1. Понятие машинного обучения. Отличие машинного обучения от других областей программирования.

Машинное обучение (ML) — это область искусственного интеллекта, которая занимается разработкой алгоритмов, способных обучаться на данных и делать прогнозы или принимать решения. В отличие от традиционного программирования, где разработчик пишет четкие правила и инструкции для выполнения задач, в машинном обучении алгоритмы создают модели, которые обучаются на данных и затем применяют эти модели для новых данных.

Машинное обучение можно представить, как процесс, в котором компьютер учится выполнять задачу, используя данные для улучшения своих прогнозов или решений. Например, вместо того чтобы вручную программировать алгоритм для классификации писем как спам или не спам, система машинного обучения обучается на большом количестве писем, помеченных как спам и не спам, чтобы самостоятельно находить закономерности.

Основные отличия машинного обучения от традиционного программирования:

- **Традиционное программирование**: программисты пишут правила и логику вручную.
- **Машинное обучение**: алгоритмы обучаются и адаптируются на основе данных, что позволяет им улучшать свои прогнозы со временем.

2. Классификация задач машинного обучения. Примеры задач из различных классов.

Задачи машинного обучения можно классифицировать на три основные категории:

1. **Обучение с учителем (Supervised Learning)**:
    - **Классификация**: задача разделения данных на категории. Примеры: распознавание рукописных цифр, классификация писем как спам или не спам, определение типа заболевания по симптомам.
    - **Регрессия**: задача предсказания непрерывного значения. Примеры: прогнозирование цен на жилье, предсказание температуры на завтра, оценка спроса на продукцию.
2. **Обучение без учителя (Unsupervised Learning)**:
    - **Кластеризация**: задача группировки данных на основании схожести. Примеры: сегментация клиентов для маркетинга, группировка новостных статей по темам, выявление генетических кластеров.
    - **Поиск аномалий**: выявление необычных или редких данных. Примеры: обнаружение мошенничества в банковских транзакциях, выявление дефектов в производственных процессах, обнаружение аномалий в сети.
3. **Обучение с подкреплением (Reinforcement Learning)**:
    - Задача, при которой агент учится, взаимодействуя с окружающей средой и получая за это вознаграждения или штрафы. Примеры: игра в шахматы, управление роботами, оптимизация торговых стратегий.

3. Основные понятия машинного обучения: набора данных, объекты, признаки, атрибуты, модели, параметры.

- **Набор данных**: совокупность данных, используемых для обучения и тестирования модели. Набор данных обычно делится на тренировочный (для обучения модели) и тестовый (для оценки качества модели).
- **Объекты (экземпляры)**: индивидуальные элементы набора данных. Например, каждый человек в наборе данных о населении или каждая транзакция в банковских данных.
- **Признаки (фичи)**: атрибуты или переменные, которые описывают объекты. Например, возраст, рост, вес, доход.
- **Атрибуты**: термин, синонимичный признакам.
- **Модели**: математическое представление алгоритма, который обучается на данных для выполнения предсказаний или решений. Модель может быть линейной или нелинейной, простой или сложной.
- **Параметры**: числовые значения внутри модели, которые изменяются в процессе обучения для минимизации функции ошибки. Например, коэффициенты в линейной регрессии.

4. Структура и представление данных для машинного обучения.

Данные для машинного обучения обычно представляются в виде матрицы, где строки представляют объекты, а столбцы — признаки. Важные аспекты структуры данных включают:

- **Чистота данных**: данные должны быть очищены от пропусков, дубликатов и ошибок. Это важно для повышения качества модели.
- **Формат данных**: данные должны быть в числовом формате для большинства алгоритмов машинного обучения. Категориальные данные часто кодируются числовыми значениями.
- **Нормализация и стандартизация**: процессы приведения данных к определенному масштабу. Нормализация масштабирует данные в диапазоне [0, 1], а стандартизация преобразует данные так, чтобы они имели среднее значение 0 и стандартное отклонение 1. Эти методы помогают ускорить обучение и улучшить качество моделей.

5. Инструментальные средства машинного обучения.

Существует множество инструментов и библиотек для машинного обучения, среди которых наиболее популярны:

- **Scikit-learn**: библиотека Python для машинного обучения, включающая множество алгоритмов и инструментов для предобработки данных, обучения и оценки моделей.
- **TensorFlow** и **Keras**: фреймворки для глубокого обучения, разработанные Google. TensorFlow предоставляет широкие возможности для создания и обучения сложных моделей, а Keras является высокоуровневым API, облегчающим разработку моделей.
- **PyTorch**: фреймворк для глубокого обучения, разработанный Facebook. PyTorch популярен благодаря своей гибкости и удобству в использовании, особенно для исследований и разработки новых алгоритмов.
- **Pandas**: библиотека для анализа данных на Python, полезная для подготовки и обработки данных. Pandas предоставляет мощные инструменты для манипуляции данными, такими как фильтрация, агрегация и преобразование данных.
- **Jupyter Notebooks**: интерактивная среда для выполнения кода, визуализации данных и создания отчетов. Jupyter Notebooks широко используется для экспериментов с данными и демонстрации результатов.

6. Задача регрессии: постановка, математическая формализация.

Задача регрессии заключается в предсказании непрерывного значения на основе набора признаков. Формализация задачи регрессии может быть представлена следующим образом:

- **Входные данные**: X=[x1,x2,…,xn]X = [x_1, x_2, \ldots, x_n]X=[x1,x2,…,xn] — матрица объектов-признаков.
- **Выходные данные**: y=[y1,y2,…,yn]y = [y_1, y_2, \ldots, y_n]y=[y1,y2,…,yn] — вектор целевых значений.
- **Модель**: y^=f(X)\hat{y} = f(X)y^=f(X), где fff — функция, приближающая зависимость между XXX и yyy.

Пример линейной регрессии: y=β0+β1x1+β2x2+…+βnxn+ϵy = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n + \epsilony=β0+β1x1+β2x2+…+βnxn+ϵ где βi\beta_iβi — коэффициенты модели, а ϵ\epsilonϵ — ошибка модели.

Пример кода на Python:

```
python
Копировать код
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Создание примера набора данных
data = {
    'x1': [1, 2, 3, 4, 5],
    'x2': [2, 3, 4, 5, 6],
    'y': [2, 3, 4, 5, 6]
}
df = pd.DataFrame(data)

# Разделение данных на тренировочный и тестовый наборы
```

```
X = df[['x1', 'x2']]
y = df['y']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Создание и обучение модели
model = LinearRegression()
model.fit(X_train, y_train)

# Предсказание
y_pred = model.predict(X_test)

# Оценка модели
mse = mean_squared_error(y_test, y_pred)
print(f'MSE: {mse}')
```

7. Метод градиентного спуска для парной линейной регрессии.

Метод градиентного спуска используется для минимизации функции ошибки (например, среднеквадратичной ошибки) путем итеративного обновления параметров модели. Для парной линейной регрессии (один признак):

1. **Модель**: y=β0+β1xy = \beta_0 + \beta_1 xy=β0+β1x
2. **Функция ошибки (MSE)**: J(β0,β1)=1m∑i=1m(y^i−yi)2J(\beta_0, \beta_1) = \frac{1}{m} \sum_{i=1}^m (\hat{y}_i - y_i)^2J(β0,β1)=m1∑i=1m(y^i−yi)2
3. **Обновление параметров**: β0:=β0−α∂J∂β0\beta_0 := \beta_0 - \alpha \frac{\partial J}{\partial \beta_0}β0:=β0−α∂β0∂J β1:=β1−α∂J∂β1\beta_1 := \beta_1 - \alpha \frac{\partial J}{\partial \beta_1}β1:=β1−α∂β1∂J

где α\alphaα — шаг обучения, а ∂J∂β0\frac{\partial J}{\partial \beta_0}∂β0∂J и ∂J∂β1\frac{\partial J}{\partial \beta_1}∂β1∂J — частные производные функции ошибки по параметрам.

