более математизировано. Пусть мы строим функцию $a : X \rightarrow Y$. Пусть у нас есть метод обучения функции $\mu : Z \rightarrow a$ который принимает на вход размеченную часть выборки $Z \subset X$. Допустим функция a имеет вид

$$a(x) = \arg \max_{y \in Y} \Gamma_y(x)$$

где $\Gamma_y(x)$ это некоторые (к примеру) линейные функции от $x \in X$ которые обучаются так, чтобы быть большими, если x принадлежит классу y и маленькими в противном случае. В таком случае уверенность классификации в принадлежности элемента x к классу y_1 (отступ):

$$M_1(x) = \Gamma(x) - \max_{y \in Y \setminus y_1} \Gamma_{y_1}(x)$$

Пусть Z это размеченная часть обучающей выборки. Тогда алгоритм выглядит так:

1. $a = \mu(Z)$ — обучить классификатор на размеченной выборке
2. $\Delta := \{x \in X \setminus Z \mid M(x) \geq M_0\}$ — выбрать несколько точек из неразмеченной части выборки которые наиболее уверенно классифицируются.
3. $Z := Z \cup \Delta$ — дополнить размеченную выборку
4. Если не все элементы выборки размечены, вернуться в начало.

Co-training

Рассмотрим более узкий кейс, допустим у нас есть не один, а целых два метода обучения классификации μ_1, μ_2 , которые принципиально отличаются друг от друга, например имеют разные парадигмы обучения, и/или используют разные признаки объектов, и/или имеют разную стартовую выборку. В таком случае мы можем получить преимущество в частичном обучении заставив их учить друг друга по следующему алгоритму:

1. $a_1 = \mu_1(Z_1)$
 $a_2 = \mu_2(Z_2)$ — два метода обучаются классификаторами на своих размеченных выборках.
2. $\Delta_1 := \{x \in X \setminus Z_1 \setminus Z_2 \mid M_1(x) \geq M_{01}\}$ — метод 1 размечает неразмеченные точки, в которых он уверен.
 $\Delta_2 := \{x \in X \setminus Z_1 \setminus Z_2 \mid M_2(x) \geq M_{02}\}$ — метод 2 делает тоже самое.
3. $Z_1 := Z_1 \cup \Delta_2$ — метод 2 дополняет обучающую выборку метода 1
 $Z_2 := Z_2 \cup \Delta_1$ — метод 1 дополняет обучающую выборку метода 2
4. Если не все элементы выборки размечены, вернуться в начало.

Co-learning

Идем еще дальше, допустим у нас теперь есть набор методов μ_i отличающиеся чем-то. Допустим что все эти методы обучились на одной выборке Z и произвели классификаторы a_i . Давайте соберем из множества классификаторов a_i один классификатор-мегазорд:

$$a(x) = \arg \max_{y \in Y} \Gamma_y(x)$$

Где на этот раз функции Γ определяются как:

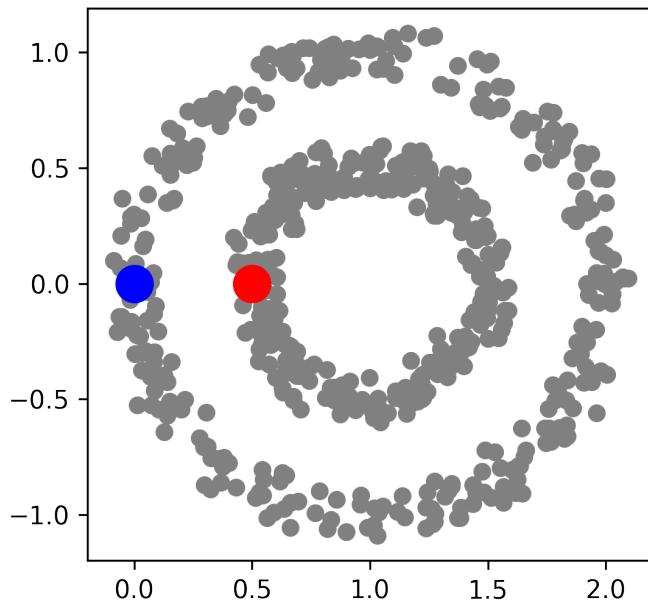
$$\Gamma_y(x) = \sum_{i=1}^I [a_i(x) = y]$$

Выражение в квадратных скобках равно 1 когда написанное в них верно, и равно 0 в противных случаях. Иначе говоря, классификаторы a_i голосуют за принадлежность к классам, и демократически выбирают к кому классу отнести каждый из объектов. Далее на основе такого объединенного классификатора строится Self-training описанный выше.

