Задача 1. Классификация изображений рукописных цифр

Набор данных: MNIST.

Предварительная обработка: масштабирование пикселей, разделение данных на обучение и тестирование.

Модели: логистическая регрессия, SVM, случайный лес, градиентный бустинг, нейронные сети.

Сравнение модели: точность, матрица ошибок, ROC-кривые.

Изучение и подбор гиперпараметров: GridSearchCV или RandomizedSearchCV.

Визуализация: отображение изображений, важности признаков или активаций нейронной сети.

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn import datasets

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

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import LogisticRegression

from sklearn.svm import SVC

from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier

from sklearn.neural\_network import MLPClassifier

from sklearn.metrics import accuracy\_score, confusion\_matrix, roc\_curve, auc

wine = datasets.load\_wine()

X = pd.DataFrame(wine.data, columns=wine.feature\_names)

y = pd.Series(wine.target)

scaler = StandardScaler()

X = scaler.fit\_transform(X)

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

log\_reg = LogisticRegression(max\_iter=1000)

log\_reg.fit(X\_train, y\_train)

y\_pred\_log\_reg = log\_reg.predict(X\_test)

accuracy\_log\_reg = accuracy\_score(y\_test, y\_pred\_log\_reg)

svm = SVC(kernel='linear', probability=True)

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

y\_pred\_svm = svm.predict(X\_test)

accuracy\_svm = accuracy\_score(y\_test, y\_pred\_svm)

rf = RandomForestClassifier(n\_estimators=100)

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

y\_pred\_rf = rf.predict(X\_test)

accuracy\_rf = accuracy\_score(y\_test, y\_pred\_rf)

gb = GradientBoostingClassifier(n\_estimators=100)

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

y\_pred\_gb = gb.predict(X\_test)

accuracy\_gb = accuracy\_score(y\_test, y\_pred\_gb)

mlp = MLPClassifier(hidden\_layer\_sizes=(100,), max\_iter=1000)

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

y\_pred\_mlp = mlp.predict(X\_test)

accuracy\_mlp = accuracy\_score(y\_test, y\_pred\_mlp)

print(f"Accuracy - Logistic Regression: {accuracy\_log\_reg:.4f}")

print(f"Accuracy - SVM: {accuracy\_svm:.4f}")

print(f"Accuracy - Random Forest: {accuracy\_rf:.4f}")

print(f"Accuracy - Gradient Boosting: {accuracy\_gb:.4f}")

print(f"Accuracy - MLP: {accuracy\_mlp:.4f}")

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

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

sns.heatmap(cm\_log\_reg, annot=True, fmt="d", cmap="Blues", xticklabels=wine.target\_names, yticklabels=wine.target\_names)

plt.title('Confusion Matrix - Logistic Regression')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.show()

from sklearn.preprocessing import label\_binarize

y\_test\_bin = label\_binarize(y\_test, classes=[0, 1, 2])

y\_score\_log\_reg = log\_reg.predict\_proba(X\_test)

fpr = {}

tpr = {}

roc\_auc = {}

for i in range(3):

    fpr[i], tpr[i], \_ = roc\_curve(y\_test\_bin[:, i], y\_score\_log\_reg[:, i])

    roc\_auc[i] = auc(fpr[i], tpr[i])

plt.figure()

colors = ['blue', 'red', 'green']

for i, color in enumerate(colors):

    plt.plot(fpr[i], tpr[i], color=color, lw=2, label=f'Class {i} (area = {roc\_auc[i]:.2f})')

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 - Logistic Regression')

plt.legend(loc="lower right")

plt.show()

param\_grid = {'C': [0.01, 0.1, 1, 10], 'solver': ['liblinear', 'saga']}

grid\_search = GridSearchCV(LogisticRegression(max\_iter=1000), param\_grid, cv=3)

grid\_search.fit(X\_train, y\_train)

print("Best parameters for Logistic Regression:", grid\_search.best\_params\_)

importances = rf.feature\_importances\_

indices = np.argsort(importances)[::-1]

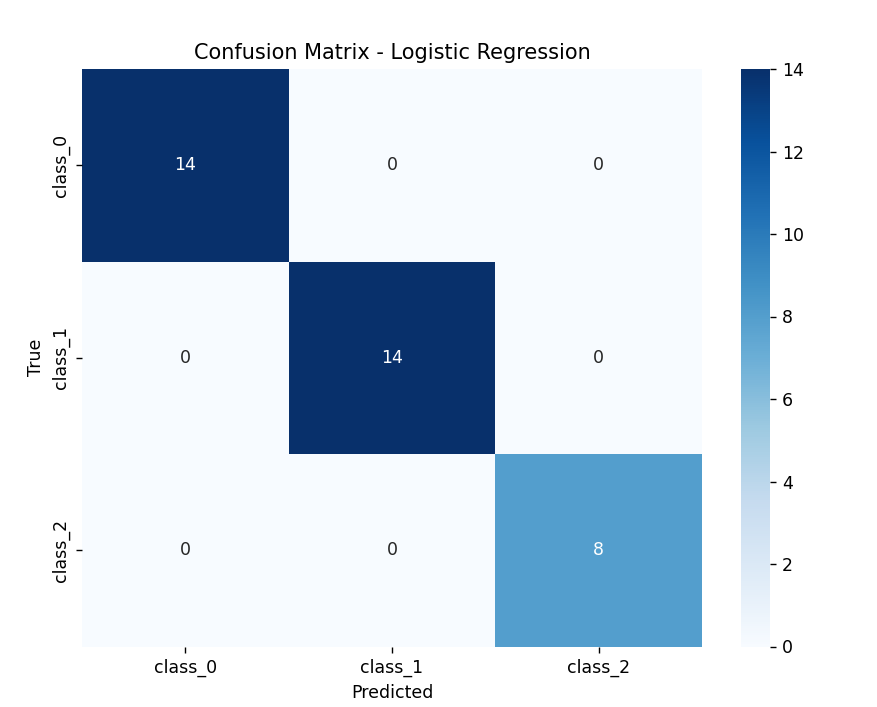
plt.figure(figsize=(10, 6))