Пример кода на Python:

```
python
Копировать код
import numpy as np

# Градиентный спуск для парной линейной регрессии
def gradient_descent(X, y, lr=0.01, epochs=1000):
    m = len(y)
    theta = np.zeros(2)
    X_b = np.c_[np.ones((m, 1)), X]  # добавление x0 = 1 для каждого экземпляра

    for epoch in range(epochs):
        gradients = 2/m * X_b.T.dot(X_b.dot(theta) - y)
        theta -= lr * gradients

    return theta

# Пример данных
X = np.array([1, 2, 3, 4, 5])
y = np.array([2, 2.8, 3.6, 4.4, 5.2])

# Обучение модели
theta = gradient_descent(X, y)
print(f'Параметры модели: {theta}')
```

8. Понятие функции ошибки: требования, использование, примеры.

Функция ошибки (или функция потерь) — это мера, которая показывает, насколько точны предсказания модели по сравнению с реальными данными. Требования к функции ошибки включают:

- **Непрерывность**: функция должна быть непрерывной для применения градиентного спуска.
- **Дифференцируемость**: должна существовать возможность вычисления градиента.
- **Неположительность**: значения функции должны быть минимизированы для улучшения модели.

Примеры:

- **Среднеквадратичная ошибка (MSE)**: J(θ)=1m∑i=1m(y^i−yi)2J(\theta) = \frac{1}{m} \sum_{i=1}^m (\hat{y}_i - y_i)^2J(θ)=m1∑i=1m(y^i−yi)2
- **Средняя абсолютная ошибка (MAE)**: J(θ)=1m∑i=1m|y^i−yi|J(\theta) = \frac{1}{m} \sum_{i=1}^m |\hat{y}_i - y_i|J(θ)=m1∑i=1m|y^i−yi|
- **Кросс-энтропийная ошибка**: используется в задачах классификации.

9. Множественная и нелинейная регрессии.

- **Множественная регрессия**: расширение линейной регрессии для нескольких признаков. Формула: y=β0+β1x1+β2x2+…+βnxn+ϵy = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n + \epsilony=β0+β1x1+β2x2+…+βnxn+ϵ
- **Нелинейная регрессия**: модели, которые могут описывать нелинейные зависимости между признаками и целевой переменной. Примеры:
  - Полиномиальная регрессия: y=β0+β1x+β2x2+…+βnxn+ϵy = \beta_0 + \beta_1 x + \beta_2 x^2 + \ldots + \beta_n x^n + \epsilony=β0+β1x+β2x2+…+βnxn+ϵ
  - Логистическая регрессия: используется для бинарной классификации, где yyy — вероятностная функция.

10. Нормализация признаков в задачах регрессии.

Нормализация признаков — процесс приведения признаков к единому масштабу, что улучшает сходимость алгоритмов оптимизации и качество моделей. Основные методы нормализации:

- **Мин-Макс нормализация**: масштабирует данные в диапазоне [0, 1]. Формула: x′=x−xminxmax−xminx' = \frac{x - x_{min}}{x_{max} - x_{min}}x′=xmax−xminx−xmin
- **Z-оценка (стандартизация)**: преобразует данные так, чтобы они имели среднее значение 0 и стандартное отклонение 1. Формула: x′=x−μσx' = \frac{x - \mu}{\sigma}x′=σx−μ

Пример кода на Python:

```
python
Копировать код
from sklearn.preprocessing import MinMaxScaler, StandardScaler

# Пример данных
data = np.array([[1, 2], [2, 3], [3, 4], [4, 5], [5, 6]])

# Мин-Макс нормализация
min_max_scaler = MinMaxScaler()
data_min_max_scaled = min_max_scaler.fit_transform(data)
print(f'Мин-Макс нормализованные данные:\n{data_min_max_scaled}')

# Стандартизация
standard_scaler = StandardScaler()
data_standard_scaled = standard_scaler.fit_transform(data)
print(f'Стандартизованные данные:\n{data_standard_scaled}')
```

11. Задача классификации: постановка, математическая формализация.

Задача классификации заключается в присвоении каждому объекту из множества данных метки класса на основе его признаков. Цель классификации — построить модель, которая может предсказать метку класса для новых, невиданных данных.

Постановка задачи:

- **Входные данные**: X=[x1,x2,…,xn]X = [x_1, x_2, \ldots, x_n]X=[x1,x2,…,xn] — матрица объектов-признаков.
- **Выходные данные**: y=[y1,y2,…,yn]y = [y_1, y_2, \ldots, y_n]y=[y1,y2,…,yn] — вектор меток классов.
- **Цель**: построить функцию fff, которая предсказывает метку класса y^\hat{y}y^ для нового объекта xxx, то есть y^=f(x)\hat{y} = f(x)y^=f(x).

Математическая формализация: y^=f(x)\hat{y} = f(x)y^=f(x) где y^\hat{y}y^ — предсказанная метка класса, fff — функция классификации, обученная на наборе данных.

12. Метод градиентного спуска для задач классификации.

Метод градиентного спуска используется для минимизации функции ошибки в задачах классификации. Например, в логистической регрессии используется кросс-энтропийная ошибка (log loss).

Функция ошибки для логистической регрессии:
J(θ)=−1m∑i=1m[yilog⁡(hθ(xi))+(1−yi)log⁡(1−hθ(xi))]J(\theta) = -\frac{1}{m} \sum_{i=1}^m \left[y_i \log(h_\theta(x_i)) + (1 - y_i) \log(1 - h_\theta(x_i))\right]J(θ)=−m1∑i=1m[yilog(hθ(xi))+(1−yi)log(1−hθ(xi))]
где hθ(x)=11+e−θTxh_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}hθ(x)=1+e−θTx1 — гипотеза логистической регрессии.
Обновление параметров: θ:=θ−α∂J(θ)∂θ\theta := \theta - \alpha \frac{\partial J(\theta)}{\partial \theta}θ:=θ−α∂θ∂J(θ)
Пример кода на Python:

```
python
Копировать код
import numpy as np

def sigmoid(z):
    return 1 / (1 + np.exp(-z))

def compute_cost(theta, X, y):
    m = len(y)
    h = sigmoid(X.dot(theta))
    cost = -1/m * (y.T.dot(np.log(h)) + (1 - y).T.dot(np.log(1 - h)))
    return cost

def gradient_descent(X, y, theta, learning_rate, epochs):
    m = len(y)
    cost_history = []

    for _ in range(epochs):
        gradient = X.T.dot(sigmoid(X.dot(theta)) - y) / m
        theta -= learning_rate * gradient
        cost_history.append(compute_cost(theta, X, y))

    return theta, cost_history

# Пример данных
X = np.array([[1, 1], [1, 2], [2, 2], [2, 3]])
y = np.array([0, 0, 1, 1])
theta = np.zeros(X.shape[1])
learning_rate = 0.1
epochs = 1000

# Добавление столбца для θ0
X = np.c_[np.ones(X.shape[0]), X]

theta, cost_history = gradient_descent(X, y, theta, learning_rate, epochs)
print(f'Параметры модели: {theta}')
```

13. Логистическая регрессия в задачах классификации.

