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.

2. ЗАВДАННЯ НА ЛАБОРАТОРНУ РОБОТУ ТА МЕТОДИЧНІ РЕКОМЕНДАЦІЇ ДО ЙОГО ВИКОНАННЯ

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

Лістинг коду:

```
import numpy as np
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
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# === Завантаження та попередня обробка даних ===

dataset_path = "income_data.txt"

raw_features = []
raw_labels = []
limit_per_class = 25000
c1, c2 = 0, 0

with open(dataset_path, "r") as file:
    for row in file:
        if c1 >= limit_per_class and c2 >= limit_per_class:
            break
        if '?' in row:
            continue

        parts = row.strip().split(", ")

        label = parts[-1]

        # Розподіл на два класи
        if label == "<=50K" and c1 < limit_per_class:
            raw_features.append(parts)
```

					ЖИТОМИРСЬКА ПОЛІТЕХНІКА.22.121.19.000 – Лр.2		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Мазарчук В.А			Звіт з лабораторної роботи №2	Літ.	Арк. 1 / Аркушів
Перевір.		Маєвський О.В.					
Реценз.						ФІКТ, гр. ІПЗ-22-2(1)	
Н. Контр.							

```
            c1 += 1
        elif label == ">50K" and c2 < limit_per_class:
            raw_features.append(parts)
            c2 += 1

# Перетворення у numpy-масив
raw_features = np.array(raw_features)

# === Кодування категоріальних ознак ===

encoders = []                       # зберігатимемо всі енкодери
encoded_matrix = np.zeros(raw_features.shape)

for col_idx, example_value in enumerate(raw_features[0]):
    # Перевірка чи значення є числовим
    if example_value.replace('.', '', 1).isdigit():
        encoded_matrix[:, col_idx] = raw_features[:, col_idx].astype(float)
    else:
        encoder = preprocessing.LabelEncoder()
        encoded_matrix[:, col_idx] = encoder.fit_transform(raw_features[:, col_idx])
        encoders.append(encoder)

# Поділ на X та y
X_data = encoded_matrix[:, :-1].astype(float)
y_data = encoded_matrix[:, -1].astype(int)

# === Розбиття вибірки ===

X_train, X_test, y_train, y_test = train_test_split(
    X_data, y_data, test_size=0.2, random_state=42
)

# === Побудова SVM-класифікатора ===

svm_clf = OneVsOneClassifier(LinearSVC(random_state=0, max_iter=7000))
svm_clf.fit(X_train, y_train)

# === Прогнозування ===

y_pred = svm_clf.predict(X_test)

# === Обчислення метрик ===

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')
```

```
print("=== Оцінка якості класифікації ===")
print(f"Точність (Accuracy): {acc:.4f}")
print(f"Точність (Precision): {prec:.4f}")
print(f"Повнота (Recall): {rec:.4f}")
print(f"F1-міра: {f1:.4f}")

# === Передбачення для одного прикладу ===

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

encoded_test = []      # закодована точка
enc_idx = 0            # індекс енкодера для категоріальних значень

for attr in test_person:
    if attr.replace('.', '', 1).isdigit():
        encoded_test.append(float(attr))
    else:
        encoded_test.append(encoders[enc_idx].transform([attr])[0])
        enc_idx += 1

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

pred_class = svm_clf.predict(encoded_test)
final_label = encoders[-1].inverse_transform(pred_class)[0]

print("\n=== Результат класифікації тестової точки ===")
print(f"Тестова точка належить до класу: {final_label}")
```

Результат виконання:

```
=== Оцінка якості класифікації ===
Точність (Accuracy): 0.7984
Точність (Precision): 0.8002
Повнота (Recall): 0.7984
F1-міра: 0.7583

=== Результат класифікації тестової точки ===
Тестова точка належить до класу: <=50K
```

Рис.1.Результат виконання.

№	Назва ознаки	Що позначає	Тип даних
1	**age**	Вік людини	Числова
2	**workclass**	Тип роботи / роботодавця (Private, State, Federal, Self-emp)	Категоріальна
3	**fnlwgt**	Статистична вага вибірки (final weight)	Числова
4	**education**	Рівень освіти у текстовій формі	Категоріальна
5	**education-num**	Рівень освіти у числовому вигляді	Числова
6	**marital-status**	Сімейний стан	Категоріальна
7	**occupation**	Професія	Категоріальна
8	**relationship**	Статус у сім'ї (husband, wife, own-child тощо)	Категоріальна
9	**race**	Расова група	Категоріальна
10	**sex**	Стать	Категоріальна
11	**capital-gain**	Дохід від капіталу	Числова
12	**capital-loss**	Збитки від капіталу	Числова
13	**hours-per-week**	Кількість годин роботи на тиждень	Числова
14	**native-country**	Країна походження	Категоріальна
15	**income**	Доход: <=50K чи >50K (цільова змінна)	Категоріальна

