

Текст - взято из лекции

Текст - взято не из лекции

[Курс "Машинное обучение" на ФКН ВШЭ](#)

[Materials for ML Course of NSU FIT](#)

[Лекции Неделько](#)

[Лекции Воронцов](#)

[SciKit-Learn на английском scikit-learn.ru](#)

Лекция 1

Метод прецедентов

Для объекта, который нужно классифицировать, находятся наиболее похожие на него объекты, для которых целевой признак известен. Прогноз осуществляется на основе голосования по прецедентам.

[Алгоритм](#)

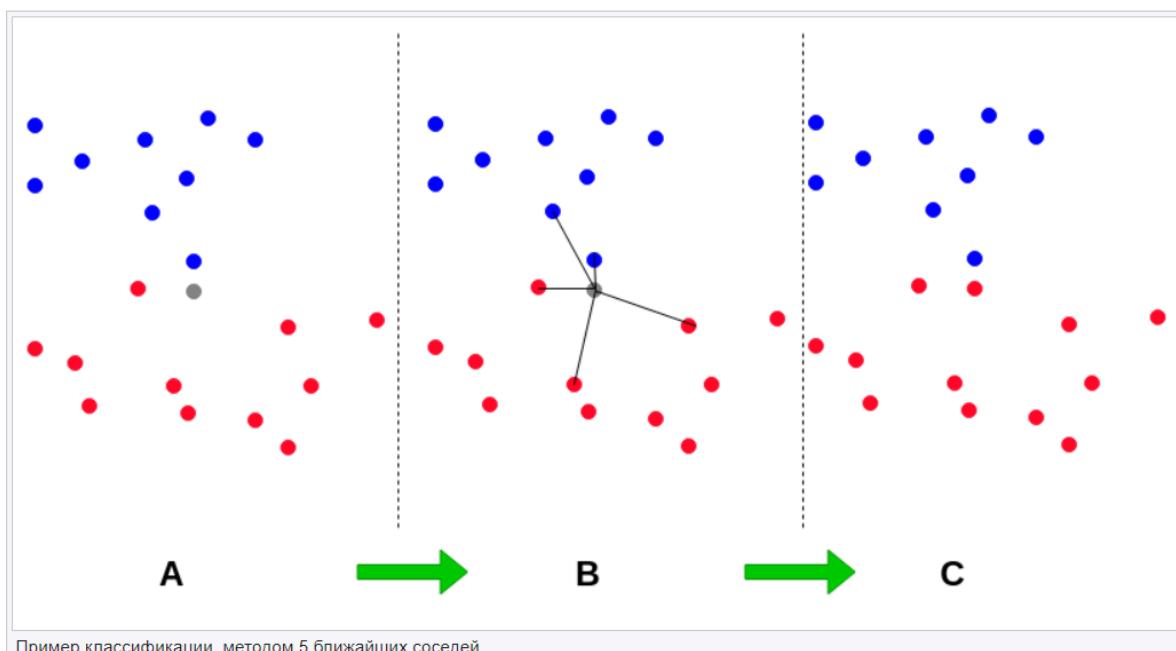
- К-ближайших соседей (прогноз на основе объектов, ближайших к исследуемому)

<http://www.ccas.ru/voron/download/MetricAlgs.pdf#page=3>

Для повышения надёжности классификации объект относится к тому классу, которому принадлежит большинство из его соседей — K-ближайших к нему объектов обучающей выборки X_i . В задачах с двумя классами число соседей берут нечётным, чтобы не возникало ситуаций неоднозначности, когда одинаковое число соседей принадлежат разным классам.

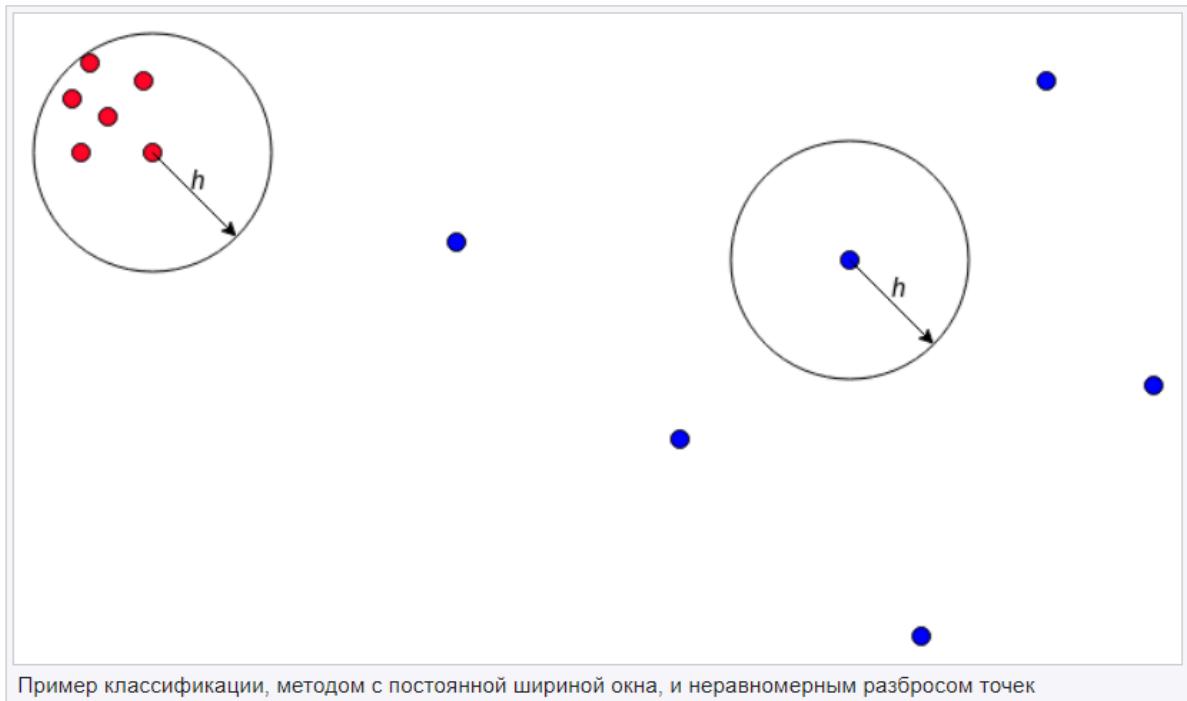
- Парзеновское окно (прогноз на основе объектов, попавших в окрестность исследуемого объекта)

<http://www.ccas.ru/voron/download/Bayes.pdf#page=11>



<http://www.ccas.ru/voron/download/MetricAlgs.pdf#page=4>

Аналогично с примером выше, но берется окружность, в центре которой предсказываемый объект.



Пример классификации, методом с постоянной шириной окна, и неравномерным разбросом точек

- Скользящий контроль

<http://www.ccas.ru/voron/download/Introduction.pdf#page=6>

<http://www.ccas.ru/voron/download/Bayes.pdf#page=12>

Эмпирические оценки обобщающей способности применяются, когда нет адекватных теоретических оценок. Пусть имеется выборка $X^L = (x_i, y_i)_{i=1}^L$. Разобьём её N различными способами на две части: $X^L = X_n^\ell \cup X_n^k$ — обучающую подвыборку длины ℓ и контрольную длины k , где $\ell + k = L$. Вычислим по всем разбиениям среднее качество на контроле:

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

Эта величина называется оценкой *скользящего контроля* (cross-validation, CV). Возможны различные варианты скользящего контроля, отличающиеся способами разбиения выборки X^L на две части, см. [18] или Главу ???. Скользящий контроль считается стандартной методикой оценивания качества обучения. Основной его недостаток — значительные вычислительные затраты, ведь задачу обучения приходится решать N раз.

Со второй ссылки:

Чтобы оценить при данном h точность локальной аппроксимации плотности в точке x_i , саму эту точку необходимо исключить из обучающей выборки. Если этого не делать, максимум правдоподобия будет достигаться при $h \rightarrow 0$. Такой способ оценивания называется скользящим контролем с исключением объектов по одному (leave-one-out, LOO):

$$\text{LOO}(h, X^\ell) = \sum_{i=1}^{\ell} [a(x_i; X^\ell \setminus \{x_i\}, h) \neq y_i] \rightarrow \min_h,$$

где $a(x; U, h)$ — алгоритм классификации с параметром ширины окна h , построенный по обучающей выборке $U \subseteq X^\ell$.

Обычно зависимость LOO от h имеет характерный минимум, соответствующий оптимальной ширине окна h^* , см. Рис ??.

- Яdrovoe Сглаживание (вклад объекта в прогноз есть некоторая функция расстояния)

При использовании линейной функции в качестве $W(i, U)$ возможно совпадение суммарного веса для нескольких классов. Это приводит к неоднозначности ответа при классификации. Чтобы такого не происходило, используют некоторую функцию ядра.

обучения. Яdrovoe сглаживание (*kernel smoothing*) - это метод, используемый для оценки плотности вероятности случайной величины или регрессии на основе набора данных.

В случае яdroвой регрессии, прогноз для данного значения зависимой переменной рассчитывается как взвешенная сумма значений зависимой переменной, где веса определяются ядерной функцией (*kernel*) в зависимости от расстояния между целевым значением и значениями в наборе данных.

Формально, предсказание $\hat{Y}(x)$ для заданного x может быть выражено как:

$$\hat{Y}(x) = \frac{\sum_{i=1}^n K\left(\frac{x-x_i}{h}\right) \cdot y_i}{\sum_{i=1}^n K\left(\frac{x-x_i}{h}\right)}$$

где:

- n - количество наблюдений в наборе данных,
- x_i и y_i - соответственно, значения признака и зависимой переменной для i -го наблюдения,
- K - ядерная функция, определенная на расстоянии $\frac{x-x_i}{h}$,
- h - параметр сглаживания (ширина окна).

Популярными ядерными функциями являются, например, гауссово ядро (нормальное распределение), прямоугольное ядро и триангулярное ядро. Выбор ядерной функции и параметра сглаживания зависит от конкретной задачи и характеристик данных.

Этот метод часто используется в статистике и машинном обучении для аппроксимации сложных функций и анализа данных, особенно в случаях, когда данные не имеют явной структуры или распределены неравномерно.

Сглаживание с использованием ядер есть наиболее общий вариант метода прецедентов.

Свойства Метода Прецедентов

Метод прецедентов, несмотря на естественность и очевидность, на практике работает относительно плохо. Недостатки метода: равное использование всех переменных, неустойчивость к "шумам", особенно при большой размерности, неиспользование возможной независимости переменных, неиспользование априорных знаний.

Лекция 2

Квази(линейные) методы классификации (LDA, QDA, дискриминант Фишера, логистическая регрессия)

- **Линейный и квадратичный дискриминант:** Это методы, используемые в статистике для классификации и разделения данных. Линейный дискриминант анализирует данные, предполагая линейную зависимость, в то время как квадратичный дискриминант учитывает квадратичные отношения.
- **Дискриминант Фишера:** Это статистический метод, используемый для определения линейной комбинации признаков, которая наилучшим образом разделяет два или несколько классов объектов или событий.
- **Логистическая регрессия:** Метод анализа, используемый для предсказания исхода переменной на основе предыдущих наблюдений. Она часто используется для двоичной классификации.
- **Наивный байесовский классификатор:** Простой вероятностный классификатор, основанный на применении теоремы Байеса с наивными предположениями о независимости.
- **Машинная опорных векторов (SVM):** Популярный метод машинного обучения, используемый для классификации и регрессионного анализа. SVM стремится найти гиперплоскость в многомерном пространстве, которая лучше всего разделяет различные классы.

1. Бинарная классификация

- **Задача:** Определить, к какому из двух классов ($y \in \{-1, 1\}$) принадлежит наблюдение x .
- **y :** Классификационная переменная, принимающая одно из двух значений - либо -1 , либо 1 .

2. Априорные вероятности и условные плотности

- **$P(y)$ = P_y :** Априорная вероятность класса y .
- **$\varphi y(x)$:** Условная плотность вероятности для класса y . Выражается формулой нормального распределения:

$$\varphi y(x) = \frac{1}{\sqrt{2\pi}\sigma_y} e^{-\frac{(x-\mu_y)^2}{2\sigma_y^2}}$$

где μ_y — среднее, а σ_y — стандартное отклонение для класса y .

3. Байесовская решающая функция

- **$y^*(x) = \arg \max y \varphi(x, y)$:** Функция определяет класс y^* , к которому лучше всего относится наблюдение x , путем максимизации совместной плотности вероятности:

$$\varphi(x, y) = \varphi y(x)P(y)$$

4. Формула Байеса для гауссовых распределений

- **$g1(x)$:** Функция вычисляет условную вероятность $P(y=1|x)$:

$$g1(x) = \frac{\varphi 1(x)P_1}{\varphi(x)}$$

- **$\varphi(x)$:** Общая плотность вероятности для x :

$$\varphi(x) = \varphi 1(x)P_1 + \varphi - 1(x)P - 1$$

5. Разделяющая функция $I(x)$

- **$I(x)$:** Функция вычисляет логарифм отношения плотностей вероятностей для двух классов. Используется для определения границы решения:

$$l(x) = \ln \varphi 1(x) - \ln \varphi - 1(x) + \ln \frac{P_1}{P - 1}$$

6. Расширенная формула для многомерных случаев

- **$\varphi y(x)$ для многомерного случая:**

$$\varphi y(x) = \frac{1}{(2\pi)^{n/2}|\lambda_y|^{n/2}} e^{-\frac{1}{2}Q_y(x)}$$

где $Q_y(x)$ - квадратичная форма, λ_y - ковариационная матрица класса y .

