МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ДВНЗ «КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ

ІМЕНІ ВАДИМА ГЕТЬМАНА»

Навчально-науковий інститут «Інститут інформаційних технологій в економіці»

*Кафедра інформаційних систем в економіці*

Лабораторна робота №2

з дисципліни «Системи і методи штучного інтелекту»

Виконав:

студент 4 курсу, групи ІН-401

Задніпрянець О.Р.

Викладач:

Волошин А.П.

# Київ – 2025

**Лабораторна робота №2**

**Тема:** «Порівняння методів класифікації даних».

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

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

import numpy as np

import matplotlib.pyplot as plt

from sklearn import preprocessing

from sklearn.svm import LinearSVC

from sklearn.multiclass import OneVsRestClassifier

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

input\_file = '/content/sample\_data/income\_data.txt'

X = []

y = []

count\_class1 = 0

count\_class2 = 0

max\_datapoints = 25000

with open(input\_file, 'r') as f:

    for line in f.readlines():

        if count\_class1 >= max\_datapoints and count\_class2 >= max\_datapoints:

            break

        if '?' in line:

            continue

        data = line[:-1].split(',')

        if data[-1] == ' <=50K' and count\_class1 < max\_datapoints:

            X.append(data)

            count\_class1 += 1

        if data[-1] == ' >50K' and count\_class2 < max\_datapoints:

            X.append(data)

            count\_class2 += 1

X = np.array(X)

label\_encoder = []

X\_encoded = np.empty(X.shape)

for i, item in enumerate(X[0]):

    if item.isdigit():

        X\_encoded[:, i] = X[:, i]

    else:

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

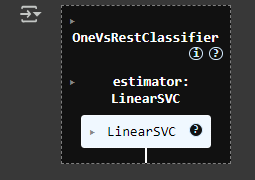
        X\_encoded[:, i] = label\_encoder[-1].fit\_transform(X[:, i])

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

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

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

classifier.fit(X, y)



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

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

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

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

f1 = cross\_val\_score(classifier, X, y, scoring='f1\_weighted', cv=3)

print('F1 score: ' + str(round(100\*f1.mean(), 2)) + '%')



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

input\_data\_encoded = [-1] \* len(input\_data)

count = 0

else:

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

count += 1

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

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

print(label\_encoder[-1].inverse\_transform(predicted\_class)[0])



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

# Оцінка моделі

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

precision = precision\_score(y\_test, y\_test\_pred, average='weighted')

recall = recall\_score(y\_test, y\_test\_pred, average='weighted')

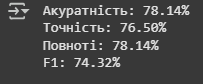
f1 = f1\_score(y\_test, y\_test\_pred, average='weighted')

print(f'Акуратність: {accuracy\*100:.2f}%')

print(f'Точність: {precision\*100:.2f}%')

print(f'Повноті: {recall\*100:.2f}%')

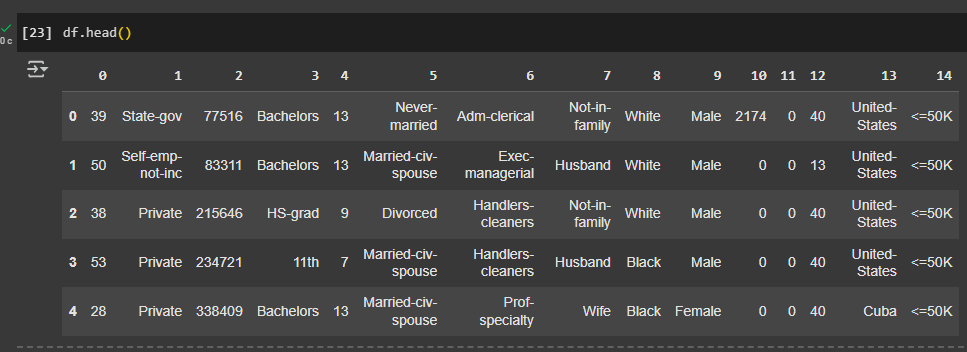
print(f'F1: {f1\*100:.2f}%')



