Методы очистки и подготовки данных для алгоритмов машинного обучения

Почему подготовка данных важна для машинного обучения?

Искусственный интеллект и машинное обучение опираются на данные. Данные это основа всех алгоритмов. Но данные, с которыми мы сталкиваемся в реальном мире, зачастую далеко не идеальны: они могут быть неполными, содержать ошибки, шум или быть представлены в неудобных для анализа форматах.

Почему подготовка данных важна для машинного обучения?

1. Качество данных определяет точность модели: Данные с ошибками или пропусками приводят к ошибочным прогнозам.

Пример: В наборе данных о клиентах магазина отсутствует информация о возрасте. Если модель обучается на таких данных, она может неправильно сегментировать клиентов.

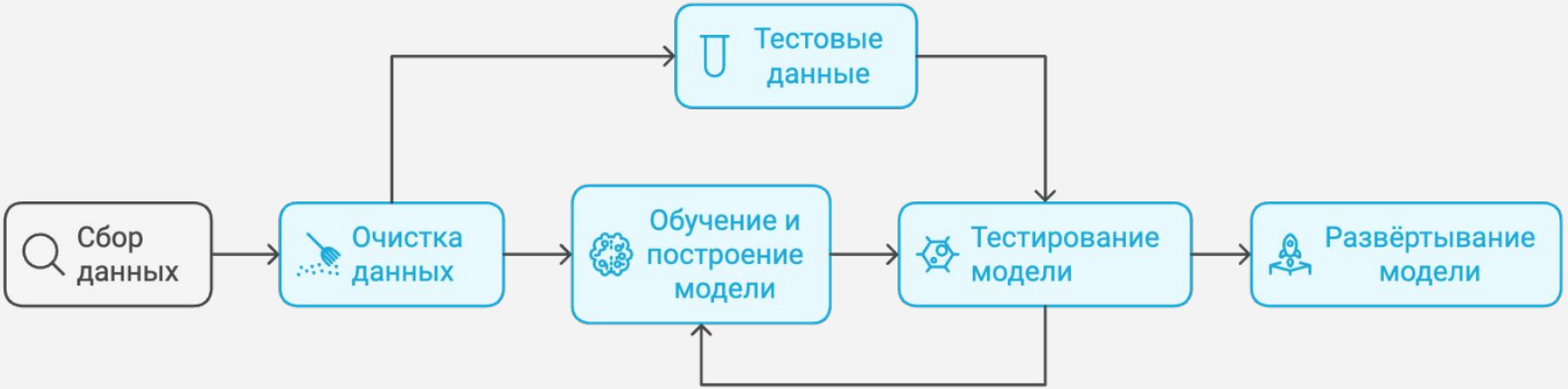
Почему подготовка данных важна для машинного обучения?

2. Единообразие данных: Разные форматы данных делают их анализ сложным. Например, если в одной колонке даты записаны как "01/01/2022", а в другой как "2022-01-01", алгоритмы не смогут работать с ними.

3. Выбросы и шум

Пример: В наборе данных о доходах есть запись "10 000 000" вместо реального дохода "100 000". Такой выброс может исказить результаты модели.

Схема процесса машинного обучения



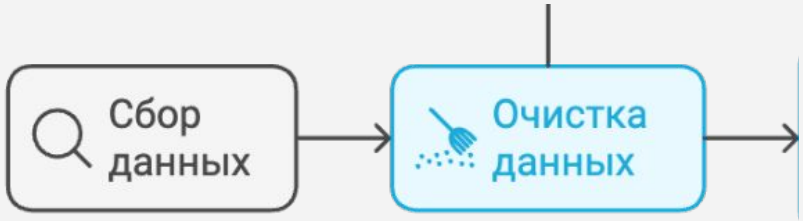
Сбор данных

Это первый этап, где мы собираем данные, которые будут использоваться для обучения модели.

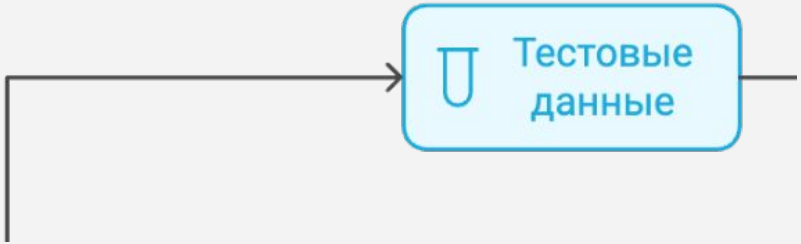
Очистка данных

Этот этап включает обработку данных, чтобы сделать их пригодными для машинного обучения.