7. Линейная разделяющая функция в многомерном случае

- **$2I(x)$:**

$$2I(x) = x'Ax + bx + c$$

где A , b , c определяются через различия в ковариационных матрицах классов.

8. Безусловные (априорные) вероятности классов и оценки параметров

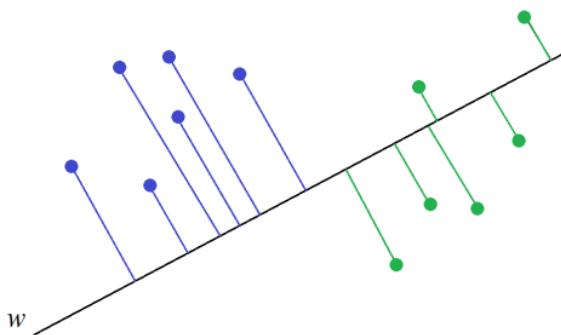
- **Априорные вероятности:** Оцениваются на основе доли каждого класса в обучающей выборке.
- **Оценка параметров:** Среднее и ковариационная матрица для каждого класса оцениваются на основе данных.

Замечания:

- В реальных задачах довольно редко можно обосновать нормальность распределений.
- Даже если известно, что распределения действительно нормальны, это не гарантирует оптимальность выборочной решающей функции.
- При большом числе переменных и малой выборке приходится строить решение как при равных матрицах ковариаций.

Дискриминант Фишера

<http://www.ccas.ru/voron/download/Bayes.pdf#page=20>



Проектирование на направление.

Дискриминант Фишера — это метод в статистике, используемый для линейного преобразования признакового пространства, чтобы максимизировать разделение между двумя или несколькими классами. Это один из способов уменьшения размерности данных, сохраняя при этом информацию, важную для различия классов.

Основная идея дискриминанта Фишера заключается в следующем:

1. **Выбор направления для проектирования:** Выбирается направление (вектор w), на которое будут проецироваться данные так, чтобы максимизировать разделение между классами.
2. **Формула для оптимизации:** Цель состоит в максимизации критерия Фишера, который определяется как:

$$\Phi(w) = \frac{(\mu_{ew,1} - \mu_{ew,-1})^2}{S_{ew,1} + S_{ew,-1}}$$

где $\mu_{ew,y}$ — среднее значение проекций класса y , а $S_{ew,y}$ — сумма квадратов отклонений проекций в классе y .

3. Расчёт средних и ковариаций:

- $\mu_{ew,y}$ вычисляется как среднее проекций точек класса y на вектор w .
- $S_{ew,y}$ — это сумма квадратов отклонений этих проекций от среднего.

4. Оптимальный вектор проекции w_Φ :

- Максимум $\Phi(w)$ достигается, когда $w_\Phi = S_e^{-1}(\mu_{e1} - \mu_{e-1})$,
- где μ_{ey} — среднее точек выборки класса y ,
- S_{ey} — ковариационная матрица класса y ,
- $S_e = S_{e1} + S_{e-1}$ — сумма ковариационных матриц классов.

Основная идея заключается в сравнении межгрупповой дисперсии (разброса средних значений групп) с внутригрупповой дисперсией (разброс средних значений внутри каждой группы). Если межгрупповая дисперсия существенно больше внутригрупповой, то это может свидетельствовать о статистически значимых различиях между группами.

Вот формула для дискриминанта Фишера:

$$F = \frac{\text{Between-group variability}}{\text{Within-group variability}}$$

где

- Between-group variability — межгрупповая дисперсия
- Within-group variability — внутригрупповая дисперсия

Идея заключается в выборе направления, при проектировании выборки на которое образы классов оказываются наиболее удалёнными друг от друга. Это выражается в максимизации критерия

$$\Phi(w) = \frac{(\tilde{\mu}_{w,1} - \tilde{\mu}_{w,-1})^2}{\tilde{S}_{w,1} + \tilde{S}_{w,-1}},$$

где $\tilde{\mu}_{w,y} = \frac{1}{N_y} \sum_{i \in I_y} wx_i$ — среднее, а $\tilde{S}_{w,y} = \frac{1}{N_y} \sum_{i \in I_y} (wx_i - \tilde{\mu}_{w,y})^2$ — средний квадрат отклонений проекций.

Замечания:

- **Метод не требует никаких вероятностных предположений.**
- **Выражение для w_Φ очень похоже на выражение для нормали к разделяющей гиперплоскости для случая нормальных распределений с равными матрицами ковариаций.**
- **Метод предполагает оценивание по выборке только p параметров и не требователен к объему выборки.**
- **Метод, выведенный при сильных предположениях, может оставаться пригодным и при нарушении предположений.**

Логистическая регрессия

<http://www.ccas.ru/voron/download/Regression.pdf#page=29>

Логистическая регрессия

Рассмотрим функцию условной вероятности для класса 1

$$g(x) = P(y = 1|x) = \frac{P(1)\varphi_1(x)}{P(1)\varphi_1(x) + P(-1)\varphi_{-1}(x)} = \frac{1}{1 + e^{-l(x)}},$$

где $l(x)$ – разделяющая функция.

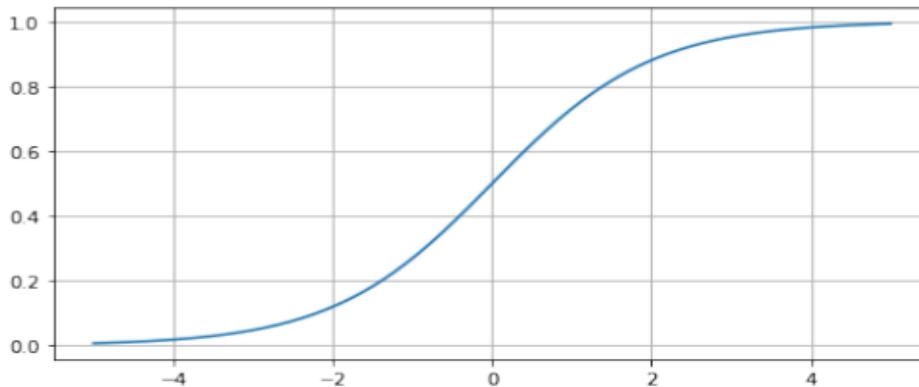
Подставив нормальные плотности при равных матрицах ковариаций, получим

$$g(x) = \frac{1}{1 + e^{-(wx+w_0)}} = \sigma(wx + w_0),$$

где w и w_0 есть b и c соответственно из линейного дискриминанта.

Сигмоид

Здесь $\frac{1}{1+e^{-z}}$ – так называемая логистическая функция (иногда также называемая сигмоидом или логит-функцией).



Оценивание параметров

Метод логистической регрессии основан на оценивании функции условной вероятности моделью $\tilde{g}(x) = \sigma(\tilde{w}x + \tilde{w}_0)$, в которой \tilde{w} и \tilde{w}_0 – настраиваемые параметры.

На практике параметры модели обычно оцениваются путём максимизации критерия правдоподобия (условной вероятности выборки).

$$-K_V(\tilde{w}, \tilde{w}_0) = \sum_{i \in I_1} \ln \tilde{g}(x^i) + \sum_{i \in I_{-1}} \ln(1 - \tilde{g}(x^i)).$$

Логистическая регрессия — это метод бинарной классификации, который используется для прогнозирования вероятности того, что экземпляр принадлежит к определенному классу. Ниже представлен базовый алгоритм логистической регрессии:

Обучение модели:

1. Инициализация весов:

- Задайте начальные значения весов w_0, w_1, \dots, w_n (где n - количество признаков) и порог b .

2. Определение входных данных:

- Подготовьте данные. Представьте каждый объект выборки как вектор признаков $x = (x_1, x_2, \dots, x_n)$ и добавьте единичный признак для удобства: $x_0 = 1$.

3. Вычисление линейной комбинации:

- Вычислите линейную комбинацию входных данных и весов: $z = w_0x_0 + w_1x_1 + \dots + w_nx_n$.

4. Применение функции логистической активации:

- Пропустите z через логистическую функцию (сигмоид): $h(z) = \frac{1}{1+e^{-z}}$.
- Получите предсказанную вероятность $h(x)$, что объект принадлежит к классу 1.

5. Определение функции стоимости:

- Определите функцию стоимости (или функцию потерь), например, кросс-энтропию:

$$J(w) = -\frac{1}{m} \sum_{i=1}^m [y^{(i)} \log(h(x^{(i)})) + (1 - y^{(i)}) \log(1 - h(x^{(i)}))],$$

где m - количество обучающих примеров, $y^{(i)}$ - истинная метка для i -го примера.

6. Градиентный спуск:

- Минимизируйте функцию стоимости с использованием градиентного спуска:
 $w_j := w_j - \alpha \frac{\partial J(w)}{\partial w_j}$,
где α - скорость обучения (learning rate).

7. Повторение:

- Повторяйте шаги 3-6 до тех пор, пока функция стоимости не сойдется к минимуму.

Прогнозирование:

После обучения модели вы можете использовать её для прогнозирования класса новых объектов:

1. Получите входные данные x .
2. Рассчитайте $z = w_0x_0 + w_1x_1 + \dots + w_nx_n$.
3. Примените функцию логистической активации: $h(z)$.
4. Если $h(z) \geq 0.5$, прогнозируйте класс 1, иначе класс 0.

Замечания:

- Метод логистической регрессии похож на линейный дискриминант, но ослабляет вероятностные предположения.
- На практике (почти) никто не проверяет вероятностные предположения.
- По сравнению с линейным дискриминантом, метод логистической регрессии более устойчив к выбросам.
- Для повышения устойчивости требуется регуляризатор.

“Наивный” байесовский классификатор

Для каждого класса вычисляется вероятность того, что объект принадлежит к данному классу, без учета признаков объекта. (на основе данных и предварительных знаний.)

Для каждого признака и каждого класса вычисляется условная вероятность, что признак равен определенному значению при условии принадлежности к данному классу.

Вранье какое-то. Мы тут вместо многомерной функции плотности от признаков, берем функцию плотности каждой фичи. (Вопрос Коли)

Вычисляем по формуле (апостериорная вероятность):

$$P(\text{класс}|\text{признаки}) = \frac{P(\text{признаки}| \text{класс}) \times P(\text{класс})}{P(\text{признаки})}$$

где:

- $P(\text{класс}|\text{признаки})$ - апостериорная вероятность принадлежности к классу,
- $P(\text{признаки}| \text{класс})$ - условная вероятность признаков при условии принадлежности к классу,
- $P(\text{класс})$ - априорная вероятность класса,
- $P(\text{признаки})$ - вероятность признаков.

Назначить объекту класс с наивысшей апостериорной вероятностью.

Из формулы Байеса можем записать

$$g(x) = P(y=1|x) = \frac{P(dx, y=1)}{P(dx, y=1) + P(dx, y=-1)} = \frac{1}{1 + \frac{1-p}{p} \cdot \frac{P(dx|y=-1)}{P(dx|y=1)}},$$

где $p = P(1)$.

Независимость переменных

Пусть переменные независимы, т.е.

$$P(dx|y) = \prod_{j=1}^n P(dx_j|y).$$

После подстановки в предыдущее выражение и преобразований имеем

$$\frac{p}{1-p} \cdot \left(\frac{1}{g(x)} - 1 \right) = \prod_{j=1}^n \frac{p}{1-p} \cdot \left(\frac{1}{g_j(x_j)} - 1 \right),$$

где $g_j(x_j) = P(y=1|x_j)$.

Решающая функция

Логарифмируем последнее выражение и получаем

$$\sigma^{-1}(g(x)) = (n-1)(\ln p - \ln(1-p)) + \sum_{j=1}^n \sigma^{-1}(g_j(x_j))$$

Получили решающую функцию в форме логистической регрессии

$$g(x) = \sigma \left(w_0 + \sum_{j=1}^n w_j z_j \right),$$

при $w_0 = (n-1)(\ln p - \ln(1-p))$, $w_j \equiv 1$, $z_j = \sigma^{-1}(g_j(x_j))$.

Замечания:

- **Предположение о независимости переменных эквивалентно аддитивности в пространстве преобразованных переменных.**
- **Полученное преобразование является разумным вариантом target encoding, может использоваться и для вещественных переменных.**
- **Наивный байесовский классификатор является частным случаем логистической регрессии с использованием target encoding.** (target encoding - значение категориальной переменной заменяется вероятностью принадлежности объекта к классу при данном значении переменной)
- **Предположение о (строгой) независимости переменных обычно оказывается неоправданно сильным, однако в ослабленном виде гипотеза независимости очень полезна.**

Линейный Дискриминантный Анализ (LDA)

<http://www.ccas.ru/voron/download/Bayes.pdf#page=14>

Страница 15-16 конкретно про линейную и квадратичную разделяющие поверхности

Основная Идея: LDA предполагает, что разные классы генерируются из гауссовых распределений с одинаковыми ковариационными матрицами, но разными средними. Это означает, что классы имеют одинаковую форму, но они смещены относительно друг друга в пространстве признаков.