Задачи

Задача 1

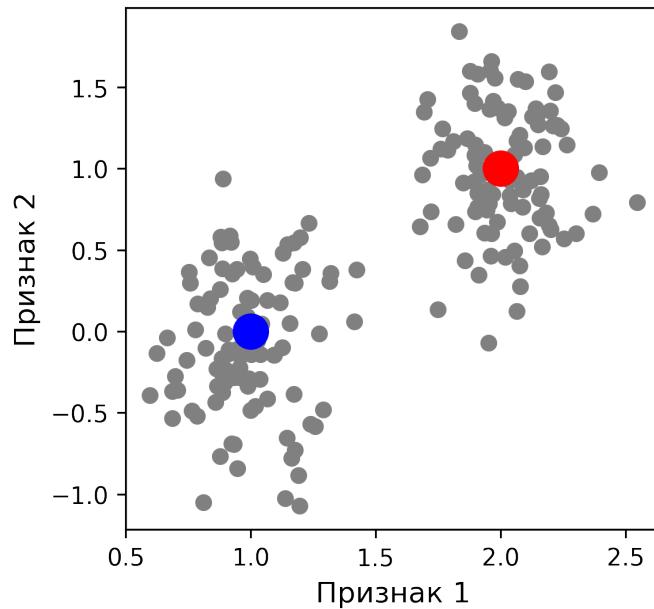
Возможно ли при помощи self-training эффективно классифицировать точки частично размеченной выборки представленной ниже:



Решение: Конечно можно! Для этого нужно построить self-training на основе классификатора который может делать разделяющую поверхность в виде окружности, например классификаторы восстанавливающие плотность или логистическая регрессия с добавлением квадратичных признаков

Задача 2

Два подхода частичного обучения применяются для классификации объектов частичной выборки представленной ниже.

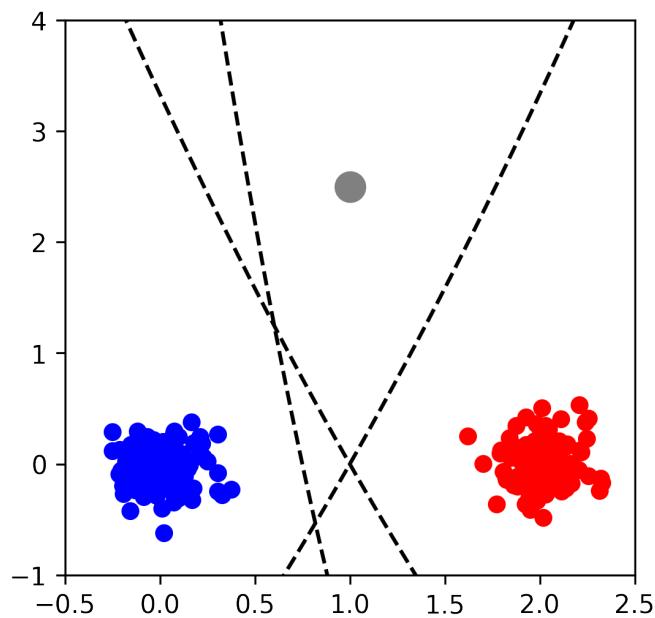


Первый подход это self-training на основе логистической регрессии, которая видит только первый признак. Второй подход это co-training на основе двух логистических регрессий, таких что первая видит первый признак, а вторая второй. Какой алгоритм приведет к лучшей классификации точек и почему?

Решение: *self-training* видящий только первый признак вероятнее всего идеально классифицирует точки, так как по этому признаку кластеры разделены и практически не пересекаются. В случае *co-training* классификатор использующий второй признак видит кластеры пересекающимися и почти наверняка наделает ошибок при классификации

Задача 3

Алгоритм co-training построенный на основе трех классификаторов классифицирует точки представленной ниже выборки. На последней итерации работы алгоритма осталась лишь одна неклассифицированная точка. На этот момент классификаторы имеют разделяющие линии представленные на картинке. К какому кластеру будет отнесена оставшаяся точка?



Решение: Судя по разделяющим линиям два классификатора проголосуют за красный кластер, и только один за синий, в итоге точка будет отнесена к красному кластеру.

