**Лабораторная работа №12-14**

**Задача 1. Классификация изображений рукописных цифр. Набор данных: [MNIST]**

Код реализует классификацию изображений рукописных цифр из набора данных MNIST с использованием алгоритма K-ближайших соседей (K-NN). MNIST — это стандартный набор данных для тестирования алгоритмов машинного обучения, содержащий 70,000 изображений цифр от 0 до 9.

**Импорт библиотек**

Код начинается с импорта необходимых библиотек:

* numpy для работы с массивами и математическими операциями.
* matplotlib.pyplot для визуализации изображений.
* sklearn для загрузки данных, разделения на выборки, создания и оценки модели K-NN.

**Загрузка данных**

mnist = fetch\_openml('mnist\_784', version=1)

X, y = mnist['data'], mnist['target']

y = y.astype(np.uint8)

Данные загружаются с помощью fetch\_openml, после чего разделяются на матрицу признаков X (изображения) и целевую переменную y (метки цифр). Метки преобразуются в тип uint8.

**Разделение данных**

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

Данные делятся на обучающую (80%) и тестовую (20%) выборки для оценки модели.

**Создание и обучение модели**

knn = KNeighborsClassifier(n\_neighbors=3)

knn.fit(X\_train, y\_train)

Создается классификатор K-NN с 3 ближайшими соседями. Модель обучается на обучающей выборке.

**Предсказание и оценка модели**

y\_pred = knn.predict(X\_test)

print("Матрица ошибок:")

print(confusion\_matrix(y\_test, y\_pred))

print("\nОтчет о классификации:")

print(classification\_report(y\_test, y\_pred))

Модель делает предсказания на тестовой выборке. Затем выводится матрица ошибок и отчет о классификации, включающий точность, полноту и F1-меру для каждой цифры.

**Визуализация результатов**

plt.figure(figsize=(10, 5))

for i in range(10):

plt.subplot(2, 5, i + 1)

plt.imshow(X\_test.iloc[i].values.reshape(28, 28), cmap='gray')

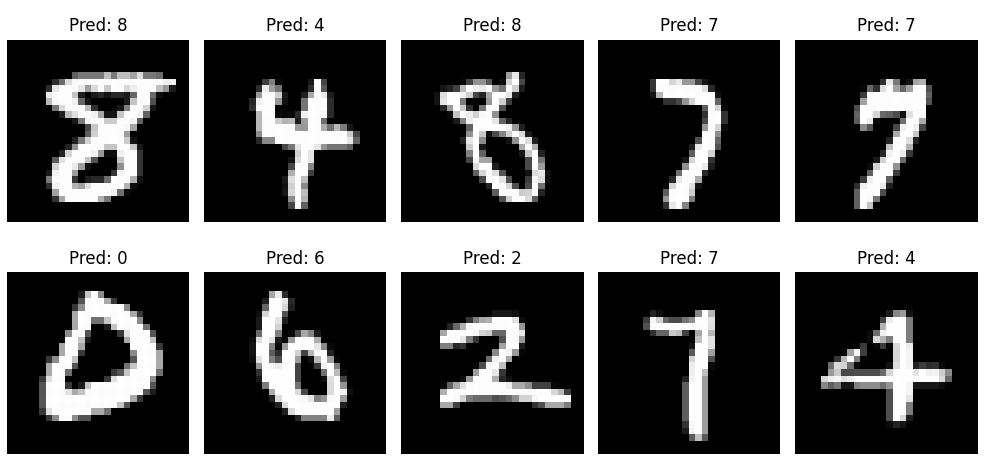
plt.title(f'Pred: {y\_pred[i]}')

plt.axis('off')

plt.tight\_layout()

plt.show()

Код визуализирует 10 случайно выбранных предсказаний, отображая изображения и соответствующие им предсказанные метки.