Цель: Найти линейную комбинацию признаков, которая наилучшим образом разделяет классы. Это достигается путем максимизации отношения межклассовой дисперсии к внутриклассовой дисперсии в этом пространстве.

Процедура:

- Вычисляется среднее каждого признака для каждого класса.
- Вычисляется общая внутриклассовая ковариационная матрица.
- Находятся линейные коэффициенты, которые максимизируют отношение межклассовой дисперсии к внутриклассовой.

Классификация: Объект классифицируется в тот класс, к центру которого он ближе в пространстве, определенном этими коэффициентами.

Квадратичный Дискриминантный Анализ (QDA)

<http://www.ccas.ru/voron/download/Bayes.pdf#page=14>

[Страница 15-16 конкретно про линейную и квадратичную разделяющие поверхности](#)

Основная Идея: QDA, в отличие от LDA, не предполагает равенства ковариационных матриц классов. Это позволяет классам иметь различные формы, не только разные положения.

Цель: Так же, как и в LDA, цель состоит в том, чтобы найти функцию, разделяющую классы, но теперь граница между классами может быть квадратичной (или криволинейной), а не обязательно линейной.

Процедура:

- Вычисляются средние и ковариационные матрицы для каждого класса отдельно.
- Используя эти параметры, формируется квадратичная функция для разделения классов.

Классификация: Как и в LDA, объекты классифицируются на основе их расстояний до центров классов, но расчет этих расстояний учитывает различные формы распределений классов.

Дискриминант Фишера (Линейный дискриминант Фишера)

<http://www.ccas.ru/voron/download/Bayes.pdf#page=20>

Основная Идея: Цель дискриминанта Фишера - найти такое линейное преобразование признаков, которое максимизирует разделение между разными классами. Это достигается путем максимизации отношения межклассовой дисперсии к внутриклассовой дисперсии.

Процесс:

- Вычисляются средние векторы для каждого класса.
- Вычисляется внутриклассовая и межклассовая ковариационные матрицы.
- Находится направление (вектор), вдоль которого проекция данных дает максимальное разделение классов.

Применение: Полученный вектор используется для проекции многомерных данных на меньшее пространство (часто одномерное), облегчая классификацию.

Логистическая Регрессия

<http://www.ccas.ru/voron/download/Regression.pdf#page=29>

Основная Идея: Логистическая регрессия моделирует вероятность принадлежности к определенному классу как логистическую функцию от линейной комбинации входных признаков. Это один из основных методов для бинарной классификации.

Процесс:

- Модель строит линейную комбинацию входных признаков (аналогично линейной регрессии).
- Применяется логистическая (сигмоидная) функция к результату линейной комбинации, преобразуя его в вероятность (от 0 до 1).
- Пороговое значение (часто 0.5) используется для принятия решения о классификации.

Различие: В отличие от дискриминанта Фишера, логистическая регрессия не предполагает нормальное распределение данных и может быть использована даже если предположения о гомоскедастичности (одинаковых ковариационных матрицах) нарушены.

Наивный Байесовский Классификатор

<http://www.ccas.ru/voron/download/Bayes.pdf#page=8>

Основная Идея: Наивный байесовский классификатор основан на применении теоремы Байеса с "наивным" предположением о независимости всех признаков. То есть, он предполагает, что значение одного признака никак не влияет на значение другого признака.

Процесс:

- Для каждого класса вычисляется априорная вероятность его появления в данных.
- Оцениваются вероятности каждого признака внутри каждого класса.
- При классификации нового примера вычисляется вероятность принадлежности к каждому классу на основе априорных вероятностей и вероятностей признаков.

Применение: Широко используется в задачах с дискретными признаками, таких как фильтрация спама, анализ текста и классификация документов.

Машина Опорных Векторов (SVM)

[Лекция Воронцова](#)

Основная Идея: SVM стремится найти гиперплоскость, которая наилучшим образом разделяет два класса данных, максимизируя зазор (маржу) между ближайшими к этой гиперплоскости точками каждого класса, называемыми опорными векторами.

Процесс:

- Выбирается гиперплоскость, которая максимизирует расстояние до ближайших точек каждого класса (опорных векторов).
- Для нелинейно разделимых данных используется метод ядер, который позволяет проецировать данные в более высокоразмерное пространство, где они могут быть линейно разделены.
- Решение задачи оптимизации приводит к определению наилучшей разделяющей гиперплоскости.

Применение: SVM используется во множестве областей, от распознавания образов и классификации изображений до анализа биомедицинских данных.

Лекция 3

Метод опорных векторов (Support Vector Machine - SVM)

Лекция Воронцова

ИТМО

ДЗЕН (поверхностно и кратко)

Метод обобщенного портрета (6070-е годы, В.Н. Вапник и др.): отдалить объекты от разделяющей поверхности. В 90-е годы метод стал называться машиной опорных векторов (support vector machine, SVM).

Свойства:

1. сводится к эффективно решаемой задаче квадратичного программирования,
2. разреженность (решение определяется опорными векторами),
3. обобщается введением функции ядра.

X = Rⁿ пространство значений прогнозирующих переменных,

Y = {-1, 1} - прогнозируемая переменная,

D = X x Y .

Решающая функция (алгоритм классификации) f : X → Y.

V = {(x_i, y_i) ∈ D | i = 1, N} случайная независимая выборка, V ⊂ D^N .

Q: D^N → Φ - метод построения решающих функций, Φ - заданный класс решающих функций.

Линейные классификаторы

Линейно разделимая выборка

где $I(\cdot)$ – индикаторная функция.

Линейный пороговый классификатор

Эмпирический риск:

$$f(x) = \text{sign}(wx - w_0).$$

$$\tilde{R}(V, f) = \frac{1}{N} \sum_{i=1}^N I(y^i \neq f(x^i)),$$

Требуется найти вектор w и скаляр w_0 , минимизирующие эмпирический риск при дополнительных требованиях.

Критерий

Нормировка

$$\min_{(x^i, y^i) \in V} y^i (x^i w - w_0) = 1.$$

Для граничных точек

$$x_+ w - w_0 = 1, \quad -(x_- w - w_0) = 1.$$

Ширина разделяющей полосы

$$(x_+ - x_-) \cdot \frac{w}{|w|} = \frac{(w_0 + 1) - (w_0 - 1)}{|w|} = \frac{2}{|w|}.$$

Оптимизационная задача

Задача квадратичной оптимизации

$$\begin{cases} w^2 \rightarrow \min_{w, w_0} \\ y^i (x^i w - w_0) \geq 1, \quad i = 1, \dots, N. \end{cases}$$

Условие нормировки выполняется автоматически.

Линейно неразделимая выборка

Задача квадратичного программирования

$$\begin{cases} \frac{w^2}{2} + C \sum_{i=1}^N \xi_i \rightarrow \min_{w, w_0, \xi} \\ y^i(x^i w - w_0) \geq 1 - \xi_i, \quad i = 1, \dots, N, \\ \xi_i \geq 0, \quad i = 1, \dots, N, \end{cases}$$

где $C > 0$ – параметр.

Применение метода

Оптимизационную задачу можно решать методом активных ограничений (incremental active set method, INCAS), частным случаем которого является симплекс-метод.

Константу C обычно выбирают по критерию скользящего контроля.

Задача сводится в задаче для линейно разделимой выборки, когда $C \rightarrow \infty$.

INCAS (incremental active set method)

Метод активных ограничений (Active Set Method) – это численный метод оптимизации, который применяется для решения задач оптимизации с ограничениями. Он особенно полезен при решении задач квадратичного программирования (QP), где целевая функция является квадратичной, а ограничения могут быть линейными или нелинейными.

Принцип метода активных ограничений заключается в том, чтобы пошагово строить оптимальное решение, активируя или деактивируя ограничения. На каждом шаге рассматриваются только подмножество ограничений, которые вероятно будут активными в оптимальном решении. Это позволяет уменьшить вычислительную сложность задачи оптимизации, поскольку неактивные ограничения не участвуют в вычислениях.

1. Инициализация:

- Начните с некоторого начального приближения к решению.

2. Выбор активного множества:

- Определите, какие ограничения вероятно будут активными в текущем оптимальном решении.

3. Решение подзадачи:

- Решите подзадачу, которая включает только активные ограничения.

4. Проверка условий останова:

- Проверьте условия останова, чтобы определить, достигнуто ли оптимальное решение.

5. Обновление активного множества:

- Обновите активное множество, добавляя или удаляя ограничения в зависимости от текущего решения.

6. Повторение:

- Повторяйте шаги 2-5 до достижения оптимального решения или условий останова.

Функция Лагранжа

Задача на нахождение экстремума Условие касания линий уровня

$$\begin{cases} f(x) \rightarrow \min_x \\ \psi(x) = 0. \end{cases} \quad \frac{\partial f(x)}{\partial x} = \lambda \frac{\partial \psi(x)}{\partial x}.$$

После переноса в левую часть получаем функцию Лагранжа.

Теорема (Условия Каруша—Куна—Таккера):

Пусть поставлена задача нелинейного программирования с ограничениями:

$$\begin{cases} f(x) \rightarrow \min_{x \in X} \\ g_i(x) \leq 0, i = 1 \dots m \\ h_j(x) = 0, j = 1 \dots k \end{cases}$$

Если x — точка локального минимума при наложенных ограничениях, то существуют такие множители $\mu_i, i = 1 \dots m, \lambda_j, j = 1 \dots k$, что для функции Лагранжа $L(x; \mu, \lambda)$ выполняются условия:

$$\begin{cases} \frac{\partial L}{\partial x} = 0, \quad L(x; \mu, \lambda) = f(x) + \sum_{i=1}^m \mu_i g_i(x) + \sum_{j=1}^k \lambda_j h_j(x) \\ g_i(x) \leq 0, h_j(x) = 0 \quad (\text{исходные ограничения}) \\ \mu_i \geq 0 \quad (\text{двойственные ограничения}) \\ \mu_i g_i(x) = 0 \quad (\text{условие дополняющей нежёсткости}) \end{cases}$$

При этом искомая точка является седловой точкой функции Лагранжа: минимумом по x и максимумом по двойственным переменным μ .

Функция Лагранжа для SVM

$$\mathfrak{L}(w, w_0, \xi; \lambda, \eta) = \frac{w^2}{2} + C \sum_{i=1}^N \xi_i - \sum_{i=1}^N \lambda_i (y^i (x^i w - w_0) - 1 + \xi_i) - \sum_{i=1}^N \xi_i \eta_i.$$

После преобразований

$$\mathfrak{L}(w, w_0, \xi; \lambda, \eta) = \frac{w^2}{2} - \sum_{i=1}^N \lambda_i (y^i (x^i w - w_0) - 1) - \sum_{i=1}^N \xi_i (\lambda_i + \eta_i - C).$$

Условия стационарности

Дифференцируем

$$\begin{cases} \frac{\partial \mathfrak{L}}{\partial w} = 0, & w = \sum_{i=1}^N \lambda_i y^i x^i, \\ \frac{\partial \mathfrak{L}}{\partial w_0} = 0, & \sum_{i=1}^N \lambda_i y^i = 0, \\ \frac{\partial \mathfrak{L}}{\partial \xi_i} = 0, & \eta_i + \lambda_i = C, \quad i = 1, \dots, N. \end{cases}$$

Двойственная задача

Выразим всё через λ_i

$$\begin{cases} -\mathfrak{L}(\lambda) = \sum_{i=1}^N \lambda_i + \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \lambda_i \lambda_j y^i y^j x^i x^j \rightarrow \min_{\lambda} \\ 0 \leq \lambda_i \leq C, \quad i = 1, \dots, N, \\ \sum_{i=1}^N \lambda_i y^i = 0. \end{cases}$$

Точки, для которых $\lambda_i > 0$, называются опорными векторами. Заметим, x^i что входят только через скалярные произведения.

Итоговый классификатор имеет вид

$$f(x) = \text{sign} \left(\sum_{i=1}^N \lambda_i y^i x^i - w_0 \right).$$

Ядра и спрямляющие пространства

Ядро (англ. *kernel*) — функция $K : X \times X \rightarrow \mathbb{R}$, которая является скалярным произведением в некотором спрямляющем пространстве: $K(\vec{x}_1, \vec{x}_2) = \langle \psi(\vec{x}_1), \psi(\vec{x}_2) \rangle$ при некотором $\psi : X \rightarrow H$, где H — пространство со скалярным произведением.

Спрямляющее пространство (feature space) — это новое пространство, которое получается в результате применения функции (ядра) к исходным признакам данных.

Идея заключается в том, что SVM может легко обрабатывать нелинейные зависимости между признаками, преобразуя их в более высокоуровневые (спрямляющие) пространства. Это достигается с использованием ядровых функций (kernel functions).