Глава 18

Обучение на неразмеченных данных

Неразмеченные данные в глубоком обучении

Постановка

Общий подход к решению задачи с неразмеченными данными следующий:
 $\mathcal{U} \subset \mathcal{X}$ — большая неразмеченная выборка,
 $\mathcal{D} \subset \mathcal{X} \times \mathcal{Y}$ — небольшая размеченная выборка,
требуется построить отображение $f : \mathcal{X} \rightarrow \mathcal{Y}$.

Суперпозиция

Предположение $f = c \circ h$ — отображение есть суперпозиция двух функций:
 h — генерация признакового описания $h : \mathcal{X} \rightarrow \mathcal{H}$,
 c — классификатор $c : \mathcal{H} \rightarrow \mathcal{Y}$.

Заметим, что c и h , например, это некоторое параметрические функции, параметры которых нужно найти.

Простой пример:

h — все слои полносвязного многослойного перцептрона,
 c — последний слой.

Обычно h является сложной моделью, а c — линейной. Рассмотрим более подробный пример.

Постановка задачи SimCLR

Постановка задачи SimCLR заключается в том, что даётся набор изображений без каких-либо меток, и нужно обучить модель на этих данных так, чтобы она могла быстро адаптироваться к любой задаче распознавания изображений впоследствии.

Цель обучения - найти такое признаковое пространство, в котором изображения одного класса располагаются близко по введённой метрике, а изображения разных классов - далеко. Для этого используют различные *контрастивные функции потерь*.

На рисунке можем схематичное представление последовательности шагов алгоритма со следующими обозначениями:

x - исходное изображение,

t, t' - разные аугментации на исходное изображение x для получения представлений \tilde{x}_i, \tilde{x}_j ,

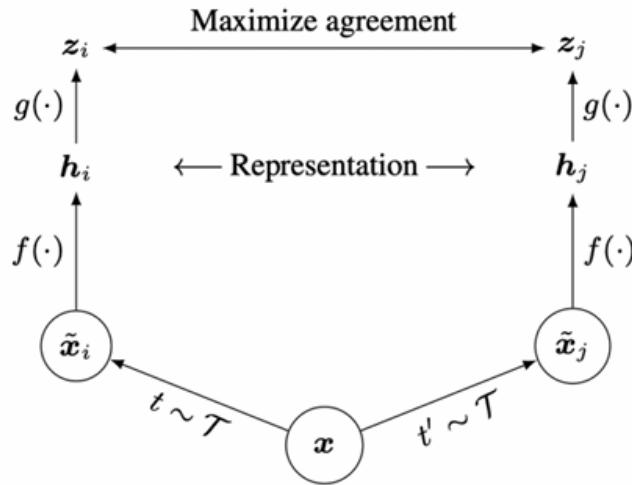


Рис. 18.1. Схематичное представление алгоритма SimCLR

f - функция, генерирующая представления (representation) h_i, h_j в виде вектора для входных изображений,

g - функция, которая переводит вектора h_i, h_j в величины z_i, z_j , для которых минимизируется расстояние (maximize agreement) в процессе обучения для аугментаций одного изображения и максимизируется для разных изображений.

Шаги в обучении следующие:

1. К исходному изображению x применяются аугментации t, t' . *Аугментация* - это процесс создания новых данных на основе существующих с помощью преобразований. Такими преобразованиями для изображений являются, например, поворот, изменение яркости, гауссовское размытие и так далее. Она необходима, чтобы в новом признаковом пространстве изменённые изображения так же находились близко к изображениям того же класса.

2. К аугментированным изображениям \tilde{x}_i, \tilde{x}_j применяется функция f , генерирующая векторное представление этих изображений h_i, h_j .

3. К векторным представлениям h_i, h_j применяется функция g для перевода в другое векторное представление. Данный шаг необязательный для обучения методом SimCLR.

4. Для z_i, z_j уменьшается расстояние, если \tilde{x}_i, \tilde{x}_j - аугментации одного и того же изображения, и увеличивается - если разных. Происходит это путём минимизации контрастивной функции потерь.

Таким образом мы находим новое признаковое описание для изображения, с помощью которого можно обучиться на задачу классификации изображений, то есть найти функция c из раздела про Суперпозицию.

Вопросы

1. Что такое суперпозиция в контексте построения модели для неразмеченных данных, и как она помогает в решении задачи с использованием методов глубокого обучения?
2. Каковы основные этапы и задачи в методе SimCLR?
3. Как аугментация данных способствует улучшению качества обучаемых представлений?

Список литературы