**Задача 2. Определение мошенничества с кредитными картами. Набор данных: [Credit Card Fraud Detection]**

Код реализует обнаружение мошенничества с кредитными картами с использованием алгоритма случайного леса (Random Forest). Набор данных содержит транзакции, каждая из которых помечена как законная или мошенническая. Целью является построение модели, способной правильно классифицировать транзакции на основе предоставленных признаков.

Импорт библиотек

Код начинается с импорта необходимых библиотек:

* pandas для работы с данными в формате таблиц.
* numpy для работы с массивами и математическими операциями.
* matplotlib.pyplot и seaborn для визуализации результатов.
* sklearn для разделения данных, создания и оценки модели случайного леса.

Загрузка данных

data = pd.read\_csv('creditcard.csv')

Данные загружаются из CSV-файла с помощью pandas. После загрузки выводятся первые строки данных для визуального анализа.

Проверка на пропущенные значения

print(data.isnull().sum())

Проверяются пропущенные значения в данных, что позволяет убедиться в их целостности перед дальнейшей обработкой.

Разделение данных

X = data.drop(['Time', 'Class'], axis=1)

y = data['Class'].astype(int)

Признаки X формируются путем удаления столбцов Time и Class. Целевая переменная y представляет собой столбец Class, который приводится к типу int.

Стандартизация данных

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

Признаки стандартизируются с помощью StandardScaler, что помогает улучшить качество модели, особенно при использовании алгоритмов, чувствительных к масштабу данных.

Разделение на обучающую и тестовую выборки

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42, stratify=y)

Данные делятся на обучающую (80%) и тестовую (20%) выборки с использованием функции train\_test\_split, что позволяет оценить модель на независимом наборе данных.

Создание и обучение модели

model = RandomForestClassifier(n\_estimators=100, random\_state=42)

model.fit(X\_train, y\_train)

Создается модель случайного леса с 100 деревьями. Модель обучается на обучающей выборке.

Предсказание и оценка модели

y\_pred = model.predict(X\_test)

print("Матрица ошибок:")

print(confusion\_matrix(y\_test, y\_pred))

print("\nОтчет о классификации:")

print(classification\_report(y\_test, y\_pred))

Модель делает предсказания на тестовой выборке. Затем выводится матрица ошибок и отчет о классификации, включающий важные метрики, такие как точность, полнота и F1-мера.

Визуализация результатов

plt.figure(figsize=(8, 6))

sns.heatmap(confusion\_matrix(y\_test, y\_pred), annot=True, fmt='d', cmap='Blues')

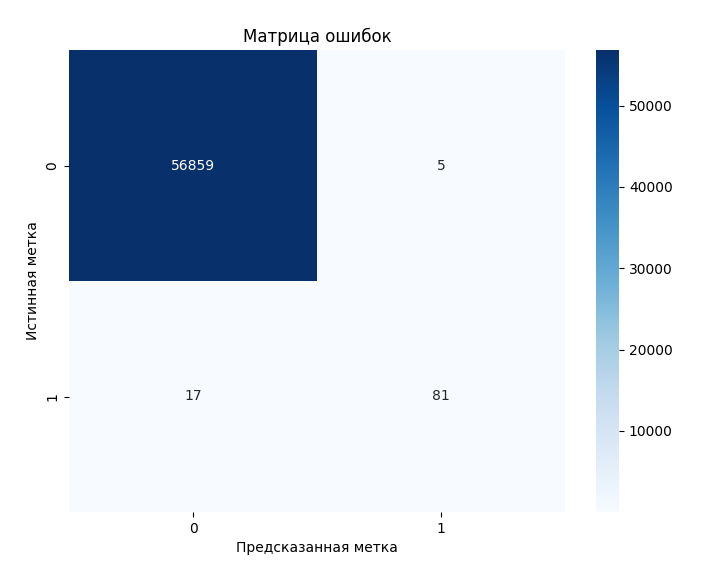
plt.xlabel('Предсказанная метка')

plt.ylabel('Истинная метка')

plt.title('Матрица ошибок')

plt.show()

Матрица ошибок визуализируется с использованием seaborn, что позволяет наглядно оценить производительность модели и увидеть, как она справляется с классификацией.