Kernel-trick

Существует ещё один подход к решению проблемы линейной разделимости, известный как трюк с ядром (kernel trick). Если выборка объектов с признаковым описанием из $X = \mathbb{R}^n$ не является линейно разделимой, мы можем предположить, что существует некоторое пространство H , вероятно, большей размерности, при переходе в которое выборка станет линейно разделимой. Пространство H здесь называют спрямляющим, а функцию перехода $\psi : X \rightarrow H$ — спрямляющим отображением. Построение SVM в таком случае происходит так же, как и раньше, но в качестве векторов признаковых описаний используются векторы $\psi(\vec{x})$, а не \vec{x} . Соответственно, скалярное произведение $\langle \vec{x}_1, \vec{x}_2 \rangle$ в пространстве X всегда заменяется скалярным произведением $\langle \psi(\vec{x}_1), \psi(\vec{x}_2) \rangle$ в пространстве H . Отсюда следует, что пространство H должно быть гильбертовым, так как в нём должно быть определено скалярное произведение.

Обратим внимание на то, что постановка задачи и алгоритм классификации не используют в явном виде признаковое описание и оперируют только скалярными произведениями признаков объектов. Это даёт возможность заменить скалярное произведение в пространстве X на ядро — функцию, являющуюся скалярным произведением в некотором H . При этом можно вообще не строить спрямляющее пространство в явном виде, и вместо подбора ψ подбирать непосредственно ядро.

Постановка задачи с применением ядер приобретает вид:

$$\begin{cases} -\mathcal{L}(\lambda) = -\sum_{i=1}^{\ell} \lambda_i + \frac{1}{2} \sum_{i=1}^{\ell} \sum_{j=1}^{\ell} \lambda_i \lambda_j y_i y_j K(\vec{x}_i, \vec{x}_j) \rightarrow \min_{\lambda} \\ 0 \leq \lambda_i \leq C, \quad i = 1, \dots, \ell \\ \sum_{i=1}^{\ell} \lambda_i y_i = 0 \end{cases}$$

$$a(x) = \text{sign} \left(\sum_{i=1}^{\ell} \lambda_i y_i K(\vec{x}_i, \vec{x}) - b \right)$$

Существует ещё один подход к решению проблемы линейной разделимости, известный как трюк с ядром (kernel trick). Если выборка объектов с признаковым описанием из $X = \mathbb{R}^n$ не является линейно разделимой, мы можем предположить, что существует некоторое пространство H , вероятно, большей размерности, при переходе в которое выборка станет линейно разделимой. Пространство H здесь называют спрямляющим, а функцию перехода $\psi : X \rightarrow H$ — спрямляющим отображением. Построение SVM в таком случае происходит так же, как и раньше, но в качестве векторов признаковых описаний используются векторы $\psi(\vec{x})$, а не \vec{x} . Соответственно, скалярное произведение $\langle \vec{x}_1, \vec{x}_2 \rangle$ в пространстве X всегда заменяется скалярным произведением $\langle \psi(\vec{x}_1), \psi(\vec{x}_2) \rangle$ в пространстве H . Отсюда следует, что пространство H должно быть гильбертовым, так как в нём должно быть определено скалярное произведение.

Обратим внимание на то, что постановка задачи и алгоритм классификации не используют в явном виде признаковое описание и оперируют только скалярными произведениями признаков объектов. Это даёт возможность заменить скалярное произведение в пространстве X на ядро — функцию, являющуюся скалярным произведением в некотором H . При этом можно вообще не строить спрямляющее пространство в явном виде, и вместо подбора ψ подбирать непосредственно ядро.

Постановка задачи с применением ядер приобретает вид:

$$\begin{cases} -\mathcal{L}(\lambda) = -\sum_{i=1}^{\ell} \lambda_i + \frac{1}{2} \sum_{i=1}^{\ell} \sum_{j=1}^{\ell} \lambda_i \lambda_j y_i y_j K(\vec{x}_i, \vec{x}_j) \rightarrow \min_{\lambda} \\ 0 \leq \lambda_i \leq C, \quad i = 1, \dots, \ell \\ \sum_{i=1}^{\ell} \lambda_i y_i = 0 \end{cases}$$

$$a(x) = \text{sign} \left(\sum_{i=1}^{\ell} \lambda_i y_i K(\vec{x}_i, \vec{x}) - b \right)$$

Открытые проблемы (реально) всё-равно

- Как соотносится SVM с метрическими методами.
- Возможно ли придумать обобщение SVM, включающее дискриминант Фишера как частный случай.
- Является ли разреженность достоинством.
- Как выбирать ядро.
- Как применить kernel trick в логистической регрессии.
- Низкая эффективность при сильно перекрывающихся распределениях классов.

Общность линейных методов

- Дискриминант Фишера не требует вероятностных предположений, но по результату почти совпадает с линейным дискриминантом, основанным на нормальности.
- Наивный байесовский классификатор является частным случаем логистической регрессии.
- Метод SVM оказывается близок логистической регрессии ввиду сходства функций потерь.

Если кратко

Основная Идея: SVM стремится найти гиперплоскость, которая наилучшим образом разделяет два класса данных, максимизируя зазор (маржу) между ближайшими к этой гиперплоскости точками каждого класса, называемыми опорными векторами.

Процесс:

- Выбирается гиперплоскость, которая максимизирует расстояние до ближайших точек каждого класса (опорных векторов).
- Для нелинейно разделимых данных используется метод ядер, который позволяет проецировать данные в более высокоразмерное пространство, где они могут быть линейно разделены.
- Решение задачи оптимизации приводит к определению наилучшей разделяющей гиперплоскости.

Применение: SVM используется во множестве областей, от распознавания образов и классификации изображений до анализа биомедицинских данных.

Лекция 4

Поиск закономерностей, решающие списки

Понятие закономерности

X – пространство значений прогнозирующих переменных,
 $Y = \{0, 1, \dots\}$ – прогнозируемая переменная, $\varphi : X \rightarrow \{0, 1\}$ – предикат.

$V = ((x^i, y^i) \mid i = \overline{1, N})$ – выборка объектов,

M – из них 1-го класса,

n – число точек, на которых предикат истинный,

m – из них 1-го класса.

«Хороший» предикат – закономерность:

$$a = \frac{m}{M} \rightarrow \max, \quad b = \frac{n - m}{n} \rightarrow \min.$$

а и b - метрики информативности

Схема направленного поиска закономерностей («жадный» алгоритм):

- дискретизация признаков (границы между проекциями точек выборки);
- интервальные предикаты;
- конъюнкции элементарных предикатов.

Известные алгоритмы:

- КОРА,
- ТЭМП.

Алгоритм Кора

1. Определение логических высказываний:

- Вводится множество характеристических логических функций $Y(x, t)$, называемых логическими высказываниями. Эти высказывания представляют собой комбинации исходных переменных x_{ij} , объединенных операцией конъюнкции (логическое И).

2. Генерация непротиворечивых логических высказываний:

- Алгоритм многократно просматривает обучающую выборку, разделенную на два класса (0 и 1).
- С использованием операций алгебры логики алгоритм выделяет непротиворечивые логические высказывания $Y(x, t^*)$, которые покрывают всю выборку.
- Непротиворечивыми считаются высказывания, которые встречаются только в одном классе и не встречаются в другом.

3. Правила генерации логических высказываний:

- Конъюнкции сортируются по продуктивности (количество наблюдений, для которых высказывание справедливо).
- Из списка исключаются подчиненные конъюнкции, полностью содержащие более короткие претенденты.
- Исключаются высказывания, считающиеся "предрассудками", используя bootstrap-процедуру.

4. Оценка прогностической ценности:

- Конъюнкции включаются в решающее правило на основе их продуктивности, превышающей порог D .

5. Исключение лишних конъюнкций:

- Избыточные конъюнкции, полностью содержащие более короткие претенденты, исключаются.

6. Исключение "предрассудков":

- Исключаются высказывания, считающиеся предрассудками, на основе bootstrap-процедуры.

7. Описание каждого класса:

- Каждый класс описывается логической суммой непротиворечивых и продуктивных конъюнкций.

8. Применение в экзамене тестируемых примеров:

- Конъюнкции могут использоваться для экзамена тестируемых примеров по принципу голосования.

Алгоритм Кора

Алгоритм TEMP

TODO: Алгоритм Тэмп Not found

(единственное что нашёл)

Алгоритм TEMP — это эвристический подход к синтезу деревьев решений, которые представляют собой модели машинного обучения, используемые для задач классификации и прогнозирования. Одной из проблем этого алгоритма является проблема дублирования закономерностей, которая возникает, когда несколько предложений в дереве решений охватывают один и тот же набор примеров и делают один и тот же прогноз. Это может привести к неоптимальной производительности, поскольку снижает общую информативность дерева решений и может привести к переподгонке к обучающим данным.

Чтобы решить эту проблему, алгоритм TEMP использует этап сокращения для удаления повторяющихся предложений из дерева решений. Это делается путем определения предложений, которые охватывают один и тот же набор примеров и имеют одинаковое предсказание, а затем удаления всех этих предложений, кроме одного, из дерева. Это помогает повысить производительность дерева решений за счет уменьшения избыточности и повышения общей информативности включенных предложений. Кроме того, сокращение может помочь предотвратить переобучение, удаляя слишком сложные или специфические предложения, которые могут плохо обобщаться на новые данные.

Метод классификации на основе списка закономерностей.

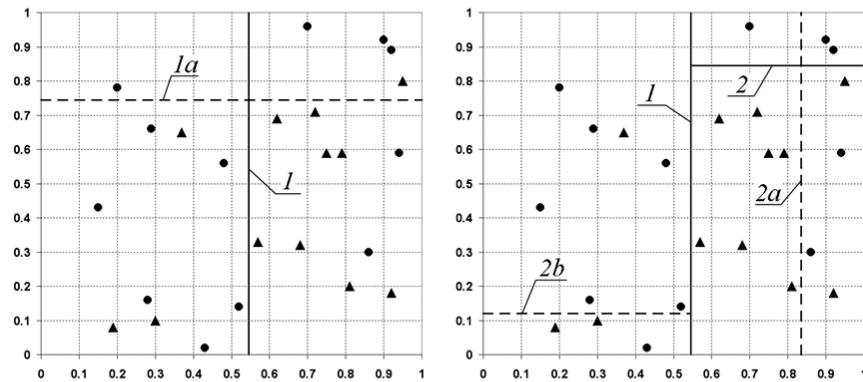
- Формируем упорядоченный по информативности список закономерностей.
- Решение принимаем по первой закономерности, которой удовлетворяет объект.

Можно применять голосование.

Решающие деревья

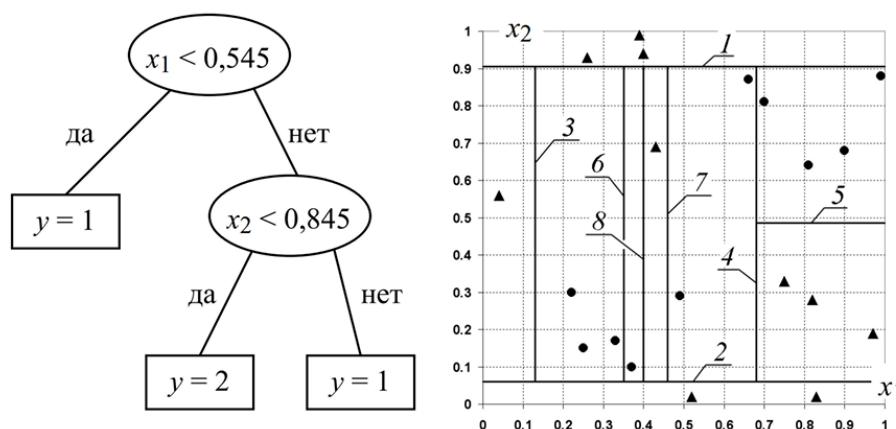
<https://scikit-learn.org/stable/modules/tree.html#tree>

Процесс построения дерева для задачи Iris



Дерево представляет собой последовательное разбиение пространства значений на области.

Представление дерева



Можно получить не лучшее дерево (пример справа для задачи XOR).

Критерии ветвления

Критерий ветвления основывается на критерии качества дерева (строится путём сравнения вариантов дерева).

Критерии оценки дерева (для классификации):

- число ошибок классификации (для детерминированного прогноза),
- критерий Джини (Gini impurity) — это по сути число ошибок для вероятностного прогноза,
- информационный (log loss, на основе функции правдоподобия для вероятностного прогноза).

Первый критерий работает плохо, остальные сопоставимо.

Критерий Джини

Критерий Джини

$$G = 1 - p_k^2,$$

где p_k — доля объектов принадлежащих k -классу.

Log Loss

Для бинарной классификации **Log Loss** выражается формулой:

$$\text{LogLoss} = -\frac{1}{N} \sum_{i=1}^n (y_i \log(p_i) + (1 - y_i) \log(1 - p_i))$$

, где:

- N - количество примеров в выборке,
- y_i - фактическая метка класса для примера i (0 или 1),
- p_i - вероятность, предсказанная моделью, что пример i принадлежит к классу 1.