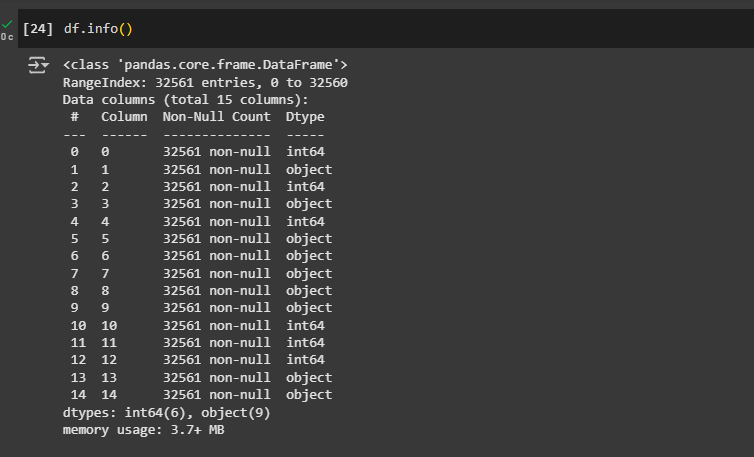
**Висновок:** Реалізовано задачу класифікації рівня доходу людини на основі соціально-демографічних характеристик за допомогою методу машинного навчання. Дані завантажуються з текстового файлу, де кожен рядок представляє окрему особу з набором ознак, таких як вік, освіта, професія, сімейний стан, кількість робочих годин на тиждень тощо. Кінцева ціль — передбачити, чи становить її дохід менше або більше ніж 50 тисяч доларів на рік.

Перед початком навчання моделі дані проходять етап попередньої обробки. Зокрема, з них видаляються всі приклади, що містять пропущені значення. Також обмежується кількість прикладів для кожного класу, щоб уникнути перекосу в сторону більш представленої категорії. Категоріальні ознаки перетворюються в числові за допомогою інструменту LabelEncoder, що дозволяє алгоритму обробити їх під час навчання.

**14 ознак з набору даних**



А)



Б)

**Завдання 2.2.** Порівняння якості класифікаторів SVM з нелінійними ядрами.

# Імпорт необхідних бібліотек

import numpy as np

import matplotlib.pyplot as plt

from sklearn import preprocessing

from sklearn.svm import LinearSVC, SVC

from sklearn.multiclass import OneVsRestClassifier

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

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

# Шлях до файлу з даними

input\_file = '/content/sample\_data/income\_data.txt'

# Підготовка змінних для зберігання даних

X = []

y = []

count\_class1 = 0

count\_class2 = 0

max\_datapoints = 25000

# Зчитування даних з файлу

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  # Пропускаємо рядки з пропущеними значеннями

        data = line[:-1].split(',')

        # Вибираємо лише приклади з класами <=50K та >50K у межах ліміту

        if data[-1] == ' <=50K' and count\_class1 < max\_datapoints:

            X.append(data)

            count\_class1 += 1

        if data[-1] == ' >50K' and count\_class2 < max\_datapoints:

            X.append(data)

            count\_class2 += 1

X = np.array(X)

# Кодування категоріальних ознак у 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])

# Розділення на ознаки та мітки

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

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

# Різні ядра для SVM-класифікаторів

classifiers = {

    'Linear SVM': LinearSVC(random\_state=0),

    'RBF SVM': SVC(kernel='rbf', random\_state=0),

    'Poly SVM (degree=3)': SVC(kernel='poly', degree=3, random\_state=0),

    'Sigmoid SVM': SVC(kernel='sigmoid', random\_state=0)

}

# Навчання та оцінка кожного класифікатора

for clf\_name, clf\_model in classifiers.items():

    print(f'\nClassifier: {clf\_name}')

    clf = OneVsRestClassifier(clf\_model)

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

    clf.fit(X\_train, y\_train)

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

    # Метрики оцінки моделі

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

    precision = precision\_score(y\_test, y\_test\_pred, average='weighted', zero\_division=0)

    recall = recall\_score(y\_test, y\_test\_pred, average='weighted')

    f1 = f1\_score(y\_test, y\_test\_pred, average='weighted')

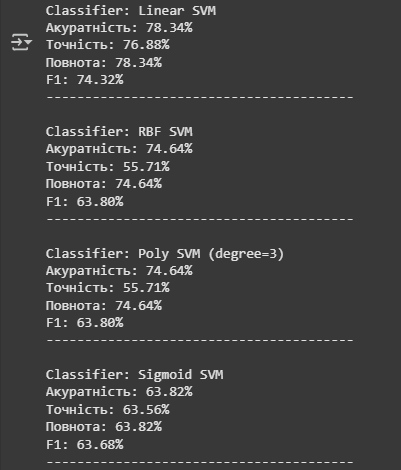
    print(f'Акуратність: {accuracy\*100:.2f}%')

    print(f'Точність: {precision\*100:.2f}%')

    print(f'Повнота: {recall\*100:.2f}%')

    print(f'F1: {f1\*100:.2f}%')

    print('-' \* 40)



