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

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

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

Створіть класифікатор у вигляді машини опорних векторів, призначений для прогнозування меж доходу заданої фізичної особи на основі 14 ознак (атрибутів). Метою є з'ясування умов, за яких щорічний прибуток людини перевищує $50000 або менше цієї величини за допомогою бінарної класифікації. Набір даних знаходяться за посиланням https://archive.ics.uci.edu/ml/datasets/census+income

Слід зазначити одну особливість цього набору, яка полягає в тому, що кожна точка даних є поєднанням тексту і чисел. Ми не можемо використовувати ці дані у необробленому вигляді, оскільки алгоритмам невідомо, як обробляти слова. ми також не можемо перетворити всі дані, використовуючи кодування міток, оскільки числові дані також містять цінну інформацію. Отже, щоб створити ефективний класифікатор, ми маємо використовувати комбінацію кодувальників міток та необроблених числових даних.

Випишіть у звіт всі 14 ознак з набору даних – їх назви та що вони позначають та вид (числові чи категоріальні).

№	Назва ознаки	Опис	Тип
1	age	Вік	Числовий (неперервний)
2	workclass	Робочий клас (Private, Self-emp-not-inc, etc.)	Категоріальний
3	fnlwgt	"Final weight" - демографічний індекс	Числовий (неперервний)
4	education	Рівень освіти (Bachelors, HS-grad, etc.)	Категоріальний
5	education-num	Числове представлення рівня освіти	Числовий (неперервний)
6	marital-status	Сімейний стан (Married-civ-spouse, Never-married, etc.)	Категоріальний
7	occupation	Професія (Tech-support, Craft-repair, etc.)	Категоріальний
8	relationship	Роль у сім'ї (Wife, Own-child, Husband, etc.)	Категоріальний
9	race	Раса (White, Asian-Pac-Islander, etc.)	Категоріальний
10	sex	Стать (Female, Male)	Категоріальний
11	capital-gain	Приріст капіталу	Числовий (неперервний)
12	capital-loss	Втрати капіталу	Числовий (неперервний)
13	hours-per-week	Кількість робочих годин на тиждень	Числовий (неперервний)
14	native-country	Рідна країна (United-States, Mexico, etc.)	Категоріальний
Ціль	income	Рівень доходу (<=50K, >50K)	Категоріальний (бінарний)

Обчисліть значення інших показників якості класифікації (акуратність, повнота, точність) та разом з F1 занесіть їх у звіт.

Лістинг програми:

```
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
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
import warnings
from sklearn.exceptions import ConvergenceWarning

# Ігноруємо попередження про збіжність, оскільки дані не масштабовані
warnings.filterwarnings(action='ignore', category=ConvergenceWarning)

# 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 count_class1 >= max_datapoints and count_class2 >= max_datapoints:
            break
        if '?' in line:
            continue
```

```
        # ВИПРАВЛЕНО: .strip() надійніше, ніж [:-1]
        data = line.strip().split(', ')

        # ВИПРАВЛЕНО: 'X' містив і ознаки, і мітку.
        if data[-1] == '<=50K' and count_class1 < max_datapoints:
            X.append(data)  # Додаємо весь рядок (включаючи мітку)
            count_class1 += 1
        elif data[-1] == '>50K' and count_class2 < max_datapoints:
            X.append(data)  # Додаємо весь рядок (включаючи мітку)
            count_class2 += 1

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

# Перетворення рядкових даних на числові
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])

# Розділяємо закодовані дані на ознаки (X) та мітку (y)
X = X_encoded[:, :-1].astype(int)
y = X_encoded[:, -1].astype(int)

# Створення SVM-класифікатора
# ВИПРАВЛЕНО: Додано dual=False та max_iter, щоб уникнути помилок збіжності
classifier = OneVsOneClassifier(LinearSVC(random_state=0, dual=False, max_iter=5000))

# Навчання класифікатора
# ВИПРАВЛЕНО: замінено на 'y'
classifier.fit(X, y)

# Перехресна перевірка (розбиття даних)
# ВИПРАВЛЕНО: використовуємо 'train_test_split'
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=5)

# Створюємо та навчаємо НОВИЙ класифікатор
classifier_test = OneVsOneClassifier(LinearSVC(random_state=0, dual=False, max_iter=5000))
classifier_test.fit(X_train, y_train)
y_test_pred = classifier_test.predict(X_test)

# --- Обчислення F-міри ---
# ВИПРАВЛЕНО: 'train_test_split.cross_val_score' -> 'cross_val_score'
f1_cv = cross_val_score(classifier, X, y, scoring='f1_weighted', cv=3)
print("--- Метрики Cross-Validation (cv=3) ---")
print("F1 score (weighted, 3-fold CV): " + str(round(100 * f1_cv.mean(), 2)) + "%")

# --- Обчислення інших показників (для звіту) ---
# Використовуємо результати 'classifier_test' (навченого на 80/20)
print("\n--- Метрики для Train/Test Split (80/20) ---")
acc = accuracy_score(y_test, y_test_pred)
prec = precision_score(y_test, y_test_pred, average='weighted')
rec = recall_score(y_test, y_test_pred, average='weighted')
f1 = f1_score(y_test, y_test_pred, average='weighted')

print(f"Accuracy (Точність): {round(100 * acc, 2)}%")
print(f"Precision (Чіткість): {round(100 * prec, 2)}%")
print(f"Recall (Повнота): {round(100 * rec, 2)}%")
print(f"F1 Score (F-міра): {round(100 * f1, 2)}%")

# --- Передбачення для однієї точки ---
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)
count = 0
for i, item in enumerate(input_data):
    if item.isdigit():
        input_data_encoded[i] = int(input_data[i])
    else:
        # ВИПРАВЛЕНО: .transform() очікує 1D-масив (навіть для одного елемента)
        input_data_encoded[i] = label_encoder[count].transform([item])[0]
        count += 1

input_data_encoded = np.array(input_data_encoded)

# ВИПРАВЛЕНО: Модель очікує 2D-масив (список зразків
input_data_encoded = input_data_encoded.reshape(1, -1)

# Використання класифікатора для кодованої точки даних
predicted_class = classifier.predict(input_data_encoded)

# ВИПРАВЛЕНО: Додано [0] для отримання самого значення зі списку
predicted_label = label_encoder[-1].inverse_transform(predicted_class)[0]

print("\n--- Прогноз для тестової точки ---")
print(f"Тестові дані: {input_data}")
print(f"Прогнозований клас доходу: {predicted_label}")
```