Логистическая регрессия — это метод классификации, который используется для предсказания вероятности принадлежности объекта к одному из двух классов. Она преобразует линейную комбинацию входных признаков в вероятность, используя логистическую функцию.
Модель логистической регрессии: hθ(x)=11+e−θTxh_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}hθ(x)=1+e−θTx1
где hθ(x)h_\theta(x)hθ(x) — вероятность принадлежности класса y=1y = 1y=1.
Функция ошибки: J(θ)=−1m∑i=1m[yilog⁡(hθ(xi))+(1−yi)log⁡(1−hθ(xi))]J(\theta) = -\frac{1}{m} \sum_{i=1}^m \left[y_i \log(h_\theta(x_i)) + (1 - y_i) \log(1 - h_\theta(x_i))\right]J(θ)=−m1∑i=1m[yilog(hθ(xi))+(1−yi)log(1−hθ(xi))]
Пример кода на Python:

```
python
Копировать код
from sklearn.linear_model import LogisticRegression

# Пример данных
X = np.array([[1, 1], [1, 2], [2, 2], [2, 3]])
```

```
y = np.array([0, 0, 1, 1])

# Создание и обучение модели логистической регрессии
model = LogisticRegression()
model.fit(X, y)

# Предсказание
y_pred = model.predict(X)
y_prob = model.predict_proba(X)[:, 1]

print(f'Предсказанные метки: {y_pred}')
print(f'Предсказанные вероятности: {y_prob}')
```

14. Множественная и многоклассовая классификация. Алгоритм “один против всех”.

Множественная классификация — это задача классификации, в которой существует более двух классов.

Многоклассовая классификация — это задача классификации, в которой алгоритм должен присвоить объекту одну метку из нескольких возможных классов.

Алгоритм "один против всех" (One-vs-All):

- Для каждого класса обучается отдельный классификатор, который отличает этот класс от всех остальных.
- Для предсказания нового объекта используются все классификаторы, и выбирается класс с наибольшей вероятностью.

Пример кода на Python:

```
python
Копировать код
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Загрузка набора данных Iris
data = load_iris()
X = data['data']
y = data['target']

# Разделение данных на тренировочный и тестовый наборы
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Создание и обучение модели логистической регрессии с многоклассовой классификацией
model = LogisticRegression(multi_class='ovr')
model.fit(X_train, y_train)

# Предсказание
y_pred = model.predict(X_test)

print(f'Точность модели: {accuracy_score(y_test, y_pred)}')
```

15. Метод опорных векторов в задачах классификации.

Метод опорных векторов (SVM) — это мощный алгоритм машинного обучения для задач классификации. Он находит гиперплоскость, которая максимально разделяет классы в пространстве признаков.

Основные идеи:

- **Гиперплоскость**: поверхность, которая разделяет пространство признаков на две части, каждая из которых соответствует одному классу.
- **Опорные векторы**: точки данных, которые находятся ближе всего к гиперплоскости и определяют ее положение.

Пример кода на Python:

```
python
Копировать код
from sklearn import datasets
```

```
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Загрузка набора данных Iris
iris = datasets.load_iris()
X = iris.data
y = iris.target

# Разделение данных на тренировочный и тестовый наборы
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Создание и обучение модели SVM
model = SVC(kernel='linear')
model.fit(X_train, y_train)

# Предсказание
y_pred = model.predict(X_test)

print(f'Точность модели: {accuracy_score(y_test, y_pred)}')
```

16. Понятие ядра и виды ядер в методе опорных векторов.

Ядро (Kernel) — функция, которая преобразует данные в более высокоразмерное пространство для упрощения задачи классификации. Ядра позволяют использовать SVM для нелинейных задач классификации.

Виды ядер:

- **Линейное ядро**: используется для линейно разделимых данных. K(x,x′)=xTx′K(x, x') = x^T x'K(x,x′)=xTx′
- **Полиномиальное ядро**: позволяет учитывать взаимодействие между признаками. K(x,x′)=(xTx′+c)dK(x, x') = (x^T x' + c)^dK(x,x′)=(xTx′+c)d
- **Радиальное базисное ядро (RBF)**: популярное ядро для нелинейных задач. K(x,x′)=exp⁡(−γ∥x−x′∥2)K(x, x') = \exp\left(-\gamma \|x - x'\|^2\right)K(x,x′)=exp(−γ∥x−x′∥2)
- **Сигмоидное ядро**: используется в нейронных сетях. K(x,x′)=tanh⁡(αxTx′+c)K(x, x') = \tanh(\alpha x^T x' + c)K(x,x′)=tanh(αxTx′+c)

17. Метод решающих деревьев в задачах классификации.

Метод решающих деревьев — алгоритм, который строит дерево решений на основе данных, разделяя пространство признаков на множество прямоугольных областей. В каждой области назначается метка класса.

Основные компоненты:

- **Вершины (nodes)**: точки разделения данных на подмножества.
- **Листья (leaves)**: конечные узлы дерева, в которых назначается метка класса.

Пример кода на Python:

```
python
Копировать код
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Загрузка набора данных Iris
data = load_iris()
X = data['data']
y = data['target']

# Разделение данных на тренировочный и тестовый наборы
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Создание и обучение модели решающего дерева
model = DecisionTreeClassifier()
model.fit(X_train, y_train)
```

```
# Предсказание
y_pred = model.predict(X_test)

print(f'Точность модели: {accuracy_score(y_test, y_pred)}')
```

18. Метод k ближайших соседей в задачах классификации.

Метод k ближайших соседей (k-NN) — это алгоритм классификации, который присваивает метку класса новому объекту на основе меток классов его k ближайших соседей.

Основные шаги:

1. Вычисление расстояния между новым объектом и всеми объектами в тренировочном наборе данных.
2. Выбор k объектов, ближайших к новому объекту.
3. Присвоение метки класса на основе большинства меток среди ближайших соседей.

Пример кода на Python:

```
python
Копировать код
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Загрузка набора данных Iris
data = load_iris()
X = data['data']
y = data['target']

# Разделение данных на тренировочный и тестовый наборы
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Создание и обучение модели k-NN
model = KNeighborsClassifier(n_neighbors=3)
model.fit(X_train, y_train)

# Предсказание
y_pred = model.predict(X_test)

print(f'Точность модели: {accuracy_score(y_test, y_pred)}')
```

19. Однослойный перцептрон в задачах классификации.

Однослойный перцептрон — это простейшая нейронная сеть, используемая для задач бинарной классификации. Он состоит из одного слоя нейронов, каждый из которых соединен со всеми входными признаками.

Основные компоненты:

- **Входной слой**: принимает признаки объекта.
- **Весовые коэффициенты**: используются для взвешивания входных признаков.
- **Активационная функция**: преобразует взвешенную сумму входных сигналов в выходной сигнал.

Обучение:

1. Инициализация весов случайными значениями.
2. Обновление весов на основе ошибки предсказания.

Пример кода на Python:

```
python
Копировать код
import numpy as np

class Perceptron:
    def __init__(self, learning_rate=0.1, epochs=1000):
        self.learning_rate = learning_rate
        self.epochs = epochs

    def fit(self, X, y):
```

```
        self.weights = np.zeros(X.shape[1] + 1)
        for _ in range(self.epochs):
            for xi, target in zip(X, y):
                update = self.learning_rate * (target - self.predict(xi))
                self.weights[1:] += update * xi
                self.weights[0] += update

    def net_input(self, X):
        return np.dot(X, self.weights[1:]) + self.weights[0]

    def predict(self, X):
        return np.where(self.net_input(X) >= 0.0, 1, 0)

# Пример данных
X = np.array([[1, 1], [1, 2], [2, 2], [2, 3]])
y = np.array([0, 0, 1, 1])

# Создание и обучение модели перцептрона
model = Perceptron()
model.fit(X, y)

# Предсказание
y_pred = model.predict(X)
print(f'Предсказанные метки: {y_pred}')
```

20. Метрики эффективности и функции ошибки: назначение, примеры, различия.

