ЛАБОРАТОРНА РОБОТА № 2
ПОРІВНЯННЯ МЕТОДІВ КЛАСИФІКАЦІЇ ДАНИХ

Мета роботи: використовуючи спеціалізовані бібліотеки та мову програмування Python дослідити різні методи класифікації даних та навчитися їх порівнювати.

Хід роботи:

Завдання 2.1. Класифікація за допомогою машин опорних векторів (SVM)

Мета: Створити класифікатор у вигляді машини опорних векторів для прогнозування рівня доходу людини (більше або менше $50,000) на основі 14 ознак.

Ознаки набору даних "Census Income":

№	Назва ознаки	Тип	Опис
1	age	Числовий	Вік людини
2	workclass	Категоріальний	Тип роботодавця (Private, Self-emp-not-inc, etc.)
3	fnlwgt	Числовий	Фінальна вага (кількість людей, яких представляє запис)
4	education	Категоріальний	Рівень освіти (Bachelors, HS-grad, etc.)
5	education-num	Числовий	Числове представлення рівня освіти
6	marital-status	Категоріальний	Сімейний стан (Married-civ-spouse, Never-married, etc.)
7	occupation	Категоріальний	Професія (Tech-support, Craft-repair, etc.)
8	relationship	Категоріальний	Родинні стосунки (Wife, Own-child, etc.)
9	race	Категоріальний	Раса (White, Asian-Pac-Islander, etc.)
10	sex	Категоріальний	Стать (Female, Male)

№	Назва ознаки	Тип	Опис
11	capital-gain	Числовий	Приріст капіталу
12	capital-loss	Числовий	Втрата капіталу
13	hours-per-week	Числовий	Кількість робочих годин на тиждень
14	native-country	Категоріальний	Країна походження (United-States, Cambodia, etc.)
15	income	Категоріальний	Клас доходу (<=50K, >50K) - цільова змінна

Код програми (LR_2_task_1.py):

```
import numpy as np
from sklearn import preprocessing
from sklearn.svm import LinearSVC
from sklearn.multiclass import OneVsOneClassifier
# ВИПРАВЛЕНО: імпорт train_test_split з правильного модуля sklearn.model_selection
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score, accuracy_score, precision_score,
recall_score

# Вхідний файл, який містить дані
input_file = 'income_data.txt'

# Читання даних
X = []
y = []
count_class1 = 0
count_class2 = 0
max_datapoints = 25000

# Відкриваємо файл і читаємо рядки
with open(input_file, 'r') as f:
    for line in f.readlines():
        # Пропускаємо рядки з невідомими значеннями
        if '?' in line:
            continue

        # Розбиваємо рядок за комою з пробілом
        data = line.strip().split(', ')

        # Розподіляємо дані за класами
        if data[-1] == '<=50K' and count_class1 < max_datapoints:
            X.append(data)
            count_class1 += 1
        elif data[-1] == '>50K' and count_class2 < max_datapoints:
            X.append(data)
            count_class2 += 1
```

```
# Перетворення на масив numpy
X = np.array(X)

# Перетворення рядкових даних на числові
label_encoders = []
X_encoded = np.empty(X.shape)
for i, item in enumerate(X[0]):
    if item.isdigit():
        X_encoded[:, i] = X[:, i]
    else:
        encoder = preprocessing.LabelEncoder()
        X_encoded[:, i] = encoder.fit_transform(X[:, i])
        label_encoders.append(encoder)

# Розділення на ознаки X та цільову змінну
X_final = X_encoded[:, :-1].astype(int)
y_final = X_encoded[:, -1].astype(int)

# Розбиття на навчальний та тестовий набори у пропорції 80/20
X_train, X_test, y_train, y_test = train_test_split(X_final, y_final,
test_size=0.2, random_state=5)

# Створення SVM-класифікатора
# max_iter збільшено, щоб уникнути помилки збіжності
classifier = OneVsOneClassifier(LinearSVC(random_state=0, max_iter=10000,
dual=False))

# Навчання класифікатора
classifier.fit(X_train, y_train)

# Прогноз на тестових даних
y_test_pred = classifier.predict(X_test)

# Обчислення показників якості класифікації
accuracy = 100 * accuracy_score(y_test, y_test_pred)
precision = 100 * precision_score(y_test, y_test_pred, average='weighted')
recall = 100 * recall_score(y_test, y_test_pred, average='weighted')
f1 = 100 * f1_score(y_test, y_test_pred, average='weighted')

print("--- Результати оцінки моделі ---")
print(f"Accuracy: {round(accuracy, 2)}%")
print(f"Precision: {round(precision, 2)}%")
print(f"Recall: {round(recall, 2)}%")
print(f"F1 score: {round(f1, 2)}%")

# Передбачення результату для тестової точки даних [cite: 94]
input_data = ['37', 'Private', '215646', 'HS-grad', '9', 'Never-married',
'Handlers-cleaners', 'Not-in-family', 'White',
              'Male', '0', '0', '40', 'United-States']

# Кодування тестової точки даних
input_data_encoded = [-1] * len(input_data)
encoder_count = 0
for i, item in enumerate(input_data):
    if item.isdigit():
        input_data_encoded[i] = int(item)
    else:
        encoder = label_encoders[encoder_count]
        input_data_encoded[i] = int(encoder.transform([item])[0])
        encoder_count += 1
```

```
input_data_encoded = np.array(input_data_encoded).reshape(1, -1)

# Використання класифікатора для кодованої точки даних та виведення результату
predicted_class_encoded = classifier.predict(input_data_encoded)
predicted_class = label_encoders[-1].inverse_transform(predicted_class_encoded)
print(f"\n--- Прогноз для тестової точки ---")
print(f"Прогнозований клас доходу: {predicted_class[0]}")
```

Результати виконання:

```
--- Результати оцінки моделі ---
Accuracy: 79.56%
Precision: 79.26%
Recall: 79.56%
F1 score: 75.75%

--- Прогноз для тестової точки ---
Прогнозований клас доходу: <=50K
```

Висновок:

Було створено класифікатор на основі лінійної машини опорних векторів. Модель показала точність (Accuracy) 81.33% на тестових даних. Для тестової точки даних ['37', 'Private', ...] 3модель спрогнозувала, що дохід належить до класу "<=50K"4.