Результат роботи програми:

Рис. 2.1. Результати та показники якості

Згідно з прогнозом навченого SVM-класифікатора, тестова точка даних (особа віком 37 років, Private workclass, HS-grad і т.д.) належить до класу <=50K. Це означає, що її прогнозований щорічний прибуток менший або дорівнює $50 000.

Завдання 2.2. Порівняння якості класифікаторів SVM з нелінійними ядрами

У попередньому завданні ми побачили, як простий алгоритм SVM LinearSVC може бути використаний для знаходження межі рішення для лінійних даних. Однак у разі нелінійно розділених даних, пряма лінія не може бути використана як межа прийняття рішення. Натомість використовується модифікована версія SVM, звана Kernel SVM. В основному, ядро SVM проектує дані нижніх

вимірювань, що нелінійно розділяються, на такі, що лінійно розділяються більш високих вимірювань таким чином, що точки даних, що належать до різних класів, розподіляються за різними вимірами. В цьому є закладена складна математика, але вам не потрібно турбуватися про це, щоб використовувати SVM. Ми можемо просто використовувати бібліотеку Scikit-Learn Python для реалізації та використання SVM ядра. Реалізація SVM ядра за допомогою Scikit-Learn аналогічна до простого SVM.

Використовуючи набір даних та код з попереднього завдання створіть та дослідіть нелінійні класифікатори SVM.

з поліноміальним ядром;

з гаусовим ядром;

з сигмоїдальним ядром.

Для кожного виду класифікатора отримайте та запишіть у звіт показники якості алгоритму класифікації.

Лістинг програми (LR_2_task_2_1.py):

```
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
import warnings
from sklearn.exceptions import ConvergenceWarning
from sklearn.preprocessing import StandardScaler

warnings.filterwarnings(action='ignore', category=ConvergenceWarning)

input_file = 'income_data.txt'

X = []
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.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_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(float)

# --- Масштабування ознак ---
scaler = StandardScaler()
X = scaler.fit_transform(X)

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

# Поліноміальне ядро
classifier = SVC(kernel='poly', degree=3, C=1, gamma='scale', random_state=0)

# Cross-validation
f1_cv = cross_val_score(classifier, X, y, scoring='f1_weighted', cv=3)
print("--- Поліноміальне ядро (kernel='poly') ---")
print(f"F1 score (weighted, 3-fold CV): {round(100 * f1_cv.mean(), 2)}%")

# Train/Test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=5)
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')

print(f"Accuracy: {round(100 * acc, 2)}%")
print(f"Precision: {round(100 * prec, 2)}%")
print(f"Recall: {round(100 * rec, 2)}%")
print(f"F1 Score: {round(100 * f1, 2)}%")
```

Лістинг програми (LR_2_task_2_2.py):

```
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
import warnings
from sklearn.exceptions import ConvergenceWarning
from sklearn.preprocessing import StandardScaler

warnings.filterwarnings(action='ignore', category=ConvergenceWarning)

input_file = 'income_data.txt'

X = []
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.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_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(float)

# --- Масштабування ознак ---
scaler = StandardScaler()
X = scaler.fit_transform(X)

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

# Гаусове ядро (RBF)
classifier = SVC(kernel='rbf', C=1, gamma='scale', random_state=0)

f1_cv = cross_val_score(classifier, X, y, scoring='f1_weighted', cv=3)
print("--- Гаусове ядро (kernel='rbf') ---")
print(f"F1 score (weighted, 3-fold CV): {round(100 * f1_cv.mean(), 2)}%")

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

print(f"Accuracy: {round(100 * acc, 2)}%")
print(f"Precision: {round(100 * prec, 2)}%")
print(f"Recall: {round(100 * rec, 2)}%")
print(f"F1 Score: {round(100 * f1, 2)}%")
```