Метрики эффективности — это показатели, используемые для оценки качества моделей машинного обучения. Они зависят от типа задачи и могут различаться для классификации и регрессии.

Примеры метрик для классификации:

- **Точность (Accuracy)**: доля правильно классифицированных объектов. Accuracy=TP+TNTP+TN+FP+FN\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}Accuracy=TP+TN+FP+FNTP+TN
- **Полнота (Recall)**: доля правильно классифицированных объектов среди всех объектов положительного класса. Recall=TPTP+FN\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}Recall=TP+FNTP
- **Точность (Precision)**: доля правильно классифицированных объектов среди всех объектов, предсказанных как положительные. Precision=TPTP+FP\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}Precision=TP+FPTP
- **F1-мера**: гармоническое среднее между точностью и полнотой. F1=2⋅Precision⋅RecallPrecision+Recall\text{F1} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}F1=2⋅Precision+RecallPrecision⋅Recall

Функции ошибки — это меры, используемые для минимизации ошибки предсказаний модели во время обучения.

Примеры функций ошибки:

- **Среднеквадратичная ошибка (MSE)**: используется для задач регрессии. J(θ)=1m∑i=1m(y^i−yi)2J(\theta) = \frac{1}{m} \sum_{i=1}^m (\hat{y}_i - y_i)^2J(θ)=m1∑i=1m(y^i−yi)2
- **Кросс-энтропийная ошибка (Log Loss)**: используется для задач классификации. J(θ)=−1m∑i=1m[yilog⁡(hθ(xi))+(1−yi)log⁡(1−hθ(xi))]J(\theta) = -\frac{1}{m} \sum_{i=1}^m \left[y_i \log(h_\theta(x_i)) + (1 - y_i) \log(1 - h_\theta(x_i))\right]J(θ)=−m1∑i=1m[yilog(hθ(xi))+(1−yi)log(1−hθ(xi))]

Различия:

- Метрики эффективности оценивают качество модели на тестовом наборе данных.
- Функции ошибки используются для оптимизации модели во время обучения.

21. Понятие набора данных (датасета) в машинном обучении. Требования, представление. Признаки и объекты.

Набор данных (датасет) — это совокупность данных, используемых для обучения и тестирования модели машинного обучения. Он состоит из объектов и признаков.

Требования к набору данных:

- **Качество данных**: данные должны быть полными, точными и актуальными.
- **Разнообразие данных**: данные должны представлять различные сценарии и условия.
- **Объем данных**: достаточное количество данных для обучения модели.

Представление данных:

- Данные обычно представляются в виде матрицы, где строки — это объекты, а столбцы — признаки.
- Объекты (экземпляры) — это индивидуальные элементы набора данных (например, каждый клиент банка).
- Признаки (фичи) — это атрибуты, описывающие объекты (например, возраст, доход, баланс счета).

Пример набора данных в виде таблицы:

ID	Возраст	Доход	Баланс	Класс
1	25	50000	2000	1
2	35	60000	3000	0
3	45	70000	4000	1
4	55	80000	5000	0

22. Шкалы измерения признаков. Виды шкал, их характеристика.

Признаки в машинном обучении могут быть измерены с использованием различных шкал:

1. **Номинальная шкала**:
   - Категории без порядка.
   - Примеры: цвета (красный, синий), типы животных (собака, кошка).
2. **Порядковая шкала**:
   - Категории с определенным порядком.
   - Примеры: уровни образования (начальное, среднее, высшее), рейтинг (низкий, средний, высокий).
3. **Интервальная шкала**:
   - Категории с равными интервалами между значениями, но без абсолютного нуля.
   - Примеры: температура в градусах Цельсия.
4. **Отношение (Ratio) шкала**:
   - Категории с равными интервалами и абсолютным нулем.
   - Примеры: вес, рост, доход.

23. Понятие чистых данных. Определение, очистка данных.

Чистые данные — это данные, которые не содержат пропусков, ошибок, дубликатов и аномалий. Очистка данных включает процессы удаления или исправления таких проблем для повышения качества данных.

Этапы очистки данных:

1. **Удаление дубликатов**: удаление повторяющихся записей.
2. **Обработка пропущенных данных**: заполнение пропусков средними значениями, медианой или модой, использование методов интерполяции.
3. **Коррекция ошибок**: исправление опечаток и некорректных значений.
4. **Удаление аномалий**: удаление или корректировка необычных значений, которые сильно отличаются от остальных данных.

Пример кода на Python:

```
python
Копировать код
import pandas as pd

# Пример данных с пропусками и дубликатами
data = {'Возраст': [25, 35, 45, 55, None],
        'Доход': [50000, 60000, 70000, 80000, 70000],
        'Баланс': [2000, 3000, 4000, 5000, 4000]}
df = pd.DataFrame(data)

# Удаление дубликатов
```

```
df = df.drop_duplicates()

# Заполнение пропущенных значений
df['Возраст'] = df['Возраст'].fillna(df['Возраст'].mean())

print(df)
```

24. Основные этапы проекта по машинному обучению.

Проект по машинному обучению включает следующие этапы:

1. **Определение задачи**: формулировка задачи, целей и критериев успеха.
2. **Сбор данных**: поиск и сбор данных, необходимых для решения задачи.
3. **Предварительный анализ данных**: исследование данных, понимание их структуры и выявление закономерностей.
4. **Очистка данных**: устранение пропусков, дубликатов и ошибок.
5. **Преобразование данных**: нормализация, стандартизация, создание новых признаков.
6. **Разделение данных**: разделение данных на тренировочную и тестовую выборки.
7. **Выбор модели**: выбор алгоритма машинного обучения.
8. **Обучение модели**: обучение модели на тренировочных данных.
9. **Оценка модели**: оценка качества модели на тестовых данных.
10. **Тюнинг модели**: оптимизация гиперпараметров модели.
11. **Внедрение модели**: интеграция модели в производственную среду.
12. **Мониторинг и обновление модели**: регулярное обновление и переобучение модели на новых данных.

25. Предварительный анализ данных: задачи, методы, цели.

Цели предварительного анализа данных:

- Понимание структуры и распределения данных.
- Выявление аномалий и пропусков.
- Оценка взаимосвязей между признаками и целевой переменной.

Задачи:

- Исследование распределения данных.
- Поиск и устранение выбросов.
- Анализ корреляций между признаками.

Методы:

- **Описание данных**: использование методов статистического анализа (среднее, медиана, стандартное отклонение).
- **Визуализация данных**: построение графиков и диаграмм (гистограммы, диаграммы рассеяния, коробчатые диаграммы).
- **Корреляционный анализ**: вычисление коэффициентов корреляции между признаками.

Пример кода на Python:

```
python
Копировать код
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Пример данных
data = {'Возраст': [25, 35, 45, 55, 65],
        'Доход': [50000, 60000, 70000, 80000, 90000],
        'Баланс': [2000, 3000, 4000, 5000, 6000]}
df = pd.DataFrame(data)

# Описание данных
print(df.describe())

# Корреляционный анализ
correlation_matrix = df.corr()
print(correlation_matrix)

# Визуализация данных
sns.pairplot(df)
```

```
plt.show()
```

26. Проблема отсутствующих данных: причины, исследование, пути решения.

Причины отсутствующих данных:

- Ошибки при сборе данных.
- Неполные анкеты или опросы.
- Технические проблемы.

Исследование пропусков:

- Анализ пропусков с использованием методов описательной статистики.
- Визуализация пропусков с помощью графиков.

