# Машинное обучение с учителем: линейные модели, измерение качества модели, ансамблевые модели

## Цель занятия

После освоения темы:

- вы познакомитесь с терминологией, используемой в машинном обучении;
- узнаете виды линейных моделей обучения, метрики измерения качества линейных моделей;
- узнаете базовые сведения об ансамблевых моделях и сравните производительность ансамблевых моделей;
- сможете строить модели линейной и логистической регрессии с использованием библиотек Python;
- сможете рассчитывать метрику качества линейной и логистической регрессии;
- сможете строить ансамблевые модели — решающее дерево, случайный лес, градиентный бустинг.

## План занятия

## Используемые термины

**Машинное обучение (Machine Learning)** — подход к анализу данных, который позволяет компьютерным системам изучать и делать прогнозы на основе данных без явного программирования.

**Обучение с учителем (supervised learning)** — алгоритм получает размеченные данные. Его цель — научиться предсказывать правильный ответ на новых данных.

**Обучение без учителя (unsupervised learning)** — алгоритм получает неразмеченные данные. Его цель — найти скрытые закономерности в данных.

**Глубокое обучение (Deep Learning)** — это подход к машинному обучению, использующий нейронные сети с большим количеством скрытых слоев.

**Обучение с подкреплением (Reinforcement Learning)** — метод, при котором модель обучается на основе награды или штрафа за каждое принятое решение.

**Полунадзорное обучение (Semi-Supervised Learning)** — метод, который используется в случаях, когда у нас есть небольшое количество маркированных данных и большое количество немаркированных данных.

**Активное обучение (Active Learning)** — метод, при котором модель активно запрашивает у эксперта метки для новых данных, чтобы улучшить свою точность.

**Функционал качества** — метрика, которая оценивает, насколько хорошо модель соответствует данным обучения.

**Среднеквадратичная ошибка (MSE)** — средняя величина квадрата отклонения прогнозируемых значений от фактических значений целевой переменной.

**Средняя абсолютная ошибка (MAE)** — среднее значение модуля отклонения прогнозируемых значений от фактических значений целевой переменной.

**Градиент** — направление наискорейшего роста функции.

**Градиентный спуск** — это метод оптимизации, который используется для минимизации функционала ошибки в линейных моделях.

**Логистическая регрессия** — метод машинного обучения, который используется для бинарной классификации, то есть разделения объектов на два класса.

**Матрица ошибок (Confusion matrix)** — таблица, которая показывает количество верных и неверных прогнозов модели бинарной классификации.

**Кросс-валидация (cross-validation)** — метод оценки качества модели машинного обучения на основе повторного выбора и разбиения данных на тренировочный и тестовый наборы.

**Регуляризация** — метод, используемый для снижения переобучения моделей машинного обучения путем добавления дополнительных ограничений на коэффициенты модели.

**Ансамблевые модели** — модели в машинном обучении, которые объединяют несколько моделей сразу, чтобы улучшить качество прогнозирования.

**Бэггинг (Bagging)** — метод, который заключается в использовании нескольких независимых моделей, каждая из которых обучается на своей подвыборке данных. После этого ответы каждой модели усредняются.

**Бустинг (Boosting)** — метод, который заключается в последовательном обучении нескольких слабых моделей, каждая из которых учится на подмножестве данных, где веса объектов учитывают ошибки предыдущих моделей.

**Решающее дерево** — структура, используемая в машинном обучении для прогнозирования значения целевой переменной.

**Максимальная глубина дерева (max_depth)** — гиперпараметр, который ограничивает максимальное количество узлов в дереве.

**Минимальное количество объектов для разделения (min_samples_split)** — гиперпараметр, который определяет минимальное количество объектов, необходимых для того, чтобы узел был разделен на две ветви.

**Случайный лес** — алгоритм машинного обучения, который строит множество решающих деревьев и усредняет их прогнозы, чтобы получить более точный прогноз.

**Градиентный бустинг** — это метод машинного обучения, который позволяет строить ансамбль из простых моделей, обычно деревьев решений.

## Конспект занятия

### 1. Введение

**Машинное обучение (Machine Learning)** — подход к анализу данных, который позволяет компьютерным системам изучать и делать прогнозы на основе данных без явного программирования.

ML использует алгоритмы, которые обучаются на данных и находят закономерности в этих данных, чтобы предсказывать результаты для новых данных.

Рассмотрим основные виды машинного обучения.

**Обучение с учителем (supervised learning)** — алгоритм получает размеченные данные. Его цель — научиться предсказывать правильный ответ на новых данных.

Примеры:

- Классификация электронных писем на спам и не спам.
- Предсказание цен на недвижимость на основе различных характеристик дома или квартиры.

**Обучение без учителя (unsupervised learning)** — алгоритм получает неразмеченные данные. Его цель — найти скрытые закономерности в данных.

Примеры:

- Кластеризация новостных статей на основе их содержания. Модель обучается на немаркированных данных (список новостных статей), чтобы определить, какие статьи относятся к одной и той же теме или кластеру.
- Снижение размерности изображений для ускорения их обработки. Модель обучается на немаркированных данных (наборы изображений), чтобы извлечь наиболее значимые признаки изображений и преобразовать их в более компактное представление.

**Глубокое обучение (Deep Learning)** — это подход к машинному обучению, использующий нейронные сети с большим количеством скрытых слоев. Часто глубокое обучение даже не относят к машинному обучению как таковому.

Примеры:

- Метод позволяет модели изучать более сложные закономерности в данных — распознавание изображений или голосовые команды.
- Глубокое обучение используется в таких областях, как распознавание речи, обработка естественного языка, компьютерное зрение и др.