В задаче классификации с двумя непересекающимися классами (0, 1), когда ответ вероятность (?) принадлежности к классу 1

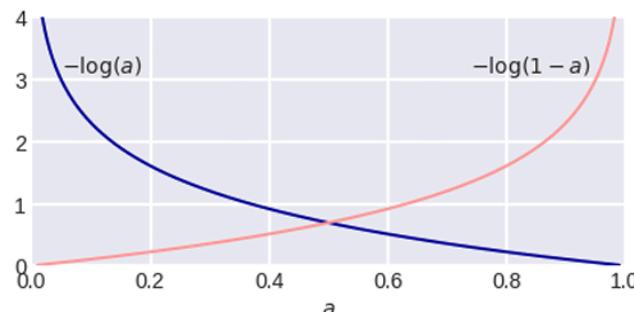
$$\text{LOGLOSS} = -\frac{1}{q} \sum_{i=1}^q (y_i \log a_i + (1 - y_i) \log(1 - a_i))$$

На что похоже?

Так понятнее...

$$-\begin{cases} \log a_i, & y_i = 1, \\ \log(1 - a_i), & y_i = 0. \end{cases}$$

Нельзя ошибаться!



Варианты алгоритма построения дерева:

- жадный (без учёта будущих ветвлений),
- рекурсивный (предикат в узле выбирается с учётом ветвления на нижнем уровне, решает задачу XOR),
- неограниченный (строит дерево, затем его оптимизирует всё доступное время).

Сейчас на практике используется «жадный» алгоритм.

Жадный алгоритм

Ссылочка

Пусть X — исходное множество объектов обучающей выборки, а X_m — множество объектов, попавших в текущий лист (в самом начале $X_m = X$). Тогда жадный алгоритм можно верхнеуровнево описать следующим образом:

- 1 Создаём вершину v .
- 2 Если выполнен критерий остановки $Stop(X_m)$, то останавливаемся, объявляем эту вершину листом и ставим ей в соответствие ответ $Ans(X_m)$, после чего возвращаем её.
- 3 Иначе: находим предикат (иногда ещё говорят сплит) $B_{j,t}$, который определит наилучшее разбиение текущего множества объектов X_m на две подвыборки X_ℓ и X_r , максимизируя критерий ветвления $Branch(X_m, j, t)$.
- 4 Для X_ℓ и X_r рекурсивно повторим процедуру.
 - $Ans(X_m)$, вычисляющая ответ для листа по попавшим в него объектам из обучающей выборки, может быть, например:
 - в случае задачи классификации — меткой самого частого класса или оценкой дискретного распределения вероятностей классов для объектов, попавших в этот лист;
 - в случае задачи регрессии — средним, медианой или другой статистикой;
 - простой моделью. К примеру, листы в дереве, задающем регрессию, могут быть линейными функциями или синусоидами, обученными на данных, попавших в лист. Впрочем, везде ниже мы будем предполагать, что в каждом листе просто предсказывается константа.
 - Критерий остановки $Stop(X_m)$ — функция, которая решает, нужно ли продолжать ветвление или пора остановиться.
 - Критерий ветвления $Branch(X_m, feature, value)$ — пожалуй, самая интересная компонента алгоритма. Это функция, измеряющая, насколько хорош предлагаемый сплит. Чаще всего эта функция оценивает, насколько улучшится некоторая финальная метрика качества дерева в случае, если получившиеся два листа будут терминальными, по сравнению с ситуацией, когда сама исходная вершина является листом. Выбирается такой сплит, который даёт наиболее существенное улучшение. Впрочем, есть и другие подходы.

Рекурсивный алгоритм

1. Выбор признаков и порогов:

- **Жадный алгоритм:** Принимает локально оптимальное решение на каждом этапе, выбирая признак и порог такие, чтобы максимизировать прирост информации или минимизировать критерий ошибки на текущем уровне дерева.
- **Рекурсивный алгоритм:** Производит выбор признаков и порогов рекурсивно на каждом уровне дерева, принимая во внимание структуру данных в текущем подмножестве, а не только локальные оптимизации.

2. Построение дерева:

- **Жадный алгоритм:** Дерево строится последовательно, принимая локально оптимальные решения на каждом этапе. На каждом уровне решается оптимизационная задача для максимизации прироста информации.
- **Рекурсивный алгоритм:** Процесс построения дерева рекурсивен, и на каждом шаге разделяется текущее подмножество данных на две части, и для каждой из них вызывается рекурсивно тот же алгоритм.

3. Остановка рекурсии:

- **Жадный алгоритм:** Остановка происходит в соответствии с заранее заданными условиями, такими как максимальная глубина дерева или минимальное количество элементов в узле.
- **Рекурсивный алгоритм:** Остановка рекурсии происходит, когда выполняются условия остановки, такие как достижение максимальной глубины, недостаточное количество элементов в узле или однородность подмножества данных.

4. Эффективность:

- **Жадный алгоритм:** Может быть менее эффективен, так как на каждом этапе принимается локальное оптимальное решение без учета будущих последствий.
- **Рекурсивный алгоритм:** Может обеспечивать более глубокое исследование структуры данных, что может привести к более сложному и точному дереву.

Неограниченный алгоритм

Разбиение происходит до тех пор, пока в каждом узле не будет объектов только одного класса.
(из лекции: **строит дерево, а потом всё оставшееся время оптимизирует**).

Выводы

Выводы

Недостатки логических методов:

- невозможность «гладких» решений,
- много эвристик (настраиваемых параметров),
- вычислительная трудоёмкость нахождения точных решений,
- неизвестен лучший критерий ветвления.

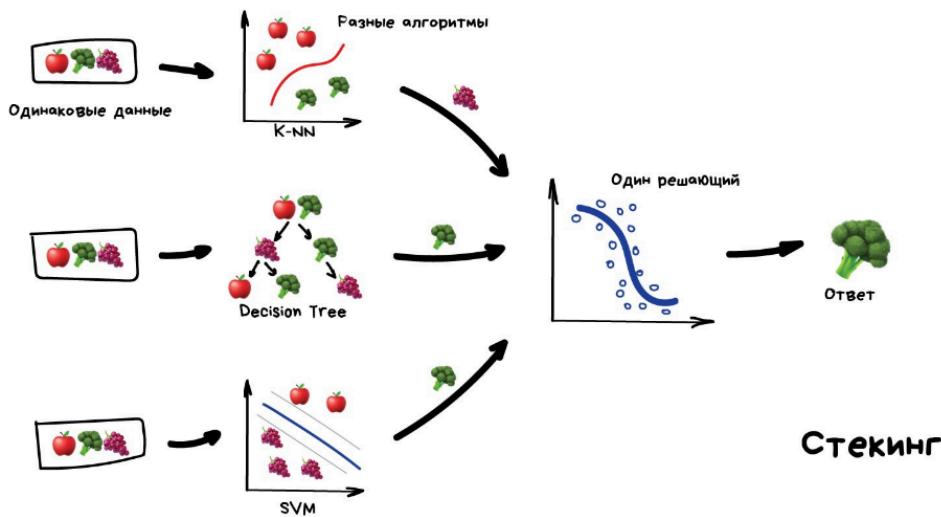
Актуальность:

- возможность получить интерпретируемое решение,
- используется в составе бустинга.

Лекция 5

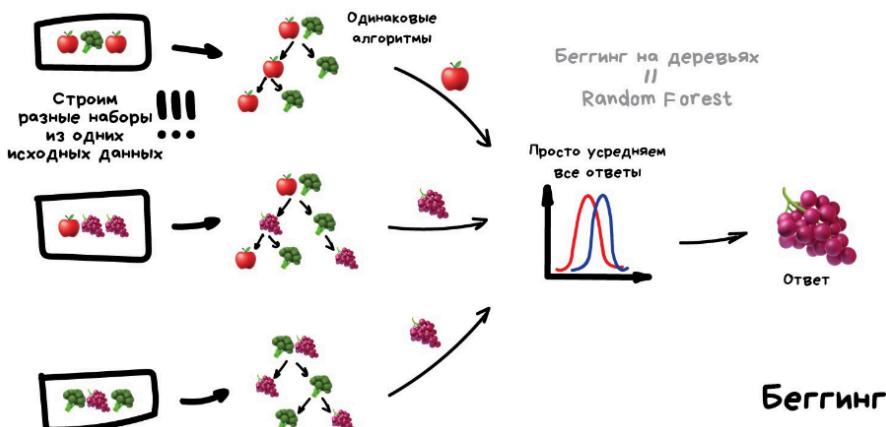
Композиции классификаторов

Стекинг Обучаем несколько разных алгоритмов и передаём их результаты на вход последнему, который принимает итоговое решение.



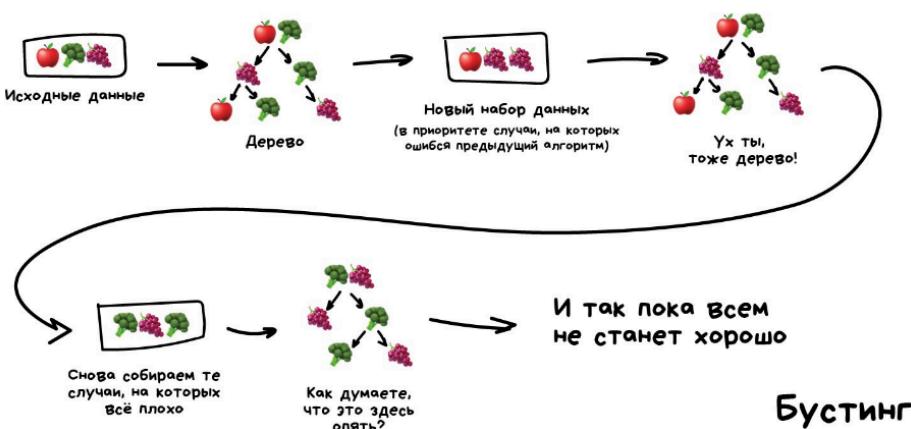
Бэггинг Он же Bootstrap AGGREGatIING. Обучаем один алгоритм много раз на случайных выборках из исходных данных. В самом конце усредняем ответы.

Данные в случайных выборках могут повторяться. То есть из набора 1-2-3 мы можем делать выборки 2-2-3, 1-2-2, 3-1-2 и так пока не надоест. На них мы обучаем один и тот же алгоритм несколько раз, а в конце вычисляем ответ простым голосованием.



Бустинг Обучаем алгоритмы последовательно, каждый следующий уделяет особое внимание тем случаям, на которых ошибся предыдущий.

Как в бэггинге, мы делаем выборки из исходных данных, но теперь не совсем случайно. В каждую новую выборку мы берём часть тех данных, на которых предыдущий алгоритм отработал неправильно. То есть как бы доучиваем новый алгоритм на ошибках предыдущего.



Бустинг

Линейная композиция

$$\lambda(x) = \text{sign} \left(\sum_{t=1}^T \alpha_t \lambda_t(x) \right), \quad \alpha_t \geq 0.$$

Методы построения композиций

- Бустинг (AdaBoost, градиентный бустинг)
- Бэггинг и метод случайных подпространств
- Голосование (простое, взвешенное)
- Стэкинг (решения в качестве новых переменных)

Смеси алгоритмов – если α_t зависит от x , идея областей компетентности.

Boosting

AdaBoost

ИТМО

В методе AdaBoost решение строится в виде композиции

$$\lambda(x) = \text{sign}(\beta(x)), \quad \beta(x) = \sum_{t=1}^T \alpha_t \lambda_t(x),$$

где базовые классификаторы $\lambda_t(x)$ и их веса α_t находятся следующим образом.

Первый базовый классификатор строится базовым методом на основе исходной выборки, объектам которой приписаны начальные веса $w^1 = (w_1^1, \dots, w_N^1)$.

Заметим, что мы будем задавать начальные веса объектом в соответствии с выбранным распределением, но в стандартном варианте метода начальные веса выбираются одинаковыми, т.е. $w_i^1 = \frac{1}{N}$.

Следующие базовые классификаторы строятся тем же базовым методом по выборке, веса объектов в которой вычисляются по формулам

$$w_i^{t+1} = \frac{\bar{w}_i^{t+1}}{\sum_{i=1}^N \bar{w}_i^{t+1}}, \quad \bar{w}_i^{t+1} = w_i^t \cdot e^{-\alpha_t y^i \lambda_t(x^i)}.$$

Веса правильно классифицированных объектов умножаются на $e^{-\alpha_t}$, а веса неправильно классифицированных объектов умножаются на e^{α_t} .

Условную вероятность $g(x) = P(y = 1 | x)$ представим как находящиеся в точке x два объекта: класса 1 с весом $w_0 g(x)$ и класса -1 с весом $w_0(1 - g(x))$.

