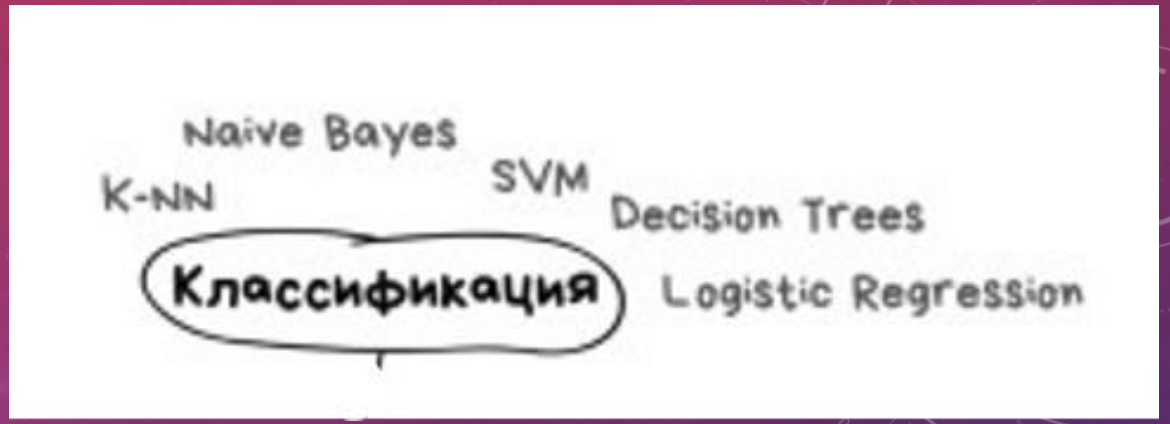
МЕТОДЫ МАШИННОГО ОБУЧЕНИЯ С УЧИТЕЛЕМ

КЛАССИФИКАЦИЯ


Машинное обучение
Классическое обучение
Без учителя
С учителем
Кластеризация
DBSCAN
K-Means
Agglomerative
Mean-Shift
Fuzzy C-Means
Классификация
Naive Bayes
K-NN
SVM
Decision Trees
Logistic Regression
Поиск правил
Euclat
Apriori
FP-Growth
Регрессия
Linear Regression
Polynomial Regression
Ridge/Lasso Regression
Уменьшение размерности (обобщение)
t-SNE
PCA
LSA
SVD
LDA
Ансамблевые методы
Стекинг
Беггинг
Random Forest
Бустинг
XGBoost
AdaBoost
LightGBM
CatBoost
Обучение с подкреплением
Генетический Алгоритм
Q-Learning
SARSA
Deep Q-Network (DQN)
A3C
Нейросети и глубокое обучение
Convolutional Neural Networks (CNN)
DCNN
Перцептроны (MLP)
Recurrent Neural Networks (RNN)
LSM
LSTM
GRU
Generative Adversarial Networks (GAN)
Autoencoders
seq2seq


Naive Bayes
K-NN
SVM
Decision Trees
Классификация
Logistic Regression

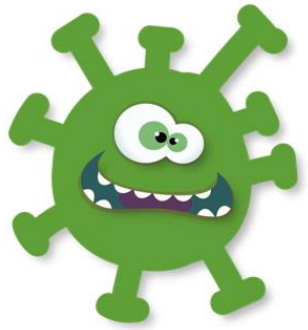
КЛАССИФИКАЦИЯ

- **Задача классификации в машинном обучении** — это задача отнесения объекта к одному из заранее определенных классов на основании его формализованных признаков. Каждый из объектов в этой задаче представляется в виде вектора в N-мерном пространстве, каждое измерение в котором представляет собой описание одного из признаков объекта. Для обучения классификатора необходимо иметь набор объектов, для которых заранее определены классы. Это множество называется **обучающей выборкой**, её разметка производится вручную, с привлечением специалистов в исследуемой области
- Задача классификации (как, впрочем и следует из названия) сводится к отнесению конкретного объекта к одному из заранее известных классов. Вот эта “метка” - название класса - и выступает в роли **целевой переменной**. И, совершенно естественно, что классов может быть только конечное количество, поэтому целевая переменная - дискретная.

ПРИМЕРЫ БИНАРНОЙ КЛАССИФИКАЦИИ

ИЛИ

ИЛИ



ИЛИ

АЛГОРИТМЫ КЛАССИФИКАЦИИ

Scikit-Learn обеспечивает легкий доступ к многочисленным различным алгоритмам классификации.

К числу таких классификаторов относятся:

- К-Ближайшие Соседи
- Логистическая регрессия
- Классификаторы дерева решений / Случайные леса
- Машины опорных Векторов
- Наивный Байес
- Линейный дискриминантный анализ

МЕТОД БЛИЖАЙШИХ СОСЕДЕЙ (K NEAREST NEIGHBORS)