**Обучение с подкреплением (Reinforcement Learning)** — метод, при котором модель обучается на основе награды или штрафа за каждое принятое решение.

Примеры:

- Обучение искусственного интеллекта в играх — в шахматах или Го.
- Беспилотные автомобили или дроны.

**Полунадзорное обучение (Semi-Supervised Learning)** — метод, который используется в случаях, когда у нас есть небольшое количество маркированных данных и очень большое количество немаркированных данных.

Примеры:

- Распознавание речи. Речь очень богата, в ней есть большое количество факторов (меток), и часто сделать их изначально до начала обучения модели довольно сложно.
- Выявление мошенничества в кредитных операциях. С каждым годом придумывают новые виды мошенничества, и старые модели, обученные давно, могут быть не очень актуальны.

**Активное обучение (Active Learning)** — метод, при котором модель во время своей работы активно запрашивает у эксперта метки для новых данных, чтобы улучшить свою точность в процессе обучения. Этот метод эффективен в случаях, когда метки для новых данных дороги, или их сложно получить.

Примеры:

- Классификация текстовых данных. Их может быть большое количество, часто их сложно отметить.
- Обнаружение аномалий в данных.

### Примеры применения машинного обучения

**ML и Google.** Машинное обучение позволяет Google анализировать и классифицировать миллионы веб-страниц, чтобы определить, какие из них наиболее релевантны для поискового запроса пользователя.

Google также использует машинное обучение в таких продуктах, как Google Photos, Google Translate и Google Assistant.

**ML и Netflix.** Netflix использует машинное обучение для персонализации контента, который предлагается каждому пользователю.

На основе предпочтений и просмотров пользователей Netflix создает индивидуальные рекомендации и рейтинги контента, которые помогают улучшить пользовательский опыт и увеличить число подписчиков.

**ML и Amazon.** Amazon использует машинное обучение для улучшения качества поиска и рекомендаций товаров, а также для оптимизации процесса логистики и управления инвентарем на складах.

Amazon также использует машинное обучение в продуктах, таких как Amazon Alexa и Amazon Prime.

## 2. Линейные модели

### Линейная регрессия

**Линейная регрессия** — одна из наиболее распространенных моделей машинного обучения. Модель используется для прогнозирования значений целевой переменной (зависимой переменной, отклика) на основе одной или нескольких независимых переменных (факторов, фичей, предикторов).

Примеры:

- Предсказание цены на недвижимость по ее параметрам.
- Предсказание конверсии веб-сайта по характеристикам его интерфейса.
- Предсказание инфляции в стране в текущий месяц.

Линейная регрессия представляет собой линейную связь между целевой переменной $y$ и независимыми переменными $x_1, x_2, \ldots, x_n$.

Модель строится на основе гипотезы о линейной зависимости:

$$y = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + \varepsilon,$$

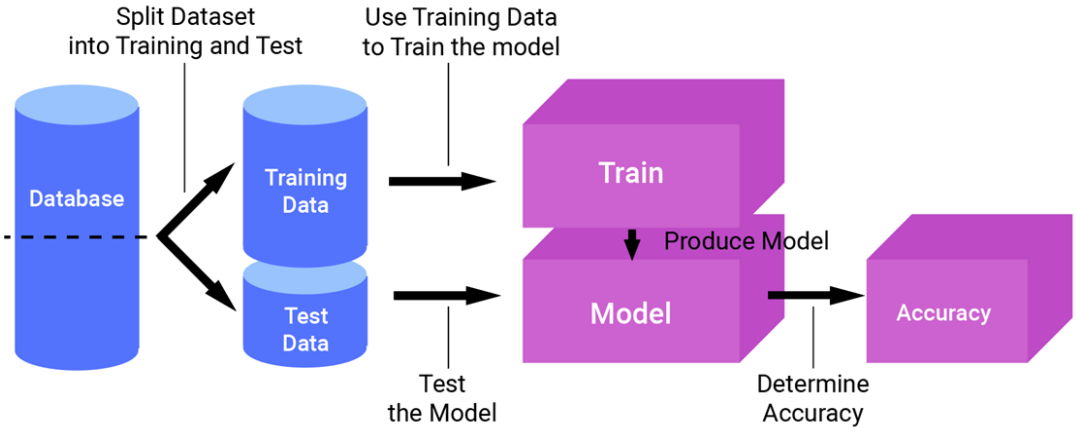
где $w_0, w_1, \ldots, w_n$ — коэффициенты (веса) регрессии,

$\varepsilon$ — ошибка (ни одна модель не может предсказать с нулевой ошибкой).

Цель обучения модели линейной регрессии — найти оптимальные значения коэффициентов $w_0, w_1, \ldots, w_n$, которые минимизируют ошибку модели.

Обычно это делается путем минимизации функции потерь — например, среднеквадратичной ошибки (MSE) или средней абсолютной ошибки (MAE).

Имеется некоторая выборка данных, которую мы делим на тренировочный и тестовый датасеты. Затем проводим обучение на тренировочном датасете, вырабатываем веса для факторов, и применяем полученный результат на тестовых данных.




**Функционал качества** — это метрика, которая оценивает, насколько хорошо модель соответствует данным обучения.

В случае линейной регрессии функционал качества измеряет разницу между прогнозируемыми и фактическими значениями целевой переменной.

Например:

- Насколько различается предсказанная цена на квартиру с реальной ценой этой квартиры?
- Какая разница между предсказанной Центральным Банком инфляцией и реальной инфляцией?

Функционал качества в линейной регрессии обычно выражается через **функцию потерь**. Она определяет, насколько сильно модель ошибается при прогнозировании значений целевой переменной.

Наиболее распространенные функции потерь в линейной регрессии — среднеквадратичная ошибка (mean squared error, MSE) и средняя абсолютная ошибка (mean absolute error, MAE).