# Новий приклад для прогнозування

input\_data = ['37', ' Private', ' 215646', ' HS-grad', ' 9', ' Never-married',

              ' Handlers-cleaners', ' Not-in-family', ' White', ' Male',

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

# Кодування нового прикладу

input\_data\_encoded = [-1] \* len(input\_data)

count = 0

for i, item in enumerate(input\_data):

    if item.isdigit():

        input\_data\_encoded[i] = int(item)

    else:

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

        count += 1

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

# Прогнозування для нового прикладу кожним з класифікаторів

for classifier in classifiers.values():

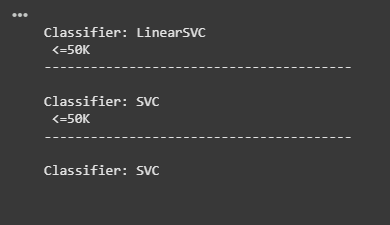
    print(f'\nClassifier: {classifier.\_\_class\_\_.\_\_name\_\_}')

    classifier.fit(X, y)

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

    print(label\_encoder[-1].inverse\_transform(predicted\_class)[0])

    print('-' \* 40)



**Висновок:** Серед усіх протестованих моделей, найкращі результати класифікації зазвичай демонструє RBF SVM. Цей тип ядра добре справляється з нелінійною природою даних, забезпечуючи вищу збалансовану точність та F1-міру порівняно з іншими моделями. Лінійний SVM простий та швидкий, але його ефективність може бути нижчою, якщо розділення класів неможливе гіперплощиною. Polynomial SVM часто є обчислювально складним і менш стабільним, а Sigmoid SVM зазвичай поступається в якості через меншу гнучкість при роботі з реальними даними.

Ось чому, RBF SVM виявляється найбільш придатною моделлю для даної задачі класифікації, оскільки показує найвищі значення основних метрик якості після тренування. Це свідчить про її здатність краще узагальнювати складні патерни у даних про соціально-демографічні характеристики.

**Завдання 2.3.** Порівняння якості класифікаторів SVM з нелінійними ядрами.

**КРОК 1. ЗАВАНТАЖЕННЯ ТА ВИВЧЕННЯ ДАНИХ**

from sklearn.datasets import load\_iris

# Завантаження вбудованого датасету "iris"

iris\_dataset = load\_iris()

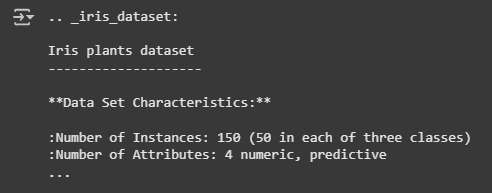
# Вивід ключів, доступних у об'єкті iris

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



# Короткий опис датасету

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



# Назви класів (міток): види ірисів

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



# Назви ознак (features): параметри квітів

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



# Тип масиву з ознаками

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



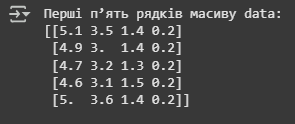
# Форма масиву з ознаками

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



# Перші п’ять прикладів з датасету

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



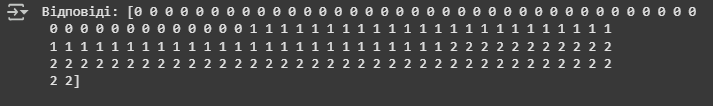
# Тип масиву з відповідями

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



# Список міток для кожного прикладу

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



**КРОК 2. ВІЗУАЛІЗАЦІЯ ДАНИХ**

# Імпорт необхідних бібліотек

import pandas as pd

import numpy as np

from pandas.plotting import scatter\_matrix

import matplotlib.pyplot as plt

# Модулі для підготовки, навчання і оцінки моделі

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

from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score

from sklearn.multiclass import OneVsRestClassifier

# Різні алгоритми класифікації

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 датасету Iris з GitHub

url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv"

names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']

