Машинное обучение
Теоретический минимум

ЭФ МГУ

2020

1 | Матричное дифференцирование

В машинном обучении часто приходится работать с объектами большой размерности (векторами, матрицами, тензорами), находить для них производные различных порядков. Для двумерного пространства это сделать не тяжело (как мы сделали это выше), а для пространства высоких порядков (например $\mathbb{R}^{100}$) находить вручную элементы матрицы Гессиана весьма проблематично (придётся вычислить $\frac{100 \cdot 101}{2}$ значений). Мы познакомимся с техникой матричного дифференцирования, которая справляется с этой проблемой более элегантно.

Имеем: $x \in \mathbb{R}^a, \quad f : \mathbb{R}^a \to \mathbb{R}^b$

Назовём матричной производной функции f в точке $x \quad (Df(x)[\Delta x])$ второй член в разложении Тейлора:

$$\boxed{f(x + \Delta x) = f(x) + Df(x)[\Delta x] + o\left(\|\Delta x\|\right)} \qquad (*)$$

Заметим, что определение $(*)$ эквивалентно:

$$\boxed{Df(x)[\Delta x] = \lim_{t \to 0} \frac{f(x + \Delta x \cdot t) - f(x)}{t}}$$

1.1 Зачем это нужно?

Оказывается, что для многомерных функций верны соответстующие утверждения про точки экстремума, как и в скалярном случае:

Необходимое условие экстремума:

Если x^* - точка локального экстремума (минимума/максимума) $\quad \Rightarrow \quad Df(x^*)[\Delta x] = 0$

Достаточное условие экстремума:

Если x^* - точка локального минимума (максимума) $\quad \Rightarrow \quad D^2 f(x^*)[\Delta x, \Delta x] > 0 \quad (< 0)$

Но как пользоваться этим $Df(x)[\Delta x]$, ведь в нём участвует загадочное Δx? Оказывается, что $Df(x)[\Delta x]$ можно выразить (в нашем случае так будет почти всегда) через привычные нам объекты - градиенты, гессианы и т.д.

1. $f : \mathbb{R} \to \mathbb{R} \Rightarrow Df(x)[\Delta x] = \nabla f(x) \cdot \Delta x$
2. $f : \mathbb{R}^n \to \mathbb{R} \Rightarrow Df(x)[\Delta x] = \nabla f(x)^T \Delta x$
3. $f : \mathbb{R}^{m \times n} \to \mathbb{R} \Rightarrow Df(x)[\Delta x] = tr\left(\nabla f(x)^T \Delta x\right)$

Во всех случаях объекты $\nabla f(x)$ будут размерности аргумента. То есть в первом случае $\nabla f(x) \in \mathbb{R}$ (обычная скалярная производная), во втором: $\nabla f(x) \in \mathbb{R}^n$ (градиент), в третьем: $\nabla f(x) \in \mathbb{R}^{m \times n}$ (матрица из производных по ij элементу матрицы x).

Это уже те объекты с которыми мы привыкли работать, и тогда в случае векторного агрумента ($x \in \mathbb{R}^n$), условие $Df(x)[\Delta x] = 0$ превращается в условие: $\nabla f(x) = 0$ - привычное необходимое условие экстремума скалярной функции нескольких переменных.

2 | Градиентный спуск

Определение 1 (Градиентный спуск)**.** *метод нахождения локального экстремума (минимума или максимума) функции с помощью движения вдоль градиента.*

Для минимизации функции в направлении градиента используются методы одномерной оптимизации, например, метод золотого сечения. Также можно искать не наилучшую точку в направлении градиента, а какую-либо лучше текущей.

Наиболее простой в реализации из всех методов локальной оптимизации. Имеет довольно слабые условия сходимости, но при этом скорость сходимости достаточно мала (линейна). Шаг градиентного метода часто используется как часть других методов оптимизации, например, метод Флетчера — Ривса.

2.1 Gradient Descent

Основное свойство антиградиента — он указывает в сторону наискорейшего убывания функции в данной точке. Соответственно, будет логично стартовать из некоторой точки, сдвинуться в сторону антиградиента, пересчитать антиградиент и снова сдвинуться в его сторону и т.д. Запишем это более формально. Пусть $w^{(0)}$ — начальный набор параметров (например, нулевой или сгенерированный из некоторого случайного распределения). Тогда градиентный спуск состоит в повторении следующих шагов до сходимости:

$$w^{(k)} = w^{(k-1)} - \eta_k \nabla Q(w^{(k-1)}). \tag{1}$$

Здесь под $Q(w)$ понимается значение функционала ошибки для набора параметров w.

Через η_k обозначается длина шага, которая нужна для контроля скорости движения. Можно делать её константной: $\eta_k = c$. При этом если длина шага слишком большая, то есть риск постоянно «перепрыгивать» через точку минимума, а если шаг слишком маленький, то движение к минимуму может занять слишком много итераций. Иногда длину шага монотонно уменьшают по мере движения — например, по простой формуле

$$\eta_k = \frac{1}{k}.$$

2.2 Stochastic GD

Проблема метода градиентного спуска (1) состоит в том, что на каждом шаге необходимо вычислять градиент всей суммы (будем его называть полным градиентом):

$$\nabla_w Q(w) = \sum_{i=1}^{\ell} \nabla_w q_i(w).$$

Это может быть очень трудоёмко при больших размерах выборки. В то же время точное вычисление градиента может быть не так уж необходимо — как правило, мы делаем не очень большие шаги в сторону антиградиента, и наличие в нём неточностей не должно сильно сказаться на общей траектории. Опишем несколько способов оценивания полного градиента.

Оценить градиент суммы функций можно градиентом одного случайно взятого слагаемого:

$$\nabla_w Q(w) \approx \nabla_w q_{i_k}(w),$$

где i_k — случайно выбранный номер слагаемого из функционала. В этом случае мы получим метод **стохастического градиентного спуска** (stochastic gradient descent, SGD):

$$w^{(k)} = w^{(k-1)} - \eta_k \nabla q_{i_k}(w^{(k-1)}).$$

Для выпуклого и гладкого функционала может быть получена следующая оценка:

$$\mathbb{E}\left[Q(w^{(k)}) - Q(w^*)\right] = O(1/\sqrt{k}).$$

Таким образом, метод стохастического градиента имеет менее трудоемкие итерации по сравнению с полным градиентом, но и скорость сходимости у него существенно меньше.

Отметим одно важное преимущество метода стохастического градиентного спуска. Для выполнения одного шага в данном методе требуется вычислить градиент лишь одного слагаемого — а поскольку одно слагаемое соответствует ошибке на одном объекте, то получается, что на каждом шаге необходимо держать в памяти всего один объект из выборки. Данное наблюдение позволяет обучать линейные модели на очень больших выборках: можно считывать объекты с диска по одному, и по каждому делать один шаг метода SGD.

2.3 Метод Ньютона

Идея метода: строится квадратичная аппроксимация (по методу Тейлора) данной функции в окрестности текущей точки x_{old}. Новая точка x_{new} является точкой минимума этой квадратичной аппроксимации.

$f(x_{old} + \epsilon) \approx f(x_{old}) + f'(x_{old}) \cdot \epsilon + \frac{1}{2} \cdot f''(x_{old}) \cdot \epsilon^2 \to \min_{\epsilon}$

$f'_\epsilon(x_{old} + \epsilon) = f'(x_{old}) + f''(x_{old}) \cdot \epsilon = 0 \quad \Rightarrow \quad \boxed{\epsilon^* = -\frac{f'(x_{old})}{f''(x_{old})}}$

Итак: $x_{new} = x_{old} + \epsilon \quad \Rightarrow \quad \boxed{x_{new} = x_{old} - \frac{f'(x_{old})}{f''(x_{old})}}$

В многомерном случае формула принимает вид: $x_{new} = x_{old} - \nabla^2 f(x_{old})^{-1} \cdot \nabla f(x_{old})$,
$\nabla^2 f(x_{old})^{-1}$ - обратная к матрице Гессиан

3 | Настройка гиперпараметров с помощью кросс-валидации

Переобучение Мы выработали достаточно общий метод обучения линейных регрессионных моделей, основанный на градиентных методах оптимизации. При этом модель может оказаться *переобученной* — её качество на новых данных может быть существенно хуже качества на обучающей выборке. Действительно, при обучении мы требуем от модели лишь хорошего качества на обучающей выборке, и совершенно не очевидно, почему она должна при этом хорошо *обобщать* эти результаты на новые объекты.

Способы борьбы

- Увеличение обучающей выборки
- Регуляризация L_1, L_2
- Прореживание (Dropout)
- Кросс-валидация

Определение 2 (Cross-validation)**.** *метод оценки аналитической модели и её поведения на независимых данных.*

При оценке модели имеющиеся в наличии данные разбиваются на k частей. Затем на k1 частях данных производится обучение модели, а оставшаяся часть данных используется для тестирования. Процедура повторяется k раз; в итоге каждая из k частей данных используется для тестирования. После этого качество каждой модели оценивается по тому блоку, который не участвовал в её обучении, и результаты усредняются. Получается оценка эффективности выбранной модели с наиболее равномерным использованием имеющихся данных.

$$\mathrm{CV} = \frac{1}{k} \sum_{i=1}^{k} Q\left(a_i(x), X_i\right).$$

Обычно кросс-валидация используется в ситуациях, где целью является предсказание, и хотелось бы оценить, насколько предсказывающая модель способна работать на практике. Один цикл кросс-валидации включает разбиение набора данных на части, затем построение модели на одной части (называемой *тренировочным набором*), и валидация модели на другой части (называемой *тестовым набором*). Чтобы уменьшить разброс результатов, разные циклы кросс-валидации проводятся на разных разбиениях, а результаты валидации усредняются по всем циклам.

3.1 Гиперпараметры

В машинном обучении принято разделять подлежащие настройке величины на *параметры* и *гиперпараметры*. Параметрами называют величины, которые настраиваются по обучающей выборке — например, веса в линейной регрессии. К гиперпараметрам относят величины, которые контролируют сам процесс обучения и не могут быть подобраны по обучающей выборке.

Хорошим примером гиперпараметра является коэффициент регуляризации α. Введение регуляризации мешает модели подгоняться под обучающие данные, и с точки зрения среднеквадратичной ошибки выгодно всегда брать $\alpha = 0$. Разумеется, такой выбор не будет оптимальным с точки зрения качества на новых данных, и поэтому коэффициент регуляризации (как и другие гиперпараметры) следует настраивать по отложенной выборке или с помощью кросс-валидации.

При подборе гиперпараметров по кросс-валидации возникает проблема: мы используем отложенные данные, чтобы выбрать лучший набор гиперпараметров. По сути, отложенная выборка тоже становится обучающей, и показатели качества на ней перестают характеризовать обобщающую способность модели. В таких случаях выборку, на которой настраиваются гиперпараметры, называют валидационной, и при этом выделяют третий, тестовый набор данных, на которых оценивается качество итоговой модели.

3.2 Распространенные типы кросс-валидации

3.2.1 Кросс-валидация по K блокам (K-fold cross-validation)

В этом случае исходый набор данных разбивается на K одинаковых по размеру блока. Из K блоков один оставляется для тестирования модели, а остающиеся K-1 блока используются как тренировочный набор. Процесс повторяется K раз, и каждый из блоков используется один раз как тестовый набор. Получаются K результатов, по одному на каждый блок, они усредняются или комбинируются каким-либо другим способом, и дают одну оценку. Преимущество такого способа перед случайным сэмплированием (random subsampling) в том, что все наблюдения используются и для тренировки, и для тестирования модели, и каждое наблюдение используется для тестирования в точности один раз. Часто используется кросс-валидация на 10 блоках, но каких-то определенных рекомендаций по выбору числа блоков нет.

В послойной кросс-валидации блоки выбираются таким образом, что среднее значение ответа модели примерно равно по всем блокам.