Все объекты расположены в многомерном пространстве

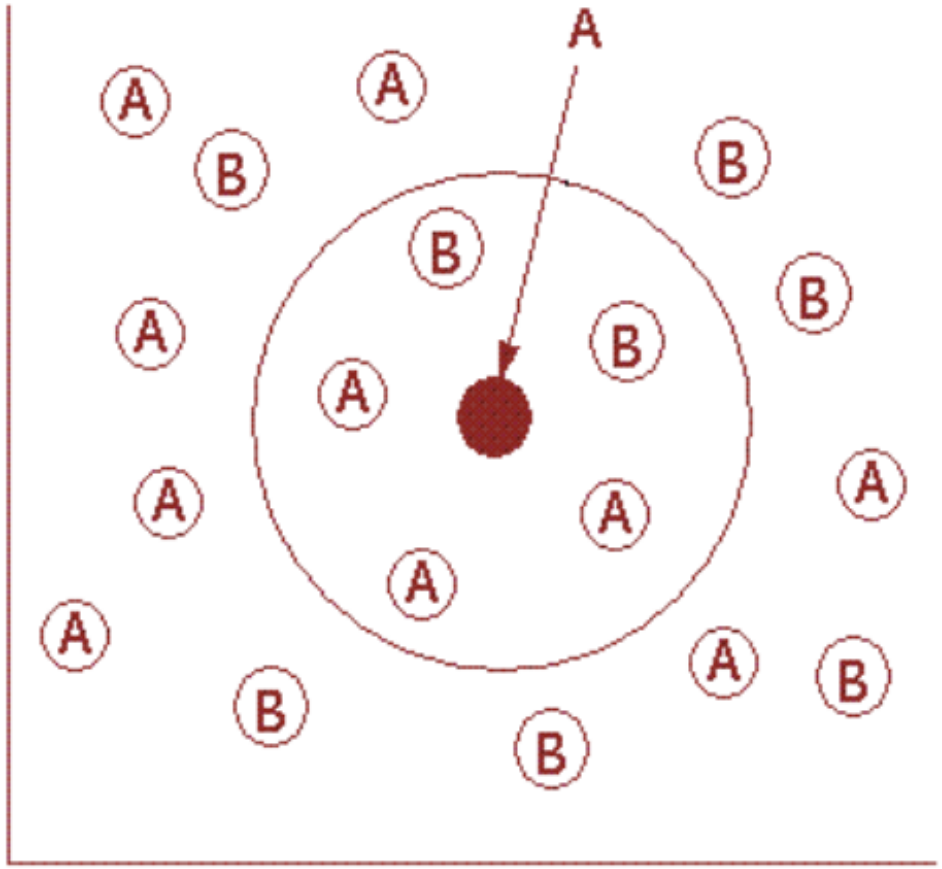
1. Задаем число *k* – количество ближайших соседей.

2. Ищем *k* объектов с минимальным расстоянием до нашего нового объекта. Используем меру для расчета расстояний.

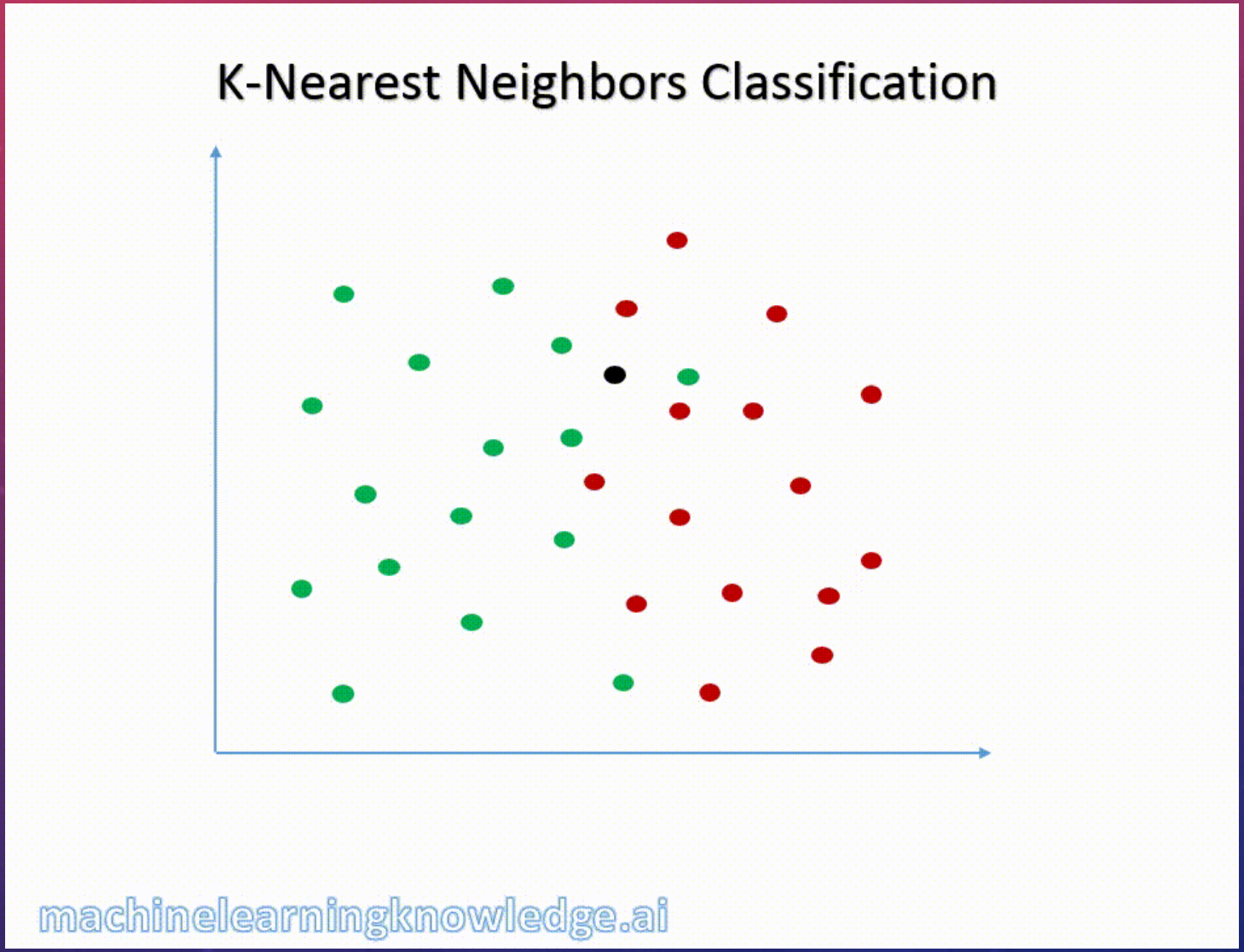
3.1 Простое невзвешенное голосование. Считаем сколько объектов с классами присутствует внутри заданного расстояния. Например, если число объектов с классом А большинство, то новый объект относится к классу А.