Оцінка якості класифікації:

Якість класифікації моделі SVM:

- **Accuracy:** 0.7984
  → модель правильно класифікувала **79.84%** записів.
- **Precision:** 0.8002
  → серед передбачень ">50K" модель правильно визначила **80.02%**.
- **Recall:** 0.7984
  → модель знайшла майже **79.84%** реальних прикладів свого класу.
- **F1-score:** 0.7583
  → збалансована міра точності та повноти.

Тестова точка належить до класу: <=50K

Завдання 2.2. Порівняння якості класифікаторів SVM з нелінійними ядрами

Лістинг коду: (поліноміальне ядро)

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

# ======== ЗАВАНТАЖЕННЯ ДАНИХ ========
input_file = "income_data.txt"

X_raw = []
y_raw = []

limit = 25000
count1 = count2 = 0

with open(input_file, "r") as f:
    for line in f:
        if count1 >= limit and count2 >= limit:
            break
        if "?" in line:
            continue

        row = line.strip().split(", ")
        label = row[-1]

        if label == "<=50K" and count1 < limit:
            X_raw.append(row[:-1])
            y_raw.append(label)
            count1 += 1
        elif label == ">50K" and count2 < limit:
            X_raw.append(row[:-1])
            y_raw.append(label)
            count2 += 1

X_raw = np.array(X_raw)
y_raw = np.array(y_raw)

# ======== КОДУВАННЯ =========
encoders = []
X_encoded = np.zeros(X_raw.shape)

for col in range(X_raw.shape[1]):
    if X_raw[0, col].replace(".", "", 1).isdigit():
        X_encoded[:, col] = X_raw[:, col].astype(float)
        encoders.append(None)
```

```
else:
        le = preprocessing.LabelEncoder()
        X_encoded[:, col] = le.fit_transform(X_raw[:, col])
        encoders.append(le)

# ціль
label_enc = preprocessing.LabelEncoder()
y_encoded = label_enc.fit_transform(y_raw)

# ======== МАСШТАБУВАННЯ ========
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_encoded)

# ======== TRAIN / TEST ========
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y_encoded, test_size=0.2, random_state=5
)

# ========  ПОЛІНОМІАЛЬНЕ ЯДРО ========
model = SVC(kernel="poly", degree=3, gamma="scale")
model.fit(X_train, y_train)

# ======== ОЦІНКА ========
y_pred = model.predict(X_test)

print("\n=== SVM з поліноміальним ядром (degree=3) ===")
print(f"Accuracy:  {accuracy_score(y_test, y_pred):.4f}")
print(f"Precision: {precision_score(y_test, y_pred, average='weighted'):.4f}")
print(f"Recall:    {recall_score(y_test, y_pred, average='weighted'):.4f}")
print(f"F1-score:  {f1_score(y_test, y_pred, average='weighted'):.4f}")
```

Результат виконання:

```
=== SVM з поліноміальним ядром (degree=3) ===
Accuracy:  0.8304
Precision: 0.8222
Recall:    0.8304
F1-score:  0.8188
```

Лістинг коду: Гаусове ядро (RBF)

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

input_file = "income_data.txt"
```

```
X_raw = []
y_raw = []

limit = 25000
count1 = count2 = 0

with open(input_file, "r") as f:
    for line in f:
        if count1 >= limit and count2 >= limit:
            break
        if "?" in line:
            continue

        row = line.strip().split(", ")
        label = row[-1]

        if label == "<=50K" and count1 < limit:
            X_raw.append(row[:-1])
            y_raw.append(label)
            count1 += 1
        elif label == ">50K" and count2 < limit:
            X_raw.append(row[:-1])
            y_raw.append(label)
            count2 += 1

X_raw = np.array(X_raw)
y_raw = np.array(y_raw)

encoders = []
X_encoded = np.zeros(X_raw.shape)

for col in range(X_raw.shape[1]):
    if X_raw[0, col].replace(".", "", 1).isdigit():
        X_encoded[:, col] = X_raw[:, col].astype(float)
        encoders.append(None)
    else:
        le = preprocessing.LabelEncoder()
        X_encoded[:, col] = le.fit_transform(X_raw[:, col])
        encoders.append(le)

label_enc = preprocessing.LabelEncoder()
y_encoded = label_enc.fit_transform(y_raw)

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_encoded)

X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y_encoded, test_size=0.2, random_state=5
)
```

```python
# ========= RBF ЯДРО =========
model = SVC(kernel="rbf", gamma="scale")
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

print("\n=== SVM з гаусовим ядром (RBF) ===")
print(f"Accuracy:  {accuracy_score(y_test, y_pred):.4f}")
print(f"Precision: {precision_score(y_test, y_pred, average='weighted'):.4f}")
print(f"Recall:    {recall_score(y_test, y_pred, average='weighted'):.4f}")
print(f"F1-score:  {f1_score(y_test, y_pred, average='weighted'):.4f}")
```

