Tinkoff Generation ML конспекты

МЛЬ БОТАЛКА
Гарипов Роман
Бородачев Сергей
Константинова Елизавета
Степанов Никита

2021
Февраль – Май

Содержание

1 Задачи машинного обучения

Supervised и unsupervised. Регрессия, классификация, кластеризация. Метрики качества. Рассмотрим типы задач, которые встречаются в машинном обучении.

1.1 Классификация

Рассмотрим следующую задачу. Мы работаем в банке и нам необходимо, чтобы система определяла, сможет ли человек вовремя погасить кредит или нет.

У нас есть различная информация о человеке, так называемые **признаки (features)**. Они бывают различных типов:

- **Бинарные признаки**: наличие телефона.
- **Номинальные признаки**: профессия, адрес проживания.
- **Порядковые признаки**: образование, занимаемая должность. По сути, они похожи на номинальные признаки, только тут у значений признака имеется порядок (высшее образование ценится выше среднего и т.д.).
- **Количественные признаки**: зарплата, возраст, количество детей в семье и т.п.

Ответ же к данной задаче либо **0** (человек не выплатит кредит вовремя) и **1** (Человек выплатит кредит вовремя).

Т.е. в задаче классификации значение, которое мы хотим предсказать (далее целевая переменная), принадлежит конечному множеству. В нашей задаче, например, это **{0; 1}**.

Бывают множества не только из 2 классов. Например, задача диагностики болезни по симптомам. Тут классы — это список болезней.

Как решается данная задача? В таких задачах нам требуется **обучающая выборка**. Это история того, как в прошлом наши клиенты выплачивали кредиты.

Мы знаем о них все: где они работают, сколько получают и т.п. А также нам известно смогли они выплатить кредит или нет.

Знание того, что мы предсказываем (целевой переменной) относит задачу к **обучению с учителем**.

Далее модель машинного обучения находит закономерности в этой выборке. Например, если человек безработный, то скорее всего, он не выплатит кредит вовремя и т.д.

Она запоминает эти закономерности, и когда приходит черёд узнать, а сможет ли новый клиент заплатить кредит вовремя, модель смотрит на эти зависимости и выдаёт ответ.

Список новых клиентов называется **тестовой выборкой**. Главное её отличие от обучающей выборки заключается в том, что для элементов из тестовой выборки неизвестна целевая переменная (в нашем случае — это выплатит клиент кредит или нет).

1.2 Регрессия

Еще одним типом является **регрессия**.

Например, можно рассмотреть такую задачу: мы хотим по росту родителей определить насколько высоким может быть их ребёнок. Действительно, как правило, есть зависимость, что у высоких родителей дети тоже имеют высокий рост и наоборот.

Т.е. от классификации задача регрессии отличается тем, что тут целевая переменная — вещественное число.

Обучающей выборкой в данной задаче будет набор троек: (рост родителя №1, рост родителя №2, рост ребёнка).

Тестовой выборкой — набор двоек: рост родителя №1 и рост родителя №2.

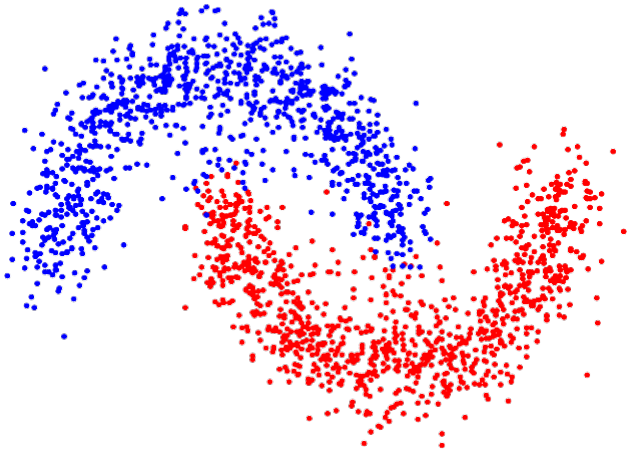
1.3 Кластеризация

В обучающих выборках, рассмотренных выше задач, нам была известна целевая переменная:

- Выплатит ли человек кредит вовремя — в задаче классификации
- Рост ребенка — в задаче регрессии.

Однако, целевая переменная не всегда известна. Например, мы провели социологический опрос, у нас есть много ответов на вопросы и нам хотелось бы сгруппировать их по поведению. Мы заранее не знаем, сколько таких кластеров получится и что они из себя представляют. Это мы можем узнать только после того как появятся сами кластеры.

Еще простой и пробирочной задачей кластеризации является кластеризация точек на плоскости



2 Приближение функции многочленами

2.1 Задача, модель и функция ошибки

1. Допустим, у нас есть функция от одной переменной $f(x)$.
2. Мы знаем ее значения на каком-то множестве точек (назовём его **тренировочным множеством**) с некоторым отклонением (шумом) $\varepsilon(x)$
3. Хотим научиться восстанавливать (хотя бы приблизительно) эту функцию.

Мы хотим найти набор коэффициентов $(\overline{a}_0, \overline{a}_1, \ldots, \overline{a}_n)$, которые «приближают» исходную функцию. Пусть

$$P(x) = \overline{a}_0 + \overline{a}_1 x + \overline{a}_2 x^2 + \ldots + \overline{a}_n x^n \approx f(x)$$

Из-за того, что значения из тренировочного множества неидеально приближают исходную функцию, мы не можем точно определить искомый набор коэффициентов.

Эта проблема подводит нас к новому определению. Введём новую функцию $L(P)$, с помощью которой будем определять, насколько хорошо наша «модель» $P(x)$ приближает исходную функцию $f(x)$.

Назовем $L(P)$ **функцией ошибки** или **функцией потерь**. Будем считать, что чем меньше $L(P)$, тем лучше наша «модель» приближает $f(x)$.

Таким образом, минимизируя $L(P)$ будем получать наилучшее приближение функции $f(x)$

Рассмотрим самые часто используемые функции потерь:

$$L(P) = \frac{1}{n}\sum_{i=0}^{n}(P(x_i) - f(x_i))^2 \ \text{— MSE}$$

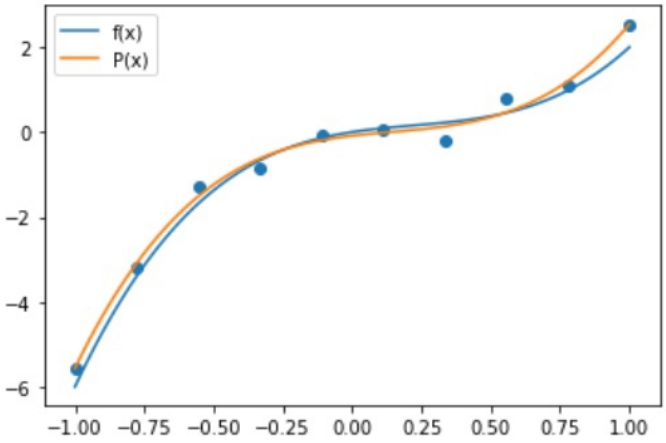
$$L(P) = \sqrt{\frac{1}{n}\sum_{i=0}^{n}(P(x_i) - f(x_i))^2} \ \text{— RMSE}$$

$$L(P) = \frac{1}{n}\sum_{i=0}^{n}|P(x_i) - f(x_i)| \ \text{— L1 loss}$$

2.2 Переобучение

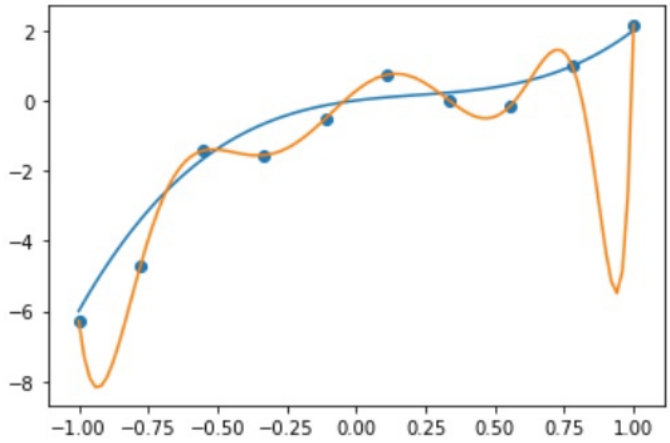
Рассмотрим простую функцию $f(x) = 3x^3 - 2x^2 + x + \varepsilon(x)$:

Давайте попробуем приблизить эту функцию многочленом третьей степени:



Видно, что мы получили довольно хорошее приближение исходной функции

Попробуем увеличить степень нашей модели до шестнадцати:



Видно, что модель стала хуже предсказывать исходную зависимость, но при этом на тренировочном множестве выдает идеальные ответы.

Мы столкнулись с проблемой **переобучения (Overfitting)**.

2.3 Разбиение на обучающую и тестовую выборку и кросс-валидация

Как узнать, находит ли исходную зависимость наша модель или переобучается?

Обычно пользуются одним из двух подходов:

1. **Разбиение на обучающую и тестовую выборку (train test split)**

   Прежде чем обучать модель, разобьем исходное тренировочное множество на обучающую и валидационную (тестовую) выборки.

   Теперь, если мы обучим модель на обучающей выборке, то сможем сравнить значение функции потерь на валидационной выборке, объекты из которой модель не видела при обучении, и на обучающей.

   Если эти значения сильно отличаются, то можно говорить о переобучении нашей модели.

