Интеллектуальные системы

Виды данных. Задача машинного обучения. Классификация задач машинного обучения. Способы предобработки данных.

Определение

Задано множество объектов X, множество допустимых ответов Y , и существует целевая функция (target function) $y*: X \to Y$, значения которой $y_i = y*(x_i)$ известны только на конечном подмножестве объектов $\{x_1, \dots, x_n\} \subset X$. Пары «объект– ответ» (xi , yi) называются прецедентами. Совокупность пар $X^n = (x_i, y_i)_{i=1}^n$ называется обучающей выборкой (training sample).

Задача обучения по прецедентам заключается в том, чтобы по выборке X^n восстановить зависимость y $*$, то есть построить решающую функцию (decision function) a: $X \to Y$, которая приближала бы целевую функцию $y*(x)$, причем не только на объектах обучающей выборки, но и на всём множестве X.

Выделяют 2 основных типа классического ML:

- Обучение с учителем - это когда необходимо найти функциональную зависимость результатов от входов и построить алгоритм, на входе принимающий описание объекта и на выходе выдающий результат.
  Сюда относят задачи классификации, регрессии, ранжирования и прогнозирования.
- Без учителя - когда ответы не задаются, а нужно искать зависимости между объектами.
  Сюда относят задачи кластеризации, поиска ассоциативных правил, фильтрации выбросов, построения доверительных областей, сокращения размерности.

К неклассическим, но весьма популярным методам относят обучение с подкреплением, в частности, генетические алгоритмы, и искусственные нейронные сети. В качестве входных объектов выступают пары «ситуация, принятое решение», а ответами являются значения функционала качества, который характеризует правильность принятых решений (реакцию среды). Эти методы успешно применяются для формирования инвестиционных стратегий, автоматического управления технологическими процессами, самообучения роботов и других подобных задач.

Виды данных

1. *Числовые данные* - любые данные, где точки данных являются точными числами. Например, количество жилых объектов в городе, цены на жилье.
   Числовые данные можно охарактеризовать как непрерывные или дискретные данные. Непрерывные данные могут принимать любое

значение в пределах диапазона, тогда как дискретные данные имеют различные значения.

2. *Категориальные данные* - представляют собой характеристики. Например, позиция хоккеиста, команда, родной город. Категориальные данные могут принимать числовые значения. Например, возможно, мы бы использовали 1 для красного цвета и 2 для синего. Но эти цифры не имеют математического значения. То есть мы не можем сложить их или взять среднее значение.
   Существует также то, что называется порядковыми данными, которые в некотором смысле представляют собой смесь числовых и категориальных данных. В порядковых данных данные по-прежнему делятся на категории, но эти категории упорядочены или ранжированы определенным образом. Примером может служить классовая сложность, такая как начальный, средний и продвинутый уровень. Эти три типа классов были бы способом, которым мы могли бы маркировать классы, и они имеют естественный порядок в возрастающей сложности.
3. *Данные временного ряда* - представляют собой последовательность чисел, собираемых через регулярные интервалы в течение некоторого периода времени. Это очень важно, особенно в таких областях, как финансы. К данным временного ряда прикреплено временное значение, поэтому это будет что-то вроде даты или отметки времени, в которой вы можете искать тренды во времени.
   Например, мы можем измерить среднее количество продаж домов за многие годы. Разница данных временных рядов и числовых данных заключается в том, что вместо числовых рядов данных, которые не имеют какого-либо временного порядка, данные временных рядов имеют некоторый подразумеваемый порядок. Получена первая точка данных и последняя точка данных.
4. *Текстовые данные*.

Задачи машинного обучения

1. *Задача классификации* - требуется построить алгоритм, который по вектору признаков возвращает метку класса или вектор оценок принадлежности к каждому из классов.
   Примеры задач:
   a. медицинская диагностика – по набору признаков определить диагноз
   b. геологоразведка – по данным зондирования почв определить наличие полезных ископаемых
   c. оптическое распознавание текстов – определение цепочек символов на изображении
   d. кредитный скоринг - по анкете заемщика принять решение о выдаче или отказа в кредите
   e. синтез химических соединений - по параметрам химических элементов спрогнозировать свойства получаемого соединения

2. *Задача восстановления регрессии* - построение алгоритма (регрессора), который по вектору признаков возвращает точечную оценку значения регрессии, доверительный интервал или апостериорное распределение на множестве значений регрессионной переменной.
   Примеры задач:
   a. оценка стоимости недвижимости - по характеристикам района оценить стоимость жилья
   b. прогноз свойств соединений - по параметрам химических элементов спрогнозировать свойства их соединений
   c. медицина - по постоперационным показателям определить время заживления органа
   d. кредитный скоринг - по анкете заемщика определить величину кредитного лимита
   e. инженерное дело - по техническим характеристикам автомобиля и режиму езды спрогнозировать расход топлива
3. *Задача кластеризации* - построение алгоритма разбиения выборки на непересекающиеся группы. В каждый класс попадают объекты в некотором смысле похожие друг на друга.
   Примеры задач:
   a. экономическая география - по физико-географическим и экономическим показателям разбить страны мира на группы схожих по экономическому положению государств
   b. финансовая сфера - по сводкам банковских операций выявить группы “подозрительных”, нетипичных банков, сгруппировать остальные остальные по степени близости проводимой стратегии
   c. маркетинг - по результатам маркетинговых исследований среди множеств потребителей выделить характерные группы по степени интереса к продвигаемому продукта
   d. социология - по результатам социологических опросов выявить группы общественных проблем, вызывающих схожую реакцию у общества, а также характерные фокус-группы населения
4. *Задача идентификации* - построение алгоритма, который по вектору признаков определил бы наличие определенного свойства у объекта либо вернул бы оценку степени его выраженности.
   Примеры задач:
   a. медицинская диагностика - установление определенного заболевания по набору признаков
   b. системы безопасности - по камерам наблюдения в подъезде идентифицировать жильца подъезда
   c. банковское дело - определить подлинность подписи на чеке
   d. обработка изображений - выделение участков изображения лиц на фотографии
   e. искусствоведение - по характеристикам произведения определить, является ли автором тот или иной автор
5. *Задача прогнозирования* - построение алгоритма, возвращающего точечную оценку, доверительный интервал или апостериорное распределение прогноза на заданную глубину. В отличии от регрессии, осуществляется прогноз по

времени, а не по признакам.
Примеры задач:
   a. банковское дело - прогнозирование биржевых индексов и котировок
   b. системы управления - прогноз показателей работы реактора по данным телеметрии
   c. экономика - прогноз цен на недвижимость
   d. демография - прогноз изменения численности различных социальных групп в конкретном ареале
   e. гидрометеорология - прогноз геомагнитной активности
6. *Задача извлечения знаний* - построение алгоритма, генерирующего набор объективных закономерностей между признаками, имеющих место в генеральной совокупности. Закономерности обычно имеют форму предикатов и могут выражаться как в цифровых терминах, так и в текстовых.
   Пример задач:
   a. медицина - поиск взаимосвязей между различными показателями при фиксировании болезни
   b. социология - определение факторов, влияющих на победу на выборах
   c. научные исследования - получение новых знаний об исследуемом процессе
   d. биржевое дело - определение закономерностей между различными биржевыми показателями

Предобработка данных

Пропуски - очень часто случается так, что в наборе данных пропущены те или иные значения. Данные с пропусками чаще всего нельзя просто так взять передать в модель.

Пропуски в категориальных признаках можно:
- Удалить объект содержащий пропуск признака(часто такое удаление приводит к потере большого количества информации, что очень плохо).
- Заменить новой категорией “Неизвестно”.
- Заменить наиболее популярным значением (модой).

Пропуски в численных признаках можно:
- Заменить средним значением (если есть выбросы, среднее значение может быть сильно искажено).
- Заменить медианой. Если в данных присутствуют выбросы, этот способ замены является предпочтительным.

Выбросы - в данных могут присутствовать значения, являющиеся выбросами. Это, как правило, не ошибки. Однако, своими значениями они "шокируют" модель

Для того, чтобы определить, является ли значение выбросом, пользуются характеристикой выборки, называемой интерквартильным размахом. Определяется он следующим образом: $IQR = Q_3 - Q_1$, где Q_1 - первая квантиль (значение признака, меньше которого ровно 25% всех значений признака), Q_3- третья квантиль (значение, меньше которого ровно 75%). Для того, чтобы понять, является ли значение выбросом,

можно воспользоваться эвристикой: выбросы лежат за пределами следующего интервала:

$$[Q_1 - 1.5IQR,\ Q_3 + 1.5IQR]$$

Нормализация - приведение всех значений признака к новому диапазону. Например, к диапазону [0,1]. Это полезно, потому что значения признаков могут изменяться в очень большом диапазоне.
Выделяют L1 и L2 нормы:

- L1 норма (расстояние городских кварталов (манхэттенское)):

$$||x||_1 = \sum_i |x_i|$$

- L2 норма (Евклидова метрика):

$$||x||_2 = \left(\sum_{i=1}^{N} |x_i|^2\right)^{1/2} = \sqrt{x_1^2 + \ldots + x_N^2}$$

Стандартизация - приведение данных к распределению со средним значением 0 и стандартным отклонением 1. На практике наиболее распространены следующие методы стандартизации признаков:

- Минимакс - линейное преобразование данных в диапазоне [0, 1], где минимальное и максимальное масштабируемые значения соответствуют 0 и 1 соответственно

$$x_{new} = \frac{x_{old} - x_{min}}{x_{max} - x_{min}}$$

- Z - масштабирование данных на основе среднего значения и стандартного отклонения: деление разницы между переменной и средним значением на стандартное отклонение

$$z = \frac{x - \mu}{\sigma}$$

На практике минимакс и Z-масштабирование имеют похожие области применимости и часто взаимозаменяемы. Однако, при вычислении расстояний между точками или векторами в большинстве случае используется Z-масштабирование. А минимакс полезен для визуализации, например, чтобы перенести признаки, кодирующие цвет пикселя, в диапазон [0..255].

Бинаризация - метод, с помощью которого мы можем сделать наши данные двоичными. Мы можем использовать двоичный порог для того, чтобы сделать наши данные двоичными. Значения выше этого порогового значения будут преобразованы в 1, а ниже этого порогового значения будут преобразованы в 0.

One-hot-encoding - способ предварительной предобработки категориальных признаков. Многие модели плохо работают с категориальными признаками как

таковыми. Дело в том, что словосочетание "Русь Матушка" нельзя просто взять и умножить на какое-нибудь число. Но многие модели работают именно так: берется коэффициент и на него умножается значение признака. Аналогичная операция выполняется с остальными признаками. Все результаты суммируются. На основе значения суммы делается вывод о принадлежности объекта к тому или иному классу (такие модели называются линейными). Чтобы бороться с этой проблемой, был придуман способ преобразовать исходный признак в несколько новых, бинарных признаков.

Сравнение моделей

Модель можно выбрать из некоторого множества моделей, проверив результат работы каждой модели из множества с помощью ручного тестирования, но ручное тестирование серьезно ограничивает количество моделей, которые можно перебрать, а также требует больших трудозатрат. Поэтому в большинстве случаев используются алгоритмы, позволяющие автоматически выбирать модель.

Кросс-валидация - процедура эмпирического оценивания обобщающей способности алгоритмов. С помощью кросс-валидации эмулируется наличие тестовой выборки, которая не участвует в обучении, но для которой известны правильные ответы.

Принцип работы:

1. Обучающая выборка разбивается на k непересекающихся одинаковых по объему частей;
2. Производится k итераций. На каждой итерации происходит следующее:
    a. Модель обучается на k−1 части обучающей выборки;
    b. Модель тестируется на части обучающей выборки, которая не участвовала в обучении.

Каждая из k частей единожды используется для тестирования.

Мета-обучение - подход, позволяющий определять наиболее подходящий алгоритм (иногда, вместе с параметрами к нему) для конкретной задачи из портфолио алгоритмов. Основная идея мета-обучения — свести задачу выбора алгоритма к задаче обучения с учителем: задачи описываются мета-признаками. Мета-признак описывает свойство задачи — например, разрежен ли датасет или нет, число категориальных или численных признаков объектов в датасете, число возможных меток, размер датасета и многое другое. От хорошей модели ожидается высокая адаптируемость к новым задачам и окружениям, на небольшом количестве примеров.

Разложение ошибки на смещение и разброс (bias-variance decomposition). Кросс-валидация

Источник - Хабр

Bias-variance

Рассуждаем на примере линейной регрессии.
Факты:

- истинное значение целевой переменной складывается из некоторой детерминированной функции $f(x)$ и случайной ошибки ϵ: $y = f(x) + \epsilon$
- ошибка распределена нормально с центром в нуле и некоторым разбросом: $\epsilon \sim N(0, \sigma^2)$
- истинное значение целевой переменной тоже распределено нормально: $y \sim N(f(x), \sigma^2)$
- мы пытаемся приблизить детерминированную, но неизвестную функцию $f(x)$ линейной функцией от регрессоров $\hat{f}(x)$ которая, в свою очередь, является точечной оценкой функции f в пространстве функций (точнее, мы ограничили пространство функций параметрическим семейством линейных функций), т.е. случайной переменной, у которой есть среднее значение и дисперсия.

  Ошибка в точке x раскладывается следующим образом:

$$Err(x) = E[(y - \hat{f}(x))^2] = E[y^2] + E[(\hat{f}(x))^2] - 2E[y\hat{f}(x)] = E[y^2] + E[\hat{f}^2] - 2E[y\hat{f}]$$

Отсюда видим, что по формуле $Var(z) = E[z^2] - E[z]^2$ можно расписать первые два члена:

$$E[y^2] = Var(y) + E[y]^2 = \sigma^2 + f^2$$
$$E[\hat{f}^2] = Var(\hat{f}) + E[\hat{f}]^2.$$

Пояснение:

$$Var(y) = E[(y - E[y])^2] = E[(y - f)^2] = E[(f + \epsilon - f)^2] = E[\epsilon^2] = \sigma^2 \text{ а}$$
$$E[y] = E[f + \epsilon] = E[f] + E[\epsilon] = f$$

Последний член суммы:

$$E[y\hat{f}] = E[(f + \epsilon)\hat{f}] = E[f\hat{f}] + E[\epsilon\hat{f}] = fE[\hat{f}] + E[\epsilon]E[\hat{f}] = fE[\hat{f}]$$

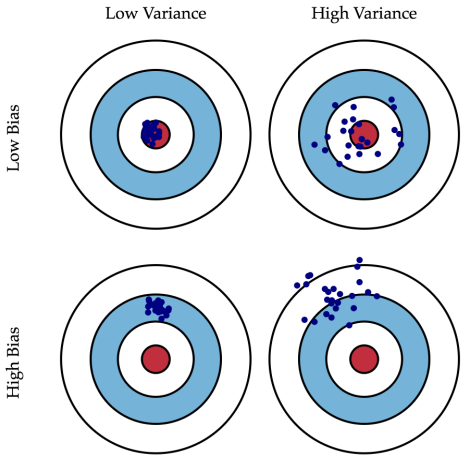
Собираем всё вместе:

$$Err(x) = E[(y - \hat{f}(x))^2] = \sigma^2 + f^2 + Var(\hat{f}) + E[\hat{f}]^2 - 2fE[\hat{f}] = (f - E[\hat{f}])^2 + Var(\hat{f}) + \sigma^2$$
$$= Bias(\hat{f})^2 + Var(\hat{f}) + \sigma^2, \text{ где}$$

$Bias(\hat{f})$ - средняя ошибка по всевозможным наборам данных,

$Var(\hat{f})$ - вариативность ошибки - то, на сколько ошибка будет отличаться, если обучать модель на разных наборах данных,

σ^2 - неустранимая ошибка.



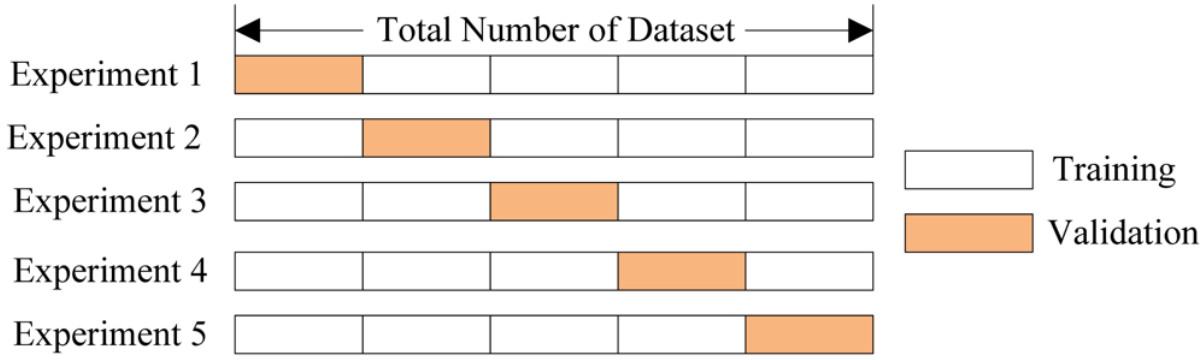
Теорема Маркова-Гаусса утверждает, что МНК-оценка параметров линейной модели является самой лучшей в классе несмещенных линейных оценок, то есть с наименьшей дисперсией. Это значит, что если существует какая-либо другая несмещенная модель *g* тоже из класса линейных моделей, то мы можем быть уверены, что $Var(\widehat{f}) \leq Var(g)$.

Кросс-валидация

Главная задача обучаемых алгоритмов – их способность обобщаться, то есть хорошо работать на новых данных. Поскольку на новых данных мы сразу не можем проверить качество построенной модели (нам ведь надо для них сделать прогноз, то есть истинных значений целевого признака мы для них не знаем), то надо пожертвовать небольшой порцией данных, чтоб на ней проверить качество модели.

Чаще всего это делается одним из 2 способов:

- отложенная выборка (held-out/hold-out set). При таком подходе мы оставляем какую-то долю обучающей выборки (как правило от 20% до 40%), обучаем модель на остальных данных (60-80% исходной выборки) и считаем некоторую метрику качества модели (например, самое простое – долю правильных ответов в задаче классификации) на отложенной выборке.
- кросс-валидация. Тут самый частый случай – K-fold кросс-валидация.



Тут модель обучается K раз на разных (K-1) подвыборках исходной выборки (белый цвет), а проверяется на одной подвыборке (каждый раз на разной, оранжевый

цвет). Получаются K оценок качества модели, которые обычно усредняются, выдавая среднюю оценку качества классификации/регрессии на кросс-валидации.

Кросс-валидация дает лучшую по сравнению с отложенной выборкой оценку качества модели на новых данных. Но кросс-валидация вычислительно дорогостоящая, если данных много.

Кросс-валидация – очень важная техника в машинном обучении (применяемая также в статистике и эконометрике), с ее помощью выбираются гиперпараметры моделей, сравниваются модели между собой, оценивается полезность новых признаков в задаче и т.д.

Подготовка данных. Методы обработки числовых, категориальных, текстовых данных.

Преобразование данных в основном ***необходимо*** для следующего:

- Неизбежное преобразование - преобразование данных чтобы соответствовать требованиям на вход модели. Например преобразование нечисловых признаков в числовые, чтобы работали матричные операции. Или сжатие/расширение изображения.
- Дополнительные качественные преобразования - повышают производительность модели. Например токенизация, приведение к нижнему регистру, нормализация и разрешение линейным моделям вводить нелинейности в пространство признаков.

Преобразования имеют место перед обучением или внутри самой модели.

Подготовка данных включает следующие ***этапы***:

- *Извлечение признаков* – анализ первичных данных с целью выделения признаков, превращение данных, специфических для предметной области, в понятные для модели векторы;
- *Очистка и преобразование данных* – имеющиеся у вас данные могут иметь неправильный формат или могут потребовать преобразования, чтобы сделать их более полезными: удаление опечаток в текстовых значениях, некорректных значений (например, число в строковом параметре и пр.), отсутствующих значений, исключение дублей и разных описаний одного и того же объекта, восстановление уникальности, целостности и логических связей, исправляются шумы и выбросы. Для числовых переменных применяется нормализация данных, чтобы привести их к одинаковой области изменения и использовать их вместе в одной модели;
- *Выборка данных или признаков* – отбор признаков и объектов с учетом их релевантности, качества и технических ограничений (объема и типа), для понимания характера записей (строк), а также смысла, типа и диапазона значений признаков (столбцов), затем формируется выборку – отобранные данные, которые потенциально имеют отношение к проверяемой гипотезе машинного обучения, используются методы главных компонент и ридж-регрессия и др.;

- *Генерация признаков* – создание производных признаков и их преобразование в векторы для модели Machine Learning, а также трансформация для повышения точности алгоритмов машинного обучения;
- *Интеграция (объединение) данных* – слияние данных из различных источников, которые содержат различную информацию об одних и тех же объектах, включая их агрегацию, когда новые значения вычисляются путем суммирования информации из множества существующих записей;
- *Форматирование* – синтаксические изменения, которые не меняют значение данных, но требуются для инструментов моделирования, например, сортировка в определенном порядке или удаление ненужных знаков препинания в текстовых полях, обрезка «длинных» слов, округление вещественных чисел до целого и т.д., а также переформатирование данных, заключающееся в изменении порядка атрибутов, когда алгоритм требует для работы их специфический порядок.

Очистка данных

1) Работа с пропущенными значениями.
   Большинство моделей машинного обучения требуют, чтобы все функции были полными, поэтому необходимо учитывать пропущенные значения.
   Мы можем:
   - удалить объект, содержащий пропуски (плохо когда у нас мало данных, или во временных рядах мы в принципе этого сделать не можем, так как потеряем зависимости)
   - заменить на наиболее встречающееся значение признака
   - заменить средним значением(сильно изменяется, если есть выбросы), медианой (норм если есть выбросы), модой
   - категориальные признаки можно заменить новой категорией “Неизвестно”
2) Работа с выбросами.
   При обнаружении выбросов их можно просто удалить.

   Если вы считаете, что при удалении или изменении выбросов важная информация будет утеряна, вы можете сохранить их и использовать алгоритм машинного обучения и оптимизационную метрику, устойчивую к выбросам.