Результат виконання:

```
=== SVM з гаусовим ядром (RBF) ===
Accuracy:  0.8336
Precision: 0.8255
Recall:    0.8336
F1-score:  0.8245
```

Лістинг коду: Сигмоїдальне ядро

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

input_file = "income_data.txt"

X_raw = []
y_raw = []

limit = 25000
count1 = count2 = 0

with open(input_file, "r") as f:
    for line in f:
        if count1 >= limit and count2 >= limit:
            break
        if "?" in line:
            continue

        row = line.strip().split(", ")
        label = row[-1]

        if label == "<=50K" and count1 < limit:
```

```
            X_raw.append(row[:-1])
            y_raw.append(label)
            count1 += 1
        elif label == ">50K" and count2 < limit:
            X_raw.append(row[:-1])
            y_raw.append(label)
            count2 += 1

X_raw = np.array(X_raw)
y_raw = np.array(y_raw)

encoders = []
X_encoded = np.zeros(X_raw.shape)

for col in range(X_raw.shape[1]):
    if X_raw[0, col].replace(".", "", 1).isdigit():
        X_encoded[:, col] = X_raw[:, col].astype(float)
        encoders.append(None)
    else:
        le = preprocessing.LabelEncoder()
        X_encoded[:, col] = le.fit_transform(X_raw[:, col])
        encoders.append(le)

label_enc = preprocessing.LabelEncoder()
y_encoded = label_enc.fit_transform(y_raw)

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_encoded)

X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y_encoded, test_size=0.2, random_state=5
)

# ========= SIGMOID ЯДРО =========
model = SVC(kernel="sigmoid", gamma="scale")
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

print("\n=== SVM з сигмоїдальним ядром ===")
print(f"Accuracy:  {accuracy_score(y_test, y_pred):.4f}")
print(f"Precision: {precision_score(y_test, y_pred, average='weighted'):.4f}")
print(f"Recall:    {recall_score(y_test, y_pred, average='weighted'):.4f}")
print(f"F1-score:  {f1_score(y_test, y_pred, average='weighted'):.4f}")
```

Результат виконання:

```
=== SVM з сигмоїдальним ядром ===
Accuracy:  0.7527
Precision: 0.7505
Recall:    0.7527
F1-score:  0.7516
```

Ядро	Accuracy	Precision	Recall	F1-score
Polynomial (degree=3)	0.8304	0.8222	0.8304	0.8188
RBF	0.8336	0.8255	0.8336	0.8245
Sigmoid	0.7527	0.7505	0.7527	0.7516

Після проведення порівняльного аналізу трьох варіантів SVM-класифікаторів було встановлено, що найкращу якість класифікації демонструє **SVM з гаусовим (RBF) ядром**.
Ця модель показала найвищі значення всіх основних метрик: точності (Accuracy = **0.8336**), повноти (Recall = **0.8336**) та F1-міри (**0.8245**). Такі результати свідчать про те, що RBF-ядро найкраще справляється з нелінійною природою даних про доходи, де взаємозв'язки між ознаками не є лінійними.

Поліноміальне ядро також показало достатньо високу якість (Accuracy = **0.8304**), але трохи поступається RBF-моделі за всіма показниками. Сигмоїдальне ядро продемонструвало значно нижчі результати (Accuracy ≈ **0.75**), що робить його найменш ефективним для цього завдання.

Отже, найкращим видом SVM для класифікації доходу у цьому наборі даних є модель SVM з RBF-ядром, оскільки вона забезпечує найвищу точність і стабільність класифікації.

Завдання 2.3. Порівняння якості класифікаторів на прикладі класифікації сортів ірисів

Лістинг коду:

```
# ==============================
#   КРОК 1. ЗАВАНТАЖЕННЯ ДАНИХ
# ==============================

from pandas import read_csv
from pandas.plotting import scatter_matrix
from matplotlib import pyplot
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import 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