**Среднеквадратичная ошибка (MSE)** — это средняя величина квадрата отклонения прогнозируемых значений от фактических значений целевой переменной:

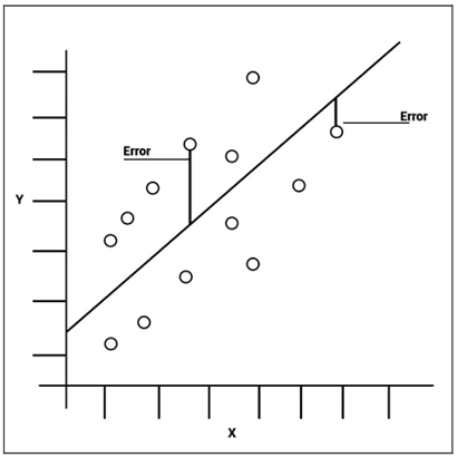
$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)^2$$

где $n$ — количество наблюдений,

$Y_i$ — фактическое значение целевой переменной,

$\hat{Y}_i$ — прогнозируемое значение целевой переменной.

Графическая интерпретация MSE:




**Средняя абсолютная ошибка (MAE)** — это среднее значение модуля отклонения прогнозируемых значений от фактических значений целевой переменной:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}\left|Y_i - \hat{Y}_i\right|$$

где $n$ — количество наблюдений,

$Y_i$ — фактическое значение целевой переменной,

$\hat{Y}_i$ — прогнозируемое значение целевой переменной.

Применение функционала качества:

- Оценка качества модели: функционал качества помогает оценить, насколько хорошо модель соответствует данным обучения.

- Сравнение моделей: позволяет сравнить качество разных моделей.
- Улучшение модели: может помочь выявить слабые места модели.
- Чувствительность к выбросам: функционал качества может быть чувствителен к выбросам, что может привести к искажению результатов оценки качества модели.

Стоит иметь в виду, что оптимизация функционала качества не гарантирует нахождение оптимальных параметров модели, особенно если функционал имеет много локальных оптимумов. В своем исследовании лучше использовать разные метрики качества и разные модели с разными подходами, чтобы подстраховать себя.

### Градиентный спуск

**Градиентный спуск** — метод оптимизации, который используется для минимизации функционала ошибки в линейных моделях.

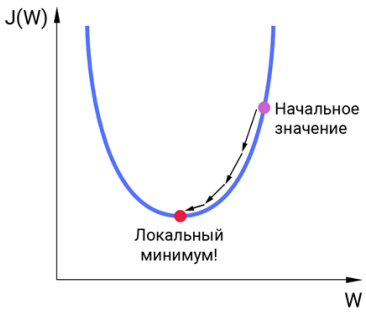
В линейной регрессии градиентный спуск используется для нахождения оптимальных значений вектора весов, которые минимизируют функцию ошибки.

**Градиент** — это направление наискорейшего роста функции. Поэтому градиент со знаком минус будет являться направлением наискорейшего убывания функции.

Почему это важно? В линейных моделях мы стремимся минимизировать функцию ошибки, и градиентный спуск нам тут и поможет.

Градиентный спуск работает путем итеративного изменения вектора весов модели в направлении, обратном градиенту функции ошибки. В этом случае градиент — это вектор первых производных функции ошибки по каждому из весов модели.

На каждом шаге градиентного спуска вектор весов изменяется на некоторую величину, называемую **скоростью обучения (learning rate)**.



### Применение линейных моделей

Рассмотрим пример предсказания цены на недвижимость. Представим, что у нас есть данные недвижимости по тысяче объектов города *X*. Есть следующие признаки:

- количество комнат ($x_1$);
- площадь ($x_2$);
- этаж ($x_3$);
- расстояние до центра ($x_4$);
- наличие балкона ($x_5$);
- год постройки ($x_6$);
- высота потолков ($x_7$);
- экология района ($x_8$);
- расстояние до метро ($x_9$);
- наличие парковки ($x_{10}$);
- цена ($y$).

| | кол-во комнат | площадь | этаж | расстояние до центра | наличие балкона | год постройки | высота потолков | экология района | расстояние до метро | наличие парковки | цена |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 3 | 46.448131 | 7 | 4.398358 | 0 | 1956 | 2.447177 | 0.346282 | 1.221261 | 1 | 6.495227e+06 |
| 1 | 4 | 52.997531 | 9 | 5.545063 | 0 | 2002 | 2.235449 | 0.633039 | 1.494096 | 1 | 3.738934e+06 |
| 2 | 4 | 40.991559 | 8 | 4.434747 | 0 | 1953 | 3.165007 | 0.701795 | 1.115183 | 0 | 5.297400e+06 |
| 3 | 2 | 56.392542 | 3 | 4.454554 | 0 | 1955 | 3.355077 | 0.473451 | 1.412109 | 0 | 4.407612e+06 |
| 4 | 3 | 57.982791 | 9 | 4.512172 | 0 | 2012 | 3.114077 | 0.591312 | 0.405313 | 0 | 4.069912e+06 |

Попробуем предсказать цену недвижимости по десяти параметрам, используя модель линейной регрессии.

Сначала применим метод наименьших квадратов (стандартный метод расчета весов) и градиентный спуск. Будем пользоваться библиотекой `sklearn` — стандартной библиотекой машинного обучения в Python.

Импортируем необходимый функционал:

```
from sklearn.linear_model import LinearRegression, SGDRegressor
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
```

Будем пытаться минимизировать MSE. Разделим датасет на матрицу независимых переменных и вектор зависимой переменной (цены):

```
X = data.drop('цена', axis=1)
y = data['цена']
```

Теперь поделим *X* и *Y* в соотношении 30% к 70%:

- 30% — тренировочная выборка (на ней обучаемся и подбираем параметры линейной регрессии).
- 70% — тестовая выборка (тестируем модель).

