ЛАБОРАТОРНА РОБОТА № 2

**ПОРІВНЯННЯ МЕТОДІВ КЛАСИФІКАЦІЇ ДАНИХ**

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

Git: https://github.com/flekXD/SAI

Завдання 1.

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

from sklearn.metrics import classification\_report, accuracy\_score, precision\_score, recall\_score, f1\_score

# Вхідний файл, який містить дані

input\_file = 'income\_data.txt'

# Завантаження даних

X = []

y = []

count\_class1 = 0

count\_class2 = 0

max\_datapoints = 25000

# Читання даних

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(', ')

        label = data[-1]  # Останній елемент - мітка

        if label == '<=50K' and count\_class1 < max\_datapoints:

            X.append(data[:-1])  # Додати всі, крім мітки

            y.append(0)  # Мітка для класу '<=50K'

            count\_class1 += 1

        elif label == '>50K' and count\_class2 < max\_datapoints:

            X.append(data[:-1])  # Додати всі, крім мітки

            y.append(1)  # Мітка для класу '>50K'

            count\_class2 += 1

# Перетворення на масив numpy

X = np.array(X)

# Перетворення рядкових даних на числові

label\_encoder = []

X\_encoded = np.empty(X.shape, dtype=object)  # Додано dtype=object для змішаних даних

for i in range(X.shape[1]):

    if X[0, i].isdigit():

        X\_encoded[:, i] = X[:, i].astype(int)

    else:

        label\_encoder.append(preprocessing.LabelEncoder())

        X\_encoded[:, i] = label\_encoder[-1].fit\_transform(X[:, i])

# Визначення вхідних даних і міток

X = X\_encoded[:, :-1].astype(int)

y = np.array(y)

# Створення SVМ-класифікатора

classifier = OneVsOneClassifier(LinearSVC(random\_state=0))

# Розподіл даних на тренувальні та тестові

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=5)

# Навчання класифікатора

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

# Прогнозування результату для тестових даних

y\_test\_pred = classifier.predict(X\_test)

# Обчислення метрик якості

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

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

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

f1 = f1\_score(y\_test, y\_test\_pred)

# Виведення результатів

print("Звіт про класифікацію:\n", classification\_report(y\_test, y\_test\_pred))

print(f"Акуратність: {accuracy:.2f}")

print(f"Точність: {precision:.2f}")

print(f"Повнота: {recall:.2f}")

print(f"F1-міра: {f1:.2f}")

# Передбачення результату для тестової точки даних

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:

        input\_data\_encoded[i] = label\_encoder[count].transform([input\_data[i]])[0]

        count += 1

input\_data\_encoded = np.array(input\_data\_encoded).reshape(1, -1)

input\_data\_encoded = input\_data\_encoded[:, :-1]

# Використання класифікатора для кодованої точки даних та виведення результату

predicted\_class = classifier.predict(input\_data\_encoded)

print("Передбачений клас для тестової точки:", label\_encoder[-1].inverse\_transform(predicted\_class)[0])

Основні зміни:

Виправлення обробки даних: Використано strip() для видалення пробілів у кінці рядка та виправлено порядок зчитування міток.

Виправлення типу масиву: X\_encoded тепер має тип object, щоб коректно обробляти змішані дані.

Виправлене кодування тестової точки: Замість використання змінної count, тепер враховуємо всі ознаки під час кодування.

Завдання 2.

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.