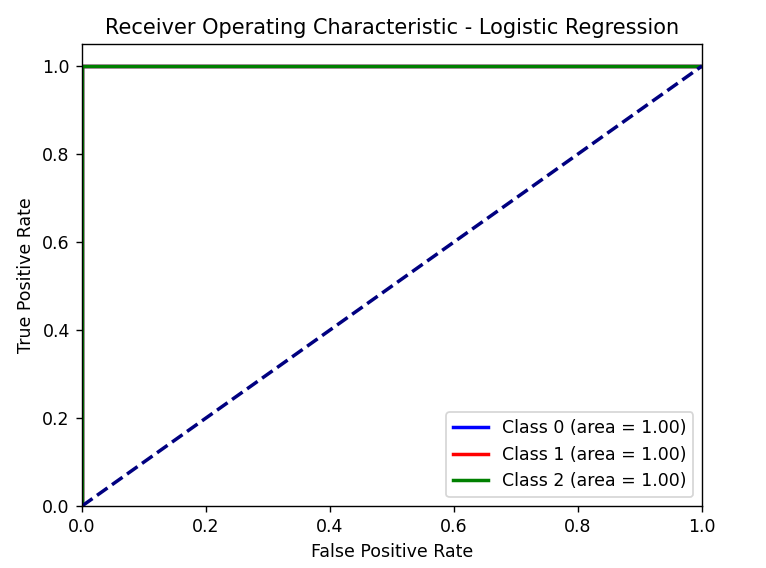
plt.bar(range(X.shape[1]), importances[indices], align="center")

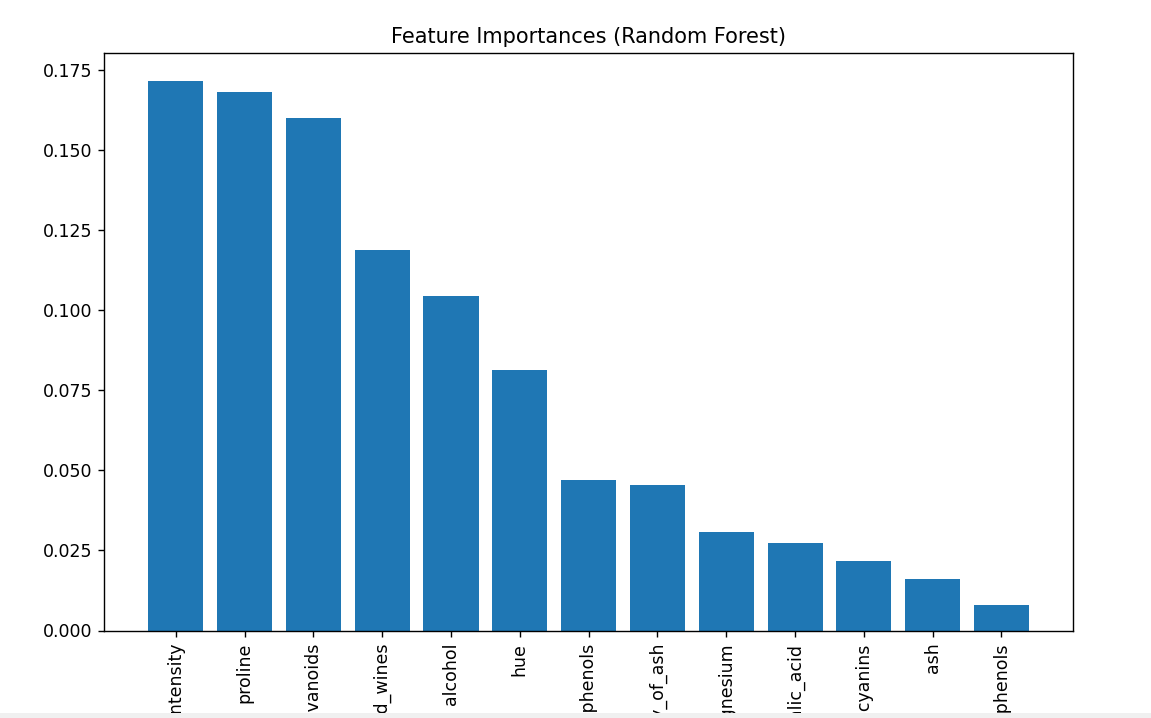
plt.xticks(range(X.shape[1]), np.array(wine.feature\_names)[indices], rotation=90)

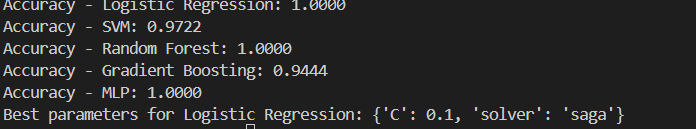
plt.title("Feature Importances (Random Forest)")

plt.show()









Задача 2. Определение мошенничества с кредитными картами

Набор данных: Credit Card Fraud Detection dataset на Kaggle.

Предварительная обработка: масштабирование, обработка несбалансированных данных.

Модели: логистическая регрессия, SVM, случайный лес, градиентный бустинг, нейронные сети.

Сравнение моделей: точность, полнота, ROC-кривые.

Изучение и подбор гиперпараметров: GridSearchCV или RandomizedSearchCV.

Визуализация: важности признаков, ROC-кривые.

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.datasets import load\_wine

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

from sklearn.linear\_model import LogisticRegression

from sklearn.svm import SVC

from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier

from sklearn.neural\_network import MLPClassifier

from sklearn.metrics import accuracy\_score, confusion\_matrix, roc\_curve, roc\_auc\_score, RocCurveDisplay

# Загрузка данных

wine = load\_wine()

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

# Разделение на тренировочные и тестовые данные

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

# Словарь для моделей и их параметров

models = {

    'Logistic Regression': LogisticRegression(max\_iter=1000, multi\_class='ovr'),

    'SVM': SVC(probability=True),

    'Random Forest': RandomForestClassifier(),

    'Gradient Boosting': GradientBoostingClassifier(),

    'Neural Network': MLPClassifier(max\_iter=1000)

}

# Обучение и оценка моделей

results = {}

for name, model in models.items():

    model.fit(X\_train, y\_train)

    y\_pred = model.predict(X\_test)

    accuracy = accuracy\_score(y\_test, y\_pred)

    roc\_auc = roc\_auc\_score(

        y\_test, model.predict\_proba(X\_test), multi\_class='ovr'

    ) if hasattr(model, 'predict\_proba') else None

    results[name] = {

        'accuracy': accuracy,

        'confusion\_matrix': confusion\_matrix(y\_test, y\_pred),

        'roc\_auc': roc\_auc

    }

# Сравнение точности моделей

for name, metrics in results.items():

    roc\_auc\_str = f", ROC AUC = {metrics['roc\_auc']:.4f}" if metrics['roc\_auc'] is not None else ""

    print(f"{name}: Accuracy = {metrics['accuracy']:.4f}{roc\_auc\_str}")

# Визуализация матрицы ошибок

def plot\_confusion\_matrix(cm, title='Confusion Matrix'):

    plt.figure(figsize=(8, 6))

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

    plt.title(title)

    plt.ylabel('True label')

    plt.xlabel('Predicted label')

    plt.show()

# Пример визуализации матрицы ошибок для SVM

plot\_confusion\_matrix(results['SVM']['confusion\_matrix'], title='SVM Confusion Matrix')