3.2.2 Валидация случайным сэмплированием (random subsampling)

Этот метод случайным образом разбивает набор данных на тренировочный и тестовый наборы. Для каждого такого разбиения, модель подгоняется под тренировочные данные, а точность предсказания оценивается на тестовом наборе. Результаты затем усредняются по всем разбиениям. Преимущество такого метода перед кросс-валидацией на K блоках в том, что пропорции тренировочного и тестового наборов не зависят от числа повторений (блоков). Недостаток метода в том, что некоторые наблюдения могут ни разу не попасть в тестовый набор, тогда как другие могут попасть в него более, чем один раз. Другими словами, тестовые наборы могут перекрываться. Кроме того, поскольку разбиения проводятся случайно, результаты будут отличаться в случае повторного анализа.

В послойном варианте этого метода, случайные выборки генерируются таким способом, при котором средний ответ модели равен по тренировочному и тестовому наборам. Это особенно полезно, когда ответ модели бинарен, с неравными пропорциями ответов по данным.

3.2.3 Поэлементная кросс-валидация (Leave-one-out, LOO)

Здесь отдельное наблюдение используется в качестве тестового набора данных, а остальные наблюдения из исходного набора – в качестве тренировочного. Цикл повторяется, пока каждое наблюдение не будет использовано один раз в качестве тестового. Это то же самое, что и K-блочная кросс-валидация, где K равно числу наблюдений в исходном наборе данных.

3.3 Применения кросс-валидации

Кросс-валидация может использоваться для сравнения результатов различных процедур предсказывающего моделирования. Например, предположим, что мы интересуемся оптическим распознаванием символов, и рассматриваем варианты использования либо поддерживающих векторов (Support Vector Machines, SVM), либо k ближайших соседей (k nearest neighbors, KNN). С помощью кросс-валидации мы могли бы объективно сравнить эти два метода в терминах относительных коэффициентов их ошибок классификаций. Если мы будем просто сравнивать эти методы по их ошибкам на тренировочной выборке, KNN скорее всего покажет себя лучше, поскольку он более гибок и следовательно более склонен к переподгонке по сравнению с SVM.

Кросс-валидация также может использоваться для выбора параметров. Предположим, у нас есть 20 параметров, которые мы могли бы использовать в модели. Задача – выбрать параметры, использование которых даст модель с лучшими предсказывающими способностями. Если мы будем сравнивать подмножества параметров по их ошибкам на тестовом наборе, лучшие результаты получатся при использовании всех параметров. Однако с кросс-валидацией, модель с лучшей способностью к обобщению обычно включает только некоторое подмножество параметров, которые достаточно информативны.

3.4 Ограничения и неверное использование кросс-валидации

Кросс-валидация дает значимые результаты только когда тренировочный набор данных и тестовый набор данных берутся из одного источника, из одной популяции. В многих применениях предсказательных моделей структура изучаемой системы меняется со временем. Это может наводить систематические отклонения тренировочного и валидационного наборов данных. К примеру, если модель для предсказания цены акции тренируется на данных из определенного пятилетнего периода, нереалистично рассматривать последующий пятилетний период как выборку из той же самой популяции.

Если выполняется правильно, и наборы данных из одной популяции, кросс-валидация дает результат практически без смещений (bias). Однако, есть много способов использовать кросс-валидацию неправильно. В этом случае ошибка предсказания на реальном валидационном наборе данных скорее всего будет намного хуже, чем ожидается по результатам кросс-валидации.

4 | Метрические методы классификации и регрессии. Проклятие размерности

4.1 Метрические методы классификации и регрессии.

В основе метрических методов лежат гипотеза компактности (классификация) и гипотеза непрерывности (регрессия). Гипотеза компактности - это предположение (допущение) о том, что похожие объекты лежат в одном классе. Гипотеза непрерывности - это предположение о том, что близким объектам соответствуют близкие ответы.

4.1.1 k-NN

Особенности

\+ Интерпретируемость решений

\+ Простота реализации.

- Нужно хранить всю выборку (lazy learning)

- На практике сложно выбрать хорошую метрику.

4.1.2 Parzen window

В основе подхода лежит идея о том, что плотность выше в тех точках, рядом с которыми находится большое количество объектов выборки.

Если мощность множества элементарных исходов много меньше размера выборки, то в качестве восстановленной по выборке плотности мы вполне можем взять и гистограмму значений выборки.

В противном случае (например, непрерывном) данный подход не применим, так как плотность концентрируется вблизи обучающих объектов, и функция распределения претерпевает резкие скачки. Приходится использовать восстановление методом Парзена-Розенблатта.

Парзеновская оценка плотности имеет вид:

$$p_{y,h}(x) = \frac{1}{l_y V(h)} \sum_{i=1}^{l} [y_i = y] K(\frac{\rho(x, x_i)}{h})$$

Соответствующее решающее правило, полученное после преобразований:

$$a(x; X^l, h) = arg \max_{y \in Y} \lambda_y \sum_{i=1}^{l} [y_i = y] K(\frac{\rho(x, x_i)}{h})$$

$K(z)$ — произвольная четная функция, называемая функцией ядра или окна. Термин окно происходит из классического вида функции:

$$K(z) = \frac{1}{2}(|z| < 1)$$

Восстановленная плотность имеет такую же степень гладкости, как и функция ядра. Поэтому на практике обычно используются все же более гладкие функции.

Вид функции окна не влияет на качество классификации определяющим образом.

Ширина окна Ширина окна сильно влияет на качество восстановления плотности и, как следствие, классификации. При слишком малом окне мы получаем тот же эффект, что и при использовании гистограммы значений. При слишком большом окне плотность вырождается в константу.

Для нахождения оптимальной ширины окна удобно использовать принцип максимума правдоподобия с исключением объектов по одному (leave-one-out, LOO).

4.1.3 Метод потенциальных функций

4.2 Проклятие размерности

Если используемая метрика $\rho(x,x)$ основана на суммировании различий по всем признакам, а число признаков очень велико, то все точки выборки могут оказаться практически одинаково далеки друг от друга. Тогда парзеновские оценки плотности становятся неадекватны. Это явление называют *проклятием размерности*. Выход заключается в понижении размерности с помощью преобразования пространства признаков, либо путём отбора информативных признаков. Можно строить несколько альтернативных метрик в подпространствах меньшей размерности, и полученные по ним алгоритмы классификации объединять в композицию.

5 | Формула классификации и регрессии в модели KNN. Примеры весов. Примеры метрик.

Определение 3 (k-Nearest-Neighbors)**.** *метрический алгоритм для автоматической классификации объектов или регрессии.*

Пусть дана обучающая выборка $X = (x_i, y_i)_{i=1}^l \subset \mathbb{X}$ и ф-ия расстояния $\rho : \mathbb{X} \times \mathbb{X} \to [0, \infty]$. Расположим объекты обучающей выборки X в порядке возрастания расстояний до u:

$$\rho(u, x_u^{(1)}) \leqslant \rho(u, x_u^{(2)}) \leqslant ... \leqslant \rho(u, x_u^{(l)}),$$

где через x_u^i обозначается i-й сосед объекта u. Алгоритм kNN относит объект к тому классу, представителей которого окажется больше среди всех k его ближайших соседей:

$$a(u; X^l, k) = \arg\max_{y \in Y} \sum_{i=1}^{k} w_i [y_u^{(i)} = y]$$

Параметр k обычно настраивается с помощью кросс-валидации.

В классическом методе k ближайших соседей все объекты имеют единичные веса: $w_i = 1$. Такой подход не является оптимальным.

Пример 1. Допустим, что $k = 3$, $\rho(u, x_u^{(1)}) = 1$, $\rho(u, x_u^{(2)}) = 2$, $\rho(u, x_u^{(3)}) = 100$. Очевидно, третий сосед находится слишком далеко и не должен оказывать сильного влияния на результат. Для реализации этой идеи используются веса, обратно пропорциональные расстоянию:

$$w_i = K(\rho(u, x_u^{(i)})),$$

где K-любая монотонно убывающая ф-ия.

С помощью метода kNN можно решать и задачи регрессии. Для этого нужно усреднить значения целевой ф-ии на соседях с весами:

$$a(u; X^l, k) = \frac{\sum_{i=1}^{k} w_{(i)} y_{(i)}}{\sum_{i=1}^{k} w_{(i)}}$$

Примеры весов

•

Примеры метрик

1. *Метрика Минковского*
2. *Расстояние Махалонобиса*
3. *Косинусная мера*
4. *Расстояние Джаккарда*
5. *Редакторское расстояние*

6 | Линейные методы классификации и регрессии.

Мы начнём с задачи бинарной классификации, а многоклассовый случай обсудим позже. Пусть $\mathbb{X} = \mathbb{R}^d$ — пространство объектов, $Y = \{-1, +1\}$ — множество допустимых ответов, $X = \{(x_i, y_i)\}_{i=1}^{\ell}$ — обучающая выборка. Иногда мы будем класс «+1» называть положительным, а класс «−1» — отрицательным.

Линейная модель классификации определяется следующим образом:

$$a(x) = \text{sign}\left(\langle w, x \rangle + w_0\right) = \text{sign}\left(\sum_{j=1}^{d} w_j x_j + w_0\right),$$

где $w \in \mathbb{R}^d$ — вектор весов, $w_0 \in \mathbb{R}$ — сдвиг (bias).

6.1 Линейный классификатор

Пусть $X \subset \mathbb{R}^d$ — пространство объектов, $Y = -1, +1$ — множество допустимых ответов, $X^l = (x_i, y_i)_{i=1}^l$ — обучающая выборка. Каждый объект $x \in X$ описывается вещественным вектором $(x_1, ..., x_d) \in \mathbb{R}^d$.

Линейный классификатор определяется следующим образом:

$$a(x,w) = \text{sign}(\langle w,x\rangle + b) = \text{sign}\left(\sum_{j=1}^{d} w_j x_j + b\right),$$

где $w \in \mathbb{R}^d$ — вектор весов, $b \in \mathbb{R}$ — сдвиг (bias).

Если не сказано иное, считается, что среди признаков есть константа $x_0 = 1$, тогда необходимость вводить сдвиг b отпадает, и линейный классификатор можно задавать как

$$a(x,w) = \text{sign}\langle x,w\rangle.$$

Обучение линейного классфификатора заключается в поиске вектора весов, на котором достигается минимум некоторого функционала качества.

$$w = \arg\min_{w \in \mathbb{R}^d} Q(w,X^l)$$

Наиболее логичным функционалом для задачи классификации является число неверно классифицированных объектов.

$$Q(w,X^l) = \sum_{i=1}^{l} [y_i(\langle w,x_i\rangle + b) < 0] \to \min_w$$

У такого функционала есть большой недостаток — он не является диффиренцируемым, из-за чего поиск оптимального вектора весов w становится крайне трудной задачей. Для преодоления этой проблемы оптимизируется гладкая верхняя оценка на данный функционал.

$$Q(w,X^l) = \sum_{i=1}^{l} [y_i(\langle w,x_i\rangle + b) < 0] \leqslant \sum_{i=1}^{l} L\left[y_i(\langle w,x_i\rangle + b)\right] \to \min_w$$

В качестве оценки $L(M)$ можно использовать, например, логистическую функцию потерь $L(M) = \log(1 + e^{-M})$

6.2 Margin (отступ)

Определение 4. *Отступом (margin) объекта $x_i \in \mathbb{X}^l$ относительно алгоритма классификации, имеющего вид $a(u) = \arg\max_{y \in Y} \Gamma_y(u)$, называется величина*

$$M(x_i) = \Gamma_{y_i}(x_i) - \max_{y \in Y} \Gamma_y(x_i)$$

Отступ показывает степень типичности объекта. Отступ отрицателен тогда и только тогда, когда алгоритм допускает ошибку на данном объекте. В зависимости от значений отступа обучающие объекты условно делятся на пять типов, в порядке убывания отступа: эталонные, неинформативные, пограничные, ошибочные, шумовые.