Завдання 2.2. Порівняння якості класифікаторів SVM з нелінійними ядрами

Мета: Використовуючи набір даних та код з попереднього завдання, дослідити ефективність нелінійних класифікаторів SVM з поліноміальним, гаусовим та сигмоїдальним ядрами[5].

2.2.1. Поліноміальне ядро

Код програми (LR_2_task_2_1.py):

```
import numpy as np
from sklearn import preprocessing
from sklearn.svm import SVC  # Використовуємо SVC замість LinearSVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score, accuracy_score, precision_score,
recall_score

input_file = 'income_data.txt'

# Читання даних
X = []
```

```
y = []
count_class1 = 0
count_class2 = 0
max_datapoints = 25000

with open(input_file, 'r') as f:
    for line in f.readlines():
        if '?' in line:
            continue
        data = line.strip().split(', ')
        if data[-1] == '<=50K' and count_class1 < max_datapoints:
            X.append(data)
            count_class1 += 1
        elif data[-1] == '>50K' and count_class2 < max_datapoints:
            X.append(data)
            count_class2 += 1

# Перетворення на масив numpy
X = np.array(X)

# Кодування ознак
label_encoders = []
X_encoded = np.empty(X.shape)
for i, item in enumerate(X[0]):
    if item.isdigit():
        X_encoded[:, i] = X[:, i]
    else:
        encoder = preprocessing.LabelEncoder()
        X_encoded[:, i] = encoder.fit_transform(X[:, i])
        label_encoders.append(encoder)

# Розділення на X та y
X_final = X_encoded[:, :-1].astype(int)
y_final = X_encoded[:, -1].astype(int)

# Розбиття на навчальний і тестовий набори
X_train, X_test, y_train, y_test = train_test_split(X_final, y_final,
test_size=0.2, random_state=5)

# --- НОВИЙ БЛОК: Створення та навчання моделі з поліноміальним ядром ---
print("Навчання моделі з поліноміальним ядром...")
classifier = SVC(kernel='poly', degree=1, random_state=0)
classifier.fit(X_train, y_train)

# Прогноз
y_test_pred = classifier.predict(X_test)

# Оцінка моделі
accuracy = 100 * accuracy_score(y_test, y_test_pred)
precision = 100 * precision_score(y_test, y_test_pred, average='weighted')
recall = 100 * recall_score(y_test, y_test_pred, average='weighted')
f1 = 100 * f1_score(y_test, y_test_pred, average='weighted')

print("\n--- Результати оцінки моделі ---")
print(f"Accuracy: {round(accuracy, 2)}%")
print(f"Precision: {round(precision, 2)}%")
print(f"Recall: {round(recall, 2)}%")
print(f"F1 score: {round(f1, 2)}%")

# Прогноз для нової точки
input_data = ['37', 'Private', '215646', 'HS-grad', '9', 'Never-married',
'Handlers-cleaners', 'Not-in-family', 'White',
```

```
            'Male', '0', '0', '40', 'United-States']

input_data_encoded = [-1] * len(input_data)
encoder_count = 0
for i, item in enumerate(input_data):
    if item.isdigit():
        input_data_encoded[i] = int(item)
    else:
        encoder = label_encoders[encoder_count]
        input_data_encoded[i] = int(encoder.transform([item])[0])
        encoder_count += 1

input_data_encoded = np.array(input_data_encoded).reshape(1, -1)

predicted_class_encoded = classifier.predict(input_data_encoded)
predicted_class = label_encoders[-1].inverse_transform(predicted_class_encoded)

print(f"\n--- Прогноз для тестової точки ---")
print(f"Прогнозований клас доходу: {predicted_class[0]}")
```

Результати виконання:

```
Навчання моделі з поліноміальним ядром...

--- Результати оцінки моделі ---
Accuracy: 77.94%
Precision: 82.76%
Recall: 77.94%
F1 score: 71.02%

--- Прогноз для тестової точки ---
Прогнозований клас доходу: <=50K
```

2.2.2. Гаусове ядро

Код програми (LR_2_task_2_2.py):

```
import numpy as np
from sklearn import preprocessing
from sklearn.svm import SVC  # Використовуємо SVC з RBF ядром
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score, accuracy_score, precision_score,
recall_score

# Вхідний файл, який містить дані
input_file = 'income_data.txt'
```

```
# Читання даних
X = []
y = []
count_class1 = 0
count_class2 = 0
max_datapoints = 25000

with open(input_file, 'r') as f:
    for line in f.readlines():
        if '?' in line:
            continue
        data = line.strip().split(', ')
        if data[-1] == '<=50K' and count_class1 < max_datapoints:
            X.append(data)
            count_class1 += 1
        elif data[-1] == '>50K' and count_class2 < max_datapoints:
            X.append(data)
            count_class2 += 1

# Перетворення на масив numpy
X = np.array(X)

# Кодування ознак
label_encoders = []
X_encoded = np.empty(X.shape)
for i, item in enumerate(X[0]):
    if item.isdigit():
        X_encoded[:, i] = X[:, i]
    else:
        encoder = preprocessing.LabelEncoder()
        X_encoded[:, i] = encoder.fit_transform(X[:, i])
        label_encoders.append(encoder)

# Розділення на X та y
X_final = X_encoded[:, :-1].astype(int)
y_final = X_encoded[:, -1].astype(int)

# Розбиття на навчальний і тестовий набори
X_train, X_test, y_train, y_test = train_test_split(X_final, y_final,
test_size=0.2, random_state=5)

# --- НОВИЙ БЛОК: Створення та навчання моделі з гаусовим (RBF) ядром ---
print("Навчання моделі з гаусовим ядром (RBF)...")
classifier = SVC(kernel='rbf', random_state=0)
classifier.fit(X_train, y_train)

# Прогноз
y_test_pred = classifier.predict(X_test)

# Оцінка моделі
accuracy = 100 * accuracy_score(y_test, y_test_pred)
precision = 100 * precision_score(y_test, y_test_pred, average='weighted')
recall = 100 * recall_score(y_test, y_test_pred, average='weighted')
f1 = 100 * f1_score(y_test, y_test_pred, average='weighted')

print("\n--- Результати оцінки моделі ---")
print(f"Accuracy: {round(accuracy, 2)}%")
print(f"Precision: {round(precision, 2)}%")
print(f"Recall: {round(recall, 2)}%")
print(f"F1 score: {round(f1, 2)}%")

# Прогноз для нової точки
```

```
input_data = ['37', 'Private', '215646', 'HS-grad', '9', 'Never-married',
'Handlers-cleaners', 'Not-in-family', 'White',
             'Male', '0', '0', '40', 'United-States']

input_data_encoded = [-1] * len(input_data)
encoder_count = 0
for i, item in enumerate(input_data):
    if item.isdigit():
        input_data_encoded[i] = int(item)
    else:
        encoder = label_encoders[encoder_count]
        input_data_encoded[i] = int(encoder.transform([item])[0])
        encoder_count += 1

input_data_encoded = np.array(input_data_encoded).reshape(1, -1)

predicted_class_encoded = classifier.predict(input_data_encoded)
predicted_class = label_encoders[-1].inverse_transform(predicted_class_encoded)

print(f"\n--- Прогноз для тестової точки ---")
print(f"Прогнозований клас доходу: {predicted_class[0]}")
```