**Интерпретация матрицы ошибок**

**Истинные положительные (TP)**: 81

Модель правильно предсказала 81 мошенническую транзакцию (класс 1).

**Истинные отрицательные (TN)**: 56859

Модель правильно предсказала 56859 законных транзакций (класс 0).

**Ложные положительные (FP)**: 5

Модель ошибочно классифицировала 5 законных транзакций как мошеннические.

**Ложные отрицательные (FN)**: 17

Модель ошибочно классифицировала 17 мошеннических транзакций как законные.

**Задача 3. Классификация рака молочной железы. Набор данных: [Breast Cancer Wisconsin (Diagnostic)]**

Код предназначен для классификации данных о раке молочной железы из набора Wisconsin Diagnostic Breast Cancer (WDBC). Используется библиотека scikit-learn, а именно алгоритм случайного леса, для построения классификатора, который будет различать злокачественные (M) и доброкачественные (B) опухоли.

Основные компоненты кода

1. Импорт библиотек

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification\_report, confusion\_matrix

from sklearn.preprocessing import StandardScaler

import seaborn as sns

import matplotlib.pyplot as plt

* pandas: для работы с данными в формате таблиц.
* sklearn: для разделения данных, обучения модели и оценки результатов.
* seaborn и matplotlib: для визуализации результатов.

1. Загрузка данных

data = pd.read\_csv('wdbc.data', header=None)

* Данные загружаются из файла wdbc.data. Параметр header=None указывает, что файл не имеет заголовков.

1. Переименование колонок

data.columns = [

'ID', 'Diagnosis',

'Radius\_Mean', 'Texture\_Mean', 'Perimeter\_Mean', 'Area\_Mean',

'Smoothness\_Mean', 'Compactness\_Mean', 'Concavity\_Mean',

'Concave\_Points\_Mean', 'Symmetry\_Mean', 'Fractal\_Dimension\_Mean',

'Radius\_SE', 'Texture\_SE', 'Perimeter\_SE', 'Area\_SE',

'Smoothness\_SE', 'Compactness\_SE', 'Concavity\_SE',

'Concave\_Points\_SE', 'Symmetry\_SE', 'Fractal\_Dimension\_SE',

'Radius\_Worst', 'Texture\_Worst', 'Perimeter\_Worst',

'Area\_Worst', 'Smoothness\_Worst', 'Compactness\_Worst',

'Concavity\_Worst', 'Concave\_Points\_Worst', 'Symmetry\_Worst',

'Fractal\_Dimension\_Worst'

]

* Назначаются имена колонок для удобного доступа к данным.

1. Преобразование целевой переменной

data['Diagnosis'] = data['Diagnosis'].map({'M': 1, 'B': 0})

* Диагноз преобразуется в числовой формат: 1 для злокачественной опухоли и 0 для доброкачественной.

1. Разделение данных на признаки и целевую переменную

X = data.drop(['ID', 'Diagnosis'], axis=1)

y = data['Diagnosis']

* X содержит все признаки, за исключением ID и диагноза.
* y содержит целевую переменную (диагноз).

1. Стандартизация данных

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

* Признаки стандартизируются для улучшения работы алгоритмов машинного обучения.

1. Разделение данных на обучающую и тестовую выборки

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42, stratify=y)

* Данные разделяются на обучающую (80%) и тестовую (20%) выборки. Параметр stratify=y обеспечивает пропорциональное распределение классов.

1. Создание и обучение модели

model = RandomForestClassifier(n\_estimators=100, random\_state=42)

model.fit(X\_train, y\_train)

* Создается модель случайного леса с 100 деревьями, которая обучается на обучающей выборке.