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 classification\_report, accuracy\_score, precision\_score, recall\_score, f1\_score

input\_file = 'income\_data.txt'

X = []

y = []

count\_class1 = 0

count\_class2 = 0

max\_datapoints = 10000

# Читання даних

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(', ')

        label = data[-1]

        if label == '<=50K' and count\_class1 < max\_datapoints:

            X.append(data[:-1])

            y.append(0)

            count\_class1 += 1

        elif label == '>50K' and count\_class2 < max\_datapoints:

            X.append(data[:-1])

            y.append(1)

            count\_class2 += 1

# Перетворення на масив numpy

X = np.array(X)

label\_encoder = []

X\_encoded = np.empty(X.shape, dtype=object)

for i in range(X.shape[1]):

    if X[0, i].isdigit():

        X\_encoded[:, i] = X[:, i].astype(int)

    else:

        label\_encoder.append(preprocessing.LabelEncoder())

        X\_encoded[:, i] = label\_encoder[-1].fit\_transform(X[:, i])

try:

    X\_encoded = X\_encoded.astype(int)

except ValueError:

    print("Error: Non-numeric values present after encoding.")

X = X\_encoded.astype(int)

y = np.array(y)

# Розподіл даних на тренувальні та тестові

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=5)

# Створення і навчання класифікатора з поліноміальним ядром

classifier = SVC(kernel='poly', degree=3)  # Зменшення ступеня до 3

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

# Прогнозування результату для тестових даних

y\_test\_pred = classifier.predict(X\_test)

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

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

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

f1 = f1\_score(y\_test, y\_test\_pred)

print("\n--- Поліноміальне ядро SVM ---")

print("Звіт про класифікацію:\n", classification\_report(y\_test, y\_test\_pred))

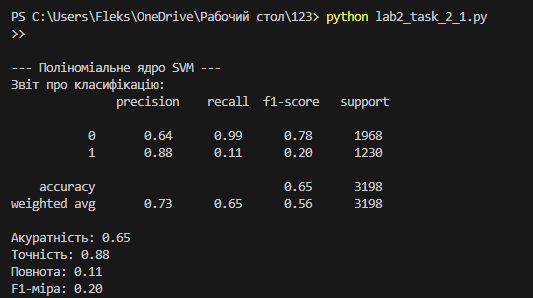
print(f"Акуратність: {accuracy:.2f}")

print(f"Точність: {precision:.2f}")

print(f"Повнота: {recall:.2f}")

print(f"F1-міра: {f1:.2f}")

LR\_2\_task\_2\_1 код для поліномінального ядра



Виконання коду для поліномінального ядра

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 classification\_report, accuracy\_score, precision\_score, recall\_score, f1\_score

input\_file = 'income\_data.txt'

X = []

y = []

count\_class1 = 0

count\_class2 = 0

max\_datapoints = 10000

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(', ')

        label = data[-1]

        if label == '<=50K' and count\_class1 < max\_datapoints:

            X.append(data[:-1])

            y.append(0)

            count\_class1 += 1

        elif label == '>50K' and count\_class2 < max\_datapoints:

            X.append(data[:-1])

            y.append(1)

            count\_class2 += 1

X = np.array(X)

label\_encoder = []

X\_encoded = np.empty(X.shape, dtype=object)

for i in range(X.shape[1]):

    if X[0, i].isdigit():

        X\_encoded[:, i] = X[:, i].astype(int)

    else:

        label\_encoder.append(preprocessing.LabelEncoder())

        X\_encoded[:, i] = label\_encoder[-1].fit\_transform(X[:, i])

X = X\_encoded.astype(int)

y = np.array(y)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=5)

# Створення і навчання класифікатора з Гаусовим ядром

classifier = SVC(kernel='rbf')

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

# Прогнозування результату для тестових даних

y\_test\_pred = classifier.predict(X\_test)

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

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

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

f1 = f1\_score(y\_test, y\_test\_pred)

print("\n--- Гауссове ядро SVM ---")

print("Звіт про класифікацію:\n", classification\_report(y\_test, y\_test\_pred))