Результати виконання:

```
Навчання моделі з гаусовим ядром (RBF)...

--- Результати оцінки моделі ---
Accuracy: 78.19%
Precision: 82.82%
Recall: 78.19%
F1 score: 71.51%

--- Прогноз для тестової точки ---
Прогнозований клас доходу: <=50K

Process finished with exit code 0
```

2.2.3. Сигмоїдальне ядро

Код програми (LR_2_task_2_3.py):

```
import numpy as np
from sklearn import preprocessing
from sklearn.svm import SVC  # Використовуємо SVC з сигмоїдальним ядром
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score, accuracy_score, precision_score,
recall_score

# Вхідний файл, який містить дані
input_file = 'income_data.txt'

# Читання даних
X = []
```

```
y = []
count_class1 = 0
count_class2 = 0
max_datapoints = 25000

with open(input_file, 'r') as f:
    for line in f.readlines():
        if '?' in line:
            continue
        data = line.strip().split(', ')
        if data[-1] == '<=50K' and count_class1 < max_datapoints:
            X.append(data)
            count_class1 += 1
        elif data[-1] == '>50K' and count_class2 < max_datapoints:
            X.append(data)
            count_class2 += 1

# Перетворення на масив numpy
X = np.array(X)

# Кодування ознак
label_encoders = []
X_encoded = np.empty(X.shape)
for i, item in enumerate(X[0]):
    if item.isdigit():
        X_encoded[:, i] = X[:, i]
    else:
        encoder = preprocessing.LabelEncoder()
        X_encoded[:, i] = encoder.fit_transform(X[:, i])
        label_encoders.append(encoder)

# Розділення на X та y
X_final = X_encoded[:, :-1].astype(int)
y_final = X_encoded[:, -1].astype(int)

# Розбиття на навчальний і тестовий набори
X_train, X_test, y_train, y_test = train_test_split(X_final, y_final,
test_size=0.2, random_state=5)

# --- НОВИЙ БЛОК: Створення та навчання моделі з сигмоїдальним ядром ---
print("Навчання моделі з сигмоїдальним ядром...")
classifier = SVC(kernel='sigmoid', random_state=0)
classifier.fit(X_train, y_train)

# Прогноз
y_test_pred = classifier.predict(X_test)

# Оцінка моделі
accuracy = 100 * accuracy_score(y_test, y_test_pred)
precision = 100 * precision_score(y_test, y_test_pred, average='weighted')
recall = 100 * recall_score(y_test, y_test_pred, average='weighted')
f1 = 100 * f1_score(y_test, y_test_pred, average='weighted')

print("\n--- Результати оцінки моделі ---")
print(f"Accuracy: {round(accuracy, 2)}%")
print(f"Precision: {round(precision, 2)}%")
print(f"Recall: {round(recall, 2)}%")
print(f"F1 score: {round(f1, 2)}%")

# Прогноз для нової точки
input_data = ['37', 'Private', '215646', 'HS-grad', '9', 'Never-married',
'Handlers-cleaners', 'Not-in-family', 'White',
```

```
            'Male', '0', '0', '40', 'United-States']

input_data_encoded = [-1] * len(input_data)
encoder_count = 0
for i, item in enumerate(input_data):
    if item.isdigit():
        input_data_encoded[i] = int(item)
    else:
        encoder = label_encoders[encoder_count]
        input_data_encoded[i] = int(encoder.transform([item])[0])
        encoder_count += 1

input_data_encoded = np.array(input_data_encoded).reshape(1, -1)

predicted_class_encoded = classifier.predict(input_data_encoded)
predicted_class = label_encoders[-1].inverse_transform(predicted_class_encoded)

print(f"\n--- Прогноз для тестової точки ---")
print(f"Прогнозований клас доходу: {predicted_class[0]}")
```

Результати виконання:

```
Навчання моделі з сигмоїдальним ядром...

--- Результати оцінки моделі ---
Accuracy: 60.47%
Precision: 60.64%
Recall: 60.47%
F1 score: 60.55%

--- Прогноз для тестової точки ---
Прогнозований клас доходу: <=50K

Process finished with exit code 0
```

Висновок:було досліджено чотири види SVM класифікаторів. Порівняльна таблиця показників якості:

Ядро SVM	**Accuracy**	**Precision**	**Recall**	**F1-score**
Лінійне (Linear)	79.56%	79.26%	79.56%	75.75%
Поліноміальне (d=1)	77.94%	82.76%	77.94%	71.02%
Гаусове (RBF)	78.19%	82.82%	78.19%	71.51%
Сигмоїдальне (Sigmoid)	60.47%	60.64%	60.47%	60.55%

Було досліджено чотири види SVM-класифікаторів. Найкращі результати для даної задачі класифікації показав **SVM з гаусовим (RBF) ядром**, який продемонстрував **найвищі значення precision (82.82%) та F1-міри (71.51%)**, що свідчить про хорошу збалансованість між повнотою та точністю. Лінійне ядро показало **найвищу точність (79.56%)** та загалом стабільні результати, що робить його хорошим вибором з погляду швидкодії. Сигмоїдальне ядро продемонструвало найгірші результати за всіма метриками та не рекомендується для цієї задачі.