2. **Кросс-валидация** Разобьем исходное тренировочное множество на k примерно равных частей. Теперь поочередно обучим k моделей на $k-1$, каждый раз выкидывая новую часть. Если мы посчитаем функцию ошибки для каждой модели на частях, которые не были включены при обучении, а затем усредним, то получим более объективное значение функции ошибки.

3 Линейная регрессия

3.1 Задача и модель линейной регрессии

Допустим, наша модель (зависимость) имеет вид

$$f(x_1, x_2, ...x_n) = a_0 + a_1x_1 + a_2x_2 + ... + a_nx_n$$

Мы хотим подобрать такие коэффициенты $a_0, a_1, ...a_n$, что какая-то функция потерь на наших данных была минимальна. Для линейной регрессии в качестве функции потерь используют MSE. Формально наша задача имеет вид:

$$\sum_{i=1}^{n} \left(f\left(x_{i1}, x_{i2}, x_{i3}, ...x_{in}\right) - yi\right)^2 \to min$$
$$\sum_{i=1}^{n} \left(a_0 + a_1x_{i1} + a_2x_{i2} + a_3x_{i3} + ... + x_{in} - y_i\right)^2 \to min$$

Если решать эту задачу, посчитав производные, то мы получим систему из n линейных уравнений, решение которой и определяет коэффициенты $a_0, a_1, ...a_n$, при которых функция потерь минимальна.

У непрерывно-дифференцируемой функции, которая при стремлении по каждой координате к плюс или минус бесконечности сама стремится к плюс бесконечности, всегда существует глобальный минимум. В точке глобального минимума все производные как раз равны нулю. Следовательно, существует всегда хотя бы одно решение, и мы его найдем.

3.2 Регуляризация.

$b = (\beta_0, \beta_1, ...\beta_n)$ тут это то же самое, что и в предыдущем пункте $(a_0, a_1, ...a_n)$

Регуляризация - это искусственное занижение всех параметров моделей $(\beta_0, \beta_1, ...)$. Это помогает упростить модель, т.к. низкие значения β_i означают, что модель будет выдавать результаты близкие к прямой (или гиперплоскости, если параметров много). А также зануление некоторых коэффициентов может убрать ненужные признаки из обучения. Однако как всё это сделать так, чтобы не убрать нужные признаки?

Посмотрим на функцию потерь, которую мы минимизируем:

$$L(\beta) = \sum_{i=1}^{N} (y_i^{true} - y_i^{pred})^2$$

Добавим туда слагаемое (это L2-регуляризация), которое поможет нам понизить значения наших β_i:

$$L(\beta) = \sum_{i=1}^{N}(y_i^{true} - y_i^{pred})^2 + \lambda \sum_{i=1}^{m} \beta_i^2$$

Действительно, если мы будем понижать нашу функцию потерь $L(\beta)$, то мы будем минимизировать и второе слагаемое, которое и отвечает за абсолютные величины β_i. Осталось указать, что λ это некоторый числовой коэффициент, который позволяет играться между очень сильной и очень слабой регуляризацией. Если он большой, то регуляризация сильная и модель будет более простой, меньше переобучаться, больше недообучаться, и наоборот.

L1-регуляризация выглядит так:

$$L(\beta) = \sum_{i=1}^{N}(y_i^{true} - y_i^{pred})^2 + \lambda \sum_{i=1}^{m} |\beta_i|$$

Она отличается от L2 тем, что она обнуляет некоторые коэффициенты, а L2 пытается всё уменьшать равномерно.

В sklearn L1-регуляризация - это Lasso, а L2-регуляризация - это Ridge.

4 Логистическая регрессия

4.1 Задача

В прошлой «главе» мы научились решать задачи *регрессии* с помощью *линейной регресии*. Возникает очевидный вопрос: как же тогда решать задачу *классификации*? Попробуем обобщить модель *линейной регрессии* на *классификацию*.

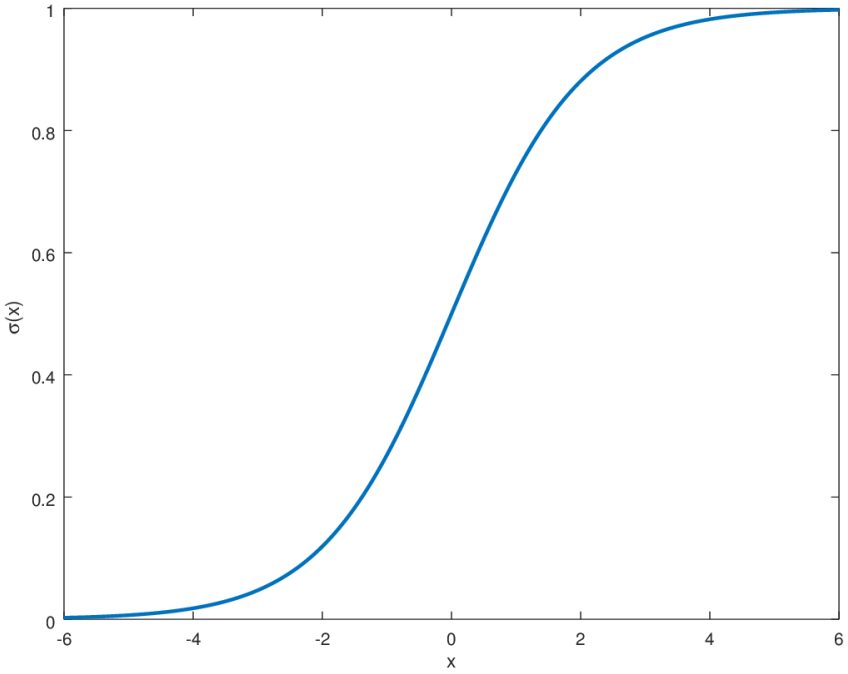
4.2 Модель

Давайте, для начала, рассмотрим задачу *бинарной классификации*.

Что будет, если мы просто предскажем 0 или 1 с помощью *линейной регрессии*? Скажем, что значения, большие 0.5 относятся к *первому классу*, меньшие - к *второму* классу.

При таком подходе модель никак не учитывает свою *«уверенность»* в ответе. Мы не увидим отличие, между значением, равным 0.5 и равным 1000. Хотелось бы научиться получать **вероятность принадлежности к классу 1**.

Давайте преобразовывать ответ модели, который находится в интервале $(-\infty; +\infty)$ в ответ из интервала $(0; 1)$. Такие преобразования называются **сигмоидами**.



Обычно используют сигмоиду $\sigma(y) = \frac{1}{1+e^{-y}}$.

4.3 Функция потерь

Так как у нашей модели изменились ответы, нам необходимо поменять функцию ошибки. Давайте посмотрим на предсказания модели и вычислим вероятность того, что модель «угадает» классы для всех объектов.

$$P_{all} = \prod_{i|y_i=1} p_i \prod_{i|y_i=0} (1 - p_i)$$

Нам необходимо максимизировать эту вероятность.

Работать с произведением неудобно, поэтому возьмем логарифм.

$$\ln P_{all} = \sum_{i|y_i=1} \ln p_i + \sum_{i|y_i=0} \ln(1 - p_i)$$

Если взять данное значение с отрицанием, получится наша новая функция потерь - **Logloss**.

$$Logloss = -\ln P_{all} = -\sum_{i|y_i=1} \ln p_i - \sum_{i|y_i=0} \ln(1 - p_i)$$

Описанный алгоритм называется **Логистическая регрессия** и служит для бинарной классификации.

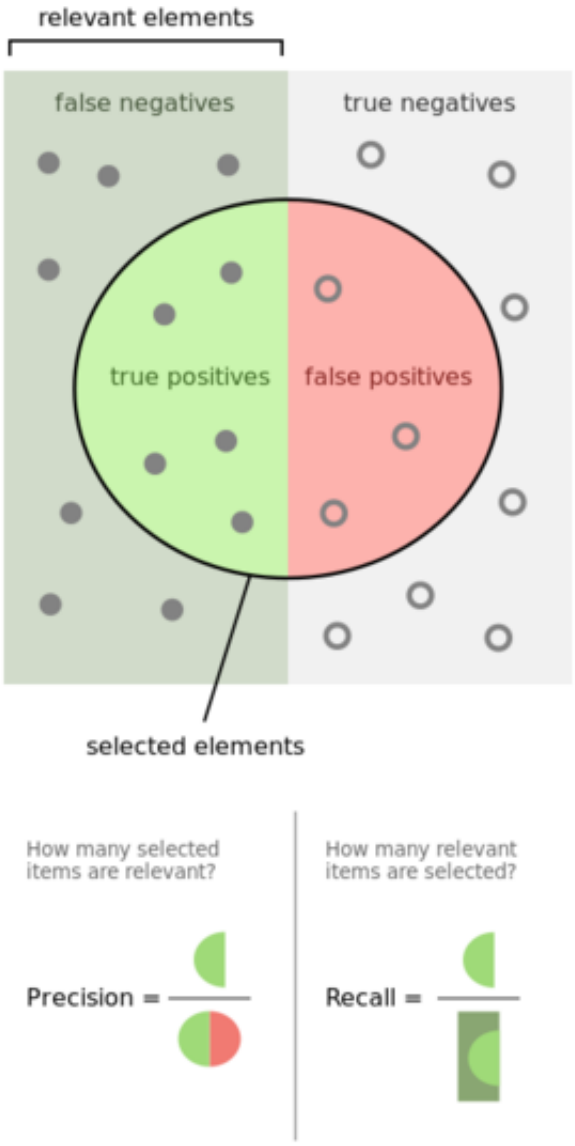
4.4 Мультиклассовая классификация

Если мы имеем задачу мультиклассовой классификации с **n** классами, давайте обучим n логистических регрессий, в i-й логистической регресии будем предсказывать вероятность, что объект относится к классу i.