Верхние оценки. Функционал оценивает ошибку алгоритма на объекте x с помощью пороговой функции потерь $L(M) = [M < 0]$, где аргументом функции является отступ $M = y\langle w, x\rangle$. Оценим эту функцию сверху во всех точках M кроме, может быть, небольшой полуокрестности левее нуля:

$$L(M) \leqslant \tilde{L}(M).$$

После этого можно получить верхнюю оценку на функционал (**??**):

$$Q(a, X) \leqslant \frac{1}{\ell} \sum_{i=1}^{\ell} \tilde{L}(y_i \langle w, x_i \rangle) \to \min_w$$

Если верхняя оценка $\tilde{L}(M)$ является гладкой, то и данная верхняя оценка будет гладкой. В этом случае её можно будет минимизировать с помощью, например, градиентного спуска. Если верхнюю оценку удастся приблизить к нулю, то и доля неправильных ответов тоже будет близка к нулю.

Приведём несколько примеров верхних оценок:

1. $\tilde{L}(M) = \log\left(1 + e^{-M}\right)$ — логистическая функция потерь (**logloss**)
2. $\tilde{L}(M) = (1 - M)_+ = \max(0, 1 - M)$ — кусочно-линейная функция потерь (используется в методе опорных векторов) (**hinge loss**)
3. $\tilde{L}(M) = (-M)_+ = \max(0, -M)$ — кусочно-линейная функция потерь (соответствует персептрону Розенблатта)
4. $\tilde{L}(M) = e^{-M}$ — экспоненциальная функция потерь
5. $\tilde{L}(M) = 2/(1 + e^{M})$ — сигмоидная функция потерь

Любая из них подойдёт для обучения линейного классификатора.

7 | Подходы one-vs-one и one-vs-rest для линейных классификаторов.

Пусть рассматривается задача классификации с C классами.

7.1 One-vs-all

1. Обучим на всех объектах тренировочной выборки C бинарных классификаторов типа "принадлежит ли объект i-ому классу".
2. В результате получим C классификаторов $f_c(x)$.
3. Ответ ищется, как $\hat{y}(x) = \underset{c \in [1,\ldots,C]}{\operatorname{argmax}} f_c(x)$.

7.2 One-vs-one

1. Обучим $\frac{C(C-1)}{2}$ бинарных классификаторов следующим образом: для каждой пары $i, j \in [1, \ldots, C], i \neq j$ обучим бинарный классификатор $f_{i,j}(x)$ на тех объектах тренировочной выборки, ответ которых принимает значение i или j .
2. В результате получим $\frac{C(C-1)}{2}$ классификаторов $f_{i,j}(x)$.
3. На этапе предсказания каждый из всех обученных классификаторов возвращает индикатор принадлежности соответствующему классу. Тот класс, за который проголосовало большинство и будет ответом на данном объекте:

$$\hat{y}(x) = \underset{c \in [1,\ldots,C]}{\operatorname{argmax}} \sum_{i \neq j \in [1,\ldots,C]} \mathbb{I}[f_{i,j}(x) == c] \ .$$

8 | Метрики качества

Чтобы обучать регрессионные модели, нужно определиться, как именно измеряется качество предсказаний. Будем обозначать через y значение целевой переменной, через a — прогноз модели. Рассмотрим несколько способов оценить отклонение $L(y, a)$ прогноза от истинного ответа.

MSE и R^2. Основной способ измерить отклонение — посчитать квадрат разности:

$$L(y, a) = (a - y)^2$$

Благодаря своей дифференцируемости эта функция наиболее часто используется в задачах регрессии. Основанный на ней функционал называется среднеквадратичным отклонением (mean squared error, MSE):

$$\text{MSE}(a, X) = \frac{1}{\ell} \sum_{i=1}^{\ell} (a(x_i) - y_i)^2 \, .$$

Отметим, что величина среднеквадратичного отклонения плохо интерпретируется, поскольку не сохраняет единицы измерения — так, если мы предсказываем цену в рублях, то MSE будет измеряться в квадратах рублей. Чтобы избежать этого, используют корень из среднеквадратичной ошибки (root mean squared error, RMSE):

$$\text{RMSE}(a, X) = \sqrt{\frac{1}{\ell} \sum_{i=1}^{\ell} (a(x_i) - y_i)^2}.$$

Среднеквадратичная ошибка подходит для сравнения двух моделей или для контроля качества во время обучения, но не позволяет сделать выводы о том, насколько хорошо данная модель решает задачу. Например, $\text{MSE} = 10$ является очень плохим показателем, если целевая переменная принимает значения от 0 до 1, и очень хорошим, если целевая переменная лежит в интервале $(10000, 100000)$. В таких ситуациях вместо среднеквадратичной ошибки полезно использовать *коэффициент детерминации* (или коэффициент R^2):

$$R^2(a, X) = 1 - \frac{\sum_{i=1}^{\ell} (a(x_i) - y_i)^2}{\sum_{i=1}^{\ell} (y_i - \bar{y})^2},$$

где $\bar{y} = \frac{1}{\ell}\sum_{i=1}^{\ell} y_i$ — среднее значение целевой переменной. Коэффициент детерминации измеряет долю дисперсии, объяснённую моделью, в общей дисперсии целевой переменной. Фактически, данная мера качества — это нормированная среднеквадратичная ошибка. Если она близка к единице, то модель хорошо объясняет данные, если же она близка к нулю, то прогнозы сопоставимы по качеству с константным предсказанием.

MAE. Заменим квадрат отклонения на модуль:

$$L(y, a) = |a - y|$$

Соответствующий функционал называется средним абсолютным отклонением (mean absolute error, MAE):

$$\text{MAE}(a, X) = \frac{1}{\ell}\sum_{i=1}^{\ell} |a(x_i) - y_i| \,.$$

Модуль отклонения не является дифференцируемым, но при этом менее чувствителен к выбросам. Квадрат отклонения, по сути, делает особый акцент на объектах с сильной ошибкой, и метод обучения будет в первую очередь стараться уменьшить отклонения на таких объектах. Если же эти объекты являются выбросами (то есть значение целевой переменной на них либо ошибочно, либо относится к другому распределению и должно быть проигнорировано), то такая расстановка акцентов приведёт к плохому качеству модели. Модуль отклонения в этом смысле гораздо более терпим к сильным ошибкам.

Приведём ещё одно объяснение того, почему модуль отклонения устойчив к выбросам, на простом примере. Допустим, все ℓ объектов выборки имеют одинаковые признаковые описания, но разные значения целевой переменной $y_1, \ldots, y_\ell$. В этом случае модель должна на всех этих объектах выдать один и тот же ответ. Если мы выбрали MSE в качестве функционала ошибки, то получаем следующую задачу:

$$\frac{1}{\ell}\sum_{i=1}^{\ell} (a - y_i)^2 \to \min_a$$

Легко показать, что минимум достигается на среднем значении всех ответов:

$$a^*_{\text{MSE}} = \frac{1}{\ell}\sum_{i=1}^{\ell} y_i.$$

Если один из ответов на порядки отличается от всех остальных (то есть является выбросом), то среднее будет существенно отклоняться в его сторону.

Рассмотрим теперь ту же ситуацию, но с функционалом MAE:

$$\frac{1}{\ell}\sum_{i=1}^{\ell} |a - y_i| \to \min_a$$

Теперь решением будет медиана ответов:

$$a^*_{\text{MAE}} = \text{median}\{y_i\}_{i=1}^{\ell}.$$

Небольшое количество выбросов никак не повлияет на медиану — она существенно более устойчива к величинам, выбивающимся из общего распределения.

MSLE. Перейдём теперь к логарифмам ответов и прогнозов:

$$L(y, a) = (\log(a + 1) - \log(y + 1))^2$$

Соответствующий функционал называется среднеквадратичной логарифмической ошибкой (mean squared logarithmic error, MSLE). Данная метрика подходит для задач с неотрицательной целевой переменной. За счёт логарифмирования ответов и прогнозов мы скорее штрафуем за отклонения в порядке величин, чем за отклонения в их значениях. Также следует помнить, что логарифм не является симметричной функцией, и поэтому данная функция потерь штрафует заниженные прогнозы сильнее, чем завышенные.

MAPE и SMAPE. В задачах прогнозирования обычно измеряется относительная ошибка:

$$L(y, a) = \left| \frac{y - a}{y} \right|$$

Соответствующий функционал называется средней абсолютной процентной ошибкой (mean absolute percentage error, MAPE). Данный функционал часто используется в задачах прогнозирования. Также используется его симметричная модификация (symmetric mean absolute percentage error, SMAPE):

$$L(y, a) = \frac{|y - a|}{(|y| + |a|)/2}$$

Lift На практике часто возникают задачи, связанные с выбором подмножества: выделение лояльных клиентов банка, обнаружение уходящих пользователей мобильного оператора и т.д. Заказчика может интересовать вопрос, насколько выгоднее работать с этим подмножеством по сравнению со всем множеством. Если при рассылке предложений о кредите клиентам из подмножества и всем клиентам будет получаться одна и та же доля откликнувшихся, то подмножество не будет представлять особой ценности. Формально это измеряется с помощью *прироста концентрации* (lift), который равен отношению точности к доле положительных объектов в выборке:

$$\text{lift} = \frac{\text{precision}}{(\text{TP} + \text{FN})/\ell}.$$

Эту величину можно интерпретировать как улучшение доли положительных объектов в данном подмножестве относительно доли в случайно выбранном подмножестве такого же размера.

Индекс Джини В задачах кредитного скоринга вместо AUC-ROC часто используется пропорциональная метрика, называемая индексом Джини (Gini index):

$$\text{Gini} = 2\text{AUC} - 1.$$

По сути это умноженная на два площадь между ROC-кривой и диагональю, соединяющей точки $(0, 0)$ и $(1, 1)$.

Отметим, что переход от AUC к индексу Джини приводит к увеличению относительных разниц. Если мы смогли улучшить AUC с 0.8 до 0.9, то это соответствует относительному улучшению в 12.5%. В то же время соответствующие индексы Джини были улучшены с 0.6 до 0.8, то есть на 33.3% — относительное улучшение повысилось почти в три раза!

8.1 Метрики качества многоклассовой классификации

В многоклассовых задачах, как правило, стараются свести подсчет качества к вычислению одной из рассмотренных выше двухклассовых метрик. Выделяют два подхода к такому сведению: микро- и макро-усреднение.

Пусть выборка состоит из K классов. Рассмотрим K двухклассовых задач, каждая из которых заключается в отделении своего класса от остальных, то есть целевые значения для k-й задаче вычисляются как $y_i^k = [y_i = k]$. Для каждой из них можно вычислить различные характеристики (TP, FP, и т.д.) алгоритма $a^k(x) = [a(x) = k]$; будем обозначать эти величины как $\mathrm{TP}_k, \mathrm{FP}_k, \mathrm{FN}_k, \mathrm{TN}_k$. Заметим, что в двухклассовом случае все метрики качества, которые мы изучали, выражались через эти элементы матрицы ошибок.

При микро-усреднении сначала эти характеристики усредняются по всем классам, а затем вычисляется итоговая двухклассовая метрика — например, точность, полнота или F-мера. Например, точность будет вычисляться по формуле

$$\mathrm{precision}(a, X) = \frac{\overline{\mathrm{TP}}}{\overline{\mathrm{TP}} + \overline{\mathrm{FP}}},$$

где, например, $\overline{\mathrm{TP}}$ вычисляется по формуле

$$\overline{\mathrm{TP}} = \frac{1}{K}\sum_{k=1}^{K} \mathrm{TP}_k.$$

При макро-усреднении сначала вычисляется итоговая метрика для каждого класса, а затем результаты усредняются по всем классам. Например, точность будет вычислена как

$$\mathrm{precision}(a, X) = \frac{1}{K}\sum_{k=1}^{K} \mathrm{precision}_k(a, X); \qquad \mathrm{precision}_k(a, X) = \frac{\mathrm{TP}_k}{\mathrm{TP}_k + \mathrm{FP}_k}.$$

Если какой-то класс имеет очень маленькую мощность, то при микро-усреднении он практически никак не будет влиять на результат, поскольку его вклад в средние TP, FP, FN и TN будет незначителен. В случае же с макро-вариантом усреднение проводится для величин, которые уже не чувствительны к соотношению размеров классов (если мы используем, например, точность или полноту), и поэтому каждый класс внесет равный вклад в итоговую метрику.

