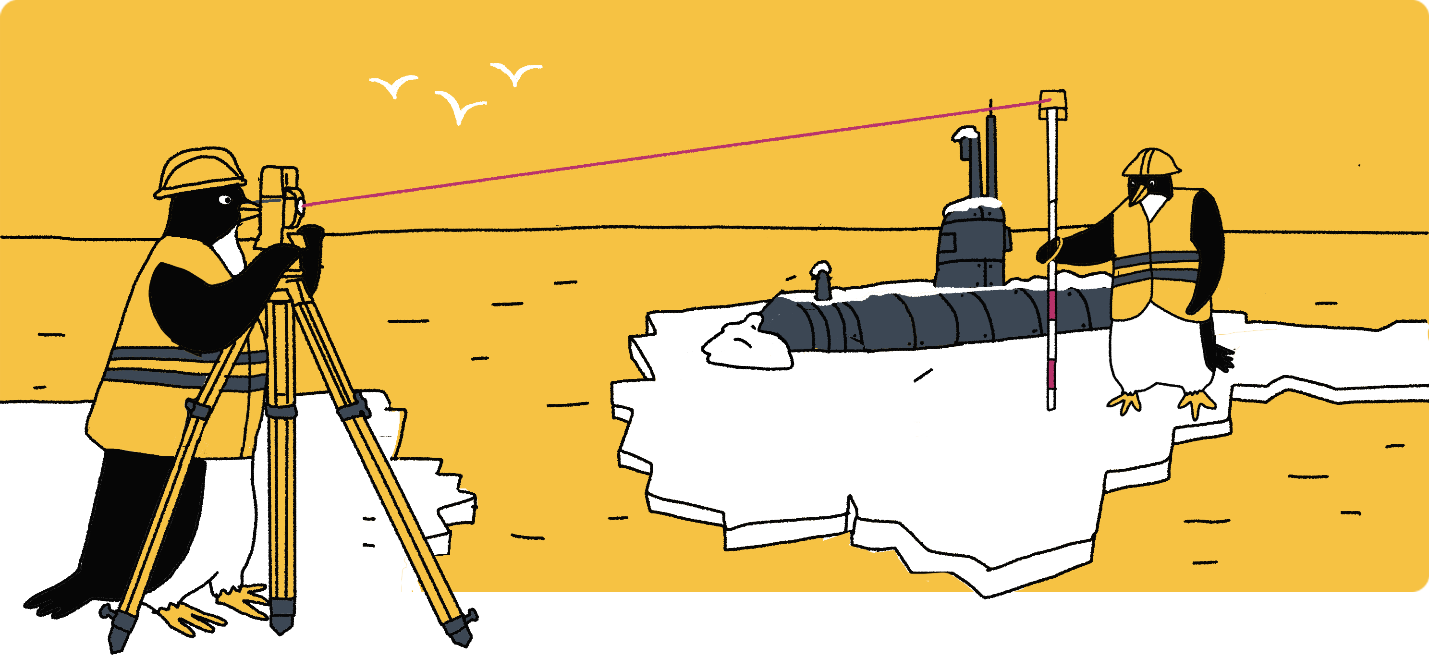
Практическая работа № 1

“Метрические методы и основные метрики машинного обучения”

Цель: изучить алгоритмы метрических методов, получить базовые навыки в обучение моделей машинного обучения. Изучение основных метрик в задачах классификации и регрессии. Знакомство с библиотеками машинного обучения и языком программирования python.

Алгоритм k-ближайших соседей



Смысл метрических методов очень хорошо раскрывает фраза «Скажи мне, кто твой друг, и я скажу, кто ты». Алгоритмы этого класса почти не имеют фазы обучения. Вместо этого они просто запоминают всю обучающую выборку, а на этапе предсказания просто ищут похожие на целевой объекты. Такой процесс называют **lazy learning**, потому что никакого обучения, по сути, не происходит. Также метрические модели являются непараметрическими, потому что они не делают явных допущений о глобальных законах, которым подчиняются данные. Так, линейная регрессия основывается на предположении о том, что изучаемая закономерность линейная (с неизвестными коэффициентами, которые восстанавливаются по выборке), а линейная бинарная классификация – что существует гиперплоскость, неплохо разделяющая классы. Метрические методы по сути своей локальны: они исходят из допущения, что свойства объекта можно узнать, имея представление о его соседях.

