**Лабораторна робота №2**

Системи штучного інтелекту

**Виконав:** ІПЗ-21-3 Осипчук Антон Олексійович

Github:

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

import numpy as np

import matplotlib.pyplot as plt

from sklearn import preprocessing

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

from sklearn.svm import LinearSVC

from sklearn.multiclass import OneVsOneClassifier

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

# Load data

input\_file = './data/income\_data.txt'

# Read the data

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

# Convert to numpy array

X = np.array(X)

# Convert string data to numerical data

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)

# Create SVM classifier

classifier = OneVsOneClassifier(LinearSVC(random\_state=0, dual=False))

# Cross validation

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

# Compute F1 score of the SVM classifier

f1 = cross\_val\_score(classifier, X, y, scoring="f1\_weighted")

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

# Predict output for a test datapoint

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

# Encode test datapoint

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

count = 0

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

if item.isdigit():

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

else:

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

count += 1

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

# Run classifier on encoded datapoint and print output

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

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

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

print("Accuracy:" + str(round(100 \* accuracy, 2)) + "%")

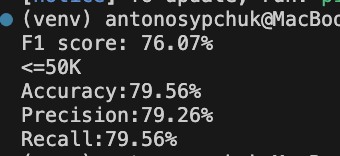
precision = precision\_score(y\_test, y\_test\_pred, average="weighted")

print("Precision:" + str(round(100 \* precision, 2)) + "%")

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

print("Recall:" + str(round(100 \* recall, 2)) + "%")

Результат виконання:



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

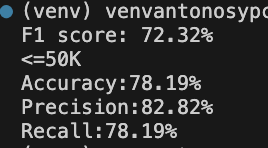
Поліноміальне (cannot run):

classifier = OneVsOneClassifier(SVC(kernel='poly', degree=8))

Гаусове:

classifier = OneVsOneClassifier(SVC(kernel='rbf', random\_state=0))

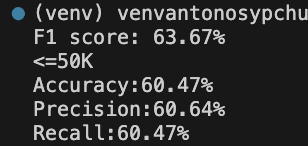
Результат:



Сигмоїдальне:

classifier = OneVsOneClassifier(SVC(kernel='sigmoid', random\_state=0))

Результат:



З поданих результатів можемо зробити висновок, що класифікатор з гаусовим ядром має кращу точність ніж з лінійним або ж сигмоїдальним, тому для кращої точності це ядро стане кращим вибором.

**2.3.** Порівняння якості класифікаторів на прикладі класифікації сортів ірисів.

Лістинг програми:

from sklearn.datasets import load\_iris

iris\_dataset = load\_iris()

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

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

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

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

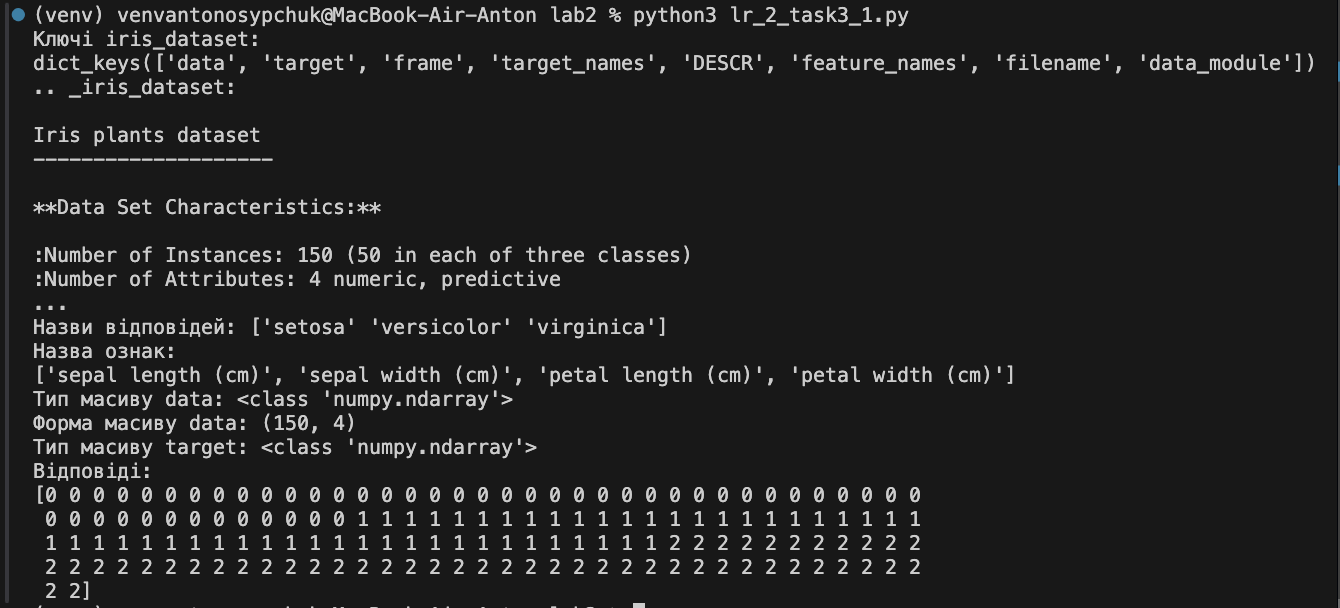
print("Тип масиву data: {}".format(type(iris\_dataset['data'])))