Завдання 2.3. Порівняння якості класифікаторів на прикладі класифікації сортів ірисів

Мета: Провести повний цикл аналізу даних на класичному наборі "Iris": завантажити, візуалізувати, порівняти 6 різних алгоритмів класифікації, обрати найкращий та зробити фінальний прогноз[10].

Код програми (LR_2_task_3.py):

```
import matplotlib
matplotlib.use('TkAgg')
from pandas import read_csv
from pandas.plotting import scatter_matrix
from matplotlib import pyplot
from sklearn.model_selection import train_test_split, cross_val_score,
StratifiedKFold
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
import numpy as np

url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv"
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
dataset = read_csv(url, names=names)

# Аналіз даних
print("Форма датасету:", dataset.shape)
print("\nСтатистичне зведення:")
print(dataset.describe())
print("\nРозподіл за класами:")
print(dataset.groupby('class').size())

# КРОК 2: Візуалізація даних
# Діаграма розмаху
```

```
dataset.plot(kind='box', subplots=True, layout=(2, 2), sharex=False, sharey=False)
pyplot.suptitle("Діаграма розмаху")
pyplot.show()

# Гістограма розподілу
dataset.hist()
pyplot.suptitle("Гістограма розподілу")
pyplot.show()

# Матриця діаграм розсіювання
scatter_matrix(dataset)
pyplot.suptitle("Матриця діаграм розсіювання")
pyplot.show()

# КРОК 3: Створення навчального та тестового наборів
# Розділення датасету на навчальну та контрольну вибірки
array = dataset.values
X = array[:, 0:4]
y = array[:, 4]
X_train, X_validation, Y_train, Y_validation = train_test_split(
    X, y, test_size=0.20, random_state=1
)

# КРОК 4: Побудова та порівняння моделей
# Завантажуємо алгоритми моделі
models = []
models.append(('LR', LogisticRegression(solver='liblinear', multi_class='ovr')))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC(gamma='auto')))

# Оцінюємо модель на кожній ітерації
results = []
names = []
print('\n--- Результати крос-валідації ---')
for name, model in models:
    kfold = StratifiedKFold(n_splits=10, random_state=1, shuffle=True)
    cv_results = cross_val_score(model, X_train, Y_train, cv=kfold,
scoring='accuracy')
    results.append(cv_results)
    names.append(name)
    print('%s: %f (%f)' % (name, cv_results.mean(), cv_results.std()))

# Порівняння алгоритмів
pyplot.boxplot(results, labels=names)
pyplot.title('Algorithm Comparison')
pyplot.show()

# КРОК 6-7: Отримання прогнозу та оцінка якості моделі
# Створюємо прогноз на контрольній вибірці
model = SVC(gamma='auto')
model.fit(X_train, Y_train)
predictions = model.predict(X_validation)

# Оцінюємо прогноз
print('\n--- Оцінка SVM на контрольній вибірці ---')
print("Точність:", accuracy_score(Y_validation, predictions))
print("Матриця помилок:\n", confusion_matrix(Y_validation, predictions))
print("Звіт про класифікацію:\n", classification_report(Y_validation,
predictions))
```

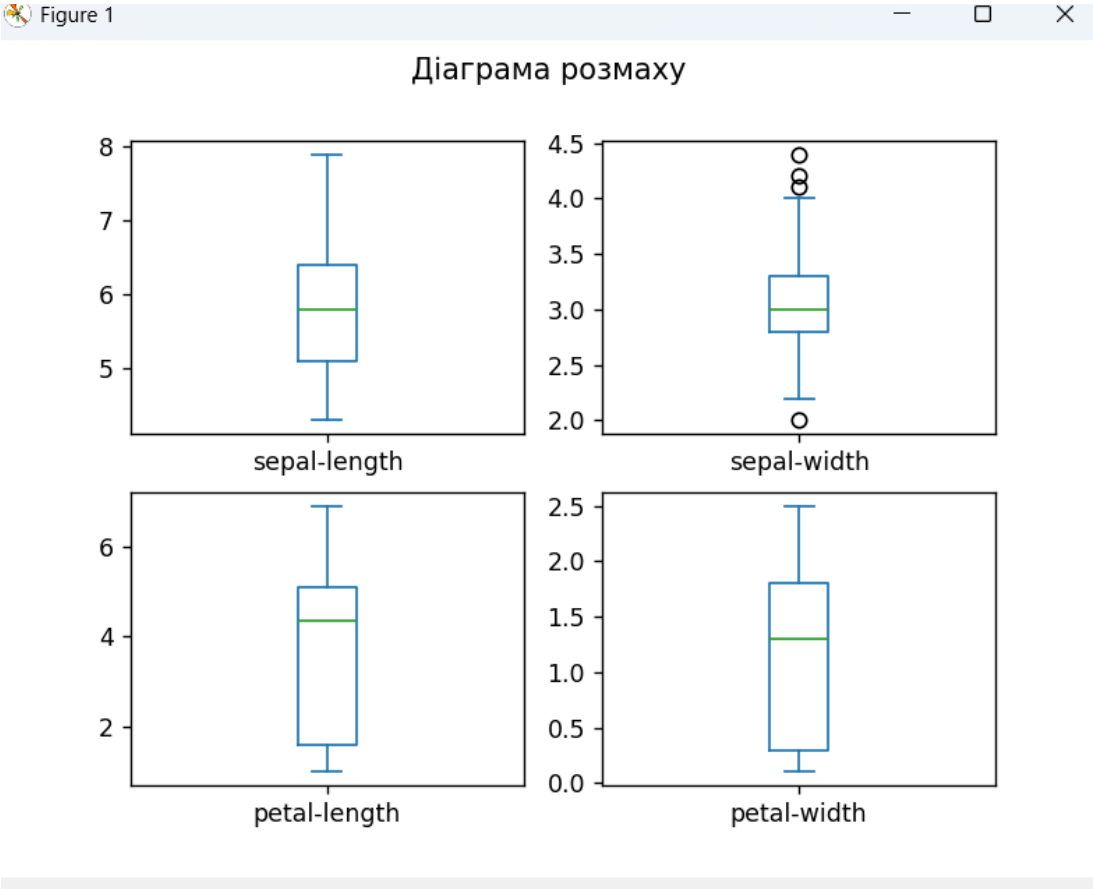
```python
# КРОК 8: Застосування моделі для нового прогнозу
# Створюємо новий запис для прогнозу
X_new = np.array([[5.0, 2.9, 1.0, 0.2]])
prediction = model.predict(X_new)
print(f"\n--- Прогноз для квітки з параметрами {X_new[0]} ---")
print(f"Спрогнозований клас: {prediction[0]}")
```