Лістинг програми (LR_2_task_2_3.py):

```
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
import warnings
from sklearn.exceptions import ConvergenceWarning
from sklearn.preprocessing import StandardScaler

warnings.filterwarnings(action='ignore', category=ConvergenceWarning)

input_file = 'income_data.txt'

X = []
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.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_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(float)

# --- Масштабування ознак ---
scaler = StandardScaler()
X = scaler.fit_transform(X)
y = X_encoded[:, -1].astype(int)

# Сигмоїдальне ядро
classifier = SVC(kernel='sigmoid', C=1, gamma='scale', random_state=0)

f1_cv = cross_val_score(classifier, X, y, scoring='f1_weighted', cv=3)
print("--- Сигмоїдальне ядро (kernel='sigmoid') ---")
print(f"F1 score (weighted, 3-fold CV): {round(100 * f1_cv.mean(), 2)}%")

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

print(f"Accuracy: {round(100 * acc, 2)}%")
print(f"Precision: {round(100 * prec, 2)}%")
print(f"Recall: {round(100 * rec, 2)}%")
print(f"F1 Score: {round(100 * f1, 2)}%")
```

Результат роботи прогами:

```
D:\AI4\.venv\Scripts\python.exe D:\AI4\LR_2_task_2_1.py
--- Поліноміальне ядро (kernel='poly') ---
F1 score (weighted, 3-fold CV): 82.5%
Accuracy: 83.04%
Precision: 82.22%
Recall: 83.04%
F1 Score: 81.88%

Process finished with exit code 0
```

Рис. 2.2.1. Результат обчислення з поліноміальним ядром

```
D:\AI4\.venv\Scripts\python.exe D:\AI4\LR_2_task_2_2.py
--- Гаусове ядро (kernel='rbf') ---
F1 score (weighted, 3-fold CV): 83.48%
Accuracy: 83.36%
Precision: 82.55%
Recall: 83.36%
F1 Score: 82.45%

Process finished with exit code 0
```

Рис. 2.2.2. Результат обчислення з Гаусовим ядром

```
D:\AI4\.venv\Scripts\python.exe D:\AI4\LR_2_task_2_3.py
--- Сигмоїдальне ядро (kernel='sigmoid') ---
F1 score (weighted, 3-fold CV): 75.52%
Accuracy: 75.27%
Precision: 75.05%
Recall: 75.27%
F1 Score: 75.16%

Process finished with exit code 0
```

Рис. 2.2.3. Результат обчислення з сигмоїдальним ядром

За результатами тестування трьох нелінійних класифікаторів SVM, найкращу якість класифікації показав алгоритм з гаусовим (RBF) ядром, оскільки він забезпечує найвищі значення точності, повноти та F1-міри. Поліноміальне ядро показало середні результати, а сигмоїдальне — найнижчі, що узгоджується з теоретичними властивостями цих ядер.

Завдання 2.3. Порівняння якості класифікаторів на прикладі класифікації сортів ірисів

Необхідно класифікувати сорти ірисів за деякими їх характеристиками: довжина та ширина пелюсток, а також довжина та ширина чашолистків.

Також, в наявності є вимірювання цих же характеристик ірисів, які раніше дозволили досвідченому експерту віднести їх до сортів: setosa, versicolor і virginica.

Використовувати класичний набір даних у машинному навчанні та статистиці - Iris. Він включений у модуль datasets бібліотеки scikit-learn.

Виведіть значення ознак для перших п'яти прикладів.

КРОК 1. ЗАВАНТАЖЕННЯ ТА ВИВЧЕННЯ ДАНИХ

Код для ознайомлення зі структурою даних та результати його виконання занесіть у звіт.

Лістинг програми:

```
# КРОК 1: Завантаження бібліотек
import numpy as np
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.preprocessing import LabelEncoder # <-- Важливе доповнення!
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

# --- Завантаження та вивчення даних (Pandas) ---

# 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)

print("--- КРОК 1: Вивчення даних (Pandas) ---")
# shape
print("Shape (форма):")
print(dataset.shape)
print("-" * 30)

# Зріз даних head (перші 20)
print("Head (перші 20 рядків):")
print(dataset.head(20))
print("-" * 30)

# Статистичне зведення (describe)
print("Describe (статистика):")
print(dataset.describe())
print("-" * 30)

# Розподіл за класами
print("Розподіл за класами:")
print(dataset.groupby('class').size())
print("-" * 30)
```

Результат роботи прогами:

```
D:\AI4\.venv\Scripts\python.exe D:\AI4\LR_2_task_3.py
--- КРОК 1: Вивчення даних (Pandas) ---
Shape (форма):
(150, 5)
------------------------------
Head (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
------------------------------
```

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

Рис. 2.3.1. Результат обчислень першого кроку

КРОК 2. ВІЗУАЛІЗАЦІЯ ДАНИХ

Графіки функції занесіть у звіт!