   Более сложные методы включают Winsorizing - замену экстремальных значений минимальным и максимальным процентилями - и дискретизацию (биннинг) - разделение непрерывной переменной на дискретные группы.
   - Винсоризация — это серия трансформаций, направленных на ограничения влияния выбросов. 90%-ая винсоризация означает, что мы берём значения меньше 5% перцентиля и выше 95% перцентиля и приравниваем их к значениям на 5-м и 95-м перцентилях соответствиино.
   - Тримминг отличается от винсоризации тем, что мы не ограничиваем крайние значения каким-либо числом, а просто удаляем их.
   - Статистические тесты.
   - Итерационные методы.
   - Метрические методы.
   - Методы машинного обучения (Метод опорных векторов для одного класса (OneClassSVM), Изолирующий лес (IsolationForest)).

3) Исправление опечаток.
4) Удаление дубликатов.
5) Группировка разреженных классов (Grouping sparse classes).

Категориальные признаки могут часто иметь большое количество различных значений, некоторые из которых имеют низкую частоту. Это может быть особенно распространено, когда данные были введены в виде произвольного текста и подвержены опечаткам.

На этапе преобразования данных мы обсудим, как категориальные данные преобразуются в формат, который может прочитать модель машинного обучения. Однако это часто включает создание новой функции для каждого отдельного значения в этой категории; если каждая категориальная особенность имеет много различных значений, это преобразование приводит к множеству дополнительных функций.

Многие алгоритмы машинного обучения могут бороться со слишком многими функциями, это называется проклятием размерности. (*Проклятие размерности* — проблема, связанная с экспоненциальным возрастанием количества данных из-за увеличения размерности пространства). Чтобы решить эту проблему, вы можете сгруппировать качественно похожие значения. Это может быть ручная работа с экспертом в данной области. Хотя для опечаток или несоответствий в заглавных буквах могут использоваться инструменты шаблонного или нечеткого сопоставления.

6) Нормализация - позволяет масштабировать числовые значения в указанный диапазон.
7) Масштабирование (scaling to a range).

Преобразование к распределению с матожиданием 0 и стандартным отклонением 1.

8) Отсечение(clipping).

Если ваш набор данных содержит экстремальные выбросы, вы можете попробовать отсечение функций, которое ограничивает все значения функций выше (или ниже) определенного значения фиксированным значением. Например, вы можете обрезать все значения температуры выше 40 до ровно 40.

Вы можете применить отсечение функций до или после других нормализаций.

Еще одна простая стратегия отсечения - отсечение по z-баллу до + -Nσ (например, ограничение до + -3σ). Обратите внимание, что σ - стандартное отклонение.

9) Логарифмирование (log scaling).

Логарифмирование вычисляет логарифм ваших значений, чтобы сжимать широкий диапазон до узкого диапазона.

10) Трансформация Бокса-Кокса.

Является одним из вариантов стабилизации дисперсии.

Для исходной последовательности $y = \{y_1, \ldots, y_n\},\ y_i > 0,\ i = 1, \ldots, n$ однопараметрическое преобразование Бокса-Кокса с параметром λ определяется следующим образом:

$$y_i^\lambda = \begin{cases} \frac{y_i^\lambda - 1}{\lambda}, & amp; \text{if } \lambda \neq 0, \\ \log(y_i), & amp; \text{if } \lambda = 0. \end{cases}$$

Параметр λ можно выбирать, максимизируя логарифм правдоподобия.

11) Работа с непрерывными данными.

Монотонное преобразование признаков критично для одних алгоритмов и не оказывает влияния на другие.

- Квантование.
  В обработке сигналов — разбиение диапазона отсчетных значений сигнала на конечное число уровней и округление этих значений до одного из двух ближайших к ним уровней.
- Бинаризация.
  Всё просто — выбираем порог, значениям ниже которого присваивается одно значение, а выше — другое.
- Биннинг, он же бакетинг.
  Это техника обработки данных, при которой исходный набор данных делится на некоторое количество интервалов, общее значение которых (обычно среднее) затем представляет все значения интервала, что позволяет уменьшить влияние отдельных наблюдений. Иными словами мы переводим непрерывную величину в дискретную. Техника бининга используется для построения гистограмм.
  Ключевым вопросом в бининге является решение относительно длины интервалов. Интервалы могут быть фиксированной или адаптивной длины.
- Квантилизация.
  Квантилизация — это адаптивный бининг, основанный на распределении данных. Если в обычном биннинге могут быть пустые интервалы, то здесь — нет.

12) One-hot encoding.

Преобразование категориальных переменных.

Когда не хотят заполонять признаковую матрицу кучей бинарных признаков, применяют кодировки, в которых категории кодируются какими-то интерпретируемыми значениями. Например, если это категория товаров в интернет-магазине, то логично её закодировать средней ценой товара. Тогда, по крайней мере, наш новый признак упорядочивает категории по дороговизне.

Самый примитивный способ кодирования — заменить каждую категорию числом входящих в неё объектов (т.е. знания других признаков вообще не нужно).

One-hot-кодировки, плодящие признаки, подходят для решения задачи линейными методами, но не подходят для многих других, например для решающих деревьев и бустинговых схем над деревьями по причине слишком большого числа признаков после кодирования. Можно предложить другие способы кодировки, которые позволят применять описанные выше методы.

Иногда используют случайные кодировки: для каждого признака на множество его значений действуют случайной перестановкой. После этого строится дерево решений (или небольшой случайный лес). Затем процедура

повторяется. Естественно, эти же перестановки действуют и на описания новых объектов (из контроля). Ответы деревьев (лесов) усредняют по всем кодировкам.

Выборка признаков и данных

Существуют различные методы, с помощью которых можно уменьшить размер данных для упрощения их обработки. В зависимости от размера данных и домена можно применить такие методы:

- Выборка записей: создание выборки записей данных и выбор репрезентативного подмножества из общего набора данных.
- Выборка признаков: процедура отбрасывания незначащих переменных из очищенной выборки перед запуском машинного обучения и интеллектуального анализа данных.

Правильный отбор признаков для анализа данных позволяет:

- повысить качество моделей машинного обучения с учителем и без,
- уменьшить время обучения и снизить требуемые вычислительные мощности,
- в случае входных данных высокой размерности позволяет ослабить «проклятие размерности».

Методы с учителем:

- Relief: Этот метод случайным образом выбирает из датасета образцы и обновляет значимость каждого признака на основе разницы между выбранным экземпляром и двумя ближайшими к нему объектами того же и противоположного классов. Если наблюдается разница в значениях признака для двух ближайших соседей одного класса, его важность снижается, а если, наоборот, наблюдается различие между значениями признака для объектов разных классов, важность, соответственно, повышается.

$$W_i = W_i - (x_i - nearHit_i)^2 + (x_i - nearMiss_i)^2$$

Вес признака уменьшается, если его значение отличается для ближайших объектов одного и того же класса больше, чем для ближайших объектов из разных классов; в противном случае вес увеличивается.
Расширенный алгоритм ReliefF использует взвешивание признаков и ищет по большему количеству ближайших соседей.

- Критерий Фишера
- Обычно используется в задачах бинарной классификации. Отношение Фишера (Fisher ratio, FiR) определяется как расстояние между средними значениями признаков для каждого класса, деленное на их дисперсии:

$$FiR_i = \frac{\left|\bar{X}_i^{(0)} - \bar{X}_i^{(1)}\right|}{\sqrt{var(X_i)^{(0)} + var(X_i)^{(1)}}}$$

- Критерий хи-квадрат
  Проверяет, есть ли значимая разница между наблюдаемой и ожидаемой частотами двух категориальных переменных. Таким образом, проверяется нулевая гипотеза об отсутствии связи между двумя переменными.

Большие значения хи-квадрат указывают на то, что наблюдаемые и ожидаемые частоты находятся далеко друг от друга. Малые значения хи-квадрат означают обратное: наблюдаемые близки к ожидаемым. То есть хи-квадрат дает меру расстояния между наблюдаемой и ожидаемой частотами.

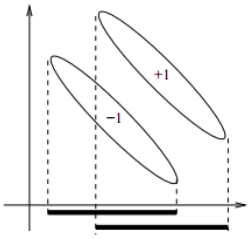
$$X^2 = \frac{(\text{Observed frequency} - \text{Expected frequency})^2}{\text{Expected frequency}}$$

Чтобы корректно применять критерий хи-квадрат для проверки связи между разными признаками из датасета и целевой переменной, необходимо соблюсти условия: переменные должны быть категориальными, независимыми и должны иметь ожидаемую частоту больше 5. Последнее условие гарантирует, что CDF (cumulative density function) статистического критерия (test statistic) может быть аппроксимирован с помощью распределения хи-квадрат.

- F-мера
  Суть метода в том, что он измеряет различие двух наборов действительных чисел. Пусть у нас дан вектор тренировочных данных xk, где содержатся как числа из положительного датасета, так и из отрицательного (например, болен - не болен). Набор положительных чисел обозначим как n+, отрицательных - n-. Тогда формула F-меры будет выглядеть следующим образом:

$$F\text{-score}(i) = \frac{\left(\overline{x}_i^{(+)} - \overline{x}_i\right)^2 + \left(\overline{x}_i^{(-)} - \overline{x}_i\right)^2}{\frac{1}{n^+ - 1}\sum_{k=1}^{n^+}\left(\overline{x}_{k,i}^{(+)} - \overline{x}_i^{(+)}\right)^2 + \frac{1}{n^- - 1}\sum_{k=1}^{n^-}\left(\overline{x}_{k,i}^{(-)} - \overline{x}_i^{(-)}\right)^2}$$

  Недостатком F-меры является то, что она не раскрывает взаимную информацию о характеристиках. Рассмотрим один простой пример (картинка):



  .

  Обе характеристики этих данных имеют низкую F-меру, поскольку в формуле знаменатель (сумма дисперсий положительного и отрицательного множеств) намного больше, чем числитель.
  Другими словами, F-мера показывает отличительную способность каждой функции независимо от других. Одна оценка вычисляется для первого признака, а другая оценка - для второго признака. Но это ничего не говорит о сочетании обеих функций (взаимная информация).
- T-статистика
  Предполагая, что это проблема бинарной классификации, где каждую выборку можно классифицировать либо в класс C1, либо в класс C2, t-статистика помогает нам оценить, значительно ли отличаются значения определенного объекта для класса C1 от значений того же признака для класса C2. Если это так, то эта функция может помочь нам лучше дифференцировать наши данные.

$$t(X_i) = \frac{|\mu_{i1} - \mu_{i2}|}{\sqrt{\frac{\sigma_{i1}^2}{n_1} + \frac{\sigma_{i2}^2}{n_2}}}$$

T-статистика вычисляется следующим образом: . После вычисления значений t-статистики для каждого признака мы сортируем эти значения в порядке убывания, чтобы выбрать наиболее важные признаки.

- Correlation-based feature selection
  Признаки релевантны, если их значения систематически меняются в зависимости от принадлежности к той или иной категории. Таким образом, хорошее подмножество признаков содержит такие признаки, которые высоко коррелируют с целевой переменной, и при этом не коррелируют друг с другом. Оценка подмножества из k признаков вычисляется так:

$$Merit_{S_k} = \frac{kr_{cf}}{\sqrt{k + k(k-1)\bar{r}_{ff}}}$$

Здесь r_{cf} — это среднее значение всех корреляций между признаком и классом, а $\bar{r}_{ff}$ — среднее значение всех корреляций между признаками. Критерий CFS определяется так:

$$CFS = \max_{S_k} \left[\frac{r_{cf_1} + r_{cf_2} + \cdots + r_{cf_k}}{\sqrt{k + 2(r_{f_1f_2} + \cdots + r_{f_if_j} + \cdots + r_{f_kf_1})}} \right]$$

Методы без учителя:

- Дисперсия - было показано, что оценка дисперсии признака может быть эффективным способом отбора признаков. Как правило признаки с почти нулевой дисперсией не являются значимыми, и их можно удалить.
- Средняя абсолютная разность - вычисляем среднюю абсолютную разность между значениями признака и его средним значением. Более высокие значения, как правило, имеют более высокую предсказательную силу.

$$MAD_i = \frac{1}{n} \sum_{j=1}^{n} |X_{ij} - \bar{X}_i|$$

- Соотношение дисперсий - среднее арифметическое, деленное на среднее геометрическое. Более высокая дисперсия соответствует более релевантным признакам (реализация).

$$AM_i = \bar{X}_i = \frac{1}{n} \sum_{j=1}^{n} X_{ij} \qquad GM_i = (\prod_{j=1}^{n} X_{ij})$$

Поскольку $AM_i \geqslant GM_i$, если и только если соблюдается равенство $X_{i1} = X_{i2} = \cdots = X_{in}$, тогда:

$$R_i = \frac{AM_i}{GM_i} \in [1, +\infty)$$

- Критерий Лапласа (Laplacian Score) - в его основе лежит наблюдение, что данные из одного класса часто расположены ближе друг к другу, поэтому можно

оценить важность признака по его способности отражать эту близость. Метод состоит из встраивания данных в граф ближайших соседей с помощью измерения произвольного расстояния с последующим вычислением матрицы весов. Затем для каждого признака вычисляем критерий Лапласа и получаем такое свойство, что наименьшие значения соответствуют самым важным размерностям. Однако на практике при отборе подмножества признаков обычно применяется другой алгоритм кластеризации (метод k-средних), с помощью которого выбирается самая эффективная группа.

- Критерий Лапласа в сочетании с энтропией на основе расстояния - в основе алгоритма лежит критерий Лапласа, где кластеризация методом k-средних заменяется на энтропию. Алгоритм демонстрирует более высокую стабильность на датасетах высокой размерности (реализация).
- MCFS (Multi-Cluster Feature selection, многокластерный отбор признаков) - для измерения корреляции между разными признаками выполняется спектральный анализ. Для кластеризации данных и оценки признаков используются собственные вектора оператора Лапласа(graph Laplacian). Их вычисление описывается в этой работе.

Извлечение признаков

Можно обнаружить, что столбцы в данных бесполезны в их текущем состоянии, возможно, потому что они слишком гранулированы. Например, временная метка вряд ли будет полезна для поиска тенденций, в то время как время дня или день недели могут быть.

Обработка текстовых данных

Токенизация

Это разбиение текста на токены – в самом простом случае это просто слова. Но, делая это слишком простой регуляркой ("в лоб"), мы можем потерять часть смысла: "Нижний Новгород" это не два токена, а один. Зато призыв "воруй-убивай!" можно напрасно разделить на два токена. Существуют готовые токенайзеры, которые учитывают особенности языка, но и они могут ошибаться, особенно если вы работаете со специфическими текстами (профессиональная лексика, жаргонизмы, опечатки).

Стемминг и лемматизация

Лемматизация и стемминг преследуют цель привести все встречающиеся формы слова к одной, нормальной словарной форме.

Стемминг – это грубый эвристический процесс, который отрезает «лишнее» от корня слов, часто это приводит к потере словообразовательных суффиксов. Действует без знания контекста.

Лемматизация – это более тонкий процесс, который использует словарь и морфологический анализ, чтобы в итоге привести слово к его канонической форме – лемме.

При работе с текстами важен порядок слов - "i have no cows" и "no, i have cows" могут быть идентичными после векторизации, хотя и противоположны семантически.

TF-IDF

Это статистическая мера, используемая для оценки важности слова в контексте документа, являющегося частью коллекции документов или корпуса. Вес некоторого слова пропорционален частоте употребления этого слова в документе и обратно пропорционален частоте употребления слова во всех документах коллекции.

TF (term frequency — частота слова) — отношение числа вхождений некоторого слова к общему числу слов документа. Таким образом, оценивается важность слова ti в пределах отдельного документа.

$$tf(t,d) = \frac{n_t}{\sum_k n_k}$$

IDF— инверсия частоты, с которой некоторое слово встречается в документах коллекции.

$$idf(t,D) = log\frac{|D|}{|\{d_i \in D | t \in d_i\}|}$$

где $|D|$ - число документов в коллекции, $|\{d_i \in D | t \in d_i\}|$ - число документов из коллекции D, в которых встречается t.

Таким образом $tf - idf(t,d,D) = tf(t,d) \times idf(t,D)$

Несбалансированные данные. Over/under sampling.

Источник - Дьяконов

Данные несбалансированы, если в обучающей выборке доли объектов разных классов существенно различаются.

При наличия дисбаланса классов необходимо:

- Установить природу дисбаланса (наличие дубликатов, объем и структура самих данных) - иногда имеет смысл решать задачу детектирования аномалий.
- Уточнить функцию ошибки - есть функции ошибки, при которых перебплпнсировка классов недопустима (LogLoss). Есть и допускающие перебалансировку классов (ROC AUC).
- Уточнить контекст задачи. Возможно дисбаланс - естественное свойство данных. Возможно необходимо изменить пайплайн решения, например используя статифицированный контроль.
- Необходимо понять, на сколько хорошо модели в данной задаче могут быть откалиброваны, какая геометрия данных и распределение классов.

Перебалансировка данных

Undersampling

Делаем классы сбалансированными, заменяя большой класс подвыборкой по мощности равной малому классу.

Простейшая стратегия - взять случайную подвыборку.

Oversampling

Увеличение в размерах малого класса.
Простейшая стратегия - продублировать объекты малого класса.

У пересэмплирования качество, как правило, выше, а недосэмплирование позволяет учить модель на маленькой выборке.

Умное недосэмплирование

Основная идея - брать из большого класса только объекты важные для решения рассматриваемой задачи.

Алгоритмы:

- Nearmiss 1 - из большего класса выбираем объекты, у которых среднее расстояние до *N* ближайших объектов малого класса наименьшее, т.е. из граничной зоны.
- Nearmiss 2 - из большего класса выбираем объекты, у которых среднее расстояний до *N* дальних малого класса наименьшее.
- Tomek links - удалить объекты большого класса, образующие связи Томека (объекты двух разных классов образуют связь Томека, если нет объекта, который ближе к одному из них при этом являясь объектом другого класса).
- ENN - сделать скользящий контроль, например по 10 фолдам, удалить объекты большого класса, на которых ближайший сосед ошибается.

В последних двух методах не гарантируется выравнивание классов по мощности.

Synthetic Minority Oversampling Techniques (SMOTE)

Основная идея - увеличить малый класс за счёт представителей выпуклых комбинаций пар. Для точки малого класса выбирается k ближайших соседей (тоже малого класса) и на отрезке между ними случайно выбирается новый объект.

Adaptive Synthetic (ADASYN)

Основная идея - для точки малого класса выбирается k ближайших соседей (тоже малого класса) и на отрезке между строятся объекты в количестве пропорциональном числу объектов большего класса в окрестностях выбранной точки.

Веса классов

Можно также задавать объектам каждого класса определённые веса, которые будут влиять на функцию ошибки при обучении.

Пороговая функция

В задаче классификации можно учитывать дисбаланс классов и выбирать пороговую функцию так, чтобы максимизировать функционал качества.

Метод уменьшения размерности PCA.

Метод главных компонент линейно преобразовывает данные в новые признака, не коррелирующие друг с другом. Для задач машинного обучения PCA, использованный в качестве метода извлечения признаков может раскрыть свой потенциал лучше чем в задачах снижения размерности.

Пусть у нас есть матрица признаков A, размера $m \times n$. Мы хотим преобразовать её в новую матрицу G, размера $m \times l$. При этом исходные признаки должны восстанавливаться по новым описаниям с помощью некоторого линейного преобразования: $\widehat{A} = GV^T$, где $\widehat{A}$ - матрица восстановленных исходных признаков (при этом она не обязана в точности совпадать с исходной матрицей A). При поиске матриц G и V минимизируется выражение:

$\Delta^2(G,V) = \sum_{i=1}^{m} ||\widehat{x}_i - x_i||^2 = \sum_{i=1}^{m} ||z_i V^T - x_i||^2 = ||GV^T - A||^2 \to min_{G,V}$, где $\widehat{x}_i$ - вектор матрицы $\widehat{A}$, x_i - вектор матрицы A, z_i - вектор матрицы G.

Теорема*:* Если $l \leq rankA$, то минимум $\Delta^2(G,V)$ достигается, когда столбцы матрицы V есть собственные векторы A^TA, соответствующие l максимальным собственным значениям. При этом $G = AV$, а матрицы V и G ортогональны.

Собственные векторы v_1,..., v_l, отвечающие максимальным собственным значениям, называют *главными компонентами*.

Свойства метода главных компонент:

- Если $l = n$, то $\Delta^2(G,V) = 0$. В этом случае $A = GV^T$ является точным и совпадает с сингулярным разложением: $A = GV^T = USV^T$, если положить $G = US$, $S = \sqrt{\Lambda}$ и Λ - диагональная матрица, состоящая из квадратных корней собственных чисел матрицы A^TA (или AA^T - они равны).
- Если $l < n$, то представление $A \approx GV^T$ является приближенным. Сингулярное разложение матрицы GV^T получается из сингулярного разложения матрицы A путём отбрасывания (обнуления) $n - l$ минимальных собственных значений.
- Главные компоненты содержат основную информацию о матрице A. Число главных компонент l называют *эффективной размерностью* задачи. На практике собственные значения матрицы A^TA упорядочивают по убыванию, задают пороговое значение $\varepsilon \in [0,1]$, достаточно близкое к 0 и определяется наименьшее целое l, при котором относительная погрешность приближения матрицы A не превышает ε: $E(l) = \frac{||GV^T - A||^2}{||A||^2} = \frac{\lambda_{l+1} + ... + \lambda_n}{\lambda_1 + ... + \lambda_n} \leq \varepsilon$. $E(l)$ показывает, какая доля информации теряется при замене исходных признаковых описаний длины n на более короткие описания длины l. PCA наиболее эффективен в тех случаях, когда $E(l)$ оказывается малым даже при малых значениях l.

SVD-разложение, снижение размерности. Latent Semantic Analysis.

Источник - Медиум

SVD-разложение (Singular Value Decomposition) - сингулярное разложение.

Любую матрицу *A*, размерности $m \times n$, можно представить в виде произведения матриц USV^T , где:

- U - ортогональная матрица размерности $m \times m$, состоящая из собственных векторов матрицы AA^T. Порядок собственных векторов задан порядком корней соответствующих собственных чисел, расположенных на главной диагонали матрицы S.
- V - ортогональная матрица размерности $n \times n$, состоящая из собственных векторов матрицы A^TA. Порядок собственных векторов задан порядком корней соответствующих собственных чисел, расположенных на главной диагонали матрицы S.
- S - диагональная матрица, содержащая $r = rank(A)$ значений на диагонали. Эти значения равны корням собственных чисел матриц AA^T и A^TA, которые равны друг другу.