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

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

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

Результат виконання:



Лістинг програми:

import numpy as np

from pandas import read\_csv

from pandas.plotting import scatter\_matrix

from matplotlib import pyplot

from sklearn.datasets import load\_iris

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

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import StratifiedKFold

from sklearn.metrics import classification\_report

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import accuracy\_score

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis

from sklearn.naive\_bayes import GaussianNB

from sklearn.svm import SVC

import matplotlib

matplotlib.get\_backend()

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)

# shape

print(dataset.shape)

# Зріз даних head

print(dataset.head(20))

# Стастичні зведення методом describe

print(dataset.describe())

# Розподіл за атрибутом class

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

# Діаграма розмаху

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

sharey=False)

pyplot.show()

# Гістограма розподілу атрибутів датасета

dataset.hist()

pyplot.show()

#Матриця діаграм розсіювання

scatter\_matrix(dataset)

pyplot.show()

# Розділення датасету на навчальну та контрольну вибірки

array = dataset.values

# Вибір перших 4-х стовпців

X = array[:,0:4]

# Вибір 5-го стовпця

y = array[:,4]

# Разделение X и y на обучающую и контрольную выборки

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

test\_size=0.20, random\_state=1)

# Завантажуємо 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 = []

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()))

# Порівняння алгоритмів

pyplot.boxplot(results, labels=names)

pyplot.title('Algorithm Comparison')

pyplot.show()

# Створюємо прогноз на контрольній вибірці

model = SVC(gamma='auto')

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

predictions = model.predict(X\_validation)

# Оцінюємо прогноз

print(accuracy\_score(Y\_validation, predictions))

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

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

# Завантаження даних Iris

iris = load\_iris()

X, Y = iris.data, iris.target

# Розділення даних на навчальну та тестову вибірки

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

# Новий вхідний масив

new\_data = np.array([[5.0, 2.9, 1.0, 0.2]])

shape = new\_data.shape

print('Форма масиву:', shape)

# Ініціалізація та навчання моделі

svc\_model = SVC(gamma='auto')