3.2. Взвешенное голосование

В такой ситуации учитывается также и расстояние до новой записи. Чем меньше расстояние, тем более значимый вклад вносит голос.



МЕТОД К-БЛИЖАЙШИХ СОСЕДЕЙ



МЕТОД К-БЛИЖАЙШИХ СОСЕДЕЙ

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
```

KNeighborsClassifier

KNeighborsClassifier()

1. Метрика расстояния (выбор влияет на результат) - metric:

1. **Евклидово расстояние**:

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

2. **Манхэттенское расстояние**:

$$d(x, y) = \sum_{i=1}^{n} |x_i - y_i|$$

3. **Косинусное расстояние** и другие.

2. Количество соседей (k) - n_neighbors:

1. **Малое** k (например, 1):
   Чувствительность к шуму, переобучение.
2. **Большое** k:
   Сглаживание границ, но риск недообучения.
   Оптимальное k выбирают через кросс-валидацию.

3. Функция голосования - weights:

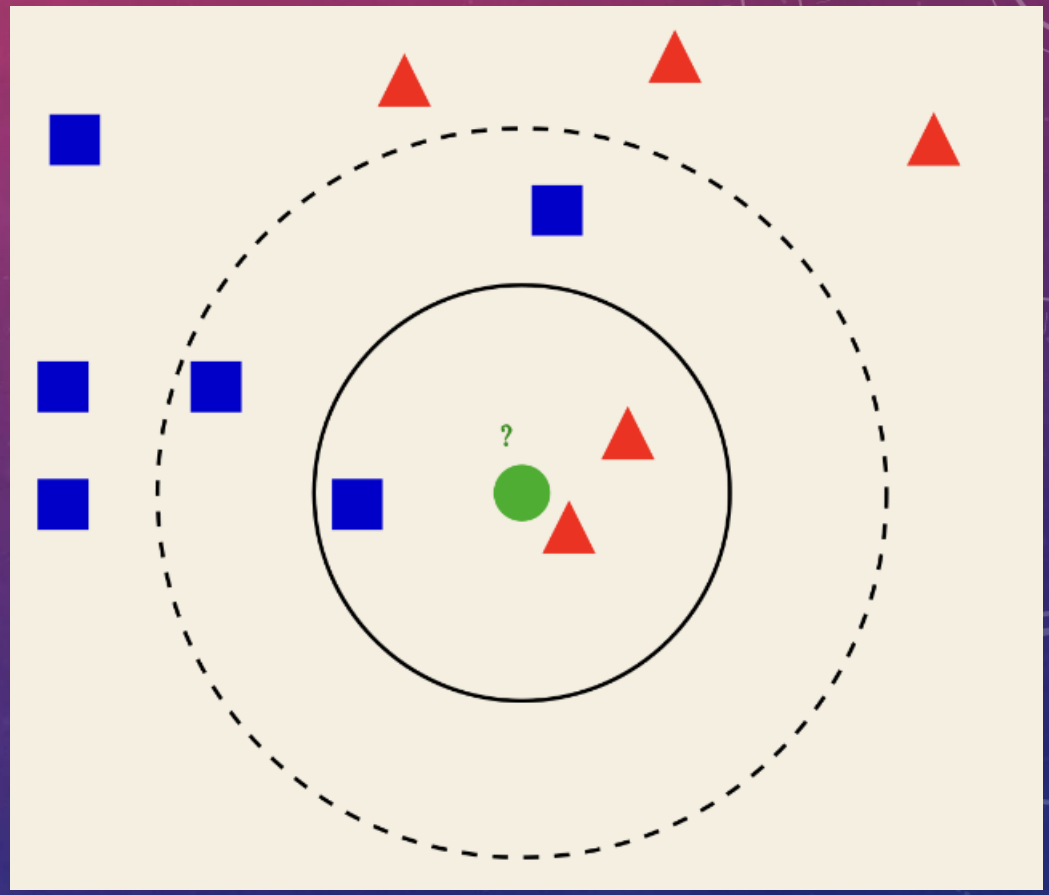
1. **Простое большинство**: класс, встречающийся чаще среди k соседей.
2. **Взвешенное голосование**: ближайшие соседи имеют больший вес.

4. Algorithm - алгоритм, используемый для поиска ближайших соседей: brute - полный перебор , ball_tree и kd_tree - на основе деревьев, auto - по умолчанию, автоматический выбор.

МЕТОД К-БЛИЖАЙШИХ СОСЕДЕЙ

Преимущества и недостатки

- Этот алгоритм довольно прост в своей реализации и устойчив к зашумленным обучающим данным. Даже если обучающие данные велики, они достаточно эффективны. Единственным недостатком алгоритма KNN является то, что нет необходимости определять значение K, а вычислительные затраты довольно высоки по сравнению с другими алгоритмами.



МЕТОД К-БЛИЖАЙШИХ СОСЕДЕЙ

Некоторые преимущества метода KNN:

- **Простота**. Алгоритм не требует сложных математических вычислений и может быть реализован с минимальными усилиями.
- **Отсутствие обучения**. Нет необходимости в обучении модели, что экономит время и ресурсы.
- **Гибкость**. Может использоваться для классификации и регрессии.

Некоторые недостатки метода KNN:

- **Высокие вычислительные затраты**. Для больших наборов данных вычисление расстояний может быть очень затратным.
- **Чувствительность к шуму**. KNN чувствителен к шуму и выбросам в данных.
- **Выбор К**. Оптимальное значение К может быть трудно определить.

ЛОГИСТИЧЕСКАЯ РЕГРЕССИЯ

Логистическая регрессия — это статистический метод для бинарной классификации, который предсказывает вероятность принадлежности объекта к одному из двух классов.

Основная идея: модель вычисляет вероятность того, что целевая переменная принадлежит к определенному классу. Для этого используется логистическая функция (сигмоида), которая преобразует линейную комбинацию признаков в значение между 0 и 1.

Уравнение линейной комбинации:

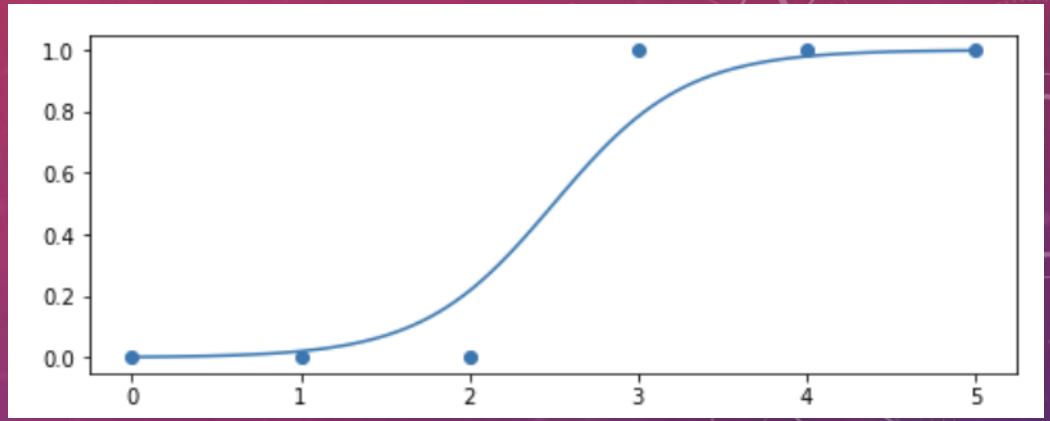
$$z = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n$$

где $w_0, w_1, \ldots, w_n$ — веса модели, а $x_1, x_2, \ldots, x_n$ — признаки.

ЛОГИСТИЧЕСКАЯ РЕГРЕССИЯ