# ROC-кривые

plt.figure(figsize=(10, 7))

for name, model in models.items():

    if hasattr(model, 'predict\_proba'):

        for i in range(len(wine.target\_names)):

            fpr, tpr, \_ = roc\_curve(y\_test == i, model.predict\_proba(X\_test)[:, i])

            plt.plot(fpr, tpr, label=f'{name} - Class {i}')

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

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC Curves')

plt.legend()

plt.show()

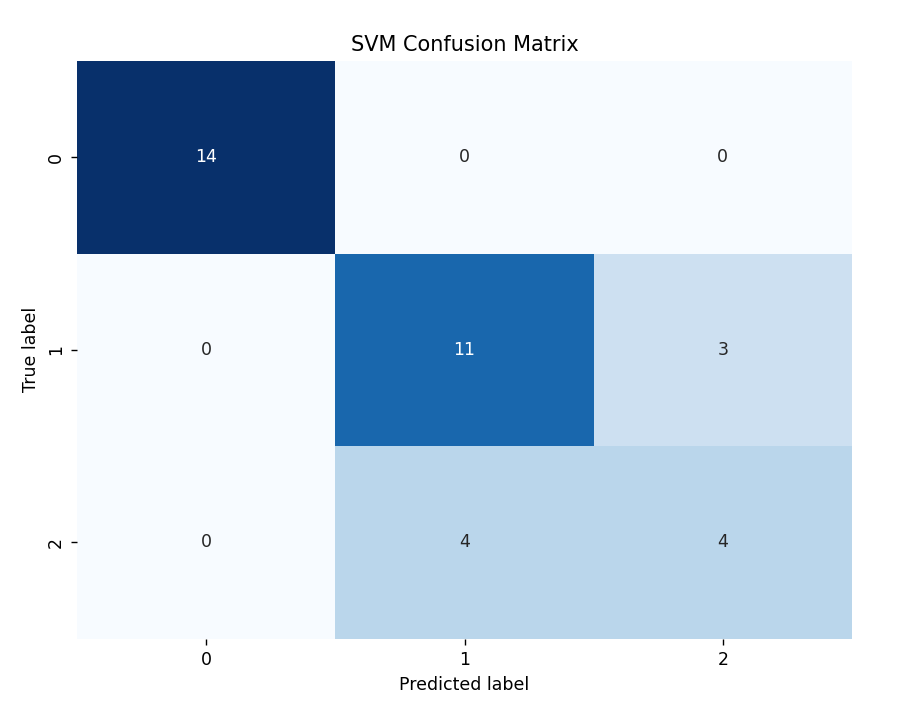
# Подбор гиперпараметров (пример для SVM)

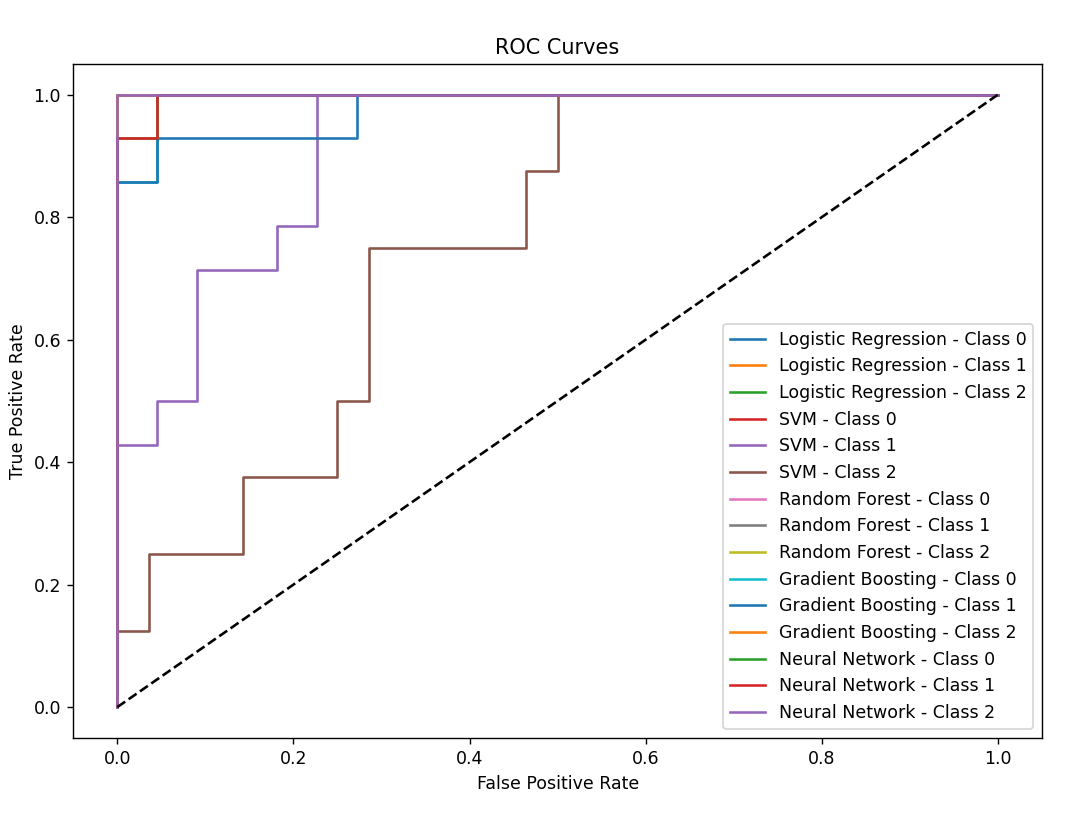
param\_grid = {'C': [0.1, 1, 10, 100], 'gamma': ['scale', 'auto']}

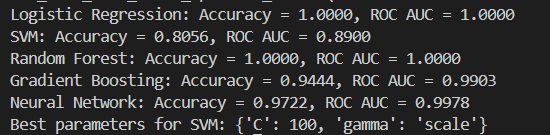
grid\_search = GridSearchCV(SVC(probability=True), param\_grid, cv=3, scoring='accuracy')

grid\_search.fit(X\_train, y\_train)

print(f"Best parameters for SVM: {grid\_search.best\_params\_}")







Задача 3. Классификация рака молочной железы

Набор данных: Breast Cancer Wisconsin (Diagnostic).

Предварительная обработка: масштабирование, разделение данных на обучающие и тестовые.

Модели: логистическая регрессия, SVM, решающие деревья, случайный лес, градиентный бустинг.

Сравнение моделей: точность, матрица ошибок, ROC-кривые.

Изучение и подбор гиперпараметров: GridSearchCV или RandomizedSearchCV.