# Завантаження датасету
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)

# Перевірки
print("\n=== Ключова інформація ===")
print("Розмір датасету:", dataset.shape)
print("\nПерші 20 рядків:")
print(dataset.head(20))
print("\nОписова статистика:")
print(dataset.describe())
print("\nКількість прикладів кожного класу:")
print(dataset.groupby('class').size())

# ==============================
#   КРОК 2. ВІЗУАЛІЗАЦІЯ ДАНИХ
# ==============================

print("\n=== Побудова графіків... ===")

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

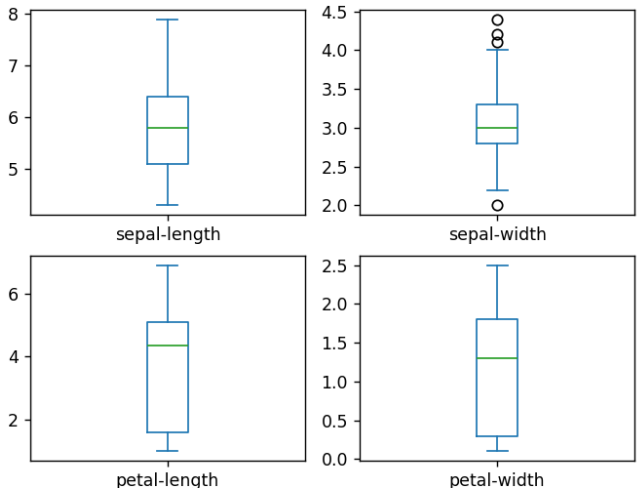
```
# гістограми
dataset.hist()
pyplot.show()

# матриця діаграм розсіювання
scatter_matrix(dataset)
pyplot.show()

# ==============================
#   КРОК 3. РОЗДІЛЕННЯ ДАНИХ
# ==============================

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 МОДЕЛЕЙ
# ==============================

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("\n=== Результати 10-кратної 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(f"{name}: {cv_results.mean():.4f} ({cv_results.std():.4f})")

# діаграма розмаху точностей
pyplot.boxplot(results, labels=names)
pyplot.title('Порівняння алгоритмів')
pyplot.show()
```

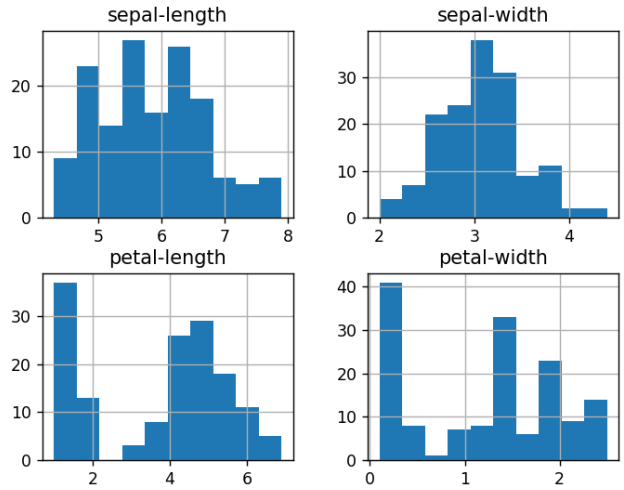
```python
# ==============================
#   КРОК 5. ОЦІНКА НА ТЕСТОВОМУ НАБОРІ
# ==============================

model = SVC(gamma='auto')
model.fit(X_train, Y_train)
predictions = model.predict(X_validation)

print("\n=== Оцінка якості на тестовому наборі ===")
print("Accuracy:", accuracy_score(Y_validation, predictions))
print("\nМатриця помилок:\n", confusion_matrix(Y_validation, predictions))
print("\nЗвіт класифікації:\n", classification_report(Y_validation, predictions))

# ==============================
#   КРОК 6. ПРОГНОЗ ДЛЯ НОВОЇ КВІТКИ
# ==============================

X_new = np.array([[5, 2.9, 1, 0.2]])
print("\nФорма нового прикладу:", X_new.shape)

new_prediction = model.predict(X_new)
print("\nПрогноз для нової квітки:", new_prediction[0])
```