В результате выполнения бустинга вес первого объекта станет равным

$$w^{+1}(x) = w_0 g(x) \cdot A e^{-\beta(x)},$$

Вес построенного базового классификатора в композиции определяется по формуле

$$\alpha_t = \frac{1}{2} \ln \frac{\widetilde{M}^+(V, w^t, \lambda_t)}{\widetilde{M}^-(V, w^t, \lambda_t)},$$

$$\widetilde{M}^+(V, w, \lambda) = \sum_{i=1}^N w_i \cdot I(y^i = \lambda(x^i)),$$

$$\widetilde{M}^-(V, w, \lambda) = \sum_{i=1}^N w_i \cdot I(y^i = -\lambda(x^i)).$$

Если бустинг не останавливать, то он будет стремиться оценить функцию условной вероятности.

- Если в точке пространства находится один объект выборки, то эмпирическая условная вероятность соответствующего класса в этой точке равна 1.
- Бустинг приближает вероятность через логистическую функцию.

Если условные вероятности нигде не равны 0 или 1, то бустинг сходится (веса деревьев стремятся к 0).

Полезно исследовать поведение методов не только на выборке, но и на распределениях.

где константа A есть произведение всех нормировочных множителей.

Конечный вес второго объекта есть

$$w^{-1}(x) = w_0(1 - g(x)) \cdot A e^{\beta(x)}.$$

Если приравнять веса объектов, то получим

$$g(x) = \frac{1}{1 + e^{-2\beta(x)}}.$$

Градиентный бустинг

[Яндекс Хабр](#)

Зададимся целью получить композицию деревьев, которая оценила бы условную вероятность в форме логистической функции от суммы прогнозов, т.е.

$$g(x) = \frac{1}{1 + e^{-\beta(x)}}.$$

Выразим теперь логарифмическую функцию потерь

$$L(y, g(x)) = \begin{cases} -\ln g(x), & y = 1 \\ -\ln(1 - g(x)), & y = -1 \end{cases} = \ln \left(1 + e^{-y\beta(x)} \right).$$

Градиентный бустинг строит композицию $\beta(x)$, минимизируя функцию потерь на выборке.

Свойства бустинга

Обобщенный наивный байесовский классификатор

Ранее мы для наивного байесовского классификатора получили выражение в виде логистической регрессии

$$g(x) = \sigma \left(u_0 + \sum_{j=1}^n u_j \sigma^{-1}(g_j(x_j)) \right),$$

при $u_0 = (n - 1)(\ln p - \ln(1 - p))$, $u_j = 1$.

Обобщим это выражение, считая веса свободными параметрами и допуская произвольные оценочные функции. Получим

$$g(x) = \sigma \left(u_0 + \sum_{j=1}^n u_j s(x_j) \right).$$

Бустинг на пороговых классификаторах

Бустинг на пороговых классификаторах («пнях») является разновидностью обобщённого наивного байесовского классификатора.

Действительно, каждая $\lambda_t(x)$ в композиции

$$\beta(x) = \sum_{t=1}^T \alpha_t \lambda_t(x)$$

зависит только от одной переменной X_{it} , поэтому после группировки слагаемых выражение можно привести к виду

$$2\beta(x) = \sum_{i=1}^n u_i s(x).$$

Подставив в выражение для $g(x)$, получим искомый вид.

Бустинг на деревьях и ряд Бахадура

Модель можно естественным образом обобщить по аналогии с рядом Бахадура, включив возможность учитывать зависимости между переменными, последовательно добавляя парные зависимости, зависимости в тройках и т.д.

$$g(x) = \sigma \left(u_0 + \sum_{j=1}^n u_j s_j(x_j) + \sum_{j,k} u_{jk} s_{jk}(x_j, x_k) + \right. \\ \left. + \sum_{j,k,l} u_{jkl} s_{jkl}(x_j, x_k, x_l) + \dots \right).$$

Понятие отступа

Иногда для обоснования бустинга вводится понятие отступа.

Отступ есть

$$\theta = \frac{y\beta(x)}{\varkappa}, \quad \varkappa = \sum_{t=1}^T \alpha_t.$$

Из-за нормировки в виде \varkappa сложность композиции влияет на оценку риска.

Проклятие размерности

Для случая независимых переменных «проклятие» размерности превращается в преимущество:

- чем больше зависимых переменных — тем больше требуемый объём выборки,
- чем больше независимых переменных — тем меньше требуемая выборка.

С увеличением числа независимых переменных качество решения только растёт.

Бустинг и случайный лес

Методы существенно различаются в настройке.

- Для бустинга параметры сложности — глубина дерева и количество деревьев.
- Для random forest сложность не увеличивается с ростом числа деревьев.

Как правило, бустинг использует деревья меньшей глубины.

Как правило, бустинг достигает лучшего качества.

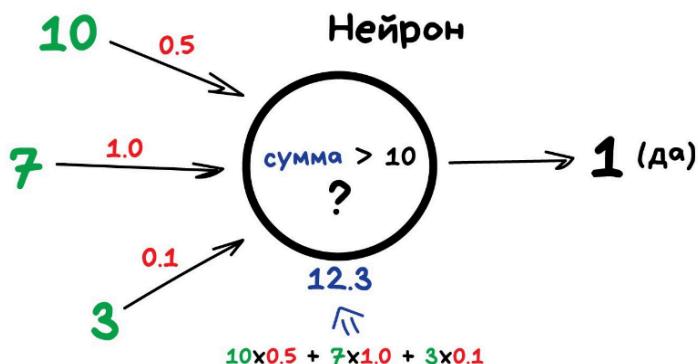
- Важнейшей причиной эффективности бустинга является использование эффекта независимости (переменных, подпространств, моделей).
- Бустинг на пороговых классификаторах является разновидностью непараметрической логистической регрессии, также его можно считать разновидностью (существенно обобщённого) наивного байесовского классификатора.
- Бустинг реализует «удачный» вариант непараметрической аппроксимации условной вероятности.

Лекция 6

Понятие нейрона

Любая нейросеть — это набор нейронов и связей между ними. Нейрон лучше всего представлять просто как функцию с кучей входов и одним выходом. Задача нейрона — взять числа со своих входов, выполнить над ними функцию и отдать результат на выход. Простой пример полезного нейрона: просуммировать все цифры со входов, и если их сумма больше N — выдать на выход единицу, иначе — ноль.

Связи — это каналы, через которые нейроны шлют друг другу циферки. У каждой связи есть свой вес — её единственный параметр, который можно условно представить как прочность связи. Когда через связь с весом 0.5 проходит число 10, оно превращается в 5. Сам нейрон не разбирается, что к нему пришло и суммирует всё подряд — вот веса и нужны, чтобы управлять на какие входы нейрон должен реагировать, а на какие нет.



Чтобы сеть не превратилась в анархию, нейроны решили связывать не как захочется, а по слоям. Внутри одного слоя нейроны никак не связаны, но соединены с нейронами следующего и предыдущего слоя. Данные в такой сети идут строго в одном направлении — от входов первого слоя к выходам последнего.

Всё что идёт далее из лекции

Реализация булевых функций

Функции И, ИЛИ, НЕ от бинарных переменных x^1 и x^2 :

$$x^1 \wedge x^2 = [x^1 + x^2 - \frac{3}{2} > 0];$$
$$x^1 \vee x^2 = [x^1 + x^2 - \frac{1}{2} > 0];$$
$$\neg x^1 = [-x^1 + \frac{1}{2} > 0];$$

The right side of the equation shows two summation nodes (\sum) representing the functions. The first node for AND ($x^1 \wedge x^2$) has inputs x^1 (weight 1), x^2 (weight 1), and a bias of -1.5. The second node for OR ($x^1 \vee x^2$) has inputs x^1 (weight 1), x^2 (weight 1), and a bias of -0.5. Both nodes output the result of the summation.

Апроксимирующая способность

- Двухслойная сеть может представить произвольную булеву функцию.
- Достаточно сложная сеть с заданной точностью аппроксимирует любую непрерывную функцию.

Архитектуры нейронных сетей

- Сети прямого распространения
 - полно связные
 - рекуррентные
 - сверточные
- Сети с памятью

Бустинг и нейронные сети

Отличие бустинга

- Делает отбор переменных
- Может эксплуатировать независимость
- Не корректирует уже построенные деревья

Свойства нейронных сетей

- Являются универсальным аппроксиматором.
- Почти все методы классификации можно описать как нейросеть, но нейросеть их не заменяет.
- Имеют меньшую гибкость в способах обучения и в выборе регуляризатора.
- Во многих задачах уступают бустингу.
- Во многих задачах не имеют конкурентных альтернатив.
- В задачах с нулевым байесовским уровнем ошибки переобучение менее критично.

Лекция 7

Риск

Решающей функцией (алгоритмом классификации) называется соответствие $\lambda: X \rightarrow Y$.

Качество принятого решения оценивается заданной функцией потерь $\mathcal{L}: Y^2 \rightarrow [0, \infty)$.

Положим $\mathcal{L}(y, y') = \begin{cases} 0, & y=y' \\ 1, & y \neq y' \end{cases}$.

Под риском будем понимать средние потери:

$$R(c, \lambda) = E\mathcal{L}(y, \lambda(x)) = \int_D \mathcal{L}(y, \lambda(x)) P_c(dx, dy),$$

$x \in X, y \in Y$.

Метод, минимизирующий эмпирический риск

Пусть $Q: D^N \rightarrow \Lambda$ — метод (алгоритм) построения решающих функций, $\lambda_{Q,V}$ — функция, построенная по выборке V методом Q , Λ — заданный класс решающих функций.

Метод \tilde{Q} , минимизирующий эмпирический риск, есть

$$\lambda_{\tilde{Q},V} = \arg \min_{\lambda \in \Lambda} \tilde{R}(V, \lambda).$$

Эмпирический риск

Функция потерь и эмпирический риск

Пусть $V = ((x^i, y^i) \in D \mid i = 1, \dots, N)$ — случайная независимая выборка из распределения P_c , $V \in D^N$.

Эмпирический риск определим как средние потери на выборке:

$$\tilde{R}(V, \lambda) = \frac{1}{N} \sum_{i=1}^N \mathcal{L}(y^i, \lambda(x^i)).$$

Функционал, поскольку функция от (решающей) функции (и выборки).

Свойства эмпирического риска

- простота вычисления,
- сильная смещённость,
- состоятельность при соответствующем ограничении на сложность метода (оценки Вапника-Червоненкиса),
- малая дисперсия.

Контрольная выборка

Пусть $V^* = \{(x^i, y^i) \in D \mid i = 1, \dots, N^*\}$ — «новая» случайная независимая выборка из распределения P_c , $V^* \in D^{N^*}$.

Оценку риска определим как средние потери на контрольной выборке:

$$R^*(V^*, \lambda) = \frac{1}{N^*} \sum_{i=1}^{N^*} \mathcal{L}(y^i, \lambda(x^i)).$$

Доверительный интервал в схеме Бернулли

Ссылка

Односторонний интервал $[0, \hat{p}]$ Двусторонний интервал $[p_1, p_2]$

$$\sum_{i=0}^M C_N^i \hat{p}^i \cdot (1 - \hat{p})^{N-i} = \alpha. \quad \sum_{i=0}^M C_N^i p_2^i \cdot (1 - p_2)^{N-i} = \sum_{i=M}^N C_N^i p_1^i \cdot (1 - p_1)^{N-i} = \frac{\alpha}{2}.$$

Байесовский подход

Положим равномерное $\varphi(p) \equiv 1$.

Формула Байеса

Используя нормировку, получаем

$$\varphi(p \mid M) = P(M \mid p) \frac{\varphi(p)}{P(M)}. \quad \varphi(p \mid M) = (N+1) C_N^M p^M \cdot (1-p)^{N-M}.$$

Можем вычислить

$$E_M p = \int_0^1 p \varphi(p \mid M) dp = \frac{M+1}{N+2}.$$

Усреднение по доверительной вероятности

Считаем $\eta(\hat{p}) = 1 - \alpha(\hat{p})$ функцией распределения.

Можно усреднить

$$\hat{E}_M p = \int_0^1 \hat{p} d\eta(\hat{p}) = \frac{M+1}{N+1}.$$

Запомните эти слова

Если нельзя, но очень хочется, то — можно.

Свойства оценки по контрольной выборке

- простота вычисления,
- несмещённость (не совсем в том смысле, в котором хотелось бы),
- состоятельность,
- известен точный доверительный интервал,
- эффективность (не в том смысле, в котором хотелось бы),
- требуют дополнительной выборки.

Скользящий экзамен

Ссылка

Функционал скользящего экзамена определяется как:

$$\check{R}(V, Q) = \frac{1}{N} \sum_{i=1}^N \mathcal{L}(y^i, \lambda_{Q, V'_i}(x^i)),$$

где $V'_i = V \setminus \{(x^i, y^i)\}$ — выборка, получаемая из V удалением i -го наблюдения,

Несмешенность скользящего экзамена

Теорема

$$E\check{R}(V_N, Q) = ER(V_{N-1}, Q).$$

Доказательство элементарно, хотя неочевидно.

Во-первых, используется факт, что математическое ожидание суммы есть сумма мат. ожиданий в т.ч. для зависимых случайных величин.

Во-вторых, используется несмешенность оценки hold-out. А поскольку кроссвалидация — это усреднение нескольких оценок hold-out, она также получается несмешенной.