Визуализация: важности признаков, корреляционная матрица.

import pandas as pd

from sklearn.datasets import load\_breast\_cancer

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

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import LogisticRegression

from sklearn.svm import SVC

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier

from sklearn.metrics import accuracy\_score, confusion\_matrix, roc\_curve, roc\_auc\_score

import matplotlib.pyplot as plt

import seaborn as sns

data = load\_breast\_cancer()

X = data.data

y = data.target

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

scaler = StandardScaler()

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

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

models = {

    'Logistic Regression': LogisticRegression(max\_iter=1000),

    'SVM': SVC(probability=True),

    'Decision Tree': DecisionTreeClassifier(),

    'Random Forest': RandomForestClassifier(),

    'Gradient Boosting': GradientBoostingClassifier()

}

accuracy = {}

roc\_auc = {}

for name, model in models.items():

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

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

    accuracy[name] = accuracy\_score(y\_test, y\_pred)

    roc\_auc[name] = roc\_auc\_score(y\_test, model.predict\_proba(X\_test\_scaled)[:, 1])

print("Точность моделей:")

for name, score in accuracy.items():

    print(f"{name}: {score:.4f}")

print("\nROC AUC для моделей:")

for name, score in roc\_auc.items():

    print(f"{name}: {score:.4f}")

for name, model in models.items():

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

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

    print(f"\nМатрица ошибок для {name}:")

    print(cm)

plt.figure(figsize=(10, 8))

for name, model in models.items():

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

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

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

plt.plot([0, 1], [0, 1], linestyle='--', color='gray')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC Curve')

plt.legend()

plt.show()

param\_grid\_lr = {'C': [0.1, 1, 10], 'solver': ['liblinear', 'saga']}

grid\_search\_lr = GridSearchCV(LogisticRegression(), param\_grid\_lr, cv=5)

grid\_search\_lr.fit(X\_train\_scaled, y\_train)

print(f"\nЛучшие гиперпараметры для логистической регрессии: {grid\_search\_lr.best\_params\_}")

rf\_model = RandomForestClassifier()

rf\_model.fit(X\_train\_scaled, y\_train)

importances = rf\_model.feature\_importances\_

indices = importances.argsort()

plt.figure(figsize=(10, 6))

plt.barh(range(len(importances)), importances[indices], align='center')

plt.yticks(range(len(importances)), [data.feature\_names[i] for i in indices])

plt.title('Важности признаков (Random Forest)')

plt.show()

correlation\_matrix = pd.DataFrame(X\_train\_scaled, columns=data.feature\_names).corr()

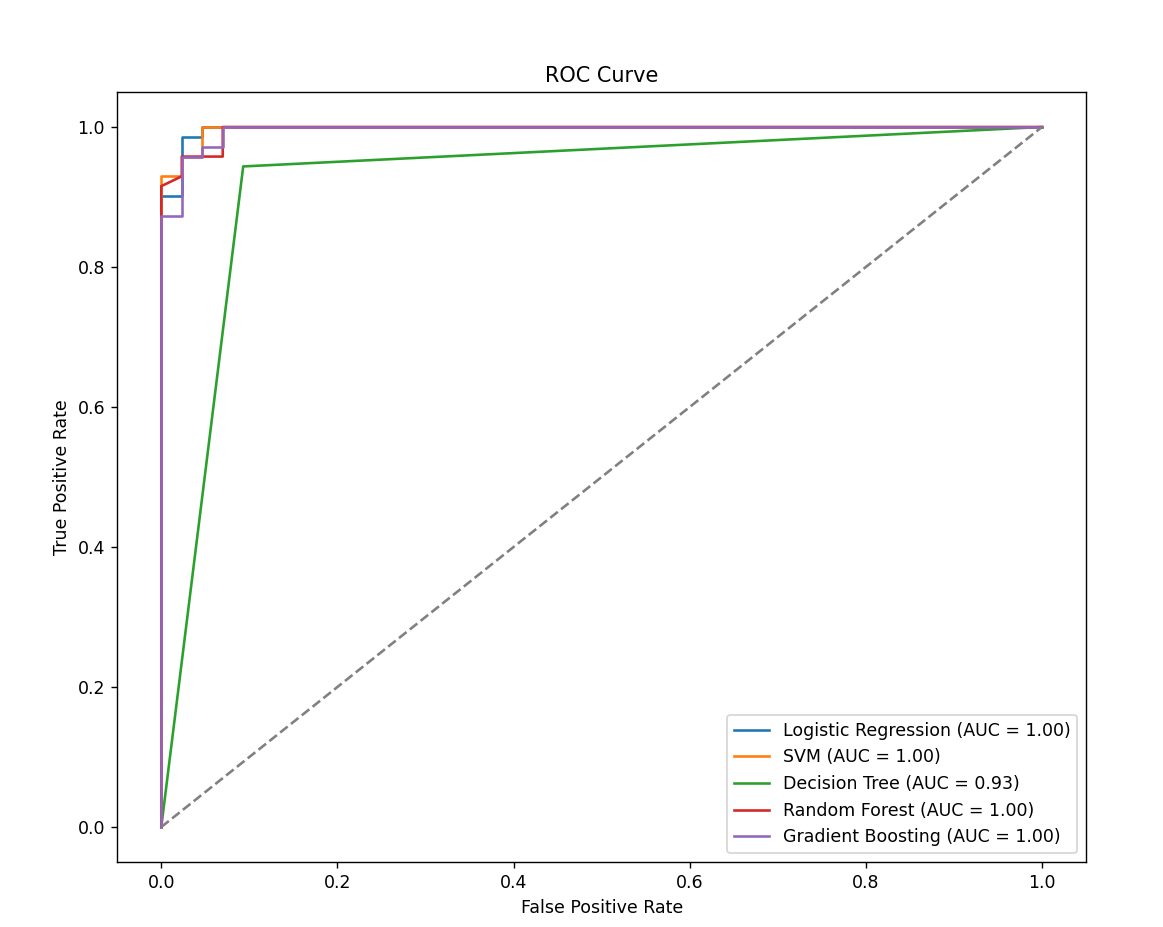
plt.figure(figsize=(12, 10))