8.2 Confusion matrix

Это способ разбить объекты на четыре категории в зависимости от комбинации истинного ответа и ответа алгоритма. Через элементы этой матрицы можно, например, выразить долю правильных ответов:

$$accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

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

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

Точность *(precision)* показывает, какая доля объектов, выделенных классификатором как положительные, действительно является таковыми. Полнота *(recall)* показывает, какая часть положительных ответов была выделена классификатором.

Существует несколько способов получить один критерий качества на основе точности и полноты. Один из них — F-мера, гармоническое среднее точности и полноты:

$$F = \frac{2 \times precision \times recall}{precision + recall}$$

Среднее гармоническое обладает важным свойством — оно близко к нулю, если хотя бы один из аргументов близок к нулю. Именно поэтому оно является более предпочтительным, чем среднее арифметическое.

9 | ROC-кривая, AUC-ROC.

9.1 ROC-кривая

Определение 5 (ROC-curve). *графичекая характеристика качества бинарного классификатора, зависимость доли верных положительных классификаций от доли ложных положительных классификаций при варьировании порога решающего правила.*

Преимуществом ROC-кривой является её инвариантность относительно отношения цены ошибки I и II рода.

Построение

1. **Вход**

   обучающая выборка X^l; $f(x) = \langle w,x \rangle$ — дискриминантная функция;

2. **Выход**

   $\{(FPR_i, TPR_i)\}_{i=0}^{l}$ — последовательность точек ROC-кривой; AUC — площадь под ROC-кривой.

3. **Алгоритм:**

   1: $l^- = \sum_{i=0}^{l}[y_i = -1]$ — число объектов класса -1;
   $l^+ = \sum_{i=0}^{l}[y_i = +1]$ — число объектов класса $+1$;

   2: упорядочить выборку X^l по убыванию значений $f(x_i)$;

   3: поставить первую точку в начало координат:
   $(FPR_0, TPR_0) := (0,0)$; $AUC := 0$;

   4: Для i := 1, ... , l

   4a: Если $y_i = -1$, то
   Сместиться на один шаг вправо:
   $FPR_i := FPR_{i-1} + \frac{1}{l^-}$; $TPR_i := TPR_{i-1}$;
   $AUC := AUC + \frac{1}{l^-} \times TPR_i$;

   4b: иначе
   Сместиться на один шаг вверх:
   $FPR_i := FPR_{i-1}$; $TPR_i := TPR_{i-1} + \frac{1}{l^+}$;

9.2 AUC-ROC

Определение 6 (AUC-ROC). *Площадь под ROC-кривой AUC (Area Under Curve) является агрегированной характеристикой качества классификации, не зависящей от соотношения цен ошибок.*

Чем больше значение AUC, тем «лучше» модель классификации. Данный показатель часто используется для сравнительного анализа нескольких моделей классификации.

10 | Решающие деревья

Решающее дерево (Decision tree) — решение задачи обучения с учителем (*supervised learning*), основанный на том, как решает задачи прогнозирования человек. В общем случае — это k-ичное дерево с решающими правилами в нелистовых вершинах (узлах) и некотором заключении о целевой функции в листовых вершинах (прогнозом). Решающее правило — некоторая функция от объекта, позволяющее определить, в какую из дочерних вершин нужно поместить рассматриваемый объект. В листовых вершинах могут находиться разные объекты: класс, который нужно присвоить попавшему туда объекту (в задаче классификации), вероятности классов (в задаче классификации), непосредственно значение целевой функции (задача регрессии).

10.1 Определение

Рассмотрим бинарное дерево, в котором:

- каждой внутренней вершине v приписана функция (или предикат) $\beta_v : \mathbb{X} \to \{0, 1\}$;
- каждой листовой вершине v приписан прогноз $c_v \in Y$ (в случае с классификацией листу также может быть приписан вектор вероятностей).

Рассмотрим теперь алгоритм $a(x)$, который стартует из корневой вершины v_0 и вычисляет значение функции β_{v_0}. Если оно равно нулю, то алгоритм переходит в левую дочернюю вершину, иначе в правую, вычисляет значение предиката в новой вершине и делает переход или влево, или вправо. Процесс продолжается, пока не будет достигнута листовая вершина; алгоритм возвращает тот класс, который приписан этой вершине. Такой алгоритм называется **бинарным решающим деревом**.

На практике в большинстве случаев используются одномерные предикаты β_v, которые сравнивают значение одного из признаков с порогом:

$$\beta_v(x; j, t) = [x_j < t].$$

Существуют и многомерные предикаты, например:

- линейные $\beta_v(x) = [\langle w, x \rangle < t]$;
- метрические $\beta_v(x) = [\rho(x, x_v) < t]$, где точка x_v является одним из объектов выборки любой точкой признакового пространства.

Многомерные предикаты позволяют строить ещё более сложные разделяющие поверхности, но очень редко используются на практике — например, из-за того, что усиливают и без того выдающиеся способности деревьев к переобучению.

10.2 Критерии расщепления

При построении дерева необходимо задать *функционал качества*, на основе которого осуществляется разбиение выборки на каждом шаге. Обозначим через R_m множество объектов, попавших в вершину, разбиваемую на данном шаге, а через R_ℓ и R_r — объекты, попадающие в левое и правое поддерево соответственно при заданном предикате. Мы будем использовать функционалы следующего вида:

$$Q(R_m, j, s) = H(R_m) - \frac{|R_\ell|}{|R_m|} H(R_\ell) - \frac{|R_r|}{|R_m|} H(R_r).$$

Здесь $H(R)$ — это *критерий информативности* (impurity criterion), который оценивает качество распределения целевой переменной среди объектов множества R. Чем меньше разнообразие целевой переменной, тем меньше должно быть значение критерия информативности — и, соответственно, мы будем пытаться минимизировать его значение. Функционал качества $Q(R_m, j, s)$ мы при этом будем максимизировать.

Как уже обсуждалось выше, в каждом листе дерево будет выдавать константу — вещественное число, вероятность или класс. Исходя из этого, можно предложить оценивать качество множества объектов R тем, насколько хорошо их целевые переменные предсказываются константой (при оптимальном выборе этой константы):

$$H(R) = \min_{c \in} \frac{1}{|R|} \sum_{(x_i, y_i) \in R} L(y_i, c),$$

где $L(y, c)$ — некоторая функция потерь. Далее мы обсудим, какие именно критерии информативности часто используют в задачах регрессии и классификации.

10.2.1 Регрессия

Как обычно, в регрессии выберем квадрат отклонения в качестве функции потерь. В этом случае критерий информативности будет выглядеть как

$$H(R) = \min_{c \in} \frac{1}{|R|} \sum_{(x_i, y_i) \in R} (y_i - c)^2.$$

Как известно, минимум в этом выражении будет достигаться на среднем значении целевой переменной. Значит, критерий можно переписать в следующем виде:

$$H(R) = \frac{1}{|R|} \sum_{(x_i, y_i) \in R} \left(y_i - \frac{1}{|R|} \sum_{(x_j, y_j) \in R} y_j \right)^2.$$

Мы получили, что информативность вершины измеряется её дисперсией — чем ниже разброс целевой переменной, тем лучше вершина. Разумеется, можно использовать и другие функции ошибки L — например, при выборе абсолютного отклонения мы получим в качестве критерия среднее абсолютное отклонение от медианы.

10.2.2 Классификация

Обозначим через p_k долю объектов класса k ($k \in \{1, \ldots, K\}$), попавших в вершину R:

$$p_k = \frac{1}{|R|} \sum_{(x_i, y_i) \in R} [y_i = k].$$

Через k_* обозначим класс, чьих представителей оказалось больше всего среди объектов, попавших в данную вершину: $k_* = \arg\max_k p_k$.

Ошибка классификации Рассмотрим индикатор ошибки как функцию потерь:

$$H(R) = \min_{c\in} \frac{1}{|R|} \sum_{(x_i,y_i)\in R} [y_i \neq c].$$

Легко видеть, что оптимальным предсказанием тут будет наиболее популярный класс k_* — значит, критерий будет равен следующей доле ошибок:

$$H(R) = \frac{1}{|R|} \sum_{(x_i,y_i)\in R} [y_i \neq k_*] = 1 - p_{k_*}.$$

Данный критерий является достаточно грубым, поскольку учитывает частоту p_{k_*} лишь одного класса.

Критерий Джини Рассмотрим ситуацию, в которой мы выдаём в вершине не один класс, а распределение на всех классах $c = (c_1, \ldots, c_K)$, $\sum_{k=1}^K c_k = 1$. Качество такого распределения можно измерять, например, с помощью критерия Бриера (Brier score):

$$H(R) = \min_{\sum_k c_k=1} \frac{1}{|R|} \sum_{(x_i,y_i)\in R} \sum_{k=1}^K (c_k - [y_i = k])^2.$$

Можно показать, что оптимальный вектор вероятностей состоит из долей классов p_k:

$$c_* = (p_1, \ldots, p_K)$$

Если подставить эти вероятности в исходный критерий информативности и провести ряд преобразований, то мы получим критерий Джини:

$$H(R) = \sum_{k=1}^K p_k(1 - p_k).$$

Энтропийный критерий Мы уже знакомы с более популярным способом оценивания качества вероятностей — логарифмическими потерями, или логарифмом правдоподобия:

$$H(R) = \min_{\sum_k c_k=1} \left(-\frac{1}{|R|} \sum_{(x_i,y_i)\in R} \sum_{k=1}^K [y_i = k] \log c_k \right).$$

Для вывода оптимальных значений c_k вспомним, что все значения c_k должны суммироваться в единицу. Как известного из методов оптимизации, для учёта этого ограничения необходимо искать минимум лагранжиана:

$$L(c, \lambda) = -\frac{1}{|R|} \sum_{(x_i,y_i)\in R} \sum_{k=1}^K [y_i = k] \log c_k + \lambda \sum_{k=1}^K c_k \to \min_{c_k}$$

Дифференцируя, получаем:

$$\frac{\partial}{\partial c_k} L(c, \lambda) = -\frac{1}{|R|} \sum_{(x_i, y_i) \in R} [y_i = k] \frac{1}{c_k} + \lambda = -\frac{p_k}{c_k} + \lambda = 0,$$

откуда выражаем $c_k = p_k / \lambda$. Суммируя эти равенства по k, получим

$$1 = \sum_{k=1}^{K} c_k = \frac{1}{\lambda} \sum_{k=1}^{K} p_k = \frac{1}{\lambda},$$

откуда $\lambda = 1$. Значит, минимум достигается при $c_k = p_k$, как и в предыдущем случае. Подставляя эти выражения в критерий, получим, что он будет представлять собой энтропию распределения классов:

$$H(R) = -\sum_{k=1}^{K} p_k \log p_k.$$

Из теории вероятностей известно, что энтропия ограничена снизу нулем, причем минимум достигается на вырожденных распределениях ($p_i = 1$, $p_j = 0$ для $i \neq j$). Максимальное же значение энтропия принимает для равномерного распределения. Отсюда видно, что энтропийный критерий отдает предпочтение более «вырожденным» распределениям классов в вершине.

10.3 Критерии останова

- Ограничение максимальной глубины дерева.
- Ограничение минимального числа объектов в листе.
- Ограничение максимального количества листьев в дереве.
- Останов в случае, если все объекты в вершине относятся к одному классу.
- Требование, что Information gain при дроблении улучшался как минимум на s процентов.