1. Предсказание и оценка модели

y\_pred = model.predict(X\_test)

print("Матрица ошибок:")

print(confusion\_matrix(y\_test, y\_pred))

print("\nОтчет о классификации:")

print(classification\_report(y\_test, y\_pred))

* Модель делает предсказания на тестовой выборке.
* Выводится матрица ошибок и отчет о классификации, включая точность, полноту и F1-меру.

1. Визуализация матрицы ошибок

plt.figure(figsize=(8, 6))

sns.heatmap(confusion\_matrix(y\_test, y\_pred), annot=True, fmt='d', cmap='Blues')

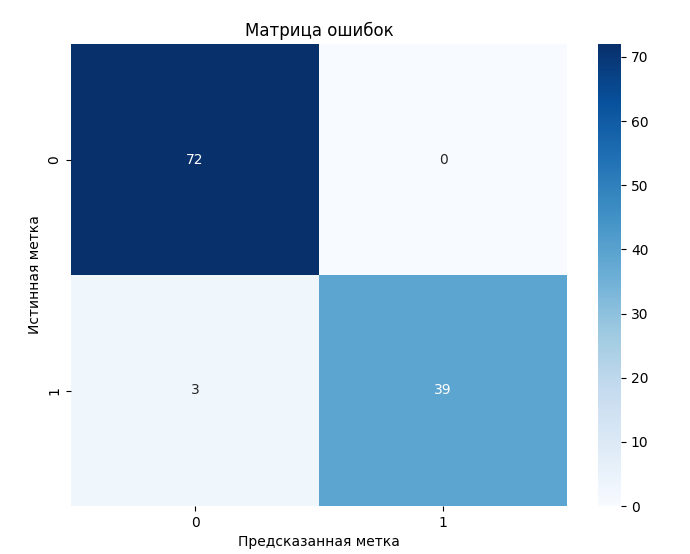
plt.xlabel('Предсказанная метка')

plt.ylabel('Истинная метка')

plt.title('Матрица ошибок')

plt.show()

* Используется seaborn для визуализации матрицы ошибок, что позволяет легко оценить производительность модели.

****

72 (в ячейке [0, 0]): Модель правильно предсказала 72 доброкачественные опухоли.

0 (в ячейке [0, 1]): Модель не ошиблась, не предсказав никаких доброкачественных опухолей как злокачественные.

3 (в ячейке [1, 0]): Модель ошибочно предсказала 3 злокачественные опухоли как доброкачественные.

39 (в ячейке [1, 1]): Модель правильно предсказала 39 злокачественных опухолей.

**Задача 4. Спам-фильтрация для текстовых сообщений. Набор данных: [SMS Spam Collection]**

Разработка модели для классификации текстовых сообщений на спам и не спам, используя логистическую регрессию. Код включает в себя предобработку данных, обучение модели и оценку ее производительности.

1. Импорт библиотек

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import classification\_report, confusion\_matrix, roc\_curve, auc

import matplotlib.pyplot as plt

* pandas: для обработки данных.
* numpy: для работы с массивами и математическими операциями.
* scikit-learn: для построения и оценки модели.
* matplotlib: для визуализации результатов.

1. Загрузка данных

data = pd.read\_csv('spam.csv', encoding='latin-1')

data = data[['v1', 'v2']]

data.columns = ['label', 'message']

* Данные загружаются из файла spam.csv, и выбираются необходимые столбцы (метки и сообщения).

1. Векторизация текста

vectorizer = CountVectorizer()

X = vectorizer.fit\_transform(data['message'])

* Текстовые сообщения преобразуются в числовые векторы с помощью CountVectorizer. Это позволяет модели работать с текстовыми данными.

1. Определение целевой переменной

y = data['label'].map({'ham': 0, 'spam': 1})

* Метки классов преобразуются в числовой формат (0 для "ham" и 1 для "spam").

1. Разделение данных на обучающую и тестовую выборки

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42, stratify=y)

* Данные делятся на обучающую (80%) и тестовую (20%) выборки.

1. Обучение модели логистической регрессии

model = LogisticRegression(max\_iter=1000)

model.fit(X\_train, y\_train)

* Модель логистической регрессии обучается на обучающей выборке.

