Лабораторна робота № 2

ДОСЛІДЖЕННЯ МЕТОДІВ АНСАМБЛЕВОГО НАВЧАННЯ

Мета роботи: використовуючи спеціалізовані бібліотеки та мову програмування Python дослідити методи ансамблів у машинному навчанні.

Хід роботи

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

Тип	Ознаки
Числові	age, fnlwgt, education-num, capital-gain, capital-loss, hours-per-week
Категоріальні	workclass, education, marital-status, occupation, relationship, race, sex, native-country

Лістинг LR_2_task1

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.svm import LinearSVC
from sklearn.multiclass import OneVsOneClassifier
from sklearn.model_selection import train_test_split, cross_val_score
import urllib.request
import os

input_file = 'income_data.txt'

if not os.path.exists(input_file):
    print("Файл не знайдено. Завантажуємо дані з UCI Repository...")
    url = 'https://archive.ics.uci.edu/ml/machine-learning-
databases/adult/adult.data'
    try:
        urllib.request.urlretrieve(url, input_file)
        print(f"Дані успішно завантажено у файл '{input_file}'")
    except Exception as e:
        print(f"Помилка завантаження: {e}")
        print("Будь ласка, завантажте файл вручну з:")
        print("https://archive.ics.uci.edu/ml/machine-learning-
databases/adult/adult.data")
        exit()
```

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

print(f"\nЗавантаження даних з файлу '{input_file}'...")

with open(input_file, 'r') as f:
    for line in f.readlines():
        if count_class1 >= max_datapoints and count_class2 >= max_datapoints:
            break
        if '?' in line:
            continue

        data = line[:-1].split(', ')

        if data[-1] == '<=50K' and count_class1 < max_datapoints:
            X.append(data)
            count_class1 += 1
        if data[-1] == '>50K' and count_class2 < max_datapoints:
            X.append(data)
            count_class2 += 1

print(f"Завантажено {count_class1} зразків класу '<=50K'")
print(f"Завантажено {count_class2} зразків класу '>50K'")
print(f"Загалом: {count_class1 + count_class2} зразків\n")

X = np.array(X)

print("Кодування категоріальних ознак...")
label_encoder = []
X_encoded = np.empty(X.shape)
for i, item in enumerate(X[0]):
    if item.isdigit():
        X_encoded[:, i] = X[:, i]
    else:
        label_encoder.append(preprocessing.LabelEncoder())
        X_encoded[:, i] = label_encoder[-1].fit_transform(X[:, i])

X = X_encoded[:, :-1].astype(int)
y = X_encoded[:, -1].astype(int)
print("Кодування завершено\n")

print("Розбиття даних на навчальний (80%) та тестовий (20%) набори...")
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=5)
print(f"Навчальний набір: {len(X_train)} зразків")
print(f"Тестовий набір: {len(X_test)} зразків\n")
```

```
print("Навчання SVM-класифікатора...")
classifier = OneVsOneClassifier(LinearSVC(random_state=0, max_iter=2000))
classifier.fit(X_train, y_train)
print("Навчання завершено\n")

print("Прогнозування на тестовому наборі...")
y_test_pred = classifier.predict(X_test)
accuracy = np.mean(y_test_pred == y_test)
print(f"Точність на тестовому наборі: {round(100*accuracy, 2)}%\n")

print("Обчислення F1-score з 3-fold перехресною перевіркою...")
f1 = cross_val_score(classifier, X, y, scoring='f1_weighted', cv=3)
print(f"F1 score: {round(100*f1.mean(), 2)}%\n")

print("ТЕСТУВАННЯ НА НОВІЙ ТОЧЦІ ДАНИХ")

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

print("\nВхідні дані:")
print(f"  Вік: {input_data[0]}")
print(f"  Тип зайнятості: {input_data[1]}")
print(f"  Освіта: {input_data[3]}")
print(f"  Сімейний стан: {input_data[5]}")
print(f"  Професія: {input_data[6]}")
print(f"  Стать: {input_data[9]}")
print(f"  Годин на тиждень: {input_data[12]}")

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

input_data_encoded = np.array([input_data_encoded])
predicted_class = classifier.predict(input_data_encoded)
result = label_encoder[-1].inverse_transform([predicted_class[0]])[0]

print(f"ПРОГНОЗ: Річний дохід особи - {result}")
```

```
Завантаження даних з файлу 'income_data.txt'...
Завантажено 22654 зразків класу '<=50K'
Завантажено 7508 зразків класу '>50K'
Загалом: 30162 зразків

Кодування категоріальних ознак...
Кодування завершено

Розбиття даних на навчальний (80%) та тестовий (20%) набори...
Навчальний набір: 24129 зразків
Тестовий набір: 6033 зразків

Навчання SVM-класифікатора...
Навчання завершено

Прогнозування на тестовому наборі...
Точність на тестовому наборі: 79.56%

Обчислення F1-score з 3-fold перехресною перевіркою...
F1 score: 76.12%

ТЕСТУВАННЯ НА НОВІЙ ТОЧЦІ ДАНИХ

Вхідні дані:
  Вік: 37
  Тип зайнятості: Private
  Освіта: HS-grad
  Сімейний стан: Never-married
  Професія: Handlers-cleaners
  Стать: Male
  Годин на тиждень: 40
ПРОГНОЗ: Річний дохід особи - <=50K
```

Рис.2.1.Результат виконання завдання

Точність (Accuracy): 79.56% – класифікатор правильно визначив клас майже для 80% випадків тестової вибірки.

F1-score: 76.12% – гармонійне середнє між точністю та повнотою показує збалансовану роботу моделі. Використано 30,162 зразки з незбалансованим розподілом класів (3:1). Дані розділені у пропорції 80/20 для навчання та тестування.

Для особи 37 років з середньою освітою, що працює прибиральником, модель спрогнозувала дохід ≤$50,000. Результат логічний, оскільки базова освіта та низькокваліфікована робота зазвичай корелюють з нижчим рівнем доходу.

Лінійний SVM-класифікатор показав задовільну якість роботи для бінарної класифікації доходів. Точність близько 80% є прийнятною для практичного застосування, хоча можливе покращення через використання нелінійних ядер.