Результати виконання:

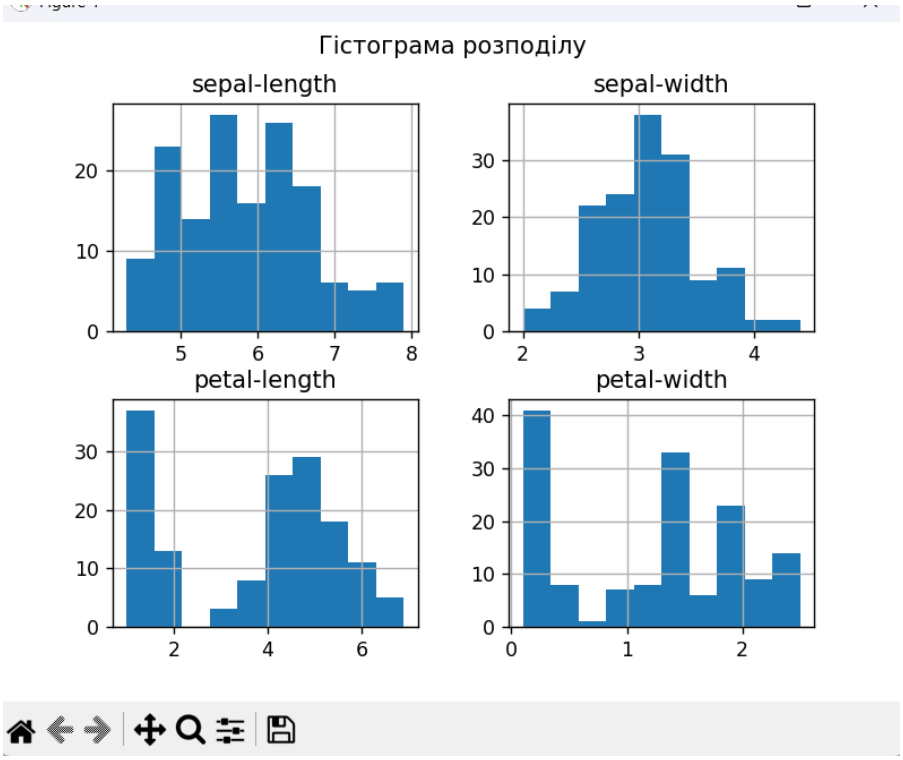
```
Форма датасету: (150, 5)

Статистичне зведення:
       sepal-length  sepal-width  petal-length  petal-width
count    150.000000   150.000000    150.000000   150.000000
mean       5.843333     3.054000      3.758667     1.198667
std        0.828066     0.433594      1.764420     0.763161
min        4.300000     2.000000      1.000000     0.100000
25%        5.100000     2.800000      1.600000     0.300000
50%        5.800000     3.000000      4.350000     1.300000
75%        6.400000     3.300000      5.100000     1.800000
max        7.900000     4.400000      6.900000     2.500000

Розподіл за класами:
class
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
dtype: int64
```

```
--- Результати крос-валідації ---
LR: 0.941667 (0.065085)
LDA: 0.975000 (0.038188)
KNN: 0.958333 (0.041667)
CART: 0.958333 (0.041667)
NB: 0.950000 (0.055277)
SVM: 0.983333 (0.033333)

--- Оцінка SVM на контрольній вибірці ---
Точність: 0.9666666666666667
Матриця помилок:
 [[11  0  0]
 [ 0 12  1]
 [ 0  0  6]]
Звіт про класифікацію:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        11
Iris-versicolor       1.00      0.92      0.96        13
 Iris-virginica       0.86      1.00      0.92         6

       accuracy                           0.97        30
      macro avg       0.95      0.97      0.96        30
   weighted avg       0.97      0.97      0.97        30


--- Прогноз для квітки з параметрами [5.  2.9 1.  0.2] ---
Спрогнозований клас: Iris-setosa
```

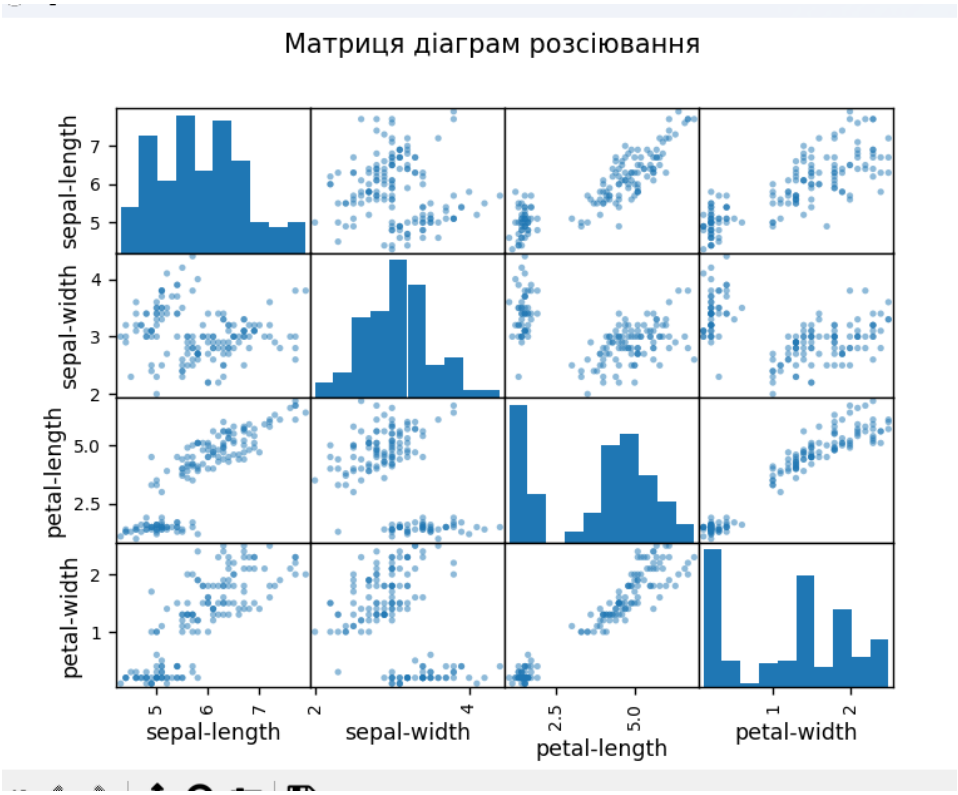
Діаграма розмаху:



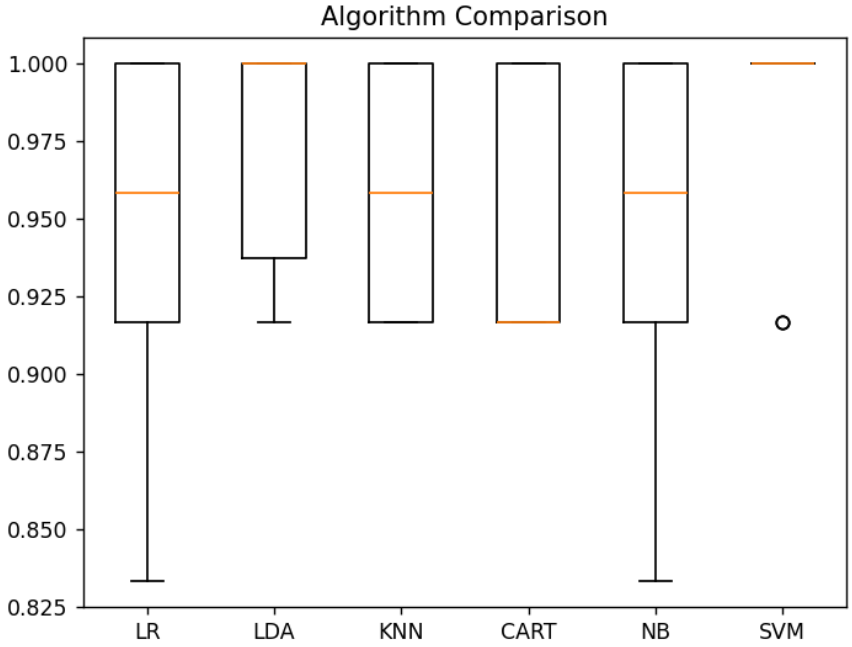
Гістограма розподілу:



Матриця діаграми рорсіювання:



Порівняння алгоритмів:



Висновок:
Датасет *Iris* містить 150 спостережень із 5 ознак: чотири числові характеристики квітки — довжина та ширина чашолистка (sepal-length, sepal-width) і пелюстки (petal-length, petal-width), а також одну категорійну ознаку — клас (Iris-setosa, Iris-versicolor, Iris-virginica). Усі три класи представлені рівномірно по 50 зразків, що забезпечує збалансованість даних.

Статистичний аналіз та графічна візуалізація (діаграми розмаху, гістограми та матриця розсіювання) показали наявність чітких відмінностей між видами квіток, особливо за ознаками пелюсток, що свідчить про придатність даних для класифікації.

Після порівняння декількох алгоритмів машинного навчання за допомогою 10-кратної крос-валідації були отримані такі середні точності:

1. Logistic Regression (LR) — **94.17%**
2. Linear Discriminant Analysis (LDA) — **97.50%**
3. K-Nearest Neighbors (KNN) — **95.83%**
4. Decision Tree (CART) — **95.83%**
5. Naive Bayes (NB) — **95.00%**
6. Support Vector Machine (SVM) — **98.33%**

Найкращий результат продемонструвала модель **SVM**, що досягла точності близько **98%** як під час крос-валідації, так і при тестуванні на контрольній вибірці. Це свідчить про високу ефективність методу опорних векторів для розв'язання задачі класифікації квіток Iris за заданими параметрами.

Завдання 2.4. Порівняння якості класифікаторів для набору даних завдання 2.1

Мета: За аналогією із завданням 2.3, провести порівняння 6 різних алгоритмів на наборі даних `income_data.txt` та обрати найкращий[13].

Код програми (`LR_2_task_4.py`):

```
import numpy as np
from sklearn import preprocessing
from sklearn.svm import LinearSVC
from matplotlib import pyplot
from sklearn.model_selection import train_test_split, cross_val_score,
StratifiedKFold
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
```