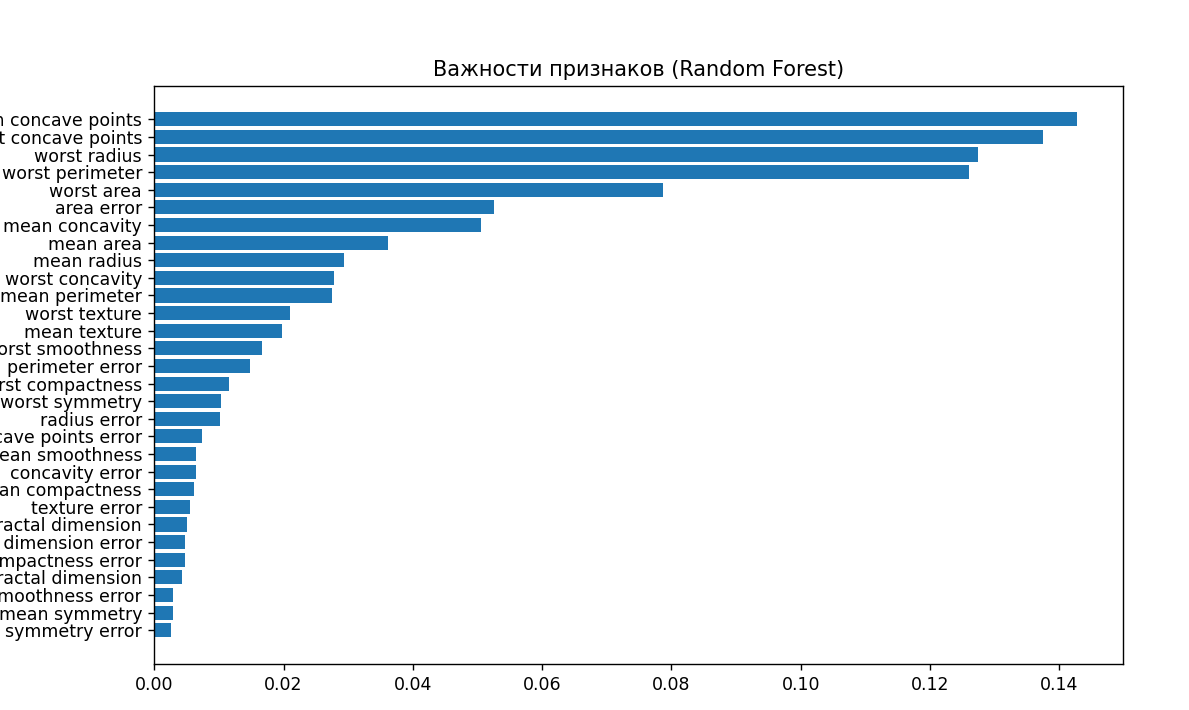
sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm', fmt='.2f')

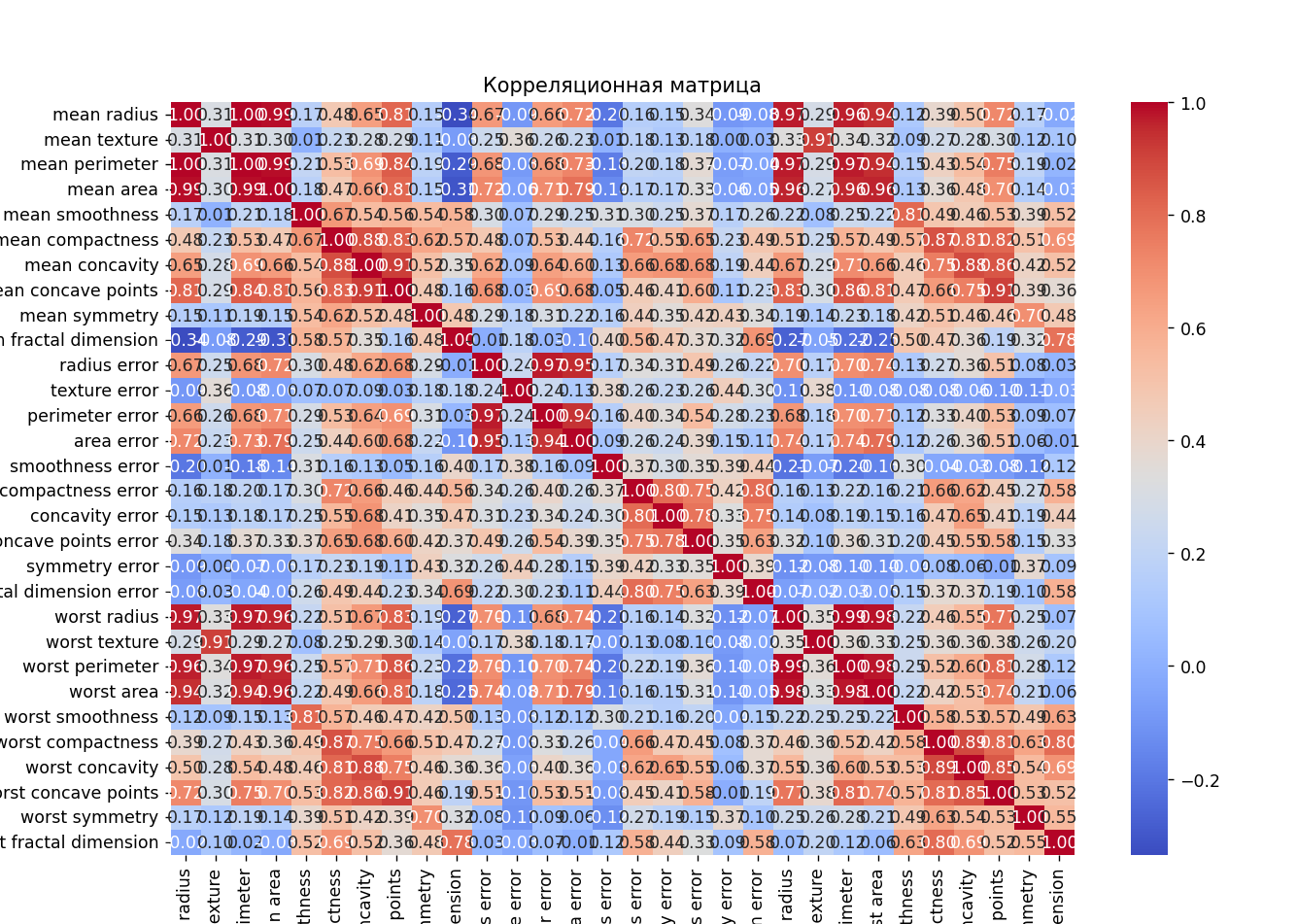
plt.title('Корреляционная матрица')

plt.show()









Задача 4. Спам-фильтрация для текстовых сообщений

Набор данных: SMS Spam Collection на Kaggle.

Предварительная обработка: векторизация текста, масштабирование, разделение данных на обучающие и тестовые.

Модели: логистическая регрессия, SVM, случайный лес, градиентный бустинг.

Сравнение моделей: точность, матрица ошибок, ROC-кривые.

Изучение и подбор гиперпараметров: GridSearchCV или RandomizedSearchCV.

Визуализация: важности признаков.