Пути решения:

1. **Удаление**: удаление строк или столбцов с пропусками, если их немного.
2. **Замена**: заполнение пропусков средними значениями, медианой, модой или использованием методов интерполяции.
3. **Моделирование**: предсказание пропущенных значений с помощью моделей машинного обучения.

Пример кода на Python:

```
python
Копировать код
import pandas as pd

# Пример данных с пропусками
data = {'Возраст': [25, None, 45, 55, None],
        'Доход': [50000, 60000, 70000, None, 90000],
        'Баланс': [2000, 3000, 4000, 5000, None]}
df = pd.DataFrame(data)

# Анализ пропусков
print(df.isnull().sum())

# Заполнение пропусков
df['Возраст'] = df['Возраст'].fillna(df['Возраст'].mean())
df['Доход'] = df['Доход'].fillna(df['Доход'].median())
df['Баланс'] = df['Баланс'].fillna(0)

print(df)
```

27. Проблема несбалансированных классов: исследование, пути решения.

Проблема несбалансированных классов возникает, когда один класс сильно преобладает над другими в наборе данных. Это может привести к плохой производительности модели для меньших классов.

Исследование несбалансированности:

- Анализ распределения классов в данных.
- Оценка метрик качества для каждого класса (точность, полнота, F1-мера).

Пути решения:

1. **Ресэмплинг**:
   - **Оверсэмплинг**: увеличение количества примеров меньшего класса.
   - **Андерсэмплинг**: уменьшение количества примеров большего класса.
2. **Создание синтетических данных**:
   - **SMOTE (Synthetic Minority Over-sampling Technique)**: генерация новых примеров меньшего класса.
3. **Использование алгоритмов, устойчивых к несбалансированности**: использование методов, таких как взвешенные SVM или случайные леса.

Пример кода на Python:

```
python
Копировать код
from sklearn.datasets import make_classification
from imblearn.over_sampling import SMOTE
from collections import Counter

# Создание несбалансированного набора данных
```

```
X, y = make_classification(n_classes=2, class_sep=2, weights=[0.1, 0.9],
n_informative=3, n_redundant=1, flip_y=0, n_features=20, n_clusters_per_class=1,
n_samples=1000, random_state=10)
print(f'Распределение классов до SMOTE: {Counter(y)}')

# Применение SMOTE
smote = SMOTE()
X_res, y_res = smote.fit_resample(X, y)
print(f'Распределение классов после SMOTE: {Counter(y_res)}')
```

28. Понятие параметров и гиперпараметров модели. Обучение параметров и гиперпараметров. Поиск по сетке.

Параметры модели — это внутренние параметры, которые обучаются во время тренировки модели. Примеры: коэффициенты в линейной регрессии, веса в нейронной сети.

Гиперпараметры модели — это параметры, которые устанавливаются до начала тренировки и не изменяются в процессе обучения. Примеры: скорость обучения, количество слоев в нейронной сети.

Обучение параметров:

- Процесс настройки параметров модели с целью минимизации функции ошибки на тренировочных данных.

Обучение гиперпараметров:

- Подбор гиперпараметров с использованием методов перекрестной проверки и поиска по сетке (Grid Search).

Поиск по сетке (Grid Search):

- Метод систематического перебора различных комбинаций гиперпараметров для выбора оптимальной модели.

Пример кода на Python:

```
python
Копировать код
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

# Пример данных
X = [[1, 2], [3, 4], [5, 6], [7, 8]]
y = [0, 0, 1, 1]

# Определение модели
model = RandomForestClassifier()

# Определение гиперпараметров для поиска по сетке
param_grid = {
    'n_estimators': [50, 100, 150],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10]
}

# Поиск по сетке
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5)
grid_search.fit(X, y)

print(f'Лучшие гиперпараметры: {grid_search.best_params_}')
```

29. Понятие недо- и переобучения. Определение, пути решения.

Недообучение (Underfitting) — это ситуация, когда модель плохо обучается на тренировочных данных и не способна уловить закономерности в данных. Модель недообучена, если она имеет высокую ошибку как на тренировочных, так и на тестовых данных.

Переобучение (Overfitting) — это ситуация, когда модель слишком хорошо обучается на тренировочных данных, включая шум и выбросы, и теряет способность обобщать информацию на новых данных. Модель переобучена, если она имеет низкую ошибку на тренировочных данных и высокую ошибку на тестовых данных.

Пути решения недообучения:

- Увеличение сложности модели (добавление признаков, использование более сложных моделей).
- Увеличение времени обучения.
- Сбор дополнительных данных.

Пути решения переобучения:
- Уменьшение сложности модели (удаление ненужных признаков, использование регуляризации).
- Использование большего количества данных для обучения.
- Применение методов регуляризации (L1, L2 регуляризация).
- Применение методов уменьшения выборки (Cross-validation).

Пример кода на Python:
```
python
Копировать код
from sklearn.linear_model import Ridge
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# Пример данных
X = [[i] for i in range(10)]
y = [2*i + 1 for i in range(10)]

# Разделение данных на тренировочный и тестовый наборы
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Обучение модели с L2 регуляризацией (Ridge)
model = Ridge(alpha=1.0)
model.fit(X_train, y_train)

# Оценка модели
y_pred_train = model.predict(X_train)
y_pred_test = model.predict(X_test)

print(f'Training MSE: {mean_squared_error(y_train, y_pred_train)}')
print(f'Test MSE: {mean_squared_error(y_test, y_pred_test)}')
```

30. Диагностика модели машинного обучения. Методы, цели.

Цели диагностики модели:
- Оценка качества модели.
- Выявление проблем недообучения и переобучения.
- Определение необходимости улучшений модели.

Методы диагностики:
1. **Оценка на тренировочных и тестовых данных**: сравнение производительности модели на разных наборах данных.
2. **Кросс-валидация**: использование методов перекрестной проверки для более точной оценки производительности модели.
3. **Анализ ошибок**: исследование ошибок модели для выявления систематических проблем.
4. **Визуализация результатов**: построение графиков и диаграмм для анализа поведения модели.

Пример кода на Python:
```
python
Копировать код
from sklearn.datasets import load_iris
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier

# Загрузка набора данных Iris
data = load_iris()
X = data['data']
y = data['target']

# Создание и обучение модели решающего дерева
```

```
model = DecisionTreeClassifier()

# Кросс-валидация
scores = cross_val_score(model, X, y, cv=5)

print(f'Оценки кросс-валидации: {scores}')
print(f'Средняя оценка: {scores.mean()}')
```

31. Проблема выбора модели машинного обучения. Сравнение моделей.

Проблема выбора модели машинного обучения возникает из-за разнообразия доступных моделей и их параметров, которые могут по-разному влиять на производительность для различных наборов данных и задач.

Основные критерии выбора модели:

- **Производительность**: точность, полнота, F1-мера и другие метрики.
- **Скорость обучения и предсказания**: время, необходимое для обучения модели и предсказания новых данных.
- **Сложность модели**: количество параметров и архитектура модели.
- **Интерпретируемость**: насколько легко понять и объяснить результаты модели.

Сравнение моделей:

- **Тестирование на одном и том же наборе данных**: обучение и оценка различных моделей на одинаковом тренировочном и тестовом наборе данных.
- **Использование перекрестной проверки (кросс-валидации)**: для получения более надежной оценки производительности моделей.
- **Графики производительности**: использование графиков ROC-кривой, Precision-Recall кривой и других визуализаций для сравнения моделей.