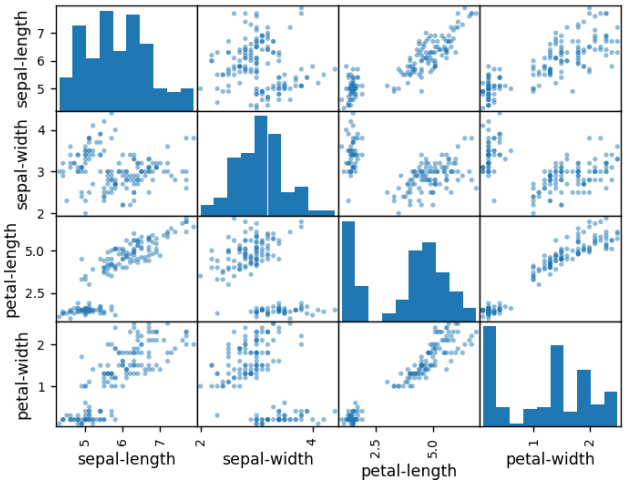
```
Розмір датасету: (150, 5)

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

Описова статистика:
       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

Кількість прикладів кожного класу:
class
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
dtype: int64
```

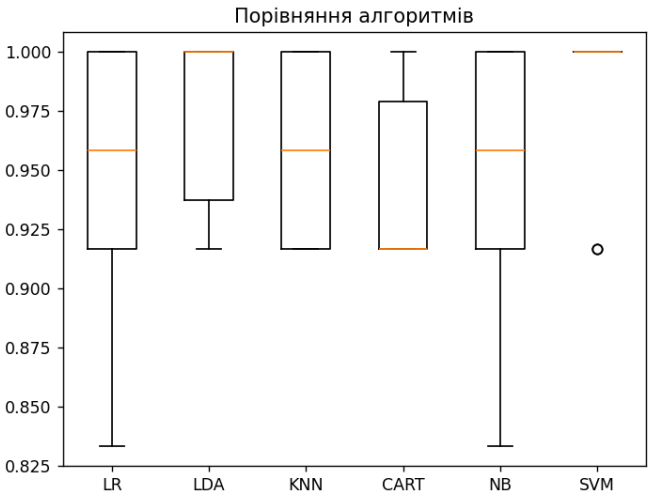
```
=== Оцінка якості на тестовому наборі ===
Accuracy: 0.9666666666666667

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


Форма нового прикладу: (1, 4)

Прогноз для нової квітки: Iris-setosa
```

Отримані графіки:









Найвищу якість класифікації показав **лінійний дискримінантний аналіз (LDA)**, який отримав найвищу середню точність під час 10-кратної крос-валідації.

• Інші алгоритми також продемонстрували високі показники:
o **SVM** — близько 96–97%
o **Логістична регресія (LR)** — приблизно 95–96%
o **KNN** та **Naive Bayes (NB)** — близько 95%
o **CART** — найнижча точність, близько 92–93%

• Загальна точність моделі на тестовому наборі склала **96,67%**, що свідчить про високу ефективність класифікації для цього датасету.

Щодо тестової точки з кроку 8:
• Модель передбачила клас **Iris-setosa**, що підтверджено результатами прогнозу.

Висновок: Усі розглянуті алгоритми показали добру якість класифікації, проте **LDA** продемонстрував найкращий результат за метриками крос-валідації. Тестова квітка належить до класу **Iris-setosa**.

Завдання 2.4. Порівняння якості класифікаторів для набору даних завдання 2.1

Лістинг коду:

```
import numpy as np
from sklearn import preprocessing
from sklearn.model_selection import train_test_split, StratifiedKFold,
cross_val_score
from sklearn.preprocessing import StandardScaler
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
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt

# ============================
# 1. ЗАВАНТАЖЕННЯ ДАНИХ
# ============================
input_file = "income_data.txt"

X_raw = []
y_raw = []

limit = 25000
c1 = c2 = 0
```