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)
```

**Обучаем модель линейной регрессии методом наименьших квадратов – методом, который используется по умолчанию:**

```
lr = LinearRegression()
lr.fit(X_train, y_train)
```

Предсказываем значения целевой переменной на тестовом наборе данных:

```
y_pred_lr = lr.predict(X_test)
```

Вычисляем MSE:

```
mse_lr = mean_squared_error(y_test, y_pred_lr)
print("MSE линейной регрессии:", mse_lr)
```

Результат выполнения кода: `MSE линейной регрессии: 878265280853.2183`

**Теперь обучим модель методом градиентного спуска:**

```
sgd = SGDRegressor()
sgd.fit(X_train, y_train)
```

Предсказываем значения целевой переменной на тестовом наборе:

```
y_pred_sgd = sgd.predict(X_test)
```

Вычисляем MSE:

```
mse_sgd = mean_squared_error(y_test, y_pred_sgd)
print("MSE SGDRegressor:", mse_sgd)
```

Результат выполнения кода: `MSE SGDRegressor: 2.2064467202927174e+29`

**Сравним MSE двух методов:**

```
mse_lr > mse_sgd
```

Результат: `False` — получаем, что обычный метод наименьших квадратов справился лучше.

### Логистическая регрессия

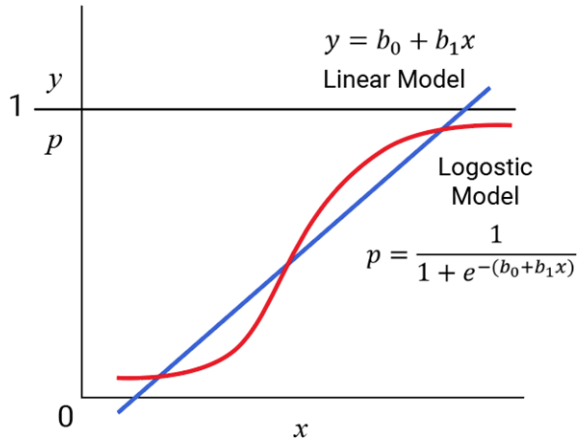
**Логистическая регрессия** — метод машинного обучения, который используется для бинарной классификации, то есть разделения объектов на два класса.

Обычная линейная регрессия хорошо работает для тех случаев, где зависимая переменная является численной и принимает большое количество значений. Для бинарной задачи обычная линейная регрессия не будет хорошо работать.

Для преобразования линейной комбинации признаков в вероятность принадлежности к классу используется функция сигмоиды:

$$y = \frac{1}{1+exp(-x)}$$

Функция потерь (log loss) используется для оценки качества логистической регрессии. Представляет собой логарифмическую функцию правдоподобия.




## 3. Измерение качества модели

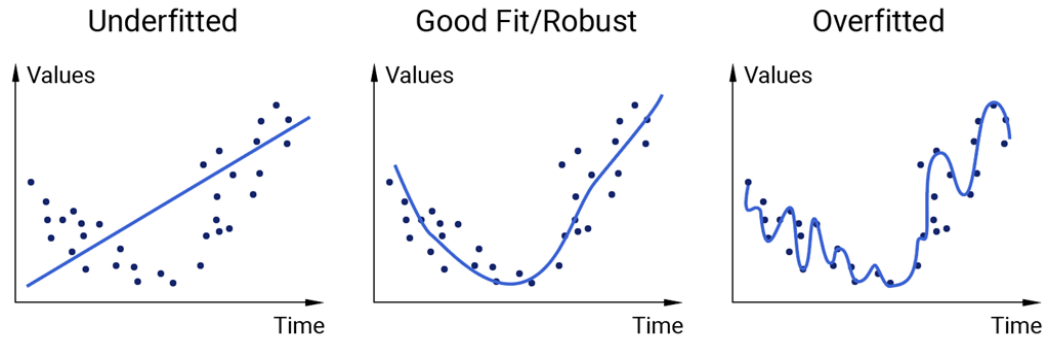
Качество моделей можно измерить с помощью метрик, которые отображают, насколько хорошо модель работает.

Часто может возникать **проблема переобучения (overfitting)** — модель слишком точно подгоняется под обучающие данные, и не покажет такого же качества на тестовых данных.

На рисунках представлены аппроксимации данных разными моделями.

В левом рисунке модель недообучена, плохо аппроксимируется линейной зависимостью, на рисунке посередине – хорошо обучена, аппроксимируется нелинейной функцией, на правом рисунке – переобучена, очень хорошо аппроксимируется полиномом высокого порядка.

В последнем случае модель будет идеально предсказывать тренировочные данные, но на тестовой выборке будет работать плохо.




В борьбе с переобучением есть три наиболее распространенных метода:

- Методы регуляризации:
    - L1 регуляризация.
    - L2 регуляризация.
- Кросс-валидация.

### Метрики качества

Основные метрики качества для линейной регрессии:

- **Mean Squared Error (MSE)** — средняя квадратичная ошибка. Показывает среднее значение квадрата разности между прогнозами модели и фактическими значениями целевой переменной.
- **Root Mean Squared Error (RMSE)** — корень из MSE, измеряет среднеквадратичное отклонение.
- **Mean Absolute Error (MAE)** — среднее значение модуля разностей между фактическими и прогнозируемыми значениями целевой переменной.

- **R-squared (R2)** — коэффициент детерминации, показывает долю объясненной вариации в зависимой переменной.

Один из способов измерения качества классификации логистической регрессии — это построение матрицы ошибок. **Матрица ошибок (Confusion matrix)** — это таблица, которая показывает количество верных и неверных прогнозов модели бинарной классификации.

| | **Предсказан класс 0** | **Предсказан класс 1** |
|---|---|---|
| **Настоящий Класс 0** | True Negative (TN) | False Positive (FP) |
| **Настоящий Класс 1** | False Negative (FN) | True Positive (TP) |

На основе абсолютных метрик TN, FP, FN и TP можно посчитать относительные метрики качества логистической регрессии:

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

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

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

$$F1 - score = 2\frac{Precision \cdot Recall}{Precision+Recall}$$

Ошибка первого рода $(false\ positive\ rate) = \frac{FP}{FP+TN}$ — неверно предсказанный позитивный класс.

Ошибка второго рода $(false\ negative\ rate) = \frac{FN}{FN+TP}$ — неверное предсказание отсутствия признака.

### Пример вычисления метрик качества: линейная регрессия

Посмотрим на конкретный пример расчета метрик на примере встроенного в библиотеку `scikit-learn` датасета цен на квартиры в Бостоне. Сначала импортируем необходимый функционал:

```
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
```

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error,
mean_squared_error, r2_score
import numpy as np
```

Загружаем датасет:

```
boston = load_boston()
```

Разделяем данные на признаки и целевую переменную:

```
X, y = boston.data, boston.target
```

Разделяем данные на тренировочный и тестовый датасеты:

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

Считаем MAE:

```
mae = mean_absolute_error(y_test, y_pred)
print(f"MAE: {mae:.2f}")
```

Результат: `MAE: 3.19`.

Считаем MSE:

```
mse = mean_squared_error(y_test, y_pred)
print(f"MSE: {mse:.2f}")
```

Результат: `MSE: 24.29`

Считаем RMSE:

```
rmse = np.sqrt(mse)
print(f"RMSE: {rmse:.2f}")
```

Результат: `RMSE: 4.93`

Считаем R2:

```
r2 = r2_score(y_test, y_pred)
print(f"R2: {r2:.2f}")
```

Результат: `R2: 0.67`

**Пример вычисления метрик качества: логистическая регрессия**

Воспользуемся датасетом, встроенным в библиотеку `scikit-learn`. Он содержит данные о свойствах клеток молочной железы, которые могут быть доброкачественными или злокачественными.

Задача — предсказание злокачественности клетки по биологическим и физическим характеристикам клетки.

Импортируем необходимые библиотеки:

```
from sklearn.datasets import load_breast_cancer
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, accuracy_score,
precision_score, recall_score, f1_score, auc, roc_auc_score,
roc_curve
import matplotlib.pyplot as plt
```

Загрузка данных:

```
data = load_breast_cancer()
```

Разделяем данные на тренировочный и тестовый датасеты:

```
data = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(data.data,
data.target, test_size=0.2, random_state=42)
```

Далее найдем требуемые метрики:

```
# Обучение модели
model = LogisticRegression(random_state=42, solver='liblinear')
model.fit(X_train, y_train)

# Предсказание меток для тестовой выборки
y_pred = model.predict(X_test)

# Вычисление accuracy
acc = accuracy_score(y_test, y_pred)
print("Accuracy:", acc)

# Вычисление precision
precision = precision_score(y_test, y_pred)
print("Precision:", precision)

# Вычисление recall
recall = recall_score(y_test, y_pred)
print("Recall:", recall)

# Вычисление f1-score
f1 = f1_score(y_test, y_pred)
```

```
print("F1-score:", f1)

# Ошибка 1 рода (false positive rate)
fpr = cm[0,1] / cm[0,:].sum()
print("False positive rate:", fpr)

# Ошибка 2 рода (false negative rate)
fnr = cm[1,0] / cm[1,:].sum()
print("False negative rate:", fnr)
```

Результат:

```
Accuracy: 0.956140350877193
Precision: 0.9459459459459459
Recall: 0.9859154929577465
F1-score: 0.9655172413793103
False positive rate: 0.09302325581395349
False negative rate: 0.014084507042253521
```

Важной метрикой в контексте логистической регрессии является расчет ROC-кривой, которая показывает взаимосвязь между False positive rate (FPR) и True positive rate (TPR). ROC-кривая может помочь посчитать метрику AUC – area under curve, площадь под ROC-кривой.

Получение вероятностей принадлежности классу 1:

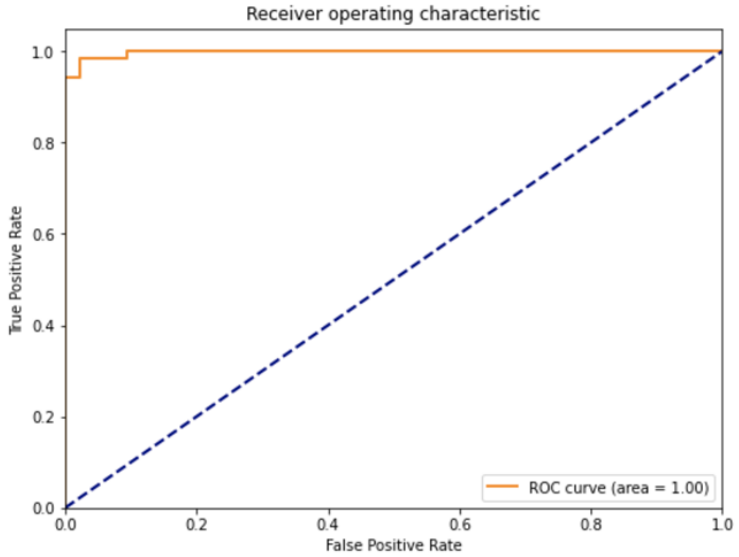
```
y_prob = model.predict_proba(X_test)[:, 1]
```