1. Предсказание и оценка модели

y\_pred = model.predict(X\_test)

print("Отчет о классификации:")

print(classification\_report(y\_test, y\_pred))

* Модель предсказывает метки для тестовой выборки, и выводится отчет о классификации, который включает точность, полноту и F1-меру.

1. Матрица ошибок

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

print("Матрица ошибок:")

print(conf\_matrix)

* Выводится матрица ошибок, показывающая количество правильно и неправильно классифицированных примеров.

1. ROC-кривая

y\_prob = model.predict\_proba(X\_test)[:, 1]

fpr, tpr, \_ = roc\_curve(y\_test, y\_prob)

roc\_auc = auc(fpr, tpr)

* Строится ROC-кривая для оценки качества модели, и рассчитывается значение AUC (площадь под кривой).

1. Визуализация результатов

plt.figure()

plt.plot(fpr, tpr, label=f'ROC curve (AUC = {roc\_auc:.2f})')

plt.plot([0, 1], [0, 1], 'k--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC Curve')

plt.legend(loc='best')

plt.show()

* Визуализируется ROC-кривая, показывающая компромисс между TPR и FPR.

1. Важность признаков

importance = model.coef\_[0]

feature\_names = vectorizer.get\_feature\_names\_out()

* Извлекаются коэффициенты модели для оценки важности признаков.

1. Визуализация важности признаков

plt.figure(figsize=(12, 6))

plt.title("Важность признаков")

plt.bar(range(top\_n), importance[indices][:top\_n], align="center")

plt.xticks(range(top\_n), feature\_names[indices][:top\_n], rotation=90)

plt.xlim([-1, top\_n])

plt.ylabel('Важность')

plt.show()

* Строится столбчатая диаграмма, показывающая топ-N признаков по важности.

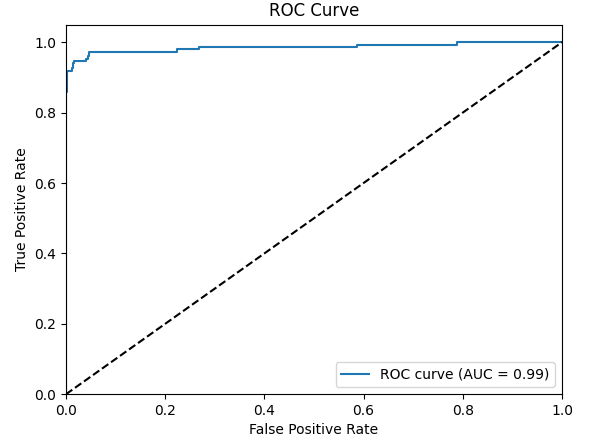
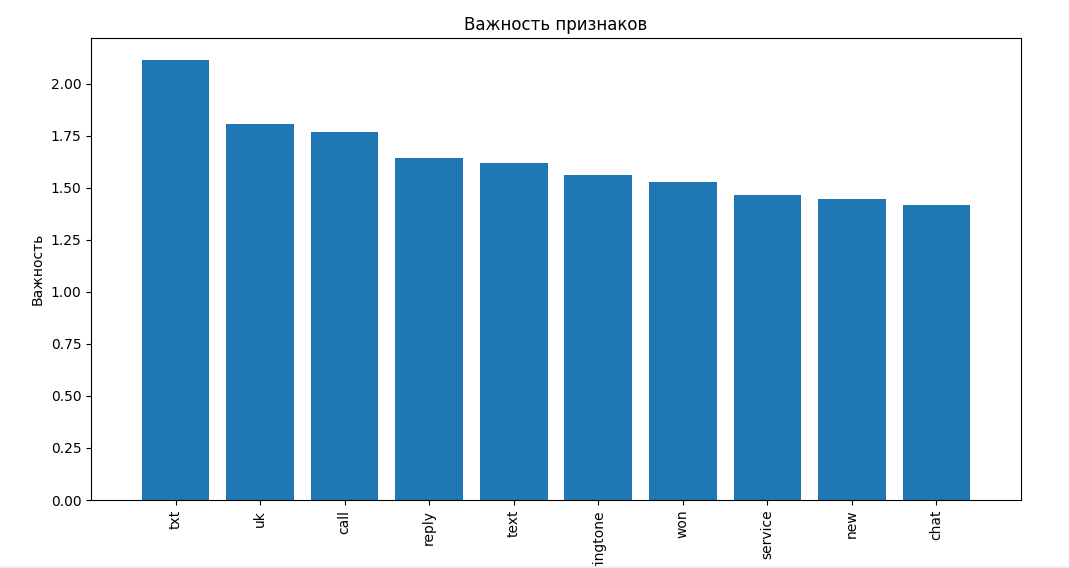