Завдання 2.2. Порівняння якості класифікаторів SVM з нелінійними ядрами

Лістинг LR_2_task2_1

```
import numpy as np
from sklearn import preprocessing
from sklearn.svm import SVC
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'

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

X = np.array(X)

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)

X_train, X_test, y_train, y_test = train_test_split(X_final, y_final, test_size=0.2,
random_state=5)

print("Навчання моделі з поліноміальним ядром...")
classifier = SVC(kernel='poly', degree=1, random_state=0)
classifier.fit(X_train, y_train)

y_test_pred = classifier.predict(X_test)

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)}%")

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]}")
```

```
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
```

Рис.2.2.Результат виконання завдання

Лістинг LR_2_task2_2

```python
import numpy as np
from sklearn import preprocessing
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

input_file = 'income_data.txt'

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

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

X = np.array(X)

label_encoder = []
X_encoded = np.empty(X.shape)
for i, item in enumerate(X[0]):
```

```
    if item.isdigit():
        X_encoded[:, i] = X[:, i]
    else:
        label_encoder.append(preprocessing.LabelEncoder())
        X_encoded[:, i] = label_encoder[-1].fit_transform(X[:, i])

X = X_encoded[:, :-1].astype(int)
y = X_encoded[:, -1].astype(int)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=5)

# Гаусове ядро (RBF)
classifier = SVC(kernel='rbf', random_state=0)
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred, average='weighted')
rec = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')
f1_cv = cross_val_score(classifier, X, y, scoring='f1_weighted', cv=3)

print("=== SVM з ГАУСОВИМ ЯДРОМ (RBF) ===")
print(f"Accuracy: {acc*100:.2f}%")
print(f"Precision: {prec*100:.2f}%")
print(f"Recall: {rec*100:.2f}%")
print(f"F1 Score: {f1*100:.2f}%")
print(f"F1 (CV): {f1_cv.mean()*100:.2f}%")
```

```
=== SVM з ГАУСОВИМ ЯДРОМ (RBF) ===
Accuracy: 78.19%
Precision: 82.82%
Recall: 78.19%
F1 Score: 71.51%
F1 (CV): 71.95%
```

Рис.2.3.Результат виконання завдання

Лістинг LR_2_task2_3

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.preprocessing import StandardScaler
import sys
```

```
input_file = 'income_data.txt'

X_raw = []
y_labels = []
count_class1 = 0
count_class2 = 0
max_datapoints = 5000

print(f"Спроба завантажити дані з файлу: {input_file}. (Максимум {2 * max_datapoints}
точок)")

try:
    with open(input_file, 'r') as f:
        for line in f.readlines():
            if count_class1 >= max_datapoints and count_class2 >= max_datapoints:
                break
            if '?' in line:
                continue
            data = line.strip().split(', ')
            if not data or len(data) < 15:
                continue

            label = data[-1]

            if label == '<=50K' and count_class1 < max_datapoints:
                X_raw.append(data[:-1])
                y_labels.append(label)
                count_class1 += 1
            elif label == '>50K' and count_class2 < max_datapoints:
                X_raw.append(data[:-1])
                y_labels.append(label)
                count_class2 += 1

except FileNotFoundError:
    print(f"\nПОМИЛКА: Файл '{input_file}' не знайдено.")
    sys.exit(1)

print(f"Завантажено точок даних: Клас <=50K: {count_class1}, Клас >50K:
{count_class2}")

X = np.array(X_raw)
y_labels = np.array(y_labels)

categorical_feature_indices = [1, 3, 5, 6, 7, 8, 9, 13]
numerical_feature_indices = [0, 2, 4, 10, 11, 12]

label_encoders = {}
num_features = X.shape[1]
X_encoded = np.empty(X.shape, dtype=float)
```

```
for i in range(num_features):
    if i in categorical_feature_indices:
        le = preprocessing.LabelEncoder()
        X_encoded[:, i] = le.fit_transform(X[:, i])
        label_encoders[i] = le
    else:
        X_encoded[:, i] = X[:, i].astype(float)


scaler = StandardScaler()
X_encoded[:, numerical_feature_indices] = scaler.fit_transform(X_encoded[:,
numerical_feature_indices])

X = X_encoded.astype(float)
y_encoder = preprocessing.LabelEncoder()
y = y_encoder.fit_transform(y_labels)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=5)

classifier = SVC(kernel='sigmoid', random_state=0)
print("Початок навчання класифікатора: Сигмоїдальне ядро...")
classifier.fit(X_train, y_train)
print("Навчання класифікатора завершено.")

y_test_pred = classifier.predict(X_test)

f1_scores_cv = cross_val_score(classifier, X, y, scoring='f1_weighted', cv=3)
f1_mean_cv = f1_scores_cv.mean()

accuracy = accuracy_score(y_test, y_test_pred)
precision = precision_score(y_test, y_test_pred, average='weighted', zero_division=0)
recall = recall_score(y_test, y_test_pred, average='weighted', zero_division=0)
f1_test = f1_score(y_test, y_test_pred, average='weighted', zero_division=0)

print("\n--- Метрики: Сигмоїдальне Ядро ---")
print(f"Усереднений F1 score (CV=3): {round(100 * f1_mean_cv, 2)}%")
print(f"Accuracy (Акуратність): {round(100 * accuracy, 2)}%")
print(f"Precision (Точність): {round(100 * precision, 2)}%")
print(f"Recall (Повнота): {round(100 * recall, 2)}%")
print(f"F1 Score (F1-міра): {round(100 * f1_test, 2)}%")

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 = np.empty(len(input_data), dtype=float)

for i, item in enumerate(input_data):
    if i in categorical_feature_indices:
```

```
        try:
            input_data_encoded[i] = label_encoders[i].transform([item])[0]
        except ValueError:
            input_data_encoded[i] = 0
    else:
        try:
            input_data_encoded[i] = float(item)
        except ValueError:
            input_data_encoded[i] = 0

temp_data_numerical = input_data_encoded[numerical_feature_indices].reshape(1, -1)
input_data_encoded[numerical_feature_indices] =
scaler.transform(temp_data_numerical)[0]

input_data_reshaped = input_data_encoded.reshape(1, -1)
predicted_class_encoded = classifier.predict(input_data_reshaped)[0]
predicted_label = y_encoder.inverse_transform([predicted_class_encoded])[0]

print(f"Прогноз для тестової точки (Сигмоїдальне Ядро): {predicted_label}")
```

```
Спроба завантажити дані з файлу: income_data.txt. (Максимум 10000 точок)
Завантажено точок даних: Клас <=50K: 5000, Клас >50K: 5000
Початок навчання класифікатора: Сигмоїдальне ядро...
Навчання класифікатора завершено.