Вычисление ROC-кривой и площади под ней (ROC-AUC):

```
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
roc_auc = roc_auc_score(y_test, y_prob)
print("ROC-AUC:", roc_auc)
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve
(area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show()
```

Чем выше график ROC кривой к верхнему левому углу, тем лучше:

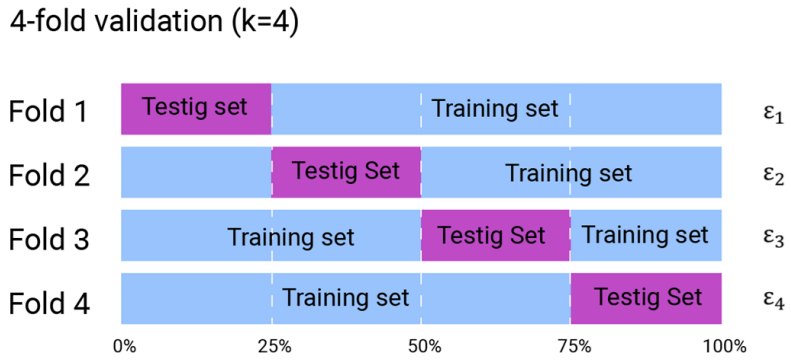
```
ROC-AUC: 0.9977071732721914
```




## Кросс-валидация

**Кросс-валидация (cross-validation)** — это метод оценки качества модели машинного обучения на основе повторного выбора и разбиения данных на тренировочный и тестовый наборы.

На рисунке представлен пример четырехступенчатой кросс-валидации:




Здесь датасет разделяли произвольным образом на ¼ и ¾ части. ¾ части брали в качестве обучающей выборки, а ¼ – в качестве тестировочной. По ходу разработки модели брали по очереди в качестве тестировочной выборки разные четверти.

Таким образом, каждое наблюдение в датасете выступает и в тестировочном, и в тренировочном качестве. Проводятся 4 обучения, и метрики таким способом получают более устойчивыми.

### Регуляризация

**Регуляризация** — это метод, используемый для снижения переобучения моделей машинного обучения путем добавления дополнительных ограничений на коэффициенты модели.

Существует два основных типа регуляризации — L1 и L2.

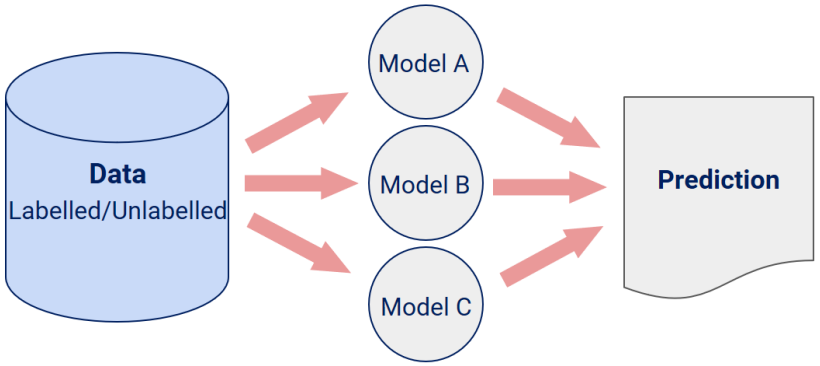
**L1-регуляризация** (Lasso-регрессия), добавляет в функцию потерь сумму абсолютных значений коэффициентов модели. В итоге из модели исключаются ненужные признаки, которые имеют малое влияние на целевую переменную, но усложняют модель и расчет.

**L2-регуляризация** (Ridge-регрессия), добавляет в функцию потерь квадрат суммы значений коэффициентов модели. Веса коэффициентов уменьшаются, уменьшая тем самым переобучение.

## 4. Ансамблевые модели

**Ансамблевые модели** — это такие модели в машинном обучении, которые объединяют несколько моделей сразу, чтобы улучшить качество прогнозирования.

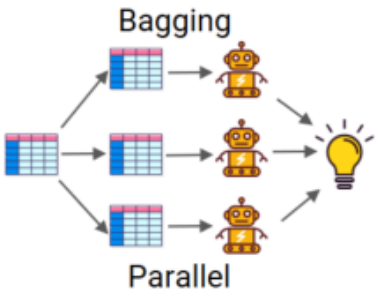
Идея состоит в том, чтобы использовать множество слабых моделей и объединить их вместе, чтобы получить более сильную модель.



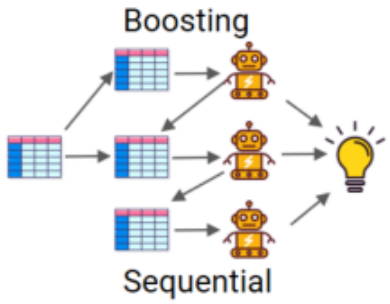

Среди ансамблемых моделей можно выделить два основных типа:

- **Бэггинг (Bagging)** — это метод, который заключается в использовании нескольких независимых моделей, каждая из которых обучается на своей

подвыборке данных. После этого ответы каждой модели усредняются. Пример — случайный лес.




- **Бустинг (Boosting)** — это метод, который заключается в последовательном обучении нескольких слабых моделей, каждая из которых учится на подмножестве данных, где веса объектов учитывают ошибки предыдущих моделей. Пример — градиентный бустинг AdaBoost, XGBoost.




Предсказание в контексте ансамблевых моделей:

| | Бэггинг | Бустинг |
|---|---|---|
| **Классификация** | Среднее голосование | Взвешенное голосование |
| **Регрессия** | Среднее значение | Взвешенное среднее |

### Решающие деревья

Решающие деревья часто являются основой ансамблевых моделей.