Затем к z применяется сигмоида:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



Сигмоида преобразует z в вероятность $P(y = 1 \mid x) = \sigma(z)$.Если эта вероятность больше 0.5 (или другого порога), то объект относится к классу 1, иначе — к классу 0.

Обучение модели заключается в нахождении таких весов w, чтобы максимизировать правдоподобие (или минимизировать функцию потерь, например, логистическую потерю).

Логистическая регрессия выводит прогнозы о точках в бинарном масштабе — нулевом или единичном. Если значение чего-либо равно либо больше 0.5, то объект классифицируется в большую сторону (к единице). Если значение меньше 0.5 — в меньшую (к нулю).

У каждого признака есть своя метка, равная только 0 или только 1. Логистическая регрессия является линейным классификатором и поэтому используется, когда в данных прослеживается какая-то линейная зависимость.

ЛОГИСТИЧЕСКАЯ РЕГРЕССИЯ

Преимущества

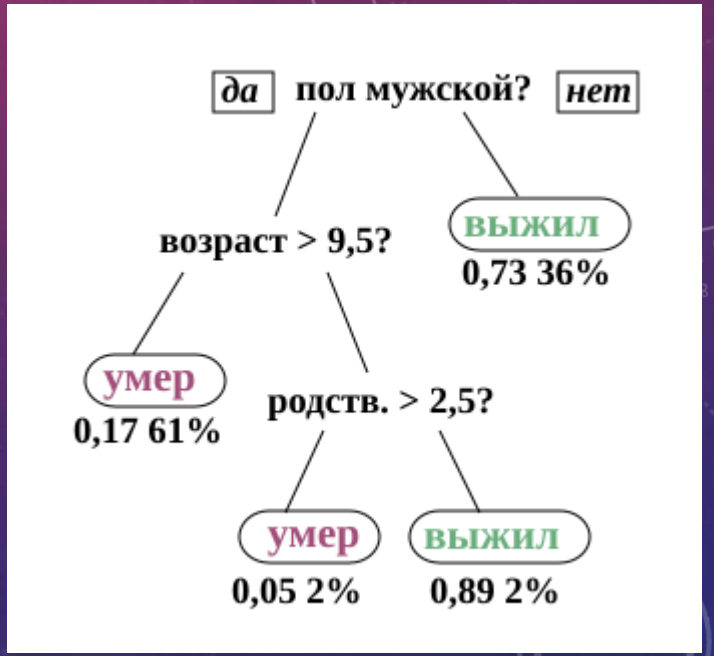
- Высокая интерпретируемость: можно анализировать важность признаков через веса
- Быстрое обучение и предсказание
- Выдает вероятности, а не только метки классов
- Устойчивость к шуму при использовании регуляризации
- Хорошо работает с линейно разделимыми данными

Недостатки

- Предполагает линейную зависимость между признаками и целевой переменной
- Чувствительность к мультиколлинеарности
- Может плохо работать с нелинейными задачами
- Требует масштабирования признаков
- Проблемы с несбалансированными данными (решается через параметр class_weight)

КЛАССИФИКАТОР ДЕРЕВА РЕШЕНИЙ (DECISION TREE CLASSIFIER)

- Дерево решений — это алгоритм, который строит модель в виде дерева, где каждый внутренний узел представляет признак, каждая ветвь — значение признака, а листовой узел — метку класса.
- **Суть алгоритма:** Алгоритм строит дерево, разбивая данные на подмножества на основе значения признаков.
- Цель — создать такие разбиения, чтобы в каждом листе оказались объекты одного класса (максимально однородные подмножества).
- Процесс построения дерева жадный: на каждом шаге выбирается признак, который лучше всего разделяет данные по заданному критерию (например, прирост информации, индекс Джини).
- Процесс продолжается рекурсивно для каждого подмножества до тех пор, пока не будет достигнута остановка (например, достигнута максимальная глубина или в узле остались объекты одного класса).



КЛАССИФИКАТОР ДЕРЕВА РЕШЕНИЙ (DECISION TREE CLASSIFIER)

- Этот классификатор разбивает данные на всё меньшие и меньшие подмножества на основе разных критериев, т. е. у каждого подмножества своя сортирующая категория. С каждым разделением количество объектов определённого критерия уменьшается.
- Классификация подойдёт к концу, когда сеть дойдёт до подмножества только с одним объектом. Если объединить несколько подобных деревьев решений, то получится так называемый *Случайный Лес* (англ. *Random Forest*).

КЛАССИФИКАТОР ДЕРЕВА РЕШЕНИЙ (DECISION TREE CLASSIFIER)