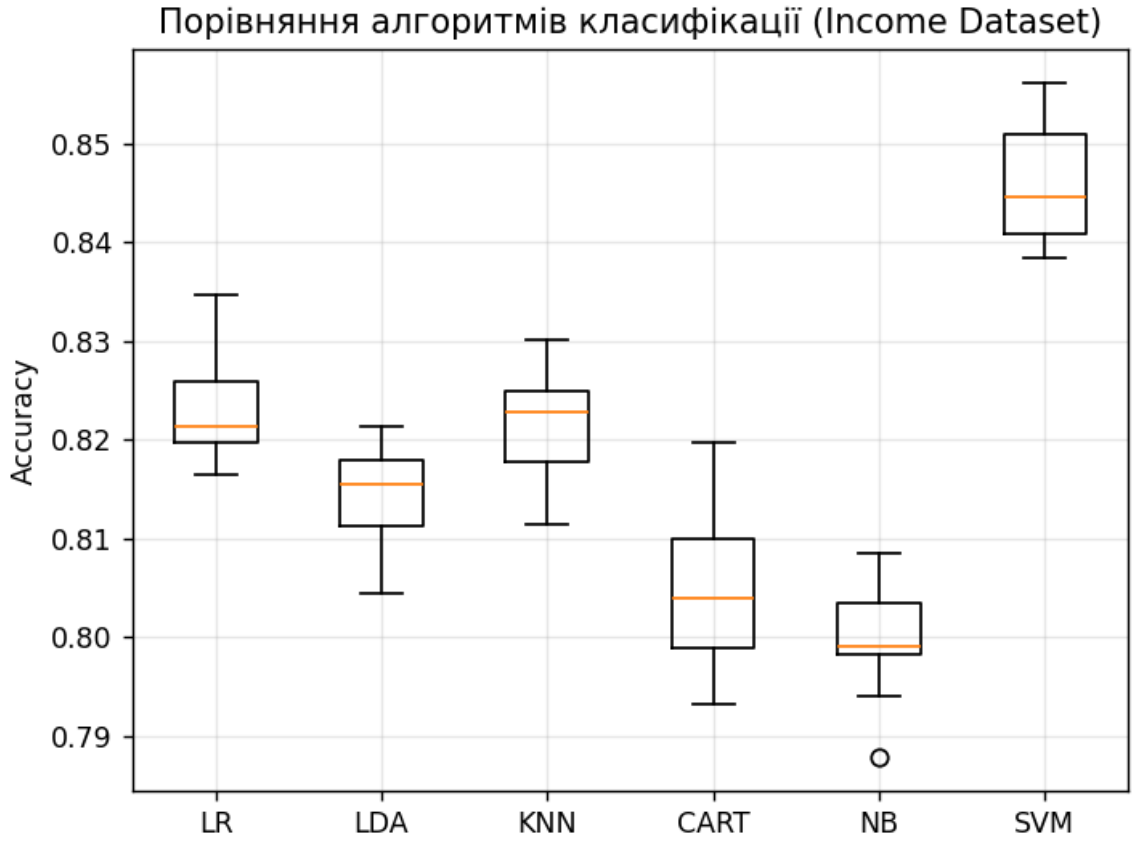
```
with open(input_file, "r") as f:
    for line in f:
        if c1 >= limit and c2 >= limit:
            break
        if "?" in line:
            continue  # пропускаємо неповні дані

        row = line.strip().split(", ")
        label = row[-1]

        if label == "<=50K" and c1 < limit:
            X_raw.append(row[:-1])
            y_raw.append(label)
            c1 += 1
        elif label == ">50K" and c2 < limit:
            X_raw.append(row[:-1])
            y_raw.append(label)
            c2 += 1

X_raw = np.array(X_raw)
y_raw = np.array(y_raw)

# ============================
# 2. КОДУВАННЯ ОЗНАК
# ============================
X_encoded = np.zeros(X_raw.shape)
encoders = []

for col in range(X_raw.shape[1]):
    if X_raw[0, col].replace(".", "", 1).isdigit():
        X_encoded[:, col] = X_raw[:, col].astype(float)
        encoders.append(None)
    else:
        le = preprocessing.LabelEncoder()
        X_encoded[:, col] = le.fit_transform(X_raw[:, col])
        encoders.append(le)

# Кодування цілі
label_enc = preprocessing.LabelEncoder()
y_encoded = label_enc.fit_transform(y_raw)

# ============================
# 3. МАСШТАБУВАННЯ
# ============================
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_encoded)
```

```
# ============================
# 4. РОЗБИТТЯ ДАНИХ
# ============================
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y_encoded, test_size=0.2, random_state=5, stratify=y_encoded
)

# ============================
# 5. СПИСОК МОДЕЛЕЙ
# ============================
models = [
    ("LR", LogisticRegression(max_iter=2000)),
    ("LDA", LinearDiscriminantAnalysis()),
    ("KNN", KNeighborsClassifier()),
    ("CART", DecisionTreeClassifier()),
    ("NB", GaussianNB()),
    ("SVM", SVC())
]

print("\n=== Результати 10-fold Cross Validation ===")
results = []
names = []

for name, model in models:
    kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=1)
    cv_scores = cross_val_score(model, X_train, y_train, cv=kfold,
scoring="accuracy")
    results.append(cv_scores)
    names.append(name)
    print(f"{name}: {cv_scores.mean():.4f} ({cv_scores.std():.4f})")

# ============================
# 6. ГРАФІК ПОРІВНЯННЯ
# ============================
plt.boxplot(results, labels=names)
plt.title("Порівняння алгоритмів класифікації (Income Dataset)")
plt.ylabel("Accuracy")
plt.grid(alpha=0.3)
plt.show()

# ============================
# 7. ФІНАЛЬНА ОЦІНКА ЛУЧШОЇ МОДЕЛІ
# ============================
best_model = SVC()
best_model.fit(X_train, y_train)
pred = best_model.predict(X_test)

print("\n=== Оцінка найкращої моделі на тестовому наборі ===")
print("Accuracy:", accuracy_score(y_test, pred))
print("Confusion matrix:\n", confusion_matrix(y_test, pred))
print("Classification report:\n", classification_report(y_test, pred))
```