Собственные числа на диагонали матрицы S идут в порядке убывания.

$$\underset{m \times n}{A} = \underset{m \times m}{U} \underset{m \times n}{S} \underset{n \times n}{V^T}$$

$$\begin{pmatrix} x_{11} & x_{12} & & x_{1n} \\ & & \ddots & \\ x_{m1} & & & x_{mn} \end{pmatrix} = \begin{pmatrix} u_{11} & & u_{m1} \\ & \ddots & \\ u_{1m} & & u_{mm} \end{pmatrix} \begin{pmatrix} \sigma_1 & \cdots & 0 \\ & \sigma_r & \\ 0 & & 0 \end{pmatrix} \begin{pmatrix} v_{11} & & v_{1n} \\ & \ddots & \\ v_{n1} & & v_{nn} \end{pmatrix}$$

Стоит сказать, что AA^T и A^TA - симметричные квадратные матрицы ранга r. Они имеют одинаковые собственные числа, большие или равные 0. Ввиду симметричности матриц, можно выбрать собственные векторы так, чтобы они были ортогональными. Собственные векторы этих матриц называются *сингулярными векторами* матрицы A, а соответствующие собственные числа, а точнее корни из этих собственных чисел - *сингулярными значениями*.

Из SVD разложения можно вывести следующее: $AV = US$
Далее получаем:

$$\underset{m \times n}{\begin{pmatrix} x_{11} & x_{12} & & x_{1n} \\ & & \ddots & \\ x_{m1} & & & x_{mn} \end{pmatrix}} \underset{n \times r}{\begin{pmatrix} v_{11} & & v_{r1} \\ & \ddots & \\ v_{1n} & & v_{rn} \end{pmatrix}} = \underset{m \times r}{\begin{pmatrix} u_{11} & & u_{r1} \\ & \ddots & \\ u_{1m} & & u_{rm} \end{pmatrix}} \underset{r \times r}{\begin{pmatrix} \sigma_1 & & 0 \\ & \ddots & \\ 0 & & \sigma_r \end{pmatrix}}$$

$$A \begin{pmatrix} v_{11} \\ \vdots \\ v_{1n} \end{pmatrix} = \begin{pmatrix} u_{11} & & u_{r1} \\ & \ddots & \\ u_{1m} & & u_{rm} \end{pmatrix} \begin{pmatrix} \sigma_1 \\ \vdots \\ 0 \end{pmatrix}$$

$$= \begin{pmatrix} u_{11} \\ \vdots \\ u_{1m} \end{pmatrix} \sigma_1$$

$$A v_1 = \sigma_1 u_1$$

, где u_1 и v_1 первые столбцы матриц U и V соответственно (то есть собственные векторы матриц AA^T и(или) A^TA), а σ_1 первый элемент на диагонали матрицы S (то есть корень из собственного вектора матриц AA^T и(или) A^TA).

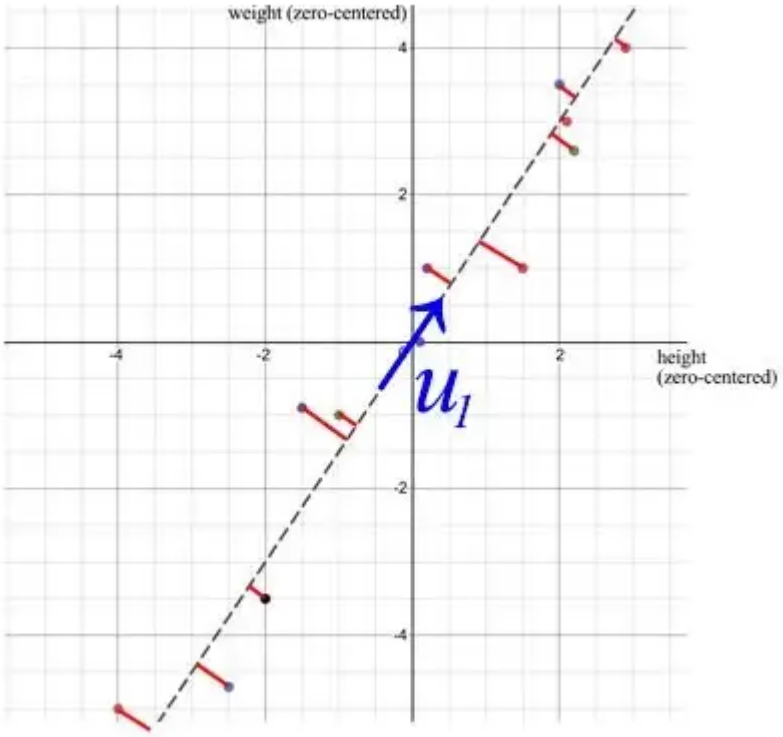
Обобщим: $Au_i = \sigma_i u_i$. Отсюда получаем разложение $A = \sum_{i=1}^{r} \sigma_i u_i v_i^T$.

Если в качестве матрицы A рассматривать матрицу признаков объектов выборки, то матрица $\frac{AA^T}{n-1}$ - матрица ковариации. Если после разложения матрицы ковариации получили первые сингулярные числа существенно больше последующих, то из разложения $A = \sum_{i=1}^{r} \sigma_i u_i v_i^T$ можно отбросить члены, соответствующие этим самым последующим сингулярным числам и обучать модель только на оставшейся матрице.

Также стоит отметить, что:

- Общая дисперсия данных равна следу выборочной ковариационной матрицы S, которая равна сумме квадратов сингулярных значений S. Имея это, мы можем рассчитать коэффициент потерянной дисперсии, если мы отбросим меньшие члены σ_i. Это отражает количество информации, утерянной после удаления.

- Первый собственный вектор u_1 указывает на наиболее важное направление внутри данных:



- Ошибка, рассчитанная как сумма перпендикулярных квадратных расстояний от точки до вектора u_1, является минимальной при использовании SVD.

Технически SVD извлекает данные в направлениях с наибольшей дисперсией (наиболее информативные) соответственно.

Для снижения размерности необходимо получить SVD разложение и взять матрицу US. Эта матрица и будет искомое описание признаков в пониженном пространстве. Матрица V^T переводит US обратно в исходное пространство.

Latent Semantic Analysis

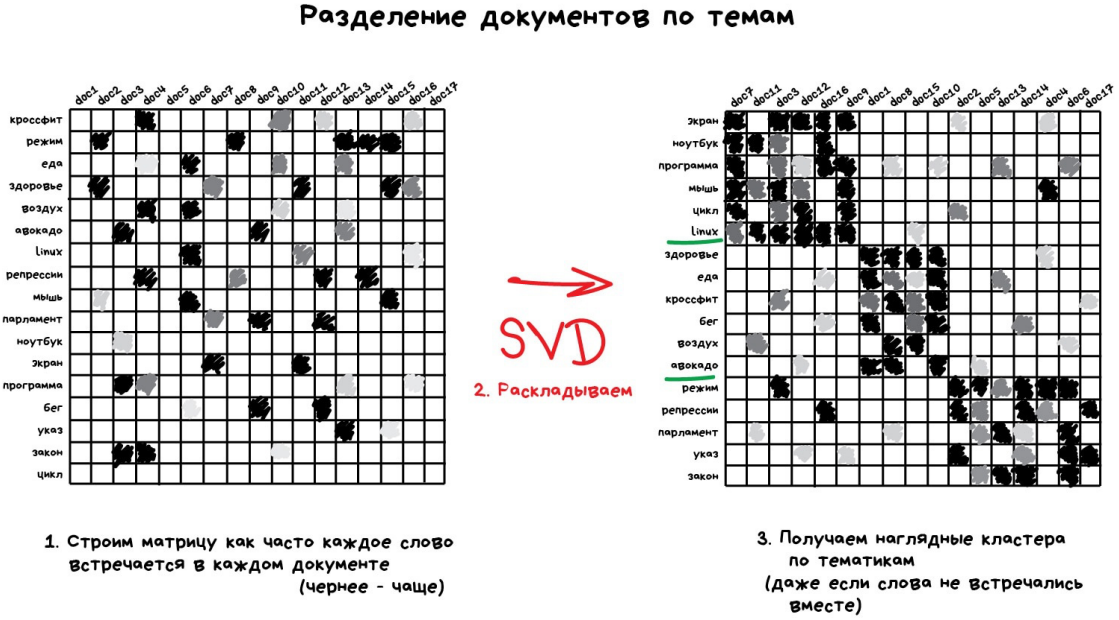
Источник - Википедия

Текстовые данные зачастую имеют высокую размерность. Скрытый семантический анализ является популярным методом по уменьшению размерности, который основывается на SVD. LSA по факту пытается определить темы внутри текста математически. Данный подход обучения без учителя основывается на двух вещах:

- Гипотеза о распределении, которая утверждает, что слова со схожими значениями часто встречаются вместе. Это лучше всего описывает цитата Дж.Р.Ферта: “You shall know a word by the company it keeps”.
- Сингулярное разложение, математика по которому была дана выше, тут ее не будет.

Заметим, что LSA - метод обучения без учителя, то есть у нас нет таргетов. Скрытые темы могут присутствовать в тексте, а могут и нет) Чаще всего LSA применяется на неструктурированных, неразмеченных данных.

Кратко: LSA трансформирует исходные документы в n различных частей, каждая из которых как бы выражает свой взгляд на тему/смысл текста. То есть, если представить текст как некоторую идею, то мы получим n различных пониманий идеи. Таким образом, LSA приводит нашу таблицу данных к таблице скрытых концепций.



Латентно-семантический Анализ (LSA)

Представим, что у нас есть таблица с текстовыми данными, где строка - это один документ, а столбцы представляют термы (слово или группа слов). Это классическое представление текстовых данных (document-term matrix). Значения в матрице показывают, насколько важно конкретное слово в конкретном документе; если значение нулевое, то в текущем документе данное слово отсутствует.

Разные документы могут иметь различные темы: например, общая тема документов “политика” и имеются 3 конкретных темы - “внешняя политика”, “выборы”, “реформы”.

Положим, что часть документов принадлежит только к 1 из категорий, часть - к 2м, а некоторые - ко всем трем. Если сосредоточиться непосредственно на темах, то естественным будет ожидать, что определенные слова будут появляться чаще в определенных темах. Так, например, в документах о внешней политике, чаще могут встречаться термы “Middle East”, “EU”, “embassies”.

Если отобразить эти темы и термы в новой таблице, где строки - это термы, то мы увидим, к какой теме наиболее относится каждый терм.

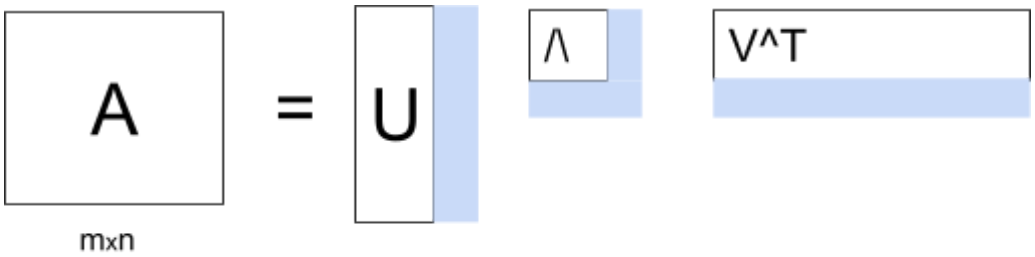
Наконец последний компонент, который тяжелее уже выразить в виде матрицы. Это набор чисел, по одному на каждую тему. Эти числа отражают то, насколько каждая тема “объясняет” исходные данные. (чувствуете, чувствуете? вот они где, сингулярные значения).

Каким же образом эти значения “объясняют” данные? Предположим, что на самом деле тема “реформы” не была мажорирующей темой в документах и бОльшая часть данных про внешнюю политику и выборы. Тогда “реформы” будут иметь

маленькое значение в упомянутом наборе чисел. Или же альтернативный сценарий - может так случиться, что все три темы будут иметь небольшие значения. Это значит, что мы плохо предположили тематику и может быть есть четвертая тема (или их даже больше), например, экономика. Ограничив себя тремя темами, мы лишили себя возможности получить более подробный и точный взгляд на наши данные.

Заметим, что наши матрицы имеют кое-что общее - они про документы, а все три компоненты (document-term matrix, матрица по термам и темам и набор чисел) имеют общее в виде тем (это отсылочка на размерность матриц из математики выше).

Ну и напоследок, для лучшего понимания, пример с циферками, чтоб показать именно *уменьшение* размерности. Пусть у нас было 100 статей и 10000 различных термов. Тогда наша исходная document-term matrix имеет размерность [100,10000]. Перед тем, как разбивать наши исходные данные на три компоненты, нам нужно выбрать число тем, пусть их будет три. Это значит, что в нашей document-topic матрице мы сократим около 99997 столбцов (то же самое для term-topic матрицы). Строки и столбцы, которые мы удаляем из матриц, показаны на рисунке ниже цветными прямоугольниками. A - исходная document-term matrix, U - document-topic матрица, Λ - сингулярные значения, а V^T - term-topic матрица (перевернутая по диагонали).



По итогу мы отказываемся от слов, переходя к темам, что является более компактным способом выразить текст.

Метод уменьшения размерности tSNE.

Источник - Википедия

t-distributed Stochastic Neighbor Embedding - метод нелинейного снижения размерности, работающий таким образом, что похожие объекты моделируются близко расположенными точками, а непохожие точки моделируются с большой вероятностью точками, далеко друг от друга отстоящими.

Алгоритм:

Пусть дан набор объектов высокой размерности $x_1,\dots,x_N$. В качестве метрики расстояния по умолчанию используется расстояние Евклида. tSNE вычисляет вероятности p_{ij}, которые пропорциональны похожести объектов x_i и x_j:

$$p_{i|j} = \frac{e^{(-||x_i - x_j||^2/2\sigma_i^2)}}{\sum_{k \neq i} e^{(-||x_i - x_k||^2/2\sigma_i^2)}}$$

Похожесть точки данных x_j точки x_i является условной вероятностью $p_{i|j}$, что для x_i будет выбрана x_j в качестве соседней точки, если соседи выбираются пропорционально их гауссовой плотности вероятности (т.е. используя нормальное распределение) с центром в x_i: $p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N}$. Вероятности с $i = j$ принимаются равными 0.

Алгоритм tSNE стремится получить отображение $y_1,\ldots, y_N$ в *d*-мерное пространство, которое отражает похожести p_{ij}, насколько это возможно. Для этого алгоритм измеряет похожесть q_{ij} между двумя точками y_i и y_j с помощью очень похожего подхода: $q_{ij} = \frac{(1+||y_i - y_j||^2)^{-1}}{\sum_{k \neq l} (1+||y_k - y_j||^2)^{-1}}$

Здесь имеющее утяжелённый хвост *t*-распределение Стьюдента используется для измерения похожести между точками в пространстве низкой размерности, чтобы иметь возможность непохожие объекты расположить на карте далеко друг от друга. В этом случае устанавливается $q_{ii} = 0$.

Расположения точек y_i в пространстве малой размерности определяется минимизацией расстояния Кульбака — Лейблера распределения Q от распределения P, то есть $KL(P||Q) = \sum_{i \neq j} p_{ij} log \frac{p_{ij}}{q_{ij}}$. Минимизация этого расстояния по отношению к точкам y_i осуществляется с помощью градиентного спуска. Результатом оптимизации является отображение, которое отражает схожесть между объектами пространства высокой размерности.

Метод понижения размерности UMAP.

Источник - Википедия

Uniform Manifold Approximation and Projection - метод нелинейного снижения размерности.

Алгоритм

Пусть есть выборка $X = \{x_1,\ldots, x_n\}$.

UMAP по заданной метрике считает расстояния между объектами и для каждого объекта x_i определяет список из его k ближайших соседей $T = \{t_1,\ldots, t_k\}$, а также рассчитывает расстояния до ближайшего соседа $\rho_i = min_{t \in T} d(x_i, t)$. Помимо этого задается величина σ_i:

$$\sum_{t \in T} e^{-\left(\frac{d(x_i,t) - \rho_i}{\sigma_i}\right)} = log_2 k.$$

Далее алгоритм строит взвешенный ориентированный граф, в котором ребра графа соединяют каждый объект с его соседями. Все рёбра от объекта x_i до его соседа t_j рассчитываются следующим образом:

$$w(x_i \to t_j) = e^{-(\frac{d(x_i,t)-\rho_i}{\sigma_i})}.$$

Полученная заранее величина σ_i нормирует сумму весов для каждого объекта к заданному числу $log_2 k$.

Далее алгоритм строит взвешенный неориентированный граф. Если между вершинами существует два ребра с разными весами, то строится одно ребро, вес которого интерпретируется как вероятность существования данного ребра от одного объекта к другому. Исходя из этого, ребра между двумя вершинами объединяются в одно с весом, равным вероятности существования хотя бы одного ребра:

$$w(x_i, x_j) = w(x_i \to x_j) + w(x_j \to x_i) - w(x_i \to x_j) \cdot w(x_j \to x_i)$$

Множество ребер E такого графа является нечетким множеством из случайных величин Бернулли. Алгоритм создает новый граф в низкоразмерном пространстве и приближает множество его ребер к исходному. Для этого он минимизирует сумму дивергенций Кульбака-Лейблера для каждого ребра E из исходного и нового нечетких множеств:

$$\sum_{e \in E} w_h(e) log \frac{w_h(e)}{w_l(e)} + (1 - w_h(e)) log(\frac{1-w_h(e)}{1-w_l(e)}) \to min_{w_l}$$

, где $w_h(e)$ - функция принадлежности нечеткого множества из ребер в высокоразмерном пространстве, $w_l(e)$ - функция принадлежности нечеткого множества из ребер в низкоразмерном пространстве.

UMAP решает задачу минимизации с помощью стохастического градиентного спуска. Полученное множество из ребер определяет новое расположение объектов и, соответственно, низкоразмерное отображение исходного пространства.

Метрики качества задач классификации. ROC-кривая, PR-кривая.

Метрики качества *используются*:

- Для задания функционала ошибки (используется при обучении).
- Для подбора гиперпараметров (используется при измерении качества на кросс-валидации). В том числе можно использовать другую метрику, которая отличается от метрики, с помощью которой построен функционал ошибки.
- Для оценивания итоговой модели: пригодна ли модель для решения задачи.

Матрица ошибок: $\widehat{y}$ - ответ алгоритма, y - истинная метка класса.

	$y = 1$	$y = 0$
$\hat{y} = 1$	True Positive (TP)	False Positive (FP)
$\hat{y} = 0$	False Negative (FN)	True Negative (TN)

Метрики качества классификации

Accuracy

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

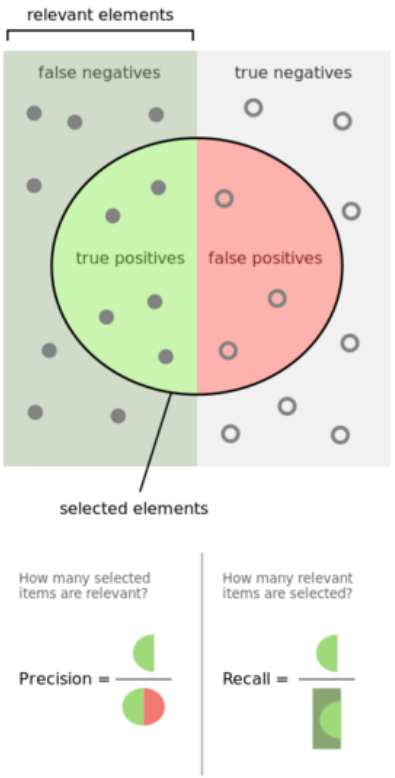
Свойства:

- Бесполезна в задачах с неравными классами, так как может выдавать более высокую точность при менее точных ответах.
- Не учитывает разные цены разных типов ошибок.

Precision/Recall

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

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



Precision - доля объектов, названных классификатором положительными и при этом действительно являющимися положительными

Recall - доля объектов положительного класса из всех объектов положительного класса нашел алгоритм.

Свойства:

- Ведение precision не позволяет нам записывать все объекты в один класс, так как в этом случае мы получаем рост уровня False Positive.
- Recall демонстрирует способность алгоритма обнаруживать данный класс вообще.
- Precision демонстрирует способность отличать искомый класс от других классов.
- Применимы в условиях дисбаланса классов так как не зависят от соотношения классов.

F-мера

$$F = 2\frac{Precision \times Recall}{Precision + Recall}$$

Данная формула придает одинаковый вес и precision и recall, из-за чего падает одинаково при их уменьшении.

Можно придать precision и recall разные веса:

$$F_\beta = (1 + \beta^2) \cdot \frac{Precision \times Recall}{(\beta^2 \cdot Precision) + Recall}$$

, где β принимает значения в диапазоне $0 < \beta < 1$ если вы хотите отдать приоритет точности, а при $\beta > 1$ приоритет отдается полноте. При $\beta = 1$ формула сводится к предыдущей и вы получаете сбалансированную F-меру.

Арифметическое среднее

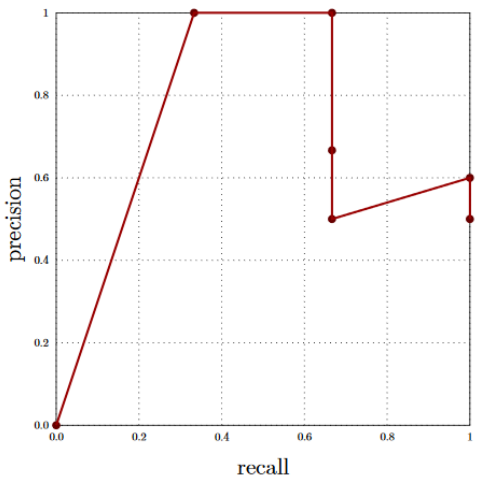
$$A = \frac{1}{2}(Precision + Recall)$$

Может вызвать ситуацию, когда константный и разумный алгоритмы могут лежать на одной линии, которая является недопустимой, поэтому следует искать другой способ построения единой метрики.