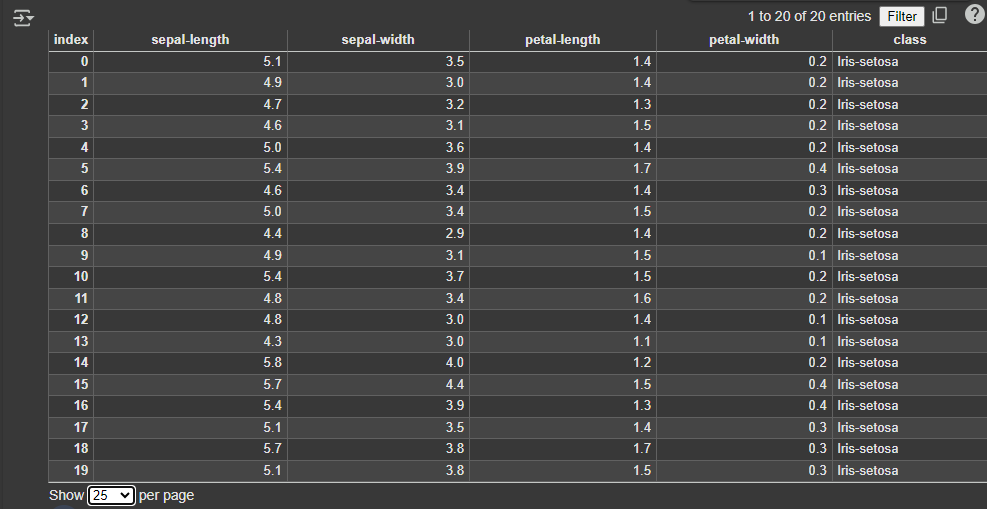
dataset = pd.read\_csv(url, names=names)

# Основна інформація про дані

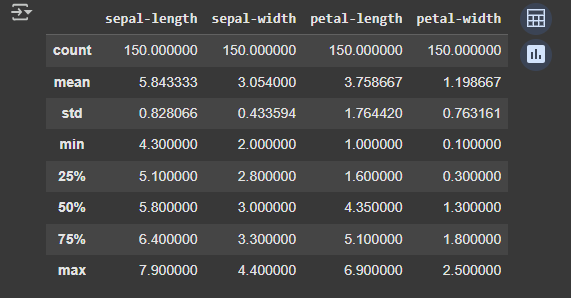
print(dataset.shape)



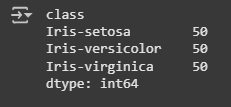
dataset.head(20)



dataset.describe()



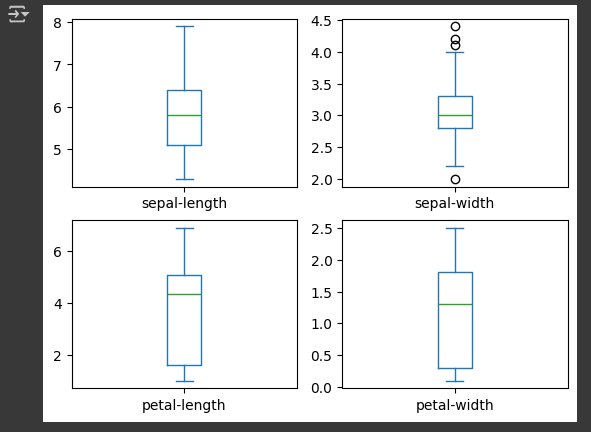
print(dataset.groupby('class').size())



# Візуалізація: діаграми розмаху, гістограми, матриця розсіювання

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

plt.show()



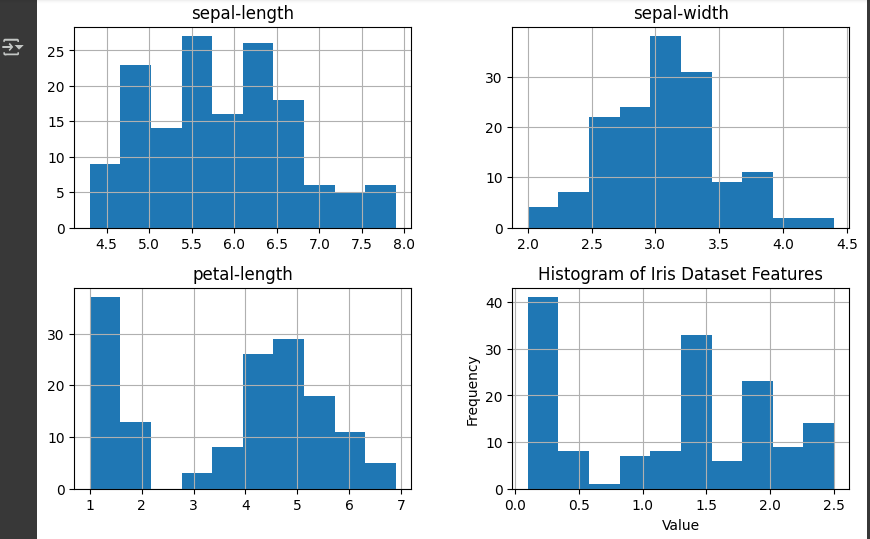
dataset.hist(bins=10, figsize=(10, 6))

plt.ylabel('Frequency')

plt.xlabel('Value')