Несмешенность оценки hold-out

Оценка hold-out строится на так называемой «отложенной» выборке.

Иногда считается, что это частный случай кроссвалидации, при котором разбиение на обучающую и контрольную подвыборки делается только один раз.

Технически hold-out выглядит как оценка по контрольной выборке (и часто используется как её синоним). Однако о контрольной выборке мы говорим, если оцениваем уже построенное решение. А метод hold-out предполагает, что полученную оценку мы будем переносить на решение, которое будет построено по полной выборке.

Если однако использовать hold-out выглядит как оценку текущего решения, то отложенная выборка эквивалентна контрольной, поэтому оценка получается несмешенной.

Cross-validation

K -fold cross-validation: исходная выборка разбивается на K равных частей (для простоты полагаем, что N кратно K).

$$\check{R}^K(V, Q) = \frac{1}{N} \sum_{i=1}^N \mathcal{L}(y^i, \lambda_{Q, V_i^K}(x^i)),$$

где V_i^K — выборка, получаемая из V удалением всей подвыборки, которой принадлежит i -е наблюдение.

Разновидности скользящего экзамена

- leave-one-out,
- k -fold crossvalidation,
- случайные подвыборки,
- со стратификацией (выравнивание частот классов по фолдам).

На практике различие в точности несущественно, поэтому достаточно использовать, например, 5-фолд (без стратификации).

Дисперсия оценки практически не зависит от числа фолдов (для детерминированных методов).

Свойства оценки скользящего экзамена

- относительная простота вычисления,
- несмещённость (не в том смысле, в котором хотелось бы),
- состоятельность (не доказана?),
- большая дисперсия, нет приемлемых оценок доверительного интервала для риска (есть эмпирические свидетельства, что точность сравнима с контролем половинной длины),
- вычислять разброс по фолдам не имеет практического смысла.

Оценка bootstrap

Ссылка

Оценка bootstrap есть

$$\check{R}(V, Q) = \frac{1}{E|J_0|} E \sum_{i \in J_0} \mathcal{L}(y^i, \lambda_{Q, \check{V}}(x^i)),$$

где \check{V} – выборка, получаемая из V путем N -кратного случайного (равновероятного) выбора ее значений с повторениями, J_0 – множество индексов объектов из V , ни разу не выбранных в \check{V} , математическое ожидание подразумевает усреднение по выборкам \check{V} .

Ввиду того, что оценка bootstrap является смененной, чаще используют ее в комбинации с эмпирическим риском

$$\check{R}(V, Q) = e^{-1} \tilde{R}(V, Q) + (1 - e^{-1}) \check{R}(V, Q).$$

Свойства оценки bootstrap

- относительная высокая трудоёмкость вычисления,
- приблизительная несмещённость,
- состоятельность (не доказана?),
- дисперсия неизвестна, но, вероятно, сопоставима со скользящим экзаменом.

Оценка out-of-bag

Ссылка

Используя кроссвалидацию (равно как и bootstrap), мы несколькими способами разбиваем выборку на обучения и контроль, и для каждого разбиения строим решающую функцию.

Далее предполагается, что в качестве итогового решения мы построим новую решающую функцию, уже по всей выборке. И свойства у неё уже не совсем как у кроссвалидации.

Однако мы можем в качестве итогового решения усреднить уже построенные решающие функции.

Таким образом, оценка out-of-bag – это та же кроссвалидация (или bootstrap), но с другим финальным

Свойства оценки out-of-bag

- относительная высокая трудоёмкость вычисления («бесплатна» для RandomForest),
- особенно актуальна для нейронных сетей (помимо RandomForest),
- приблизительная несмешённость (возможно, несколько пессимистична),
- состоятельность (не доказана?),
- дисперсия неизвестна, но, вероятно, сопоставима со скользящим экзаменом.

Гистограммный классификатор

Пусть $X = \{1, \dots, k\}$. Тогда вероятностная мера $P_c[D]$, $c \in C$, Выборка представляется совокупностью пар задается набором вероятностей

$$\alpha_j = \mathbb{P}(x = j), \quad p_j = \mathbb{P}(y = 0 \mid x = j).$$

$$V = (v_j \mid j = \overline{1, k}), \quad v_j = (m_j, n_j).$$

Решающая функция минимизирует эмпирический риск независимо в каждой точке $x \in X$: $f(x) = I(m_j < n_j)$.

Выражения для риска

$$\begin{aligned} \tilde{R}(V) &= \sum_{j=1}^k \tilde{r}(m_j, n_j), \\ \tilde{r}(m, n) &= \frac{1}{N} \tilde{\nu}(m, n), \quad \tilde{\nu}(m, n) = \min(m, n - m); \\ R(c, \lambda_{Q,V}) &= \sum_{j=1}^k r(m_j, n_j, \alpha_j, p_j), \\ r(m, n, \alpha, p) &= \alpha \nu(m, n, p), \\ \nu(m, n, p) &= \begin{cases} 1 - p, & m > n - m; \\ p, & m < n - m; \\ 0,5, & m = n - m. \end{cases} \end{aligned}$$

Средний квадрат уклонения

В общем случае оценочный функционал — это некоторая функция выборки.

Качество эмпирического функционала $\bar{R}(V, Q)$ как оценки риска обычно характеризуют средним квадратом уклонения, т.е.

$$\Delta = \mathbb{E} (\bar{R}(V, Q) - R(c, \lambda_{Q,V}))^2.$$

Существенная проблема заключается в том, что выражения зависят от c — распределения, которое неизвестно.

Кроме того, одно и то же отклонение при разных значениях риска имеет разную значимость.

Доверительный интервал для риска

Доверительный интервал для R зададим в виде $[0, \hat{R}(V)]$, где $\hat{R}(V)$ — оценочная функция или просто оценка (риска). При этом должно выполняться условие:

$$\forall c, P_c(R \leq \hat{R}(V)) \geq \eta,$$

где η — заданная доверительная вероятность.

На практике интервальную оценку будем строить как $\hat{R}(\bar{R}(V))$ — функцию точечной оценки.

Качество интервальной оценки будем характеризовать величиной $\mathbb{E}\hat{R}(V)$, которая зависит от c , ввиду чего выбор наилучшей оценки становится многокритериальной задачей.

Эмпирические доверительные интервалы для риска

Эмпирический доверительный интервал для R зададим в виде $[0, \hat{R}(\bar{R}(V))]$.

При этом должно выполняться условие:

$$\forall c \in \tilde{C}, P_c(R \leq \hat{R}(V)) \geq \eta,$$

где η — заданная доверительная вероятность, а \tilde{C} — эвристически выбранное множество распределений.

Лекция 8

Лекция 9

Разложения критерия качества решающих функций

[bias-variance decomposition](#)

Другие вопросы

Переобучение и недообучение

Переобучение (англ. overfitting) — негативное явление, возникающее, когда алгоритм обучения вырабатывает предсказания, которые слишком близко или точно соответствуют конкретному набору данных и поэтому не подходят для применения алгоритма к дополнительным данным или будущим наблюдениям.

Недообучение (англ. underfitting) — негативное явление, при котором алгоритм обучения не обеспечивает достаточно малой величины средней ошибки на обучающей выборке. Недообучение возникает при использовании недостаточно сложных моделей.

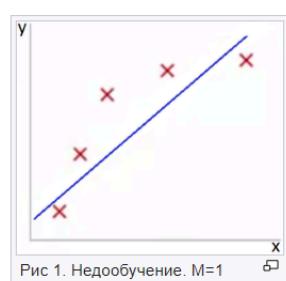


Рис 1. Недообучение. M=1

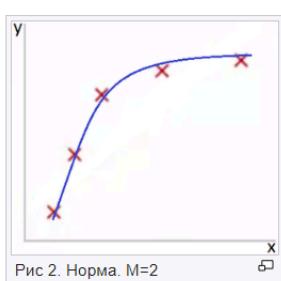


Рис 2. Норма. M=2

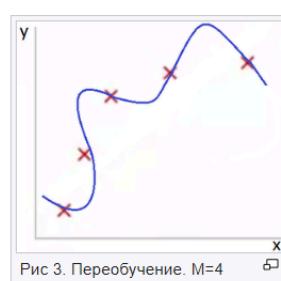


Рис 3. Переобучение. M=4

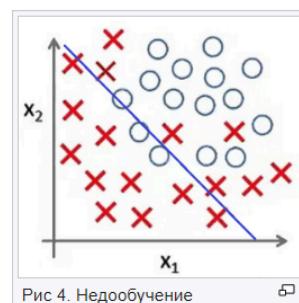


Рис 4. Недообучение

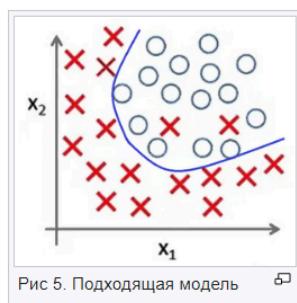


Рис 5. Подходящая модель

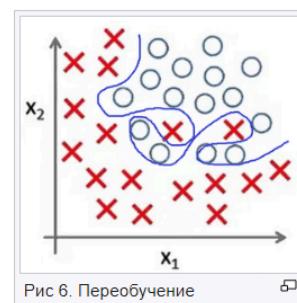
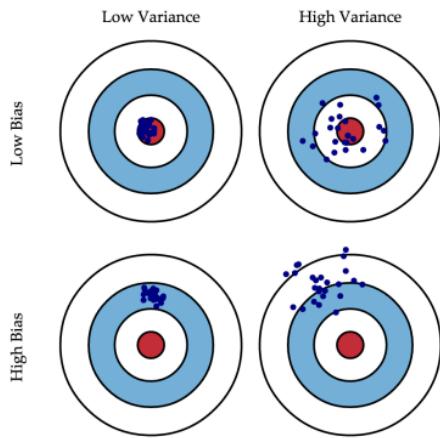


Рис 6. Переобучение

High variance и high bias



Bias — смещение, ошибка неверных предположений в алгоритме обучения. Высокий bias может привести к недообучению.

Variance — разброс, ошибка, вызванная большой чувствительностью к небольшим отклонениям в тренировочном наборе. Высокая дисперсия может привести к переобучению.

Для устранения **high variance** и **high bias** можно использовать смеси и ансамбли. Например, можно составить ансамбль (boosting) из нескольких моделей с высоким bias и получить модель с небольшим bias. В другом случае при bagging соединяются несколько моделей с низким bias, а результирующая модель позволяет уменьшить variance.

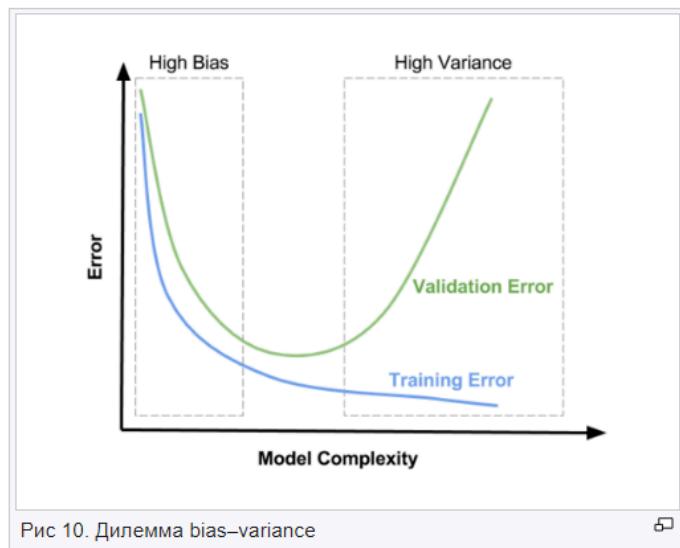


Рис 10. Дilemma bias–variance

Возможные решения при переобучении

- Увеличение количества данных в наборе;
- Уменьшение количества параметров модели;
- Добавление регуляризации / увеличение коэффициента регуляризации.

Возможные решения при недообучении

- Добавление новых параметров модели;
- Использование для описания модели функций с более высокой степенью;
- Уменьшение коэффициента регуляризации.

метрики, почему их должно быть много

1. Одна метрика может не отражать все аспекты модели. Например, точность (accuracy) может быть неинформативной в случае сильно несбалансированных классов. Дополнительные метрики, такие как precision, recall и F1-мера, могут дать более полное представление о производительности модели.
2. Разные метрики могут быть более или менее подходящими для различных типов данных. Например, для задач регрессии могут быть полезны метрики, такие как среднеквадратичная

- ошибка (MSE) и коэффициент детерминации (R-squared), в то время как для задач классификации - precision, recall и F1-мера.
3. Некоторые метрики могут быть более или менее чувствительными к определенным типам ошибок. Например, если ложноположительные ошибки более критичны, чем ложноотрицательные, то precision может быть более важной метрикой.
 4. Разные метрики могут фокусироваться на различных аспектах модели, таких как стабильность, интерпретируемость или способность обобщения. Например, ROC-AUC может быть полезен для оценки способности модели различать классы, в то время как метрики калибровки могут быть важными для оценки вероятностей, предсказанных моделью.