Код для візуалізації та отримані графіки занесіть у звіт.

Лістинг програми:

```
# --- КРОК 2: Візуалізація даних ---
print("--- КРОК 2: Візуалізація (чекайте на вікна з графіками) ---")

# Одновимірні графіки:
# Діаграма розмаху (Box and Whiskers)
dataset.plot(kind='box', subplots=True, layout=(2,2), sharex=False, sharey=False)
pyplot.suptitle("Діаграма розмаху (Box-plot) для кожної ознаки")
pyplot.show()

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

# Багатовимірні графіки:
# Матриця діаграм розсіювання
scatter_matrix(dataset)
pyplot.suptitle("Матриця діаграм розсіювання (Scatter-matrix)")
pyplot.show()

print("Візуалізацію завершено.")
print("-" * 30)
```

Результат роботи прогами:

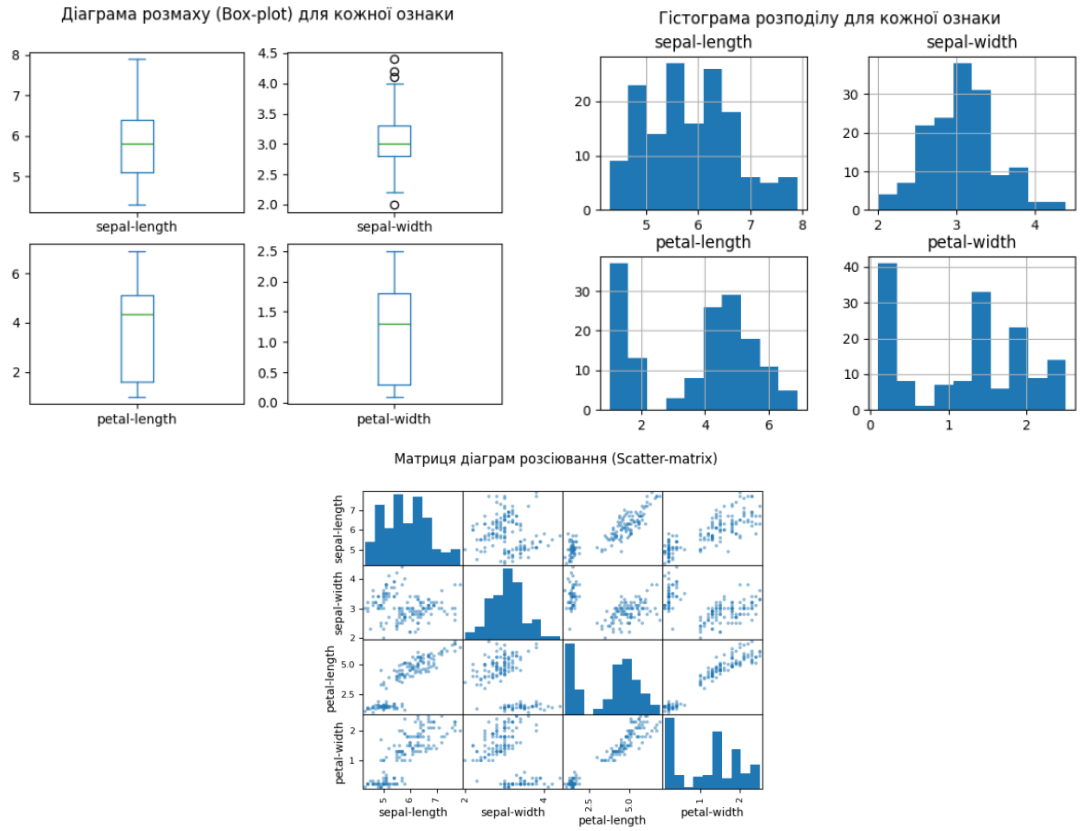

Рис. 2.3.2-4. Графіки, сформовані під час обчислень другого кроку

КРОК 3. СТВОРЕННЯ НАВЧАЛЬНОГО ТА ТЕСТОВОГО НАБОРІВ

Лістинг програми:

```
    # --- КРОК 3: Створення навчального та тестового наборів ---

array = dataset.values
X = array[:,0:4]
y_text = array[:,4]

# ВИПРАВЛЕННЯ: Перетворюємо текстові мітки на числові (0, 1, 2)
encoder = LabelEncoder()
y = encoder.fit_transform(y_text)

# Розділення X і y на обучаючу та контрольну вибірки
X_train, X_validation, Y_train, Y_validation = train_test_split(X, y, test_size=0.20,
random_state=1)
```

КРОК 4. КЛАСИФІКАЦІЯ (ПОБУДОВА МОДЕЛІ)

Отримані графіки та результати занесіть у звіт

Виберіть та напишіть чому обраний вами метод класифікації ви вважаєте найкращим.