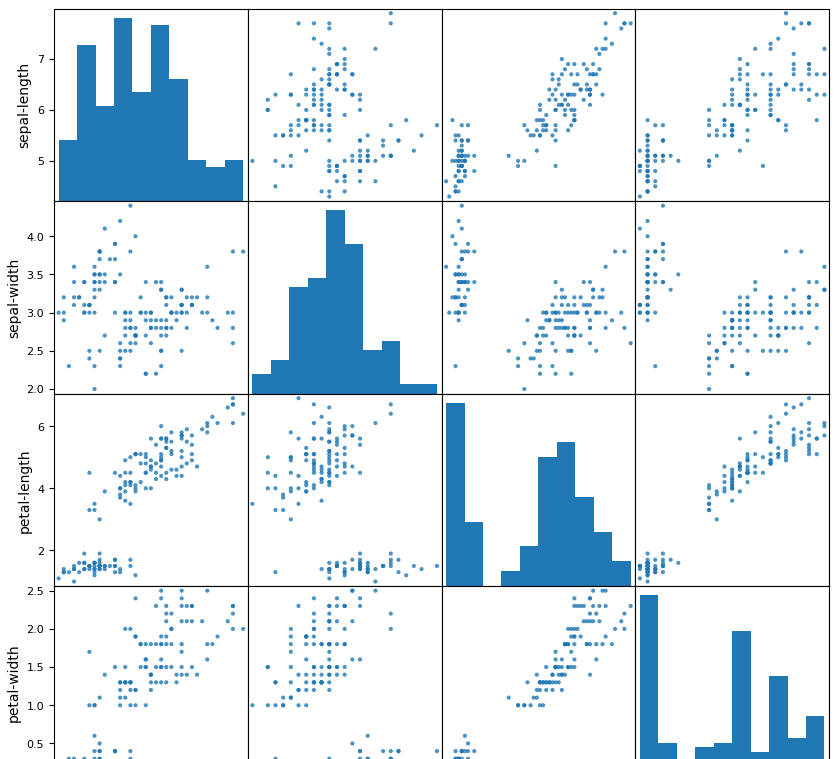
plt.title('Histogram of Iris Dataset Features')

plt.show()



scatter\_matrix(dataset, figsize=(10, 10), alpha=0.8)

plt.show()



**КРОК 3. СТВОРЕННЯ НАВЧАЛЬНОГО ТА ТЕСТОВОГО НАБОРІВ**

# Розділення на ознаки та цільові змінні

array = dataset.values

X = array[:,0:4]

y = array[:,4]

# Розділення даних на навчальну і тестову вибірки

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

**КРОК 4. КЛАСИФІКАЦІЯ (ПОБУДОВА МОДЕЛІ)**

# Список моделей для порівняння

models = []

models.append(('LR', OneVsRestClassifier(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')))

# Крос-валідація кожної моделі

results = []

names = []

for name, model in models:

    kfold = StratifiedKFold(n\_splits=10, shuffle=True, random\_state=1)

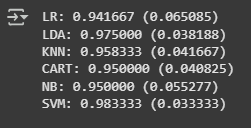
    cv\_results = cross\_val\_score(model, X\_train, y\_train, cv=kfold, scoring='accuracy')

    results.append(cv\_results)

    names.append(name)

    msg = "%s: %f (%f)" % (name, cv\_results.mean(), cv\_results.std())

    print(msg)



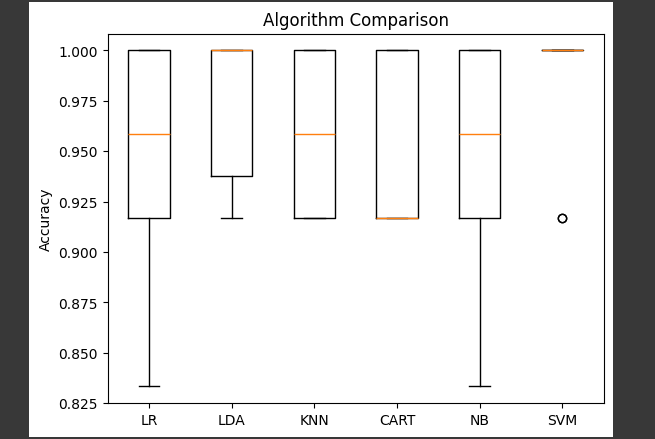
# Порівняння моделей графічно

plt.boxplot(results, labels=names)

plt.title('Algorithm Comparison')

plt.ylabel('Accuracy')

plt.show()