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split, GridSearchCV, RandomizedSearchCV

from sklearn.feature\_extraction.text import TfidfVectorizer

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

from sklearn.linear\_model import LogisticRegression

from sklearn.svm import SVC

from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier

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

print(df.head())

print(df.info())

df = df[['v1', 'v2']]

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

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

print(df['label'].value\_counts())

X = df['message']

y = df['label']

vectorizer = TfidfVectorizer(stop\_words='english', max\_features=3000)

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

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

log\_reg = LogisticRegression(max\_iter=1000, random\_state=42)

log\_reg.fit(X\_train, y\_train)

y\_pred\_log = log\_reg.predict(X\_test)

svm = SVC(kernel='linear', probability=True, random\_state=42)

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

y\_pred\_svm = svm.predict(X\_test)

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

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

y\_pred\_rf = rf.predict(X\_test)

gb = GradientBoostingClassifier(random\_state=42)

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

y\_pred\_gb = gb.predict(X\_test)

def evaluate\_model(y\_true, y\_pred, model\_name):

    print(f"=== {model\_name} ===")

    print(f"Accuracy: {accuracy\_score(y\_true, y\_pred):.4f}")

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

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

    plt.figure(figsize=(6, 4))

    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Ham', 'Spam'], yticklabels=['Ham', 'Spam'])

    plt.title(f"Confusion Matrix: {model\_name}")

    plt.xlabel('Predicted')

    plt.ylabel('True')

    plt.show()

evaluate\_model(y\_test, y\_pred\_log, "Logistic Regression")

evaluate\_model(y\_test, y\_pred\_svm, "SVM")

evaluate\_model(y\_test, y\_pred\_rf, "Random Forest")

evaluate\_model(y\_test, y\_pred\_gb, "Gradient Boosting")

def plot\_roc\_curves(models, X\_test, y\_test):

    plt.figure(figsize=(10, 8))

    for name, model in models.items():

        if hasattr(model, "predict\_proba"):

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

        else:

            y\_prob = model.decision\_function(X\_test)

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

        roc\_auc = auc(fpr, tpr)

        plt.plot(fpr, tpr, label=f"{name} (AUC = {roc\_auc:.2f})")

    plt.plot([0, 1], [0, 1], 'k--')

    plt.xlabel("FPR")

    plt.ylabel("TPR")

    plt.title("ROC-кривые")

    plt.legend()

    plt.show()

models = {

    "Logistic Regression": log\_reg,

    "SVM": svm,

    "Random Forest": rf,

    "Gradient Boosting": gb

}

plot\_roc\_curves(models, X\_test, y\_test)

param\_grid\_rf = {

    'n\_estimators': [50, 100, 200],

    'max\_depth': [None, 10, 20],

    'min\_samples\_split': [2, 5, 10]

}

grid\_rf = GridSearchCV(RandomForestClassifier(random\_state=42), param\_grid\_rf, cv=3)

grid\_rf.fit(X\_train, y\_train)

print("Лучшие параметры для Random Forest:", grid\_rf.best\_params\_)

param\_dist\_gb = {

    'n\_estimators': [50, 100, 200],

    'learning\_rate': [0.01, 0.1, 0.2],

    'max\_depth': [3, 5, 10]

}

random\_gb = RandomizedSearchCV(GradientBoostingClassifier(random\_state=42), param\_dist\_gb, n\_iter=10, cv=3, random\_state=42)

random\_gb.fit(X\_train, y\_train)

print("Лучшие параметры для Gradient Boosting:", random\_gb.best\_params\_)

importances = rf.feature\_importances\_

indices = np.argsort(importances)[::-1]

top\_features = 10

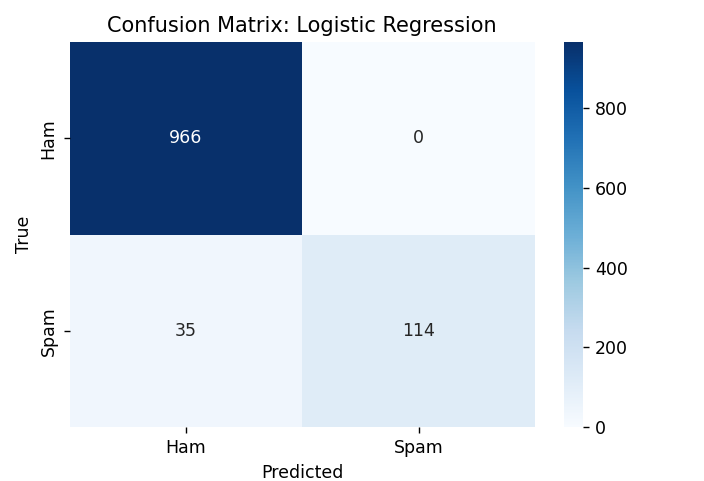
plt.figure(figsize=(12, 6))

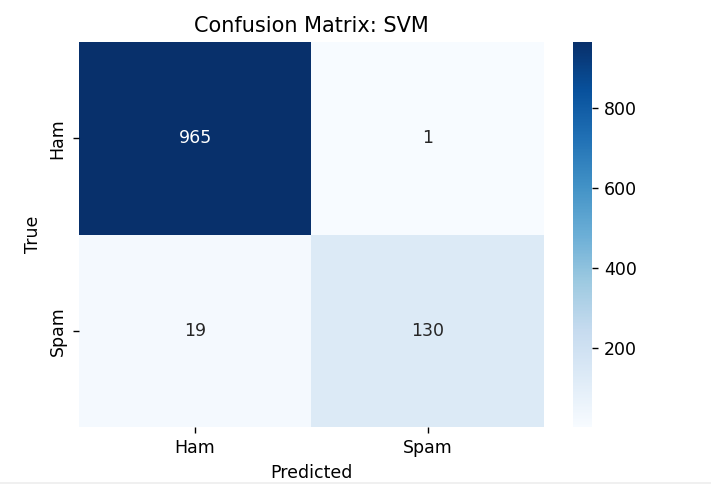
plt.bar(range(top\_features), importances[indices[:top\_features]])

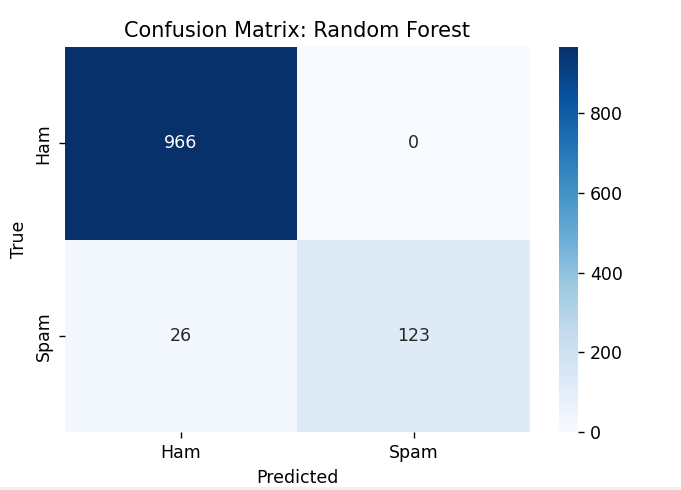
plt.xticks(range(top\_features), np.array(vectorizer.get\_feature\_names\_out())[indices[:top\_features]], rotation=90)