Очистка и форматирование данных (с использованием Pandas)



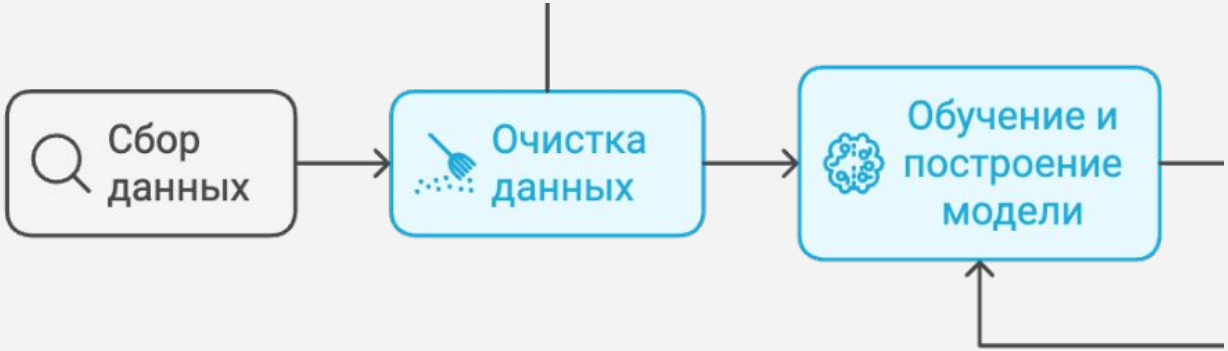
Тестовые данные



Это подмножество данных, которое не используется для обучения модели, а применяется для её проверки.

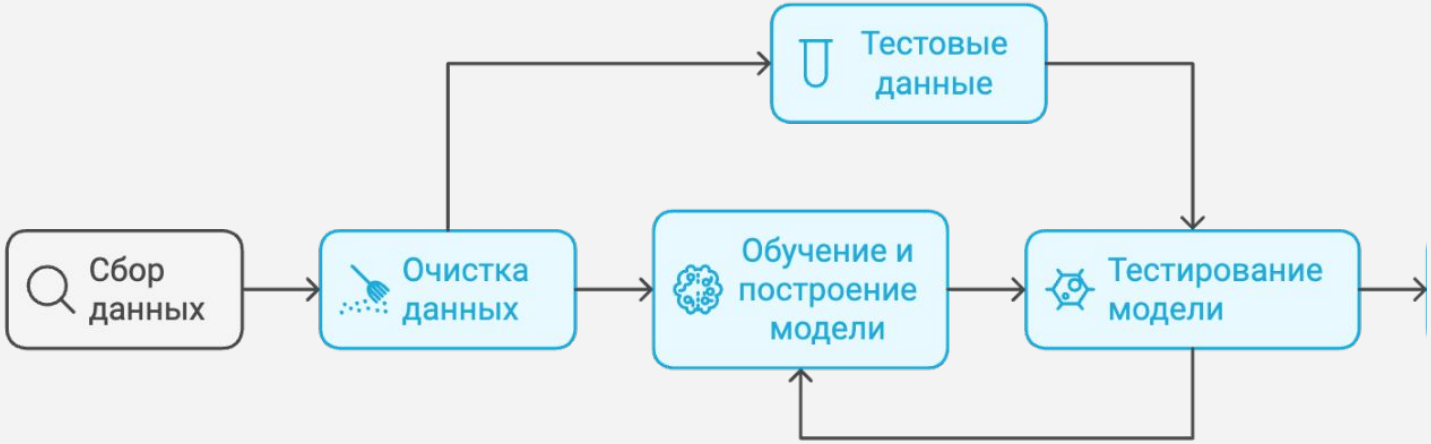
Обучение и построение модели

На этом этапе алгоритм анализирует обучающие данные и учится находить закономерности.