**Решающее дерево** — это структура, используемая в машинном обучении для прогнозирования значения целевой переменной. Решающее дерево состоит из узлов

и листьев, где узлы представляют собой решающие правила, а листья — предсказания.

Алгоритм построения решающего дерева:

1. Выбираем признак, который лучше всего разделяет выборку на две части.
2. Создаем узел, который разделяет выборку на две части на основе выбранного признака.
3. Рекурсивно повторяем шаги 1–2 для каждой из двух частей, пока не будет выполнено условие остановки. Например, достигнуто максимальное количество узлов, глубина дерева или ошибка на валидационной выборке.

Гиперпараметры – такие параметры, которые выбираются непосредственно разработчиком.

- **Максимальная глубина дерева (max_depth)** — это гиперпараметр, который ограничивает максимальное количество узлов в дереве. Если установлено слишком большое значение, дерево может переобучиться, если слишком маленькое — недообучиться.
- **Минимальное количество объектов в листе (min_samples_leaf)** определяет минимальное количество объектов, необходимых для того, чтобы узел стал листом.
- **Минимальное количество объектов для разделения (min_samples_split)** — это гиперпараметр, который определяет минимальное количество объектов, необходимых для того, чтобы узел был разделен на две ветви.

Решающие деревья, как отдельные самостоятельные модели, не всегда эффективны, поскольку склонны к переобучению, так как слишком хорошо подстраиваются под обучающую выборку.

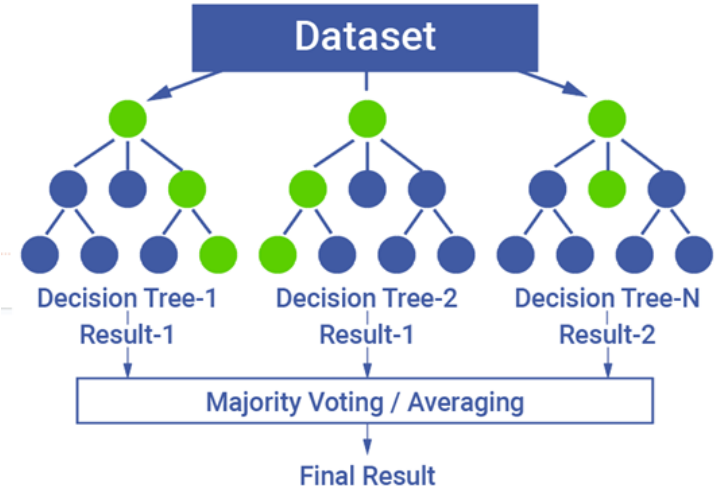
### Случайный лес

Случайный лес как комплекс множества деревьев будет показывать более качественную работу предсказаний и классификаций различных объектов.

**Случайный лес** — это алгоритм машинного обучения, который строит множество решающих деревьев и усредняет их прогнозы, чтобы получить более точный прогноз.

Он использует бэггинг, то есть обучает каждое дерево на подмножестве случайно выбранных объектов из обучающей выборки.

Случайный лес случайным образом выбирает подмножество признаков для каждого дерева и таким образом уменьшает корреляцию между деревьями.




Гиперпараметры случайного леса:

- **Количество деревьев (n_estimators)** — это гиперпараметр, который определяет количество деревьев в лесу. Чем больше деревьев, тем более точным может быть прогноз.
- **Максимальная глубина дерева (max_depth)** — применяется в случайном лесе, чтобы ограничить максимальное количество узлов в каждом дереве.
- **Минимальное количество объектов в листе (min_samples_leaf)** — определяет минимальное количество объектов, необходимых для того, чтобы узел стал листом.
- **Минимальное количество объектов для разделения (min_samples_split)** — определяет минимальное количество объектов, необходимых для того, чтобы узел был разделен на две ветви.
- **Количество признаков для случайного выбора (max_features)** — определяет количество признаков, которые будут случайно выбраны для каждого дерева. Это помогает уменьшить корреляцию между деревьями и повысить их разнообразие.

### Градиентный бустинг

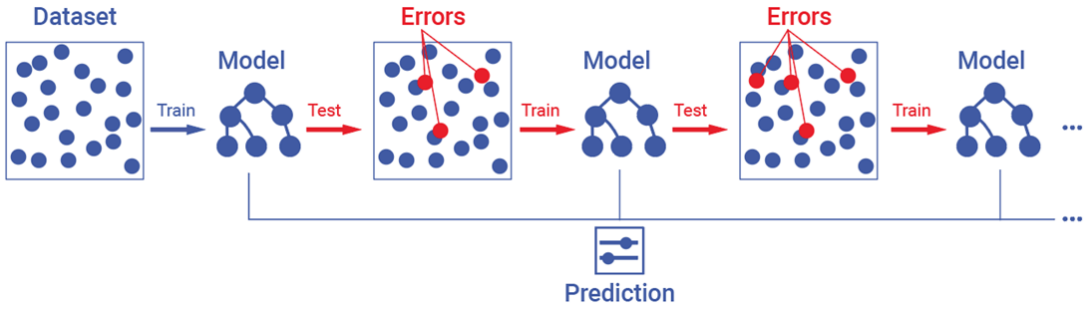
Наиболее популярным алгоритмом бустинга является градиентный бустинг.

**Градиентный бустинг** — это метод машинного обучения, который позволяет строить ансамбль из простых моделей, обычно деревьев решений.

Он основан на идее последовательного обучения моделей, где каждая следующая модель исправляет ошибки предыдущей:

- Процесс обучения градиентного бустинга начинается с построения базовой модели, например, дерева решений.
- Затем, используя функцию потерь, находится ошибка, сделанная базовой моделью, на каждом объекте обучающей выборки.
- Далее строится новая модель, которая пытается исправить ошибки, допущенные предыдущей моделью.