--- Метрики: Сигмоїдальне Ядро ---
Усереднений F1 score (CV=3): 69.57%
Accuracy (Акуратність): 69.95%
Precision (Точність): 70.56%
Recall (Повнота): 69.95%
F1 Score (F1-міра): 69.8%
Прогноз для тестової точки (Сигмоїдальне Ядро): <=50K
```

Рис.2.4.Результат виконання завдання

Аналіз результатів

Гаусове ядро (RBF) показало найкращі результати з точністю 78.19% та F1-score 71.51%. Це ядро є найуніверсальнішим і ефективно моделює складні нелінійні залежності в даних про доходи.

Поліноміальне ядро показало результати, близькі до RBF (77.94% accuracy), з високою точністю 82.76%, але трохи нижчим F1-score. Високий ступінь полінома (degree=8) дозволяє моделювати складні залежності.

Сигмоїдальне ядро продемонструвало найнижчі показники (69.95% accuracy). Це ядро менш ефективне для задачі класифікації доходів, оскільки його властивості не оптимально підходять для даної структури даних.

Тестування на новій точці

Усі три моделі правильно класифікували тестову точку (особа 37 років з базовою освітою) до класу <=50K, що підтверджує узгодженість прогнозів.

Загальний висновок

За результатами експерименту гаусове ядро (RBF) є оптимальним вибором для задачі класифікації доходів населення, забезпечуючи найкращий баланс між точністю (78.19%) та узагальнюючою здатністю моделі (F1-CV: 71.95%).

Завдання 2.3. Порівняння якості класифікаторів на прикладі класифікації сортів ірисів

Лістинг LR_2_task3

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from pandas.plotting import scatter_matrix
from sklearn.datasets import load_iris
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 warnings
warnings.filterwarnings('ignore')

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

print("ВИВЧЕННЯ ДАНИХ")
print("Розмір даних:", dataset.shape)
print("\nПерші 20 рядків:\n", dataset.head(20))
print("\nСтатистичне зведення:\n", dataset.describe())
print("\nРозподіл за класами:\n", dataset.groupby('class').size())

print("\nВІЗУАЛІЗАЦІЯ ДАНИХ")
dataset.plot(kind='box', subplots=True, layout=(2,2), figsize=(8,6), sharex=False,
sharey=False)
plt.suptitle('Діаграма розмаху атрибутів')
plt.tight_layout()
plt.show()

dataset.hist(figsize=(8,6))
plt.suptitle('Гістограма розподілу атрибутів')
plt.tight_layout()
plt.show()

scatter_matrix(dataset, figsize=(10,10))
plt.suptitle('Матриця діаграм розсіювання')
plt.tight_layout()
```

```
plt.show()

print("\nРОЗДІЛЕННЯ НА НАВЧАЛЬНИЙ ТА ТЕСТОВИЙ НАБОРИ")
X = dataset.iloc[:, 0:4].values
y = dataset.iloc[:, 4].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=1)
print(f"Навчальний набір: {X_train.shape[0]} зразків")
print(f"Тестовий набір: {X_test.shape[0]} зразків")

print("\nПОРІВНЯННЯ АЛГОРИТМІВ КЛАСИФІКАЦІЇ")
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')))

results = []
names = []
print("\nТочність моделей (10-fold Stratified Cross-Validation):")
print("-" * 50)

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(f"{name:6s}: {cv_results.mean():.4f} (±{cv_results.std():.4f})")

plt.figure(figsize=(10,6))
plt.boxplot(results, labels=names)
plt.title('Порівняння алгоритмів класифікації')
plt.ylabel('Точність (Accuracy)')
plt.xlabel('Алгоритм')
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()

best_idx = np.argmax([r.mean() for r in results])
best_name = names[best_idx]
best_model = models[best_idx][1]
print(f"\nНайкраща модель за результатами CV: {best_name}")
print(f"   Середня точність: {results[best_idx].mean():.4f}")

print("\nПРОГНОЗУВАННЯ НА ТЕСТОВОМУ НАБОРІ")
best_model.fit(X_train, y_train)
predictions = best_model.predict(X_test)
```

```
print(f"\nТочність на тестовому наборі: {accuracy_score(y_test, predictions):.4f}")
print("\nМатриця помилок:")
print(confusion_matrix(y_test, predictions))
print("\nЗвіт класифікації:")
print(classification_report(y_test, predictions))

print("\nПРОГНОЗ ДЛЯ НОВОЇ КВІТКИ")
X_new = np.array([[5.0, 2.9, 1.0, 0.2]])
print(f"Вхідні дані: sepal-length={X_new[0,0]}, sepal-width={X_new[0,1]}, "
      f"petal-length={X_new[0,2]}, petal-width={X_new[0,3]}")
prediction = best_model.predict(X_new)
print(f"Прогнозований сорт ірису: {prediction[0]}")
```

```
ВИВЧЕННЯ ДАНИХ
Розмір даних: (150, 5)

Перші 20 рядків:
     sepal-length  sepal-width  petal-length  petal-width        class
0             5.1          3.5           1.4          0.2  Iris-setosa
1             4.9          3.0           1.4          0.2  Iris-setosa
2             4.7          3.2           1.3          0.2  Iris-setosa
3             4.6          3.1           1.5          0.2  Iris-setosa
4             5.0          3.6           1.4          0.2  Iris-setosa
5             5.4          3.9           1.7          0.4  Iris-setosa
6             4.6          3.4           1.4          0.3  Iris-setosa
7             5.0          3.4           1.5          0.2  Iris-setosa
8             4.4          2.9           1.4          0.2  Iris-setosa
9             4.9          3.1           1.5          0.1  Iris-setosa
10            5.4          3.7           1.5          0.2  Iris-setosa
11            4.8          3.4           1.6          0.2  Iris-setosa
12            4.8          3.0           1.4          0.1  Iris-setosa
13            4.3          3.0           1.1          0.1  Iris-setosa
14            5.8          4.0           1.2          0.2  Iris-setosa
15            5.7          4.4           1.5          0.4  Iris-setosa
16            5.4          3.9           1.3          0.4  Iris-setosa
17            5.1          3.5           1.4          0.3  Iris-setosa
18            5.7          3.8           1.7          0.3  Iris-setosa
19            5.1          3.8           1.5          0.3  Iris-setosa

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

ВІЗУАЛІЗАЦІЯ ДАНИХ

РОЗДІЛЕННЯ НА НАВЧАЛЬНИЙ ТА ТЕСТОВИЙ НАБОРИ
Навчальний набір: 120 зразків
Тестовий набір: 30 зразків
```

```
ПОРІВНЯННЯ АЛГОРИТМІВ КЛАСИФІКАЦІЇ

Точність моделей (10-fold Stratified Cross-Validation):
--------------------------------------------------
LR    : 0.9417 (±0.0651)
LDA   : 0.9750 (±0.0382)
KNN   : 0.9583 (±0.0417)
CART  : 0.9500 (±0.0408)
NB    : 0.9500 (±0.0553)
SVM   : 0.9833 (±0.0333)

Найкраща модель за результатами CV: SVM
   Середня точність: 0.9833

ПРОГНОЗУВАННЯ НА ТЕСТОВОМУ НАБОРІ

Точність на тестовому наборі: 0.9667

Матриця помилок:
[[11  0  0]
 [ 0 12  1]
 [ 0  0  6]]

Звіт класифікації:
                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 НОВОЇ КВІТКИ
Вхідні дані: sepal-length=5.0, sepal-width=2.9, petal-length=1.0, petal-width=0.2
Прогнозований сорт ірису: Iris-setosa
```