Как ответ можно взять класс с максимальной вероятностью.

5 Метрики классификации

Ошибки классификации бывают **False Positive** и **False Negative**. В статистике первый вид ошибок называют ошибкой I-го рода, а второй — ошибкой II-го рода.



5.1 Accuracy

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

Accuracy - это доля правильных ответов алгоритма. Эта метрика бесполезна в задачах с неравными классами.

5.2 Precision

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

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

5.3 Recall

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

Recall - это доля объектов положительного класса из всех объектов положительного класса.

Recall демонстрирует способность алгоритма обнаруживать данный класс вообще, а *precision* — способность отличать этот класс от других классов.

5.4 F - мера

$$F_\beta = \left(1 + \beta^2\right) \cdot \frac{\text{precision } \cdot \text{ recall}}{(\beta^2 \cdot \text{ precision }) + \text{ recall}}$$

F-мера (в общем случае F_β) — среднее гармоническое *precision* и *recall*.

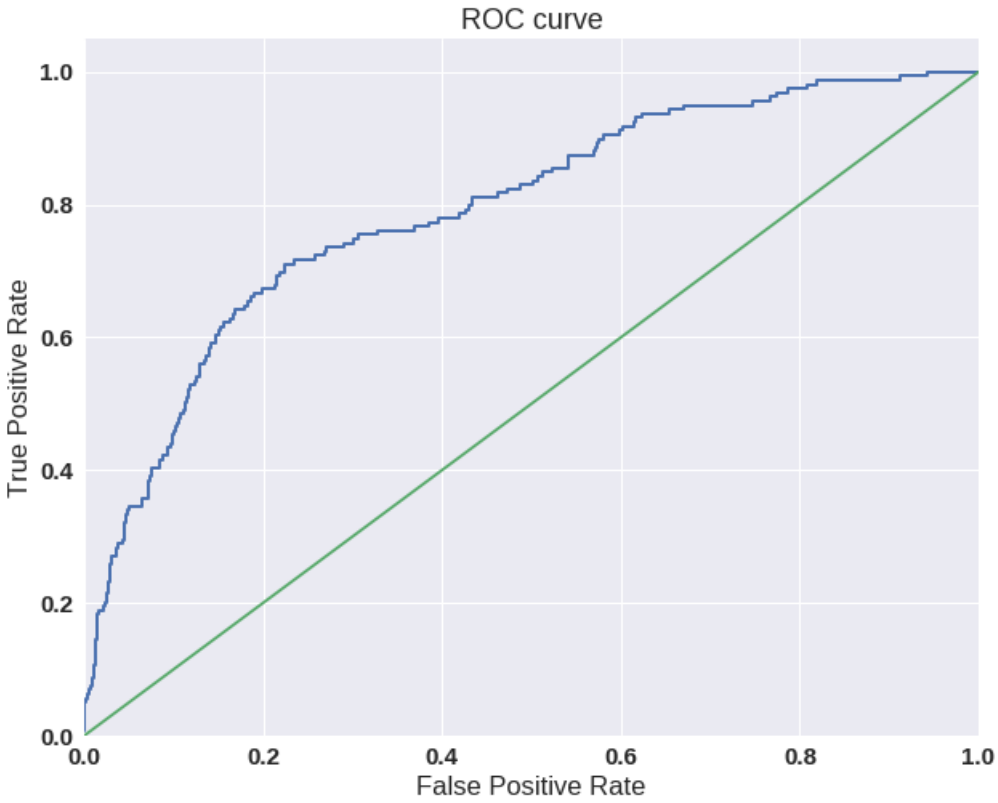
β в данном случае определяет вес точности в метрике, и при $\beta = 1$ это среднее гармоническое (с множителем 2, чтобы в случае *precision* $= 1$ и *recall* $= 1$ иметь $F_1 = 1$).

F-мера достигает максимума при полноте и точности, равными единице, и близка к нулю, если один из аргументов близок к нулю (гармоническое среднее, в отличие от арифметического, стремится к нулю, когда хотя бы одно из значений стремится к нулю).

5.5 ROC-AUC

ROC AUC — площадь (Area Under Curve) под кривой ошибок (Receiver Operating Characteristic curve). Данная кривая представляет из себя линию от (0,0) до (1,1) в координатах *True Positive Rate (TPR)* и *False Positive Rate (FPR)*:

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



5.6 LogLoss

$$\text{logloss } = -\frac{1}{l} \cdot \sum_{i=1}^{l} \left(y_i \cdot \log\left(\hat{y}_i\right) + (1 - y_i) \cdot \log\left(1 - \hat{y}_i\right)\right)$$

Здесь $\hat{y}$ - это ответ алгоритма на i -ом объекте, $y-$ истинная метка класса на i -ом объекте, а l размер выборки.

Можно представить минимизацию *logloss* как задачу максимизации *accuracy* путем штрафа за неверные предсказания. Однако *logloss* крайне сильно штрафует за уверенность классификатора в неверном ответе.

5.7 Мультиклассовые задачи

Основные подходы сведения мультиклассовой задачи к бинарным: **one-vs-all** (K бинарных классификаторов для каждого класса против всего

остального) и **all-vs-all** (C_K^2 классификаторов для попарно различных классов, результат - класс с наибольшим количеством голосов).

Выделяют два подхода сведения подсчета качества к метрикам классификации: **микро-** и **макро-усреднение**.

Пусть выборка состоит из K классов. Рассмотрим K двухклассовых задач, каждая из которых заключается в отделении своего класса от остальных, для них можно вычислить $\mathrm{TP}_k, \mathrm{FP}_k, \mathrm{FN}_k, \mathrm{TN}_k$.

Микро-усреднение

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

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

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

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

Макро-усреднение

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

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

6 Кластеризация

Данная задача состоит в разделении точек в пространстве на кластеры близко расположенных.

6.1 Метрика качества кластеризации

Как и для задач обучения с учителем, хочется уметь понимать, хорошо ли мы решили задачу - нужно научиться измерять качество кластеризации.

Так, например, даже не зная число кластеров, можно перебрать K в **K-Means** и выбрать гиперпараметр с лучшим качеством.

Одним из примеров метрики может быть метрика **силуэт**.

Для одного элемента x она считается так:

$S(x) = \frac{b(x)-a(x)}{\max(a(x),b(x))}$

$a(x)$ = среднее расстояние от x до точек того же кластера.

$b(x)$ = среднее расстояние от x до точек ближайшего кластера.

Значение *силуэта* для кластера равно среднему значению $S(x)$ от каждого элемента.

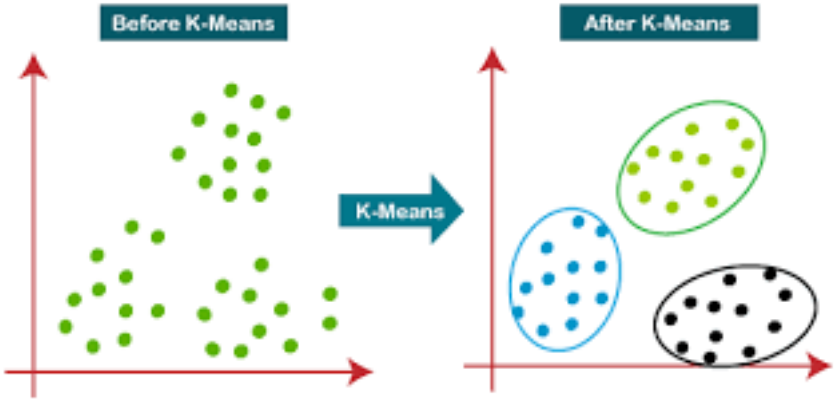
Чем он больше тем лучше кластер отделим от других кластеров.

Примеры из выборки можно отсортировать по значению *силуэта* для каждого примера и по присвоенной метке кластера. Такое представление позволяет отфильтровать шумные примеры (у которых силуэт меньше некоторого порога).

6.2 Алгоритм K-Means

Нужно в начале выбрать K центров кластеров (K - гиперпараметр). Далее итеративно выполняются 2 шага, пока обновления не перестанут происходить:

1. Обновить кластеры, приписав каждой точке кластер самого близкого к ней центра
2. Обновить центр каждого кластера как центр масс его точек



Плюсы :

- простой и понятный

Минусы :

- нужно знать K
- слишком простая модель, кластер = выпуклая околокруглая штука, так как это по сути диаграмма Вороного
- если плохо выбрать начальные центры, может сойтись к плохому результату

Поэтому обычно K-Means запускают несколько раз и выбирают лучший результат.

6.3 DBSCAN

DBSCAN строит столько кластеров, сколько получится, причем многие вершины могут не войти ни в один кластер, они называются выбросами.