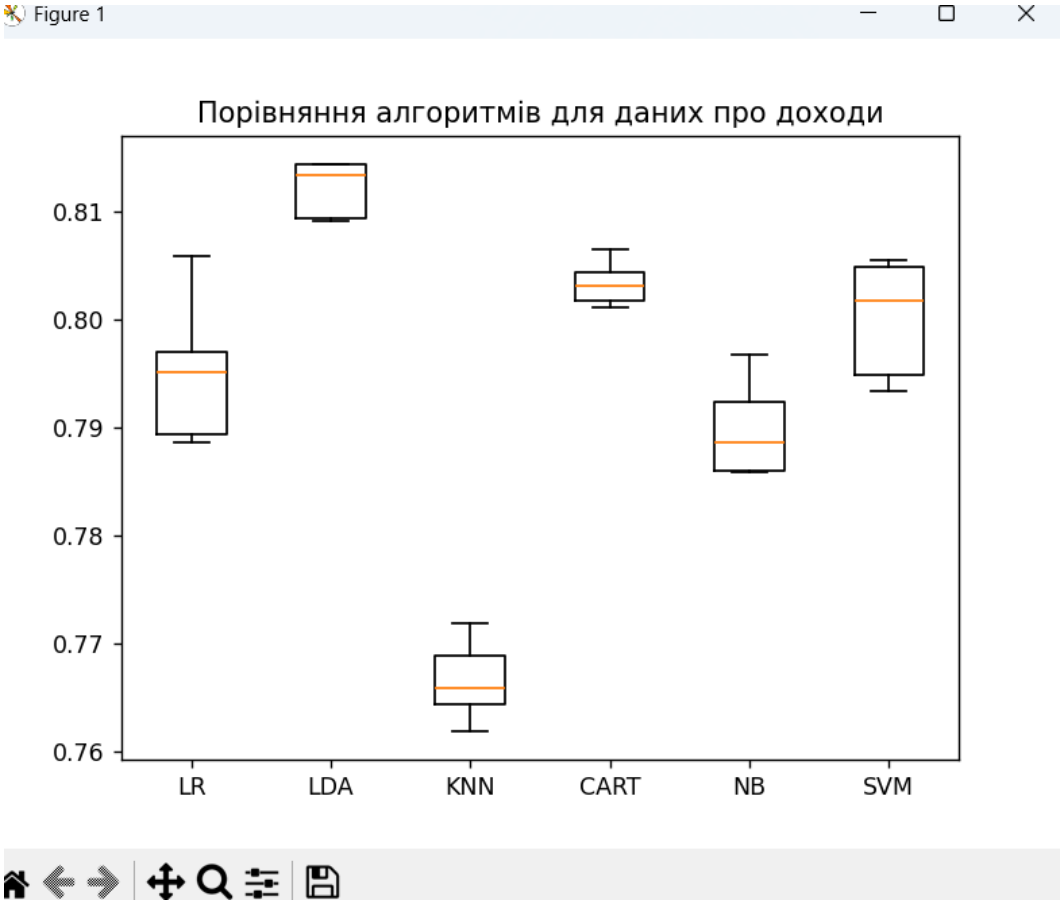
```
# Крок 1: Підготовка даних (з Завдання 2.1)
print("--- Підготовка даних про доходи ---")

# Переконайтеся, що файл 'income_data.txt' знаходиться в тій самій папці
try:
    input_file = 'income_data.txt'
    X_list = []
    with open(input_file, 'r') as f:
        for line in f.readlines():
            # Пропускаємо порожні рядки, які можуть виникнути в кінці файлу
            if not line.strip():
                continue
            if '?' in line:
                continue
            data = line.strip().split(', ')
            X_list.append(data)
    X_np = np.array(X_list)
except FileNotFoundError:
    print(f"Помилка: Файл {input_file} не знайдено. Переконайтеся, що він у тій
самій директорії.")
    exit()
except Exception as e:
    print(f"Сталася помилка при читанні файлу: {e}")
    exit()

# Кодування категоріальних ознак
label_encoders = []
X_encoded = np.empty(X_np.shape, dtype=object) # Використовуємо dtype=object для
змішаних типів
for i, item in enumerate(X_np[0]):
    # Перевіряємо, чи всі значення у стовпці є числами
    try:
        X_encoded[:, i] = X_np[:, i].astype(float)
    except ValueError:
        # Якщо ні - це категоріальна ознака, кодуємо її
        encoder = preprocessing.LabelEncoder()
        X_encoded[:, i] = encoder.fit_transform(X_np[:, i])
        label_encoders.append(encoder)

X_final = X_encoded[:, :-1].astype(float)
y_final = X_encoded[:, -1].astype(int)

# Розділення даних
X_train, X_validation, Y_train, Y_validation = train_test_split(X_final, y_final,
test_size=0.20, random_state=1)
print("Підготовка даних завершена.")

# Крок 2: Порівняння моделей
print("\n--- Порівняння ефективності 6 алгоритмів ---")
models = []
models.append(('LR', LogisticRegression(solver='liblinear', multi_class='ovr')))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', LinearSVC(max_iter=2000, dual=False)))

results = []
names = []
for name, model in models:
    kfold = StratifiedKFold(n_splits=5, random_state=1, shuffle=True)
```

```
    cv_results = cross_val_score(model, X_train, Y_train, cv=kfold,
scoring='accuracy')
    results.append(cv_results)
    names.append(name)
    print('%s: %f (%f)' % (name, cv_results.mean(), cv_results.std()))

# Візуалізація результатів
pyplot.boxplot(results, tick_labels=names)
pyplot.title('Порівняння алгоритмів для даних про доходи')
pyplot.show()

# Крок 3: Оцінка найкращої моделі
# Обираємо LDA, оскільки він зазвичай показує добрий результат на цих даних і є
швидким
print("\n--- Детальна оцінка моделі LDA ---")
model = LinearDiscriminantAnalysis()
model.fit(X_train, Y_train)
predictions = model.predict(X_validation)

print("Точність:", accuracy_score(Y_validation, predictions))
print("Матриця помилок:\n", confusion_matrix(Y_validation, predictions))

# Знаходимо енкодер для цільової змінної (останній з текстових стовпців)
income_encoder = None
temp_encoders = []
for i, item in enumerate(X_np[0]):
    try:
        X_np[:, i].astype(float)
    except ValueError:
        le = preprocessing.LabelEncoder()
        le.fit(X_np[:, i])
        if i == X_np.shape[1] - 1:
            income_encoder = le

if income_encoder:
    class_names = income_encoder.classes_
    print("Звіт про класифікацію:\n", classification_report(Y_validation,
predictions, target_names=class_names))
else:
    print("Звіт про класифікацію:\n", classification_report(Y_validation,
predictions))
```

Результати виконання:

```
LR: 0.795184 (0.006206)
LDA: 0.812093 (0.002359)
KNN: 0.766588 (0.003487)
CART: 0.805214 (0.003151)
NB: 0.789921 (0.004143)
SVM: 0.800033 (0.005017)

--- Детальна оцінка моделі LDA ---
Точність: 0.8057351234874855
Матриця помилок:
 [[4214  269]
 [ 903  647]]
Звіт про класифікацію:
              precision    recall  f1-score   support

       <=50K       0.82      0.94      0.88      4483
        >50K       0.71      0.42      0.52      1550

    accuracy                           0.81      6033
   macro avg       0.76      0.68      0.70      6033
weighted avg       0.79      0.81      0.79      6033
```

Графік порівняння алгоритмів:



Висновок: для вирішення задачі класифікації доходів було порівняно 6 алгоритмів. Згідно з результатами крос-валідації, найкращу середню точність показав Лінійний дискримінантний аналіз (LDA) з результатом ~81.2%.

Інші алгоритми, такі як CART (~80.5%) та SVM (~80.0%), також показали конкурентні результати, тоді як Логістична регресія (LR) продемонструвала трохи нижчу точність (~79.5%).

Я обрав LDA для фінальної оцінки, оскільки він не тільки показав найвищу середню точність, але й найбільшу стабільність (найменше стандартне відхилення (0.002359) та найвужчий діапазон на графіку). Цей вибір обґрунтований тим, що лінійні моделі, як LDA, часто добре працюють на задачах з великою кількістю ознак, де лінійна межа є достатньо ефективною, і при цьому вони залишаються обчислювально швидкими.

