Машинное обучение
Теоретический минимум

ЭФ МГУ

2020

1 | Матричное дифференцирование

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.

3 | Настройка гиперпараметров с помощью кросс-валидации

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 | 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:

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

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*

5 | Определение линейного классификатора. Понятие отступа (Margin). Примеры верхних оценок на долю ошибок. Как обучаются веса?

5.1 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.

$$w = \underset{w\in\mathbb{R}^d}{\text{argmin}} Q(w,X^l)$$

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 $L(M) = \log(1 + e^{-M})$

5.2 Margin (отступ)

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

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

Пусть рассматривается задача классификации с C классами.

6.1 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 $\hat{y}(x) = \underset{c \in [1,\dots,C]}{\operatorname{argmax}} f_c(x)$.

6.2 One-vs-one

1. Обучим $\dfrac{C(C-1)}{2}$ бинарных классификаторов следующим образом: для каждой пары $i, j \in [1, \dots, C], i \neq j$ обучим бинарный классификатор $f_{i,j}(x)$ на тех объектах тренировочной выборки, ответ которых принимает значение i или j .
2. В результате получим $\dfrac{C(C-1)}{2}$ 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:

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

7 | 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.

8 | Матрица ошибок (confusion matrix). Определение accuracy, precision, recall, f1-measure.

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 | Определение решающего дерева. Критерии расщепления в случае задачи классификации и регрессии.

10.1 Решающие деревья

Рассмотрим бинарное дерево, в котором:

- каждой внутренней вершине v приписана функция (или предикат) $\beta_v :\to \{0, 1\}$;
- 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 ($k \in \{1, \dots, K\}$), 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: $k_* = \text{argmax}_k \, p_k$.

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 $c = (c_1, \dots, c_K)$, $\sum_{k=1}^K c_k = 1$. 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:

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

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.

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.

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 $X = (x_i)_{i=1}^{\ell}$, $x_i \in$. 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 $a :\to \{1, \ldots, K\}$, 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:
$$a(x_i) = \operatorname*{argmin}_{1 \leq k \leq K} \rho(x_i, c_k).$$

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 приближения. 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

$$X = U\Sigma V^T$$
$$\Sigma = diag\{\sigma_1, ..., \sigma_R\}, \sigma_1 \geq \sigma_2 \geq ... \geq \sigma_R > 0$$
$$U^TU = I, V^TV = I$$

. 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 | Постановка задачи обучения логистической регрессии.

16 | Постановка задачи SVM (Soft Hard margin).

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

$$a(x) = (\langle w, x\rangle + b), \qquad w \in^d, b \in .$$

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

$$\min_{x\in X} |\langle w, x\rangle + b| = 1. \tag{2}$$

Можно показать, что расстояние от произвольной точки $x_0 \in^d$ 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):

$$\begin{cases} \dfrac{1}{2}\|w\|^2 \to \min\limits_{w,b} \\ y_i\left(\langle w, x_i\rangle + b\right) \geq 1, \quad i = 1, \ldots, \ell. \end{cases} \tag{3}$$

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

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 $\xi_i \geq 0$ за их нарушение:

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

Отметим, что если отступ объекта лежит между нулем и единицей ($0 \leq y_i\left(\langle w, x_i\rangle + b\right) < 1$), 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)

$$\begin{cases} \dfrac{1}{2}\|w\|^2 + C\sum\limits_{i=1}^{\ell} \xi_i \to \min\limits_{w,b,\xi} \\ y_i\left(\langle w, x_i\rangle + b\right) \geq 1 - \xi_i, \quad i = 1, \ldots, \ell, \\ \xi_i \geq 0, \quad i = 1, \ldots, \ell. \end{cases} \tag{4}$$

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

Данная задача также является выпуклой и имеет единственное решение.

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:

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

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^TX 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: $\beta^* = (X^TX)^{-1}X^TY$, однако при вырожденности матрицы X^TX 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:

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

$$(X^TX + \lambda I)\beta = X^TY$$

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

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.

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 | 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:

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

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 | 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):

$$b_0(x) = \text{argmax}_{y\in} \sum_{i=1}^{\ell} [y_i = y]$$

- 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.

21 | Оценка качества 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:

$$\sum_{k=1}^{K}\sum_{i=1}^{\ell}[a(x_i)=k]\rho(x_i,c_k), \tag{5}$$

   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):

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

   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.

22 | Многослойный персептрон

22.1 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.*

22.2 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.

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

23.1 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.