Указанные выше свойства могут быть полезными, особенно в случае сложно устроенных данных, для которых мы не можем придумать глобальную модель, однако, с другой стороны, из-за lazy learning алгоритм становится абсолютно неприменимым при большом количестве данных. Несмотря на то, что эти алгоритмы очень просты для понимания, они являются довольно точными и хорошо интерпретируемыми и часто используются как минимум в качестве бейзлайнов в разных задачах.

Обзору одного из самых известных метрических алгоритмов — методу **k-ближайших соседей**, или **k-nearest neighbors (KNN)**, будет посвящена первая часть главы. Этот подход в основном чисто инженерный из-за отсутствия фазы обучения и в настоящее время уже почти нигде не применяется, однако многие техники, на которых основан алгоритм, используются и в других методах. Например, алгоритмы поиска ближайших соседей, являющиеся неотъемлемой частью метода, имеют намного более широкую область применения. Плюс ко всему KNN — очень простой и легко интерпретируемый алгоритм, поэтому изучить его всё равно полезно. Мы обсудим подробнее его преимущества, недостатки, область его применения, а также возможные обобщения.

Для метрических методов очень важно уметь эффективно находить ближайшие объекты, поэтому задача их поиска неизбежно возникает при применении любого такого алгоритма. Возможные подходы к быстрому поиску ближайших соседей мы рассмотрим во второй части главы.

Метод k-ближайших соседей (KNN)

Представим, что мы проводим классификацию объектов на два класса — красный или жёлтый. Нам дана некоторая обучающая выборка и целевой объект (серый):



Мы хотим определить, к какому классу относится серый объект. Интуитивно очевидно, что он должен быть жёлтым, потому что все его соседи жёлтые. Эта интуиция и отражает суть метода KNN — классифицировать целевой объект, исходя из того, какие классы у объектов, которые максимально похожи на него.

Перейдём теперь к более формальному описанию алгоритма. Рассмотрим сначала задачу многоклассовой классификации, а регрессией займёмся позже.

Пусть дана обучающая выборка , где , . Пусть также задана некоторая симметричная по своим аргументам функция расстояния ρ:X×X → [0, +∞). Предположим, что требуется классифицировать новый объект u. Для этого найдём k наиболее близких к u в смысле расстояния ρ объектов обучающей выборки :

Метку класса объекта  будем обозначать . Класс нового объекта тогда естественным образом определим как наиболее часто встречающийся класс среди объектов из :

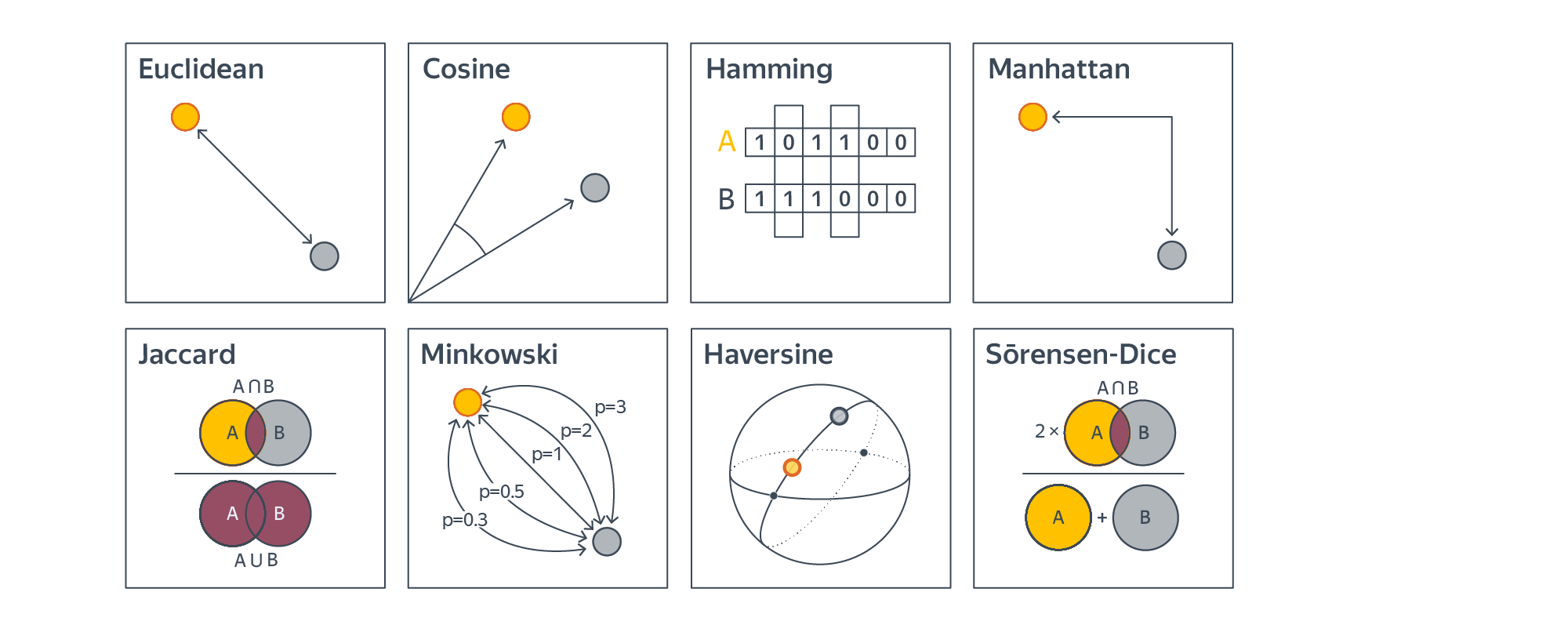
Формула может показаться страшной, но на самом деле все довольно просто: для каждой метки класса  количество соседей u с такой меткой можно посчитать, просто просуммировав по всем соседям индикаторы событий, соответствующих тому, что метка соседа равна y.