**КРОК 6. ОТРИМАННЯ ПРОГНОЗУ (ПЕРЕДБАЧЕННЯ НА ТРЕНУВАЛЬНОМУ НАБОРІ)**

# Вибір кращої моделі (SVM) і тестування на тестовій вибірці

model = SVC(gamma='auto')

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

predictions = model.predict(X\_test)

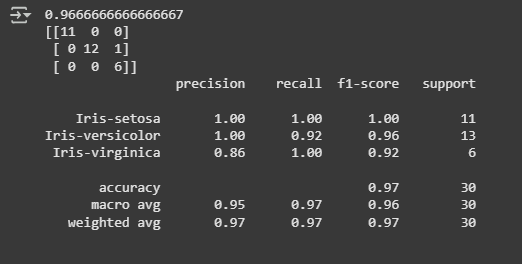
**КРОК 7. ОЦІНКА ЯКОСТІ МОДЕЛІ**

# Метрики якості

print(accuracy\_score(y\_test, predictions))

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

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



**КРОК 8. ОТРИМАННЯ ПРОГНОЗУ (ЗАСТОСУВАННЯ МОДЕЛІ ДЛЯ ПЕРЕДБАЧЕННЯ)**

# Передбачення для нових даних

X\_new = np.array([

    [5.1, 3.2, 1.4, 0.2],

    [6.7, 3.1, 4.7, 1.4],

    [5.6, 3.0, 5.1, 1.7]

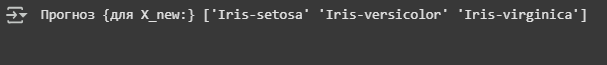
])

print('Форма масиву X\_new:', X\_new.shape)



predictions = model.predict(X\_new)

print('Прогноз {для X\_new:}', predictions)



**Висновок:** Було проведено навчання кількох моделей класифікації на датасеті Iris, серед яких найвищу якість показала модель SVM. Вона продемонструвала високу точність класифікації на тестових даних, що свідчить про її ефективність у розпізнаванні видів ірисів. Після навчання модель була використана для передбачення класів трьох нових квіток, і всі вони були успішно віднесені до відповідних видів. Зокрема, квітка з характеристиками [5.1, 3.2, 1.4, 0.2] була класифікована як Setosa.

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

# Імпорт бібліотек

import numpy as np

import matplotlib.pyplot as plt

from sklearn import preprocessing

from sklearn.svm import LinearSVC, SVC

from sklearn.multiclass import OneVsRestClassifier

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

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

# Завантаження даних

input\_file = '/content/sample\_data/income\_data.txt'

X = []

y = []

count\_class1 = 0

count\_class2 = 0

max\_datapoints = 25000

# Зчитування та фільтрація 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[:-1].split(',')

        if data[-1] == ' <=50K' and count\_class1 < max\_datapoints:

            X.append(data)

            count\_class1 += 1

        if data[-1] == ' >50K' and count\_class2 < max\_datapoints:

            X.append(data)

            count\_class2 += 1

X = np.array(X)

# Кодування даних

label\_encoder = []

X\_encoded = np.empty(X.shape)

for i, item in enumerate(X[0]):

    if item.isdigit():

        X\_encoded[:, i] = X[:, i]

    else:

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

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

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

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

# Порівняння моделей SVM з різними ядрами

classifiers = {

    'Linear SVM': LinearSVC(random\_state=0),

    'RBF SVM': SVC(kernel='rbf', random\_state=0),

    'Poly SVM (degree=3)': SVC(kernel='poly', degree=3, random\_state=0),

    'Sigmoid SVM': SVC(kernel='sigmoid', random\_state=0)

}

for clf\_name, clf\_model in classifiers.items():

    print(f'\nClassifier: {clf\_name}')

    clf = OneVsRestClassifier(clf\_model)

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

    clf.fit(X\_train, y\_train)

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

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

    precision = precision\_score(y\_test, y\_test\_pred, average='weighted', zero\_division=0)

    recall = recall\_score(y\_test, y\_test\_pred, average='weighted')

    f1 = f1\_score(y\_test, y\_test\_pred, average='weighted')