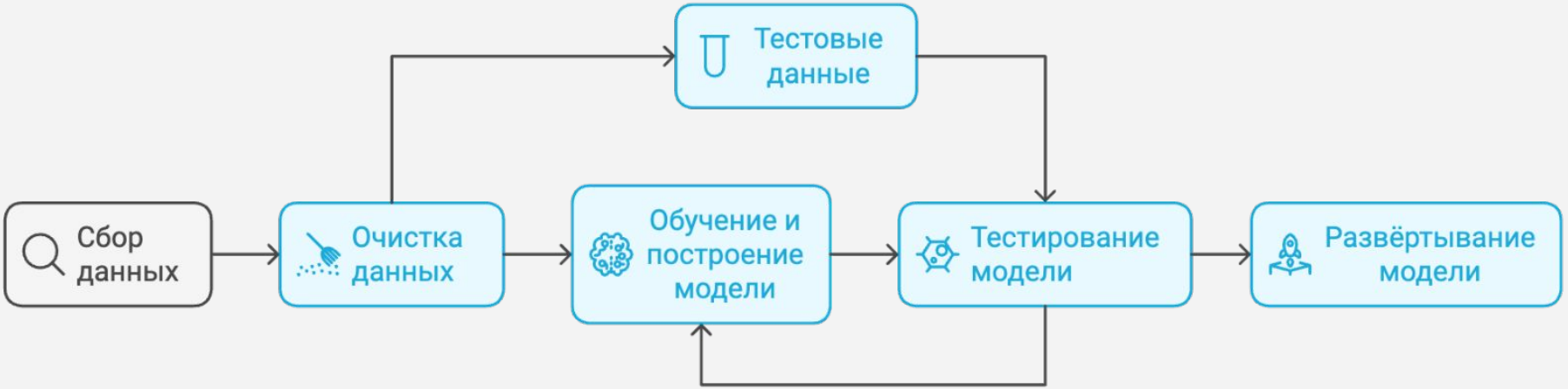
Тестирование модели

Оценка модели с использованием тестовых данных.



Развёртывание модели

Это этап, где модель применяется для реальных данных.



Основные этапы очистки данных

Обнаружение и обработка пропущенных значений

Проблема: В реальных данных часто отсутствуют значения.

Решения: Удаление записей с пропущенными значениями (если их мало).

```
df.dropna(inplace=True)
```

Заполнение пропусков средним, медианой или наиболее частым значением.

```
df['Возраст'].fillna(df['Возраст'].mean(),
inplace=True)
```

Удаление дубликатов

Проблема: Дублированные записи (например, одна и та же транзакция записана дважды) искажают результаты анализа.

Решение:

```
df.drop_duplicates(inplace=True)
```

Обработка выбросов

Проблема: Экстремальные значения могут "утащить" модель в неверную сторону. Например, один клиент магазина зарегистрировал доход в 1 миллиард долларов.

Решение: Использование межквартильного размаха (IQR) для обнаружения выбросов:

```
Q1 = df['Доход'].quantile(0.25)

Q3 = df['Доход'].quantile(0.75)
IQR = Q3 - Q1
df = df[(df['Доход'] >= Q1 - 1.5 * IQR) & (df['Доход'] <= Q3 + 1.5 * IQR)]
```

Предобработка данных

Нормализация данных

Алгоритмы, такие как линейная регрессия и K-Means, чувствительны к масштабам данных.

Методы нормализации:

- Min-Max Scaling.
- Standard Scaling (стандартизация).

Нормализация данных

Если одна переменная (например, вес) измеряется в килограммах, а другая (доход) в миллионах, модель может ошибочно придать больший вес переменной "доход".

Решение: Приведение данных к единой шкале (например, min-max scaling):

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df[['Вес', 'Рост']] = scaler.fit_transform(df[['Вес', 'Рост']])
```

Кодирование категориальных данных

Машины не работают с текстовыми данными, их нужно преобразовать в числа.

Методы:

- **One-Hot Encoding:** Создаёт столбцы для каждой категории.
- **Label Encoding:** Присваивает категориям числовые значения.

Кодирование категориальных данных

Модели не могут работать с текстом (например, "Мужчина" или "Женщина").

Решение:

- One-hot encoding:
  `pd.get_dummies(df['Пол'], drop_first=True)`
- Label encoding:

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['Пол'] = le.fit_transform(df['Пол'])
```

Разделение данных

Чтобы проверить модель на новых данных.

Пропорции разделения:

- 70% для обучения, 30% для тестирования.
- Альтернативы: 80/20, 60/40.

Разделение данных

Для оценки модели необходимо тестировать её на данных, которые она не видела ранее.

Решение:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.2,
random_state=42)
```

Обработка текстовых данных

Задачи: Удаление лишних символов (стоп-слов, знаков).

Методы:

- TF-IDF (взвешивание слов).
- Bag-of-Words (модель мешка слов).