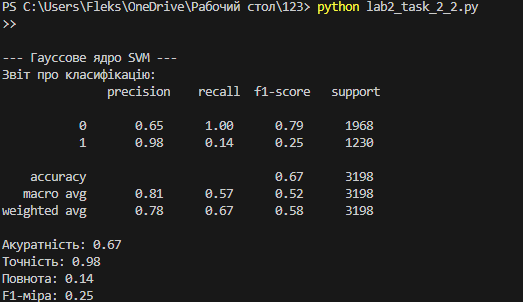
print(f"Акуратність: {accuracy:.2f}")

print(f"Точність: {precision:.2f}")

print(f"Повнота: {recall:.2f}")

print(f"F1-міра: {f1:.2f}")

LR\_2\_task\_2\_2 код для Гаусового ядра



Виконання коду для Гаусового ядра

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 classification\_report, accuracy\_score, precision\_score, recall\_score, f1\_score

input\_file = 'income\_data.txt'

X = []

y = []

count\_class1 = 0

count\_class2 = 0

max\_datapoints = 10000

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(', ')

        label = data[-1]

        if label == '<=50K' and count\_class1 < max\_datapoints:

            X.append(data[:-1])

            y.append(0)

            count\_class1 += 1

        elif label == '>50K' and count\_class2 < max\_datapoints:

            X.append(data[:-1])

            y.append(1)

            count\_class2 += 1

X = np.array(X)

label\_encoder = []

X\_encoded = np.empty(X.shape, dtype=object)

for i in range(X.shape[1]):

    if X[0, i].isdigit():

        X\_encoded[:, i] = X[:, i].astype(int)

    else:

        label\_encoder.append(preprocessing.LabelEncoder())

        X\_encoded[:, i] = label\_encoder[-1].fit\_transform(X[:, i])

X = X\_encoded.astype(int)

y = np.array(y)

# Розподіл даних на тренувальні та тестові

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=5)

# Створення і навчання класифікатора з сигмоїдальним ядром

classifier = SVC(kernel='sigmoid')

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

# Прогнозування результату для тестових даних

y\_test\_pred = classifier.predict(X\_test)

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

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

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

f1 = f1\_score(y\_test, y\_test\_pred)

print("\n--- Сигмоїдальне ядро SVM ---")

print("Звіт про класифікацію:\n", classification\_report(y\_test, y\_test\_pred))

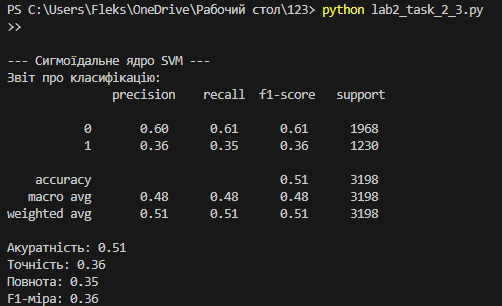
print(f"Акуратність: {accuracy:.2f}")

print(f"Точність: {precision:.2f}")

print(f"Повнота: {recall:.2f}")

print(f"F1-міра: {f1:.2f}")

LR\_2\_task\_2\_3 код для сигмоїдного ядра



Виконання коду для сигмоїдного ядра

Найкраще для завдання класифікації спрацювало гауссове ядро (RBF), оскільки воно досягло найвищої загальної точності (0.67). Це свідчить про те, що гауссове ядро краще обробляє нелінійні особливості у даних. Однак низька повнота для класу "1" вказує на потребу подальшого налаштування гіперпараметрів або використання інших методів, таких як зважування класів або балансування даних.

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.

Крок 1:

from sklearn.datasets import load\_iris

# Завантаження набору даних

iris\_dataset = load\_iris()

# Ключі об'єкта iris\_dataset

print("Ключі iris\_dataset: \n{}".format(iris\_dataset.keys()))

# Опис набору даних

print("Опис набору даних:\n", iris\_dataset['DESCR'][:193] + "\n...")