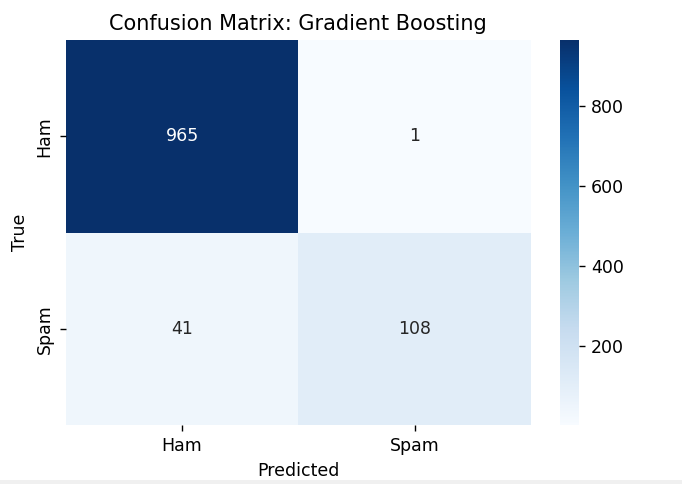
plt.title("Важность признаков (Random Forest)")

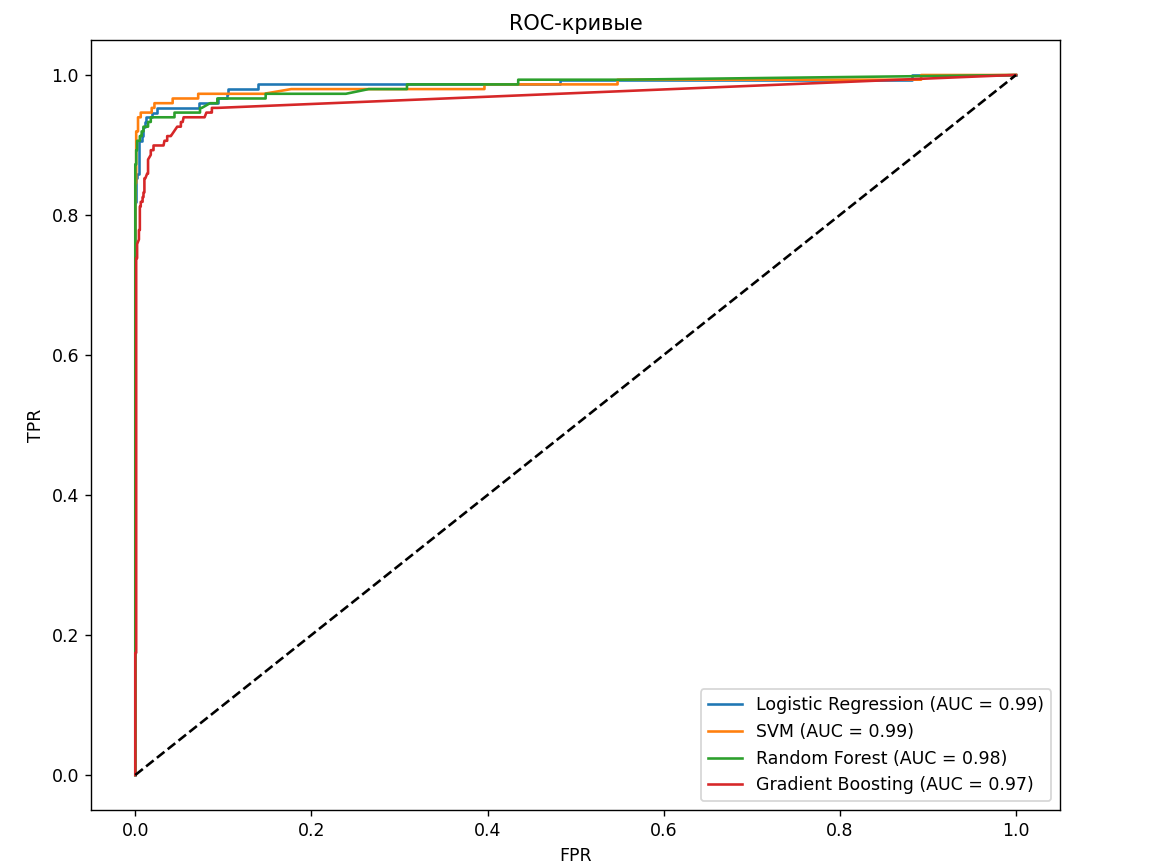
plt.show()











Задача 5. Классификация новостных статей

Набор данных: 20 Newsgroups dataset на sklearn.

Предварительная обработка: векторизация текста/TF-IDF, масштабирование, разделение данных на обучающие и тестовые.

Модели: логистическая регрессия, SVM, случайный лес, градиентный бустинг.

Сравнение моделей: точность, матрица ошибок.

Изучение и подбор гиперпараметров: GridSearchCV или RandomizedSearchCV.

Визуализация: важности признаков.

from sklearn.datasets import fetch\_20newsgroups

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

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.linear\_model import LogisticRegression

from sklearn.svm import SVC

from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier

from sklearn.metrics import accuracy\_score, confusion\_matrix

import seaborn as sns

import matplotlib.pyplot as plt

import numpy as np

categories = None

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

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

tfidf = TfidfVectorizer(max\_features=5000, stop\_words='english')

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

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

models = {

    "Logistic Regression": LogisticRegression(max\_iter=1000, random\_state=42),

    "SVM": SVC(kernel='linear', random\_state=42),

    "Random Forest": RandomForestClassifier(n\_estimators=100, random\_state=42),

    "Gradient Boosting": GradientBoostingClassifier(n\_estimators=100, random\_state=42)

}

results = {}

for model\_name, model in models.items():

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

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

    acc = accuracy\_score(y\_test, y\_pred)

    results[model\_name] = acc

    print(f"{model\_name} Accuracy: {acc:.4f}")

def plot\_confusion\_matrix(y\_true, y\_pred, title):

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

    plt.figure(figsize=(10, 8))

    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=data.target\_names, yticklabels=data.target\_names)

    plt.title(title)

    plt.xlabel("Predicted")

    plt.ylabel("Actual")

    plt.show()

best\_model\_name = max(results, key=results.get)

best\_model = models[best\_model\_name]

y\_pred\_best = best\_model.predict(X\_test\_tfidf)

plot\_confusion\_matrix(y\_test, y\_pred\_best, f"Confusion Matrix: {best\_model\_name}")

param\_grid = {

    'C': [0.1, 1, 10],

    'penalty': ['l2']

}

grid = GridSearchCV(LogisticRegression(max\_iter=1000, random\_state=42), param\_grid, cv=3, scoring='accuracy')