10.4 Метод обработки пропущенных значений

10.4.1 Пропуск пропущенных значений

При обучении дерева объекты с пропущенными значениями у признака, по которому идет разбиение, игнорируются:

$$\Delta(X_t, t) \approx \frac{\#\{x \in X_t : x_{i(t)} \text{not missing}\}}{N(t)} \cdot \Delta(\{x \in X_t \ : x_{i(t)} \text{not missing}\}, t)$$

При построении прогноза при необходимости разбить подвыборку в вершине t по отсутствующему признаку происходит следующая процедура: будем как бы предполагать, что этот признак принимает случайное значение. Определим по обучающей выборке вероятности, с которой новый объект попадёт к каждому потомку — $w_1, w_2, \ldots, w_R, w_i = \dfrac{N(t_i)}{N(t)}$. Затем отправим объект независимо к каждому потомку, получим прогнозы $y_1, y_2, \ldots, y_R$. В случае регрессии это будут непосредственно значения целевой функции, в случае классификации — вероятности принадлежности какому-то зафиксированному классу y (который, как будет видно ниже, надо перебирать): $y_i = \mathbb{P}(y|x, t_i)$

Дальше можно поступать образом, схожим с выбором прогноза в листе: так, для регрессии можно просто объединить отклики с вероятностями в качестве весов,

$$\hat{y} = w_1 y_1 + \cdots + w_R y_R$$

Пусть t_0 — корень дерева. В случае классификации

$$\hat{y} = \underset{y \in \mathbb{Y}}{\operatorname{argmax}} \mathbb{P}(y|x, t_0) \stackrel{\text{def}}{=} \underset{y \in \mathbb{Y}}{\operatorname{argmax}} \sum_{i=1}^{R} w_i \mathbb{P}(y|x, t_i)$$

Такой алгоритм работы с пропущенными значениями используется в ID3, C4.5.

10.4.2 Суррогатные разбиения

Находим другой признак, по которому разбиение будет максимально похожим. Выкидываем те объекты, по которому по данному признаку есть пропущенные значения, делаем разбиение. Ищем другой признак (видимо, предполагаем, что по нему нет пропущенных значений, либо сразу выкидывать объекты с пропущенными значениями и по первому приведенному признаку и по второму), берем максимально похожее разбиение на изначальное. Например, можно сравнивать как можно большее пересечение левых и правых поддеревьев (в случае бинарного дерева).

Такой алгоритм работы с пропущенными значениями используется в CART

10.5 Работа с категориальными признаками

10.5.1 One-hot кодирование

Из вершины t делаем столько детей, сколько уникальных значений у категориального признака. При таком подходе размер дерева увеличивается $\Rightarrow$ увеличивается риск переобучения.

10.5.2 Перевод категориальных в вещественные

Пусть категориальный признак x_j имеет множество значений $Q = \{u_1, \ldots, u_q\}, |Q| = q$. Разобьем множество значений на два непересекающихся подмножества: Q_1, Q_2 , и определим предикат как индикатор попадания в первое подмножество: $\beta(x) = \mathbb{I}[x_j \in Q_1]$. Таким образом, объект будет попадать в левое поддерево, если признак x_j попадает в множество Q_1 , и в правое поддерево в противном случае. Основная проблема заключается в том, что для построения оптимального предиката нужно перебрать $2^{q-1} - 1$ вариантов разбиения, что может быть не вполне возможным.

Оказывается, можно обойтись без полного перебора в случаях с бинарной классификацией и регрессией. Обозначим через $R_m(u)$ множество объектов, которые попали в вершину m и у которых j-й признак имеет значение u; через $N_m(u)$ обозначим количество таких объектов. В случае с бинарной классификацией упорядочим все значения категориального признака на основе того, какая доля объектов с таким значением имеет класс $+1$:

$$\frac{1}{N_m(u_{(1)})} \sum_{x_i \in R_m(u_{(1)})} \mathbb{I}[y_i = +1] \leq \cdots \leq \frac{1}{N_m(u_{(q)})} \sum_{x_i \in R_m(u_{(q)})} \mathbb{I}[y_i = +1]$$

после чего заменим категорию $u_{(i)}$ на число i, и будем искать разбиение как для вещественного признака. Можно показать, что если искать оптимальное разбиение по критерию

Джини или энтропийному критерию, то мы получим такое же разбиение, как и при переборе по всем возможным $2^{q-1}-1$ вариантам. Для задачи регрессии с MSE-функционалом это тоже будет верно, если упорядочивать значения признака по среднему ответу объектов с таким значением:

$$\frac{1}{N_m(u_{(1)})}\sum_{x_i\in R_m(u_{(1)})} y_i \leq \cdots \leq \frac{1}{N_m(u_{(q)})}\sum_{x_i\in R_m(u_{(q)})} y_i$$

10.6 Методы построения деревьев

Существует несколько популярных методов построения деревьев:

- ID3: использует энтропийный критерий. Строит дерево до тех пор, пока в каждом листе не окажутся объекты одного класса, либо пока разбиение вершины дает уменьшение энтропийного критерия.
- C4.5: использует критерий Gain Ratio (нормированный энтропийный критерий). Критерий останова — ограничение на число объектов в листе. Стрижка производится с помощью метода Error-Based Pruning, который использует оценки обобщающей способности для принятия решения об удалении вершины. Обработка пропущенных значений осуществляется с помощью метода, который игнорирует объекты с пропущенными значениями при вычислении критерия ветвления, а затем переносит такие объекты в оба поддерева с определенными весами.
- CART: использует критерий Джини. Стрижка осуществляется с помощью Cost-Complexity Pruning. Для обработки пропусков используется метод суррогатных предикатов.

10.7 Переобучение в деревьях

Ранее мы решали проблему переобучения путем ограничения глубины деревьев, но можно подойти к вопросу и более гибко. Мы выясняли, что дерево $b(x)$ можно описать формулой

$$b(x) = \sum_{j=1}^{J} b_j[x \in R_j]$$

Его сложность зависит от двух показателей:

1. Число листьев J. Чем больше листьев имеет дерево, тем сложнее его разделяющая поверхность, тем больше у него параметров и тем выше риск переобучения.
2. Норма коэффициентов в листьях $\|b\|_2^2 = \sum_{j=1}^{J} b_j^2$. Чем сильнее коэффициенты отличаются от нуля, тем сильнее данный базовый алгоритм будет влиять на итоговый ответ композиции.

Добавляя регуляризаторы, штрафующие за оба этих вида сложности, получаем следующую задачу:

$$\sum_{i=1}^{\ell}\left(-s_i b(x_i) + \frac{1}{2}h_i b^2(x_i)\right) + \gamma J + \frac{\lambda}{2}\sum_{j=1}^{J} b_j^2 \to \min_b$$

Если вспомнить, что дерево $b(x)$ дает одинаковые ответы на объектах, попадающих в один лист, то можно упростить функционал:

$$\sum_{j=1}^{J}\left\{\underbrace{\left(-\sum_{i\in R_j} s_i\right)}_{=-S_j} b_j + \frac{1}{2}\left(\lambda + \underbrace{\sum_{i\in R_j} h_i}_{=H_j}\right) b_j^2 + \gamma\right\} \to \min_b$$

Каждое слагаемое здесь можно минимизировать по b_j независимо. Заметим, что отдельное слагаемое представляет собой параболу относительно b_j, благодаря чему можно аналитически найти оптимальные коэффициенты в листьях:

$$b_j = \frac{S_i}{H_j + \lambda}$$

Подставляя данное выражение обратно в функционал, получаем, что ошибка дерева с оптимальными коэффициентами в листьях вычисляется по формуле

$$H(b) = -\frac{1}{2}\sum_{j=1}^{J}\frac{S_j^2}{H_j+\lambda} + \gamma J \tag{2}$$

11 | Постановка задачи PCA. Как выбирать оптимальную размерность маломерного пространства?

11.1 Постановка задачи PCA

В машинном обучении часто возникает задача уменьшения размерности признакового пространства. Для этого можно, например, удалять признаки, которые слабо коррелируют с целевой переменной; выбрасывать признаки по одному и проверять качество модели на тестовой выборке; перебирать случайные подмножества признаков в поисках лучших наборов. Ещё одним из подходов к решению задачи является поиск новых признаков, каждый из которых является линейной комбинацией исходных признаков. В случае использования квадратичной функции ошибки при поиске такого приближения получается *метод главных компонент* (principal component analysis, PCA), о котором и пойдет речь.

Пусть $X \in \mathbb{R}^{\ell\times D}$ — матрица «объекты-признаки», где ℓ — число объектов, а D — число признаков. Поставим задачу уменьшить размерность пространства до d. Будем считать, что данные являются центрированными — то есть среднее в каждом столбце матрицы X равно нулю.

Будем искать главные компоненты $u_1, \ldots, u_D \in \mathbb{R}^D$, которые удовлетворяют следующим требованиям:

1. Они ортогональны: $\langle u_i, u_j\rangle = 0$, $i \neq j$;
2. Они нормированы: $\|u_i\|^2 = 1$;
3. При проецировании выборки на компоненты $u_1, \ldots, u_d$ получается максимальная дисперсия среди всех возможных способов выбрать d компонент.

Чтобы понизить размерность выборки до d, мы будем проецировать её на первые d компонент — из последнего свойства следует, что это оптимальный способ снижения размерности.

Дисперсия проецированной выборки показывает, как много информации нам удалось сохранить после понижения размерности — и поэтому мы требуем максимальной дисперсии от проекций.

Проекция объекта x на компоненту u_i вычисляется как $\langle x, u_i \rangle$, а проекция всей выборки на эту компоненту — как Xu_i. Если за U_d обозначить матрицу, столбцы которой равны первым d компонентам, проекция выборки на них будет записываться как XU_d, а дисперсия проецированной выборки будет вычисляться как след ковариационной матрицы:

$$\operatorname{tr} U_d^T X^T X U_d = \sum_{i=1}^{d} \|Xu_i\|^2.$$

Начнём с первой компоненты. Сведём все требования к ней в оптимизационную задачу:

$$\begin{cases} \|Xu_1\|^2 \to \max_{u_1} \\ \|u_1\|^2 = 1 \end{cases}$$

Запишем лагранжиан:

$$L(u_1, \lambda) = \|Xu_1\|^2 + \lambda(\|u_1\|^2 - 1).$$

Продифференцируем его и приравняем нулю:

$$\frac{\partial L}{\partial u_1} = 2X^T X u_1 + 2\lambda u_1 = 0.$$

Отсюда получаем, что u_1 должен быть собственным вектором ковариационной матрицы $X^T X$. Учтём это и преобразуем функционал:

$$\|Xu_1\|^2 = u_1^T X^T X u_1 = \lambda u_1^T u_1 = \lambda \to \max_{u_1}$$

Значит, собственный вектор u_1 должен соответствовать максимальному собственному значению.

Для следующих компонент к оптимизационной задаче будут добавляться требования ортогональности предыдущим компонентам. Решая эти задачи, мы получим, что главная компонента u_i равна собственному вектору, соответствующему i-му собственному значению.

После того, как найдены главные компоненты, можно проецировать на них и новые данные. Если нам нужно работать с тестовой выборкой X', то её проекции вычисляются как $Z' = X'U_d$. Отметим также, что в методе главных компонент новые признаки вычисляются как линейные комбинации старых:

$$z'_{ij} = \sum_{k=1}^{D} x'_{ik} u_{kj}.$$

Альтернативные постановки Существует несколько других постановок задачи понижения размерности, приводящих к методу главных компонент.

Первый способ основан на матричном разложении. Будем искать матрицу с новыми признаковыми описаниями $Z \in \mathbb{R}^{\ell \times d}$ и матрицу проецирования $U \in \mathbb{R}^{D \times d}$, произведение которых даёт лучшее приближение исходной матрицы X:

$$\|X - ZU^T\|^2 \to \min_{Z,U}$$

Решением данной задачи также являются собственные векторы ковариационной матрицы.

Второй способ состоит в поиске такого линейного подпространства, что расстояние от исходных объектов до их проекций на это подпространство будет минимальным. В этом случае задача оказывается эквивалентной задаче максимизации дисперсии проекций.

11.2 Выбор оптимальной размерности

12 | Постановка задачи кластеризации. Алгоритм K-means.