DBSCAN опирается на два гиперпараметра:

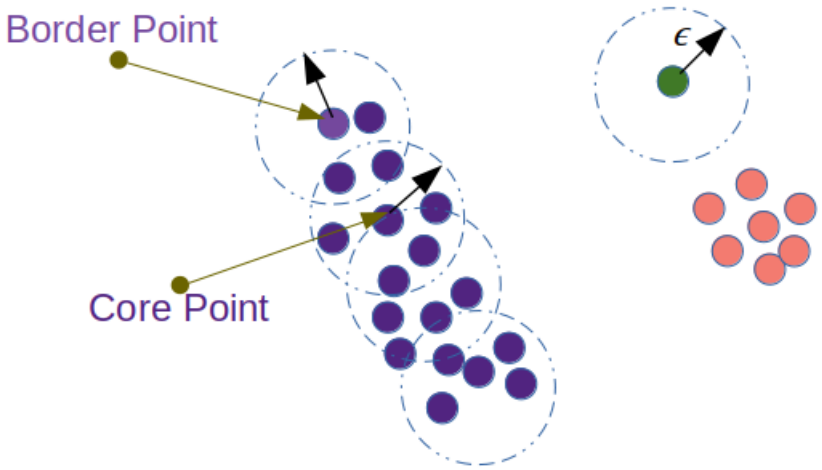
eps - означает расстояние, на котором две вершины считаются соседями.

min-samples - означает сколько нужно соседей из кластера, чтобы считать вершину коренной вершиной кластера.

Сам алгоритм состоит из таких шагов:

1. Выбрать соседей для каждой вершины на расстоянии до eps
2. Найти компоненты связности коренных вершин - добавляем вершину в компоненту коренных, если у нее хотя бы min-samples соседей лежат в этой компоненте

3. Добавить оставшиеся вершины в самый популярный кластер соседей, если есть соседи
4. Оставшиеся вершины - это выбросы



$$N_{Eps}(p) = \{q \in D \text{ such that } dist(p,q) \leq \epsilon\}$$

ϵ = 1 unit, MinPts = 7

Плюсы :

- сам подберет число кластеров
- опирается на плотность точек, кластеры могут быть вытянутыми и даже невыпуклыми

Минусы :

- нужно подбирать два других параметра
- алгоритм считает, что в разных частях данных плотности должны быть примерно одинаковыми

6.4 Агломеративная кластеризация

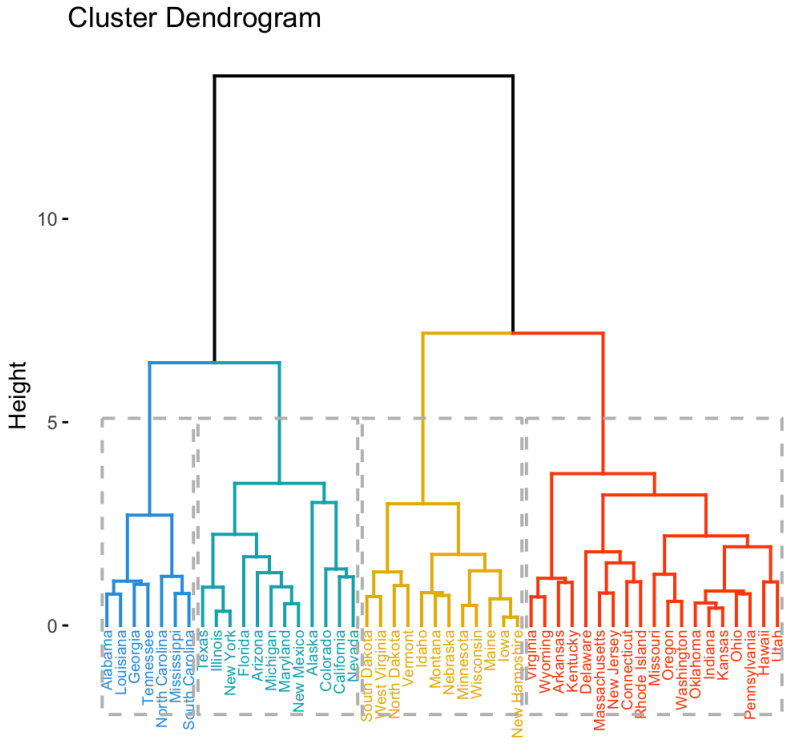
Интуиция у алгоритма очень простая:

1. Начинаем с того, что высыпаем на каждую точку свой кластер

2. Сортируем попарные расстояния между центрами кластеров по возрастанию
3. Берём пару ближайших кластеров, склеиваем их в один и пересчитываем центр кластера
4. Повторяем п. 2 и 3 до тех пор, пока все данные не склеятся в один кластер

Чтобы найти пару ближайших кластеров берут не только расстояние между центрами, бывают и такие метрики:

- Single linkage — минимум попарных расстояний между точками из двух кластеров
- Complete linkage — максимум попарных расстояний между точками из двух кластеров
- Average linkage — среднее попарных расстояний между точками из двух кластеров
- Centroid linkage — расстояние между центроидами двух кластеров



По итогам выполнения такого алгоритма строится дерево склеивания кластеров. Глядя на него можно определить, на каком этапе оптимальнее всего остановить алгоритм.

7 Уменьшение размерности

Уменьшение размерности и визуализация.

Обе эти задачи состоят в сжатии выборки: нужно перевести точки из *N-мерного* пространства в *M-мерное* пространство, где $M < N$, причем так, чтобы близкие точки остались близкими. То есть хочется значительно уменьшить число признаков, не сильно потеряв (или даже улучшив) их качество.

7.1 SVD-разложение

Пусть A - прямоугольная матрица признаков. Тогда она представима единственным образом в следующем виде:

$$A = U\Sigma V \tag{1}$$

где:

Σ – прямоугольная диагональная матрица с неотрицательными убывающими элементами на диагонали,

U, V – квадратные унитарные матрицы

Смысл такой, что преобразование пространства под действием оператора с унитарной матрицей – сохраняет скалярное произведение, то есть грубо говоря это поворот пространства.

Существует теорема Эккарта-Янга, которая утверждает, что можно приблизить матрицу A матрицей меньшего ранга $k < n$, причем наилучшая матрица(в плане минимальной нормы разницы матриц) – будет матрица полученная усечённым **SVD-разложением**.

Усечённое SVD-разложение

Выберем k – ранг матрицы, которой будем приближать исходную матрицу A, с практической точки зрения это количество признаков, которые мы хотим оставить.

Применим SVD-разложение.

$$A = U\Sigma V$$

Оставим только k столбцов в U и k строк в V, а так же занулим все, кроме k первых чисел на диагонали в Σ, обрежем ее до размерности $k \cdot k$.

Назовём это $A_k = U_k \Sigma_k V_k$.

Такое приближение матрицы A матрицей ранга k – оптимально, а матрица $U_k \Sigma_k$ – называется **усечённым SVD-разложением** матрицы A и содержит ровно k столбцов, то есть уменьшилось количество признаков. В sklearn этот метод лежит в sklearn.decomposition.TruncatedSVD.

7.2 PCA - метод главных компонент

Метод **PCA** заключается в том, чтобы найти в N-мерном пространстве такое K-мерное пространство, что проекция всех точек на него будет как можно более рассеянной (то есть иметь наибольшую дисперсию).

Оказывается, подходит пространство, сумма квадратов расстояния от которого до всех точек минимальна.

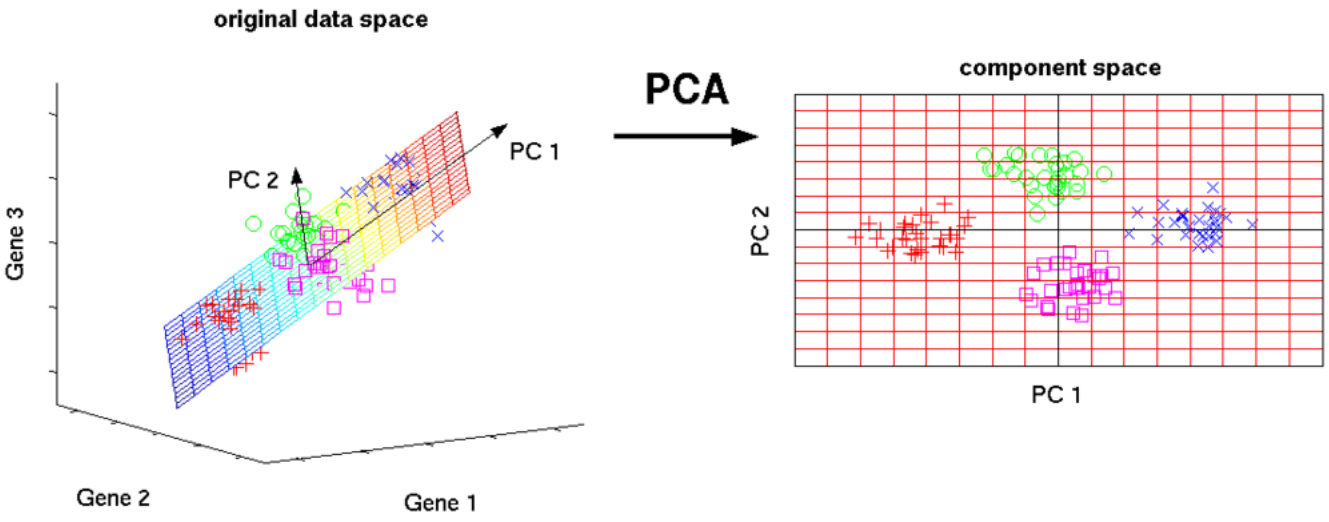
Первая компонента (ось) выбирается так, чтобы дисперсия проекции вдоль нее была максимальна.