График ROC (Receiver Operating Characteristic) показывает, как изменяется доля истинных положительных результатов (True Positive Rate, TPR) в зависимости от доли ложных положительных результатов (False Positive Rate, FPR) при различных пороговых значениях классификации.



Матрица ошибок:

[[965 1]

[ 20 129]]

**TP (True Positives)**: 129

Это число правильно классифицированных положительных примеров (спам).

**TN (True Negatives)**: 965

Это число правильно классифицированных отрицательных примеров (не спам).

**FP (False Positives)**: 1

Это число отрицательных примеров, которые были ошибочно классифицированы как положительные (не спам, но классифицированы как спам).

**FN (False Negatives)**: 20

Это число положительных примеров, которые были ошибочно классифицированы как отрицательные (спам, но классифицирован как не спам).

**Задача 5. Классификация новостных статей. Набор данных: [20 Newsgroups]**

Импорт библиотек:

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import fetch\_20newsgroups

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.naive\_bayes import MultinomialNB

from sklearn.model\_selection import train\_test\_split

from sklearn import metrics

from sklearn.metrics import confusion\_matrix, ConfusionMatrixDisplay

Загрузка данных:

* Загружаем данные только для двух категорий с использованием fetch\_20newsgroups.

categories = ['alt.atheism', 'sci.space']

newsgroups = fetch\_20newsgroups(subset='all', categories=categories, remove=('headers', 'footers', 'quotes'))

Разделение данных:

* Разделяем данные на обучающую и тестовую выборки с использованием функции train\_test\_split.

X\_train, X\_test, y\_train, y\_test = train\_test\_split(newsgroups.data, newsgroups.target, test\_size=0.2, random\_state=42)

Векторизация текста:

* Преобразуем текстовые данные в TF-IDF векторы с использованием TfidfVectorizer.

vectorizer = TfidfVectorizer()

X\_train\_tfidf = vectorizer.fit\_transform(X\_train)

X\_test\_tfidf = vectorizer.transform(X\_test)

Обучение модели:

* Обучаем модель наивного байесовского классификатора (MultinomialNB) на обучающих данных.

classifier = MultinomialNB()

classifier.fit(X\_train\_tfidf, y\_train)

Предсказание:

* Получаем предсказания для тестового набора данных.

y\_pred = classifier.predict(X\_test\_tfidf)

Создание матрицы ошибок:

* Создаем и отображаем матрицу ошибок для визуализации результатов.

cm = confusion\_matrix(y\_test, y\_pred)

disp = ConfusionMatrixDisplay(confusion\_matrix=cm, display\_labels=newsgroups.target\_names)

disp.plot(cmap=plt.cm.Blues)

plt.title('Confusion Matrix')