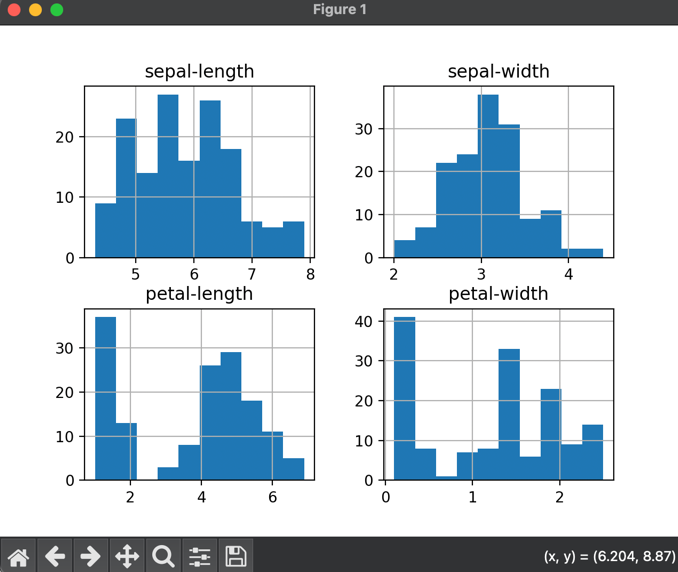
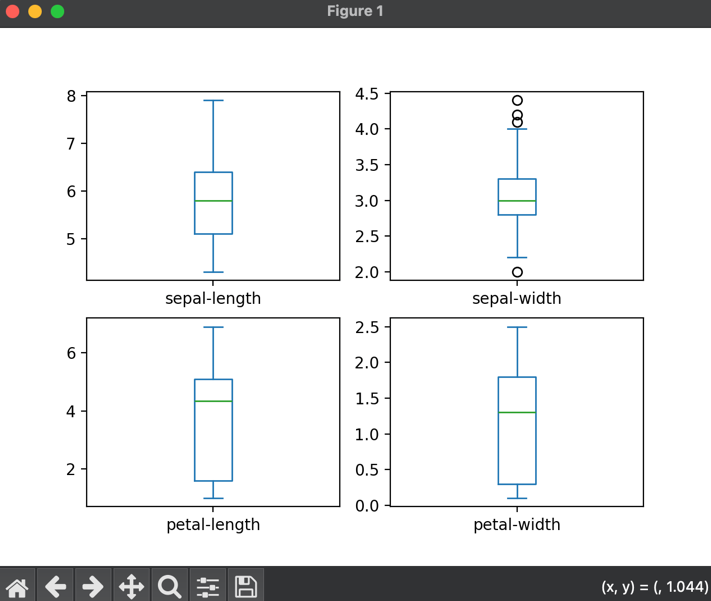
svc\_model.fit(X\_train, Y\_train)

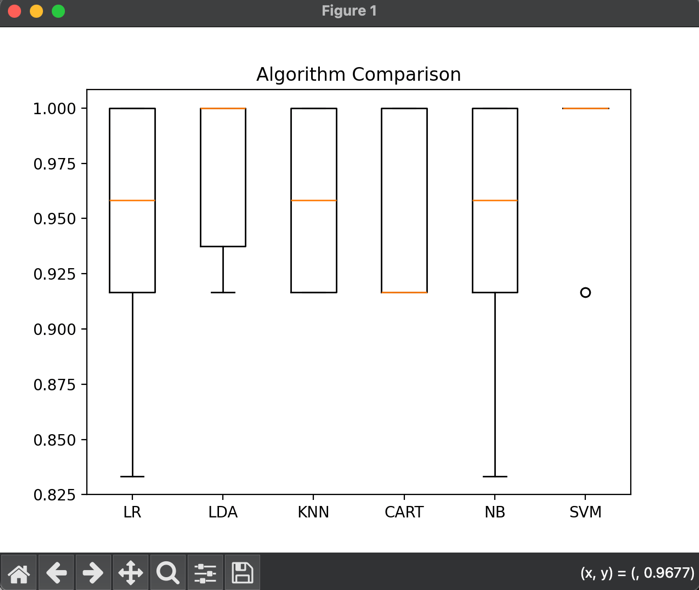
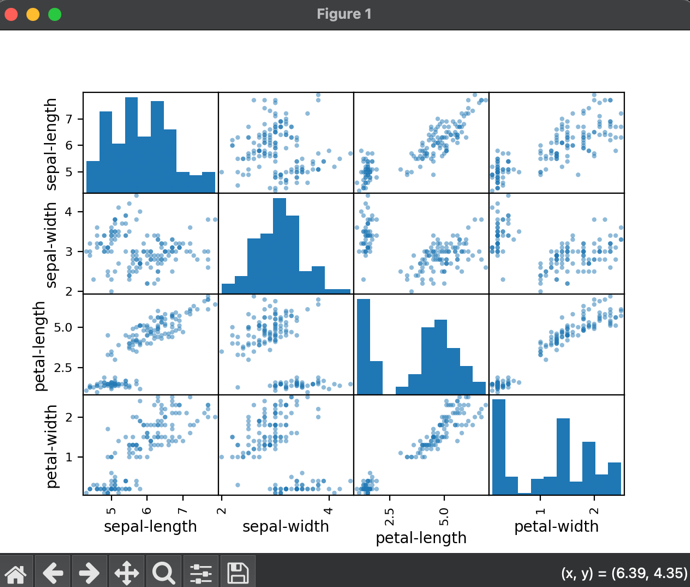
# Прогнозування на нових даних