Пример кода на Python:

```
python
Копировать код
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC

# Загрузка набора данных Iris
data = load_iris()
X = data['data']
y = data['target']

# Разделение данных на тренировочный и тестовый наборы
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Модели для сравнения
models = {
    "Decision Tree": DecisionTreeClassifier(),
    "Random Forest": RandomForestClassifier(),
    "SVM": SVC()
}

# Оценка моделей с использованием кросс-валидации
for name, model in models.items():
    scores = cross_val_score(model, X_train, y_train, cv=5)
    print(f'{name}: {scores.mean()}')
```

32. Измерение эффективности работы моделей машинного обучения. Метрики эффективности.

Измерение эффективности работы моделей машинного обучения — это процесс оценки качества моделей на основе различных метрик. Метрики эффективности помогают понять, насколько хорошо модель справляется с поставленной задачей.

Основные метрики эффективности:

- **Для классификации**: точность, полнота, точность, F1-мера, ROC-AUC, логарифмическая потеря (log loss).
- **Для регрессии**: среднеквадратичная ошибка (MSE), средняя абсолютная ошибка (MAE), коэффициент детерминации (R^2).

33. Метрики эффективности моделей классификации. Виды, характеристика, выбор.

Основные метрики для классификации:

- **Точность (Accuracy)**: доля правильно классифицированных объектов. Accuracy=TP+TNTP+TN+FP+FN\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}Accuracy=TP+TN+FP+FNTP+TN
- **Полнота (Recall)**: доля правильно классифицированных объектов среди всех объектов положительного класса. Recall=TPTP+FN\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}Recall=TP+FNTP
- **Точность (Precision)**: доля правильно классифицированных объектов среди всех объектов, предсказанных как положительные. Precision=TPTP+FP\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}Precision=TP+FPTP
- **F1-мера**: гармоническое среднее между точностью и полнотой. F1=2⋅Precision⋅RecallPrecision+Recall\text{F1} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}F1=2⋅Precision+RecallPrecision⋅Recall
- **ROC-AUC**: площадь под кривой ROC, характеризующая качество бинарного классификатора.
- **Логарифмическая потеря (Log Loss)**: мера несоответствия предсказанных вероятностей истинным меткам классов.

Пример кода на Python:

```
python
Копировать код
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
roc_auc_score, log_loss
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

# Пример данных
X = [[i] for i in range(10)]
y = [0, 1, 0, 1, 0, 1, 0, 1, 0, 1]

# Разделение данных на тренировочный и тестовый наборы
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Обучение модели логистической регрессии
model = LogisticRegression()
model.fit(X_train, y_train)

# Предсказание
y_pred = model.predict(X_test)
y_prob = model.predict_proba(X_test)[:, 1]

# Метрики
print(f'Accuracy: {accuracy_score(y_test, y_pred)}')
print(f'Precision: {precision_score(y_test, y_pred)}')
print(f'Recall: {recall_score(y_test, y_pred)}')
print(f'F1 Score: {f1_score(y_test, y_pred)}')
print(f'ROC AUC: {roc_auc_score(y_test, y_prob)}')
print(f'Log Loss: {log_loss(y_test, y_prob)}')
```

34. Метрики эффективности моделей регрессии. Виды, характеристика, выбор.

Основные метрики для регрессии:

- **Среднеквадратичная ошибка (MSE)**: среднее значение квадратов ошибок. MSE=1n∑i=1n(yi−y^i)2\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2MSE=n1 ∑i=1n(yi−y^i)2
- **Средняя абсолютная ошибка (MAE)**: среднее значение абсолютных ошибок. MAE=1n∑i=1n|yi−y^i|\text{MAE} = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|MAE=n1∑i=1n |yi−y^i|
- **Коэффициент детерминации (R^2)**: доля объясненной дисперсии. R2=1−∑i=1n(yi−y^i)2∑i=1n(yi−y¯)2R^2 = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2}R2=1−∑i=1n(yi−y¯)2∑i=1n(yi−y^i)2

Пример кода на Python:

```
python
Копировать код
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

# Пример данных
X = [[i] for i in range(10)]
y = [2*i + 1 for i in range(10)]

# Разделение данных на тренировочный и тестовый наборы
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Обучение модели линейной регрессии
model = LinearRegression()
model.fit(X_train, y_train)

# Предсказание
y_pred = model.predict(X_test)

# Метрики
print(f'MSE: {mean_squared_error(y_test, y_pred)}')
print(f'MAE: {mean_absolute_error(y_test, y_pred)}')
print(f'R^2: {r2_score(y_test, y_pred)}')
```

35. Перекрестная проверка (кросс-валидация). Назначение, схема работы.

Перекрестная проверка (кросс-валидация) — это метод оценки качества модели, который использует повторное разбиение данных на тренировочные и тестовые наборы для получения более надежной оценки производительности модели.

Схема работы:

1. Разделение данных на kkk подмножеств (фолдов).
2. Обучение модели kkk раз, каждый раз используя k−1k-1k−1 подмножество для тренировки и 1 подмножество для тестирования.
3. Среднее значение метрик качества по всем kkk моделям.

Пример кода на Python:

```
python
Копировать код
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import load_iris

# Загрузка набора данных Iris
data = load_iris()
X = data['data']
y = data['target']

# Создание модели
model = RandomForestClassifier()

# Кросс-валидация
scores = cross_val_score(model, X, y, cv=5)
```

```
print(f'Оценки кросс-валидации: {scores}')
print(f'Средняя оценка: {scores.mean()}')
```

36. Конвейеры в библиотеке sklearn. Назначение, использование.

Конвейеры (Pipeline) в sklearn позволяют объединить несколько этапов обработки данных и моделирования в единый процесс, упрощая код и минимизируя вероятность ошибок.

Назначение:

- Автоматизация последовательных этапов машинного обучения.
- Обеспечение согласованности в обработке данных.
- Упрощение кода и улучшение воспроизводимости.

Пример кода на Python:

```
python
Копировать код
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split, cross_val_score

# Загрузка набора данных Iris
data = load_iris()
X = data['data']
y = data['target']

# Разделение данных на тренировочный и тестовый наборы
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Создание конвейера
pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('classifier', RandomForestClassifier())
])

# Обучение и оценка модели с использованием кросс-валидации
scores = cross_val_score(pipeline, X_train, y_train, cv=5)

print(f'Оценки кросс-валидации: {scores}')
print(f'Средняя оценка: {scores.mean()}')
```

37. Использование методов визуализации данных для предварительного анализа.

Методы визуализации данных используются для понимания структуры данных, выявления закономерностей и аномалий, а также для представления результатов анализа и моделей.

Цели:

- Понимание распределения данных.
- Выявление выбросов и пропусков.
- Оценка взаимосвязей между признаками.

Методы:

- **Гистограммы**: показывают распределение одного признака.
- **Диаграммы рассеяния (scatter plots)**: показывают взаимосвязь между двумя признаками.
- **Коробчатые диаграммы (box plots)**: показывают распределение данных и выбросы.
- **Матрицы корреляции**: показывают корреляцию между признаками.

Пример кода на Python:

```
python
Копировать код
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris

# Загрузка набора данных Iris
data = load_iris()
df = sns.load_dataset('iris')
```

```
# Гистограмма
sns.histplot(df['sepal_length'], kde=True)
plt.show()

# Диаграмма рассеяния
sns.scatterplot(x='sepal_length', y='sepal_width', hue='species', data=df)
plt.show()

# Коробчатая диаграмма
sns.boxplot(x='species', y='sepal_length', data=df)
plt.show()

# Матрица корреляции
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.show()
```

38. Исследование коррелированности признаков: методы, цели, выводы.

Исследование коррелированности признаков помогает понять взаимосвязи между различными признаками, что важно для выбора признаков, устранения избыточности и улучшения производительности модели.