Легко заметить, что этот алгоритм позволяет также оценивать вероятности классов. Для этого достаточно просто посчитать частоты классов соседей:

Стоит, однако, понимать, что, хоть такая функция и удовлетворяет свойствам вероятности (она неотрицательна, аддитивна и ограничена единицей), это есть не более чем эвристика.

Несмотря на то, что формально фаза обучения отсутствует, алгоритм может легко переобучиться. Вы можете убедиться в этом сами, использовав маленькое количество соседей (например, 1 или 2), — границы классов оказываются довольно сложными. Происходит это из-за того, что параметрами алгоритма можно считать всю обучающую выборку, довольно большую по размеру. Из-за этого алгоритму легко подстроиться под конкретные данные.

Метрики

Может возникнуть закономерный вопрос, как же правильно выбрать функцию расстояния ρ. В подавляющем большинстве случаев обычное евклидово расстояние будет хорошим выбором. Однако в некоторых случаях другие функции будут подходить лучше, поэтому давайте разберём ещё несколько функций, используемых на практике.

* **Манхэттенская метрика**

Часто используется в высокоразмерных пространствах из-за лучшей устойчивости к выбросам. Представим, что два объекта в 1000-размерном пространстве почти идентичны, но сильно отличаются по одному из признаков. Это почти наверняка свидетельствует о выбросе в этом признаке, и объекты, скорее всего, очень близки. Однако евклидово расстояние усилит различие в единственном признаке и сделает их более далёкими друг от друга, в отличие от манхэттенской метрики, в которой используется модуль вместо квадрата.

* **Метрика Минковского**

Является обобщением евклидовой (p=2) и манхэттенской (p=1) метрик.

* **Косинусное расстояние**

Эта метрика хороша тем, что не зависит от норм векторов. Такое поведение бывает полезно в некоторых задачах, например при поиске похожих документов. В качестве признаков там часто используются количества слов. При этом интуитивно кажется, что если в тексте использовать каждое слово в два раза больше, то тема этого текста поменяться не должна. Поэтому как раз в этом случае нам не важна норма вектор-признака, и в задачах, связанных с текстами, часто применяется именно косинусное расстояние.

**Замечание**. Упомянутые в этом параграфе функции мы называем «метриками», но, конечно же, они не обязаны быть метриками в строгом математическом смысле. Они неотрицательны и симметричны, но могут не удовлетворять неравенству треугольника.

Обобщение алгоритма или взвешенный KNN.

У оригинального алгоритма есть один большой недостаток: он никак не учитывает расстояния до соседних объектов, хотя эта информация может быть полезной.

Давайте попробуем придумать, как исправить этот недостаток. Нам нужно каким-то образом увеличивать вклад близких объектов и уменьшать вклад далёких. Возникает идея — назначить этим индикаторам веса, которые тем больше, чем ближе объект к целевому. Таким образом, получаем следующую формулу:

Такой алгоритм называется **взвешенным KNN** (**weighted KNN**).