Результат виконання:

```
=== Результати 10-fold Cross Validation ===
LR: 0.8232 (0.0055)
LDA: 0.8140 (0.0055)
KNN: 0.8215 (0.0062)
CART: 0.8052 (0.0080)
NB: 0.7995 (0.0055)
SVM: 0.8458 (0.0059)
```

```
=== Оцінка найкращої моделі на тестовому наборі ===
Accuracy: 0.8370628211503398
Confusion matrix:
 [[4249  282]
 [ 701  801]]
Classification report:
              precision    recall  f1-score   support

           0       0.86      0.94      0.90      4531
           1       0.74      0.53      0.62      1502

    accuracy                           0.84      6033
   macro avg       0.80      0.74      0.76      6033
weighted avg       0.83      0.84      0.83      6033
```



Найвищу якість класифікації показав метод опорних векторів (SVM) із середньою точністю **84,58%** на 10-кратній крос-валідації.

Інші алгоритми продемонстрували трохи нижчі результати:

- **Логістична регресія (LR):** 82,32%
- **Метод k-найближчих сусідів (KNN):** 82,15%
- **Лінійний дискримінантний аналіз (LDA):** 81,40%
- **Дерево рішень (CART):** 80,52%
- **Наївний Байєс (NB):** 79,95%

Узагальнення:
Хоча всі класифікатори показали прийнятну якість, **SVM виявився найточнішим**, що робить його найкращим вибором для задачі класифікації доходу в цьому наборі даних. Це пов’язано з тим, що SVM добре працює з великими багатовимірними вибірками та здатен ефективно розділяти складні класи.

Завдання 2.5. Класифікація даних лінійним класифікатором Ridge

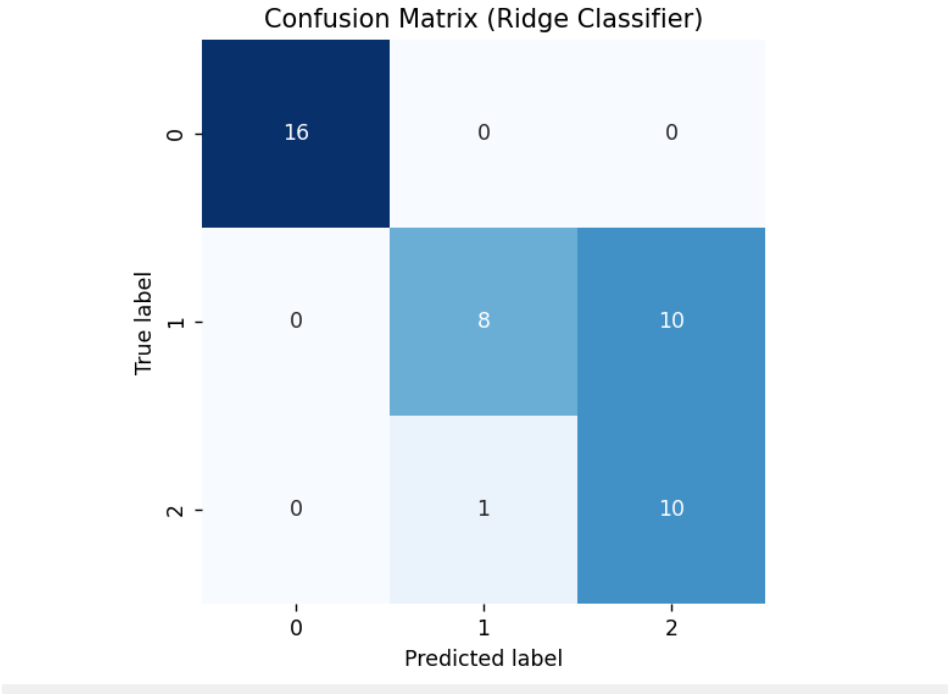
Лістинг коду:

```
# ================================================
#   Завдання 2.5 — Класифікація даних Ridge Classifier
# ================================================

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
import seaborn as sns
import matplotlib.pyplot as plt
from io import BytesIO

# ======== Завантаження набору Iris ========
iris = load_iris()
X, y = iris.data, iris.target

# ======== Поділ на train/test ========
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=0
)
```