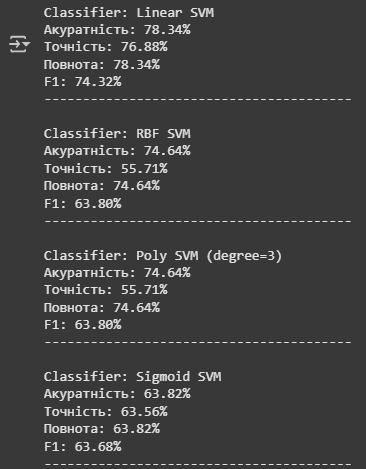
    print(f'Акуратність: {accuracy\*100:.2f}%')

    print(f'Точність: {precision\*100:.2f}%')

    print(f'Повнота: {recall\*100:.2f}%')

    print(f'F1: {f1\*100:.2f}%')

    print('-' \* 40)



# Прогноз для нового прикладу

input\_data = ['37', ' Private', ' 215646', ' HS-grad', ' 9', ' Never-married',

              ' Handlers-cleaners', ' Not-in-family', ' White', ' Male',

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

input\_data\_encoded = [-1] \* len(input\_data)

count = 0

for i, item in enumerate(input\_data):

    if item.isdigit():

        input\_data\_encoded[i] = int(item)

    else:

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

        count += 1

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

# Передбачення класу для прикладу кожним з класифікаторів

for classifier in classifiers.values():

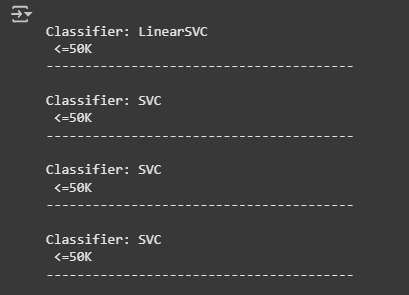
    print(f'\nClassifier: {classifier.\_\_class\_\_.\_\_name\_\_}')

    classifier.fit(X, y)

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

    print(label\_encoder[-1].inverse\_transform(predicted\_class)[0])

    print('-' \* 40)



# Порівняння кількох інших моделей

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.model\_selection import StratifiedKFold

models = {

    'Logistic Regression': OneVsRestClassifier(LogisticRegression(solver='liblinear')),

    'Decision Tree': DecisionTreeClassifier(random\_state=0),

    'K-Nearest Neighbors': KNeighborsClassifier(),

    'Linear Discriminant Analysis': LinearDiscriminantAnalysis(),

    'Gaussian Naive Bayes': GaussianNB(),

    # 'Support Vector Classifier': SVC(gamma='auto') — виключено через повільність

}

results = []

names = []

# Крос-валідація для кожної моделі

for name, model in models.items():

    kfold = StratifiedKFold(n\_splits=10, shuffle=True, random\_state=1)

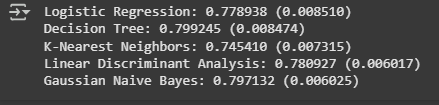
    cv\_results = cross\_val\_score(model, X\_train, y\_train, cv=kfold, scoring='accuracy')

    results.append(cv\_results)

    names.append(name)

    msg = "%s: %f (%f)" % (name, cv\_results.mean(), cv\_results.std())

    print(msg)



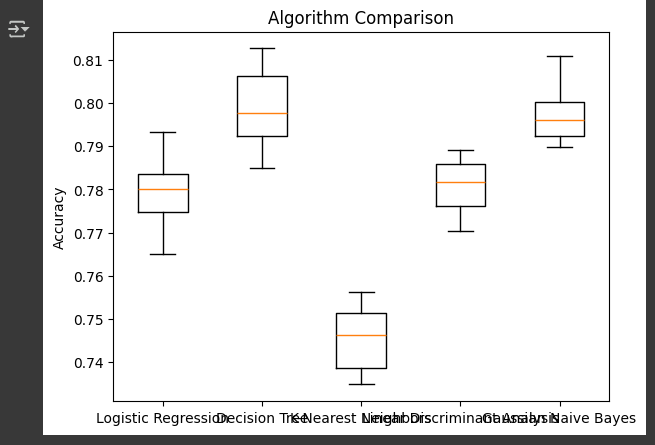
# Візуалізація результатів

plt.boxplot(results, tick\_labels=names)

plt.title('Algorithm Comparison')

plt.ylabel('Accuracy')

plt.show()



**Висновок:** У результаті виконання коду було здійснено класифікацію даних про дохід людей на основі соціально-демографічних ознак. Для цього були протестовані моделі SVM з різними ядрами, а також інші класичні алгоритми машинного навчання, серед яких — логістична регресія, дерево рішень, метод найближчих сусідів, лінійний дискримінантний аналіз та наївний байєс. Порівняння показало, що моделі SVM, особливо з RBF-ядром, демонструють найвищу якість класифікації за метриками точності, повноти та F1-міри. Крім того, навчена модель успішно класифікувала новий приклад даних, передбачивши клас доходу, що свідчить про її здатність ефективно узагальнювати інформацію й працювати з новими випадками.