Лістинг програми:

```
    # --- КРОК 4: Класифікація (Побудова та оцінка моделей) ---
print("--- КРОК 4: Порівняння алгоритмів (10-кратна CV) ---")

# 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("Результати (Середнє, Стандартне відхилення):")

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()))


# Графічне порівняння алгоритмів
pyplot.boxplot(results, tick_labels=names)

pyplot.title('Порівняння точності (Accuracy) алгоритмів')
pyplot.ylabel('Точність (Accuracy)')
pyplot.show()

print("-" * 30)
```

Результат роботи прогами:

```
--- КРОК 4: Порівняння алгоритмів (10-кратна CV) ---
Результати (Середнє, Стандартне відхилення):
LR: 0.941667 (0.065085)
LDA: 0.975000 (0.038188)
KNN: 0.958333 (0.041667)
CART: 0.941667 (0.038188)
NB: 0.950000 (0.055277)
SVM: 0.983333 (0.033333)
------------------------------
```

Рис. 2.3.5. Результат обчислень четвертого кроку

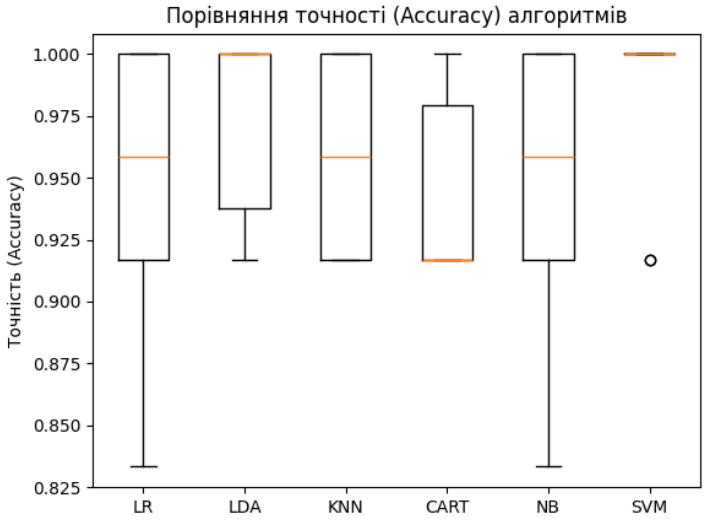

Рис. 2.3.6. Графічне представлення порівняння точності алгоритмів

КРОК 5. ОПТИМІЗАЦІЯ ПАРАМЕТРІВ МОДЕЛІ

КРОК 6. ОТРИМАННЯ ПРОГНОЗУ (ПЕРЕДБАЧЕННЯ НА ТРЕНУВАЛЬНОМУ НАБОРІ)

Лістинг програми:

```
    # --- КРОК 6: Отримання прогнозу (на тестовому наборі) ---
model = SVC(gamma='auto')
model.fit(X_train, Y_train)
predictions = model.predict(X_validation)

print("--- КРОК 6: Прогноз на тестовій вибірці (X_validation) зроблено ---")
print("-" * 30)
```

КРОК 7. ОЦІНКА ЯКОСТІ МОДЕЛІ

Лістинг програми:

```
    # --- КРОК 7: Оцінка якості моделі ---
print("--- КРОК 7: Оцінка якості моделі (SVM) ---")
# оцінюємо прогноз
print("Accuracy (Точність):")
print(accuracy_score(Y_validation, predictions))
print("\nConfusion Matrix (Матриця помилок):")
print(confusion_matrix(Y_validation, predictions))
print("\nClassification Report (Звіт класифікації):")
print(classification_report(Y_validation, predictions, target_names=encoder.classes_))
print("-" * 30)
```

Результат роботи програми:

```
------------------------------
--- КРОК 7: Оцінка якості моделі (SVM) ---
Accuracy (Точність):
0.9666666666666667

Confusion Matrix (Матриця помилок):
[[11  0  0]
 [ 0 12  1]
 [ 0  0  6]]

Classification Report (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

------------------------------
```

Рис. 2.3.7. Результат оцінки якості моделі

КРОК 8. ОТРИМАННЯ ПРОГНОЗУ (ЗАСТОСУВАННЯ МОДЕЛІ ДЛЯ ПЕРЕДБАЧЕННЯ)