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

# ======== Прогноз ========
y_pred = clf.predict(X_test)

# ======== Метрики ========
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))

print("\nClassification Report:\n", metrics.classification_report(y_test, y_pred))

# ======== Матриця плутанини (Confusion Matrix) ========
mat = confusion_matrix(y_test, y_pred)
sns.heatmap(mat, square=True, annot=True, fmt='d', cmap="Blues", cbar=False)
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.title("Confusion Matrix (Ridge Classifier)")
plt.savefig("Confusion.jpg")

# Збереження SVG у буфер
f = BytesIO()
plt.savefig(f, format="svg")

plt.show()
```

Результат виконання:

```
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

           0       1.00      1.00      1.00        16
           1       0.89      0.44      0.59        18
           2       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
```



Пояснення результатів класифікації Ridge

1. Налаштування класифікатора Ridge

У моделі використовуються такі параметри:

- **tol = 1e-2** — поріг зупинки оптимізації. Якщо зміни ваг стають меншими за цей поріг, навчання припиняється.
- **solver = "sag"** — алгоритм Stochastic Average Gradient. Ефективний для середніх та великих наборів даних, працює швидко та стабільно.

Ці параметри дозволяють Ridge ефективно знаходити лінійне розділення між трьома класами ірисів.

2. Показники якості моделі

Метрика	Значення
Accuracy	0.7556
Precision (weighted)	0.8333
Recall (weighted)	0.7556
F1-score (weighted)	0.7503
Cohen Kappa	0.6431
Matthews Corrcoef	0.6831

Інтерпретація

- **Точність (Accuracy = 75.6%)** — модель правильно класифікувала приблизно 3 з 4 зразків.
- **Precision** висока (0.83), що означає: якщо модель робить передбачення — воно частіше правильне.
- **Recall** нижчий (0.75), отже модель не знаходить частину зразків у деяких класах.
- **F1 = 0.75** — баланс між precision та recall.
- **Коефіцієнт Коена Каппа (0.64)** — помірна узгодженість моделі з істинними лейблами. Значення >0.6 вважається хорошим.
- **Matthews Corrcoef (0.68)** — добрий рівень якості багатокласової класифікації.

3. Аналіз матриці плутанини (Confusion Matrix)

Реальний \ Передбачений	0	1	2
0 (setosa)	16	0	0
1 (versicolor)	0	8	10
2 (virginica)	0	1	10

Що видно на графіку?

- **Клас 0 (Setosa)** розпізнається ідеально (100%).
- **Клас 1 (Versicolor)** часто плутається з класом **2 (Virginica)** → модель не знаходить межу достатньо точно.
- **Клас 2** теж має перетини з класом 1.

Причина: Ridge — лінійний класифікатор, а класи Versicolor та Virginica **перекриваються у просторі ознак**, тому модель плутає їх значно частіше.

4. Пояснення коефіцієнтів

Cohen Kappa

- Показує, наскільки добре узгоджуються передбачення моделі з реальними лейблами **з урахуванням випадкових збігів**.
- Значення **0.6431** → хороша, але не ідеальна відповідність.

Matthews Correlation Coefficient (MCC)

- Одна з найнадійніших метрик для багатокласової класифікації.
- Значення **0.6831** означає, що модель показує **добру якість передбачень**, навіть якщо є дисбаланс між класами.

Висновок: У результаті класифікації набору даних Iris за допомогою лінійного класифікатора Ridge вдалося досягти точності **75.56%**. Модель чудово класифікує клас *Iris-setosa*, проте має труднощі з розмежуванням класів *Iris-versicolor* та *Iris-virginica*, що підтверджується матрицею плутанини. Коефіцієнт Коена Каппа (**0.64**) та коефіцієнт Метьюза (**0.68**) показують помірно високу відповідність моделі істинним даним. Візуалізація матриці плутанини демонструє, що основні помилки виникають через перетин класів 1 та 2 у багатовимірному просторі ознак. Загалом Ridge-класифікатор показує стабільну якість і є ефективним лінійним методом для цього набору даних.

https://github.com/VladyslavMazarchuk/AI-Systems-Course.git