Есть множество вариантов выбора весов для объектов, которые можно поделить на две большие группы. **В первой группе** веса зависят лишь от **порядкового номера** объекта в отсортированном по близости к u массиве Xk(u). Чаще всего берутся линейно () или экспоненциально () затухающие веса.

Однако здесь мы также не используем всю информацию, которая нам доступна. Зачем использовать порядок соседей, порождаемый расстояниями, если можно использовать сами **расстояния**? **Во второй группе** методов вес является некоторой функцией от расстояния. Давайте подумаем, какие должны быть свойства у этой функции. Очевидно, она должна быть положительной на своей области определения, иначе модель будет поощрять несовпадение с некоторыми ближайшими соседями. Также необходимо, чтобы функция монотонно не возрастала, чтобы вес близких соседей был больше, чем далёких. Таким образом вводится так называемая ядерная функция (kernel function) , обладающая перечисленными выше свойствами, с помощью которой и высчитывается вес каждого соседа:

где h — некое положительное число, которое называется *шириной окна*.

От выбора ядра зависит гладкость аппроксимации, но на её качество этот выбор почти не влияет. Примеры ядерных функций в порядке увеличения их гладкости.

На практике чаще всего используют либо прямоугольное для простоты, либо гауссовское, в случае, когда важна гладкость модели (немного забегая вперёд — это особенно важно в регрессии).

Ширина окна, в свою очередь, сильно влияет как раз на качество модели. При слишком маленькой ширине модель сильно подстраивается под обучающую выборку и теряет свою обобщающую способность. При слишком большой ширине, напротив, модель становится слишком простой. Универсальной ширины окна не существует, поэтому для каждой задачи её приходится подбирать отдельно.

### **Kernel regression**

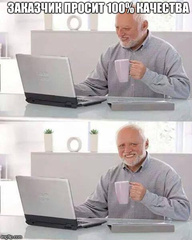
Алгоритм KNN можно довольно легко обобщить и на задачу регрессии. Самые очевидные способы — брать либо обычное среднее:

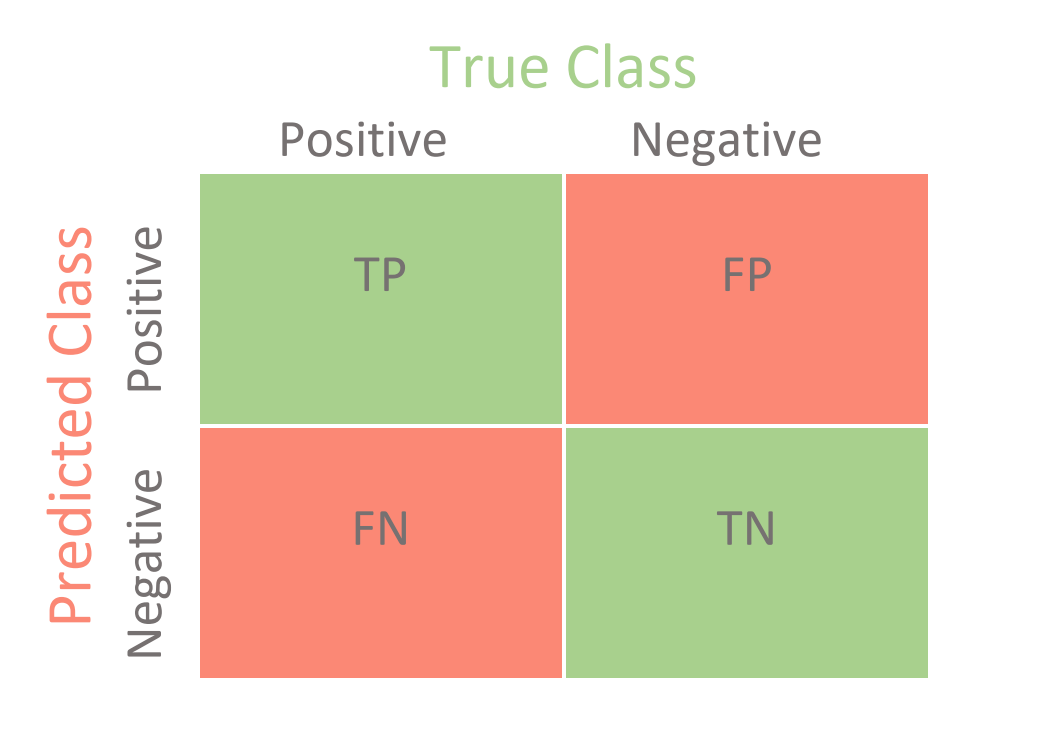
либо взвешенный вариант:

для некоторого ядра KK.