Цели:

- Определение зависимости между признаками.
- Выявление избыточных признаков.
- Улучшение интерпретируемости модели.

Методы:

- **Коэффициент корреляции Пирсона**: измеряет линейную зависимость между признаками.
- **Коэффициент корреляции Спирмена**: измеряет монотонную зависимость между признаками.
- **Матрицы корреляции**: визуализация корреляций между множеством признаков.

Пример кода на Python:

```
python
Копировать код
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Пример данных
data = {'Возраст': [25, 35, 45, 55, 65],
        'Доход': [50000, 60000, 70000, 80000, 90000],
        'Баланс': [2000, 3000, 4000, 5000, 6000]}
df = pd.DataFrame(data)

# Матрица корреляции
correlation_matrix = df.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.show()
```

39. Решкалирование данных. Виды, назначение, применение. Нормализация и стандартизация данных.

Решкалирование данных — процесс преобразования признаков данных в определенный диапазон для улучшения производительности модели и ускорения сходимости алгоритмов.

Виды решкалирования:

- **Нормализация (Min-Max Scaling)**: преобразование данных в диапазон [0, 1]. x′=x−xminxmax−xminx' = \frac{x - x_{min}}{x_{max} - x_{min}}x′=xmax−xminx−xmin
- **Стандартизация (Standardization)**: преобразование данных с нулевым средним и единичным стандартным отклонением. x′=x−μσx' = \frac{x - \mu}{\sigma}x′=σx−μ

Назначение:

- Уменьшение влияния масштаба признаков на модель.
- Ускорение сходимости градиентных методов.
- Улучшение стабильности и производительности модели.

Пример кода на Python:

```
python
```

```
Копировать код
from sklearn.preprocessing import MinMaxScaler, StandardScaler

# Пример данных
data = [[-1, 2], [-0.5, 6], [0, 10], [1, 18]]
scaler_minmax = MinMaxScaler()
scaler_standard = StandardScaler()

# Нормализация данных
data_minmax = scaler_minmax.fit_transform(data)
print(f'Нормализованные данные:\n{data_minmax}')

# Стандартизация данных
data_standard = scaler_standard.fit_transform(data)
print(f'Стандартизованные данные:\n{data_standard}')
```

40. Преобразование категориальных признаков в числовые.

Преобразование категориальных признаков в числовые необходимо, чтобы алгоритмы машинного обучения могли работать с этими признаками.

Методы:

- **One-Hot Encoding**: преобразование категориальных признаков в набор бинарных признаков.
- **Label Encoding**: преобразование категориальных признаков в числовые метки.
- **Target Encoding**: замена категориальных значений на средние значения целевой переменной для каждой категории.

Пример кода на Python:

```
python
Копировать код
import pandas as pd
from sklearn.preprocessing import OneHotEncoder, LabelEncoder

# Пример данных
data = {'Цвет': ['красный', 'синий', 'зеленый', 'красный']}
df = pd.DataFrame(data)

# One-Hot Encoding
encoder = OneHotEncoder(sparse=False)
one_hot = encoder.fit_transform(df[['Цвет']])
print(f'One-Hot Encoding:\n{one_hot}')

# Label Encoding
label_encoder = LabelEncoder()
labels = label_encoder.fit_transform(df['Цвет'])
print(f'Label Encoding:\n{labels}')
```

41. Методы визуализации данных для машинного обучения

Методы визуализации данных помогают анализировать и интерпретировать данные, выявлять закономерности, аномалии и зависимости между признаками. Основные методы включают:

- **Гистограммы (Histograms)**: показывают распределение значений одного признака.
- **Диаграммы рассеяния (Scatter Plots)**: показывают взаимосвязь между двумя признаками.
- **Коробчатые диаграммы (Box Plots)**: показывают распределение данных и выбросы.
- **Матрицы корреляции (Correlation Matrices)**: показывают корреляцию между признаками.
- **Диаграммы Парето (Pareto Charts)**: показывают вклад различных категорий в общее значение.
- **Диаграммы с тепловыми картами (Heatmaps)**: визуализируют корреляции и другие метрики.
- **Диаграммы распределения (Distribution Plots)**: показывают плотность распределения признаков.
- **Графики ROC-кривой (ROC Curve)** и Precision-Recall кривой для оценки качества моделей классификации.

Пример кода на Python:

```
python
Копировать код
import seaborn as sns
import matplotlib.pyplot as plt
```

```
from sklearn.datasets import load_iris

# Загрузка набора данных Iris
data = load_iris()
df = sns.load_dataset('iris')

# Гистограмма
sns.histplot(df['sepal_length'], kde=True)
plt.show()

# Диаграмма рассеяния
sns.scatterplot(x='sepal_length', y='sepal_width', hue='species', data=df)
plt.show()

# Коробчатая диаграмма
sns.boxplot(x='species', y='sepal_length', data=df)
plt.show()

# Матрица корреляции
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.show()
```

42. Задача выбора модели. Оценка эффективности, валидационный набор

Задача выбора модели включает в себя выбор наилучшей модели машинного обучения для конкретной задачи на основе данных и требований. Основные этапы включают:

1. **Определение критериев оценки**: выбор метрик, которые будут использоваться для оценки моделей.
2. **Разделение данных**: на тренировочный, валидационный и тестовый наборы.
3. **Обучение моделей**: на тренировочном наборе данных.
4. **Оценка моделей**: на валидационном наборе с использованием выбранных метрик.
5. **Выбор модели**: на основе производительности на валидационном наборе.
6. **Проверка на тестовом наборе**: окончательная оценка производительности.

Пример кода на Python:

```
python
Копировать код
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import load_iris
from sklearn.metrics import accuracy_score

# Загрузка набора данных Iris
data = load_iris()
X = data['data']
y = data['target']

# Разделение данных на тренировочный, валидационный и тестовый наборы
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.4,
random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5,
random_state=42)

# Обучение модели
model = RandomForestClassifier()
model.fit(X_train, y_train)

# Оценка модели на валидационном наборе
y_val_pred = model.predict(X_val)
val_score = accuracy_score(y_val, y_val_pred)
print(f'Validation Accuracy: {val_score}')

# Окончательная оценка на тестовом наборе
y_test_pred = model.predict(X_test)
test_score = accuracy_score(y_test, y_test_pred)
print(f'Test Accuracy: {test_score}')
```

43. Кривые обучения для диагностики моделей машинного обучения

Кривые обучения (Learning Curves) помогают диагностировать проблемы недообучения и переобучения, показывая зависимость ошибки модели от количества обучающих данных.

Основные компоненты:

- **Ошибка на тренировочных данных**: показывает, как модель обучается на известном наборе данных.
- **Ошибка на валидационных данных**: показывает, как модель обобщает на новых данных.

Пример кода на Python:

```
python
Копировать код
import matplotlib.pyplot as plt
from sklearn.model_selection import learning_curve
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import load_iris

# Загрузка набора данных Iris
data = load_iris()
X = data['data']
y = data['target']

# Создание модели
model = RandomForestClassifier()

# Вычисление кривых обучения
train_sizes, train_scores, val_scores = learning_curve(model, X, y, cv=5,
scoring='accuracy')

# Средние и стандартные отклонения ошибок
train_scores_mean = train_scores.mean(axis=1)
train_scores_std = train_scores.std(axis=1)
val_scores_mean = val_scores.mean(axis=1)
val_scores_std = val_scores.std(axis=1)

# Построение кривых обучения
plt.figure()
plt.fill_between(train_sizes, train_scores_mean - train_scores_std, train_scores_mean
+ train_scores_std, alpha=0.1, color="r")
plt.fill_between(train_sizes, val_scores_mean - val_scores_std, val_scores_mean +
val_scores_std, alpha=0.1, color="g")
plt.plot(train_sizes, train_scores_mean, 'o-', color="r", label="Training score")
plt.plot(train_sizes, val_scores_mean, 'o-', color="g", label="Cross-validation
score")
plt.xlabel("Training examples")
plt.ylabel("Score")
plt.legend(loc="best")
plt.show()
```

44. Регуляризация моделей машинного обучения. Назначение, виды, формализация

Регуляризация — это метод предотвращения переобучения моделей путем добавления штрафа за сложность модели.