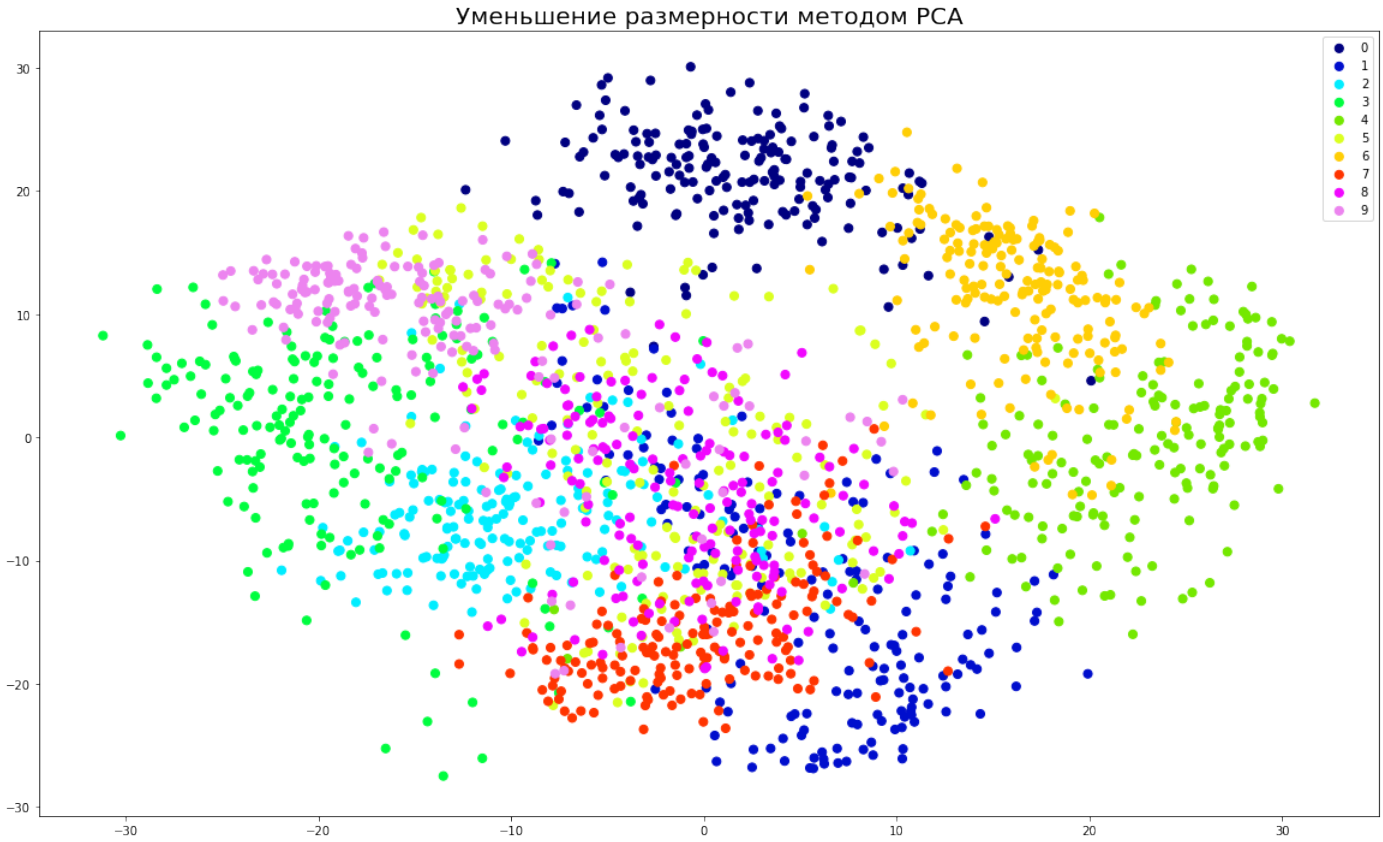
После этого все точки проецируются вдоль первой компоненты и там таким же способом выбирается вторая компонента и так далее.

Получившиеся k компонент будут образовывать K-мерное пространство, проекция точек на которое будет иметь наибольшую дисперсию.

Оказывается, если применить *TruncatedSVD* к центрированным данным (среднее по каждой координате равно 0), то именно это и получится!

В каком-то смысле PCA - это лучший линейный способ уменьшить размерность пространства.





Плюсы PCA

- считается относительно быстро
- детерминированность (зависит от реализации, в sklearn не так)
- любая размерность
- полученную модель уменьшения размерности можно применять и к новым точкам (нужно просто спроецировать ее на выделенное k-мерное пространство)
- визуализация при $k = 2$ - это честная проекция на какую-то плоскость, легкая интерпретируемость результатов
- при увеличении размерности старые координаты не меняются
- отлично подходит для отбора признаков

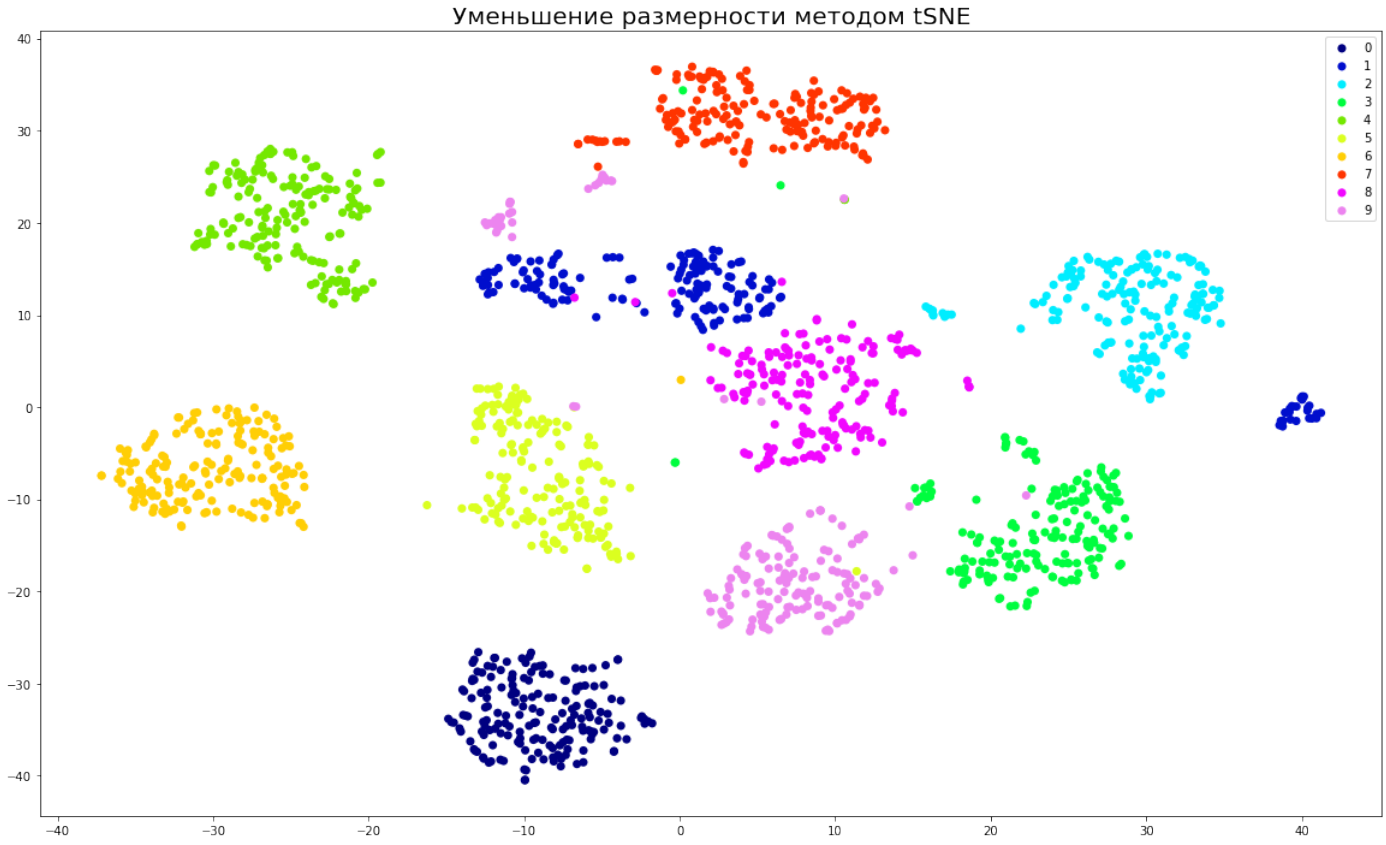
Минусы PCA

- алгоритм слишком простой - он линейный
- некоторые непохожие точки могут спроецироваться в одну и ту же при $k = 2$

7.3 t-SNE

Это гораздо более сложный нелинейный алгоритм. Он очень крут для визуализации данных, и работает только с переводом в 2D и 3D.

Внутри там происходит что-то физическое - элементы будто разлетаются, причем чем дальше они друг от друга, тем сильнее они друг друга оттакливают.



Плюсы t-SNE

- отлично визуализирует (2D, 3D) данные
- физически разносит разные точки дальше друг от друга
- сильно разные точки не могут в конце оказаться рядом

Недостатки t-SNE

- недетерминированность, результат меняется при каждом запуске
- координаты ничего не значат сами по себе
- нельзя легко добавить новые данные
- долго работает

- уменьшает только до 2 и 3 размерности

8 Решающие деревья

Дерево принятия решений — средство поддержки принятия решений.