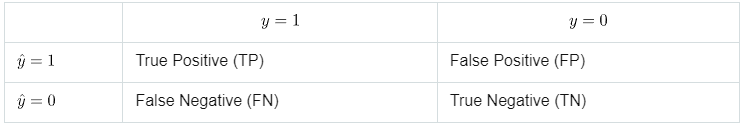
Последняя формула называется формулой Надарая — Ватсона, она является одним из непараметрических методов восстановления регрессии, объединённых названием, ядерная регрессия (kernel regression).

**Основные метрики в задаче классификации**





Для начала введём понятие **confusion matrix** **(матрица ошибок)** - в области машинного обучения и, в частности, проблемы статистической классификации, матрица неточностей, также известная как матрица ошибок, представляет собой конкретный макет таблицы, который позволяет визуализировать производительность алгоритма, обычно контролируемого обучения. Чаще всего отображается в графическом виде.



**Accuracy (доля правильных ответов):**

Интуитивно понятной, очевидной и почти неиспользуемой метрикой является accuracy — доля правильных ответов алгоритма:

Эта метрика бесполезна в задачах с неравными классами, и это легко показать на примере.

Допустим, мы хотим оценить работу спам-фильтра почты. У нас есть 100 не-спам писем, 90 из которых наш классификатор определил верно (True Negative = 90, False Positive = 10), и 10 спам-писем, 5 из которых классификатор также определил верно (True Positive = 5, False Negative = 5).

Тогда accuracy:

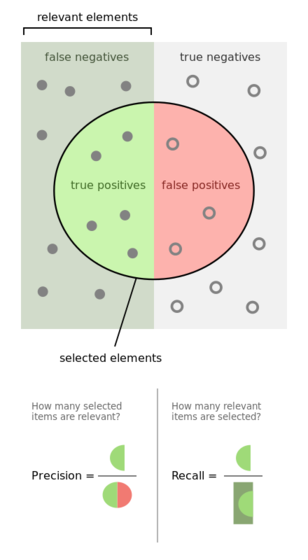
Однако если мы просто будем предсказывать все письма как не-спам, то получим более высокую accuracy:

При этом, наша модель совершенно не обладает никакой предсказательной силой, так как изначально мы хотели определять письма со спамом. Преодолеть это нам поможет переход с общей для всех классов метрики к отдельным показателям качества классов.

**Precision (точность), Recall (полнота) и F-мера:**

Для оценки качества работы алгоритма на каждом из классов по отдельности введем метрики precision (точность) и recall (полнота).

Precision можно интерпретировать как долю объектов, названных классификатором положительными и при этом действительно являющимися положительными, а recall показывает, какую долю объектов положительного класса из всех объектов положительного класса нашел алгоритм.



Именно введение 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, а precision — способность отличать этот класс от других классов.

Precision и recall не зависят, в отличие от accuracy, от соотношения классов и потому применимы в условиях несбалансированных выборок. Часто в реальной практике стоит задача найти оптимальный (для заказчика) баланс между этими двумя метриками. Классическим примером является задача определения оттока клиентов. Очевидно, что мы не можем находить **всех** уходящих в отток клиентов и **только** их. Но, определив стратегию и ресурс для удержания клиентов, мы можем подобрать нужные пороги по precision и recall. Например, можно сосредоточиться на удержании только высокодоходных клиентов или тех, кто уйдет с большей вероятностью, так как мы ограничены в ресурсах колл-центра.

Обычно при оптимизации гиперпараметров алгоритма (например, в случае перебора по сетке GridSearchCV ) используется одна метрика, улучшение которой мы и ожидаем увидеть на тестовой выборке.

Существует несколько различных способов объединить precision и recall в агрегированный критерий качества. F-мера— среднее гармоническое precision и recall:

Бета в данном случае определяет вес точности в метрике, и при равном 1 - это среднее гармоническое.

F-мера достигает максимума при полноте и точности, равными единице, и близка к нулю, если один из аргументов близок к нулю.