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

print(f"Best Parameters: {grid.best\_params\_}")

best\_log\_reg = grid.best\_estimator\_

if isinstance(best\_model, RandomForestClassifier):

    feature\_importances = best\_model.feature\_importances\_

    indices = np.argsort(feature\_importances)[-20:]

    feature\_names = np.array(tfidf.get\_feature\_names\_out())

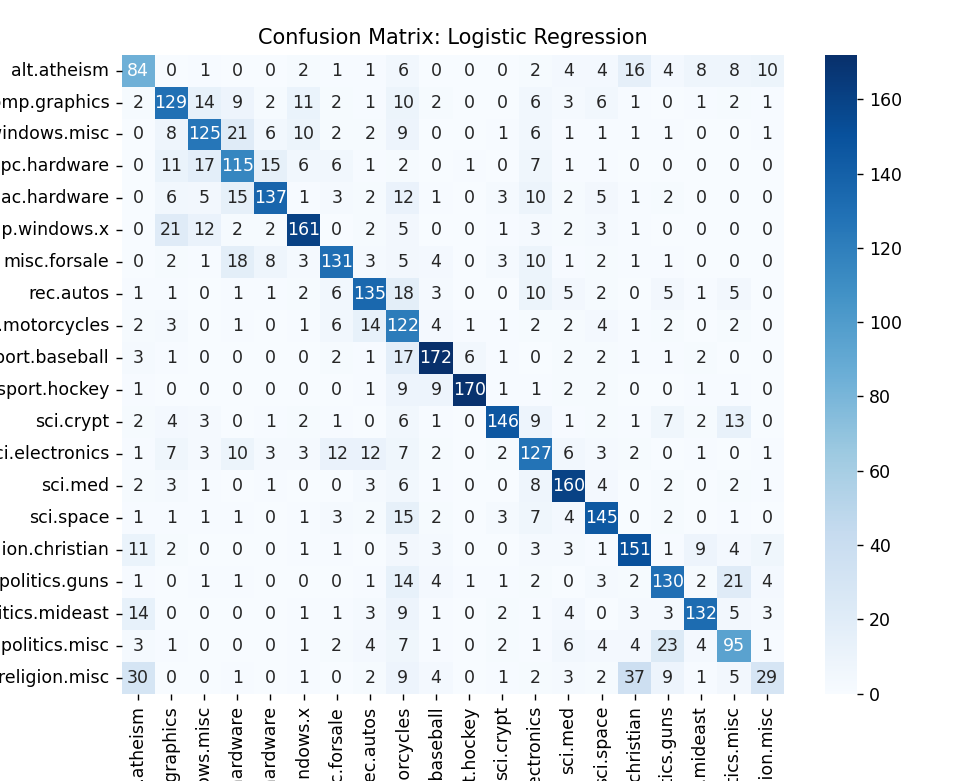
    plt.figure(figsize=(10, 8))

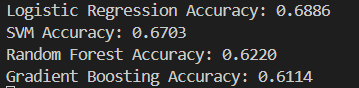
    plt.barh(range(len(indices)), feature\_importances[indices], align='center')

    plt.yticks(range(len(indices)), feature\_names[indices])

    plt.title("Feature Importances")

    plt.show()







Задача 6. Определение типа стекла

Набор данных: Glass Identification Dataset на UCI Machine Learning Repository.

Предварительная обработка: масштабирование, разделение данных на обучающие и тестовые.

Модели: логистическая регрессия, SVM, k-ближайших соседей, случайный лес, градиентный бустинг.

Сравнение моделей: точность, матрица ошибок.

Изучение и подбор гиперпараметров: GridSearchCV или RandomizedSearchCV.

Визуализация: важности признаков, корреляционная матрица.

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split, GridSearchCV, RandomizedSearchCV

from sklearn.preprocessing import StandardScaler

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

from sklearn.linear\_model import LogisticRegression

from sklearn.svm import SVC

from sklearn.neighbors import KNeighborsClassifier

from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/glass/glass.data"

column\_names = [

    "Id", "RI", "Na", "Mg", "Al", "Si", "K", "Ca", "Ba", "Fe", "Type"

]

df = pd.read\_csv(url, names=column\_names, index\_col="Id")

X = df.drop(columns=["Type"])

y = df["Type"]

scaler = StandardScaler()

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

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

log\_reg = LogisticRegression(max\_iter=1000, random\_state=42)

log\_reg.fit(X\_train, y\_train)

y\_pred\_log = log\_reg.predict(X\_test)

svm = SVC(kernel='linear', probability=True, random\_state=42)

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

y\_pred\_svm = svm.predict(X\_test)

knn = KNeighborsClassifier(n\_neighbors=5)

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

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

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

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

y\_pred\_rf = rf.predict(X\_test)

gb = GradientBoostingClassifier(random\_state=42)

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

y\_pred\_gb = gb.predict(X\_test)

def evaluate\_model(y\_true, y\_pred, model\_name):

    print(f"=== {model\_name} ===")

    print(f"Accuracy: {accuracy\_score(y\_true, y\_pred):.4f}")

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

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

    plt.figure(figsize=(6, 4))

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

    plt.title(f"Confusion Matrix: {model\_name}")

    plt.xlabel('Predicted')

    plt.ylabel('True')

    plt.show()

evaluate\_model(y\_test, y\_pred\_log, "Logistic Regression")

evaluate\_model(y\_test, y\_pred\_svm, "SVM")

evaluate\_model(y\_test, y\_pred\_knn, "KNN")

evaluate\_model(y\_test, y\_pred\_rf, "Random Forest")

evaluate\_model(y\_test, y\_pred\_gb, "Gradient Boosting")

param\_grid\_svm = {

    'C': [0.1, 1, 10],

    'kernel': ['linear', 'rbf', 'poly'],

    'gamma': ['scale', 'auto']

}

grid\_svm = GridSearchCV(SVC(probability=True, random\_state=42), param\_grid\_svm, cv=3)

grid\_svm.fit(X\_train, y\_train)

print("Лучшие параметры для SVM:", grid\_svm.best\_params\_)

param\_dist\_rf = {

    'n\_estimators': [50, 100, 200],

    'max\_depth': [None, 10, 20],

    'min\_samples\_split': [2, 5, 10]

}

random\_rf = RandomizedSearchCV(RandomForestClassifier(random\_state=42), param\_dist\_rf, n\_iter=10, cv=3, random\_state=42)

random\_rf.fit(X\_train, y\_train)

print("Лучшие параметры для Random Forest:", random\_rf.best\_params\_)

plt.figure(figsize=(10, 8))

sns.heatmap(df.corr(), annot=True, cmap='coolwarm')

plt.title("Корреляционная матрица признаков")

plt.show()

importances = rf.feature\_importances\_

indices = np.argsort(importances)[::-1]

features = X.columns