Метрики качества оценок принадлежности классу

PR-кривая

1. Сортируем объекты выборки по степени возрастания оценки принадлежности целевому классу.
2. Перебираем все пороги классификации, начиная с максимального - то есть в начале мы ни один объект не относим к целевому классу, затем только первый с максимальной оценкой, затем второй, и так далее.
3. Для каждого порога находим precision и recall.
4. Наносим соответствующую точку в осях precision-recall.
5. Соединяем все полученные точки и получаем PR-кривую.



Свойства:

- Левая точка: всегда (0, 0) (все объекты относим к классу 0).
- Правая точка: (1, r), где r - число объектов класса 1 в выборке.
- Если выборка идеально разделима, то кривая пройдет через точку (1, 1).
- Чем больше площадь под кривой, тем лучше.
- Precision нормируется на число положительных прогнозов, изменится при перемене соотношений классов.
- Максимально возможная площадь под PR-кривой зависит от соотношений классов.
- Хорошо подходит для измерения качества при сильном дисбалансе классов.

AUC-PRC

Площадь под PR-кривой. При изменении баланса классов значение AUR-PRC меняется.

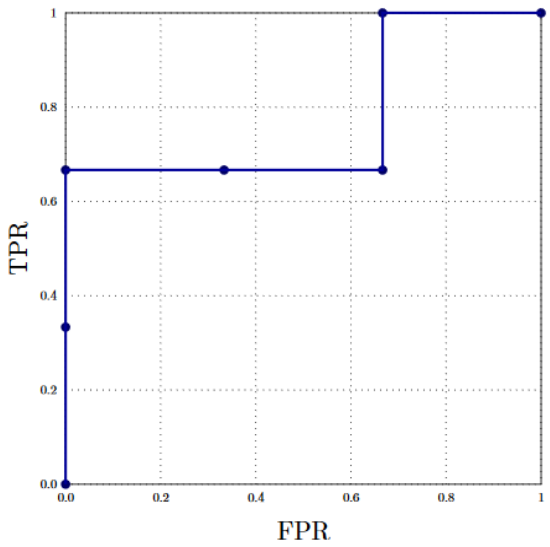
ROC-кривая

Строится в осях False Positive Rate (ось абсцисс) и True Positive Rate (ось ординат):

$FPR = \frac{FP}{FP+TN}$ - доля ошибочных положительных классификаций (специфичность алгоритма),

$TPR = \frac{TP}{TP+FN}$ - доля правильных положительных классификаций (чувствительность алгоритма).

Строится аналогично PR-кривой.



Свойства:

- Левая точка: всегда (0, 0) (все объекты относим к классу 0).
- Правая точка: всегда (1, 1) (все объекты относим к классу 1).
- Если выборка идеально разделима, то кривая пройдет через точку (0, 1).
- Площадь меняется от ½ до 1.
- Чем больше площадь под кривой тем лучше.
- Метрики качества FPR и TPR нормируются на размеры классов, поэтому ROC-кривая не изменится при перемене соотношения классов, следовательно и AUC-ROC не изменится.
- Интерпретация: AUC-ROC равен вероятности того, что случайно взятый объект класса 1 получит оценку принадлежности к классу 1 выше, чем случайно взятый объект класса 0.
- Имеет проблемы при сильном дисбалансе классов.

AUC-ROC

Площадь под кривой ROC.

Линейная регрессия

Источник - Хабр Википедия

Регрессия: $E(Y|X) = f(X)$, или эквивалентно

$Y = f(X) + \varepsilon$, ε − ошибка, f − функция регрессии, X − матрица признаков объектов.

Линейная регрессия:

$$f_w(x) = w_0 + \sum_{i=1}^{p} w_i x_i \equiv w^T x$$

w - вектор весов.

Вводим фиктивную размерность $x_0 = 1$.

Заметим, что x и w включают bias:

$$w = (w_0, w_1, \ldots w_n)^T,\ x = (1, x_1, \ldots, x_n)^T$$

Задача линейной регрессии:

Пусть есть объекты (x^i, y^i), $i = 1,..,n$, где $x^i \in R^p$, $y^i \in R$. x - описание объекта, y - метка объекта.

В матричном виде данные есть:

$$X = [x^1,...,x^n]^T,\ X \in R^{n \times p}$$
$$Y = [y^1,...,y^n]^T,\ Y \in R^n$$

Линейная регрессия тогда: $f_w(X) = Xw = \widehat{Y} \approx Y$.

Как выбирать веса?

$$Empirical\,risk\ =\ Q(X) = \sum_{i=1}^{n} L(y^i, f_w(x^i)) \to min_w$$

В качестве метрик выбираются различные функции: MSE, RMSE, MAE и др.

Минимизация MSE эквивалентна Maximum Likelihood Estimation (максимизации правдоподобия) при определенных условиях (нормальное распределение ошибки с матожиданием 0 и постоянной дисперсией σ^2).

Аналитическое решение:

$$Q_{MSE}(X)\ =\ \sum_{i=1}^{n} (y^i - f_w(x^i))^2 = ||Y - Xw||^2 = (Y - Xw)^T(Y - Xw) \to min_w$$
$$\nabla_w Q(X) = \nabla_w (Y^T Y - (Xw)^T Y - Y^T Xw + (Xw)^T Xw) = 0 - XY^T - XY^T + 2X^T Xw\ = 0$$
$$w^* = (X^T X)^{-1} X^T Y$$

Корреляция - это мера силы связи между двумя переменными. Корреляционное отношение - мера нелинейной связности. Коэффициент корреляции определяет степень изменения одной переменной на основе изменения другой переменной. Коэффициент корреляции Пирсона или просто коэффициент корреляции r представляет собой значение от -1 до 1. Это наиболее часто используемый коэффициент корреляции, действительный только для линейной зависимости между переменными. Если r = 0, связи не существует, а если r≥0, связь прямо пропорциональна; значение одной переменной увеличивается с увеличением другой. Если r≤0, соотношение обратно пропорционально; одна переменная уменьшается по мере увеличения другой.

Ковариация - мера линейной зависимости двух с.в. Если величины независимы, то ковариация равна нулю.

Теорема Гаусса-Маркова

Пусть есть задача регрессии $Y = f(X) + \varepsilon$ и соблюдены следующие условия:

1. $E(\varepsilon_i) = 0\ \forall i$ (базовое предположение любой регрессии)
2. $Var(\varepsilon_i) = \sigma_i^2 < inf\ \forall i$ (у каждой ошибки конечные дисперсии)
3. $Cov(\varepsilon_i, \varepsilon_j) = 0\ \forall i$ (ошибки независимы).

Тогда с минимизацией MSE loss'а мы получим Best Linear Unbiased Estimation (BLUE, лучшую линейную несмещенную оценку, то есть estimator с минимальной дисперсией)

$$w^* = (X^T X)^{-1} X^T Y$$

, где w^* - случайная величина, так как Y - случайная,

$E(w^*) = w_{true}$ - несмещенная оценка, так как совпадает с истинным значением,

$Var(w^*) = min$ - выбираем из всех векторов омега такой, чтоб дисперсия была минимальной.

Это позволяет сделать вывод, что w^* существует и единственная, при этом равна $w^* = (X^T X)^{-1} X^T Y$. Об этом и говорит теорема Гаусса-Маркова.

Проблемы: матрица $X^T X$ может оказаться сингулярной (определитель равен нулю), то есть неполного ранга (значит, признаки линейно-зависимы => не удовлетворяет условиям теоремы Маркова-Гаусса => решение МНК не существует).

Численные алгоритмы обращения в таком случае неустойчивы. В таком случае, множество решений получилось вырожденным, то есть получилась гиперплоскость решений и все решения на ней примерно одинаковы. Чувствительность к неточной арифметике с плавающей точкой оказывается в таком случае огромной, см. разницу в значениях омеги на скрине ниже.

```
w_true
```

```
array([ 2.68647887, -0.52184084, -1.12776533])
```

```
w_star = np.linalg.inv(X.T.dot(X)).dot(X.T).dot(Y)
w_star
```

```
array([   2.68027723, -186.0552577 ,  184.41701118])
```

L2 регуляризация (Тихонова, Ridge Regression)

Добавим к плохо обращаемой матрице хорошо обращаемую.

Прекрасно обращается диагональная, ее и добавим с положительным коэффициентом (поэтому и квадрат):

$$w = (X^T X + \lambda^2 I)^{-1} X^T Y$$

Теперь с вычислительной точки зрения обратимость матрицы улучшилась, и чем больше константа, тем устойчивее решение. Но это меняет решение - теперь оно смещенное (Гаусс-Марков не работает), и w не сойдется к настоящей w_{true}.

Однако, если поменять loss функцию, то все будет чудесно

$$L_2 = \sum_{i=1}^{n} L(y_i, x_i w^T) + \lambda^2 ||w||_2^2$$

Важно: регуляризация ограничивает веса, причем при добавлении L2 регуляризации все веса уменьшаются +- равномерно.

L1 регуляризация (Lasso regression)

Добавим к лоссу L1 норму:

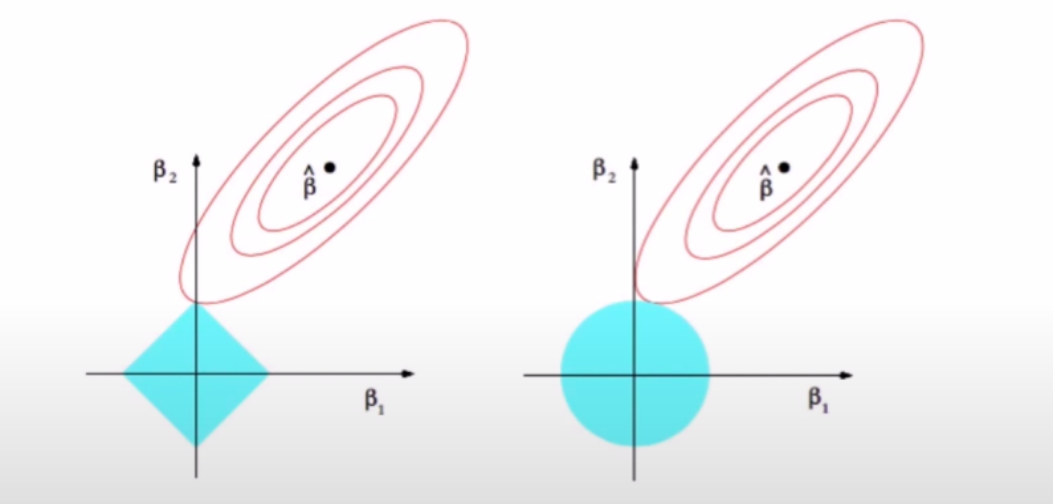
$$L_1 = \sum_{i=1}^{n} L(y_i, x_i w^T) + \lambda^2 ||w||_1$$

Аналитическое решение уже выписать нельзя. Но можно выписать для ортонормированного случая, когда мы диагонализовали матрицу признаков:

$$\widehat{w_j} = \{(w_j^* - \lambda^2)_+ \quad , if\ w_j^* \geq 0;\ -(|w_j^*| - \lambda^2)_+, if\ w_j^* < 0\}$$

Как это работает - мы взяли выборку, обучили веса с помощью МНК, затем взяли положительные веса и прижали их к нулю на величину λ^2, а отрицательные веса тоже прижимаем к нулю, но с другой стороны. Поэтому часть весов прижались к нулю, так что выполняется отбор признаков.

Геометрическая интерпретация



ElasticNet

Применим сразу две регуляризации, получим *ElasticNet*:

$$L_{EN} = \sum_{i=1}^{n} L(y_i, x_i w^T) + \lambda_1^2 ||w||_1 + \lambda_2^2 ||w||_2^2$$

Плюсы:

- Хорошо изучены.
- Очень быстрые, могут работать на больших выборках.
- Практически вне конкуренции, когда признаков много (от сотен тысяч и более), и они разреженные (хотя есть ещё факторизационные машины).
- Коэффициенты перед признаками могут интерпретироваться (при условии что признаки масштабированы) - в линейной регрессии как частные производные зависимой переменной от признаков, в логистической - как изменение шансов на отнесение к одному из классов в e^{β_i} раз при изменении признака x_i на 1 единицу.
- Логистическая регрессия выдаёт вероятности отнесения к разным классам (что ценится в кредитном скоринге).
- Модель может строить и нелинейную границу, если на вход подать полиномиальные признаки.

Минусы:

- Плохо работают в задачах, в которых зависимость ответов от признаков не линейная.
- На практике предложения теоремы Маркова-Гаусса почти никогда не выполняются, поэтому чаще линейные методы работают хуже чем, например, SVM и ансамбли (по качеству решения задачи классификации/регрессии).

Логистическая регрессия

Источник - Хабр

Рассматривается бинарная классификация.
Логистическая регрессия является частным случаем линейного классификатора, но она обладает хорошим "умением" – прогнозировать вероятность p_+ отнесения примера x_i классу “+”: $p_+ = P(y_i = 1|x_i, w)$.

Из линейной регрессии вероятность принадлежности получается с помощью функции сигмоиды $\sigma(z) = \frac{1}{1+e^{-z}}$ следующим образом:

$P(X)$ - вероятность происходящего события X. Отношение вероятностей $OR(X) = \frac{P(X)}{1-P(X)}$. Далее вычисляем логарифм из отношения вероятностей. Именно его значение мы будем прогнозировать.

Логистическая регрессия дает прогноз следующим образом:

- Вычисляется $w^T x$. Веса w мы заранее получили в процессе обучения.
- Вычисляется логарифм отношения шансов на отнесения к классу “+” $log(OR(X)) = w^T x$.
- Вычисляем p_+:

$$p_+ = P(y_i = 1|x_i, w) = \frac{OR(X)}{1+OR(X)} = \frac{e^{w^T x}}{1+e^{w^T x}} = \frac{1}{1+e^{-w^T x}} = \sigma(w^T x).$$

Как обучается?

Вероятность принадлежности классу “+” мы уже получили, а вероятность принадлежности классу “-” равна:

$$p_- = P(y_i = -1|x_i, w) = 1 - \sigma(w^T x_i) = \sigma(-w^T x_i).$$

Эти выражения можно объединить в одно:

$$P(y = y_i|x_i, w) = \sigma(y_i w^T x_i).$$

$M(x_i) = y_i w^T x_i$ - отступ классификации на объекте x_i:

- Если отступ большой по модулю и положительный, то метка класса поставлена правильно.

- Если отступ большой (по модулю) и отрицательный, значит метка класса поставлена неправильно, а объект находится далеко от разделяющей гиперплоскости (скорее всего такой объект – аномалия, например, его метка в обучающей выборке поставлена неправильно).
- Если отступ малый (по модулю), то объект находится близко к разделяющей гиперплоскости, а знак отступа определяет, правильно ли объект классифицирован.

Вероятность наблюдать вектор y в выборке X:

$P(y|X,w) = \prod_{i=1}^{N} P(y = y_i|x_i, w)$, где N - длина выборки (произведение по всем элементам выборки так как эти события должны произойти одновременно).
Прологарифмируем это выражение:

$$logP(y|X,w) = log\prod_{i=1}^{N} P(y = y_i|x_i, w) = log\prod_{i=1}^{N} \sigma(y_i w^T x_i)$$

$$= \sum_{i=1}^{N} log\sigma(y_i w^T x_i) = \sum_{i=1}^{N} log(\frac{1}{1+e^{-yw^T x}}) = -\sum_{i=1}^{N} log(1 + e^{-yw^T x})$$

То есть в данном случае принцип максимизации правдоподобия приводит к минимизации выражения:

$$L_{log}(X,y,w) = \sum_{i=1}^{N} log(1 + e^{-yw^T x}).$$

Это *логистическая функция потерь*, просуммированная по всем объектам обучающей выборки.

Можно сделать вывод, что в задаче классификации, не умея напрямую минимизировать число ошибок (по крайней мере, градиентными методами это не сделать – производная 1/0 функции потерь в нуле обращается в бесконечность), мы минимизируем некоторую ее верхнюю оценку. В данном случае это логистическая функция потерь (где логарифм двоичный, но это не принципиально), и справедливо:
$L_{1/0}(X,y,w) = \sum_{i=1}^{N} [M(x) < 0] \leq \sum_{i=1}^{N} log(1 + e^{-yw^T x}) = L_{log}(X,y,w)$, где $L_{1/0}(X,y,w)$ - число ошибок логистической регрессии с весами w на выборке (X,y), а $M(x)$ - отступ. То есть уменьшая верхнюю оценку $L_{log}(X,y,w)$ на число ошибок классификации, мы таким образом надеемся уменьшить и само число ошибок.

К логистической регрессии можно применять L1 и L2 регуляризации.

Достоинства:
- Один из простейших алгоритмов машинного обучения, обеспечивающий большую эффективность.
- Низкая дисперсия.
- Может использоваться для извлечения объектов.

- Логистические модели могут быть легко обоснованы новыми данными с использованием стохастического градиентного спуска.

Недостатки:

- Не очень хорошо обрабатывает большое количество категориальных переменных.
- Требуется преобразование нелинейных функций.
- Они не достаточно гибки, чтобы естественно охватить более сложные отношения.

Стратегии мультиклассовой классификации:

1. One vs Rest
   Берем один класс, остальные классы помечаем противоположным. Перебираем так все классы, по итогу число классификаторов равно числу классов. Каждый классификатор тренируется на всем датасете. Все сломается, если центроиды расположены на одной прямой. Плюс есть серые зоны.
2. One vs One
   Сравниваем классы попарно, откидывая все остальные классы. Тогда получим $\frac{n(n-1)}{2}$ классификаторов. Каждый обучается на подвыборке.
   Проблема возникнет в случае наличия маленьких по мощности классов - будут маленькие подвыборки.

Метрики качества задачи регрессии

Источники: академия Яндекса

MAE

Mean Absolute Error - средний модуль отклонения:

$$MAE(y^{true}, y^{pred}) = \frac{1}{N}\sum_{i=1}^{N} |y_i - f(x_i)|$$

Оптимальный константный алгоритм с точки зрения этой функции ошибки

$$a = median(\{y_i\}_{i=1}^{N})$$

т.е. решением задачи MAE min в классе констант является медиана.

В задачах регрессии с MAE при ансамблировании часто вместо усреднения нескольких алгоритмов берут их медиану – это, как правило, повышает качество. Если в такой задаче целевой признак целочисленный, то округление ответа часто не ухудшает качество решения.

При минимизации данной метрики методом, например, градиентного спуска, проявляется слабая зависимость от выбросов.

MSE

Mean Squared Error - среднеквадратичная ошибка:

$$MSE(y^{true}, y^{pred}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - f(x_i))^2$$

MSE квадратично штрафует за большие ошибки на объектах. При обучении моделей методом минимизации квадратичных ошибок модель старается хорошо подстроиться под выбросы.

Если большие ошибки для нас действительно неприемлемы, то квадратичный штраф за них - очень полезное свойство (и его даже можно усиливать, повышая степень, в которую мы возводим ошибку на объекте). Однако если в наших тестовых данных присутствуют выбросы, то нам будет сложно объективно сравнить модели между собой: ошибки на выбросах будет маскировать различия в ошибках на основном множестве объектов.

Таким образом, если мы будем сравнивать две модели при помощи MSE, у нас будет выигрывать та модель, у которой меньше ошибка на объектах-выбросах, а это, скорее всего, не то, чего требует от нас наша бизнес-задача.

RMSE

Root Mean Squared Error - корень из среднеквадратичной ошибки:

$$RMSE(y^{true}, y^{pred}) = \sqrt{MSE(y^{true}, y^{pred})} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - f(x_i))^2}$$

Это сделано для того, чтобы показатель эффективности MSE имел размерность исходных данных.

Такой функционал легко оптимизировать, используя, например, метод градиентного спуска. Этот функционал сильно штрафует за большие ошибки, так как отклонения возводятся в квадрат. Это приводит к тому, что штраф на выбросе будет очень сильным, и алгоритм будет настраиваться на выбросы. Другими словами, алгоритм будет настраиваться на такие объекты, на которые не имеет смысла настраиваться.

R^2

Коэффициент детерминации

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

Этот коэффициент показывает, какую долю дисперсии (разнообразия ответов) во всем целевом векторе модель смогла объяснить. Для разумных моделей коэффициент детерминации лежит в следующих пределах:$0 \le R^2 \le 1$,причем случай R^2=1 соответствует случаю идеальной модели, R^2=0— модели на уровне оптимальной

«константной», а R^2<0— модели хуже «константной» (такие алгоритмы никогда не нужно рассматривать). Оптимальным константным алгоритмом называется такой алгоритм, который возвращает всегда среднее значение ответов $\overline{y}$ для объектов обучающей выборки.

С точки зрения оптимизации функции среднеквадратичной ошибки и коэффициента детерминации эквивалентны.

В коэффициенте детерминации происходит нормировка: MSE-ошибка алгоритма делится на MSE-ошибку оптимального константного алгоритма.

Квантильная ошибка

$$\rho_\tau(a, X) = \frac{1}{N}\sum_{i=1}^{N}((\tau - 1)[y_i < a(x_i)] + \tau[y_i \geq a(x_i)])(y_i - a(x_i))$$

Или по другому:

$$\rho_\tau(a, X) = \frac{1}{N}(\sum_{i:y_i<y_i^P}(1 - \tau)|y_i - y_i^P| + \sum_{i:y_i\geq y_i^P}\tau|y_i - y_i^P|)$$

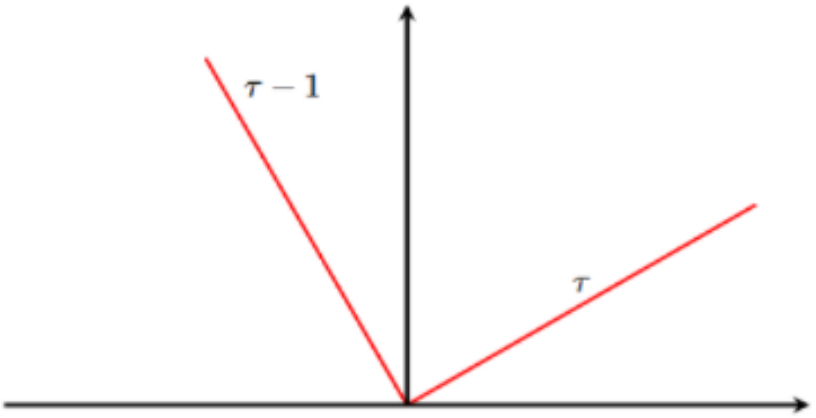


Рис. 4.1: График квантильной функции потерь

Параметр $\tau \in [0,1]$ определяет то, за что нужно штрафовать сильнее — за недопрогноз или перепрогноз. Если τ ближе к 1, штраф будет больше за недопрогноз, а если, наоборот, ближе к 0 — за перепрогноз.