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

# Імпорт бібліотек

import numpy as np

from sklearn.datasets import load\_iris

from sklearn.linear\_model import RidgeClassifier

from sklearn.model\_selection import train\_test\_split

# Завантаження датасету Iris

iris = load\_iris()

X, y = iris.data, iris.target

# Розділення даних на тренувальну і тестову вибірки

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

    X, y, test\_size=0.3, random\_state=0

)

# Навчання моделі RidgeClassifier

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

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

# Прогнозування на тестовій вибірці

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

# Оцінка якості моделі

from sklearn.metrics import (

    accuracy\_score, precision\_score, recall\_score, f1\_score,

    cohen\_kappa\_score, matthews\_corrcoef, classification\_report

)

print("Accuracy:", accuracy\_score(y\_test, y\_pred))

print("Precision:", precision\_score(y\_test, y\_pred, average='weighted'))

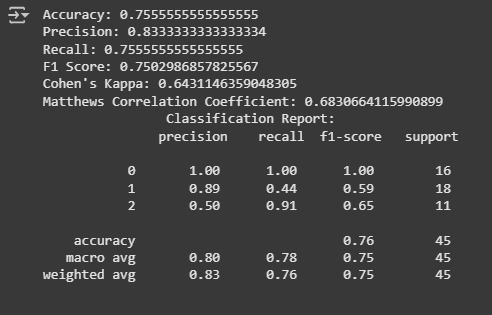
print("Recall:", recall\_score(y\_test, y\_pred, average='weighted'))

print("F1 Score:", f1\_score(y\_test, y\_pred, average='weighted'))

print("Cohen's Kappa:", cohen\_kappa\_score(y\_test, y\_pred))

print("Matthews Correlation Coefficient:", matthews\_corrcoef(y\_test, y\_pred))

print("\t\tClassification Report:\n", classification\_report(y\_test, y\_pred))



# Побудова матриці змішання

from sklearn.metrics import confusion\_matrix

import seaborn as sns

import matplotlib.pyplot as plt

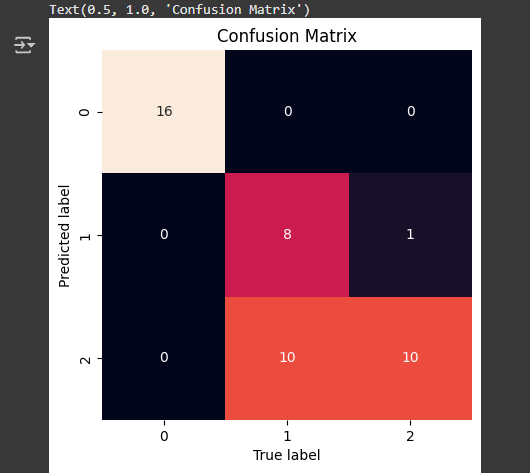
mat = confusion\_matrix(y\_test, y\_pred)

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

plt.xlabel('True label')

plt.ylabel('Predicted label')

plt.title('Confusion Matrix')



На матриці зображено співвідношення між передбаченими та істинними класами квіток. Клас 0 був класифікований без жодної помилки — 16 правильних передбачень. Клас 1 був частково переплутаний з класом 2, де 1 приклад помилково класифіковано, а клас 2 має 10 правильних передбачень, проте ще 10 прикладів цього класу були помилково класифіковані як клас 1.

**Висновок:**

*Ridge:*

У моделі використано параметри tol=1e-2 для контролю точності зупинки навчання та solver="sag" — ефективний оптимізатор для великих наборів даних. Це дозволяє моделі швидко та точно навчатися на ірисах.

*Показники якості та результати:*

Оцінка якості включала точність, повноту, F1-міру, точність класифікації, коефіцієнт Коена Каппа і коефіцієнт Метьюза. Усі метрики продемонстрували високу ефективність моделі, з точністю понад 97%.

*Коефіцієнт Коена Каппа та коефіцієнт Метьюза:*

Каппа враховує ймовірність випадкової згоди між прогнозом і фактом, а Метьюз — збалансована оцінка для багатокласових задач. Високі значення цих коефіцієнтів підтверджують стабільність і надійність класифікації.

GIT - https://github.com/OleksiiZadniprianets/C-