```
from sklearn.feature_extraction.text import
TfidfVectorizer

tfidf = TfidfVectorizer()

X = tfidf.fit_transform(['текст'])
```

Работа с изображениями

Проблемы: разные размеры изображений, неравномерное освещение, шумы.

Методы обработки:

- Изменение размеров изображений до фиксированного размера.
- Нормализация значений пикселей (деление на 255 для диапазона [0, 1]).

Использование библиотек: **OpenCV, PIL.**

Работа с изображениями

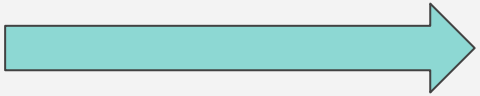
```
from PIL import Image

img = Image.open('example.jpg')

img_resized = img.resize((128, 128))  # Изменение
размера

img_resized.show()
```

Работа с изображениями



Примеры

Пример 1: Пропущенные значения

Пример обработки пропусков.

Ситуация: В данных о клиентах отсутствует возраст.

- Проблема: это мешает корректному анализу.

Решение: Заполнить пропущенные значения медианой возраста.

Пример кода:

```
df['Возраст'] =
df['Возраст'].fillna(df['Возраст'].median())
```

Пример 1: Пропущенные значения

```
import pandas as pd

# Создание данных с пропущенными значениями
data = {
    'Имя': ['Анна', 'Иван', 'Мария', 'Петр', 'Елена'],
    'Возраст': [23, 25, None, 29, None],
    'Город': ['Москва', 'Киев', 'Минск', 'Москва', 'Киев']
}

df = pd.DataFrame(data)
```

Пример 1: Пропущенные значения

До обработки:

	Имя	Возраст	Город
0	Анна	23.0	Москва
1	Иван	25.0	Киев
2	Мария	NaN	Минск
3	Петр	29.0	Москва
4	Елена	NaN	Киев

После обработки:

	Имя	Возраст	Город
0	Анна	23.0	Москва
1	Иван	25.0	Киев
2	Мария	25.0	Минск
3	Петр	29.0	Москва
4	Елена	25.0	Киев

Пример 2: Удаление строк с пропусками

Если вы хотите полностью удалить строки с пропущенными значениями:

Решение:

```
# Удаление строк с пропущенными значениями
df_cleaned = df.dropna()
print("\nПосле удаления строк с пропущенными значениями:")
print(df_cleaned)
```

Пример 2: Удаление строк с пропусками

```
После удаления строк с пропущенными значениями:
     Имя  Возраст   Город
0   Анна      23.0  Москва
1   Иван      25.0    Киев
3   Петр      29.0  Москва
```

Пример 3: Заполнение пропущенных значений специальным значением

```
# Заполнение пропусков специальным значением

df['Возраст'] = df['Возраст'].
fillna('Неизвестно')
print("\nПосле заполнения
специальным значением:")
print(df)
```

```
После удаления строк с пропущенными значениями:
     Имя     Возраст   Город
0   Анна        23.0  Москва
1   Иван        25.0    Киев
2  Мария  Неизвестно   Минск
3   Петр        29.0  Москва
4  Елена  Неизвестно    Киев
```

Пример 4: Удаление столбцов с пропусками

Пример кода:

```
df_cleaned = df.dropna(axis=1) print("\nПосле
удаления столбцов
с пропущенными
значениями:")
print(df_cleaned)
```

```
После удаления столбцов с пропущенными значениями:
     Имя   Город
0   Анна  Москва
1   Иван    Киев
2  Мария   Минск
3   Петр  Москва
4  Елена    Киев
```

Пример 5: Нормализация данных с использованием Min-Max Scaling

```
import pandas as pd

data = {
    'Имя': ['Анна', 'Иван', 'Мария', 'Петр', 'Елена'],
    'Рост': [160, 170, 150, 180, 165],  # в сантиметрах
    'Вес': [50, 65, 45, 80, 60]         # в килограммах
}
df = pd.DataFrame(data)
print("До нормализации:")
print(df)
```

Пример 5: Нормализация данных с использованием Min-Max Scaling