Пусть один и тот же объект x с одним и тем же признаковым описанием повторяется в выборке n раз, но на каждом из повторов — свой ответ y1, ..., yn. Такое может возникнуть при измерении роста человека. Измерения роста одного и того же человека могут отличаться ввиду ошибки прибора, а также зависеть от самого человека (может сгорбиться или выпрямиться). При этом алгоритм должен для одного и того же признакового описания возвращать одинаковый прогноз. Другими словами, необходимо решить, какой прогноз оптимален для x с точки зрения различных функционалов ошибки. Оказывается, что если используется квадратичный функционал ошибки, то наиболее оптимальным прогнозом будет средний ответ на объектах, если

абсолютный, то медиана ответов. Если же будет использоваться квантильная функция потерь, наиболее оптимальным прогнозом, будет т-квантиль. В этом и состоит вероятностный смысл квантильной ошибки.

Метрические методы классификации. Метод K ближайших соседей.

Метод k ближайших соседей (kNN) - метод классификации, также иногда используемый в задачах регрессии. Основой метода является гипотеза компактности - если метрика расстояния между примерами введена достаточно удачно, то схожие примеры гораздо чаще лежат в одном классе, чем в разных.

В качестве метрики расстояния между объектами могут быть использованы следующие ***метрики***:

- Хэмминга
- Евклидовой (L2-норма)
- cosine
- расстояние городских кварталов (L1-норма)
- расстояние Минковского и др.

Важно: матрицу признаков перед использованием kNN нужно нормировать.

***Ход алгоритма*:**

1. Вычисляем расстояние от рассматриваемого объекта до каждого объекта в датасете.
2. Выбираем k наиболее близких.
3. Смотрим, какой класс доминирует, этот и присваиваем нашему объекту.

При использовании линейной функции в качестве весовой (можно добавить функцию веса, задающую “важность” соседа в зависимости от расстояния до него), возможно совпадение суммарного веса для нескольких классов. Это приводит к неоднозначности ответа при классификации. Чтобы такого не происходило, используют функцию ядра (ядро может задавать операцию сходства для сложных объектов типа графов, а сам подход kNN остается тем же)

Примеры ядер:

- Triangular: $K(r) = (1 - |r|)$
- Parabolic: $K(r) = \frac{3}{4}(1 - r^2)$
- Tricube: $K(r) = \frac{70}{81}(1 - |r|^3)^3$

Существуют проблемы с выбором k:

- Маленькие K
  - создает множество небольших областей для каждого класса
  - может привести к негладким границам и переобучению
- Большие K
  - создает более крупные регионы

- обычно приводит к более гладким границам (слишком гладкая граница приводит к недообучению)

Таким образом для *выбора k:*

- Необходимо отталкиваться от самих данных и эвристик на них.
- Можно использовать кросс-валидацию (с использованием некоторых отложенных данных).

В общем, k слишком маленький или слишком большой, это плохо!

Плюсы kNN:

1. Простая реализация;
2. Неплохо изучен теоретически - для метода ближайших соседей существует немало важных теорем, утверждающих, что на "бесконечных" выборках это оптимальный метод классификации;
3. Как правило, метод хорош для первого решения задачи, причем не только классификации или регрессии, но и, например, рекомендации;
4. Можно адаптировать под нужную задачу выбором метрики или ядра;
5. Неплохая интерпретация, можно объяснить, почему тестовый пример был классифицирован именно так. Хотя этот аргумент можно атаковать: если число соседей большое, то интерпретация ухудшается (условно: "мы не дали ему кредит, потому что он похож на 350 клиентов, из которых 70 – плохие, что на 12% больше, чем в среднем по выборке").

Минусы kNN:

1. Метод считается быстрым в сравнении, например, с композициями алгоритмов, но в реальных задачах, как правило, число соседей, используемых для классификации, будет большим (100-150), и в таком случае алгоритм будет работать не так быстро, как дерево решений;
2. Если в наборе данных много признаков, то трудно подобрать подходящие веса и определить, какие признаки не важны для классификации/регрессии;
3. Зависимость от выбранной метрики расстояния между примерами. Выбор по умолчанию евклидового расстояния чаще всего ничем не обоснован. Можно отыскать хорошее решение перебором параметров, но для большого набора данных это отнимает много времени;
4. Нет теоретических оснований выбора определенного числа соседей — только перебор (впрочем, чаще всего это верно для всех гиперпараметров всех моделей). В случае малого числа соседей метод чувствителен к выбросам, то есть склонен переобучаться;
5. Как правило, плохо работает, когда признаков много, из-за "проклятия размерности".

Дополнительно:

Метод парзеновского окна

Алгоритм kNN можно обобщить с помощью функции ядра. Рассмотрим два способа, которыми это можно сделать.

1. Метод парзеновского окна фиксированной ширины h:
   $w(i,u) = K(\frac{\rho(u,x_{i,u})}{h})$, где $w(i,u)$ – вес i-го соседа объекта u.
2. Метод парзеновского окна переменной ширины: $w(i,u) = K(\frac{\rho(u,x_{i,u})}{\rho(u,x_{k+1,u})})$

Сравним два метода. Сперва запишем классификаторы, полученные при использовании этих методов, в явном виде:

1. Фиксированной ширины: $a_h = a(u, X^m, h, K) = argmax_{y \in Y} \sum_{i=1}^{m} [y_{i,u} = y] K(\frac{\rho(u,x_{i,u})}{h})$
2. Переменной ширины: $a_k = a(u, X^m, k, K) = argmax_{y \in Y} \sum_{i=1}^{m} [y_{i,u} = y] K(\frac{\rho(u,x_{i,u})}{\rho(u,x_{k+1,u})})$

a_h не будет учитывать соседей на расстоянии больше, чем h, а всех остальных учтет в соответствии с функцией ядра K. a_k является аналогом kNN, т.к для всех $k + i$-ых соседей функция K вернет 0, но при этом чем ближе $k - i$-ый сосед, тем больший вклад в сторону своего класса он даст.

Часто используют окно переменной ширины, т.е классификатор a_k, по следующим причинам:

1. Удобнее оптимизировать целочисленный параметр k, чем вещественный параметр h по некоторой сетке.
2. Существует большое количество задач, где точки разбросаны неравномерно. В них могут существовать области, где достаточно брать небольшую h и области, где в окно ширины h попадает только одна точка. Тогда для классификатора a_h будут существовать области, в которых не будет ни одного объекта (кроме того, который нужно классифицировать). Для таких областей непонятно как классифицировать объекты.

Обобщенная линейная модель (GLM).

***Проблемы линейной регрессии*:**

- Не линейная зависимость y от x.
- Дисперсия ошибки не постоянна и зависит от x.
- Переменная y не непрерывная, а дискретная или категориальная. Линейная регрессия подразумевает нормальное распределение y, что значит что она может быть лишь непрерывной переменной. При попытке построения линейной регрессии на дискретную или категориальную переменную, модель будет предсказывать негативные значения, что не есть правильно. Может здесь имеется в виду что возможны случаи, когда целевая переменная не может быть меньше нуля, а линейная регрессия будет предсказывать отрицательные значения?

Обобщенные линейные модели расширяют рамки стандартной линейной модели для решения обеих этих проблем. Мы хотим обобщить нашу линейную модель на случаи, когда y_i не принадлежит нормальному распределению. Как пример, мы хотим предсказать рейтинг, который принимает дискретные значения. Обобщенная

линейная модель рассматривает экспоненциальное семейство распределений, которое включается в себя множество распределений, таких как нормальное, биномиальное и т.д. Но также в него не включается много распределений, таких как t-Student, гиперпараметрическое и т.д.

Обобщенная линейная модель(GLM) состоит из **линейного предиктора(linear predictor):** $\lambda_i = \beta_0 + \beta_1 x_{1i} + ... + \beta_p x_{pi}$

и **двух функций:**

- **связи** - связывает линейный предиктор и параметр вероятностного распределения.
  Для случая распределения Пуассона функция связи - логарифм, так как параметр распределения Пуассона больше 0.
- **распределения** - описывает способ получения целевой переменной, то есть функцию распределения (из семейства экспоненциальных функций распределения). Параметры распределения зависят от функции предиктора.

Для случая, когда целевая переменная распределена по Пуассону:

Link function Linear predictor

$$\ln \lambda_i = b_0 + b_1 x_i$$

$$y_i \sim \text{Poisson}(\lambda_i)$$

Probability distribution

Взаимосвязь функции предиктора и параметров распределения:

$$\ln \lambda_i = b_0 + b_1 x_i$$

$$\Leftrightarrow \lambda_i = \exp(b_0 + b_1 x_i)$$

Линейная регрессия является частным случаем обобщенной линейной регрессии - она использует нормальное распределение и функцию идентичности в качестве функции связи:

$$\mu_i = b_0 + b_1 x_i$$

$$y_i \sim \mathcal{N}(\mu_i, \varepsilon)$$

Трансформация против GLM

В некоторых ситуациях переменная отклика (response variable, ответ) может быть преобразована для улучшения линейности и однородности дисперсии, чтобы можно было применить общую линейную модель.

Этот подход имеет некоторые недостатки:

- переменная отклика изменилась!
- преобразование должно одновременно улучшать линейность и однородность дисперсии.
- преобразование не может быть определено на границах выборочного пространства.

Например, распространенным средством для увеличения дисперсии со средним значением является применение логарифмического преобразования, например:

$log(y_i) = \beta_0 + \beta_1 x_1 + i \Rightarrow E(log\, Y_i) = \beta_0 + \beta_1 x_1$

Это линейная модель для среднего значения log Y, которая не всегда может быть подходящей. Например, если Y является доходом, возможно, нас действительно интересует средний доход подгрупп населения, и в этом случае было бы лучше смоделировать E(Y) с использованием GLM:

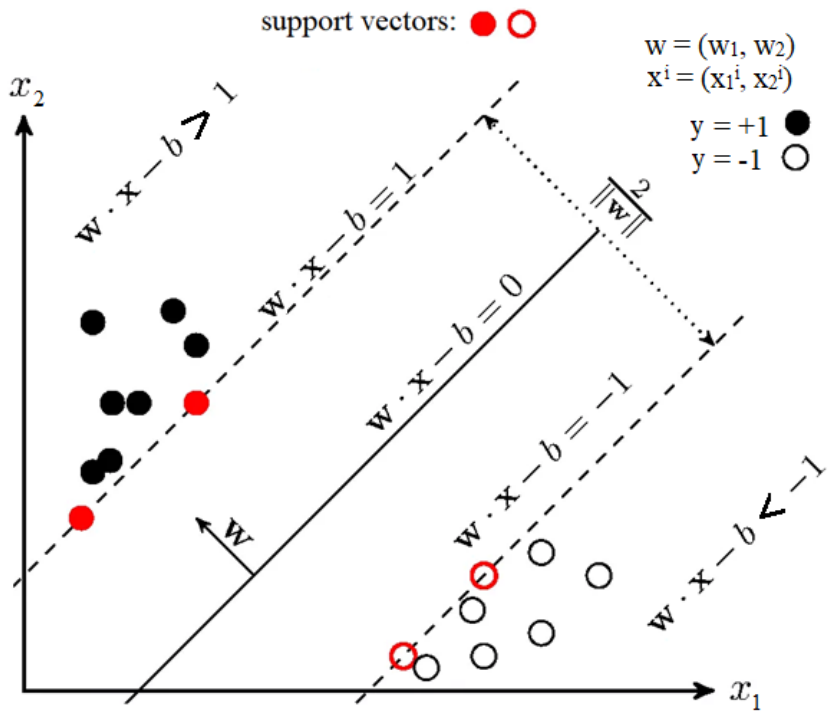
$log\, E(Y_i) = \beta_0 + \beta_1 x_1$ с $V(\mu) = \mu$. Это также позволяет избежать трудностей с y = 0.

Линейные классификаторы SVM, kernel trick.

Источники - Воронцов, стр 67, Википедия

Support Vector Machines (SVM) - метод классификации, требование оптимальности которого заключается в том, что обучающие объекты должны быть удалены от разделяющей гиперплоскости настолько далеко, насколько это возможно.

Внутри функции sign стоит линейная комбинация признаков объекта с весами алгоритма, именно поэтому SVM относится к линейным алгоритмам. Разделяющую гиперплоскость можно построить разными способами, но в SVM веса w и b настраиваются таким образом, чтобы объекты классов лежали как можно дальше от разделяющей гиперплоскости. Другими словами, алгоритм максимизирует отступ (*margin*) между гиперплоскостью и объектами классов, которые расположены ближе всего к ней. Такие объекты и называют опорными векторами (support vectors).



Обучение SVM сводится к задаче квадратичного программирования, имеющей единственное решение, которое вычисляется достаточно эффективно даже на выборках в сотни тысяч объектов. Решение обладает свойством разреженности: положение оптимальной разделяющей гиперплоскости зависит лишь от небольшой доли обучающих объектов. Они и называются опорными векторами. Остальные объекты фактически не задействуются. С помощью изящного математического приема — введения функции ядра — метод обобщается на случай нелинейных разделяющих поверхностей. Вопрос о выборе ядра, оптимального для данной прикладной задачи, до сих пор остается открытой теоретической проблемой.
Более подробно смотреть в Википедиии!

Логические методы классификации. Дерево решений. Критерий расщепления.

Источник - академия Яндекса

Определение решающего дерева

Пусть задано бинарное дерево, в котором:

- каждой внутренней вершине v приписан предикат $B_v : X \to 0, 1$
- каждой листовой вершине v приписан прогноз $c_v \in Y$, где Y - область значений целевой переменной (в случае классификации листу может быть также приписан вектор вероятностей классов).

В ходе предсказания осуществляется проход по этому дереву к некоторому листу. Для каждого объекта выборки x движение начинается из корня. В очередной внутренней вершине v проход продолжится вправо, если $B_v(x) = 1$, и влево, если $B_v(x) = 0$.

Проход продолжается до момента, пока не будет достигнут некоторый лист, и ответом алгоритма на объекте x считается прогноз c_v, приписанный этому листу.

Предикат может иметь произвольную структуру, но на практике обычно используют сравнение с порогом $t \in R$ по какому то признаку j-тому признаку: $B_v(x, j, t) = [x_j \leq t]$. При проходе через узел дерева с данным предикатом объекты будут отправлены в правое поддерево, если значение j-го признака у них меньше либо равно t, и в левое — если больше.

Свойства:

- выученная функция является кусочно-постоянной, из-за чего производная равна нулю везде, где задана. Следовательно, о градиентных методах при поиске оптимального решения можно забыть;
- дерево решений (в отличие от, например, линейной модели) не сможет экстраполировать зависимости за границы области значений обучающей выборки;
- дерево решений способно идеально приблизить обучающую выборку и ничего не выучить (то есть такой классификатор будет обладать низкой обобщающей способностью): для этого достаточно построить такое дерево, в каждый лист которого будет попадать только один объект. Следовательно, при обучении нам надо не просто приближать обучающую выборку как можно лучше, но и стремиться оставлять дерево как можно более простым, чтобы результат обладал хорошей обобщающей способностью.

Построение оптимального решающего дерева

У нас есть классическая задача машинного обучения с матрицей признаков и вектором целевой переменной, а также задана функция потерь $L(f, X, y)$.

- **Полный перебор**

Так как использовать градиентный спуск невозможно, то начинаем построение дерева с построения решающих пней - деревьев глубины 1. Рассмотрим всевозможные простые предикаты $B_{j,t}(x_i) = [x_{ij} \leq t]$, $j \in [1, D]$, коих существует не более $(N - 1)D$. Таким образом решение, которое мы ищем имеет вид

$$(j_{opt}, t_{opt}) = argmin_{j,t} L(B_{j,t}, X, y).$$

Для каждого из предикатов $B_{j,t}$ нам нужно посчитать значение функции потерь на всей выборке, что, в свою очередь, тоже занимает $O(N)$. *Итоговая сложность* - $O(N^2 D)$.

Проблемы:

- Не обобщен для дерева произвольной глубины.
- При попытке обобщить посредством рекурсии поиска решающего пня и вызова в теле цикла исходной функции для всех возможных разбиений, алгоритм просто запоминает обучающую выборку, а на тестовых данных имеет плохой результат.
- Можно поставить другую задачу: построить оптимальное с точки зрения качества на обучающей выборке дерево минимальной глубины. Такая задача является NP-полной.

Ситуацию можно улучшить двумя не исключающими друг друга способами:

- Разрешить себе искать не оптимальное решение, а просто достаточно хорошее. Начать можно с того, чтобы строить дерево с помощью жадного алгоритма, то есть не искать всю структуру сразу, а строить дерево этаж за этажом. Тогда в каждой внутренней вершине дерева будет решаться задача, схожая с задачей построения решающего пня. Для того чтобы этот подход хоть как-то работал, его придётся прокачать внушительным набором эвристик.
- Заняться оптимизацией с точки зрения computer science — наивную версию алгоритма (перебор наборов возможных предикатов и порогов) можно ускорить и асимптотически, и в константу раз.

- **Жадный алгоритм**

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

- Создаем вершину v.
- Если выполнен критерий останова $Stop(X_m)$, то останавливаемся, объявляем эту вершину листом и ставим ей в соответствие ответ $Ans(X_m)$, после чего возвращаем её.
- Иначе: находим предикат (иногда ещё говорят сплит) $B_{j,t}$, который определит наилучшее разбиение текущего множества объектов X_m на две подвыборки X_l и X_r, максимизируя критерий ветвления $Branch(X_m, j, t)$.
- Для X_l и X_r рекурсивно повторим процедуру.

$Ans(X_m)$ - функция, вычисляющая ответ для листа по попавшим в него объектам из обучающей выборки:

- в случае задачи классификации — меткой самого частого класса или оценкой дискретного распределения вероятностей классов для объектов, попавших в этот лист;
- в случае задачи регрессии — средним, медианой или другой статистикой;
- простой моделью. К примеру, листы в дереве, задающем регрессию, могут быть линейными функциями или синусоидами, обученными на данных, попавших в лист. Впрочем, везде ниже мы будем предполагать, что в каждом листе просто предсказывается константа.

$Stop(X_m)$ - функция, которая решает, нужно ли продолжать ветвление или пора остановиться. Это может быть какое-то тривиальное правило: например, остановиться только в тот момент, когда объекты в листе получились достаточно однородными и/или их не слишком много.

$Branch(X_m, j, t)$ - функция, измеряющая, насколько улучшится некоторая финальная метрика качества дерева в случае, если получившиеся два листа будут терминальными, по сравнению с ситуацией, когда сама исходная вершина является листом.

Критерий ветвления

Пусть $c \in R$ - ответы регрессии и меток класса, а $c \in R^K$ в случае необходимости возврата вектора вероятностей принадлежностей к классам.

Задана некоторая функция потерь $L(y_i, c)$. В момент, когда мы ищем оптимальный сплит $X_m = X_l \cup X_r$, мы можем вычислить для объектов из X_m константный таргет c которые предсказало бы дерево, будь текущая вершина терминальной, и связанное с ними значение исходного функционала качества L. Таким образом получаем *информативность:*

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

Чем она ниже, тем лучше объекты в листе можно приблизить константным значением. При этом информативность решающего пня равна:

$$\frac{1}{|X_m|}\left(\sum_{x_i \in X_l} L(y_i, c_l) + \sum_{x_i \in X_r} L(y_i, c_r)\right) = \frac{|X_l|}{|X_m|} H(X_l) + \frac{|X_r|}{|X_m|} H(X_r).$$

Таким образом

$$Branch(X_m, j, t) = |X_m| H(X_m) - |X_l| H(X_l) + |X_r| H(X_r).$$

Получившаяся величина неотрицательна: ведь, разделив объекты на две кучки и подобрав ответ для каждой, мы точно не сделаем хуже. Кроме того, она тем больше, чем лучше предлагаемый сплит.

Далее вместо функции L подставляются конкретные функции потерь. Так же при этой подстановке вычисляется оптимальное константное предсказание c.
Например

- для задачи регрессии:
  - минимизация MSE - $c = \frac{\Sigma y_i}{|X_m|}$
  - минимизация MAE - $c = MEDIAN(Y)$
- для задач классификации - функция потерь это индикатор ошибки $L(y_i, c) = I[y_i \neq c]$,

  а оптимальным предсказанием c в листе будет наиболее частотный класс k^*. Следовательно выражение информативности выглядит следующим образом:

  $H(X_m) = min_{c \in Y} \frac{1}{|X_m|} \sum_{(x_i, y_i) \in X_m} I[y_i \neq k^*] = 1 - p_{k^*}$, где p - доля объектов класса k.
- классификация: энтропия - максимизируем логарифм правдоподобия распределения Бернулли. Пусть в вершине дерева предсказывается фиксированное распределение c, не зависящее от x_i, тогда правдоподобие имеет вид $P(y|x, c) = P(y|c) = \prod_{(x_i, y_i) \in X_m} P(y_i|c) = \prod_{(x_i, y_i) \in X_m} \prod_{k=1}^{K} c_k^{I[y_i = k]}$, где K - число

классов. Тогда $H(X_m) = \min_{\sum_k c_k = 1} \left(-\frac{1}{|X_m|} \sum_{(x_i, y_i) \in X_m} \sum_{k=1}^{K} I[y_i = k] \log c_k \right) = -\sum_{k=1}^{K} p_k \log p_k$, если подставить в c_k вектор, равный доле попавших в лист объектов этого класса k.