12.1 Кластеризация

Пусть дана выборка объектов $X = (x_i)_{i=1}^{\ell}$, $x_i \in \mathbb{X}$. В задаче кластеризации требуется выявить в данных K кластеров — таких областей, что объекты внутри одного кластера похожи друг на друга, а объекты из разных кластеров друг на друга не похожи. Более формально, требуется построить алгоритм $a : \mathbb{X} \to \{1, \ldots, K\}$, определяющий для каждого объекта номер его кластера; число кластеров K может либо быть известно, либо являться параметром.

Кластеризовать можно много что: новости по сюжетам, пиксели на изображении по принадлежности объекту, музыку по жанрам, сообщения на форуме по темам, клиентов по типу поведения.

12.2 K-Means

Одним из наиболее популярных методов кластеризации является *K-Means*, который оптимизирует внутрикластерное расстояние, в котором используется квадрат евклидовой метрики.

Заметим, что в данном функционале имеется две степени свободы: центры кластеров c_k и распределение объектов по кластерам $a(x_i)$. Выберем для этих величин произвольные начальные приближения, а затем будем оптимизировать их по очереди:

1. Зафиксируем центры кластеров. В этом случае внутрикластерное расстояние будет минимальным, если каждый объект будет относиться к тому кластеру, чей центр является ближайшим:
$$a(x_i) = \arg\min_{1 \leqslant k \leqslant K} \rho(x_i, c_k).$$

2. Зафиксируем распределение объектов по кластерам. В этом случае внутрикластерное расстояние с квадратом евклидовой метрики можно продифференцировать по центрам кластеров и вывести аналитические формулы для них:
$$c_k = \frac{1}{\sum_{i=1}^{\ell}[a(x_i) = k]} \sum_{i=1}^{\ell}[a(x_i) = k]x_i.$$

Повторяя эти шаги до сходимости, мы получим некоторое распределение объектов по кластерам. Новый объект относится к тому кластеру, чей центр является ближайшим.

Результат работы метода K-Means существенно зависит от начального приблжения. Существует большое количество подходов к инициализации; одним из наиболее успешных считается k-means++.

13 | Сингулярное разложение (SVD) определение, его связь с PCA.

13.1 Определение

Предположим, что $X \in \mathbb{R}^{N \times D}$, $rankX = R$. Тогда существуют

$$U \in \mathbb{R}^{N \times R}$$

$$\Sigma \in \mathbb{R}^{R \times R}$$

$$V^T \in \mathbb{R}^{R \times D}$$

такие, что

$$X = U\Sigma V^T$$

$$\Sigma = diag\{\sigma_1, ..., \sigma_R\}, \sigma_1 \geqslant \sigma_2 \geqslant ... \geqslant \sigma_R > 0$$

$$U^T U = I, V^T V = I$$

. Для любой матрицы существует ее SVD-разложение. Оно определено с точностью до перестановок одинаковых собственных значений $\sigma_i^2, i = 1 \dots R$, и соответствующих им собственных векторов. Если же все собственные значения уникальны, то разложение единственно.

Свойства SVD-разложения и Связь с главными компонентами

13.2 Связь с PCA

Приглядимся поближе к SVD-разложению. Что мы увидим?

1. Столбцы U — ортонормированный базис (ОНБ) столбцов X .
2. Строки V^T — ОНБ строк X .
3. Строки U — нормированные координаты строк V^T .
4. Σ — масштабирование, то есть $\sigma_1, \dots, \sigma_R$ — величины "вхождения"каждой строки V^T .
5. $XX^TU = U\Sigma^2 \Rightarrow$ столбцы U — собственные векторы матрицы XX^T , отвечающие собственным значениям $\sigma_1^2, \dots, \sigma_R^2$.
6. $X^TXV = V\Sigma^2 \Rightarrow$ столбцы V — собственные векторы матрицы X^TX , отвечающие собственным значениям $\sigma_1^2, \dots, \sigma_R^2$, а значит, матрица V состоит из первых R главных компонент.

14 | l-1, l-2 регуляризация в задаче линейной регрессии и классификации. Какая из них позволяет отбирать признаки и почему?

14.1 Виды регуляризации

Часто регуляризация представляет собой просто некоторую добавку $R(w)$ (где w — параметры модели) к функции потерь $L(f(x, w), y)$ так что задача приобретает вид:

$$\min_w \sum_{i=1}^{N} L(f(x_i, w), y_i)) + \lambda R(w)$$

Часто используемые $R(w)$:

- L2 регуляризация — $R(w) = ||w||_2^2 = \sum_{i=1}^{d} {w_i}^2$
- L1 регуляризация — $R(w) = ||w||_1 = \sum_{i=1}^{d} |w_i|$
- ElasticNet регуляризация — $R(w) = \lambda_1 ||w||_2^2 + \lambda_2 ||w||_1$

14.2 Отбор признаков

Предположим, что перед нами стоит задача линейной регрессии с L1 регуляризацией (lasso regression):

- $f(x, w) = w^T x + w_0$
- $L(w) = \min_w \sum_{i=1}^{N} (w^T x_i + w_0 - y_i)^2 + \lambda ||w||_1$

L1 регуляризация приведет к занулению весов некоторых признаков так как градиент равен:

$$\nabla_w L(w) = \sum_{i=1}^{N} 2x_i^T (w^T x_i + w_0 - y_i) + \lambda \operatorname{sign}(w)$$

Соответственно при достаточно большом λ некоторые признаки обязательно обнулятся. Тогда отбираются те признаки, при которых вес не равен нулю.

15 | Логистическая регрессия

Метод обучения, которые получается при использовании логистической функции потерь, называется логистической регрессией. Основным его свойством является тот факт, что он корректно оценивает вероятность принадлежности объекта к каждому из классов.

Пусть в каждой точке пространства объектов $x \in \mathbb{X}$ задана вероятность $p(y = +1 \mid x)$ того, что объект x будет принадлежать классу $+1$. Это означает, что мы допускаем наличие в выборке нескольких объектов с одинаковым признаковым описанием, но с разными значениями целевой переменной; причём если устремить количество объекта x в выборке к бесконечности, то доля положительных объектов среди них будет стремиться к $p(y = +1 \mid x)$.

Итак, рассмотрим точку x пространства объектов. Как мы договорились, в ней имеется распределение на ответах $p(y = +1 \mid x)$. Допустим, алгоритм $b(x)$ возвращает числа из отрезка $[0, 1]$. Наша задача — выбрать для него такую процедуру обучения, что в точке x ему будет оптимально выдавать число $p(y = +1 \mid x)$. Если в выборке объект x встречается n раз с ответами $\{y_1, \ldots, y_n\}$, то получаем следующее требование:

$$\arg\min_{b \in \mathbb{R}} \frac{1}{n} \sum_{i=1}^{n} L(y_i, b) \approx p(y = +1 \mid x).$$

При стремлении n к бесконечности получим, что функционал стремится к матожиданию ошибки:

$$\arg\min_{b \in \mathbb{R}} [L(y, b) \mid x] = p(y = +1 \mid x).$$

Этим свойством обладает, например, квадратичная функция потерь $L(y,z) = (y-z)^2$, если в ней для положительных объектов использовать истинную метку $y = 1$, а для отрицательных брать $y = 0$.

Примером функции потерь, которая не позволяет оценивать вероятности, является модуль отклонения $L(y,x) = |y-z|$. Можно показать, что с точки зрения данной функции оптимальным ответом всегда будет либо ноль, либо единица.

Это требование можно воспринимать более просто. Пусть один и тот же объект встречается в выборке 1000 раз, из которых 100 раз он относится к классу $+1$, и 900 раз — к классу -1. Поскольку это один и тот же объект, классификатор должен выдавать один ответ для каждого из тысячи случаев. Можно оценить матожидание функции потерь в данной точке по 1000 примеров при прогнозе b:

$$\Big[L(y,b)\,|\,x\Big] \approx \frac{100}{1000}L(1,b) + \frac{900}{1000}L(-1,b).$$

Наше требование, по сути, означает, что оптимальный ответ с точки зрения этой оценки должен быть равен $1/10$:

$$\arg\min_{b\in\mathbb{R}} \left(\frac{100}{1000}L(1,b) + \frac{900}{1000}L(-1,b)\right) = \frac{1}{10}.$$

Хотя квадратичная функция потерь и приводит к корректному оцениванию вероятностей, она не очень хорошо подходит для решения задачи классификации. Причиной этому в том числе являются и слишком низкие штрафы за ошибку — так, если объект положительный, а модель выдаёт для него вероятность первого класса $b(x) = 0$, то штраф за это равен всего лишь единице: $(1-0)^2 = 1$.

15.1 Вывод формулы логистической регрессии

Попробуем сконструировать функцию потерь из других соображений. Если алгоритм $b(x) \in [0,1]$ действительно выдает вероятности, то они должны согласовываться с выборкой. С точки зрения алгоритма вероятность того, что в выборке встретится объект x_i с классом y_i, равна $b(x_i)^{[y_i=+1]}(1-b(x_i))^{[y_i=-1]}$. Исходя из этого, можно записать правдоподобие выборки (т.е. вероятность получить такую выборку с точки зрения алгоритма):

$$Q(a,X) = \prod_{i=1}^{\ell} b(x_i)^{[y_i=+1]}(1-b(x_i))^{[y_i=-1]}.$$

Данное правдоподобие можно использовать как функционал для обучения алгоритма — с той лишь оговоркой, что удобнее оптимизировать его логарифм:

$$-\sum_{i=1}^{\ell}\big([y_i=+1]\log b(x_i) + [y_i=-1]\log(1-b(x_i))\big) \to \min$$

Данная функция потерь называется логарифмической (log-loss). Покажем, что она также позволяет корректно предсказывать вероятности. Запишем матожидание функции потерь в точке x:

$$\Big[L(y,b)\,|\,x\Big] = \Big[-[y=+1]\log b - [y=-1]\log(1-b)\,|\,x\Big] =$$
$$= -p(y=+1\,|\,x)\log b - (1-p(y=+1\,|\,x))\log(1-b).$$

Продифференцируем по b:

$$\frac{\partial}{\partial b}\Big[L(y,b)\,|\,x\Big] = -\frac{p(y=+1\,|\,x)}{b} + \frac{1-p(y=+1\,|\,x)}{1-b} = 0.$$

Легко видеть, что оптимальный ответ алгоритма равен вероятности положительного класса:

$$b_* = p(y=+1\,|\,x).$$

Везде выше мы требовали, чтобы алгоритм $b(x)$ возвращал числа из отрезка $[0,1]$. Этого легко достичь, если положить $b(x) = \sigma(\langle w, x\rangle)$, где в качестве σ может выступать любая монотонно неубывающая функция с областью значений $[0,1]$. Мы будем использовать сигмоидную функцию: $\sigma(z) = \frac{1}{1+\exp(-z)}$. Таким образом, чем больше скалярное произведение $\langle w, x\rangle$, тем больше будет предсказанная вероятность. Как при этом можно интерпретировать данное скалярное произведение? Чтобы ответить на этот вопрос, преобразуем уравнение

$$p(y=1\,|\,x) = \frac{1}{1+\exp(-\langle w, x\rangle)}.$$

Выражая из него скалярное произведение, получим

$$\langle w, x\rangle = \log\frac{p(y=+1\,|\,x)}{p(y=-1\,|\,x)}.$$

Получим, что скалярное произведение будет равно логарифму отношения вероятностей классов (log-odds).

Как уже упоминалось выше, при использовании квадратичной функции потерь алгоритм будет пытаться предсказывать вероятности, но данная функция потерь является далеко не самой лучшей, поскольку слабо штрафует за грубые ошибки. Логарифмическая функция потерь подходит гораздо лучше, поскольку не позволяет алгоритму сильно ошибаться в вероятностях.