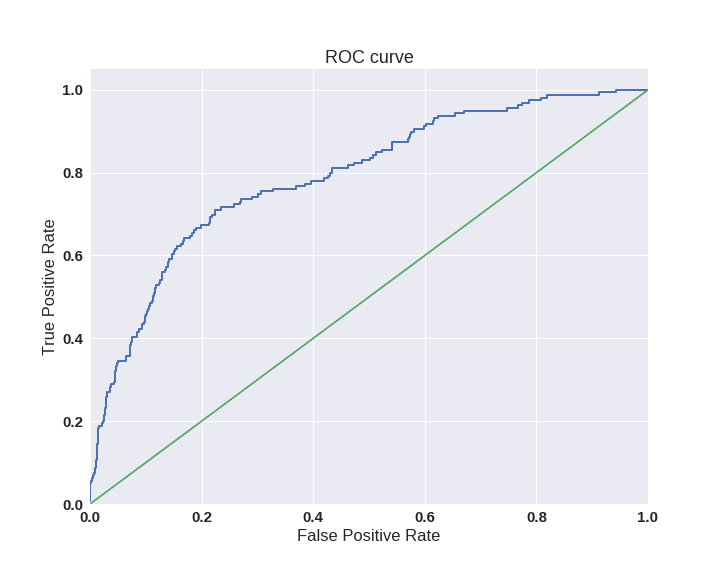
Здесь необходимо отметить, что в случае задач с несбалансированными классами, которые превалируют в реальной практике, часто приходится прибегать к техникам искусственной модификации датасета для выравнивания соотношения классов. Их существует много, и мы не будем их касаться.

**AUC-ROC:**

При конвертации вещественного ответа алгоритма в бинарную метку, мы должны выбрать какой-либо порог, при котором 0 становится 1. Естественным и близким кажется порог, равный 0.5, но он не всегда оказывается оптимальным, например, при вышеупомянутом отсутствии баланса классов.

Одним из способов оценить модель в целом, не привязываясь к конкретному порогу, является AUC-ROC (или 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 нам уже известна, это полнота, а FPR показывает, какую долю из объектов negative класса алгоритм предсказал неверно. В идеальном случае, когда классификатор не делает ошибок (FPR = 0, TPR = 1) мы получим площадь под кривой, равную единице; в противном случае, когда классификатор случайно выдает вероятности классов, AUC-ROC будет стремиться к 0.5, так как классификатор будет выдавать одинаковое количество TP и FP.  
Каждая точка на графике соответствует выбору некоторого порога. Площадь под кривой в данном случае показывает качество алгоритма (больше — лучше), кроме этого, важной является крутизна самой кривой — мы хотим максимизировать TPR, минимизируя FPR, а значит, наша кривая в идеале должна стремиться к точке (0,1).



Критерий AUC-ROC устойчив к несбалансированным классам (спойлер: увы, не всё так однозначно) и может быть интерпретирован как вероятность того, что случайно выбранный positive объект будет проранжирован классификатором выше (будет иметь более высокую вероятность быть positive), чем случайно выбранный negative объект.

Рассмотрим следующую задачу: нам необходимо выбрать 100 релевантных документов из 1 миллиона документов. Мы сделали два алгоритма:

**Алгоритм 1**возвращает 100 документов, 90 из которых релевантны. Таким образом,

**Алгоритм 2** возвращает 2000 документов, 90 из которых релеванты. Таким образом,

Скорее всего, мы бы выбрали первый алгоритм, который выдает очень мало False Positive на фоне своего конкурента. Но разница в False Positive Rate между этими двумя алгоритмами *крайне* мала — всего 0.0019. Это является следствием того, что AUC-ROC измеряет долю False Positive относительно True Negative и в задачах, где нам не так важен второй (больший) класс, может давать не совсем адекватную картину при сравнении алгоритмов.

Для того чтобы поправить положение, вернемся к полноте и точности:

**Алгоритм 1**

**Алгоритм 2**

Здесь уже заметна существенная разница между двумя алгоритмами — 0.855 в точности!

Precision и recall также используют для построения кривой и, аналогично AUC-ROC, находят площадь под ней.

**Как делить наборы**

Как вам известно машинное обучение отличается от задачи аппроксимации (приближения) функции тем, что мы не просто приближаем функцию, а пытаемся найти реальную закономерность (закон природы). Из этого следует, то, что нам просто необходимо оценка модели, на тех данных, которые модель не видела.

Для этого делят наборы на разные части:

Hold-out

Этот метод представляет собой обычное разделение на train и test части.