# Назви відповідей (класів)

print("Назви відповідей: {}".format(iris\_dataset['target\_names']))

# Назва ознак

print("Назва ознак: \n{}".format(iris\_dataset['feature\_names']))

# Тип масиву data

print("Тип масиву data: {}".format(type(iris\_dataset['data'])))

# Форма масиву data

print("Форма масиву data: {}".format(iris\_dataset['data'].shape))

# Виведення значень ознак для перших п'яти прикладів

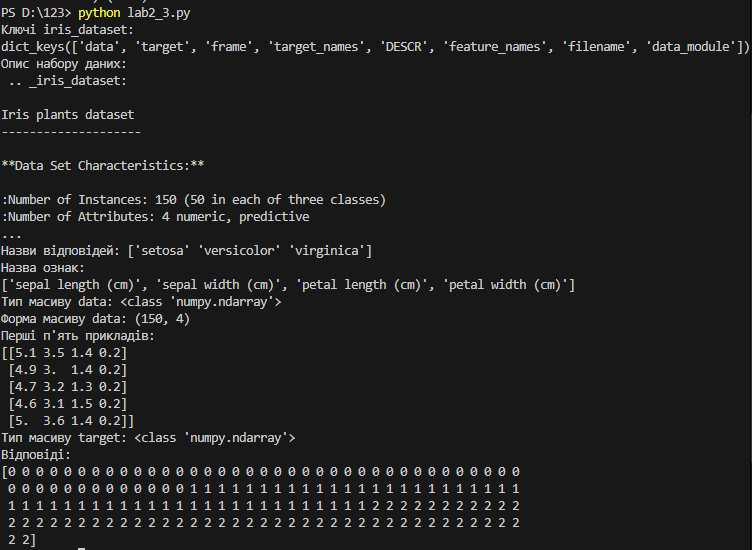
print("Перші п'ять прикладів:\n{}".format(iris\_dataset['data'][:5]))

# Тип масиву target

print("Тип масиву target: {}".format(type(iris\_dataset['target'])))

# Виведення цільових значень

print("Відповіді:\n{}".format(iris\_dataset['target']))



Завантаження та вивчення даних

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("Розмір датасету:", dataset.shape)

print("\nПерші 20 рядків:")

print(dataset.head(20))

print("\nСтатистичне зведення:")

print(dataset.describe())

print("\nРозподіл за класами:")

print(dataset.groupby('class').size())

# Діаграма розмаху для кожного атрибута

dataset.plot(kind='box', subplots=True, layout=(2, 2), sharex=False, sharey=False, figsize=(10, 6))

plt.suptitle("Діаграми розмаху для атрибутів")

plt.show()

# Гістограма для кожного атрибута

dataset.hist(figsize=(10, 6))

plt.suptitle("Гістограми розподілу атрибутів")

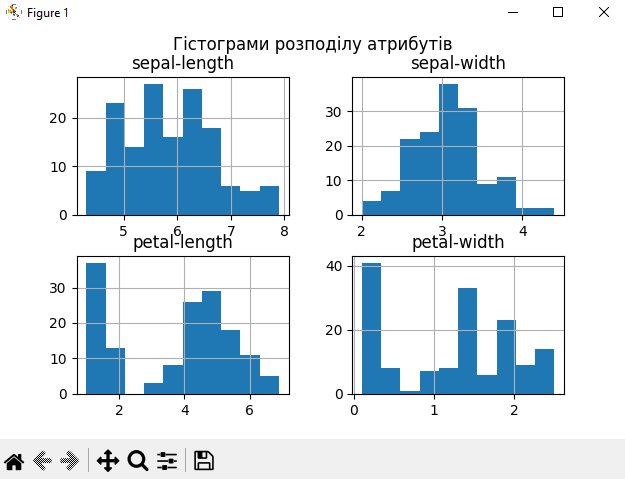
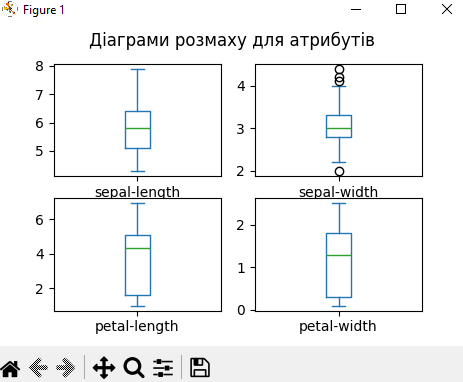
plt.show()