```
DecisionTreeClassifier(
    max_depth=5,            # Максимальная глубина дерева
    min_samples_split=20,   # Минимальное samples для разделения узла
    min_samples_leaf=10,    # Минимальное samples в листе
    max_leaf_nodes=30,      # Максимальное количество листьев
)
```

Основные параметры (на примере sklearn.tree.DecisionTreeClassifier):

criterion: Критерий для разделения узлов. Варианты: "gini" (индекс Джини) или "entropy" (прирост информации).

splitter: Стратегия разделения узлов. "best" (лучшее разделение) или "random" (случайное разделение).

max_depth: Максимальная глубина дерева. Если None, то дерево растет до тех пор, пока все листья не станут чистыми.

min_samples_split: Минимальное количество образцов, необходимое для разделения внутреннего узла.

min_samples_leaf: Минимальное количество образцов, которое должно находиться в листовом узле.

max_features: Количество признаков, рассматриваемых для лучшего разделения. Может быть фиксированным числом, процентом или строкой ("auto", "sqrt", "log2").

random_state: Фиксирует случайность для воспроизводимости.

class_weight: Веса классов. Может быть "balanced" или словарем.

КЛАССИФИКАТОР ДЕРЕВА РЕШЕНИЙ (DECISION TREE CLASSIFIER)

Преимущества

- простота и интерпретируемость, требует очень небольшой подготовки данных
- может работать с данными различных типов (категориальные, бинарные, числовые)
- может работать с несбалансированными данными
- может использоваться для отбора признаков

Некоторые недостатки:

- склонен к переобучению (overfitting) при большой глубине дерева или при отсутствии ограничений на минимальное количество объектов в листе
- неустойчив к шуму в данных
- не гарантирует наилучшее решение, так как выбор оптимального разделения происходит на каждом шаге по отдельности, а не глобально

НАИВНЫЙ БАЙЕСОВСКИЙ КЛАССИФИКАТОР (NAIVE BAYES)

Формула Байеса:

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)}$$

- Наивный байесовский классификатор — вероятностный классификатор на основе формулы Байеса со строгим (наивным) предположением о независимости признаков между собой при заданном классе.
- Такой классификатор вычисляет вероятность принадлежности объекта к какому-то классу. Эта вероятность вычисляется из шанса, что какое-то событие произойдёт, с опорой на уже на произошедшие события.
- Каждый параметр классифицируемого объекта считается независимым от других параметров.

НАИВНЫЙ БАЙЕСОВСКИЙ КЛАССИФИКАТОР (NAIVE BAYES)

Преимущества

Вычислительная эффективность:

- Быстрое обучение и предсказание
- Мало памяти для хранения модели
- Хорошо масштабируется на большие datasets

Простота и надежность:

- Мало параметров для настройки
- Устойчивость к шуму
- Хорошо работает с малым объемом данных

Вероятностный вывод:

- Возвращает вероятности принадлежности к классам
- Интерпретируемость результатов

Универсальность:

- Разные версии для разных типов данных
- Хорошо работает с текстовыми данными
- Эффективен в задачах с большим количеством признаков

Недостатки

Основное ограничение:

- "Наивное" предположение о независимости признаков редко выполняется в реальности
- Снижение качества при наличии коррелированных признаков

Ограничения точности:

- Обычно уступает по точности более сложным алгоритмам
- Плохо улавливает сложные нелинейные зависимости

Проблемы с данными:

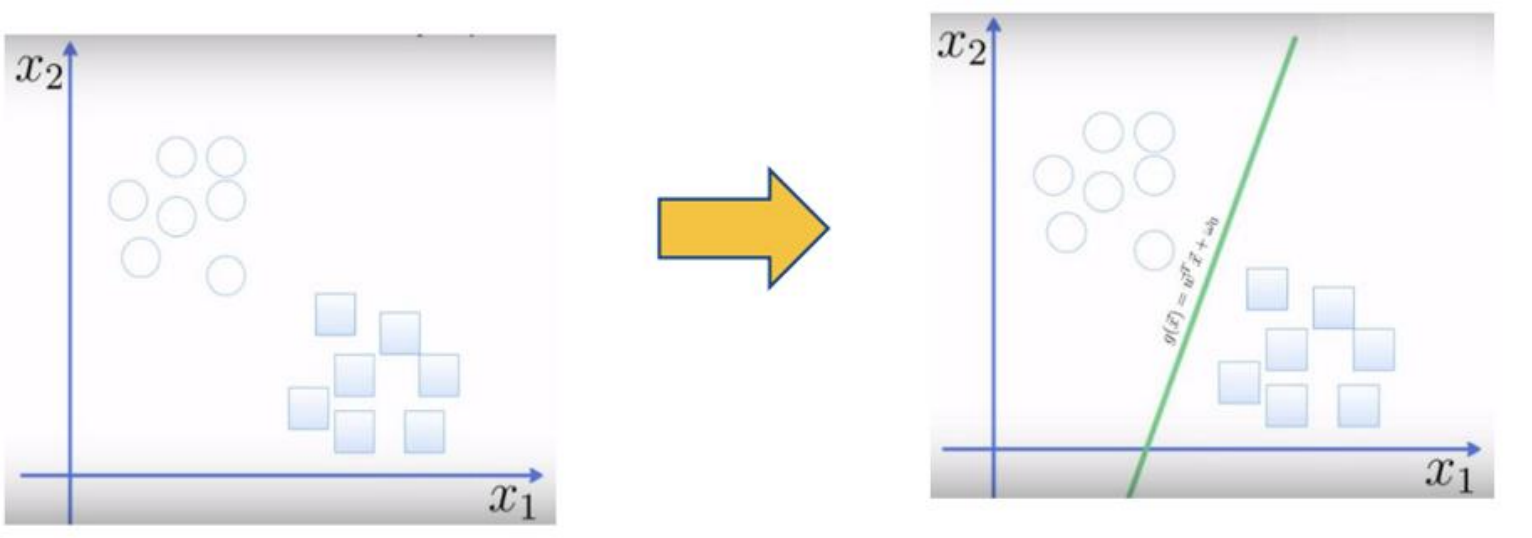
- Чувствительность к "проклятию размерности"
- Проблемы с нулевыми вероятностями (решается сглаживанием)

МЕТОД ОПОРНЫХ ВЕКТОРОВ

- SVM (Support Vector Machine) — это алгоритм, который ищет оптимальную разделяющую гиперплоскость с максимальным зазором между классами.

Каждый объект данных (например, документ, котировки ценных бумаг или компании) представлен как вектор в **P** мерном пространстве (последовательность чисел). Пусть у нас есть тестовая коллекция, в которой есть набор объектов (features) и есть набор классов. Математическая задача обучения заключается в том что бы найти функцию, которая адекватно сопоставляла объекты и классы, то есть найти такую функцию, которая эффективно разделяла бы объекты в пространстве features.

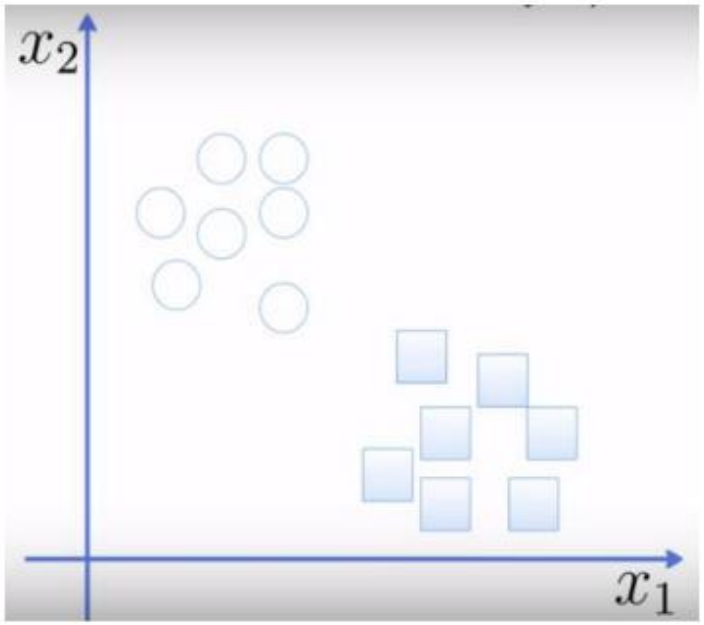
Рассмотрим пример на плоскости: У нас есть два класса с двумя features (x1, x2). Нужной найти прямую линию, которая оптимально разделяла два класса.

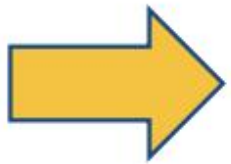


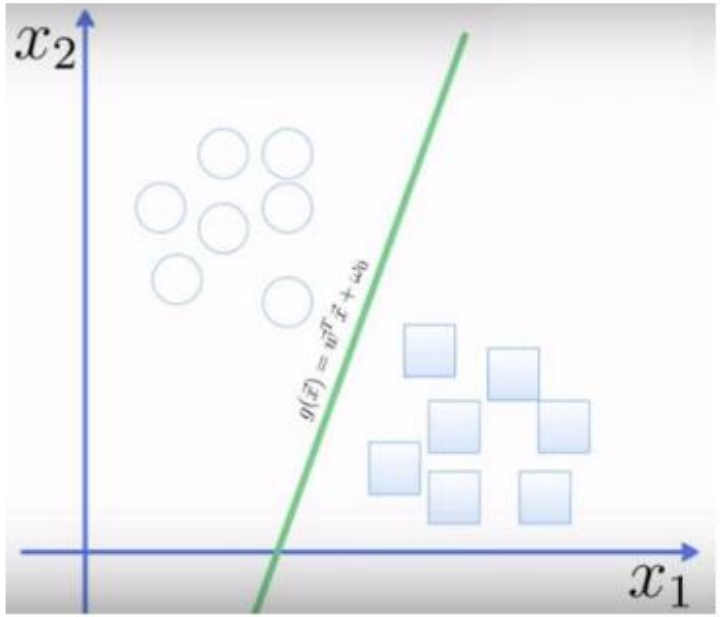
МЕТОД ОПОРНЫХ ВЕКТОРОВ (SVM)

Каждый объект данных (например, документ, котировки ценных бумаг или компании) представлен как вектор в **P** мерном пространстве (последовательность чисел). Пусть у нас есть тестовая коллекция, в которой есть набор объектов (features) и есть набор классов. Математическая задача обучения заключается в том что бы найти функцию, которая адекватно сопоставляла объекты и классы, то есть найти такую функцию, которая эффективно разделяла бы объекты в пространстве features.

Рассмотрим пример на плоскости: У нас есть два класса с двумя features (x1, x2). Нужной найти прямую линию, которая оптимально разделяла два класса.