Структура дерева представляет собой «листья» и «ветки». На рёбрах («ветках») дерева решения записаны признаки, от которых зависит целевая функция, в «листьях» записаны значения целевой функции, а в остальных узлах — признаки, по которым различаются случаи.

Чтобы классифицировать новый случай, надо спуститься по дереву до листа и выдать соответствующее значение.

8.1 Энтропия

8.1.1 Энтропия состояния системы

$$S = -\sum_{i=1}^{N} p_i \log_2 p_i$$

где p_i — вероятности нахождения системы в i-ом состоянии. Интуитивно, энтропия соответствует степени хаоса в системе. Чем выше *энтропия*, тем менее упорядочена система и наоборот.

8.1.2 Прирост информации

Поскольку *энтропия* - по сути степень хаоса в системе, уменьшение *энтропии* называют **приростом информации**. Формально *прирост информации* (*information gain, IG*) при разбиении выборки по призна-

ку Q (например признаком Q может быть « x $\leqslant$ 12 ») определяется как

$$IG(Q) = S_O - \sum_{i=1}^{q} \frac{N_i}{N} S_i$$

где q — число групп после разбиения, N_i — число элементов выборки, у которых признак Q имеет i-ое значение.

8.2 Алгоритм построения решающего дерева.

Интуитивно на каждом шаге мы хотим задавать вопрос, который давал бы максимальное количество информации и лучше всего бы на конкретном шаге разделял выборку.

В алгоритмах используется **принцип жадной максимизации прироста информации** — на каждом шаге выбирается тот признак, при разделении по которому *прирост информации* оказывается *наибольшим*.

Дальше процедура повторяется рекурсивно, пока энтропия не окажется равной нулю или какой-то малой величине. В разных алгоритмах применяются разные эвристики для "ранней остановки"или "отсечения чтобы избежать построения переобученного дерева.

8.2.1 Критерии останова при построения дерева

- Ограничение максимальной глубины дерева.
- Ограничение минимального числа объектов в листе.
- Ограничение максимального количества листьев в дереве.
- Остановка в случае, если все объекты в листе относятся к одному классу.
- Требование, что функционал качества при дроблении улучшался как минимум на x процентов.

8.3 Кросс-энтропия.

Кросс-энтропия — мера разности между распределениями (Вроде тут вместо кросс-энтропии должна быть просто энтропия).

9 Ансамблирование

Ансамбль алгоритмов. Бэггинг. Случайный лес. Градиентный бустинг.

9.1 Стэккинг

Возьмем несколько алгоритмов, KNN, DecisionTree, SVN и применим простой алгоритм принятия решений, например логистическую регрессию. Данные на вход одни и те же, алгоритмы разные.

9.2 Бэггинг

Из исходных данных строим различные выборки(повторение элементов допустимо), для каждой применяется одинаковый алгоритм, все ответы усредняются. Очень часто для этого используют деревья. Много деревьев на случайных подвыборках – RandomForest.

9.3 Бустинг

Берём исходные данные, строим на них модель, получаем ответы. Зафиксируем ошибки модели, добавим их в исходные данные, можем добавить веса ошибкам. Опять строим модель, получаем ответы, считаем ошибки, добавляем в данные. И т д, пока не сделаем фиксированное число итераций бустинга или пока ошибка не станет достаточно маленькой.

10 Обработка текстов

Токенизация, Лемматизация, CountVectorizer, OneHotVectorizer, Стемминг. Word2vec (идея, какие полезные свойства эмбеддингов даёт)

10.1 Стоп-слова

Хотим уменьшить словарь. Плохие слова:

- Слишком частые
- русский язык: и, но, я, ты, ...
- английский язык: a, the, I, ...
- специфичные для коллекции: "сообщать"в новостях
- Слишком редкие
- Предлоги, междометия, частицы, цифры

10.2 Токенизация

Токенизация – это разделение текста на токены, элементарные единицы.
В ноутбуках были примеры токенизации просто разбиением текста по пробелам, по регулярке, чтобы объединять целиком слова или имена с фамилиями, которые считаются одним словом, или по регулярке, чтобы токеном было целое предложение.

10.3 Стемминг

Стемминг - нормализация слов путем отбрасывания окончаний (согласно правилам, основанным на грамматике языка).
Стеммеры(nltk):

- Porter stemmer
- Snowball stemmer
- Lancaster stemmer

- MyStem

В целом :

- Плохо работает для русского языка
- Нормально работает для английского
- Повышает качество модели

10.4 Лемматизация

Лемматизация - приведение слов к начальной морфологической форме (с помощью словаря и грамматики языка).
Лемматизаторы:

- pymorphy2 (язык русский, украинский)
- mystem3 (язык русский)
- Wordnet Lemmatizer (NLTK, язык английский, требует POS метку)
- Metaphraz (язык русский)
- Coda/Cadenza (языки русский и английский)

Лемматизатор на самом деле довольно сложно устроены, им нужны теги частей речи (POS).

По умолчанию функция WordNetLemmatizer.lemmatize () будет считать, что это слово является существительным, если на входе не обнаружен тег POS.

Сначала вам понадобится функция pos_tag, чтобы пометить предложение и использовать тег, чтобы преобразовать его в теги WordNet, а затем передать его в WordNetLemmatizer.

Примечание. Лемматизация не будет работать только на одиночных словах без контекста или знании своего тега POS

В целом:

- Лучше стемминга для русского языка
- Хорошо работает для английского языка
- Повышает качество модели
- Гораздо медленнее чем стемминг

10.5 One-hot encoding

Представление словаря в виде бинарных векторов, у которых все значения равны 0, кроме одного, отвечающего за соответствующее слово

10.6 CountVectorizer

Представление словаря в виде векторов, у которых i-ая координата – количество вхождений i-ого слова в предложение соотвествующее этому вектору.

Может возникнуть проблема, что какое-то незначительное для классификации предложения слово может встречаться очень часто и весить очень много, хотя особого смысла в себе не несёт.

10.7 TF-IDF Vectorizer

Для того чтобы решить эту проблему есть TF-IDF Vectorizer.

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

Где n_t – число вхождений слова t в документ d, а в знаменателе – общее число слов в документе.

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

Где $|D|$ – число документов в коллекции, $|\{d_i \in D | t \in d_i\}|$ – число документов из коллекции D, в которых встречается t(когда $n_t \neq 0$)

$$tfidf(t,d,D) = tf(t,d) \times idf(t,D)$$

Большой вес в TF-IDF получат слова с высокой частотой в пределах конкретного документа и с низкой частотой употреблений в других документах.

10.8 Word2Vec

Word2Vec - это нейросетевая модель.

Результатом работы данной модели является словарь эмбеддингов, позволяющий сопоставлять словам их векторные представления.

Используя такое представление, можно по какому-то новому слову, найти несколько ближайших к нему слов(имеется в виду несколько ближайших векторных представлений других слов).

Плюс такого подхода в том, что в нем похожим по значению словам соответствуют похожие вектора - т.е. близкие друг к другу точки в n-мерном пространстве.

У эмбеддингов, получаемых с помощью Word2Vec есть еще одна интересная особенность - если мы можем сказать, что ‘A относится к B, как C к D‘, то вектора слов ‘A - B‘ и ‘C - D‘ будут довольно похожи. Это значит, что если мы рассмотрим вектор ‘A - B + D‘, то вектор ‘C‘ часто будет оказываться среди его ближайших соседей:

$'Paris' -' France' +' Germany' =' Berlin$,

$'king' -' man' +' woman' =' queen$‘

Как мы видели выше, модели Word2Vec дают нам хороший способ представлять слова в виде векторов. Но для решения задач ML нам нужно научиться представлять в виде векторов тексты, а не отдельные слова.

Построенные на основе Word2Vec эмбеддинги текстов будут сохранять их смысл и при этом их размерность будет гораздо меньше, чем у векторных представлений, получаемых с помощью tf-idf. Такие эмбеддинги бывают очень полезны в разных задачах ML.

Попробуем считать вектор документа, как среднее векторов слов из этого документа.

Оказывается, что усреднять слова в документе - не лучший способ получить его векторное представление. Есть много слов, которые встречаются часто и есть почти во всех документах. При таком способе подсчета они будут делать вектора документов слишком похожими друг на друга.

Значит, нам нужно давать словам разные веса, когда мы вычисляем вектор документа. Большой вес должен быть у слов, которые часто встречаются в этом документе, но редко в других.

Но это как раз **tf-idf**. (Также можно использовать просто idf. TF - частота слова в документе - итак будет учтена, т.к. мы суммируем вектора слов в документе).

Осталось понять, как посчитать такую взвешенную сумму векторов. Для этого можно составить матрицу из векторов слов, встречающихся в

нашей коллекции (они обязательно должны идти в том же порядке, что и в матрице из ‘TfidfVectorizer‘ - но там слова будут столбцами, а тут - строками), после чего перемножить 2 матрицы.

Пусть $DOCS$ – число документов в коллекции, $WORDS$ – число уникальных слов, DIM – размерность пространства эмбеддингов из Word2Vec.

Тогда первая матрица будет размера $(DOCS, WORDS)$,

вторая - $(WORDS, DIM)$,

а их произведение - $(DOCS, DIM)$ - это и будет матрица векторных представлений для каждого документа в коллекции. Размерность матрицы понижена, без использования PCA и других алгоритмов.

11 Релевантность документа

11.1 Понятие релевантности

Релевантность в информационном поиске — соответствие поискового намерения, заложенного в запросе, и выдачи в поисковой системе, полученной в результате этого запроса. Пользователь, который вводит запрос в поисковую систему ожидает, что результаты будут соответствовать намерению, которое он заложил в запросе, и он получит релевантную выдачу.