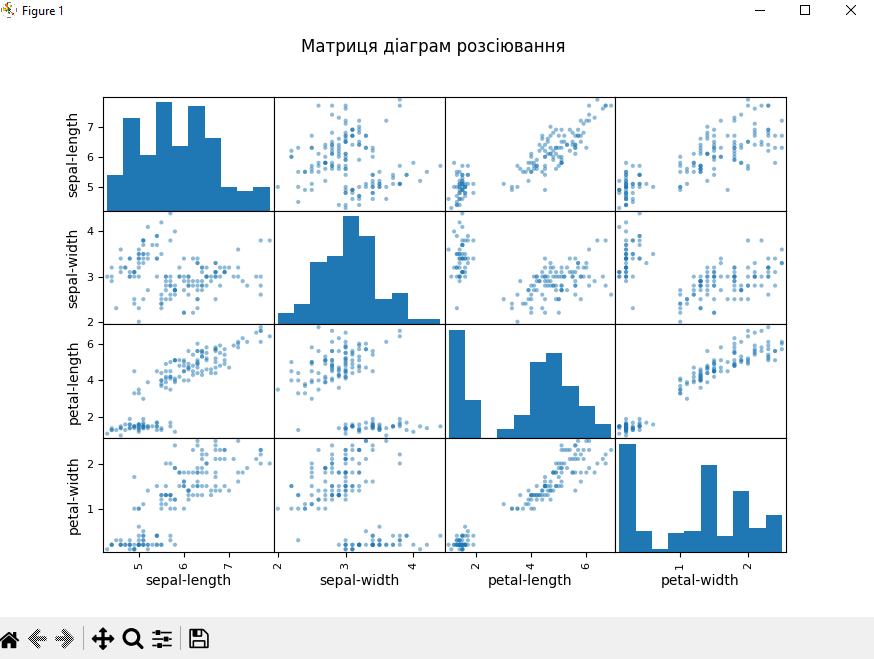
# Матриця діаграм розсіювання

scatter\_matrix(dataset, figsize=(12, 8), diagonal='hist')

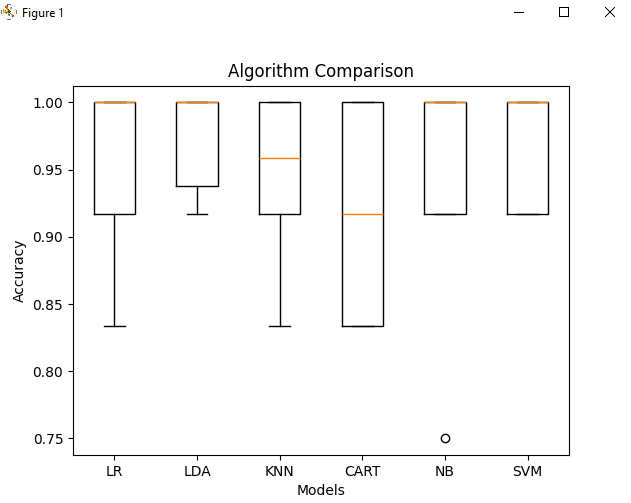
plt.suptitle("Матриця діаграм розсіювання")

plt.show()

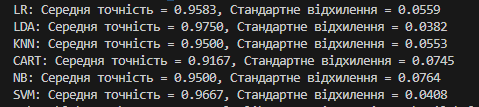




Крок 3-4:



Найкращим виявився “Наївний баєсовський класифікатор” (NB) з найбішою точністю та мінімальним відхиленням



Крок 5-8

import numpy as np

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split, StratifiedKFold, cross\_val\_score

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 matplotlib.pyplot as plt

iris = load\_iris()