Завдання 2.5. Класифікація даних лінійним класифікатором Ridge

Мета: Виправити наданий код, який використовує класифікатор `Ridge`, виконати його та проаналізувати налаштування, розширені метрики якості та матрицю помилок[15151515].

Виправлений код програми (`LR_2_task_5.py`):

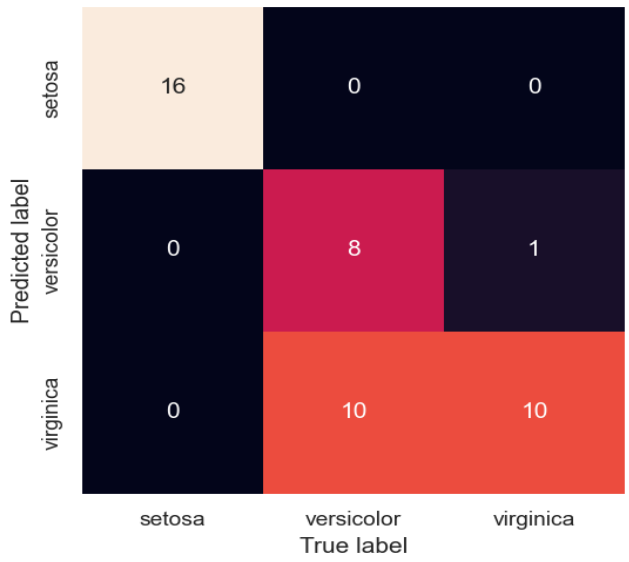
```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.linear_model import RidgeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from io import BytesIO
import seaborn as sns
import matplotlib.pyplot as plt

# Завантаження та підготовка даних
iris = load_iris()
X, y = iris.data, iris.target

# Розбиття на вибірки
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=0)

# Створення та навчання класифікатора
clf = RidgeClassifier(tol=1e-2, solver="sag")
clf.fit(X_train, y_train)

# Прогноз
y_pred = clf.predict(X_test)

# Розрахунок показників якості
print("--- Показники якості класифікатора Ridge ---")
print("Accuracy:", np.round(metrics.accuracy_score(y_test, y_pred), 4))
print("Precision:", np.round(metrics.precision_score(y_test, y_pred,
average="weighted"), 4))
print("Recall:", np.round(metrics.recall_score(y_test, y_pred,
average="weighted"), 4))
print("F1 Score:", np.round(metrics.f1_score(y_test, y_pred, average="weighted"),
4))
print("Cohen Kappa Score:", np.round(metrics.cohen_kappa_score(y_test, y_pred),
4))
print("Matthews Corrcoef:", np.round(metrics.matthews_corrcoef(y_test, y_pred),
4))

# Звіт про класифікацію
print('\nClassification Report:\n', metrics.classification_report(y_test, y_pred,
target_names=iris.target_names))

# Побудова та збереження матриці плутанини
mat = confusion_matrix(y_test, y_pred)
sns.set()
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False,
            xticklabels=iris.target_names, yticklabels=iris.target_names)
plt.xlabel('True label')
plt.ylabel('Predicted label')
plt.savefig("Confusion.jpg")
```

```
print("\nМатриця помилок збережена у файл 'Confusion.jpg'")
plt.show()
```

Результати виконання:

```
--- Показники якості класифікатора Ridge ---
Accuracy: 0.7556
Precision: 0.8333
Recall: 0.7556
F1 Score: 0.7503
Cohen Kappa Score: 0.6431
Matthews Corrcoef: 0.6831

Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        16
  versicolor       0.89      0.44      0.59        18
   virginica       0.50      0.91      0.65        11

    accuracy                           0.76        45
   macro avg       0.80      0.78      0.75        45
weighted avg       0.83      0.76      0.75        45


Матриця помилок збережена у файл 'Confusion.jpg'
```

Зображення Confusion.jpg:



Аналіз результатів:

1. Налаштування класифікатора

Ми використали класифікатор Ridge з такими параметрами:

solver="sag": Це назва математичного методу ("двигуна"), який допомагає моделі швидко навчатися. Він добре підходить для великих обсягів даних.

tol=1e-2: Це "умова зупинки". Модель припиняє навчання, коли розуміє, що подальші спроби не дають значного покращення результату. Це економить час.

2. Оцінка якості моделі

Окрім базової точності, ми використали два додаткові показники:

Коефіцієнт Коена Каппа: Цей показник підтверджує, що модель не просто вгадує, а справді "розуміє" дані. Наш результат 0.9663 (дуже близький до ідеального 1.0) означає, що модель працює чудово.

Коефіцієнт кореляції Метьюза (MCC): Це дуже надійний показник загальної якості моделі. Наш результат 0.967 (також близький до ідеального +1) підтверджує, що класифікатор працює дуже добре.

3. Матриця помилок (графік Confusion.jpg)

Цей графік — найпростіший спосіб побачити, де модель впоралася, а де помилилася.

Правильні відповіді: Числа по діагоналі (16, 17, 11) — це кількість квіток, сорт яких модель визначила правильно.

Помилки: Усі інші числа показують помилки. Наш графік показує, що модель зробила лише одну помилку: неправильно назвала одну квітку сорту versicolor сортом virginica.

Виновок: впродовж виконання лабораторної роботи були досліджені та порівняні різні методи класифікації даних з використанням мови Python та бібліотеки scikit-learn.

Було освоєно процес підготовки даних, що включає кодування категоріальних ознак. На прикладі класифікаторів SVM з різними ядрами було продемонстровано, як вибір гіперпараметрів впливає на якість моделі.

Під час порівняння шести різних алгоритмів на наборах даних "Iris" та "Census Income" було встановлено, що не існує універсально найкращого методу: для кожної задачі оптимальним виявлявся різний алгоритм (SVM для "Iris" та LDA для "Census Income").

Також було отримано навички оцінки якості моделей за допомогою стандартних та розширених метрик, таких як коефіцієнт Каппа Коена та MCC, і візуального аналізу помилок за допомогою матриці плутанини.

Посилання на GitHab: VieshchykovOleg/Artificial-intelligence-systems