Существует несколько подходов к оценке релевантности:

- Содержательная релевантность — соответствие ответов информационному запросу, определяемое неформальным путём.
- Формальная релевантность — соответствие, определяемое путём сравнения образа поискового запроса с поисковым образом ответа по определённому алгоритму.

11.2 TF-IDF-метод

Одним из распространённых методов для оценки релевантности является TF-IDF-метод. Его смысл сводится к тому, что чем больше локальная частота термина (запроса) в документе (TF) и больше «редкость» (то есть, чем реже он встречается в других документах) термина в коллекции (IDF), тем выше вес данного ответа по отношению к термину — то есть ответ будет выдаваться раньше в результатах поиска по данному термину. Автор метода — Джерард Солтон, в дальнейшем доработан Карен Спарк Джонс.

11.3 Факторы, влияющие на релевантность

Факторы, которые оказывают влияние на релевантность, принято делить на внешние и внутренние. К внешним относят ссылочную массу, к внутренним — технические составляющие и содержимое.

- Ссылочная масса. Чем больше тематических и качественных ссылок ведёт на страницы ресурса, тем больше вероятность ценности ресурса для пользователя.

- Технические составляющие. Большая группа параметров, по которым поисковая система оценивает как сайт в целом, так и отдельные страницы (например, наличие метатегов, отсутствие ошибок в HTML-разметке и так далее).
- Контент. Ключевой фактор от которого зависит релевантность страницы и конверсия.

11.4 Ранжирование

Ранжирование — это сортировка каких-либо объектов из соображения их относительной релевантности («важности»). Как измерять качество ранжирования.

У ранжируемых объектов есть какое-то число r_i , соответствующее «эталонной» релевантности. Существует два основных способа его получения:

- На основе исторических данных. Например, в случае рекомендаций контента, можно взять просмотры (клики, лайки, покупки) пользователя и присвоить просмотренным весам соответствующих элементов 1, а всем остальным — 0.
- На основе экспертной оценки. Например, в задаче поиска, для каждого запроса можно привлечь команду асессоров, которые вручную оценят релевантности документов запросу. Обычно оценка категориальная (то есть не действительное число, а несколько градаций, скажем, 2, 3 или 5).

11.5 Метрики ранжирования

Дальше возникает задача оценить качество ранжирования в целом, уже имея информацию о релевантности каждого элемента по отдельности. Условно, от идеального ранжирования мы хотим, чтобы объекты шли в порядке убывания релевантности, но есть несколько способов задать такие метрики.

Precision at k (p@k) — «точность на K элементах». Допустим, наш алгоритм ранжирования выдал оценки релевантности для каждого элемента.

Отобрав среди них первые k элементов с наибольшим ri посчитаем среди них долю релевантных — это и называется precision at k.

MRR - mean reciprocal rank, дословно «средний обратный ранг». Эта метрика никак не учитывает абсолютные значения релевантности, а только их относительный порядок. Пусть i -тый самый релевантный элемент в нашем упорядочивании оказался на позиции $\operatorname{ran} k_i \in [1, N]$. Тогда MRR считается как:

$$MRR@k = \frac{1}{K} \sum_{i=1}^{K} \frac{1}{\text{rank}_i}$$

Чаще всего считают MRR@1, то есть просто по большому количеству запросов смотрят, на какой позиции находится самый подходящий документ, и усредняют обратные к этим позициям.

DCG - discounted cumulative gain. Учитывает одновременно и значения релевантностей, и их порядок, путем домножения релевантности на вес, равный обратному логарифму номера позиции:

$$DCG = \sum_{i=1}^{N} \frac{rel_i}{\log_2(i+1)}$$

у неё есть проблема, связанная с тем, что, по-первых, оценки релевантности могут быть неотмасштабированными, и, во-вторых, на больших последовательностях она выше. Поэтому вводят $nDCG$ («нормализованный DCG» принимающий значения от о до 1), равный отношению DCG данного ранжирования к «идеальному», то есть максимально возможному (iDCG):

$$nDCG = \frac{DCG}{iDCG}$$

iDCG можно получить, отсортировав выдачу по реальным релевантностям и посчитав DCG на нём.

pFound. В Яндексе для поисковых задач когда-то придумали и до сих пор повсеместно используют метрику, которая называется pfound - грубо говоря, вероятность того, что пользователь найдёт на странице выдачи (среди первых n документов) то, что искал.

$$\text{pfound} = \sum_{i=1}^{n} p\,\text{Look}(i) \cdot p\,\text{Rel}(i)$$

Где p Look определяется рекурсивно, как вероятность того, что пользователь вообще посмотрит на $\boldsymbol{i}$ -тый документ:

$$p\,\mathrm{Look}(i) = \begin{cases} 1, & i = 1 \\ p\,\mathrm{Look}(i-1) \cdot (1 - p\,\mathrm{Rel}(i-1)), & i > 1 \end{cases}$$

Интуиция такая: пользователь просматривает страницы выдачи сверху вниз, пока не найдёт релевантный документ, вероятность чего равна $p\,\mathrm{Rel}(i) = f\,(r_i)$, то есть как-то зависит от асессорской оценки релевантности.

12 Рекомендательные системы

12.1 Задача

Задача рекомендательной системы — найти для пользователя такие элементы заданного множества, которые с высокой вероятностью понравятся пользователю.

Введем некоторые обозначения:

- Пользователи — множество U (users).
- Товары — множество I (items) - товары/фильмы/музыка/техника.
- Оценки - пусть парам <пользователь, товар> соответствует некоторая оценка r, выражающая заинтересованность пользователя в этом товаре $r : U \times I \to R$

Требуется по известным оценкам r для каждого пользователя получить набор из k товаров, наиболее подходящих пользователю (т.е с максимальной оценкой r). Кроме того, часто влияет и порядок выданных товаров.

12.2 Модели и признаки

Рассмотрим **baselines** - простые решения, позволяющие сделать начальную модель, с значениями метрик которой потом будем сравнивать будущие модели.

Самым очевидным подходом будет найти среднее всех известных оценок μ и выдавать μ как ответ, если рейтинг неизвестен.

Этот подход никак не учитывает свойства товара и пользователя. Например какой-то пользователь может ставить оценки в среднем меньшие, чем другой.

Давайте найдем для каждого пользователя и товара средние значения отдельно и найдем, насколько они отличаются от μ:

$$b_u = \mu_u - \mu$$

$$b_i = \mu_i - \mu$$

$$r_{ui} = b_u + b_i + \mu$$

В данный момент мы не учитываем, насколько вкусы пользователей похожи между собой. Поэтому можем немного модифицировать подсчет среднего:

Будем считать среднее для всех товаров для выбранного пользователя, при этом влияние других пользователей будет зависеть от их схожести с выбранным пользователем:

$$\mu_{u_0} = \frac{\sum_U sim(u, u_0) \sum_I r_{ui}}{|Y|}$$

12.3 Похожесть пользователей и товаров

Обозначим за I_{uv} - множество товаров, оценки для которых известны у пользователей u и v одновременно

Теперь можем посчитать сходство между пользователями с помощью корреляции Пирсона (либо другой метрики - например, посчитать какую-нибудь метрику, обратную расстоянию).

$$sim(u, v) = \frac{\sum_{I_{uv}} (r_{ui} - \mu_u)(r_{vi} - \mu_v)}{\sqrt{\sum_{I_{uv}} (r_{ui} - \mu_u)^2 \sum_{I_{uv}} (r_{vi} - \mu_v)^2}}$$

То же самое можно сделать и с товарами.

12.4 Коллаборативная фильтрация

Существуют подходы на основе сходств пользователей и на основе сходств товаров:

12.4.1 User-based collaborative filtering

Для текущего пользователя найдем множество пользователей, наиболее похожих на него:

$$U(u_0) = \{u \in U | sim(u_0, u) > \alpha\}$$

Теперь найдем средние значения оценок для всех товаров у этих пользователей.

Проблема данного подхода в том, что он позволяет строить рекомендации только в том случае, если для данного пользователя существуют похожие на него. Если же пользователь новый или нетипичный, то подобрать что-либо не получится.

12.4.2 Item-based collaborative filtering

Для текущего пользователя найдем множество товаров, наиболее похожих на заинтересовавшие пользователя.

В таком подходе решается проблема нетипичного пользователя, так как необязательно иметь пользователей со схожими интересами, и подход позволяет найти товары, похожие на интересные ему.

Но в таком подходе есть и проблемы: есть вероятность, что вместо действительно интересных товаров будем рекомендовать популярные.

12.5 Контентные модели

Опишем товары векторами, называемыми **эмбэдингами**. Теперь мы можем посчитать расстояния в пространстве эмбэдингов. Это позволит нам обучить модели, которые будут предсказывать целевую переменную на основе характеристик товара, а не только на основе рейтингов, поставленных другими пользователями

13 Теория вероятностей

13.1 Вероятность

Наивное определение вероятности события A:

$$P(A) = \frac{N_A}{N}$$

где N_A — число появлений события A, N — число событий

13.2 Условная вероятность

Условная вероятность — вероятность события, когда уже что-то случилось. Обозначается $P(A|B)$, где $P(A \mid B)$ — вероятность события A при условии, что B сбылось.