X = iris.data

y = iris.target

X\_train, X\_validation, y\_train, y\_validation = train\_test\_split(X, y, test\_size=0.2, random\_state=1, stratify=y)

# КРОК 2: Побудова та оцінка 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')))

# Оцінка моделей

results = []

names = []

print("Результати оцінки моделей (точність):")

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})")

# Порівняння алгоритмів на графіку

plt.boxplot(results, labels=names)

plt.title('Порівняння алгоритмів')

plt.xlabel('Модель')

plt.ylabel('Точність')

plt.show()

# КРОК 6: Створення прогнозу на тестовій вибірці

model = SVC(gamma='auto')

model.fit(X\_train, y\_train)  # Навчаємо модель

predictions = model.predict(X\_validation)  # Прогноз на тестовій вибірці

# КРОК 7: Оцінка якості моделі

print("\nТочність моделі на тестовій вибірці:", accuracy\_score(y\_validation, predictions))

print("\nМатриця помилок:")

print(confusion\_matrix(y\_validation, predictions))

print("\nЗвіт про класифікацію:")

print(classification\_report(y\_validation, predictions))

# КРОК 8: Передбачення для нових даних

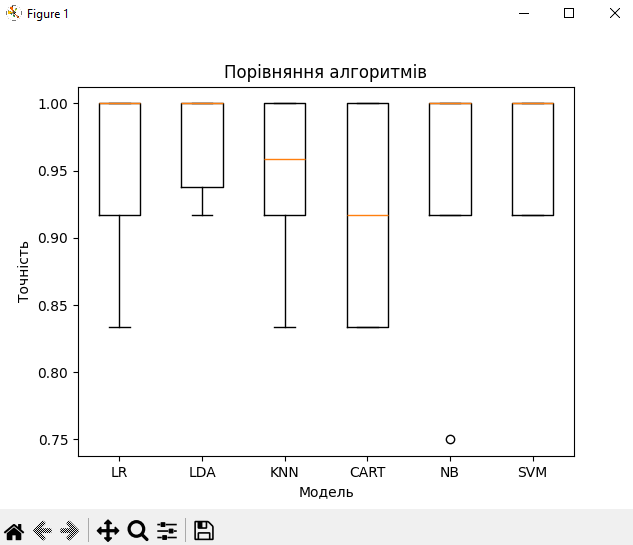
X\_new = np.array([[5, 2.9, 1, 0.2]])

print("\nФорма масиву X\_new:", X\_new.shape)

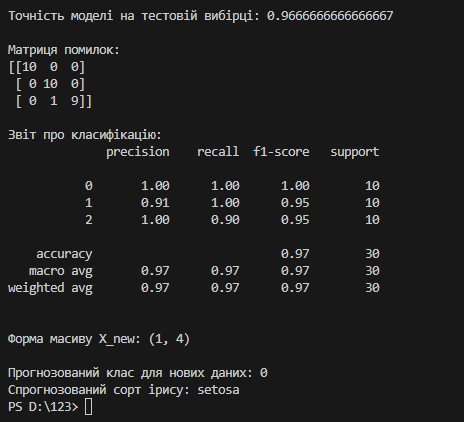
new\_prediction = model.predict(X\_new)

print("\nПрогнозований клас для нових даних:", new\_prediction[0])

print("Спрогнозований сорт ірису:", iris.target\_names[new\_prediction[0]])



Вивид графіків



Вивід даних в консоль

**Висновок:**

**Якість класифікації:**

Точність моделі: 96.67%, що свідчить про високу ефективність класифікації.

Матриця помилок: більшість класів класифікуються правильно, з деякими помилками між класами `1` (versicolor) та `2` (virginica).

Звіт про класифікацію: висока точність і повний збіг для класу `0`, хороші результати для інших класів.

**Прогноз для нових даних:**

Квітка з новими характеристиками була класифікована як setosa.