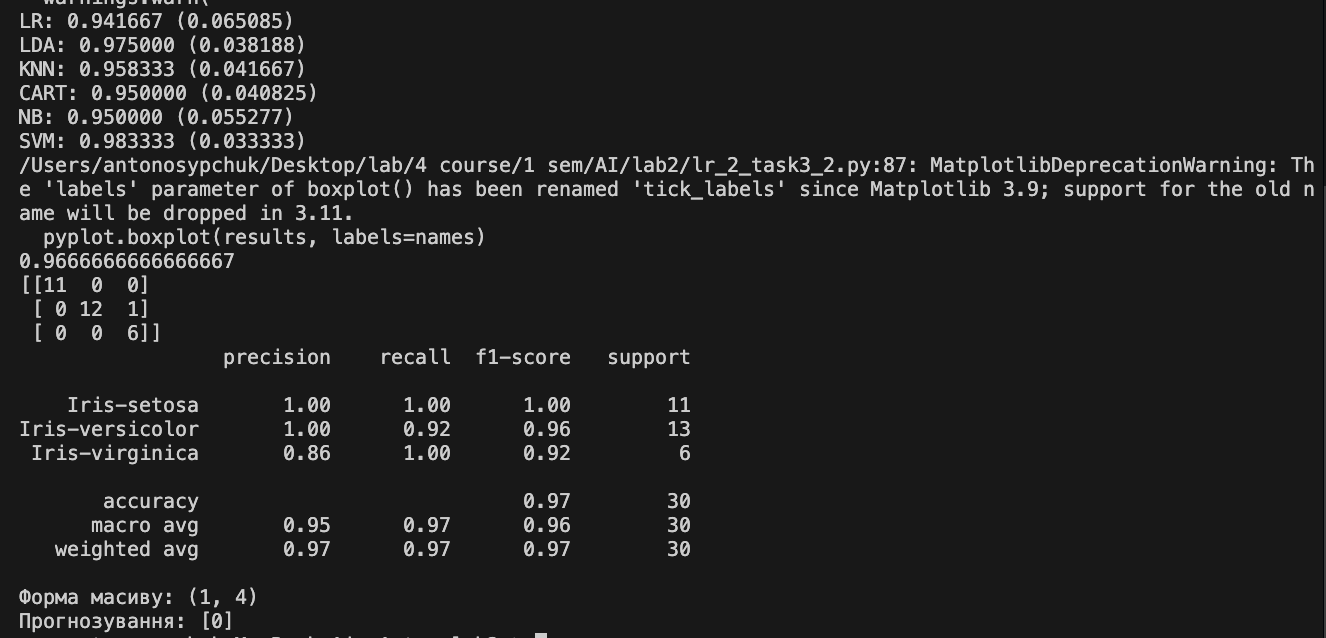
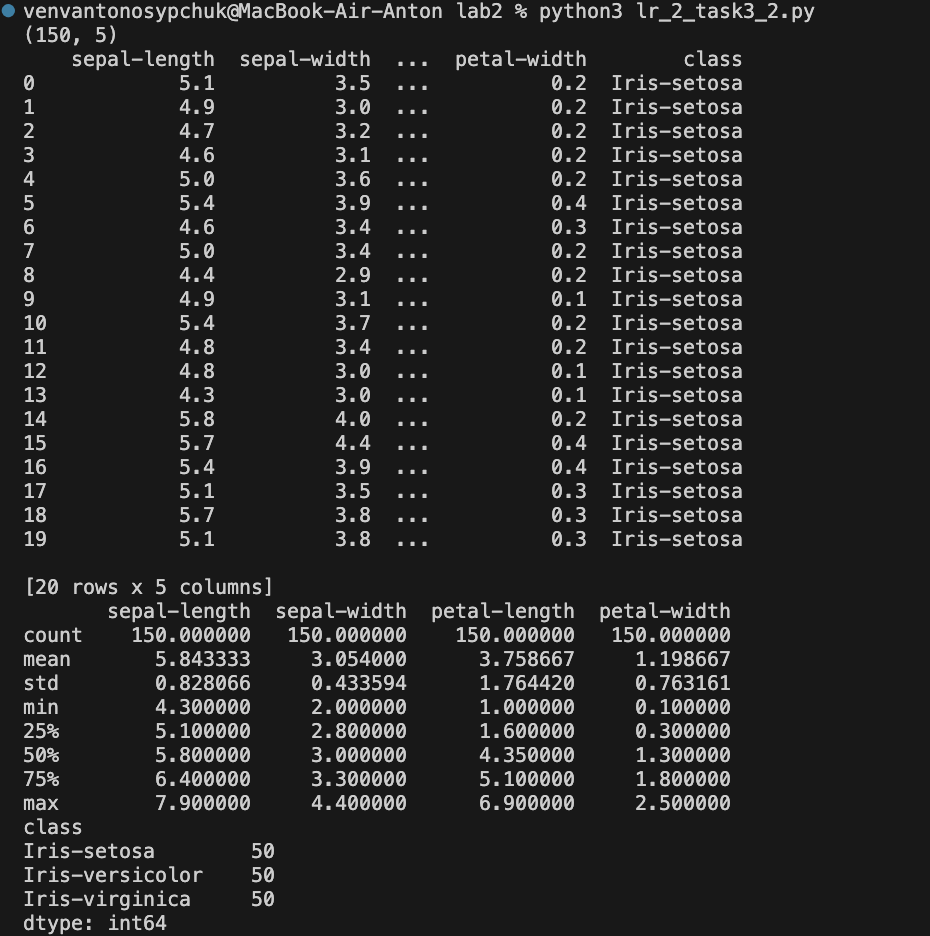
new\_prediction = svc\_model.predict(new\_data)