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}$$

События A и B называются независимыми, если $P(A \cap B) = P(A)P(B)$ или что то же самое $P(A) = P(A \mid B)$

13.3 Формула Байеса

Знаем, что $P(A \mid B) = \frac{P(A \cap B)}{P(B)}$. Найдём вероятность $P(B \mid A)$. По формуле условной вероятности:

$$P(B \mid A) = \frac{P(A \cap B)}{P(A)}$$

Так как $P(A \cap B)$ — общее в этих формулах, из формулы условной вероятности $P(A \cap B) = P(A \mid B)P(B)$. Подставим:

$$P(B \mid A) = \frac{P(A \mid B)P(B)}{P(A)}$$

Это и есть **формула Байеса**, которая позволяет "переставить причину и следствие то есть найти вероятность того, что событие B вызвано причиной A.

13.4 Формула полной вероятности

Пусть $B_1, B_2, ..., B_n$ — несовместные события, то есть те, которые не могут произойти одновременно. Кроме того, пусть

$$P(B_1) + P(B_2) + ... + P(B_n) = 1$$

То есть хотя бы одно из событий обязательно произойдёт. В таком случае говорят, что $B_1, B_2, ..., B_n$ образуют полную группу несовместных событий.

Пусть ещё есть событие A.

Рассмотрим вероятности $P(A \cup B_1), P(A \cup B_2), ...P(A \cup B_n)$.

$$P(A) = P(A \cup B_1) + P(A \cup B_2) + ... + P(A \cup B_n)$$

Распишем каждую условную вероятность по формуле условной вероятности:

$$P(A) = P(A \mid B_1)P(B_1) + P(A \mid B_2)P(B_2) + ... + P(A \mid B_n)P(B_n)$$

$$P(A) = \sum_{i=1}^{n} P(B_i)P(A \mid B_i)$$

С помощью этого результата можно усовершенствовать формулу Байеса:

$$P(B_i \mid A) = \frac{P(A \mid B_i)P(B_i)}{P(A)} = \frac{P(A \mid B_i)P(B_i)}{\sum_{i=1}^{n} P(B_i)P(A \mid B_i)}$$

Вероятности $P(B_i \mid A)$ называют **апостериорными** вероятностями (то есть вероятности, которые немного уточнены, так как произошло событие A). А $P(B_i)$ — **априорные** вероятности, то есть те, что известны до проведения эксперимента, связанного с A.

13.5 Распределения вероятностей

Распределение вероятностей — это функция, которая говорит нам, какая вероятность у каждого результата эксперимента.

Например, распределение результатов выпадения игральной кости — это равномерное распределение с вероятностью $1/6$.

Результат случайного эксперимента называют **случайной величиной**. Говорят, что случайная величина подчиняется какому-то распределению вероятностей. Если X — результат подброса игральной кости, то пишут:

$$X \sim Uniform(1/6)$$

13.6 ММП - Метод максимального правдоподобия

Пусть на входе мы имеем какую-то случайную выборку чисел X. Нам хотелось бы определить, из какого они распределения.

Если мы это узнаем, то можем, например, генерировать больше данных для обучения. Обычно, если построить гистограмму, то можно на глаз оценить, к какому семейству распределений оно относится (нормальное, экспоненциальное, пуассона и т.д.).

Осталось найти параметры этих распределений, которые обозначим за θ.

Так как мы знаем, к какому семейству относится наше распределение, то мы можем записать вероятность или плотность вероятности $p(x)$, как функцию, в которой кроме x будет ещё фигурировать θ.

Суть метода заключается в следующем:

1. Находим вероятность нашей выборки :

$$P(X \mid \theta) = \prod_{i=1}^{n} P(x_i \mid \theta)$$

   – функция правдоподобия.

2. Давайте будем искать такой θ, при котором эта вероятность максимальна. Почему так делают?

Приведём пример.

Пусть у нас есть игральная кость, в которой одна из граней утяжелена, т.е. она выпадает чаще. Пусть у вас есть 1000 бросков этой кости, и вас просят на основе этих данных понять, какая грань утяжелена.

Естественно, что вы ответите, что утяжелена та грань, которая чаще всего выпала. Это и есть метод максимального правдоподобия.

Параметром в нашем странном распределении является номер грани, которая утяжелена. Если больше всего выпадала грань '3', то вероятность того, что она утяжелена намного больше, чем то, что утяжелена грань '2'.

Поэтому мы перебираем все варианты θ и находим из них ту, которая даёт максимальную вероятность:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} \{\log(\prod_{i=1}^{n} P(x_i \mid \theta))\} = \underset{\theta}{\operatorname{argmax}} \{\sum_{i=1}^{n} \log(P(x_i \mid \theta))\}$$

Если θ — вещественный параметр, то обычно используют производную функции, чтобы найти минимум.

Совет: можно искать не минимум произведения, а логарифм от этого минимума. Так вы будете оперировать с меньшими числами и производную суммы легче посчитать, чем производную от произведения.

13.6.1 Связь MSE и ММП

Рассмотрим, классическую регрессионную модель:

$$y_i = w \cdot x_i + \varepsilon, \varepsilon \sim N(0; 1)$$

Другими словами, у нас есть данные какого-то измерения, которые лежат на линии(гиперплоскости в многомерном случае) с каким-то шумом ε, которые подчиняется стандартному нормальному распределению $N(0; 1)$.

Давайте с помощью метода максимального правдоподобия найдем параметры w. Что такое y_i?. Т.к. $\varepsilon \sim N(0, 1)$, то $y_i \sim N(w \cdot x_i, 1)$, т.к. это просто сдвиг на какое-то число.

Запишем функцию правдоподобия для одного элемента выборки :

$$P(y_i \mid x_i) = \frac{1}{\sqrt{2\pi}} \exp\{-\frac{(y_i - x_i \cdot w)^2}{2}\}$$

Мы просто подставили в плотность нормального распределения наши величины. Возьмём от него натуральный логарифм.

$$\ln\{(P(y_i \mid x_i, w, \varepsilon))\} = \ln\{\frac{1}{\sqrt{2\pi}}\exp\{-\frac{(y_i - x_i \cdot w)^2}{2}\}\} = -\frac{\ln(2 \cdot \pi)}{2} - \frac{1}{2}(y_i - x_i \cdot w)^2$$

Теперь запишем логарифм функции правдоподобия для всей выборки :

$$\ln P(Y \mid X, w, \varepsilon) = -\frac{n}{2} \cdot \ln(2 \cdot \pi) - \sum_{i=1}^{n}(y_i - x_i \cdot w)^2 \cdot \frac{1}{2}$$

Первое слагаемое можно отбросить так как оно константа, а второе слагаемое – MSE.

Т.е когда мы минимизируем ошибку MSE, то получившиеся веса w будут также являться и оценкой максимального правдоподобия, и будут обладать всеми свойствами этой оценки.

13.7 Наивный байесовский классификатор

Воспользуемся предыдущими формулами для задачи классификации.

Пусть у нас есть задача классификации текстов на $C = \{c_1, \ldots, c_n\}$ классов.

В качестве признаков возьмём какие слова встречаются в текстах. Т.е. пусть во всех текстах всего N уникальных слов. Мы дадим каждому слову номер от 1 до N.

Признаками каждого текста будет кол-во каждого слова, которое есть в тексте. Например, 2 слова «дом», 3 слова «кот», 0 слов «аниме».

Таким образом, каждый текст мы можем представить в виде вектора длины N : $(w_1, \ldots, w_N)$ где w_i— количество слов под номером i в нашем тексте.

У нас есть много таких текстов, поэтому мы можем вычислить различные вероятности.

Например:

- $P(c)$ – Сколько есть текстов каждого класса
- $P(w_1)$ – Какова вероятность встречи слова под номером 1:

- $P(w_2 \mid c)$ – Сколько раз встречается слово под номером 2 в разных классах .

  Для этого берём все тексты, содержащие слово w_2. Смотрим, сколько из этих писем помечено классом c_1, сколько классом c_2 и т.д.

В задаче же классификации у нас есть текст, состоящее из таких-то слов, т.е. имеет признаки. Наша задача состоит в том, чтобы понять класс текста c.

Т.е. найти вероятности $P(c \mid w_1, \ldots, w_N)$, перебрать все c и выбрать из них тот класс, который даёт максимальную вероятность.

Формально:

$$\hat{c} = \operatorname{argmax} P(c \mid w_1, \ldots w_N)$$

где $\hat{c}$- наше предсказание.

Преобразуем эту вероятность по формуле Байеса:

$$P(c \mid w_1, \ldots, w_N) = \frac{P(c, w_1, \ldots w_N)}{P(w_1, \ldots, w_N)}$$

Т.к. мы максимизируем по c, а вероятности $w_1, \ldots w_N$ остаются для письма теми же, то их можно убрать из argmax.

$$P(c, w_1, \ldots, w_N) = P(c)P(w_1, \ldots, w_N \mid c)$$

Вероятность справа достаточно сложная. Давайте сделаем наивное предположение: все слова в текстах появляются независимо друг от друга.

Т.е. если в тексте есть 2 слова "котик то количество "собак"может быть любым. В этом и заключается слово "наивный"в названии данного алгоритма.

$$P(c, w_1, \ldots, w_N) = P(c)P(w_1 \mid c) \ldots P(w_N \mid c)$$

Все вероятности справа мы можем подсчитать по обучающей выборке. Тогда получаем итоговый ответ алгоритма:

$$\hat{c} = \operatorname*{argmax}_c P(c)P(w_1 \mid c) \ldots P(w_N \mid c)$$