- классификация: критерий Джини - предсказание модели - распределение вероятностей классов. Если вместо логарифма правдоподобия в качестве критерия правдоподобия взять метрику Бриера, то получим: $H(X_m) = \min_{\sum_k c_k = 1} \frac{1}{|X_m|} \sum_{(x_i, y_i) \in X_m} \sum_{k=1}^{K} (c_k - I[y_i = k])^2 = \sum_{k=1}^{K} p_k (1 - p_k)$, если подставить вместо c_k подставить вектор, состоящий из выборочных оценок частот классов $(p_1, \ldots, p_k)$, $p_i = \frac{1}{|X_m|} \sum_i I[y_i = k]$

Ансамбли. Основные идеи методов bagging, bootstrap, boosting. Основная идея метода случайных подпространств.

Ансамбли

Ансамбль - алгоритм, который состоит из нескольких алгоритмов машинного обучения - результаты работы нескольких алгоритмов усредняется некоторым образом. При построении ансамбля стараются делать ансамбль из достаточно разнообразных алгоритмов - ошибки на отдельных алгоритмах компенсируются правильностью работы других алгоритмов ансамбля. Конечный ответ может выбираться несколькими методами:

- простое голосование
- взвешенное голосование

Существуют следующие модели ассемблирования:

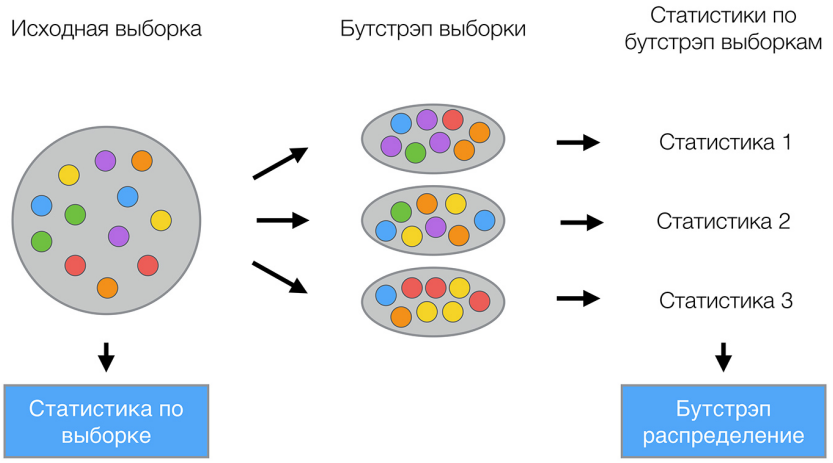
- *комитеты (голосование/усреднение)* - построение независимых алгоритмов и их усреднение / голосование по ним, в том числе с помощью бэгинга и предварительной деформации ответов. Здесь же и обобщения бэгинга – случайные леса.
- *кодировки / перекодировки ответов* - специальные кодировки целевых значений и сведение решения задачи к решению нескольких задач. Один из самых популярных приёмов – ECOC (error-correcting output coding). Другой – настройка на разные деформации целевого признака.
- *стекинг* - построение метапризнаков – ответов базовых алгоритмов на объектах выборки, обучение на них мета-алгоритма.
- *бустинг* - построение суммы нескольких алгоритмов. Каждое следующее слагаемое строится с учётом ошибок предыдущих: AdaBoost, градиентный бустинг.

- *ручные методы* - эвристические способы комбинирования ответов базовых алгоритмов (с помощью визуализаций, обучения в специальных подпространствах и т.п.)
- *однородные ансамбли* - например, нейронные сети. Формула мета-алгоритм – базовые алгоритмы разворачивается рекурсивно, применяется общая схема оптимизации полученной конструкции.

Bootstrap

Идея: Пусть имеется выборка X размера N. Равномерно возьмем из выборки N объектов с возвращением. Это означает что мы будем N раз выбирать произвольный объект выборки, причем каждый раз мы выбираем из всех исходных N объектов. Отметим, что из-за возвращения среди них окажутся повторы. Обозначим новую выборку через X_1. Повторяя процедуру M раз, сгенерируем M подвыборок X_1,..., X_M.

Теперь мы имеем достаточно большое число выборок и можем оценивать различные статистики исходного распределения.

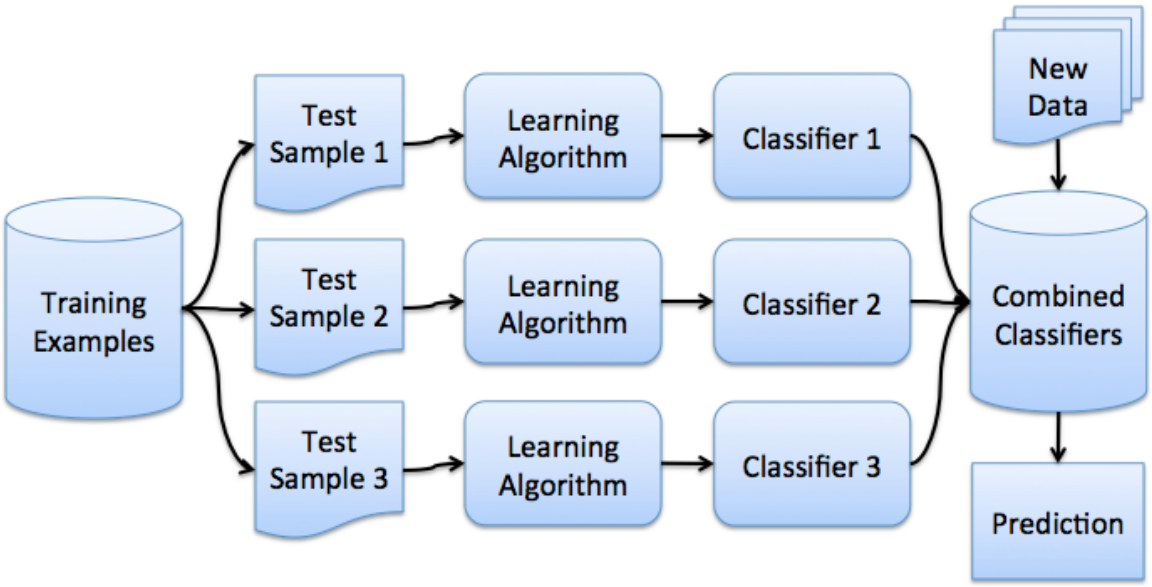


Bagging

Идея: Пусть имеется обучающая выборка X. С помощью bootstrap сгененерируем выборки X_1,..., X_M. На каждой полученной выборке обучаем свой классификатор $a_i(x)$. Итоговый классификатор будет усреднять ответы всех этих алгоритмов (в случае классификации это соответствует голосованию):

$a(x) = \frac{1}{M}\sum_{i=1}^{M} a_i(x)$.

Схема работы:



Boosting

Идея: базовые алгоритмы строятся не независимо, каждый следующий мы строим так, чтобы он исправлял ошибки предыдущих и повышал качество всего ансамбля.

Основной принцип реализации бустинга – Forward stagewise additive modeling (FSAM). Пусть, например, решается задача регрессии с функцией ошибки $L(y, a)$ Предположим, мы уже построили алгоритм $a(x)$, теперь строим $b(x)$ таким образом, чтобы $\sum_{i=1}^{m} L(y_i, a(x_i) + b(x_i)) \to min$. Если эту идею применить рекурсивно, получится следующий алгоритм:

1. Начать с $a_0(x) \equiv 0$
2. Цикл

$$(b, \eta) = argmin_{b,\eta} \sum_{i=1}^{m} L(y_i, a_{k-1}(x_i) + \eta b(x_i))$$

$$a_k = a_{k-1} + \eta b$$

Случайные подпространства

Источник - Википедия

Идея: ансамблевое обучение, которое пытается уменьшить корреляцию между оценщиками в ансамбле, обучая их на случайных выборках признаков вместо всего набора признаков. Аналогичен бэггингу, за исключением того, что признаки («атрибуты», «предикторы», «независимые переменные») выбираются случайным образом с возвращением.

Алгоритм:

- Пусть есть выборка X из N элементов, каждый из которых обладает D признаками.
- Пусть L количество независимых моделей в ансамбле.
- Для каждой модели l выбираем подвыборку из $n_l < N$ элементов с возвращением. Обычно мощность n_l равна для всех моделей.
- Для каждой модели l создаем обучающую выборку посредством выбора d_l признаков из множества всех признаков D.
- Обучаем модель.

Случайный лес. Регуляризация.

Источник: - Дьяконов

Случайный лес

Определение:

Это множество решающих деревьев. В задаче регрессии их ответы усредняются, в задаче классификации принимается решение голосованием по большинству. Все деревья строятся независимо по следующей схеме:

- Выбирается подвыборка обучающей выборки размера samplesize (может быть с возвращением) – по ней строится дерево (для каждого дерева — своя подвыборка).
- Для построения каждого расщепления в дереве просматриваем max_features случайных признаков (для каждого нового расщепления — свои случайные признаки).
- Выбираем наилучшие признак и расщепление по нему (по заранее заданному критерию). Дерево строится, как правило, до исчерпания выборки (пока в листьях не останутся представители только одного класса), но в современных реализациях есть параметры, которые ограничивают высоту дерева, число объектов в листьях и число объектов в подвыборке, при котором проводится расщепление.

Базовые алгоритмы должны быть хорошими и разнообразными (поэтому каждое дерево строится на своей обучающей выборке и при выборе расщеплений есть элемент случайности).

Достоинства:

- имеет высокую точность предсказания, на большинстве задач будет лучше линейных алгоритмов; точность сравнима с точностью бустинга,
- практически не чувствителен к выбросам в данных из-за случайного сэмлирования,
- не чувствителен к масштабированию (и вообще к любым монотонным преобразованиям) значений признаков, связано с выбором случайных подпространств,
- не требует тщательной настройки параметров, хорошо работает «из коробки». С помощью «тюнинга» параметров можно достичь прироста от 0.5 до 3% точности в зависимости от задачи и данных,

- способен эффективно обрабатывать данные с большим числом признаков и классов,
- одинаково хорошо обрабатывает как непрерывные, так и дискретные признаки,
- редко переобучается, на практике добавление деревьев почти всегда только улучшает композицию, но на валидации, после достижения определенного количества деревьев, кривая обучения выходит на асимптоту,
- для случайного леса существуют методы оценивания значимости отдельных признаков в модели,
- хорошо работает с пропущенными данными; сохраняет хорошую точность, если большая часть данных пропущена,
- предполагает возможность сбалансировать вес каждого класса на всей выборке, либо на подвыборке каждого дерева,
- вычисляет близость между парами объектов, которые могут использоваться при кластеризации, обнаружении выбросов или (путем масштабирования) дают интересные представления данных,
- возможности, описанные выше, могут быть расширены до неразмеченных данных, что приводит к возможности сделать кластеризацию и визуализацию данных, обнаруживать выбросы,
- высокая параллелизуемость и масштабируемость.

Недостатки:

- в отличие от одного дерева, результаты случайного леса сложнее интерпретировать,
- нет формальных выводов (p-values), доступных для оценки важности переменных,
- алгоритм работает хуже многих линейных методов, когда в выборке очень много разреженных признаков (тексты, Bag of words),
- случайный лес не умеет экстраполировать, в отличие от той же линейной регрессии (но это можно считать и плюсом, так как не будет экстремальных значений в случае попадания выброса),
- алгоритм склонен к переобучению на некоторых задачах, особенно на зашумленных данных,
- для данных, включающих категориальные переменные с различным количеством уровней, случайные леса предвзяты в пользу признаков с большим количеством уровней: когда у признака много уровней, дерево будет сильнее подстраиваться именно под эти признаки, так как на них можно получить более высокое значение оптимизируемого функционала (типа прироста информации),
- если данные содержат группы коррелированных признаков, имеющих схожую значимость для меток, то предпочтение отдается небольшим группам перед большими,
- больший размер получающихся моделей. Требуется O(NK) памяти для хранения модели, где K— количество деревьев,
- результаты случайного леса сложнее интерпретировать.

Случайный лес способен давать оценку важности признаков. Если построить много деревьев решений, то чем выше в среднем признак в дереве решений, тем он важнее в данной задаче классификации/регрессии. При каждом разбиении в каждом дереве улучшение критерия разделения — показатель важности, связанный с

переменной разделения, и накапливается он по всем деревьям леса отдельно для каждой переменной. Среднее снижение точности, вызываемое переменной, определяется во время фазы вычисления out-of-bag ошибки. Чем больше уменьшается точность предсказаний из-за исключения (или перестановки) одной переменной, тем важнее эта переменная, и поэтому переменные с бóльшим средним уменьшением точности более важны для классификации данных.
Оценка важности признаков используется для идентифицирования наиболее релевантных фичей для конкретной задачи, а также для генерирования метода отбора признаков. Важность фичи x_j определена как $Importance_j = \frac{1}{N_{trees}} \sum_{v \in S} G(x_j, v)$, где S - множество узлов, а x_j был выбран для разбиения, $G(x_j, v)$ - так называемый random forest gain для x_j (information gain - это разница между неопределенностью начального узла и взвешенной суммой неопределенности дочерних узлов; мера того, насколько мы уменьшили неопределенность).

Регуляризованный случайный лес умеет хорошо выделять подмножества нужных признаков, следовательно, уменьшает число фичей в задачах классификации и регрессии.

Регуляризация

Случайный лес можно регуляризировать путем добавления ограничений на:

- число деревьев - чем больше деревьев, тем лучше качество, но время настройки и работы случайного леса также пропорционально увеличиваются. Обратите внимание, что часто при увеличении числа деревьев качество на обучающей выборке повышается (может даже доходить до 100%), а качество на тесте выходит на асимптоту.
- максимальную глубину каждого дерева - ясно, что чем меньше глубина, тем быстрее строится и работает случайный лес. При увеличении глубины резко возрастает качество на обучении, но и на контроле оно, как правило, увеличивается. Рекомендуется использовать максимальную глубину (кроме случаев, когда объектов слишком много и получаются очень глубокие деревья, построение которых занимает значительное время). При использовании неглубоких деревьев изменение параметров, связанных с ограничением числа объектов в листе и для деления, не приводит к значимому эффекту (листья и так получаются «большими»). Неглубокие деревья рекомендуют использовать в задачах с большим числом шумовых объектов (выбросов).
- минимальное количество объектов в листе и минимальное число объектов, при котором выполняется расщепление - этот параметр, как правило, не очень важный и можно оставить значение по умолчанию (2). График качества на контроле может быть похожим на «расчёску» (нет явного оптимума). При увеличении параметра качество на обучении падает, а время построения случайного леса сокращается.
- максимальное число признаков для выбора расщепления - график качества на тесте от значения этого параметра унимодальный, на обучении он строго возрастает. При увеличении числа признаков для выбора расщепления увеличивается время построения леса, а деревья становятся «более

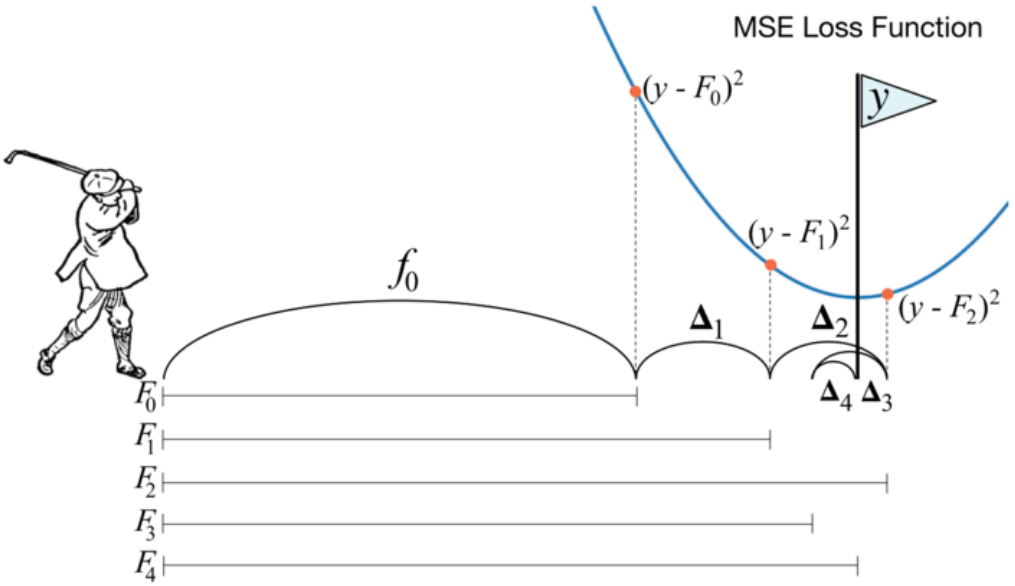
однообразными». По умолчанию он равен sqrt(n) в задачах классификации и n/3 в задачах регрессии. Это самый важный параметр! Его настраивают в первую очередь (при достаточном числе деревьев в лесе).

Градиентный бустинг. Регуляризация. Особенности бустинга над деревьями.

Источник - Википедия, Википедия

XGBoost

Градиентный бустинг — это техника машинного обучения для задач классификации и регрессии, которая строит модель предсказания в форме ансамбля слабых предсказывающих моделей, обычно деревьев решений. Обучение ансамбля проводится последовательно. На каждой итерации вычисляются отклонения предсказаний уже обученного ансамбля на обучающей выборке. Следующая модель, которая будет добавлена в ансамбль будет предсказывать эти отклонения. Таким образом, добавив предсказания нового дерева к предсказаниям обученного ансамбля мы можем уменьшить среднее отклонение модели, которое является таргетом оптимизационной задачи. Новые деревья добавляются в ансамбль до тех пор, пока ошибка уменьшается, либо пока не выполняется одно из правил "ранней остановки".



Рассмотрим иллюстрацию бустинга. На ней рассматривается поведение модели на одной точке абстрактной задачи линейной регрессии. Предположим, что первая модель ансамбля F всегда выдает выборочное среднее предсказываемой величины f_0. Такое предсказание довольно грубое, поэтому среднеквадратичное отклонение на выбранной нами точке будет довольно большим. Мы попробуем это исправить обучив модель Δ_1, которая будет "корректировать" предсказание предыдущего ансамбля F_0. Таким образом мы получим ансамбль F_1, предсказание которого будет суммироваться

из предсказаний моделей f_0 и Δ_1. Продолжая такую последовательность мы приходим к ансамблю F_4 предсказание которого суммируется из предсказаний f_0, Δ_1, Δ_2, Δ_3, Δ_4 и предсказывает в точности значение заданного таргета.

Математика алгоритма:

Функция оптимизации градиентного бустинга:

$L^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t)$, где

l - функция потерь.

$y_i, \hat{y}_i^t$ - значение i-го элемента обучающей выборки и сумма предсказаний первых t деревьев соответственно.

x_i - набор признаков i-го элемента обучающей выборки.

f_t - функция (в нашем случае дерево), которую мы хотим обучить на шаге t. $f_t(x_i)$ - предсказание на i-ом элементе обучающей выборки.

$\Omega(f)$ - регуляризация функции f. $\Omega(f) = \gamma T + \frac{1}{2}\lambda||w||^2$, T - количество вершин в дереве, w - значения в листьях, а γ и λ - параметры регуляризации.

Дальше с помощью разложения Тейлора до второго члена можем приблизить оптимизируемую функцию $L^{(t)}$ следующим выражением:

$L^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + g_i f_i(x_i) + 0.5 h_i f_t^2(x_i)) + \Omega(f_t)$ где $g_i = \frac{\partial l(y_i, \hat{y}_i^{(t-1)})}{\partial \hat{y}_i^{(t-1)}}$, $h_i = \frac{\partial^2 l(y_i, \hat{y}_i^{(t-1)})}{\partial^2 \hat{y}_i^{(t-1)}}$.

Поскольку мы хотим минимизировать ошибку модели на обучающей выборке, нам нужно найти минимум $L^{(t)}$ для каждого t.

Минимум этого выражения относительно $f_t(x_i)$ находится в точке $f_t(x_i) = \frac{-g_i}{h_i}$.

Каждое отдельное дерево ансамбля $f_t(x_i)$ обучается стандартным алгоритмом.

Особенности:

Реализация алгоритма была разработана для эффективности вычислительных ресурсов времени и памяти. Цель проекта заключалась в том, чтобы наилучшим образом использовать имеющиеся ресурсы для обучения модели. Некоторые ключевые функции реализации алгоритма включают:

- Различные стратегии обработки пропущенных данных.
- Блочная структура для поддержки распараллеливания обучения деревьев.
- Продолжение обучения для дообучения на новых данных.

Регуляризация

Количество итераций

Увеличение количества итераций работы алгоритма позволяет уменьшить ошибку, но слишком большое количество итераций ведёт к переобучению. Оптимальное значение количества итераций часто выбирается путем мониторинга ошибки прогнозирования в отдельном наборе данных для проверки.

Глубина деревьев

Чем больше глубина деревьев, тем с большей вероятностью модель будет переобучаться.

Shrinkage

Заключается в изменении правила обновления следующим образом:

$$F_m(x) = F_{m-1}(x) + v\gamma_m h_m(x),\ 0 < v \leq 1,$$

где v - “скорость” обучения ("learning rate”).
Эмпирически установлено, что использование малых "learning rate” приводит к значительному улучшению способности моделей к обобщению по сравнению с градиентным бустингом без сокращения shrinking ($v = 1$). Однако это приводит к увеличению времени вычисления и во время обучения и при выполнении инференсов - низкая скорость обучения требует большего количества итераций.

Стохастический градиентный бустинг

На каждой итерации базовая обучаемая модель учится на подмножестве всей выборки, выбранной случайным образом без замены. Размер подвыборок - постоянная доля всей выборки (если подвыборки равны всей выборке, то алгоритм аналогичен вышеописанному). Таким образом в алгоритм вносится доля случайности что предотвращает переобучение. Также это способствует ускорению работы алгоритма. Оптимальная доля подвыборок - 0.5 (для малых датасетов - от 0.5 до 0.8).

Кроме того, как и в бэггинге, подвыборка позволяет определить out-of-bag ошибку улучшения производительности прогнозирования путем оценки прогнозов по тем наблюдениям, которые не использовались при построении следующего базового обучаемого. out-of-bag оценки помогают избежать необходимости в независимом наборе данных для проверки, но часто недооценивают фактическое улучшение производительности и оптимальное количество итераций.