Назначение:

- Снижение переобучения.
- Улучшение обобщающей способности модели.

Виды регуляризации:

- **L1-регуляризация (Lasso)**: добавление суммы абсолютных значений коэффициентов в функцию потерь. Loss=MSE+λ∑j=1n|βj|\text{Loss} = \text{MSE} + \lambda \sum_{j=1}^n |\beta_j|Loss=MSE+λ∑j=1n|βj|
- **L2-регуляризация (Ridge)**: добавление суммы квадратов коэффициентов в функцию потерь. Loss=MSE+λ∑j=1nβj2\text{Loss} = \text{MSE} + \lambda \sum_{j=1}^n \beta_j^2Loss=MSE+λ∑j=1nβj2
- **Elastic Net**: комбинирование L1 и L2 регуляризаций. Loss=MSE+λ1∑j=1n|βj|+λ2∑j=1nβj2\text{Loss} = \text{MSE} + \lambda_1 \sum_{j=1}^n |\beta_j| + \lambda_2 \sum_{j=1}^n \beta_j^2Loss=MSE+λ1∑j=1n|βj|+λ2∑j=1nβj2

Пример кода на Python:

```
python
Копировать код
from sklearn.linear_model import Ridge, Lasso
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# Пример данных
X = [[i] for i in range(10)]
y = [2*i + 1 for i in range(10)]

# Разделение данных на тренировочный и тестовый наборы
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Обучение модели Ridge-регрессии
ridge_model = Ridge(alpha=1.0)
ridge_model.fit(X_train, y_train)
ridge_pred = ridge_model.predict(X_test)

# Обучение модели Lasso-регрессии
lasso_model = Lasso(alpha=0.1)
lasso_model.fit(X_train, y_train)
lasso_pred = lasso_model.predict(X_test)

print(f'Ridge MSE: {mean_squared_error(y_test, ridge_pred)}')
print(f'Lasso MSE: {mean_squared_error(y_test, lasso_pred)}')
```

45. Проблема сбора и интеграции данных для машинного обучения

Проблемы сбора и интеграции данных включают в себя:

- **Качество данных**: данные могут содержать ошибки, пропуски, дубликаты.
- **Разнообразие источников данных**: данные могут поступать из разных источников с различными форматами и стандартами.
- **Объем данных**: обработка и хранение больших объемов данных требует значительных ресурсов.
- **Конфиденциальность и безопасность данных**: соблюдение требований безопасности и конфиденциальности при сборе и использовании данных.

Пути решения:

- Использование ETL-процессов (Extract, Transform, Load) для очистки и преобразования данных.
- Применение методов распределенной обработки данных для работы с большими объемами данных.
- Соблюдение стандартов и протоколов безопасности для защиты данных.

46. Понятие чистых данных и требования к данным

Чистые данные — это данные, которые не содержат ошибок, пропусков, дубликатов и аномалий.

Требования к чистым данным:

- **Полнота**: отсутствие пропущенных значений.
- **Точность**: корректность значений данных.
- **Актуальность**: данные должны быть актуальными и обновленными.
- **Однородность**: данные должны быть в едином формате.
- **Консистентность**: данные должны быть логически согласованными.

47. Основные задачи описательного анализа данных

Описательный анализ данных — это начальный этап анализа данных, включающий суммирование и описание характеристик набора данных.

Основные задачи:

- **Описание распределения данных**: использование статистических методов (среднее, медиана, стандартное отклонение).
- **Выявление выбросов**: обнаружение и анализ необычных значений.
- **Анализ корреляций**: исследование взаимосвязей между признаками.
- **Визуализация данных**: построение графиков и диаграмм для понимания структуры данных.

48. Полиномиальные модели машинного обучения

Полиномиальные модели позволяют учитывать нелинейные зависимости между признаками и целевой переменной.
Пример полиномиальной регрессии: y=β0+β1x+β2x2+…+βnxny = \beta_0 + \beta_1 x + \beta_2 x^2 + \ldots + \beta_n x^ny=β0+β1x+β2x2+…+βnxn
Пример кода на Python:

```
python
Копировать код
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# Пример данных
X = [[i] for i in range(10)]
y = [2*i**2 + 3*i + 1 for i in range(10)]

# Разделение данных на тренировочный и тестовый наборы
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Создание полиномиальной модели
poly_model = Pipeline([
    ('poly_features', PolynomialFeatures(degree=2)),
    ('linear_regression', LinearRegression())
])

# Обучение модели
poly_model.fit(X_train, y_train)

# Предсказание
y_pred = poly_model.predict(X_test)

print(f'MSE: {mean_squared_error(y_test, y_pred)}')
```

49. Основные виды преобразования данных для подготовки к машинному обучению

Преобразование данных включает различные методы подготовки данных для использования в моделях машинного обучения.
Основные виды преобразования:

- **Нормализация**: приведение признаков к диапазону [0, 1].
- **Стандартизация**: приведение признаков к нулевому среднему и единичному стандартному отклонению.
- **One-Hot Encoding**: преобразование категориальных признаков в набор бинарных признаков.
- **Label Encoding**: преобразование категориальных признаков в числовые метки.
- **PCA (Principal Component Analysis)**: понижение размерности данных.
- **Удаление пропусков**: замена или удаление пропущенных значений.

Пример кода на Python:

```
python
Копировать код
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.decomposition import PCA
import pandas as pd

# Пример данных
data = {'Цвет': ['красный', 'синий', 'зеленый', 'красный'], 'Значение': [1, 2, 3, 4]}
df = pd.DataFrame(data)

# Стандартизация данных
scaler = StandardScaler()
df['Значение'] = scaler.fit_transform(df[['Значение']])

# One-Hot Encoding
```

```
encoder = OneHotEncoder(sparse=False)
encoded_colors = encoder.fit_transform(df[['Цвет']])
encoded_df = pd.DataFrame(encoded_colors,
columns=encoder.get_feature_names_out(['Цвет']))

# Объединение преобразованных данных
final_df = pd.concat([df, encoded_df], axis=1).drop(columns=['Цвет'])
print(final_df)
```

50. Задача выбора признаков в машинном обучении

Задача выбора признаков заключается в отборе наиболее значимых признаков для построения модели машинного обучения. Это помогает улучшить производительность модели и уменьшить ее сложность.

Методы выбора признаков:

- **Методы фильтрации (Filter Methods)**: основаны на статистических свойствах данных (корреляция, ANOVA).
- **Методы оболочки (Wrapper Methods)**: используют модель для оценки значимости признаков (RFE, последовательный отбор).
- **Методы встроенного отбора (Embedded Methods)**: выбор признаков встроен в процесс обучения модели (Lasso, Decision Tree).

Пример кода на Python:

```
python
Копировать код
from sklearn.datasets import load_iris
from sklearn.feature_selection import SelectKBest, chi2
import pandas as pd

# Загрузка набора данных Iris
data = load_iris()
X = data['data']
y = data['target']

# Выбор лучших признаков с помощью метода chi2
selector = SelectKBest(chi2, k=2)
X_new = selector.fit_transform(X, y)

print(f'Выбранные признаки:\n{X_new}')
```