Подставим трансформированный ответ линейной модели в логарифмическую функцию потерь:

$$\begin{aligned}
&-\sum_{i=1}^{\ell}\left([y_i=+1]\log\frac{1}{1+\exp(-\langle w,x_i\rangle)} + [y_i=-1]\log\frac{\exp(-\langle w,x_i\rangle)}{1+\exp(-\langle w,x_i\rangle)}\right) = \\
&\quad= -\sum_{i=1}^{\ell}\left([y_i=+1]\log\frac{1}{1+\exp(-\langle w,x_i\rangle)} + [y_i=-1]\log\frac{1}{1+\exp(\langle w,x_i\rangle)}\right) = \\
&\quad= \sum_{i=1}^{\ell}\log\left(1+\exp(-y_i\langle w,x_i\rangle)\right).
\end{aligned}$$

Полученная функция в точности представляет собой логистические потери, упомянутые в начале. Линейная модель классификации, настроенная путём минимизации данного функционала, называется логистической регрессией. Как видно из приведенных рассуждений, она оптимизирует правдоподобие выборки и дает корректные оценки вероятности принадлежности к положительному классу.

16 | Support Vector Machines

Определение 7 (Support Mector Machine). *один из наиболее популярных методов обучения, который применяется для решения задач классификации и регрессии. Основная идея метода*

заключается в построении гиперплоскости, разделяющей объекты выборки оптимальным способом. Алгоритм работает в предположении, что чем больше расстояние (зазор) между разделяющей гиперплоскостью и объектами разделяемых классов, тем меньше будет средняя ошибка классификатора.

Рассмотрим задачу бинарной классификации, в которой объектам из $X = \mathbb{R}^n$ соответствует один из двух классов $Y = \{-1, +1\}$.

Пусть задана обучающая выборка пар "объект-ответ": $T^\ell = (\vec{x}_i, y_i)_{i=1}^\ell$. Необходимо построить алгоритм классификации $a(\vec{x}) : X \to Y$.

Рассмотрим подход к построению функции потерь, основанный на максимизации зазора между классами. Будем рассматривать линейные классификаторы вида

$$a(x) = \text{sign}(\langle w, x\rangle + b), \qquad w \in \mathbb{R}^d, b \in \mathbb{R}.$$

16.1 Разделимый случай

Будем считать, что существуют такие параметры w_* и b_*, что соответствующий им классификатор $a(x)$ не допускает ни одной ошибки на обучающей выборке. В этом случае говорят, что выборка *линейно разделима*.

Пусть задан некоторый классификатор $a(x) = \text{sign}(\langle w, x\rangle + b)$. Заметим, что если одновременно умножить параметры w и b на одну и ту же положительную константу, то классификатор не изменится. Распорядимся этой свободой выбора и отнормируем параметры так, что

$$\min_{x\in X} |\langle w, x\rangle + b| = 1. \tag{3}$$

Можно показать, что расстояние от произвольной точки $x_0 \in \mathbb{R}^d$ до гиперплоскости, определяемой данным классификатором, равно

$$\rho(x_0, a) = \frac{|\langle w, x\rangle + b|}{\|w\|}.$$

Тогда расстояние от гиперплоскости до ближайшего объекта обучающей выборки равно

$$\min_{x\in X} \frac{|\langle w, x\rangle + b|}{\|w\|} = \frac{1}{\|w\|} \min_{x\in X} |\langle w, x\rangle + b| = \frac{1}{\|w\|}.$$

Данная величина также называется *отступом (margin)*.

Таким образом, если классификатор без ошибок разделяет обучающую выборку, то ширина его разделяющей полосы равна $\frac{2}{\|w\|}$. Известно, что максимизация ширины разделяющей полосы приводит к повышению обобщающей способности классификатора [**?**]. Вспомним также, что на повышение обобщающей способности направлена и регуляризация, которая штрафует большую норму весов — а чем больше норма весов, тем меньше ширина разделяющей полосы.

Итак, требуется построить классификатор, идеально разделяющий обучающую выборку, и при этом имеющий максимальный отступ. Запишем соответствующую оптимизационную задачу, которая и будет определять метод опорных векторов для линейно разделимой выборки (hard margin support vector machine):

$$\begin{cases} \dfrac{1}{2}\|w\|^2 \to \min\limits_{w,b} \\ y_i\left(\langle w, x_i\rangle + b\right) \geqslant 1, \quad i = 1, \ldots, \ell. \end{cases} \tag{4}$$

Здесь мы воспользовались тем, что линейный классификатор дает правильный ответ на объекте x_i тогда и только тогда, когда $y_i(\langle w, x_i\rangle + b) \geqslant 0$. Более того, из условия нормировки (2) следует, что $y_i(\langle w, x_i\rangle + b) \geqslant 1$.

В данной задаче функционал является строго выпуклым, а ограничения линейными, поэтому сама задача является выпуклой и имеет единственное решение. Более того, задача является квадратичной и может быть решена крайне эффективно.

16.2 Неразделимый случай

Рассмотрим теперь общий случай, когда выборку невозможно идеально разделить гиперплоскостью. Это означает, что какие бы w и b мы не взяли, хотя бы одно из ограничений в задаче (3) будет нарушено:

$$\exists x_i \in X: \; y_i\left(\langle w, x_i\rangle + b\right) < 1.$$

Сделаем эти ограничения «мягкими», введя штраф $\xi_i \geqslant 0$ за их нарушение:

$$y_i\left(\langle w, x_i\rangle + b\right) \geqslant 1 - \xi_i, \quad i = 1, \ldots, \ell.$$

Отметим, что если отступ объекта лежит между нулем и единицей ($0 \leqslant y_i\left(\langle w, x_i\rangle + b\right) < 1$), то объект верно классифицируется, но имеет ненулевой штраф $\xi > 0$. Таким образом, мы штрафуем объекты за попадание внутрь разделяющей полосы.

Величина $\frac{1}{\|w\|}$ в данном случае называется *мягким отступом (soft margin)*. С одной стороны, мы хотим максимизировать отступ, с другой — минимизировать штраф за неидеальное разделение выборки $\sum_{i=1}^{\ell} \xi_i$. Эти две задачи противоречат друг другу: как правило, излишняя подгонка под выборку приводит к маленькому отступу, и наоборот — максимизация отступа приводит к большой ошибке на обучении. В качестве компромисса будем минимизировать взвешенную сумму двух указанных величин. Приходим к оптимизационной задаче, соответствующей методу опорных векторов для линейно неразделимой выборки (soft margin support vector machine)

$$\begin{cases} \dfrac{1}{2}\|w\|^2 + C\sum\limits_{i=1}^{\ell} \xi_i \to \min\limits_{w,b,\xi} \\ y_i\left(\langle w, x_i\rangle + b\right) \geqslant 1 - \xi_i, \quad i = 1, \ldots, \ell, \\ \xi_i \geqslant 0, \quad i = 1, \ldots, \ell. \end{cases} \tag{5}$$

Чем больше здесь параметр C, тем сильнее мы будем настраиваться на обучающую выборку.

Данная задача также является выпуклой и имеет единственное решение.

16.3 Теория двойственности

Двойственность, или принцип двойственности — принцип, по которому задачи оптимизации можно рассматривать с двух точек зрения, как прямую задачу или двойственную задачу. Решение двойственной задачи даёт нижнюю границу прямой задачи (при минимизации). Однако, в общем случае, значения целевых функций оптимальных решений прямой и двойственной задач не обязательно совпадают. Разница этих значений, если она наблюдается, называется разрывом двойственности. Для задач выпуклого программирования разрыв двойственности равен нулю при выполнении условий регулярности ограничений.

17 | Определение ядровой функции (K(x, z)). Обобщение SVM с помощью ядер.

Пусть X – некоторое пространство. Тогда отображение $K: X \times X \to \mathbb{R}$ называется *ядром* или *kernel function*, если оно представимо в виде:
$K(x,x') = \langle \psi(x), \psi(x') \rangle_H$, где ψ – некоторое отображение $\psi: X \to H$.

17.1 Обобщение SVM с помощью ядер.

Вместо скалярного произведения можно использовать ядра:

$$\begin{cases} \sum_{i=1}^{l} \lambda_i - \frac{1}{2} \sum_{i,j=1}^{l} \lambda_i \lambda_j y_i y_j K(x_i,x_j) \to \max\limits_{\lambda} \\ 0 \le \lambda_i \le C,\ i = 1,2,...,l \\ \sum_{i=1}^{l} \lambda_i y_i = 0 \end{cases}$$

В этом случае классификатор будет иметь вид:

$$a(x) = sign(\sum_{i=1}^{l} \lambda_i y_i K(x_i, x) + b)$$

18 | Постановка задачи гребневой регрессии (Ridge-regression). Формула оптимальных весов для неё.

Ридж-регрессия (англ. ridge regression) - это один из методов понижения размерности. Часто его применяют для борьбы с переизбыточностью данных, когда независимые переменные коррелируют друг с другом (т.е. имеет место мультиколлинеарность). Следствием этого является плохая обусловленность матрицы $X^T X$ и неустойчивость оценок коэффициентов регрессии. Оценки, например, могут иметь неправильный знак или значения, которые намного превосходят те, которые приемлемы из физических или практических соображений.

Метод стоит использовать, если:

- сильная обусловленность;
- сильно различаются собственные значения или некоторые из них близки к нулю;
- в матрице X есть почти линейно зависимые столбцы.

18.1 Постановка задачи

Решается задача регрессии. Применяется линейная модель (вообще говоря, один из признаков полагается константным для того, чтобы аппроксимирующая гиперплоскость не обязательно проходила через нуль, я не знаю, почему это практически всюду опускается): $f(x, \beta) = \langle \beta, x \rangle$. В изначальной постановке полагается, что вектор β находится методом Обычных Наименьших Квадратов (ОНК):

$$\sum_{n=1}^{N} (f(x_n, \beta) - y_n)^2 \longrightarrow \min_{\beta}$$

Аналитическое решение данной задачи: $\beta^* = (X^T X)^{-1} X^T Y$, однако при вырожденности матрицы $X^T X$ решение оказывается не единственным, а при ее плохой обусловленности — неустойчивым. Поэтому целесообразно ввести регуляризацию по параметру β , например, L_2 .

Таким образом, приходим к следующей задаче минимизации (гребневая (ridge) регрессия):

$$Q(\beta) = \|Y - X\beta\|^2 + \lambda \|\beta\|^2 \longrightarrow \min_{\beta}$$

где $\|\beta\|^2 = \sum_{i=1}^{D} {\beta_i}^2$, λ — параметр регуляризации(неотрицательное число).

18.2 Вывод оптимальных весов

Для нахождения оптимальных весов продифференцируем функционал по β и приравняем к 0:

$$\frac{\partial Q}{\partial \beta} = 2X^T(X\beta - Y) + 2\lambda\beta = 0$$

$$(X^T X + \lambda I)\beta = X^T Y$$

$$\beta^* = (X^T X + \lambda I)^{-1} X^T Y$$

При увеличении параметра λ решение становится более устойчивым, но с другой стороны — смещенным. При уменьшении — приходим к задаче ОНК без регуляризации: имеем шанс переобучиться. Поэтому нужно искать что-то посерединке.

19 | Бэггинг и случайный лес

19.1 Bagging

Бэггинг (bagging) — это технология классификации, использующая композиции алгоритмов, каждый из которых обучается независимо. Результат классификации определяется путем голосования. Бэггинг позволяет снизить процент ошибки классификации в случае, когда высока дисперсия ошибки базового метода.

Идея метода *Бэггинг* – технология классификации, где в отличие от бустинга все элементарные классификаторы обучаются и работают параллельно (независимо друг от друга). Идея заключается в том, что классификаторы не исправляют ошибки друг друга, а компенсируют их при голосовании. Базовые классификаторы должны быть независимыми, это могут быть классификаторы основанные на разных группах методов или же обученные на независимых наборах данных. Во втором случае можно использовать один и тот же метод.