Число наблюдений в листьях

Реализации градиентного бустинга деревьев часто также используют регуляризацию, ограничивая минимальное количество наблюдений в конечных узлах деревьев. Он используется в процессе построения дерева, игнорируя любые разбиения, которые приводят к узлам, содержащим меньшее количество экземпляров обучающего набора, чем это.

Наложение этого ограничения помогает уменьшить дисперсию прогнозов на листьях.

Штраф за сложность дерева

Сложность модели определяется как пропорциональное количество листьев в обученных деревьях. Совместная оптимизация потерь и сложности модели соответствует алгоритму post-pruning для удаления ветвей, которые не могут уменьшить потери до порога. Также может быть добавлена L2 регуляризация в листья чтобы избежать переобучения.

Модель временного ряда. Стационарность, идентификация моделей, проверка моделей. Кросс-валидация.

Модель временного ряда

Временной ряд – это последовательность значений, описывающих протекающий во времени процесс, измеренных в последовательные моменты времени, обычно через равные промежутки.

Задача прогнозирования - найти функцию f_T:

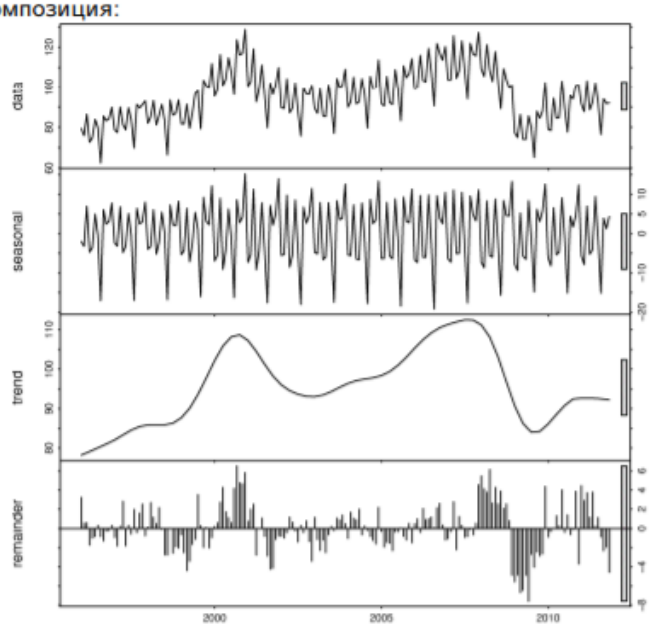
$y_{T+d} \approx f_T(y_T,..., y_1, d) \equiv \widehat{y}_{T+d|T}$, где $d \in \{1,..., D\}$ - отсрочка прогноза, D - горизонт прогнозирования.

Предсказательный интервал - интервал, в котором предсказываемая величина окажется с вероятностью не меньше заданной.

Компоненты временных рядов:

- Тренд — плавное долгосрочное изменение уровня ряда.
- Сезонность — циклические изменения уровня ряда с постоянным периодом. Цикл — изменения уровня ряда с переменным периодом (цикл жизни товара, экономические волны, периоды солнечной активности).
- Ошибка — непрогнозируемая случайная компонента ряда.

STL-декомпозиция:



Стационарность

Ряд $y_1,..., y_T$ *стационарен*, если $\forall s$ распределение $y_t,..., y_{t+s}$ не зависит от t, то есть его свойства не зависят от времени.

Стоит отметить, что стационарность подразумевает свойство процесса не менять своих характеристик, а именно постоянство матожидания, постоянство дисперсии и независимость ковариационной функции от времени (должна зависеть только от расстояния между наблюдениями (то есть мера линейной зависимости между периодами времени одинаковой длины должна быть идентична ковариации некоторого другого периода такой же длины)).

Таким образом получаются следующие *свойства*:

- тренд => нестационарность - в зависимости от расположения окна изменяется средний уровень ряда
- сезонность => нестационарность - если ширина окна меньше сезонного периода, то распределение ряда будет разным, в зависимости от положения окна
- цикл ≠> нестационарность - нельзя предсказать заранее, где будут находится максимумы и минимумы.

По стационарному ряду просто строить прогноз, так как мы полагаем, что его будущие статистические характеристики не будут отличаться от наблюдаемых текущих. Большинство моделей временных рядов так или иначе моделируют и предсказывают эти характеристики (например, матожидание или дисперсию), поэтому в случае нестационарности исходного ряда предсказания окажутся неверными.

Тест Дикки-Фуллера - методика анализа временных рядов для проверки на стационарность.

Простая модель авторегрессии: $y_t = \rho y_{t-1} + u_t$, где y_t - представляющая интерес переменная в момент времени t, ρ - коэффициент, определяющий единичный корень, u_t - является шумом или может рассматриваться как погрешность. Если $\rho = 1$ во временном ряду присутствует единичный корень, и временной ряд нестационарен.

Регрессионную модель можно представить в виде:
$\Delta y_t = (\rho - 1) y_{t-1} + u_t = \delta y_{t-1} + u_t$, где Δ - разностный оператор, $\delta = \rho - 1$.
Поэтому проверка гипотезы о единичном корне в данном представлении означает проверку нулевой гипотезы о равенстве нулю коэффициента □. Поскольку случай «взрывных» процессов исключается, то тест является односторонним, то есть альтернативной гипотезой является гипотеза о том, что коэффициент □ меньше нуля. Нулевая гипотеза H_0: □ = 0 (существует единичный корень, ряд нестационарный). Альтернативная гипотеза: H_1: □ < 0 (единичного корня нет, ряд стационарный).

Статистика теста (DF-статистика) — это обычная t-статистика для проверки значимости коэффициентов линейной регрессии. Однако, распределение данной статистики отличается от классического распределения t-статистики (распределение Стьюдента). Распределение DF-статистики выражается через винеровский процесс и называется распределением Дики — Фуллера.

Преобразование ряда в стационарный

Логарифмирование

Данный метод можно применить если во временном ряде монотонно по времени изменяется дисперсия:

$$y'_t = \begin{cases} \ln y_t, & \lambda = 0, \\ \left(y_t^{\lambda} - 1\right)/\lambda, & \lambda \neq 0. \end{cases}$$

Данный метод принадлежит к семейству преобразований Бокса-Кокса, в котором параметр λ определяет, как именно будет преобразован ряд: λ = 0 — это логарифмирование, λ = 1 — тождественное преобразование ряда, а при других значениях λ — степенное преобразование. Значение параметра можно подбирать так, чтобы дисперсия была как можно более стабильной во времени.

Дифференцирование

Переход к попарным разностям соседних значений: $y'_t = y_t - y_{t-1}$. Для нестационарного ряда часто оказывается, что получаемый после дифференцирования ряд является стационарным. Такая операция позволяет стабилизировать среднее значение ряда и избавиться от тренда, а иногда даже от сезонности. Кроме того, дифференцирование можно применять неоднократно: от ряда первых разностей, продифференцировав его, можно прийти к ряду вторых разностей, и т. д. Длина ряда при этом каждый раз будет немного сокращаться, но при этом он будет стационарным.

Также может применяться сезонное дифференцирование ряда, переход к попарным разностям значений в соседних сезонах. Если длина периода сезона составляет s, то новый ряд задаётся разностями $y'_t = y_t - y_{t-s}$. Сезонное и обычное дифференцирование могут применяться к ряду в любом порядке. Однако если у ряда есть ярко выраженный сезонный профиль, то рекомендуется начинать с сезонного дифференцирования. Уже после такого преобразования может оказаться, что ряд стационарен.

Кросс-валидация

Ничего необычного здесь нет, по-прежнему сначала необходимо выбрать подходящую для данной задачи функцию потерь: RMSE, MSE, MAE и др., которая будет следить за качеством подгонки модели под исходные данные. Затем будем оценивать на кросс-валидации значение функции потерь при данных параметрах модели, искать градиент, менять в соответствии с ним параметры и бодро опускаться в сторону глобального минимума ошибки.

Небольшая загвоздка возникает только в кросс-валидации. Проблема состоит в том, что временной ряд имеет, как ни парадоксально, временную структуру, и случайно перемешивать в фолдах значения всего ряда без сохранения этой структуры нельзя, иначе в процессе потеряются все взаимосвязи наблюдений друг с другом. Поэтому придется использовать "cross-validation on a rolling basis", что не дословно можно перевести как кросс-валидация на скользящем окне.

Суть достаточно проста — начинаем обучать модель на небольшом отрезке временного ряда, от начала до некоторого t, делаем прогноз на t+n шагов вперед и считаем ошибку. Далее расширяем обучающую выборку до t+n значения и прогнозируем с t+n до t + 2*n, так продолжаем двигать тестовый отрезок ряда до тех пор, пока не упремся в последнее доступное наблюдение. В итоге получим столько фолдов, сколько n уместится в промежуток между изначальным обучающим отрезком и всей длиной ряда.

Модель ARIMA

Источник - Дзен

Модели авторегрессии порядка p ($AR(p)$) основываются на предположении, что прошлые значения влияют на текущие. Пока справедливо предположение, мы можем построить модель линейной регрессии, которая пытается предсказать значение зависимой переменной сегодня, учитывая значения, которые она "узнала" о предыдущих днях. Предполагая что ряд стационарный:
$y_t = \alpha + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \ldots + \phi_p y_{t-p} + \varepsilon_t$, где

ε - гауссов белый шум $\varepsilon_t \sim N(0, \sigma^2)$, σ - постоянна.

y_t - линейная комбинация p предыдущих значений ряда и шумовой компоненты.

Также действуют ограничения для стационарности ряда (константы лежат в единичном круге):

- Для $AR(1)$: $-1 < \phi_1 < 1$
- Для $AR(2)$: $-1 < \phi_2 < 1, \phi_1 + \phi_2 < 1, \phi_2 - \phi_1 < 1$

При увеличении параметра p ограничения усложняются.

Модели скользящих средних порядка q *(*$MA(q)$*)* предполагают что значение зависимой переменной в текущий день зависит от ошибки предыдущих дней - текущее значение предсказываемой переменной зависит от линейной комбинации предыдущих значений ошибки. Полагая что ряд стационарный:
$y_t = \alpha + \varepsilon_t + \Theta_1 \varepsilon_{t-1} + \Theta_2 \varepsilon_{t-2} + \ldots + \Theta_p \varepsilon_{t-p}$ или

$y_t = \Theta(B)\varepsilon_t = (1 + \Theta_1 B + \Theta_2 B^2 + \ldots + \Theta_q B^q)\varepsilon_t$, где B - разностный оператор, y_t - линейная комбинация последних значений шумовой компоненты.
Для обратимости $MA(q)$ есть ограничения - они означают что модель обратима, т.е. её можно настроить по данным:

- Для $MA(1)$: $-1 < \Theta_1 < 1$
- Для $MA(2)$: $-1 < \Theta_2 < 1, \Theta_1 + \Theta_2 > -1, \Theta_2 - \Theta_1 < 1$

С ростом q ограничения усложняются.

Модель $ARMA(p, q)$ - авторегрессия с интегрированным скользящим средним. В итоге получается комбинация двух предыдущих моделей:
$y_t = \alpha + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \ldots + \phi_p y_{t-p} + \varepsilon_t + \Theta_1 \varepsilon_{t-1} + \Theta_2 \varepsilon_{t-2} + \ldots + \Theta_p \varepsilon_{t-p}$

$\phi(B)y_t = \Theta(B)\varepsilon_t$

Теорема Вольда: любой стационарный ряд может быть описан моделью $ARMA(p, q)$ с любой заранее заданной точность.

Модель $ARIMA(p, d, q)$. Она применяет дифференцирование к модели ARMA. При дифференцировании текущее значение вычитается из предыдущего, и полученная разность используется для преобразования временного ряда в стационарный. Таким образом ряд описывается моделью $ARIMA$, если ряд его разностей:

$\nabla^d y_t = (1 - B)^d y_t$, где B - разностный оператор (аналогичен $ARMA$)

$\phi(B)\nabla^d y_t = \Theta(B)\varepsilon_t$

То есть, как было сказано, мы d раз продифференцировали авторегрессионную часть ряда и получившийся ряд можно описать моделью $ARMA$. Фактически просто добавляется дифференцирование внутрь модели $ARMA$.

SARMA - Seasonal Autoregressive moving average. Рассматривается ряд с сезонным периодом длины *s*. Имея модель $ARMA(p, q)$ (формула которой приведена выше) добавим *P* авторегрессионных компонент ($\phi_s y_{t-s} + \phi_{2s} y_{t-2s} + \ldots + \phi_{Ps} y_{t-Ps}$) и Q компонент скользящего среднего ($+ \Theta_s \varepsilon_{t-s} + \Theta_{2s} \varepsilon_{t-2s} + \ldots + \Theta_{Qs} \varepsilon_{t-Qs}$). Как можно заметить, эти компоненты используют измерения временного ряда, находящиеся на расстоянии s друг от друга. Таким образом получим класс SARMA(p,q)x(P,Q).

Модель SARIMA(p,d,q)x(P,D,Q) получается из модели SARMA(p,q)x(P,Q) для ряда, к которому применили *d* раз обычное дифференцирование и D раз сезонное.

Подбор параметров для ARIMA

- α, ϕ, Θ - если все остальные параметры фиксированы, то коэффициенты регрессии подбираем по МНК; для Θ предварительно оцениваем шумовую компоненту с помощью остатков авторегрессии; если шум белый, то МНК дает оценки максимального правдоподобия.
- d,D - берем их так, чтоб ряд после дифференцирования стал стационарным; если ряд сезонный, то начинаем с сезонного дифференцирования; чем меньше раз продифференцируем, тем меньше дисперсия итогового прогноза.
- q,Q,p,P - их нельзя выбирать по МНК. Для сравнения моделей с разными q,Q,p,P можно использовать критерий Акаике:
  $AIC = -2logL + 2k,\ k = p + q + P + Q + 1$ (число параметров модели)
  Начальные приближения берем с помощью автокорреляции.

Функция автокорреляции (ACF) – корреляция между наблюдениями в текущий момент времени и наблюдениями во все предыдущие моменты времени. Мы можем использовать ACF для определения оптимального количества условий скользящей средней. Количество элементов определяет порядок модели.

Функция частичной автокорреляции (PACF): как следует из названия, PACF является подмножеством ACF. PACF выражает корреляцию между наблюдениями,

сделанными в два момента времени, с учетом любого влияния со стороны других точек данных. Мы можем использовать PACF, чтобы определить оптимальное количество элементов для использования в модели AR. Количество элементов определяет порядок модели.

Подбор модели ARIMA:

1. Смотрим на ряд
2. При необходимости стабилизируем дисперсию
3. Если ряд нестационарен, подбираем порядок дифференцирования
4. Анализируем ACF/PACF (autocorrelation, partial autocorrelation), определяем примерные p,q,P,Q
5. Обучаем модели-кандидаты, сравниваем из по AIC, выбираем лучшую
6. Смотрим на остатки модели - если они плохие, нужно что-то менять

Анализ остатков модели

Остаток - разность между фактом и прогнозом

$\widehat{\varepsilon_t} = y_t - \widehat{y_{t|t-1}}$

Остатки должны быть несмещенными ($E(\varepsilon) = 0$). Проверяем это по критерию Стьюдента или Уилкинсона.

Остатки должны быть стационарными(их характеристики не зависели от времени), проверяем по критерию KPSS или Дики-Фуллера.

Остатки должны быть не автокоррелированными (есть зависимости от предыдущих наблюдений, т.е. в них есть информация) . Проверяем по коррелограмме или Q-критерию Льюнга-Бокса.

Построение предсказательного интервала

Если остатки нормальны и стационарны - определяем интервал теоретически. Например, для прогноза на следующую точку интервал $\widehat{y_{t+1|t}} \pm 1.96\widehat{\sigma_\varepsilon}$

Если нормальность или стационарность не выполнены - генерируем симуляцию (например, бутстрапированием).

SARIMAX- модель с внешними регрессорами

$$y_t = \beta_t x_t + u_t$$
$$\varphi_p(L)\tilde{\phi}_p(L^s)\Delta^d\Delta_s^D u_t = A(t) + \theta_q(L)\tilde{\theta}_Q(L^s)\zeta_t$$

- эндогенные переменные - это наши целевые переменные. По ним мы оцениваем все компоненты временного ряда,
- экзогенные переменные - независимые переменные, дающие модели дополнительную корреляцию.

Постановка задачи обучения без учителя. Оценка качества моделей кластеризации.

Обучение без учителя - один из разделов машинного обучения. Изучает широкий класс задач обработки данных, в которых известны только описания

множества объектов (обучающей выборки), и требуется обнаружить внутренние взаимосвязи, зависимости, закономерности, существующие между объектами.

Задача обучения без учителя отличается от задачи обучения с учителем тем, что метки исходных объектов yi не заданы и могут быть неизвестны.

Цели обучения без учителя являются ответы на следующие вопросы:

- Существует информативный способ визуализации данных?
- Возможно ли сформулировать правила-ассоциации, описывающие большую часть данных?
- Можем ли мы выделить подгруппы среди переменных?

Типы входных данных:

- Признаковое описание объектов
- Матрица расстояний между объектами. Каждый объект описывается расстояниями до всех остальных объектов обучающей выборки.

Качество моделей кластеризации

Качество моделей кластеризации в метрическом пространстве:

Известные попарные расстояния между точками. Тогда у нас есть следующие метрики качества:

- Среднее внутрикластерное расстояние

$$F_0 = \frac{\sum_{i<j}[a_i=a_j]\rho(x_i,x_j)}{\sum_{i<j}[a_i=a_j]} \to min$$

- Среднее межкластерное расстояние

$$F_1 = \frac{\sum_{i<j}[a_i\neq a_j]\rho(x_i,x_j)}{\sum_{i<j}[a_i\neq a_j]} \to max$$

- Отношение пары функционалов

$$F_0/F_1 \to min$$

Качество моделей в линейном векторном пространстве:

Объекты задаются векторами. Тогда имеются следующие метрики качества:

- Сумма средних внутрикластерных расстояний:

$$\Phi_0 = \frac{1}{|X_a|}\sum_{i:a_i=a}\rho(x_i,\mu_a) \to min,$$

$X_a = \{x_i \in X^l | a_i = a\}$ - кластер a,

μ_a - центр масс класса a.

- Сумма межкластерных расстяний:

$$\Phi_0 = \sum_{a,b\in Y}\rho(\mu_a,\mu_b) \to max$$

- Отношение пары функционалов:

$$\Phi_0/\Phi_1 \to min$$

- Силуэт - показывает насколько объект похож на свой кластер по сравнению с другими кластерами. Оценка всей кластерной структуры:

$$Sil(C) = \frac{1}{N}\sum_{c_k \in C}\sum_{x_i \in c_k}\frac{b(x_i, c_k) - a(x_i, c_k)}{max\{a(x_i, c_k), b(x_i, c_k)\}},$$

где:

$a(x_i, c_k) = \frac{1}{|c_k|}\sum_{x_j \in c_k}\|x_i - x_j\|$ — среднее расстояние от $x_i \in c_k$ до других объектов из кластера c_k (компактность),

$b(x_i, c_k) = min_{c_l \in C \setminus c_k}\{\frac{1}{|c_l|}\sum_{x_j \in c_l}\|x_i - x_j\|\}$ — среднее расстояние от $x_i \in c_k$ до объектов из другого кластера $c_l : k \neq l$ (отделимость).

Можно заметить, что

$$-1 \leq Sil(C) \leq 1.$$

Чем ближе данная оценка к 1, тем лучше.

Есть также упрощенная вариация силуэта: $a(x_i, c_k)$ и $b(x_i, c_k)$ вычисляются через центры кластеров.

- Индекс Калински-Харабаш

$$CH(C) = \frac{N - K}{K - 1} \cdot \frac{\sum_{c_k \in C}|c_k| \cdot \|\overline{c_k} - \overline{X}\|}{\sum_{c_k \in C}\sum_{x_i \in c_k}\|x_i - \overline{c_k}\|}$$

Компактность основана на расстоянии от точек кластера до их центроидов, а разделимость - на расстоянии от центроид кластеров до глобального центроида. Должен возрастать.

- Индекс Дэвида-Болдуина - вычисляет компактность как расстояние от объектов кластера до их центров, а отделимость - как расстояние между центроидами.

$$DB(C) = \frac{1}{K}\sum_{c_k \in C}\max_{c_l \in C \setminus c_k}\left\{\frac{S(c_k) + S(c_l)}{\|\overline{c_k} - \overline{c_l}\|}\right\},$$

где:

$$S(c_k) = \frac{1}{|c_k|}\sum_{x_i \in c_k}\|x_i - \overline{c_k}\|$$

Существует еще одна вариация данной меры, которая была предложена автором вместе с основной версией:

$$DB^*(C) = \frac{1}{K}\sum_{c_k \in C}\frac{\max_{c_l \in C \setminus c_k}\{S(c_k) + S(c_l)\}}{\min_{c_l \in C \setminus c_k}\{\|\overline{c_k} - \overline{c_l}\|\}}$$

C-индекс и индекс Дэвиса-Болдуина должны минимизироваться для роста кластеризации.

- Индекс Данна

$$D(C) = \frac{min_{c_k \in C}\{min_{c_l \in C \setminus c_k}\{\delta(c_k, c_l)\}\}}{max_{c_k \in C}\{\Delta(c_k)\}},$$

где:

δ — межкластерное расстояние (оценка разделения), $\delta(c_k, c_l) = min_{x_i \in c_k, x_j \in c_l}\|x_i - x_j\|$,

$\Delta(c_k)$ — диаметр кластера (оценка сплоченности), $\Delta(c_k) = max_{x_i, x_j \in c_k}\|x_i - x_j\|$.

Обучение без учителя, кластеризация. K-Means.

Задача кластеризации:

Дана обучающая выборка объектов из некоторого пространства и функция расстояния между объектами в этом пространстве.

Необходимо найти множество кластеров и алгоритм кластеризации такие, что каждый кластер состоит из близких объектов, а объекты из разных кластеров существенно различны.

Решение задачи кластеризации принципиально неоднозначно:

- точной постановки задачи кластеризации нет,
- существует много критериев качества кластеризации,
- существует много эвристических методов кластеризации,
- число кластеров, как правило, неизвестно заранее,
- результат кластеризации сильно зависит от метрики, выбор которой тоже является эвристикой.