Стратификация

Этот методы представляет собой разделение на train и test части с учётом процентного соотношения классов в датасете.

K-fold

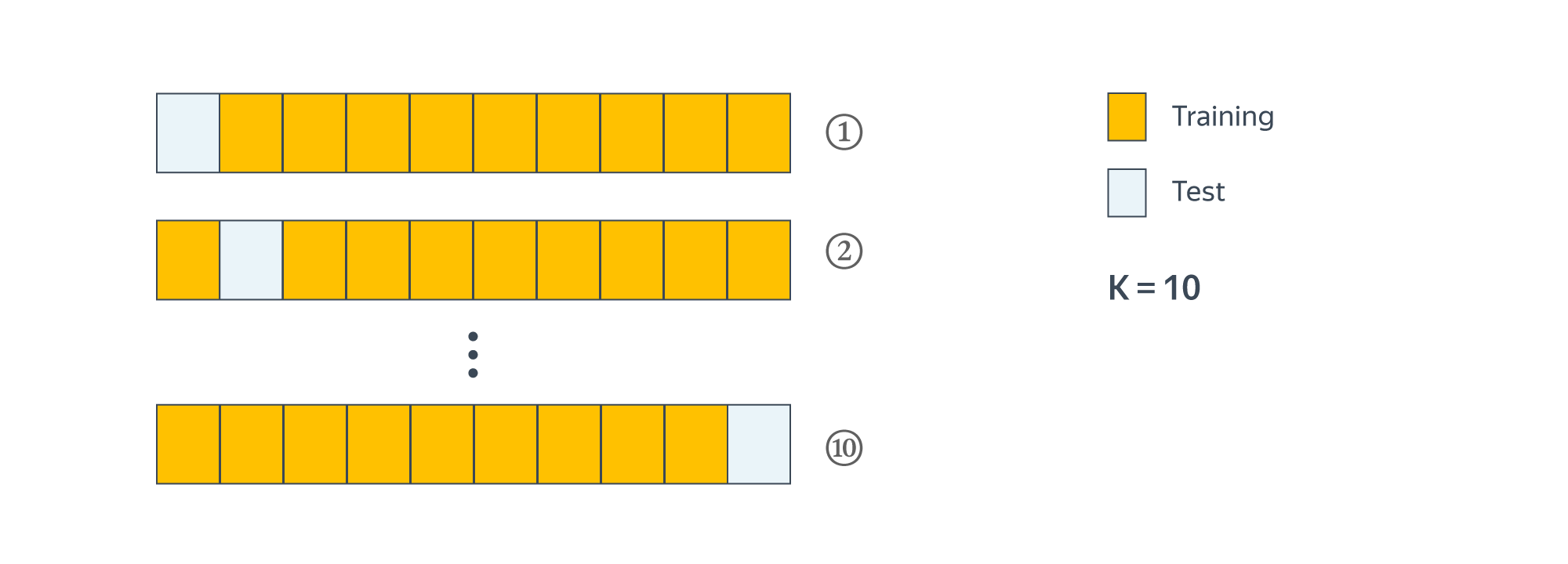
Метод k-Fold чаще всего имеют в виду, когда говорят о кросс-валидации. Он является обобщением метода hold-out и представляет из себя следующий алгоритм:

1) Фиксируется некоторое целое число k (обычно от 5 до 10), меньшее числа семплов в датасете.

2) Датасет разбивается на k одинаковых частей (в последней части может быть меньше семплов, чем в остальных). Эти части называются фолдами.

3) Далее происходит k итераций, во время каждой из которых один фолд выступает в роли тестового множества, а объединение остальных — в роли тренировочного. Модель учится на k−1 фолде и тестируется на оставшемся.

4) Финальный скор модели получается либо усреднением k получившихся тестовых результатов, либо измеряется на отложенном тестовом множестве, не участвовавшем в кросс-валидации.



Стратифицированный K-fold

Тот же K-fold только с учетом процентного соотношения в классах.

**Grid Search**

Параметр модели – изменяемые во время обучения переменные.

Гиперпараметр модели – неизменяемые (зафиксированные) во время обучения переменные.

Очень важно предоставить модели такие гиперпараметры, которые были бы оптимальными для текущей задачи. Для этого используют поиск по сетке.

Метод строит сетку из предоставленных гиперпараметров, проводя обучения с каждым из ячеек сетки, использует для оценки модели кросс валидацию.