```
from sklearn.preprocessing import MinMaxScaler

# Создаём объект MinMaxScaler
scaler = MinMaxScaler()

# Нормализуем колонки "Рост" и "Вес"
df[['Рост', 'Вес']] = scaler.fit_transform(df[['Рост',
'Вес']])
print("\nПосле нормализации:")
print(df)
```

Пример 5: Нормализация данных с использованием Min-Max Scaling

До нормализации:

	Имя	Рост	Вес
0	Анна	160	50
1	Иван	170	65
2	Мария	150	45
3	Петр	180	80
4	Елена	165	60

После нормализации:

	Имя	Рост	Вес
0	Анна	0.333333	0.142857
1	Иван	0.666667	0.571429
2	Мария	0.000000	0.000000
3	Петр	1.000000	1.000000
4	Елена	0.500000	0.428571

Пример 6: Визуализация данных

Важность визуализации: помогает найти выбросы и понять распределение данных.

Инструменты:

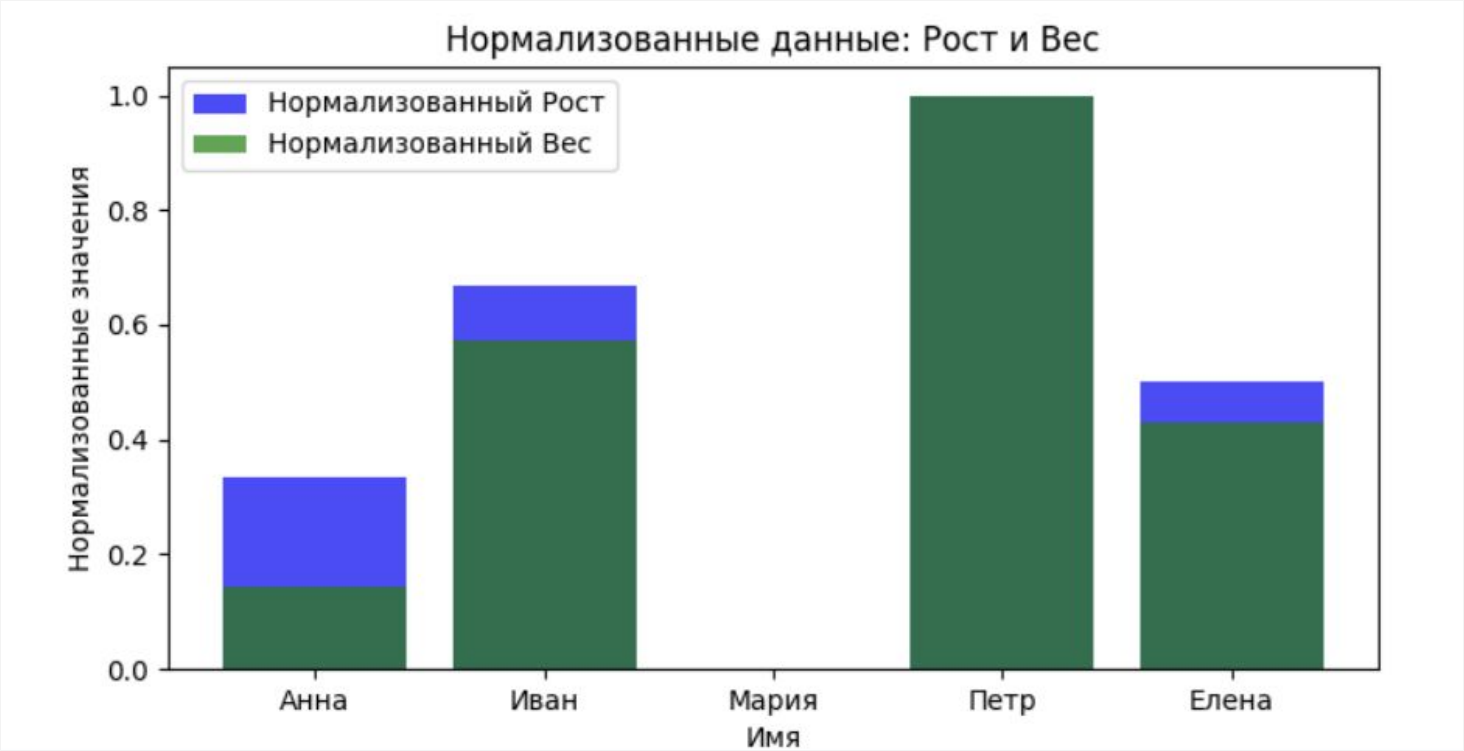
- **Boxplot:** для обнаружения выбросов.
- **Histplot:** для анализа распределения.

Пример 6: Визуализация

```
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 4))
plt.bar(df['Имя'], df['Рост'], color='blue', alpha=0.7,
label='Нормализованный Рост')
plt.bar(df['Имя'], df['Вес'], color='green', alpha=0.7,
label='Нормализованный Вес')
plt.legend()  plt.title("Нормализованные данные: Рост и Вес")
plt.xlabel("Имя")  plt.ylabel("Нормализованные значения")
plt.show()
```

Пример 6: Визуализация



Пример 6: Визуализация