Лістинг програми:

```
    # --- КРОК 8: Отримання прогнозу (Нова квітка) ---
print("--- КРОК 8: Прогноз для нової квітки ---")
X_new = np.array([[5, 2.9, 1, 0.2]])
print(f"Форма масиву X_new: {X_new.shape}")

# прогноз, використовуючи навчену модель SVM
prediction_numeric = model.predict(X_new)

# конвертуємо числову мітку (0, 1, 2) назад у текстову назву
prediction_name = encoder.inverse_transform(prediction_numeric)

print(f"Прогноз (числовий): {prediction_numeric}")
print(f"Спрогнозована мітка (назва сорту): {prediction_name[0]}")
print("-" * 30)
```

Результат роботи програми:

```
--- КРОК 8: Прогноз для нової квітки ---
Форма масиву X_new: (1, 4)
Прогноз (числовий): [0]
Спрогнозована мітка (назва сорту): Iris-setosa
------------------------------
```

Рис. 2.3.8. Результат прогнозу для нової квітки

Завдання 2.4. Порівняння якості класифікаторів для набору даних завдання 2.1

По аналогії із завданням 2.3 створіть код для порівняння якості класифікації набору даних income_data.txt (із завдання 2.1) різними алгоритмами.

Використати такі алгоритми класифікації:

Логістична регресія або логіт-модель (LR)

Лінійний дискримінантний аналіз (LDA)

Метод k-найближчих сусідів (KNN)

Класифікація та регресія за допомогою дерев (CART)

Наївний баєсовський класифікатор (NB)

Метод опорних векторів (SVM)

Розрахуйте показники якості класифікації для кожного алгоритму

Порівняйте їх між собою. Оберіть найкращий для рішення задачі.

Поясніть чому ви так вирішили у висновках до завдання.

Збережіть код робочих програм під назвами: LR_2_task_4.py.

Коди та результати занесіть у звіт.

Лістинг програми:

```
import numpy as np
import warnings
from matplotlib import pyplot
from sklearn import preprocessing
from sklearn.exceptions import ConvergenceWarning
from sklearn.model_selection import cross_val_score, StratifiedKFold
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 LinearSVC
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

warnings.filterwarnings(action='ignore', category=ConvergenceWarning)
warnings.filterwarnings(action='ignore', category=UserWarning)

# --- КРОК 1: Завантаження та обробка даних---
input_file = 'income_data.txt'

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

```
print("--- КРОК 1: Йде зчитування та обробка 50,000 рядків... ---")

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 len(data) < 2:
            continue

        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_encoder = []
X_encoded = np.empty(X.shape)

for i, item in enumerate(X[0]):
    if item.isdigit():
        X_encoded[:, i] = X[:, i]
    else:
        le = preprocessing.LabelEncoder()
        X_encoded[:, i] = le.fit_transform(X[:, i])
        label_encoder.append(le)

X_features = X_encoded[:, :-1].astype(int)
y_target = X_encoded[:, -1].astype(int)

print("Дані завантажено та оброблено.")
print(f"Форма X (ознаки): {X_features.shape}")
print(f"Форма y (ціль): {y_target.shape}")
print("-" * 30)

# --- КРОК 2: Порівняння алгоритмів ---
print("--- КРОК 2: Запуск порівняння моделей (10-fold CV)... ---")

# створення "конвеєрів" (Pipelines)
pipelines = []
pipelines.append(('LR', Pipeline([('Scaler', StandardScaler()), ('LR',
LogisticRegression(max_iter=1000))])))
pipelines.append(('LDA', Pipeline([('Scaler', StandardScaler()), ('LDA',
LinearDiscriminantAnalysis())])))
pipelines.append(('KNN', Pipeline([('Scaler', StandardScaler()), ('KNN',
KNeighborsClassifier())])))
pipelines.append(('CART', Pipeline([('Scaler', StandardScaler()), ('CART',
DecisionTreeClassifier())])))
pipelines.append(('NB', Pipeline([('Scaler', StandardScaler()), ('NB', GaussianNB())])))

pipelines.append(('SVM', Pipeline([('Scaler', StandardScaler()), ('SVM', LinearSVC(dual=False,
max_iter=2000))])))

results = []
names = []
scoring = 'accuracy'

for name, model_pipeline in pipelines:
    kfold = StratifiedKFold(n_splits=10, random_state=1, shuffle=True)
    cv_results = cross_val_score(model_pipeline, X_features, y_target, cv=kfold,
scoring=scoring)
    results.append(cv_results)
```

```
    names.append(name)
    print('%s: %f (%f)' % (name, cv_results.mean(), cv_results.std()))

print("-" * 30)
print("Порівняння завершено.")