В *бэггинге (bagging, bootstrap aggregation)* предлагается обучить некоторое число алгоритмов $b_n(x)$ с помощью метода $\tilde{\mu}$, и построить итоговую композицию как среднее данных базовых алгоритмов:

$$a_N(x) = \frac{1}{N}\sum_{n=1}^{N} b_n(x) = \frac{1}{N}\sum_{n=1}^{N} \tilde{\mu}(X)(x).$$

Algorithm 19.1 Random Forest

1: **for** $n = 1, \ldots, N$ **do**
2: Сгенерировать выборку $\tilde{X}_n$ с помощью бутстрэпа
3: Построить решающее дерево $b_n(x)$ по выборке $\tilde{X}_n$:

- дерево строится, пока в каждом листе не окажется не более $n_{\min}$ объектов
- при каждом разбиении сначала выбирается m случайных признаков из p, и оптимальное разделение ищется только среди них

4: Вернуть композицию $a_N(x) = \frac{1}{N}\sum_{n=1}^{N} b_n(x)$

19.2 Random Forest

Как мы выяснили, бэггинг позволяет объединить несмещенные, но чувствительные к обучающей выборке алгоритмы в несмещенную композицию с низкой дисперсией. Хорошим семейством базовых алгоритмов здесь являются решающие деревья — они достаточно сложны и могут достигать нулевой ошибки на любой выборке (следовательно, имеют низкое смещение), но в то же время легко переобучаются.

Метод *случайных лесов* [?] основан на бэггинге над решающими деревьями, см. алгоритм 19.1. Выше мы отметили, что бэггинг сильнее уменьшает дисперсию базовых алгоритмов, если они слабо коррелированы. В случайных лесах корреляция между деревьями понижается путем рандомизации по двум направлениям: по объектам и по признакам. Во-первых, каждое дерево обучается по бутстрапированной подвыборке. Во-вторых, в каждой вершине разбиение ищется по подмножеству признаков.

20 | Градиентный бустинг

21 | Бустинг в задаче регрессии

Рассмотрим задачу минимизации квадратичного функционала:

$$\frac{1}{2}\sum_{i=1}^{\ell}(a(x_i) - y_i)^2 \to \min_a$$

Будем искать итоговый алгоритм в виде суммы *базовых моделей* (weak learners) $b_n(x)$:

$$a_N(x) = \sum_{n=1}^{N} b_n(x),$$

где базовые алгоритмы b_n принадлежат некоторому семейству $\mathbb{A}$.

Построим первый базовый алгоритм:

$$b_1(x) := \arg\min_{b \in \mathbb{A}} \frac{1}{2}\sum_{i=1}^{\ell}(b(x_i) - y_i)^2$$

Решение такой задачи не представляет трудностей для многих семейств алгоритмов. Теперь мы можем посчитать остатки на каждом объекте — расстояния от ответа нашего алгоритма до истинного ответа:

$$s_i^{(1)} = y_i - b_1(x_i)$$

Если прибавить эти остатки к ответам построенного алгоритма, то он не будет допускать ошибок на обучающей выборке. Значит, будет разумно построить второй алгоритм так, чтобы его ответы были как можно ближе к остаткам:

$$b_2(x) := \arg\min_{b \in \mathcal{A}} \frac{1}{2} \sum_{i=1}^{\ell} (b(x_i) - s_i^{(1)})^2$$

Каждый следующий алгоритм тоже будем настраивать на остатки предыдущих:

$$s_i^{(N)} = y_i - \sum_{n=1}^{N-1} b_n(x_i) = y_i - a_{N-1}(x_i), \qquad i = 1, \ldots, \ell;$$

$$b_N(x) := \arg\min_{b \in \mathcal{A}} \frac{1}{2} \sum_{i=1}^{\ell} (b(x_i) - s_i^{(N)})^2$$

Описанный метод прост в реализации, хорошо работает и может быть найден во многих библиотеках — например, в `scikit-learn`.

Заметим, что остатки могут быть найдены как антиградиент функции потерь по ответу модели, посчитанный в точке ответа уже построенной композиции:

$$s_i^{(N)} = y_i - a_{N-1}(x_i) = -\frac{\partial}{\partial z} \frac{1}{2} (z - y_i)^2 \Big|_{z = a_{N-1}(x_i)}$$

Получается, что выбирается такой базовый алгоритм, который как можно сильнее уменьшит ошибку композиции — это свойство вытекает из его близости к антиградиенту функционала на обучающей выборке. Попробуем разобраться с этим свойством подробнее, а также попытаемся обобщить его на другие функции потерь. Это наблюдение наводит нас на мысль, что аналогичным образом можно было бы строить композиции, оптимизирующие и другие функции потерь — достаточно заменить остатки $z_i^{(N)}$ на градиент нужного функционала. Именно так и работает градиентный бустинг, о котором пойдет речь ниже.

21.1 Алгоритм градиентного бустинга

Пусть дана некоторая дифференцируемая функция потерь $L(y, z)$. Будем строить взвешенную сумму базовых алгоритмов:

$$a_N(x) = \sum_{n=0}^{N} \gamma_n b_n(x)$$

Заметим, что в композиции имеется начальный алгоритм $b_0(x)$. Как правило, коэффициент γ_0 при нем берут равным единице, а сам алгоритм выбирают очень простым, например:

- нулевым $b_0(x) = 0$;
- возвращающим самый популярный класс (в задачах классификации):

$$b_0(x) = \arg\max_{y \in} \sum_{i=1}^{\ell} [y_i = y]$$

- возвращающим средний ответ (в задачах регрессии):

$$b_0(x) = \frac{1}{\ell} \sum_{i=1}^{\ell} y_i$$

Допустим, мы построили композицию $a_{N-1}(x)$ из $N-1$ алгоритма, и хотим выбрать следующий базовый алгоритм $b_N(x)$ так, чтобы как можно сильнее уменьшить ошибку:

$$\sum_{i=1}^{\ell} L(y_i, a_{N-1}(x_i) + \gamma_N b_N(x_i)) \to \min_{b_N, \gamma_N}$$

Ответим в первую очередь на следующий вопрос: если бы в качестве алгоритма $b_N(x)$ мы могли выбрать совершенно любую функцию, то какие значения ей следовало бы принимать на объектах обучающей выборки? Иными словами, нам нужно понять, какие числа $s_1, \ldots, s_\ell$ надо выбрать для решения следующей задачи:

$$\sum_{i=1}^{\ell} L(y_i, a_{N-1}(x_i) + s_i) \to \min_{s_1, \ldots, s_\ell}$$

Понятно, что можно требовать $s_i = y_i - a_{N-1}(x_i)$, но такой подход никак не учитывает особенностей функции потерь $L(y, z)$ и требует лишь точного совпадения предсказаний и истинных ответов. Более разумно потребовать, чтобы сдвиг s_i был противоположен производной функции потерь в точке $z = a_{N-1}(x_i)$:

$$s_i = -\left.\frac{\partial L}{\partial z}\right|_{z=a_{N-1}(x_i)}$$

В этом случае мы сдвинемся в сторону скорейшего убывания функции потерь. Заметим, что вектор сдвигов $s = (s_1, \ldots, s_\ell)$ совпадает с антиградиентом:

$$\left(-\left.\frac{\partial L}{\partial z}\right|_{z=a_{N-1}(x_i)}\right)_{i=1}^{\ell} = -\nabla_z \sum_{i=1}^{\ell} L(y_i, z_i)\big|_{z_i=a_{N-1}(x_i)}$$

При таком выборе сдвигов s_i мы, по сути, сделаем один шаг градиентного спуска, двигаясь в сторону наискорейшего убывания ошибки на обучающей выборке. Отметим, что речь идет о градиентном спуске в ℓ-мерном пространстве предсказаний алгоритма на объектах обучающей выборки. Поскольку вектор сдвига будет свой на каждой итерации, правильнее обозначать его как $s_i^{(N)}$, но для простоты будем иногда опускать верхний индекс.

22 | Метрики кластеризации

Существует два подхода к измерению качества кластеризации: внутренний и внешний. Внутренний основан на некоторых свойствах выборки и кластеров, а внешний использует дополнительные данные — например, информацию об истинных кластерах.

Приведём несколько примеров внутренних метрик качества. Будем считать, что каждый кластер характеризуется своим *центром* c_k.

1. Внутрикластерное расстояние:

$$\sum_{k=1}^{K} \sum_{i=1}^{\ell} [a(x_i) = k] \rho(x_i, c_k), \tag{6}$$

   где $\rho(x, z)$ — некоторая функция расстояния. Данный функционал требуется минимизировать, поскольку в идеале все объекты кластера должны быть одинаковыми.

2. Межкластерное расстояние:

$$\sum_{i,j=1}^{\ell}[a(x_i) \neq a(x_j)]\rho(x_i, x_j).$$

Данный функционал нужно максимизировать, поскольку объекты из разных кластеров должны быть как можно менее похожими друг на друга.

3. Индекс Данна (Dunn Index):

$$\frac{\min\limits_{1\leqslant k<k'\leqslant K} d(k,k')}{\max\limits_{1\leqslant k\leqslant K} d(k)},$$

где $d(k,k')$ — расстояние между кластерами k и k' (например, евклидово расстояние между их центрами), а $d(k)$ — внутрикластерное расстояние для k-го кластера (например, сумма расстояний от всех объектов этого кластера до его центра). Данный индекс необходимо максимизировать.

Внешние метрики возможно использовать, если известно истинное распределение объектов по кластерам. В этом случае задачу кластеризации можно рассматривать как задачу многоклассовой классификации, и использовать любую метрику оттуда — F-меру с микро- или макро-усреднением.

23 | Многослойный персептрон

23.1 Определение

Многослойный персептрон — нейронная сеть прямого распространения сигнала (без обратных связей), в которой входной сигнал преобразуется в выходной, проходя последовательно через несколько слоев.

Первый из таких слоев называют входным, последний - выходным. Эти слои содержат так называемые вырожденные нейроны и иногда в количестве слоев не учитываются. Кроме входного и выходного слоев, в многослойном персептроне есть один или несколько промежуточных слоев, которые называют скрытыми.

В этой модели персептрона должен быть хотя бы один скрытый слой. Присутствие нескольких таких слоев оправдано лишь в случае использования нелинейных *функций активации*.

23.2 Функции активации

Наиболее часто в качестве функций активации используются следующие виды сигмоид:
Функция Ферми (экспоненциальная сигмоида):

$$f(s) = \frac{1}{1+e^{-2\alpha s}}$$

Рациональная сигмоида (при $\alpha = 0$ вырождается в т. н. пороговую функцию активации):

$$f(s) = \frac{s}{|s|+\alpha}$$

Гиперболический тангенс:

$$f(s) = \text{th}\,\frac{s}{\alpha} = \frac{e^{\frac{s}{\alpha}} - e^{-\frac{s}{\alpha}}}{e^{\frac{s}{\alpha}} + e^{-\frac{s}{\alpha}}}$$

,

где s — выход сумматора нейрона, α — произвольная константа.

Менее всего, сравнительно с другими сигмоидами, процессорного времени требует расчёт рациональной сигмоиды. Для вычисления гиперболического тангенса требуется больше всего тактов работы процессора. Если же сравнивать с пороговыми функциями активации, то сигмоиды рассчитываются очень медленно. Если после суммирования в пороговой функции сразу можно начинать сравнение с определённой величиной (порогом), то в случае сигмоидальной функции активации нужно рассчитать сигмоид (затратить время в лучшем случае на три операции: взятие модуля, сложение и деление), и только потом сравнивать с пороговой величиной (например, нулём). Если считать, что все простейшие операции рассчитываются процессором за примерно одинаковое время, то работа сигмоидальной функции активации после произведённого суммирования (которое займёт одинаковое время) будет медленнее пороговой функции активации в 4 раза.

24 | Chain rule для производной сложной функции. Алгоритм Backpropagation.

24.1 Backpropagation

Метод обратного распространения ошибки (Back propagation, backprop) - алгоритм обучения многослойных персептронов, основанный на вычислении градиента функции ошибок. В процессе обучения веса нейронов каждого слоя нейросети корректируются с учетом сигналов, поступивших с предыдущего слоя, и невязки каждого слоя, которая вычисляется рекурсивно в обратном направлении от последнего слоя к первому.