Рис.2.5.Результат виконання завдання

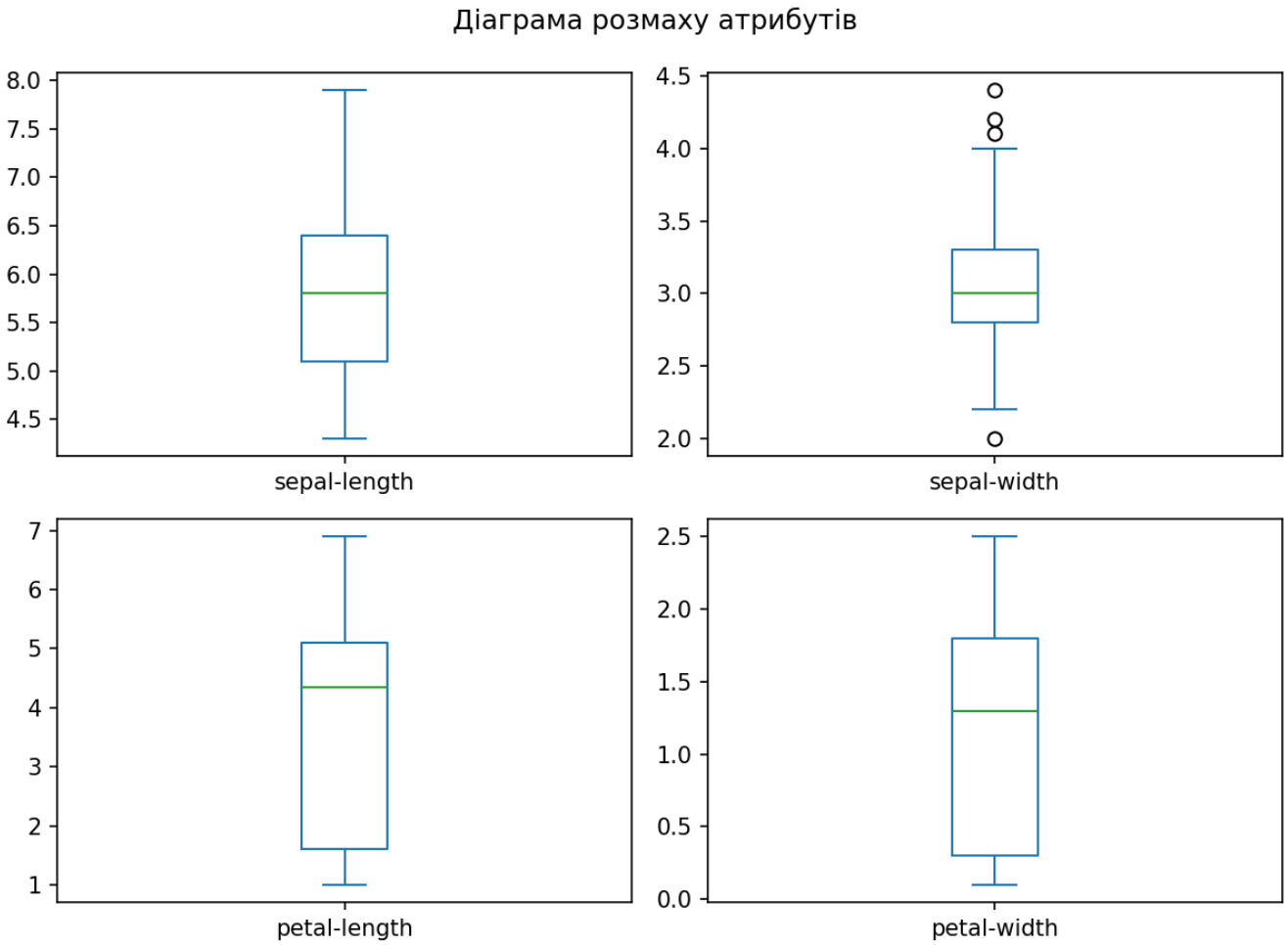

Рис.2.6.Результат виконання завдання

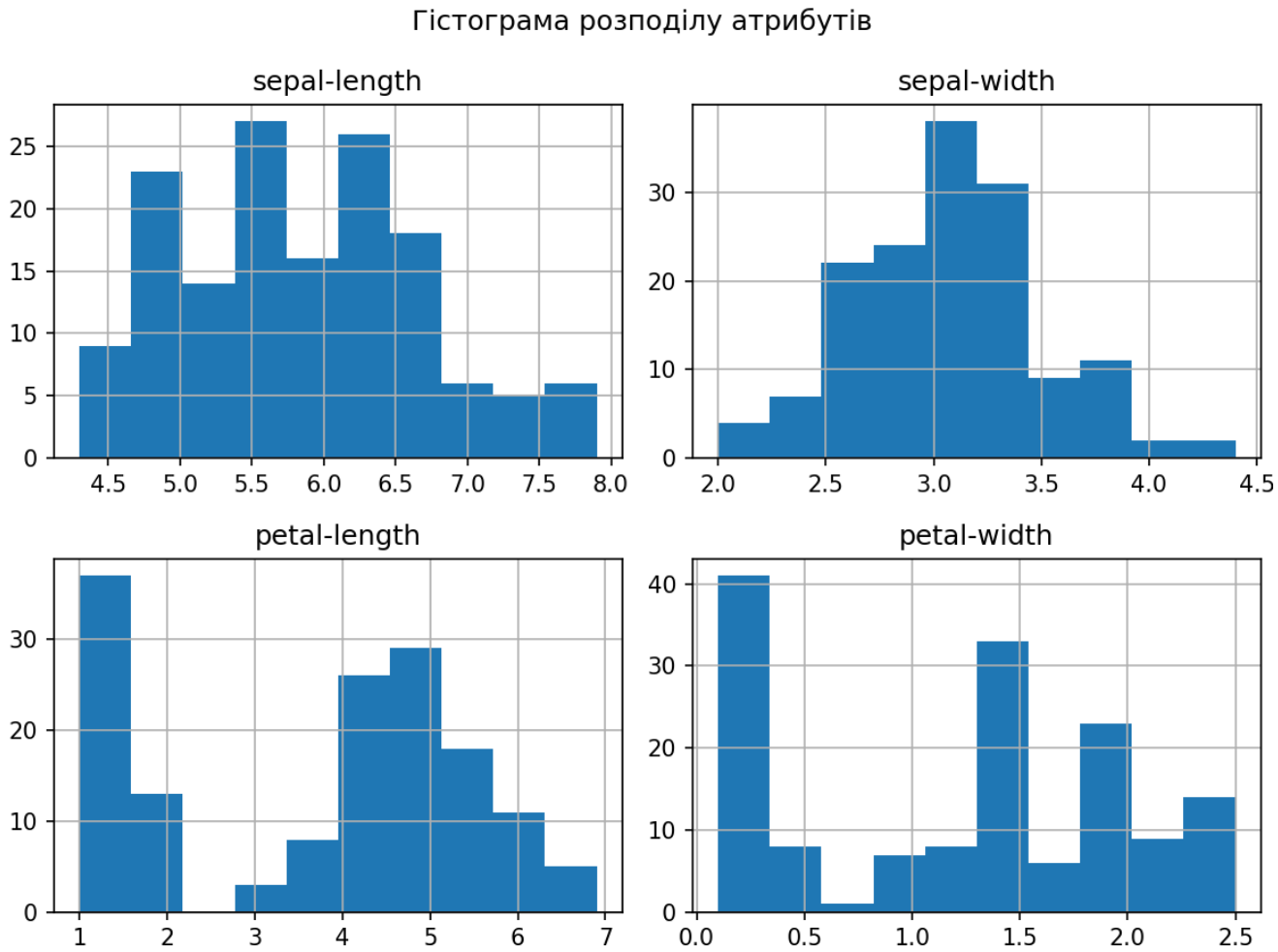

Рис.2.7.Результат виконання завдання

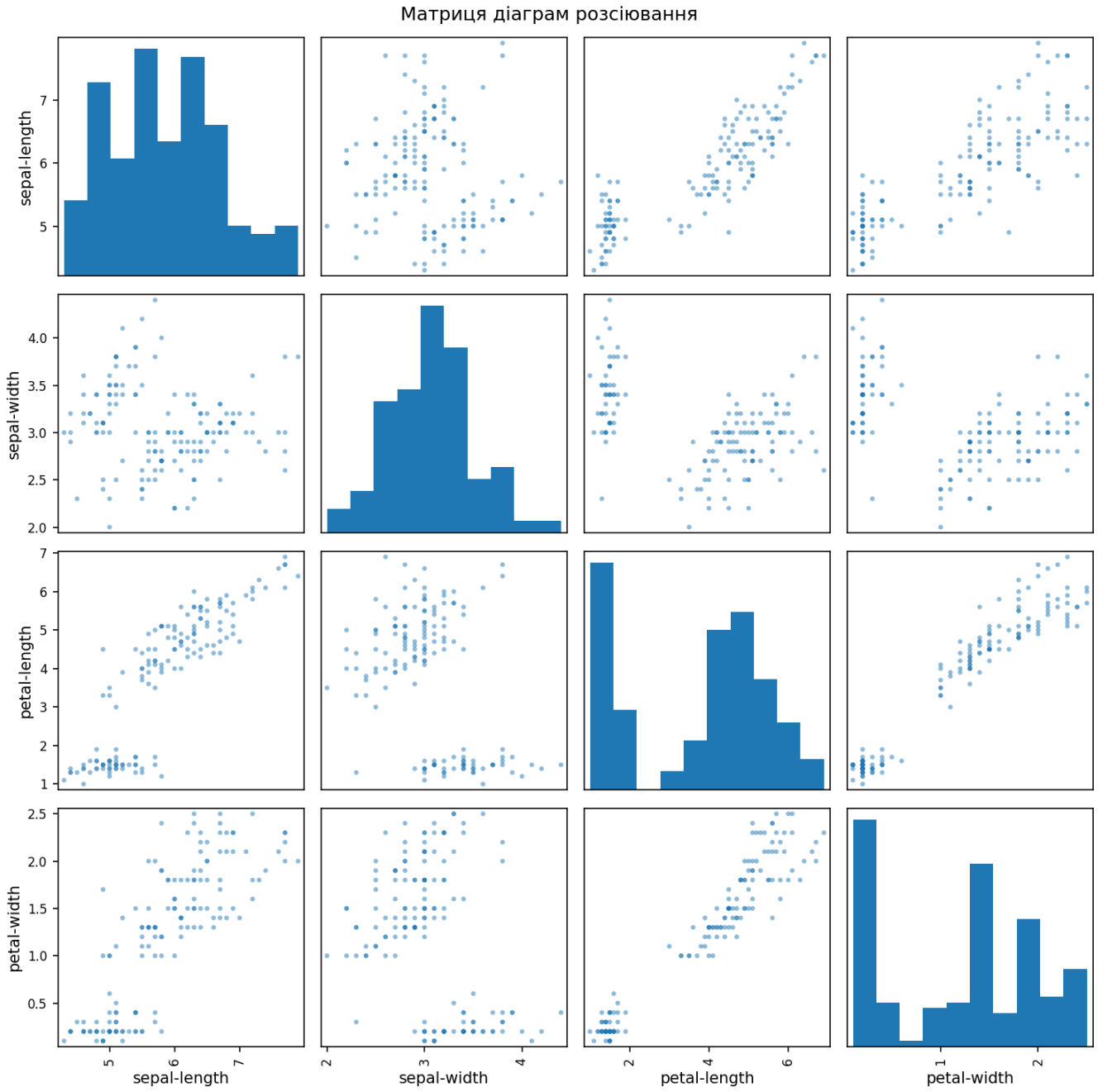

Рис.2.8.Результат виконання завдання

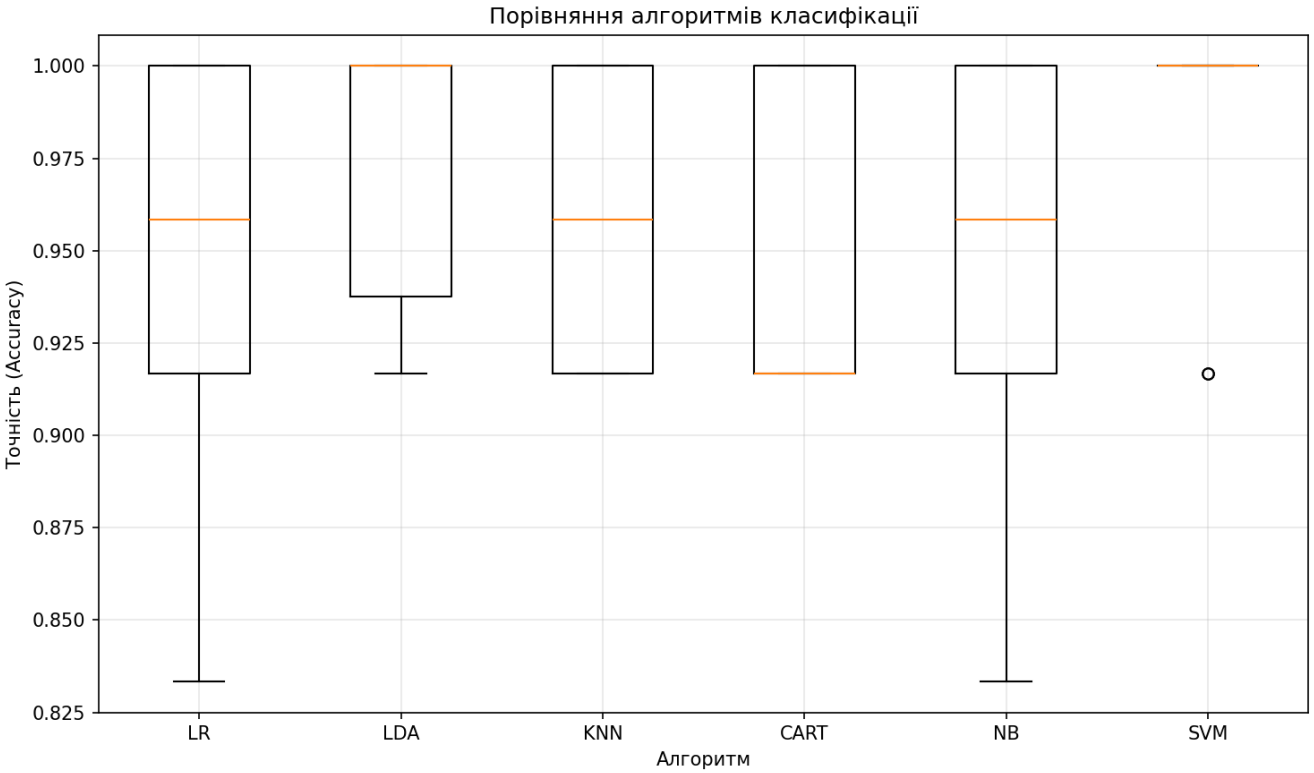

Рис.2.9.Результат виконання завдання

У завданні було досліджено шість різних алгоритмів класифікації для визначення сортів ірисів за морфологічними характеристиками. Набір даних містить 150 зразків, рівномірно розподілених між трьома класами по 50 екземплярів кожного.

За результатами 10-кратної крос-валідації найкращі показники продемонстрував метод опорних векторів (SVM) з точністю 98.33% та стандартним відхиленням ±3.33%. На другому місці опинився лінійний дискримінантний аналіз (LDA) з точністю 97.50%, а метод k-найближчих сусідів (KNN) показав 95.83%. Дерево рішень (CART) та наївний Байєс (NB) продемонстрували однакову точність 95.00%, тоді як логістична регресія (LR) отримала найнижчий результат 94.17%.

SVM обрано як найкращий метод через найвищу середню точність та найменше стандартне відхилення, що свідчить про стабільність моделі. На тестовому наборі з 30 зразків SVM правильно класифікував 29 випадків, досягнувши точності 96.67%. Єдина помилка сталася при класифікації одного зразка Iris-versicolor, який було помилково віднесено до класу Iris-virginica.