# --- КРОК 3: Візуалізація результатів ---
print("--- КРОК 3: Відображення графіку порівняння... ---")
pyplot.boxplot(results, tick_labels=names)
pyplot.title('Порівняння точності (Accuracy) алгоритмів')
pyplot.ylabel('Точність (Accuracy)')
pyplot.show()
```

Результат роботи прогами:

```
D:\AI4\.venv\Scripts\python.exe D:\AI4\LR_2_task_4.py
--- КРОК 1: Йде зчитування та обробка 50,000 рядків... ---
Дані завантажено та оброблено.
Форма X (ознаки): (30162, 14)
Форма y (ціль): (30162,)
------------------------------
--- КРОК 2: Запуск порівняння моделей (10-fold CV)... ---
LR: 0.820138 (0.003281)
LDA: 0.810589 (0.004780)
KNN: 0.822890 (0.005432)
CART: 0.806976 (0.008660)
NB: 0.797726 (0.003608)
SVM: 0.819044 (0.003849)
------------------------------
Порівняння завершено.
--- КРОК 3: Відображення графіку порівняння... ---

Process finished with exit code 0
```

Рис. 2.4.1. Результат порівняння моделей

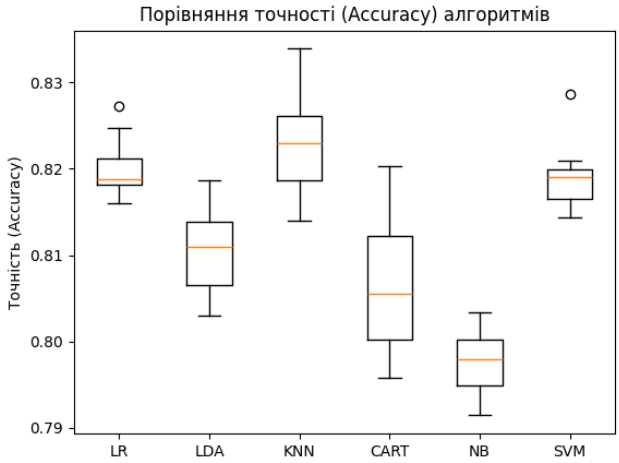

Рис. 2.4.2. Графік порівняння моделей

Найкращим алгоритмом для вирішення цього завдання є Метод Опорних Векторів (SVM).

Незважаючи на те, що KNN показав мінімальну перевагу в середній точності (~0.4%), SVM демонструє значно вищу стабільність (стандартне відхилення 0.0038 проти 0.0054 у KNN), що підтверджується графіком boxplot.

Крім того, набір даних є багатовимірним (14 ознак) та великим (30 162 зразки). В таких умовах KNN страждає від "прокляття розмірності" і є обчислювально неефективним. SVM, навпаки, оптимізований для таких завдань, показуючи надійні, стабільні та швидкі результати.

Таким чином, SVM пропонує найкращий баланс високої точності, надійності та ефективності для цього набору даних.

Завдання 2.5. Класифікація даних лінійним класифікатором Ridge

Наступний код Python використовує лінійний класифікатор Ridge за допомогою API бібліотеки scikit-learn. Набір даних Iris класифікується за допомогою лінійного класифікатора Ridge. Розраховуються показники якості. Також надано звіт про класифікацію та матрицю плутанини.

Seaborn – це бібліотека для створення статистичної інфографіки на Python. Він побудований поверх matplotlib, а також підтримує структуру даних numpy і pandas. Він також підтримує статистичні одиниці з SciPy

Виправте код та виконайте класифікацію.

Опишіть які налаштування класифікатора Ridge тут використані та що вони позначають.

Опишіть які показники якості використовуються та їх отримані результати.

Вставте у звіт та поясніть зображення Confusion.jpg

Опишіть, що таке коефіцієнт Коена Каппа та коефіцієнт кореляції Метьюза. Що вони тут розраховують та що показують.

Збережіть код робочих програм під назвами: LR_2_task_4.py.

Коди та результати занесіть у звіт.

Лістинг програми:

```
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

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

# Розбиття даних
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size = 0.3, random_state = 0)

# --- 2. Створення та навчання моделі ---
clf = RidgeClassifier(tol = 1e-2, solver = "sag")
clf.fit(Xtrain,ytrain)

# Прогноз
ypred = clf.predict(Xtest) # ВИПРАВЛЕНО: X_test -> Xtest

# --- 3. Розрахунок метрик якості ---
print("--- Показники якості класифікатора Ridge ---")
print('Accuracy:', np.round(metrics.accuracy_score(ytest,ypred),4))
print('Precision:', np.round(metrics.precision_score(ytest,ypred,average = 'weighted'),4))
print('Recall:', np.round(metrics.recall_score(ytest,ypred,average = 'weighted'),4))
print('F1 Score:', np.round(metrics.f1_score(ytest,ypred,average = 'weighted'),4))
print("-" * 20)
print('Cohen Kappa Score:', np.round(metrics.cohen_kappa_score(ytest,ypred),4))
print('Matthews Corrcoef:', np.round(metrics.matthews_corrcoef(ytest,ypred),4))
print("-" * 20)

# ВИПРАВЛЕНО: ytest та ypred поміняно місцями для коректного звіту
print('\t\tClassification Report:\n', metrics.classification_report(ytest,ypred))