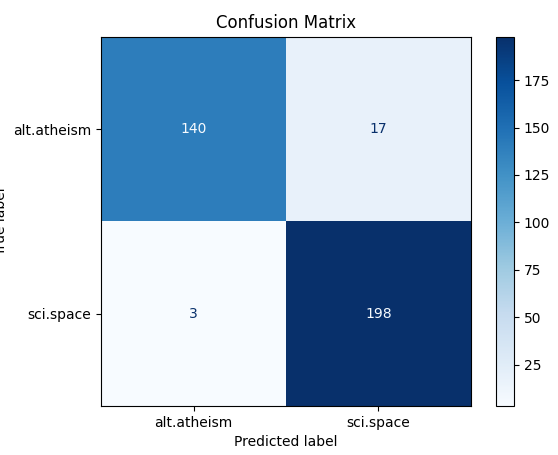
plt.show()

Печать матрицы ошибок:

* Выводим матрицу ошибок на экран.

print("Матрица ошибок:")

print(cm)



**Правильные предсказания:**

**140**: Примеры из категории "alt.atheism" правильно классифицированы.

**198**: Примеры из категории "sci.space" также правильно классифицированы.

**Неправильные предсказания:**

**17**: Примеры из "alt.atheism" ошибочно классифицированы как "sci.space".

**3**: Примеры из "sci.space" ошибочно классифицированы как "alt.atheism".

**Импорт библиотек**:

* Использовались библиотеки pandas, numpy, sklearn, matplotlib, и seaborn для обработки данных, обучения модели и визуализации результатов.

**Чтение данных**:

* Данные загружались из файла

glass.data с помощью pandas.

glass\_data = pd.read\_csv('glass.data', header=None)

**Предобработка данных**:

* Удаление ненужного столбца Index.
* Разделение данных на признаки (X) и целевую переменную (y).

glass\_data.drop('Index', axis=1, inplace=True)

X = glass\_data.drop('Type', axis=1)

y = glass\_data['Type']

**Разделение данных**:

* Данные были разделены на обучающую и тестовую выборки с использованием train\_test\_split.

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

**Стандартизация данных**:

* Признаки были стандартизированы с помощью StandardScaler, чтобы улучшить производительность модели.

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

**Обучение модели**:

* Модель классификации была создана с использованием алгоритма Random Forest.

classifier = RandomForestClassifier(random\_state=42)

classifier.fit(X\_train, y\_train)

**Предсказание и оценка модели**:

* Модель предсказывала тип стекла для тестовых данных, и результаты оценивались с помощью отчета о классификации и матрицы ошибок.

y\_pred = classifier.predict(X\_test)

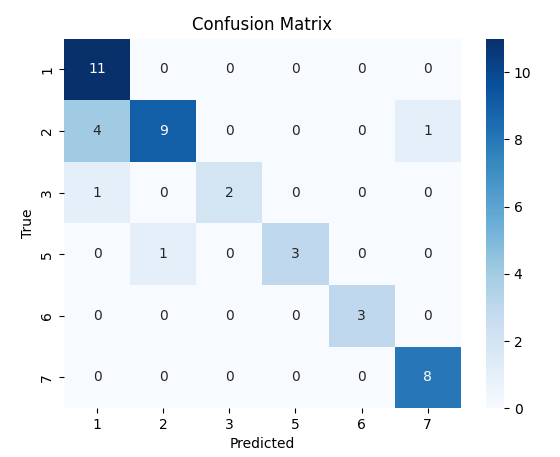
print(classification\_report(y\_test, y\_pred))

print(confusion\_matrix(y\_test, y\_pred))

**Визуализация**:

* Матрица ошибок была визуализирована с использованием тепловой карты для лучшего понимания результатов.

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')



**Тип 1**: 11 образцов правильно классифицированы, 0 неправильно.

**Тип 2**: 9 образцов правильно классифицированы, 4 образца неправильно классифицированы как тип 1.

**Тип 3**: 2 образца правильно классифицированы, 1 неправильно классифицирован как тип 2.

**Тип 4**: 3 образца правильно классифицированы, 0 неправильно.

**Тип 5**: 0 образцов правильно классифицированы, 3 неправильно классифицированы как тип 6.

**Тип 7**: 8 образцов правильно классифицированы, 0 неправильно.