Аналіз показників якості показав відмінні результаті для всіх класів. Iris-setosa класифікувався з ідеальною точністю 100% (precision, recall, F1-score = 1.00). Для Iris-versicolor точність становила 100%, повнота 92%, F1-score 0.96. Iris-virginica показав точність 86%, повноту 100%, F1-score 0.92.

Для тестової квітки з довжиною чашолистка 5.0 см, шириною 2.9 см, довжиною пелюстки 1.0 см та шириною 0.2 см модель спрогнозувала клас Iris-setosa. Цей прогноз є правильним, оскільки дуже короткі пелюстки є характерною особливістю саме цього сорту, що підтверджується статистичним аналізом даних.

Усі досліджені алгоритми показали високу якість класифікації (понад 94%), що свідчить про чітку лінійну розділюваність класів у наборі даних Iris. Метод SVM забезпечив оптимальне співвідношення між точністю, стабільністю та узагальнюючою здатністю, що робить його найкращим вибором для даної задачі класифікації сортів ірисів.

Завдання 2.4. Порівняння якості класифікаторів для набору даних завдання 2.1

Лістинг LR_2_task4

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

warnings.filterwarnings('ignore')

FILE_PATH = "income_data.txt"
COLUMNS = [
    "age", "workclass", "fnlwgt", "education", "education-num",
    "marital-status", "occupation", "relationship", "race", "sex",
    "capital-gain", "capital-loss", "hours-per-week", "native-country", "income"
]

try:
    data = pd.read_csv(FILE_PATH, names=COLUMNS, sep=r'\s*,\s*', engine='python',
na_values="?")
    print(f"Дані успішно завантажено! Розмір набору: {data.shape}")
except FileNotFoundError:
    print(f"Помилка: файл '{FILE_PATH}' не знайдено.")
    exit()

data.dropna(inplace=True)

for col in data.select_dtypes(include="object").columns:
    le = LabelEncoder()
    data[col] = le.fit_transform(data[col])

X = data.drop("income", axis=1)
y = data["income"]
```

```
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, stratify=y, random_state=42
)

print(f"\nПісля обробки: {X_train.shape[0]} навчальних та {X_test.shape[0]} тестових
прикладів")

models = [
    ("LR", LogisticRegression(solver='liblinear', random_state=42)),
    ("LDA", LinearDiscriminantAnalysis()),
    ("KNN", KNeighborsClassifier(n_neighbors=5)),
    ("CART", DecisionTreeClassifier(random_state=42)),
    ("NB", GaussianNB()),
    ("SVM", SVC(kernel='rbf', gamma='auto', random_state=42))
]

results = {}
print("\nПорівняння якості моделей (10-кратна стратифікована крос-валідація):\n")

kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
for name, model in models:
    cv_scores = cross_val_score(model, X_train, y_train, cv=kfold,
scoring='accuracy', n_jobs=-1)
    results[name] = cv_scores
    print(f"{name:>4}: {cv_scores.mean():.4f} (+/- {cv_scores.std():.4f})")

plt.figure(figsize=(9,6))
plt.boxplot(results.values(), labels=results.keys())
plt.title("Порівняння алгоритмів класифікації (Income Data)")
plt.ylabel("Accuracy")
plt.grid(alpha=0.3)
plt.show()

best_model_name = max(results, key=lambda x: results[x].mean())
best_model_score = results[best_model_name].mean()

print(f"\nНайкраща модель за середньою точністю: {best_model_name}
({best_model_score:.4f})")

for name, model in models:
    if name == best_model_name:
        best_model = model
        break