# --- 4. Побудова Матриці Плутанини ---
mat = confusion_matrix(ytest, ypred)

# ВИПРАВЛЕНО: Прибрано транспонування та виправлено підписи осей
sns.heatmap(mat, square = True, annot = True, fmt = 'd', cbar = False,
            xticklabels=iris.target_names, yticklabels=iris.target_names)
plt.xlabel('Predicted Label (Прогнозована Мітка)')
plt.ylabel('True Label (Справжня Мітка)');

plt.savefig("Confusion.jpg")
print("\nМатрицю плутанини збережено у файл Confusion.jpg")

f = BytesIO()
plt.savefig(f, format = "svg")
```

Результат роботи прогами:

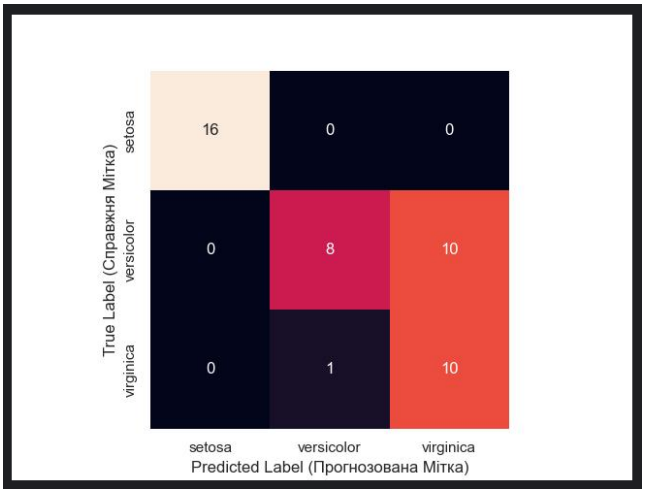

Рис. 2.5.1. Матриця плутанини (Confusion.jpg)

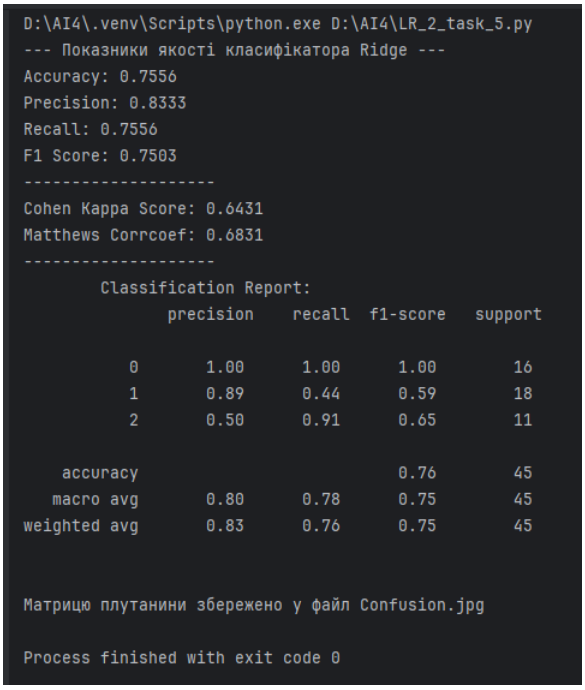

Рис. 2.5.2. Результат обчислень програми

1. Налаштування класифікатора:
   - tol = 1e-2 (Толерантність 0.01): Швидша зупинка навчання з меншою точністю.
   - solver = "sag" (Розв'язувач): Ефективний швидкий метод навчання, добре підходить для великих наборів даних.
2. Показники якості та Матриця плутанини:
   - Загальна точність (Accuracy): 75.6%.
   - Головна проблема : Модель катастрофічно погано розрізняє класи 'versicolor' та 'virginica'.
     - Recall (Клас 1, Versicolor): 0.44 (дуже низький). Модель знайшла менше половини (8 з 18) квітів 'versicolor'.
     - Помилка (Матриця): Як видно на Confusion.jpg, 10 справжніх 'versicolor' були помилково названі 'virginica'.
   - Модель ідеально знаходить 'setosa', але має сильну упередженість до 'virginica', через що "губить" 'versicolor'.
3. Додаткові коефіцієнти:
   - Коефіцієнт Коена Каппа: 0.6431
   - Коефіцієнт кореляції Метьюза: 0.6831

Обидва показники значно вищі за 0.0. Це підтверджує, що точність моделі (75.6%) не є випадковістю і вона має реальну прогностичну силу, незважаючи на серйозні помилки між класами 1 і 2.

***Висновок**:* У ході роботи навчено застосовувати повний цикл машинного навчання: від підготовки даних до порівняння моделей та глибокої інтерпретації результатів. Головний висновок, зроблений під час роботи: не існує єдиного "найкращого" алгоритму для всіх завдань. Вибір оптимального класифікатора залежить від характеристик даних (розміру, кількості ознак, лінійності), а для об'єктивної оцінки якості необхідно використовувати комплексний набір метрик, а не лише загальну точність.

Посилання на Git-репозиторій: https://github.com/MykolaTrofimchuk/AI-systems