Для этого новая модель строится на основе градиентного спуска — метода оптимизации, который позволяет находить направление наискорейшего убывания функции потерь. На каждой итерации градиентного бустинга новая модель настраивается таким образом, чтобы минимизировать ошибки предыдущих моделей.




Гиперпараметры градиентного бустинга:

- **Коэффициент обучения (learning_rate)** — определяет величину, на которую будут изменяться веса при обучении каждой новой модели.
- **Количество деревьев(n_estimators)** — определяет количество базовых моделей, которые будут построены в процессе обучения.
- **Максимальная глубина дерева (max_depth)** — ограничивает глубину каждого дерева, чтобы предотвратить переобучение.
- **Размер подвыборки (subsample)** — определяет долю объектов обучающей выборки, которые будут использоваться для обучения каждой модели.

- **Функция потерь (Loss)** — определяет расхождение между предсказаниями модели и фактическими значениями целевой переменной, которое будет минимизироваться при обучении каждой модели.

### Применение ансамблевых моделей

В качестве примера применения изученных моделей рассмотрим уже знакомый нам набор данных Boston Housing. Цель заключается в прогнозировании медианной стоимости занятых владельцами домов в тысячах долларов на основе различных характеристик дома и его местоположения.

Мы сравним производительность моделей регрессии дерева решений, случайного леса и градиентного бустинга.

Сначала импортируем необходимый функционал:

```
import pandas as pd
from sklearn.datasets import load_boston
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor,
GradientBoostingRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
```

Загрузим датасет:

```
boston = load_boston()
X = pd.DataFrame(boston.data, columns=boston.feature_names)
y = pd.DataFrame(boston.target, columns=["MEDV"])
```

Поделим данные на тренировочные и тестовые:

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

Обучим простое дерево и посчитаем метрики качества:

```
dt = DecisionTreeRegressor(random_state=42)
dt.fit(X_train, y_train)
dt_y_pred = dt.predict(X_test)
dt_mse = mean_squared_error(y_test, dt_y_pred)
dt_r2 = r2_score(y_test, dt_y_pred)
```

Обучим случайный лес:

```
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
rf_y_pred = rf.predict(X_test)
rf_mse = mean_squared_error(y_test, rf_y_pred)
rf_r2 = r2_score(y_test, rf_y_pred)
```

Обучим модель градиентного бустинга:

```
gb = GradientBoostingRegressor(n_estimators=100,
learning_rate=0.1, random_state=42)
gb.fit(X_train, y_train)
gb_y_pred = gb.predict(X_test)
gb_mse = mean_squared_error(y_test, gb_y_pred)
gb_r2 = r2_score(y_test, gb_y_pred)
```

Выводим на экран метрики:

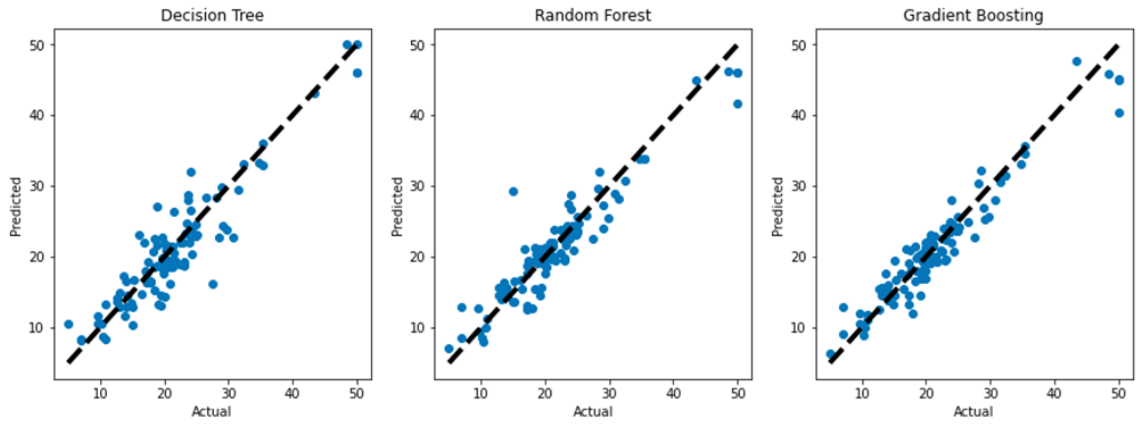
```
print("Decision Tree:")
print("MSE: ", dt_mse)
print("R^2: ", dt_r2)
print("\nRandom Forest:")
print("MSE: ", rf_mse)
print("R^2: ", rf_r2)
print("\nGradient Boosting:")
print("MSE: ", gb_mse)
print("R^2: ", gb_r2)
```

Результат выполнения:

```
Decision Tree:
MSE:  10.416078431372549
R^2:  0.8579634380978161

Random Forest:
MSE:  7.901513892156864
R^2:  0.8922527442109116

Gradient Boosting:
MSE:  6.208861361528038
R^2:  0.9153342280466539
```




Сравним модели по значению средней квадратичной ошибке (MSE) и метрике R2. Мы видим, что в случае градиентного бустинга MSE будет минимальной, а R2 –

максимальной. То есть в модели градиентного бустинга мы могли примерно объяснить 91,5% вариаций в ценах на квартиру.

## Дополнительные материалы для самостоятельного изучения

1. Unbiased Research
2. Введение в машинное обучение с помощью scikit-learn (перевод документации) / Хабр (habr.com)
3. 1. Supervised learning — scikit-learn 1.3.0 documentation