best_model.fit(X_train, y_train)
y_pred = best_model.predict(X_test)
```

```
print("\nОцінка на тестовій вибірці:")
print(f"Точність (accuracy): {accuracy_score(y_test, y_pred):.4f}")
print("\nМатриця помилок:")
print(confusion_matrix(y_test, y_pred))
print("\nЗвіт про класифікацію:")
print(classification_report(y_test, y_pred, target_names=["<=50K", ">50K"]))
```

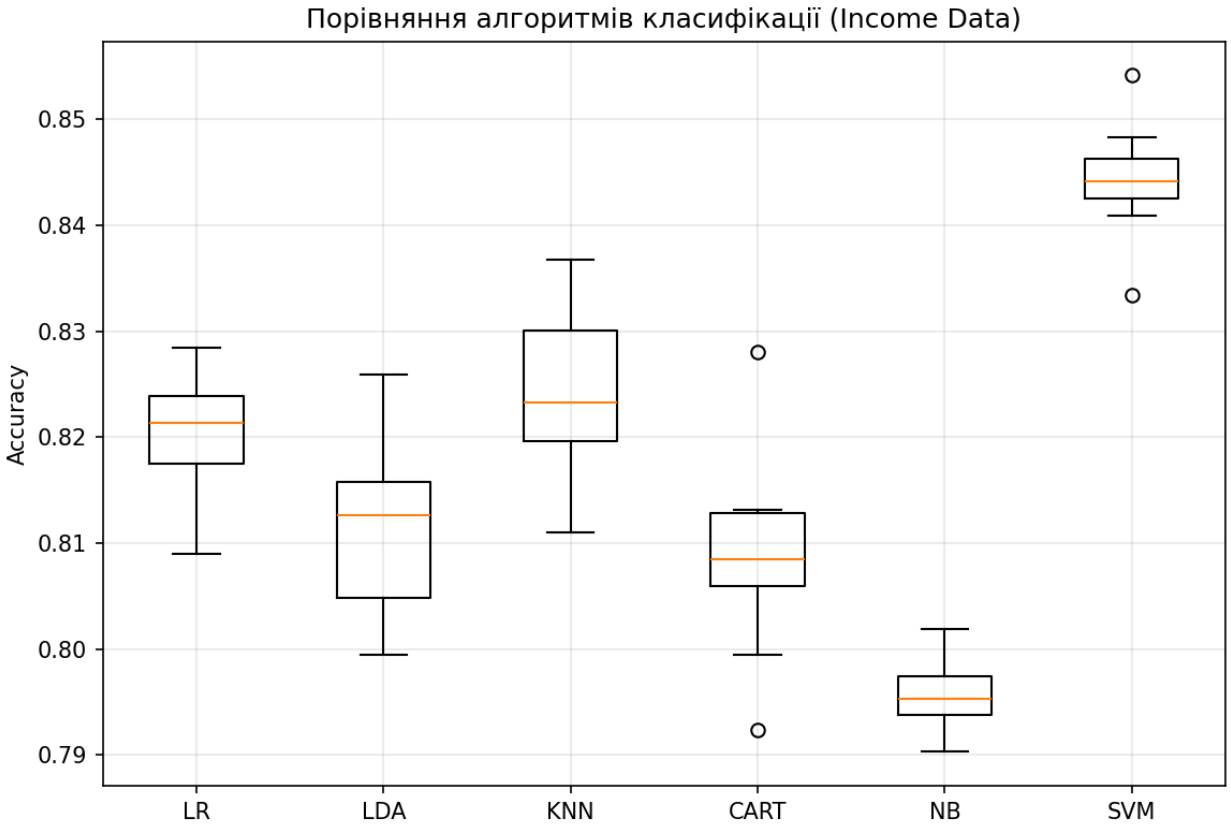

Рис.2.10.Результат виконання завдання

```
Дані успішно завантажено! Розмір набору: (32561, 15)

Після обробки: 24129 навчальних та 6033 тестових прикладів

Порівняння якості моделей (10-кратна стратифікована крос-валідація

  LR: 0.8202 (+/- 0.0058)
 LDA: 0.8113 (+/- 0.0077)
 KNN: 0.8244 (+/- 0.0074)
CART: 0.8088 (+/- 0.0089)
  NB: 0.7959 (+/- 0.0032)
 SVM: 0.8442 (+/- 0.0050)

Найкраща модель за середньою точністю: SVM (0.8442)

Оцінка на тестовій вибірці:
Точність (accuracy): 0.8410

Матриця помилок:
[[4251  280]
 [ 679  823]]

Звіт про класифікацію:
              precision    recall  f1-score   support

       <=50K       0.86      0.94      0.90      4531
        >50K       0.75      0.55      0.63      1502

    accuracy                           0.84      6033
   macro avg       0.80      0.74      0.77      6033
weighted avg       0.83      0.84      0.83      6033
```

Рис.2.11.Результат виконання завдання

Усі досліджені алгоритми показали прийнятну якість класифікації (понад 79%), проте SVM з гаусовим ядром забезпечив найкращий баланс між точністю, стабільністю та узагальнюючою здатністю. Для подальшого покращення якості класифікації класу ">50K" рекомендується застосувати техніки балансування класів або налаштувати порогові значення класифікації з урахуванням важливості правильного розпізнавання високих доходів.

Завдання 2.5. Класифікація даних лінійним класифікатором Ridge

Лістинг LR_2_task5

```
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
sns.set()
import matplotlib.pyplot as plt

print("Класифікація Ridge\n")

iris = load_iris()
X, y = iris.data, iris.target

print("Дані завантажено:")
print(f"  Кількість зразків: {X.shape[0]}")
print(f"  Кількість ознак: {X.shape[1]}")
print(f"  Класи: {iris.target_names}\n")

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=0)

print(f"Навчальний набір: {X_train.shape[0]} зразків")
print(f"Тестовий набір: {X_test.shape[0]} зразків\n")

print("Налаштування класифікатора Ridge:")
print("  - tol=1e-2 (допуск для критерію зупинки)")
print("  - solver='sag' (стохастичний градієнтний спуск)\n")

clf = RidgeClassifier(tol=1e-2, solver="sag")
clf.fit(X_train, y_train)

print("Навчання завершено.\n")

y_pred = clf.predict(X_test)

print("ПОКАЗНИКИ ЯКОСТІ КЛАСИФІКАЦІЇ\n")

accuracy = np.round(metrics.accuracy_score(y_test, y_pred), 4)
precision = np.round(metrics.precision_score(y_test, y_pred, average='weighted'), 4)
recall = np.round(metrics.recall_score(y_test, y_pred, average='weighted'), 4)
f1 = np.round(metrics.f1_score(y_test, y_pred, average='weighted'), 4)
cohen_kappa = np.round(metrics.cohen_kappa_score(y_test, y_pred), 4)
matthews = np.round(metrics.matthews_corrcoef(y_test, y_pred), 4)

print(f"Accuracy (Точність):           {accuracy}")
print(f"Precision (Точність):          {precision}")
```