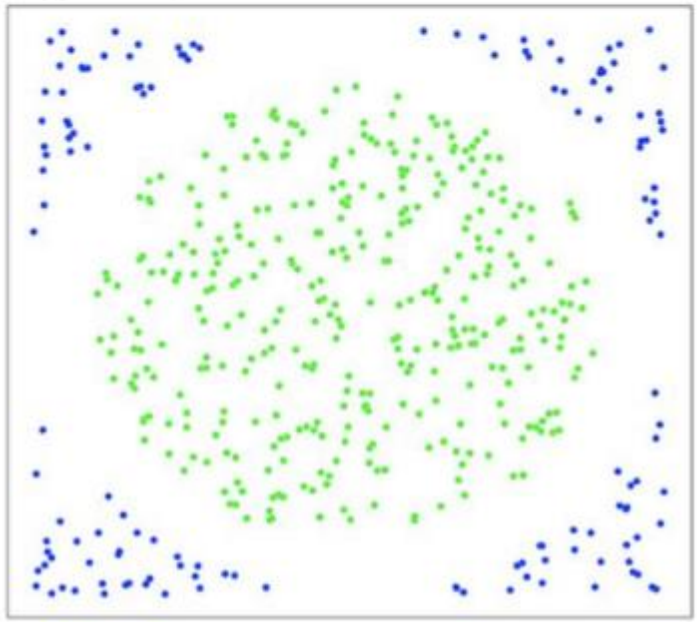


МЕТОД ОПОРНЫХ ВЕКТОРОВ (SVM)

Нахождение уравнения плоскости является стандартной задачей квадратичного программирования и решается с помощью множителей Лагранжа. Собственно в этом заключается процесс обучения.

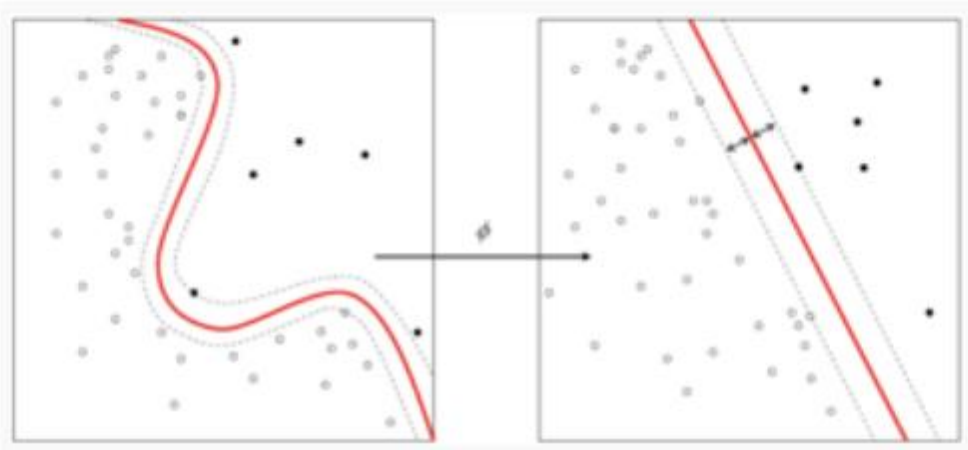
Как только плоскость найдена, берем новый объект и смотрим где он расположен относительно плоскости. Если справа, то принадлежит одному классу, если слева, то наш объект принадлежит другому классу.

Однако, ка правило на практике встречаются случаи когда объекты расположены, так что на плоскости невозможно провести разделяющую прямую. В этом случае плоскость вкладывается в пространство большей размерности. При вложении плоскость трансформируется таким образом, что бы появилась возможность провести разделяющую плоскость.



Демонстрация подобного преобразования

https://youtu.be/3liCbRZPrZA



SVM ищет гиперплоскость, которая разделяет данные на классы с максимальным зазором (margin).

Шаги алгоритма:

- Данные представляются как точки в n-мерном пространстве.
- Для линейно разделимых данных ищется гиперплоскость, которая разделяет классы с максимальным зазором.

Гиперплоскость задается уравнением: $w \cdot x + b = 0$, где:

- w — вектор весов (нормаль к гиперплоскости)
- b — смещение (bias)

Максимизация зазора (margin):

- Зазор — это расстояние от гиперплоскости до ближайших точек каждого класса (эти точки называются опорными векторами).
- Цель: максимизировать зазор, чтобы улучшить обобщающую способность модели.

Опорные векторы - это точки данных, которые находятся ближе всего к гиперплоскости и влияют на ее положение. Именно они определяют конечное решение модели.

Популярные ядра: линейное, полиномиальное, RBF (радиальная базисная функция), сигмоидальное.

Решающее правило:

- Для нового объекта x вычисляется значение: $f(x) = w \cdot x + b$
- Если $f(x) >= 0$, то объект относится к классу 1, иначе — к классу -1 (в бинарной классификации).

Плюсы и минусы классического SVM:

Преимущества:

- хорошо работает с пространством признаков большого размера;
- хорошо работает с данными небольшого объема;
- алгоритм максимизирует разделяющую полосу, которая, как подушка безопасности, позволяет уменьшить количество ошибок классификации;
- так как алгоритм сводится к решению задачи квадратичного программирования в выпуклой области, то такая задача всегда имеет единственное решение (разделяющая гиперплоскость с определенными гиперпараметрами алгоритма всегда одна).