**Завдання 2.4** Порівняння якості класифікаторів для набору даних завдання 2.1

import pandas as pd

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

from matplotlib import pyplot as plt

data\_path = "income\_data.txt"

columns = ['age', 'workclass', 'education', 'education-num', 'marital-status',

           'occupation', 'relationship', 'race', 'sex', 'hours-per-week',

           'native-country', 'income']

dataset = pd.read\_csv(data\_path, header=None, names=columns)

dataset = pd.get\_dummies(dataset, drop\_first=True)

X = dataset.drop('income\_ >50K', axis=1)  # Вхідні ознаки

y = dataset['income\_ >50K']  # Мітки класів

X\_train, X\_validation, Y\_train, Y\_validation = train\_test\_split(X, y, test\_size=0.2, random\_state=1, stratify=y)

models = []

models.append(('LR', LogisticRegression(solver='liblinear')))

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("\nРезультати моделей:")

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()))

plt.boxplot(results, labels=names)

plt.title('Порівняння алгоритмів класифікації')

plt.show()

model = SVC(gamma='auto')

model.fit(X\_train, Y\_train)

predictions = model.predict(X\_validation)

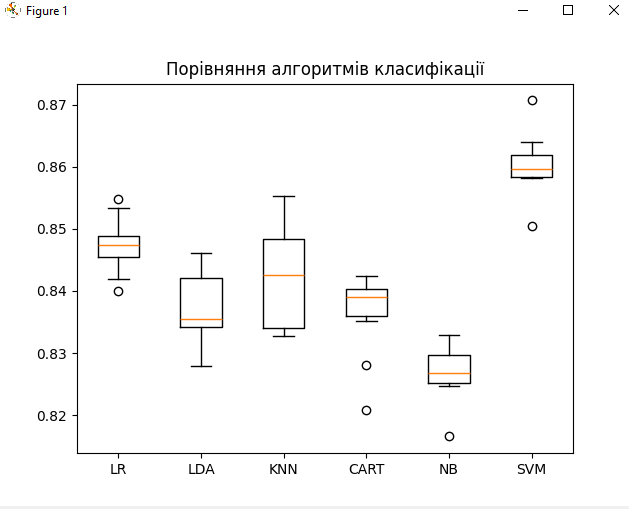
print("\nТочність:", accuracy\_score(Y\_validation, predictions))

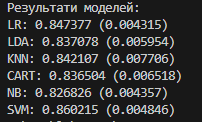
print("Матриця помилок:")

print(confusion\_matrix(Y\_validation, predictions))

print("\nЗвіт про класифікацію:")

print(classification\_report(Y\_validation, predictions))





З результатів видно, що SVM має найвищу точність (0.8602), що на 1.5% вищий за точність логістичної регресії (0.8474) та метод k-найближчих сусідів (0.8421). Всі моделі мають подібну варіативність (стандартне відхилення не перевищує 0.01), що свідчить про стабільність цих моделей на тестових даних.Проте , дана модель найповільніша.

Пояснення вибору найкращої моделі:

Вибір SVM як найкращої моделі для цієї задачі обґрунтовано кількома факторами: Найвища точність: SVM продемонстрував найвищу точність, що є головним критерієм для порівняння класифікаторів. Стабільність: Стандартне відхилення для SVM (0.0048) є одним з найменших, що свідчить про стабільність моделі.

Підходить для високовимірних даних: SVM добре працює з високовимірними даними, що може бути корисним, якщо в майбутньому з’являться нові змінні.

Висновок :

Найкраща модель: SVM є найкращим вибором для цієї задачі на основі точності та стабільності результатів. Рекомендація: Хоча інші моделі також можуть бути корисними, SVM дає найкращі результати для поточного набору даних.