Когда можно считать частоту ошибки на обучающей выборке вероятностью ошибки
Коля - я думаю, когда классификатор хорошо откалиброван либо обучающая выборка равна тестовой.

1. **Репрезентативность выборки:** Если обучающая выборка является хорошо репрезентативной для всего распределения данных, то частота ошибок на этой выборке может аппроксимировать вероятность ошибки на новых данных. Это означает, что выборка должна адекватно отражать разнообразие и распределение данных в целевой популяции.
2. **Отсутствие переобучения:** Если модель не переобучена, то есть не настроена слишком точно под конкретные данные обучающей выборки, то ошибка на обучающей выборке может быть приближением к ошибке на новых данных. Переобученные модели обычно показывают низкую ошибку на обучающей выборке, но высокую на новых данных.
3. **Достаточный размер выборки:** Чем больше размер обучающей выборки, тем более точной будет оценка вероятности ошибки. Маленькие выборки могут привести к нерепрезентативной оценке ошибки из-за высокой вариативности.
4. **Сбалансированность данных:** Если в выборке неравномерно представлены классы (в случае задач классификации) или ключевые характеристики (в случае регрессии), то частота ошибок может быть искажена. Сбалансированная выборка помогает обеспечить, что оценка ошибки будет точной для всех групп.
5. **Стабильность модели:** Если модель обладает высокой стабильностью, то есть мало изменяет своё поведение в ответ на малые изменения в данных, то ошибка на обучающей выборке может быть более точной оценкой общей ошибки.

Когда откалиброваны вероятности (отдельная процедура от тренировки)

Random Forest

Bagging + Random Subspace Method + Решающие деревья

Bootstrap - создаем n подвыборок размером m из начальной выборки. Некоторые элементы могут повторяться (некоторые могут вовсе не быть), что говорит о весе элемента в данной выборке.

Bagging (Bootstrap aggregating) - использует bootstrap, обучаем решающее дерево на каждой из подвыборок, предсказываем на тех, что не попали в подвыборку (**предсказывает по своей подвыборке, нет?**, по-другим это **out-of-bag**, **bagging** и **out-of-bag** синтаксически похожи, это разные вещи **bagging** - метод ансамблевого обучения, **out-of-bag** - техника оценивания модели в ансамбле). уменьшает разброс

Про **out-of-bag**: тут все данные разбиваются на подвыборки, а предсказывают уже по другим подвыборкам, за счёт этого сразу при обучении можно получить такую метрику как ООВЕ (out-of-bag error)

Random Subspace Method - строим деревья для подвыборок, выбираем случайное кол-во фич для обучения классификатора. (снижает смещение)

В конце все предсказания усредняем. (или выбираем голосованием)

Out-of-bag

Out-of-Bag оценка использует тот факт, что примеры данных, которые не были включены в подмножество при построении дерева (называемые "out-of-bag" примерами), могут быть использованы для оценки производительности модели без необходимости отдельного тестового набора. Вот общий алгоритм:

1. Постройка ансамбля:

- Строится несколько (например, N) деревьев решений, каждое из которых обучается на случайной подвыборке данных.

2. Оценка Out-of-Bag:

- Для каждого примера данных i в обучающем наборе, проверяется, был ли он использован при построении дерева.
- Для каждого дерева, которое не использовало пример i при обучении (то есть был out-of-bag для этого дерева), пример i используется для оценки производительности дерева.
- Повторяется для всех деревьев, и усредненные оценки производительности используются для оценки общей производительности случайного леса.

3. Оценка общей производительности:

- Можно использовать усредненные оценки Out-of-Bag для оценки общей производительности случайного леса.

Этот метод позволяет эффективно использовать каждый пример данных для обучения и оценки, уменьшая потребность в дополнительном тестовом наборе данных. Оценки Out-of-Bag также могут быть полезны для настройки гиперпараметров модели.

Про 3 пункт: либо строим новую решающую функцию для всей выборки

Линейная регрессия

Линейная регрессия — это метод статистики и машинного обучения, который используется для анализа отношения между зависимой переменной y и одной или несколькими независимыми переменными x . Основная идея заключается в поиске линейной зависимости между переменными. Модель линейной регрессии представляет собой уравнение прямой:

где:

- y - зависимая переменная,
- x_1, x_2, \dots, x_n - независимые переменные,
- b_0 - коэффициент смещения (intercept),
- b_1, b_2, \dots, b_n - коэффициенты наклона (slope),
- ε - ошибка (резидуальная ошибка).

$$y = b_0 + b_1 \cdot x_1 + b_2 \cdot x_2 + \dots + b_n \cdot x_n + \varepsilon$$

Цель состоит в подборе коэффициентов b_0, b_1, \dots, b_n таким образом, чтобы минимизировать сумму квадратов ошибок (сумму квадратов разностей между фактическими и предсказанными значениями). Процесс обучения модели линейной регрессии обычно включает в себя использование метода наименьших квадратов (**OLS**), который минимизирует сумму квадратов остатков. Математически это выглядит следующим образом:

$$\sum_{i=1}^m (y_i - (\hat{b}_0 + \hat{b}_1 \cdot x_{i1} + \hat{b}_2 \cdot x_{i2} + \dots + \hat{b}_n \cdot x_{in}))^2 \rightarrow \text{минимум}$$

где $\hat{b}_0, \hat{b}_1, \dots, \hat{b}_n$ - оцененные коэффициенты после обучения модели.

Этот процесс можно реализовать с использованием различных методов, включая градиентный спуск. Градиентный спуск помогает находить оптимальные значения коэффициентов, минимизируя функцию потерь.

Градиентный спуск

Градиентный спуск — это оптимизационный алгоритм, используемый для нахождения минимума (или максимума) функции. Он основывается на идеи изменения параметров модели в направлении, противоположном градиенту функции, с тем чтобы постепенно приближаться к минимуму функции.

Градиент функции указывает на направление наибольшего роста функции, поэтому движение в противоположном направлении должно привести к уменьшению значения функции.

Для функции $J(\theta)$, где θ представляет собой вектор параметров модели, градиент

$\nabla J(\theta)$ представляет собой вектор частных производных функции J по каждому параметру. Градиентный спуск обновляет параметры модели по следующему правилу:

$$\theta = \theta - \alpha \cdot \nabla J(\theta)$$

где:

- θ — вектор параметров модели,
- α — шаг обновления (learning rate), который определяет размер шага при каждой итерации.

Градиентный спуск выполняется до тех пор, пока не будет достигнуто условие останова, такое как фиксированное количество итераций или достижение достаточно малого изменения функции потерь.

Если learning rate выбран слишком большим, алгоритм может осциллировать или даже расходиться. Слишком маленький learning rate может привести к медленной сходимости и задерживать процесс обучения.

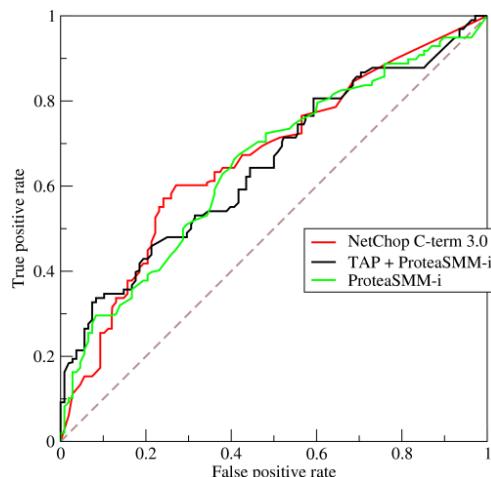
Кривые ROC-AUC

Ссылка

Есть 3 параметра - TPR, FPR, порог вероятности.

$$TPR = \frac{TP}{TP + FN} \quad FPR = \frac{FP}{FP + TN}$$

Наш классификатор дает вероятность, что объект принадлежит классу +1. Давайте будем перебирать эти вероятности в каком-нибудь порядке (по-убыванию) и считать, что если у объекта вероятность больше порога, то он принадлежит классу +1. Далее считаем для каждого порога TPR и FPR, и отмечаем точки на графике.



Precision-recall

- **Точность (precision)** — доля верно предсказанных объектов класса 1, среди всех объектов отмеченных первым классом

$$Precision = \frac{TP}{TP + FP}$$

- **Полнота (recall)** — доля верно предсказанных объектов класса 1, среди всех объектов класса 1

$$Recall = \frac{TP}{TP + FN}$$

Тоже самое, перебираем пороги. Но теперь precision на oY, recall на oX.

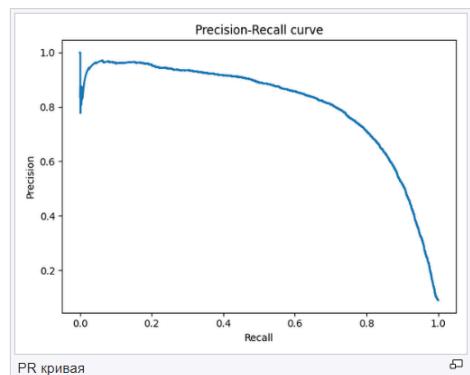
Precision-Recall кривая

Обоснование: Чувствительность к соотношению классов.

Рассмотрим задачу выделения математических статей из множества научных статей. Допустим, что всего имеется 1.000.100 статей, из которых лишь 100 относятся к математике. Если нам удастся построить алгоритм $a(x)$, идеально решающий задачу, то его TPR будет равен единице, а FPR — нулю. Рассмотрим теперь «плохой» алгоритм, дающий положительный ответ на 95 математических и 50.000 нематематических статьях. Такой алгоритм совершенно бесполезен, но при этом имеет TPR = 0.95 и FPR = 0.05, что крайне близко к показателям идеального алгоритма. Таким образом, если положительный класс существенно меньше по размеру, то AUC-ROC может давать неадекватную оценку качества работы алгоритма, поскольку измеряет долю неверно принятых объектов относительно общего числа отрицательных. Так, алгоритм $b(x)$, помещающий 100 релевантных документов на позиции с 50.001-й по 50.101-ю, будет иметь AUC-ROC 0.95.

Precision-recall (PR) кривая.

Избавиться от указанной проблемы с несбалансированными классами можно, перейдя от ROC-кривой к PR-кривой. Она определяется аналогично ROC-кривой, только по осмам откладываются не FPR и TPR, а полнота (по оси абсцисс) и точность (по оси ординат). Критерием качества семейства алгоритмов выступает площадь под PR-кривой (англ. Area Under the Curve — AUC-PR).



Градиентный бустинг и его отличие от AdaBoost

Алгоритм градиентного бустинга:

1. Инициализация:

- Инициализируются прогнозы модели. Обычно используется простая модель, такая как константное значение.

2. Для каждого дерева:

а. Вычисление остатков:

- Вычисляются остатки (разница между фактическими и предсказанными значениями).

б. Построение дерева:

- Строится дерево решений, которое предсказывает остатки.

в. Шаг градиентного спуска:

- Обученное дерево добавляется к текущей модели с учетом шага градиентного спуска, который оптимизирует прогнозы по остаткам.

3. Финальный прогноз:

- Полученные прогнозы ансамбля деревьев объединяются для получения финального прогноза.

Различия с AdaBoost:

1. Взвешивание ошибок:

- В AdaBoost каждый новый классификатор строится с учетом взвешенных ошибок предыдущих классификаторов. В градиентном бустинге используется градиент (остатки), чтобы скорректировать модель.

2. Функции потерь:

- AdaBoost минимизирует экспоненциальную функцию потерь, тогда как градиентный бустинг минимизирует произвольную функцию потерь, которая может быть выбрана в зависимости от задачи (например, среднеквадратичная ошибка для регрессии или логистическая функция потерь для классификации).

3. Последовательное построение:

- AdaBoost строит классификаторы последовательно, при этом каждый новый классификатор уделяет больше внимания тем примерам, которые были неправильно классифицированы предыдущими. В градиентном бустинге каждое дерево строится для улучшения текущей модели, сосредотачиваясь на остатках.

4. Learning rate:

- Градиентный бустинг включает параметр, называемый "learning rate" или шаг градиентного спуска, который управляет величиной коррекции прогнозов в каждом шаге.

K-fold Cross-validation

K-fold cross-validation - это метод оценки производительности модели машинного обучения, который помогает уменьшить влияние случайности в выборе обучающего и тестового наборов данных. В этом методе данные разбиваются на K частей (или подмножеств), называемых "фолдами". Модель обучается K раз, каждый раз используя K-1 фолдов в качестве обучающего набора данных, а оставшийся фолд используется для тестирования. Процесс повторяется K раз, каждый раз выбирая другой фолд для тестирования. Затем оценки производительности усредняются.

Вот базовый алгоритм K-fold cross-validation:

1. Разбиение данных:

- Разделите ваш набор данных на K равных частей, или фолдов.

2. Итерации:

- Для каждой итерации от 1 до K:
 - Выберите один из фолдов как тестовый набор данных.
 - Обучите модель на оставшихся K-1 фолдах (обучающий набор данных).
 - Оцените производительность модели на тестовом наборе данных.
 - Запишите результаты оценки производительности.

3. Усреднение результатов:

- Усредните оценки производительности для получения общей оценки.