```
print(f"Recall (Повнота):              {recall}")
print(f"F1 Score (F1-міра):            {f1}")
print(f"Cohen Kappa Score:             {cohen_kappa}")
print(f"Matthews Corrcoef:             {matthews}\n")

print("ЗВІТ ПРО КЛАСИФІКАЦІЮ\n")
print(metrics.classification_report(y_test, y_pred, target_names=iris.target_names))

print("\nМАТРИЦЯ ПОМИЛОК\n")
mat = confusion_matrix(y_test, y_pred)
print(mat)

plt.figure(figsize=(8, 6))
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.title('Confusion Matrix - Ridge Classifier')
plt.tight_layout()

plt.savefig("Confusion.jpg", dpi=150, bbox_inches='tight')
print("\nГрафік збережено як 'Confusion.jpg'\n")

plt.show()
```

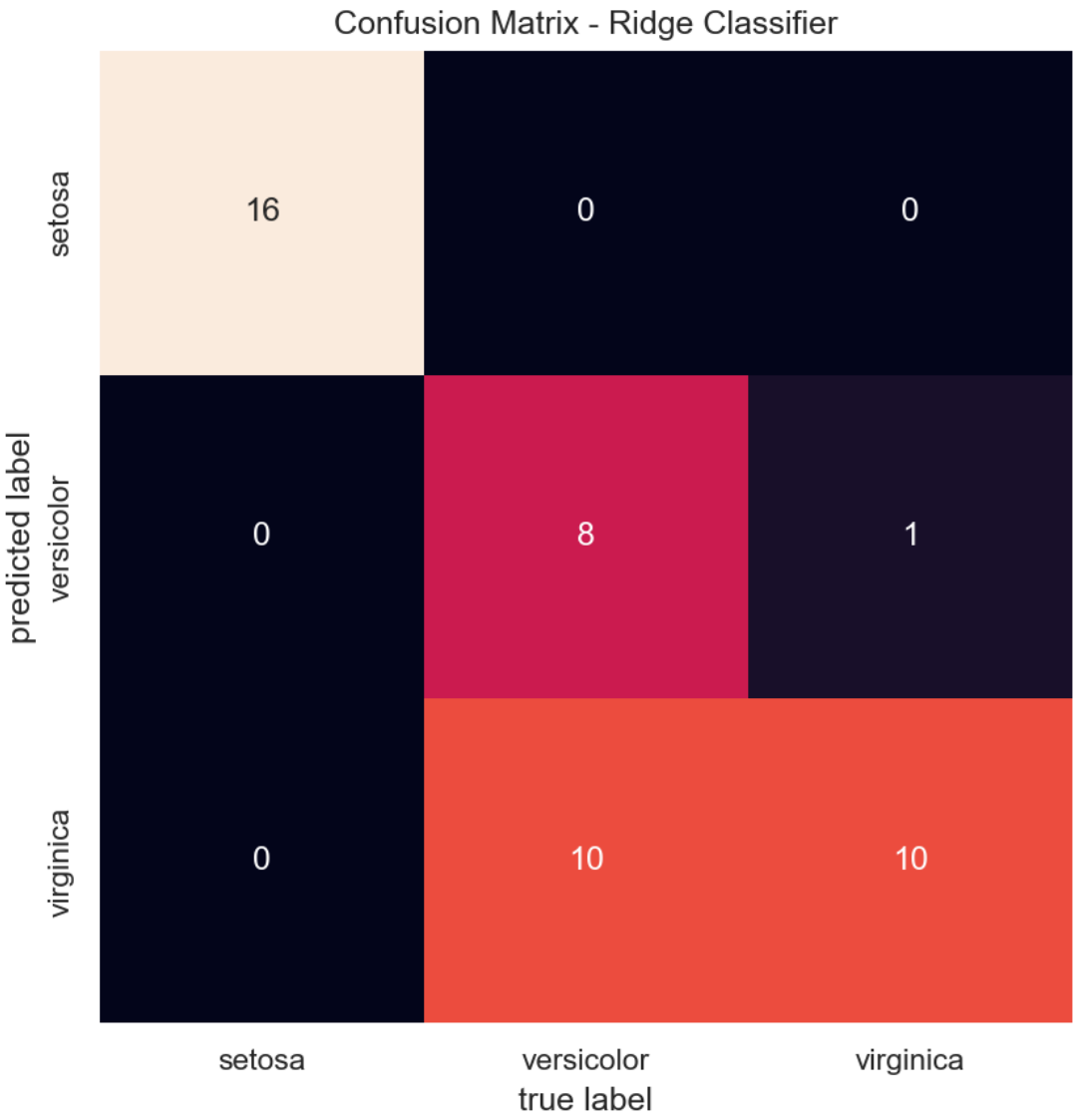

Рис.2.12.Результат виконання завдання

```
Класифікація Ridge

Дані завантажено:
  Кількість зразків: 150
  Кількість ознак: 4
  Класи: ['setosa' 'versicolor' 'virginica']

Навчальний набір: 105 зразків
Тестовий набір: 45 зразків

Налаштування класифікатора Ridge:
  - tol=1e-2 (допуск для критерію зупинки)
  - solver='sag' (стохастичний градієнтний спуск)

Навчання завершено.

ПОКАЗНИКИ ЯКОСТІ КЛАСИФІКАЦІЇ

Accuracy (Точність):           0.7556
Precision (Точність):          0.8333
Recall (Повнота):              0.7556
F1 Score (F1-міра):            0.7503
Cohen Kappa Score:             0.6431
Matthews Corrcoef:             0.6831

ЗВІТ ПРО КЛАСИФІКАЦІЮ

              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

[[16  0  0]
 [ 0  8 10]
 [ 0  1 10]]

Графік збережено як 'Confusion.jpg'
```

Рис.2.13.Результат виконання завдання

У завданні було досліджено лінійний Ridge-класифікатор для класифікації набору даних Iris. Ridge-класифікатор є методом регуляризованої лінійної регресії, адаптованим для задач класифікації.

Класифікатор було налаштовано з наступними параметрами: tolerance (tol=1e-2) визначає допуск для критерію зупинки алгоритму оптимізації, що дорівнює 0.01. Solver='sag' означає використання методу стохастичного усередненого градієнта (Stochastic Average Gradient), який є ефективним для великих наборів даних та забезпечує швидку збіжність алгоритму навчання.

За результатами класифікації Ridge-класифікатор показав такі показники якості: Accuracy становила 75.56%, Precision - 83.33%, Recall - 75.56%, F1-score -

75.03%. Ці результати значно нижчі порівняно з SVM з завдання 2.3 (96.67%), що свідчить про обмеження лінійних методів без kernel-трюку.
Ridge-класифікатор показав значно нижчі результати (75.56%) порівняно з SVM (96.67%) з завдання 2.3. Це пояснюється тим, що Ridge використовує лише лінійну регресію з L2-регуляризацією, тоді як SVM з kernel-трюком здатний моделювати нелінійні залежності. Ridge-класифікатор краще підходить для простих лінійно розділюваних задач або як базовий метод для швидкого прототипування.

Висновок: було проведено комплексне дослідження різних методів класифікації даних з використанням бібліотеки scikit-learn та мови програмування Python. Робота охоплювала як теоретичне вивчення алгоритмів, так і практичну реалізацію на реальних наборах даних.

Репозиторій: https://github.com/Svistaniuk/AIS