**Завдання 2.5.** Класифікація даних лінійним класифікатором Ridge

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

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.metrics import confusion\_matrix

from io import BytesIO

# Завантаження даних Iris

iris = load\_iris()

X, y = iris.data, iris.target

# Розподіл на тренувальні та тестові дані

Xtrain, Xtest, ytrain, ytest = train\_test\_split(X, y, test\_size=0.3, random\_state=0)

# Ініціалізація і тренування класифікатора Ridge

clf = RidgeClassifier(tol=1e-2, solver="sag")

clf.fit(Xtrain, ytrain)

# Прогнозування на тестових даних

ypred = clf.predict(Xtest)

# Виведення показників якості класифікації

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('Cohen Kappa Score:', np.round(metrics.cohen\_kappa\_score(ytest, ypred), 4))

print('Matthews Corrcoef:', np.round(metrics.matthews\_corrcoef(ytest, ypred), 4))

# Звіт про класифікацію

print('\t\tClassification Report:\n', metrics.classification\_report(ytest, ypred))

# Матриця плутанини

mat = confusion\_matrix(ytest, ypred)

sns.set()

sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False)

plt.xlabel('True label')

plt.ylabel('Predicted label')

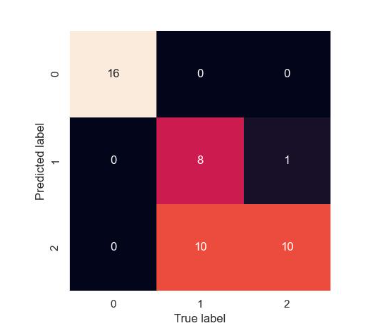
# Збереження зображення у файл

plt.savefig("Confusion.jpg")

# Збереження зображення у форматі SVG

f = BytesIO()

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



Пояснення:

**Завантаження та розподіл даних:**

Данні Iris завантажуються за допомогою load\_iris().

Використовуємо train\_test\_split() для розподілу даних на тренувальні та тестові набори. 70% даних йде на тренування, а 30% — на тестування.

Ініціалізація та тренування класифікатора:

Класифікатор Ridge ініціалізується з параметрами:

tol=1e-2: поріг для завершення ітерацій.

solver="sag": використовується метод Stochastic Average Gradient.

Модель тренується за допомогою clf.fit(Xtrain, ytrain).

Прогнозування та оцінка моделі:

Прогнозування здійснюється за допомогою clf.predict(Xtest).

Для оцінки точності моделі використовуються різні метрики:

Accuracy: частка правильних прогнозів.

Precision: точність класифікації.

Recall: повнота класифікації.

F1 Score: середнє гармонійне точності та повноти.

Cohen Kappa: вимірює узгодженість між класифікатором та реальними мітками.

Matthews Correlation Coefficient (MCC): вимірює кореляцію між реальними та прогнозованими мітками.

Матриця плутанини:

Використовуємо confusion\_matrix() для створення матриці плутанини.

За допомогою бібліотеки Seaborn sns.heatmap() будується теплове зображення цієї матриці для візуалізації.

Збереження результатів:

Зображення матриці плутанини зберігається у файл Confusion.jpg.

Також зберігається векторне зображення у форматі SVG.

Пояснення використаних показників:

Cohen Kappa вимірює узгодженість класифікаційних результатів, зважаючи на випадкові ймовірності для кожного класу.

Matthews Correlation Coefficient (MCC) забезпечує більш стабільну оцінку ефективності класифікатора, особливо у разі незбалансованих класів, враховуючи всі чотири категорії: True Positive (TP), True Negative (TN), False Positive (FP) і False Negative (FN).

Завдяки цим метрикам можна отримати комплексну оцінку якості моделі, яка дозволяє краще зрозуміти її ефективність для класифікації даних Iris.

**Висновок:**

У роботі були порівняні різні методи класифікації, зокрема SVM, LDA, KNN, LR, CART, NB, та Ridge. Найвищу точність показав метод опорних векторів (SVM). Вибір найкращого методу залежить від конкретної задачі, проте для даних Iris SVM виявився найбільш ефективним.