Цели кластеризации:

- Упростить дальнейшую обработку данных, разбить множество объектов обучающей выборки на группы схожих объектов чтобы работать с каждой группой по отдельности (задачи классификации, регрессии, прогнозирования).
- Сократить объем хранимых данных, оставив по одному представителю от каждого кластера (задачи сжатия данных).
- Выделить нетипичные объекты, которые не подходят ни к одному из кластеров(задачи одноклассовой классификации).
- Построить иерархию множества объектов(задачи таксономии).

Метрики качества кластеризации описаны ранее.

K-means

Это метод для разбиения немаркированного набора данных на K различных неперекрывающихся кластеров.

Принцип работы: на каждой итерации перевыбирается центр-масс(центроида) для каждого кластера, полученного на предыдущем шаге, затем векторы разбиваются на кластеры вновь в соответствии с тем, какой из новых центров оказался ближе по выбранной метрике. Алгоритм завершается, когда на какой-то итерации не происходит изменения внутрикластерного расстояния. Это происходит за конечное число итераций, так как количество возможных разбиений конечного множества конечно, а на каждом шаге суммарное квадратичное отклонение уменьшается, поэтому зацикливание невозможно.

Задача оптимизации ставится как задача минимизации суммы квадратов внутрикластерных расстояний:

$$\sum_{i=1}^{l} ||x_i - \mu_{a_i}||^2 \to min_{\{a_i\},\{\mu_a\}}, ||x_i - \mu_a||^2 = \sum_{j=1}^{n} (f_j(x_i) - \mu_{a_i})^2$$

Межкластерные расстояния в этом алгоритме не используются.

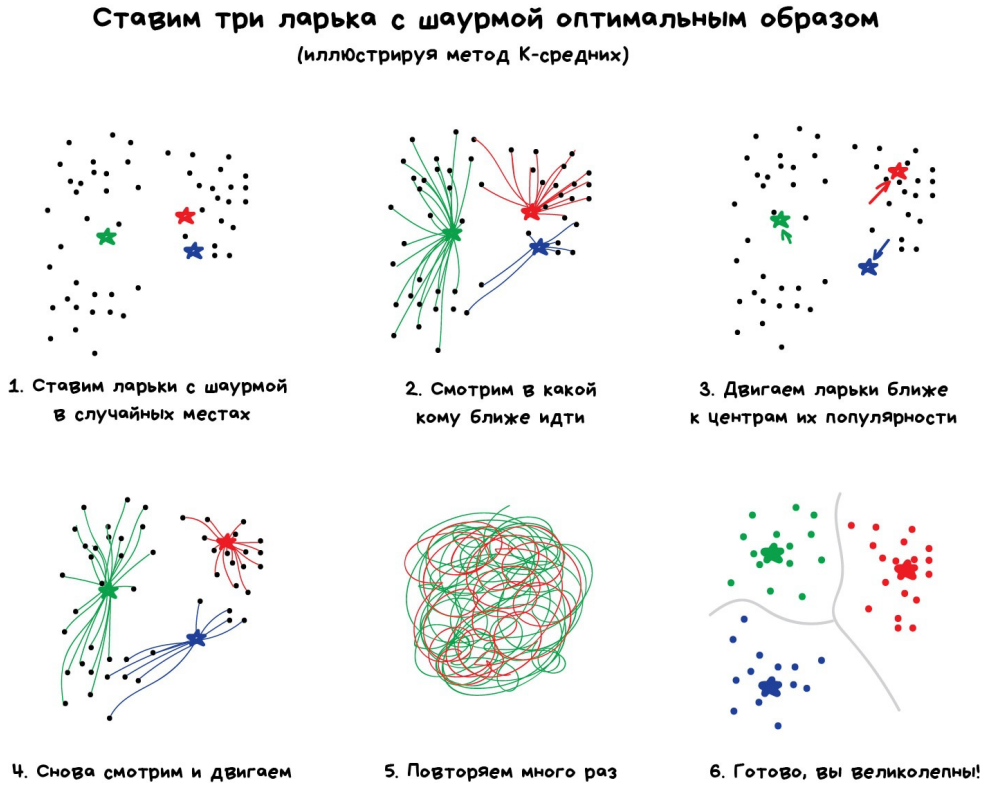
Плюсы:

- Простота реализации.
- Масштабируемость до огромных наборов данных.
- Метод очень быстро обучается на новых примерах.
- Поддержка сложных форм и размеров.

Минусы:

- Необходимо знать число кластеров k, а задающего это число можно назвать учителем.
- Чувствительность к выбросам.
- Не гарантируется достижение глобального минимума.

- Сильная зависимость от изначального выбора центров кластеров.



Иерархическая кластеризация. Метрики.

Иерархическая кластеризация — совокупность алгоритмов упорядочивания данных, направленных на создание иерархии (дерева) вложенных кластеров. Выделяют два класса методов иерархической кластеризации:

- Агломеративные методы: новые кластеры создаются путем объединения более мелких кластеров и, таким образом, дерево создается от листьев к стволу;
- Дивизивные или дивизионные методы: новые кластеры создаются путем деления более крупных кластеров на более мелкие и, таким образом, дерево создается от ствола к листьям.

Агломеративная иерархическая кластеризация

Источник - Википедия

Главная особенность метода агломеративной иерархической кластеризации состоит в том, что для изменения желаемого числа кластеров запускать заново алгоритм не нужно. Для этого достаточно рассмотреть дерево объединения кластеров и обрезать его на желаемом шаге.

Дерево строится от листьев к корню. В начальный момент времени каждый объект содержится в собственном кластере. Далее происходит итеративный процесс слияния двух ближайших кластеров до тех пор, пока все кластеры не объединятся в один или не будет найдено необходимое число кластеров. На каждом шаге

необходимо уметь вычислять расстояние между кластерами и пересчитывать расстояние между новыми кластерами. Расстояние между одноэлементными кластерами определяется через расстояние между объектами.

Способом вычисления расстояния между не одноэлементными кластерами уже и будет определяться конкретный алгоритм.

Виды функций расстояния между кластерами

- одиночная связь - минимальное расстояние между элементами кластеров $R_{min}(U,V) = min_{u\in U, v\in V}\rho(u,v)$
- полная связь - максимальное расстояние между элементами кластеров $R_{max}(U,V) = max_{u\in U, v\in V}\rho(u,v)$
- средняя связь - среднее расстояние между всеми парами элементов кластеров

  $R_{avg}(U,V) = \frac{1}{|U||V|}\sum_{u\in U}\sum_{v\in V}\rho(u,v)$
- центроидная - расстояние между центром масс (центроидами) кластеров (не удовлетворяет теореме Миллигана)

  $R_c(U,V) = \rho^2(\sum_{u\in U}\frac{u}{|U|}, \sum_{v\in V}\frac{v}{|V|})$
- метод Уорда - прирост дисперсии после объединения

  $R_{ward}(U,V) = \frac{|U||V|}{|U|+|V|}\rho^2(\sum_{u\in U}\frac{u}{|U|}, \sum_{v\in V}\frac{v}{|V|})$

Формула Ланса-Уильямса:

На каждом шаге необходимо уметь быстро подсчитывать расстояние от образовавшегося кластера $W = U \cup V$ до любого другого кластера S, используя известные расстояния с предыдущих шагов:
$R(W,S) = \alpha_U R(U,S) + \alpha_V R(V,S) + \beta R(U,V) + \gamma|R(U,S) - R(V,S)|$, где α_U, α_V, β, γ - числовые параметры.

Каждая из указанных выше функций расстояния это формула Ланса–Уильямса с определенными коэффициентами.

Функция расстояния является ***монотонной***, если на каждом следующем шаге расстояние между кластерами не уменьшается: $R_2 \le R_3 \le \ldots \le R_m$

Свойства расстояний между кластерами, теорема Миллигана:

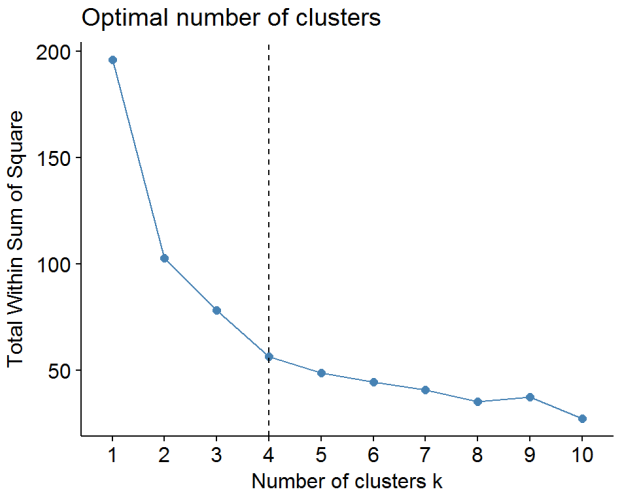
Кластеризация является монотонной (то есть расстояние между кластерами не уменьшается на каждом следующем шаге), если выполняются следующие условия:

- $\alpha_U \ge 0,\ \alpha_V \ge 0$
- $\alpha_U + \alpha_V + \beta \ge 1$
- $min\{\alpha_U, \alpha_V\} + \gamma \ge 0$

Если ***количество кластеров*** не задано, то оно определяется по расстоянию между объединяемыми кластерами. Наиболее популярный способ - эвристический подход: метод локтя.

По оси *x* - число кластеров, а по *y* - расстояние между объединяемыми кластерами (или некоторая метрика качества кластеризации (взятая с минусом на картинке ниже)). Останавливаемся на том числе кластеров, когда график начинает расти слишком быстро. при чем это ”слишком” определяется безо всякого конкретного алгоритма, а просто на глаз.

Есть более строго формализованные подходы. Например аппроксимация графика локтя линейной и параболической функциями. Останавливаем кластеризацию, когда вторая аппроксимирует лучше.



Дивизимная иерархическая кластеризация

Дивизимные кластерные алгоритмы на первом шаге представляют все множество элементов как единственный кластер. На каждом шаге алгоритма один из существующих кластеров рекурсивно делится на два дочерних. Таким образом, итерационно образуются кластеры сверху вниз. Его применяют, когда необходимо разделить все множество объектов на относительно небольшое количество кластеров или когда необходимо распределить объекты для дальней обработки (как пример в задаче маршрутизации транспорта).

Алгоритм

Есть 2 способа разделения одного кластера на 2 меньших:

- Выбираем из разделяемого кластера элемент, среднее расстояние от которого до всех остальных максимально. На его основе будет формировать второй кластер. Далее все элементы, для которых расстояние до центра второго кластера меньше, чем до центра первого, переносятся во второй.
- Второй способ немного проще с вычислительной точки зрения. Случайно выбирается элемент из разделяемого кластера. Находится наиболее удаленный элемент. 2 выбранных элемента будут формировать наши 2 кластера. Все остальные элементы будут относится к тому кластеру, расстояние до формирующего элемента которого меньше. Здесь на надо пересчитывать центроиды или считать расстояние между всеми элементами кластера, как в предыдущем.

Есть несколько способов выбора кластера для разбиения:

- По оценке дисперсии или плотности (разбивается кластер с наибольшей дисперсией, с наименьшей плотностью).
- Пока размер кластера не будет удовлетворительным (то есть берём все кластеры, размеры которых больше определённого порога).

Остановка алгоритма происходит, когда мы достигли необходимого числа кластеров или не можем выбрать кластер для дальнейшего разбиения.

Метрики расстояний между точками

- Квадрат Евклидова расстояния
  $\rho(x, y) = \sum_{i}^{n}(x_i - y_i)^2$
- Манхэттенское расстояние
  $\rho(x, y) = \sum_{i}^{n}|x_i - y_i|$
- Расстояние Чебышева
  $\rho(x, y) = max(|x_i - y_i|)$

DBSCAN

Источник - Википедия

Это алгоритм кластеризации, основанной на плотности — если дан набор точек в некотором пространстве, алгоритм группирует вместе точки, которые тесно расположены, помечая как выбросы точки, которые находятся одиноко в областях с малой плотностью (ближайшие соседи которых лежат далеко).

Для выполнения кластеризации DBSCAN точки делятся на *основные* точки, *достижимые* по плотности точки и *выпадающие* следующим образом:

- Точка *p* является *основной* точкой, если по меньшей мере *minPts* точек находятся на расстоянии, не превосходящем ϵ (ϵ - максимальный радиусом соседства от *p*) до неё (включая саму *p*). Говорят, что эти точки *достижимы* прямо из *p*.
- Точка *q прямо достижима* из *p,* если точка *q* находится на расстоянии, не большем ϵ, от точки *p* и *p* является основной точкой.
- Точка *q достижима* из *p*, если имеется путь $p_1, p_2, ..., p_n$, в котором $p_1 = p$, $p_n = q$ и каждая точка p_{i+1} достижима прямо из p_i. При этом все точки на пути должны быть основными, за исключением q.
- *Выбросы* - точки, не достижимые из основных точек.

Таким образом, если *p* является основной точкой, то она формирует *кластер* вместе со всеми точками, достижимыми из этой точки. Каждый кластер содержит по меньшей мере одну основную точку. Неосновные точки могут быть частью кластера, но они формируют его «край», поскольку не могут быть использованы для достижения других точек.

Достижимость не является симметричным отношением, поскольку, по определению, никакая точка не может быть достигнута из неосновной точки, независимо от расстояния (так что неосновная точка может быть достижимой, но ничто не может быть достигнуто из неё). Поэтому дальнейшее понятие связности необходимо для формального определения области кластеров, найденных алгоритмом DBSCAN. Две точки *p* и *q* *связаны по плотности*, если имеется точка *o*, такая что и *p*, и *q* достижимы из *o*. Связность по плотности является *симметричной*. Тогда кластер удовлетворяет двум свойствам:

- Все точки в кластере попарно связны по плотности.
- Если точка достижима по плотности из какой-то точки кластера, она также принадлежит кластеру.

Алгоритм требует указания ϵ и минимального числа точек, которые должны образовывать плотную область *minPts.*

Выбирается ϵ-окрестность точки и, если она содержит достаточно много точек, образуется кластер, в противном случае точка помечается как шум. Заметим, что эта точка может быть позже найдена в ϵ-окрестности другой точки и включена в какой-то кластер.

Если точка найдена как плотная точка кластера, её ϵ-окрестность также является частью этого кластера. Следовательно, все точки, найденные в ϵ-окрестности этой точки, добавляются к кластеру. Этот процесс продолжается, пока не будет найден связный по плотности кластер. Затем выбирается и обрабатывается новая непосещенные точка, что ведёт к обнаружению следующего кластера или шума.

DBSCAN может быть использован с любой функцией расстояния, а так же с функцией похожести или логическим условием. Функция расстояния может рассматриваться как дополнительный параметр.

Плюсы:

- DBSCAN не требует спецификации числа кластеров в данных априори в отличие от метода k-средних.
- DBSCAN может найти кластеры произвольной формы. Он может найти даже кластеры полностью окружённые (но не связанные с) другими кластерами. Благодаря параметру minPts уменьшается так называемый эффект одной связи (связь различных кластеров тонкой линией точек).
- DBSCAN имеет понятие шума и устойчив к выбросам.
- DBSCAN требует лишь двух параметров и большей частью нечувствителен к порядку точек в базе данных. (Однако, точки, находящиеся на границе двух различных кластеров могут оказаться в другом кластере, если изменить порядок точек, а назначение кластеров единственно с точностью до изоморфизма.)
- DBSCAN разработан для применения с базами данных, которые позволяют ускорить запросы в диапазоне значений, например, с помощью R*-дерева.
- Параметры *minPts* и ϵ могут быть установлены экспертами в рассматриваемой области, если данные хорошо понимаются.

Минусы:

- DBSCAN не полностью однозначен — краевые точки, которые могут быть достигнуты из более чем одного кластера, могут принадлежать любому из этих кластеров, что зависит от порядка просмотра точек. Для большинства наборов

данных эти ситуации возникают редко и имеют малое влияние на результат кластеризации — основные точки и шум DBSCAN обрабатывает однозначно. DBSCAN является вариантом, который трактует краевые точки как шум и тем самым достигается полностью однозначный результат, а также более согласованная статистическая интерпретация связных по плотности компонент.

- Качество DBSCAN зависит от измерения расстояния, используемого в функции regionQuery(P,ε). Наиболее часто используемой метрикой расстояний является евклидова метрика. Особенно для кластеризации данных высокой размерности эта метрика может оказаться почти бесполезной ввиду так называемого «проклятия размерности», что делает трудным делом нахождение подходящего значения ϵ. Этот эффект, однако, присутствует в любом другом алгоритме, основанном на евклидовом расстоянии.
- DBSCAN не может хорошо кластеризовать наборы данных с большой разницей в плотности, поскольку не удается выбрать приемлемую для всех кластеров комбинацию $minPts - \epsilon$.
- Если данные и масштаб не вполне хорошо поняты, выбор осмысленного порога расстояния ϵ может оказаться трудным.

Spectral Clustering

Источник - [MachineLearningMastery]

Техники *спектральной кластеризации* используют спектр (собственные значения) матрицы сходства данных для осуществления снижения размерности перед кластеризацией в пространствах меньших размерностей. Матрица сходства подаётся в качестве входа и состоит из количественных оценок относительной схожести каждой пары точек в данных.

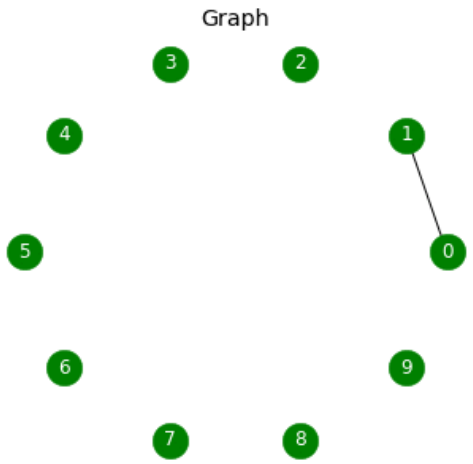
(Если для матрицы A, если существует ненулевой вектор x и скалярный λ такой, что Ax = λx, то x называется собственным вектором A с соответствующим собственным значением λ.)

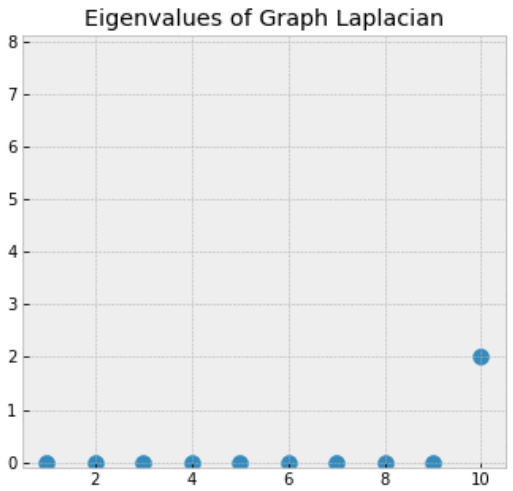
Представляем данные в виде графа, а именно в виде матрицы смежности *A*: $a_{ij} = 1$ если есть ребро между вершинами *i* и *j,* иначе 0.

Также построим матрицу степеней *D*, где на диагонали стоят степени соответствующих вершин.

Затем по построенным матрицам находим матрицу Лапласа (Кирхгофа): *L=D-A*. $k_{ij} = \{deg(v_i),\ \text{при}\ i = j;\ \ -1\ \text{при}\ (v_i, v_j) \in E(G);\ 0\ \text{иначе}\}$.

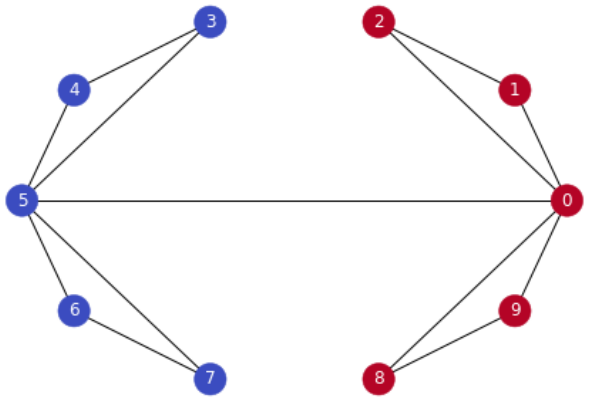
Матрица Лапласа обладает важными свойствами. Когда граф не полностью связан все собственные числа равны 0. При добавлении ребер некоторые собственные числа увеличиваются - количество нулевых собственных чисел соответствует числу связных компонент в графе.





Первое ненулевое собственное значение - *спектральная щель*. Это значение дает нам некоторое представление о плотности графика. Если бы этот граф был плотно связан (все пары из 10 узлов имели ребро), то спектральный разрыв был бы равен 10.

Второе собственное значение называется *значением Фидлера*, а соответствующий ему вектор - *вектор Фидлера*. Значение Фидлера аппроксимирует минимальный разрез графа, необходимый для разделения графа на две связные компоненты. Если бы наш граф уже состоял из двух связанных компонент, то значение Фидлера было бы равно 0. Каждое значение в векторе Фидлера дает нам информацию о том, какой стороне разреза принадлежит этот узел. Давайте покрасим узлы в зависимости от того, является ли их вход в вектор Фидлера положительным или нет:



Этот приём делит граф на два кластера. Нулевые собственные значения представляют собой связанные компоненты. Собственные значения, близкие к нулю, говорят нам, что существует почти разделение двух компонентов, а второе собственное значение как раз не велико.

Для приведенного примера третье и четвертое собственные значения также довольно малы. Это говорит о том, что можно разделить граф еще два раза и получить 4 кластера. Таким образом мы ищем первый большой разрыв между собственными значениями, чтобы найти число кластеров, выраженных в наших данных. Наличие четырех собственных значений перед разрывом указывает на то, что существует, вероятно, четыре кластера. Векторы, связанные с первыми тремя положительными собственными значениями, должны дать нам информацию о том, какие три разреза

необходимо сделать в графе, чтобы назначить каждый узел одной из четырех аппроксимируемых компонент.

Наконец выполним K-Means на этих векторах, чтобы получить метки для узлов.

Вывод

Сначала мы строим граф, описывающий наши данные. Ребра графа фиксируют сходство между точками. Собственные значения матрицы лапласа могут быть использованы для нахождения наилучшего числа кластеров, а собственные векторы-для нахождения фактических меток кластеров.