Недостатки:

- долгое время обучения (для больших наборов данных);
- неустойчивость к шуму: выбросы в обучающих данных становятся опорными объектами-нарушителями и напрямую влияют на построение разделяющей гиперплоскости;
- не описаны общие методы построения ядер и спрямляющих пространств, наиболее подходящих для конкретной задачи в случае линейной неразделимости классов. Подбирать полезные преобразования данных – искусство.

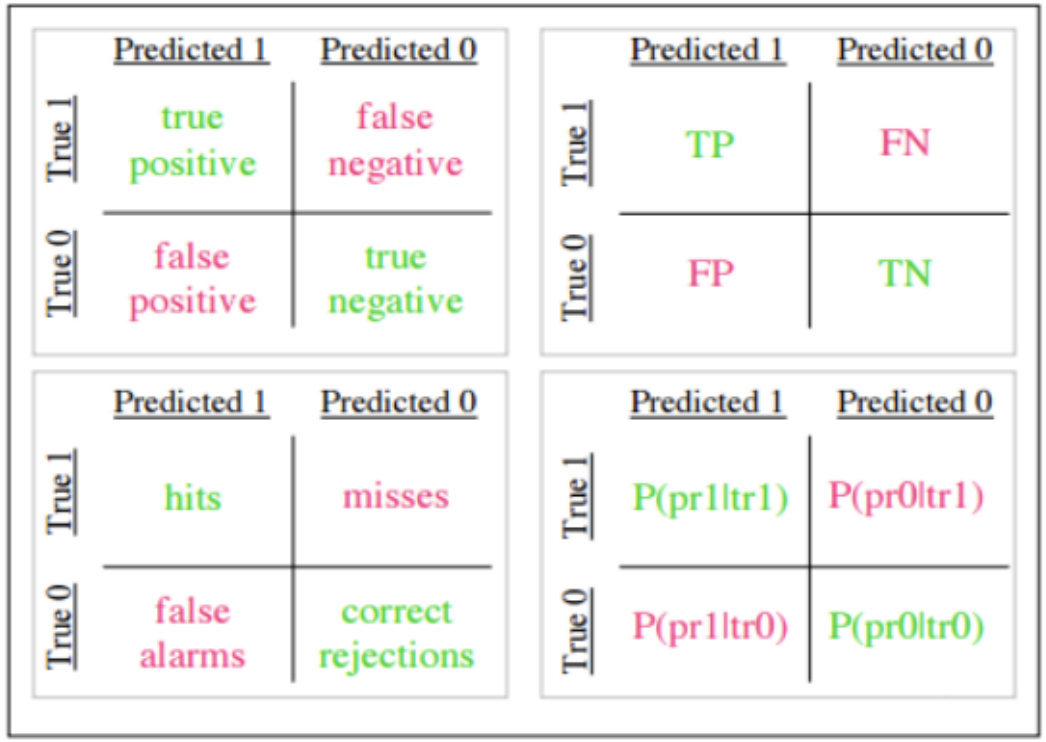
Меры качества классификаторов для бинарных классов

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

Precision: число правильно предсказанных положительных значений поделенных на число предсказанных классификатором положительных значений.

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

Recall : число правильно предсказанных положительных значений поделенных на число положительных ответов в данных.



Confusion Matrix

TP – число правильно предсказанных положительных значений

FN – число неправильно предсказанных положительных значений

FN – число правильно предсказанных негативных значений

FP – число неправильно предсказанных негативных значений

МЕТРИКИ КЛАССИФИКАЦИИ

	precision	recall	f1-score	support
0	0.87	0.82	0.85	186
1	0.72	0.80	0.76	109
accuracy			0.81	295
macro avg	0.80	0.81	0.80	295
weighted avg	0.82	0.81	0.82	295

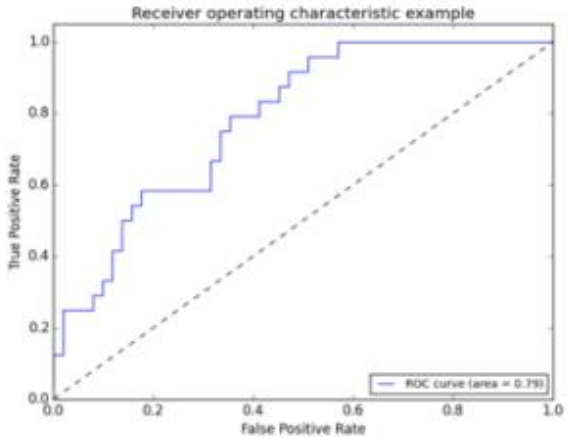
МЕРЫ КАЧЕСТВА КЛАССИФИКАТОРОВ ДЛЯ БИНАРНЫХ КЛАССОВ

F − measure

$$F - measure = \frac{(\beta^2 + 1) * precision * recall}{\beta^2 * precision + recall} \Longleftrightarrow \frac{(\beta^2+1)tp}{(\beta^2+1)tp+\beta^2 fn+fp}$$

β– обычно берут равной 1. F measure = 2 * (precision * recall) / (precision + recall)

The F measure (F1, Fscore) можно интерпретировать как взвешенное среднее precision и recall. Если F1=1, то классификатор отработал на 100% и F1=0 тогда классификатор не справился с задачей.



$$ROC = \frac{P(x|positive)}{P(x|negative)}$$

Рассчитывает отношение числа правильно распознанных случаев к числу не правильных. Процесс расчета таков: берутся данные, последовательно, и в них вычисляется это отношение. В какой то момент отношение становится константой.

AUC – интеграл под кривой.

Меры качества классификаторов для многих классов

$$\text{Precision} = \frac{\sum_{i=1}^{l} tp_i}{\sum_{i=1}^{l} (tp_i + fp_i)}$$

$$\text{Recall} = \frac{\sum_{i=1}^{l} tp_i}{\sum_{i=1}^{l} (tp_i + fn_i)}$$

$$\text{F measure} = \frac{(\beta^2 + 1)\,Precision_\mu Recall_\mu}{\beta^2\,Precision_\mu + Recall_\mu}$$

СПАСИБО ЗА ВНИМАНИЕ!