print('Прогнозування:', new\_prediction)

Результат виконання:







Після багаторазового тестування кожного з методів класифікації, було визначено, що для розв'язання даної задачі найефективнішим є метод опорних векторів. Цей алгоритм продемонстрував найвищу середню точність серед усіх моделей, а також найменше стандартне відхилення результатів, що свідчить про його стабільність.

Під час тренування моделі з використанням цього методу вдалося досягти точності класифікації близько 96.7% (~0.967), при цьому правильно визначений клас ірису – **Iris-setosa.**

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

Лістинг програми:

import numpy as np

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

from sklearn.linear\_model import LogisticRegression

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis

from sklearn.neighbors import KNeighborsClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.naive\_bayes import GaussianNB

from sklearn.svm import SVC

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

from sklearn import preprocessing

# Load data

input\_file = './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)

# Encode categorical features

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)

# Split dataset into training and testing sets

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

# Define classifiers

models = [

('LR', LogisticRegression(solver='liblinear')),

('LDA', LinearDiscriminantAnalysis()),

('KNN', KNeighborsClassifier()),

('CART', DecisionTreeClassifier()),

('NB', GaussianNB()),

# ('SVM', SVC(kernel='linear', random\_state=1))

]

# Evaluate each model

results = []

names = []

print("Classifier Performance:")

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}: Mean accuracy = {cv\_results.mean():.4f}, Std = {cv\_results.std():.4f}")

# Compare algorithms

import matplotlib.pyplot as plt

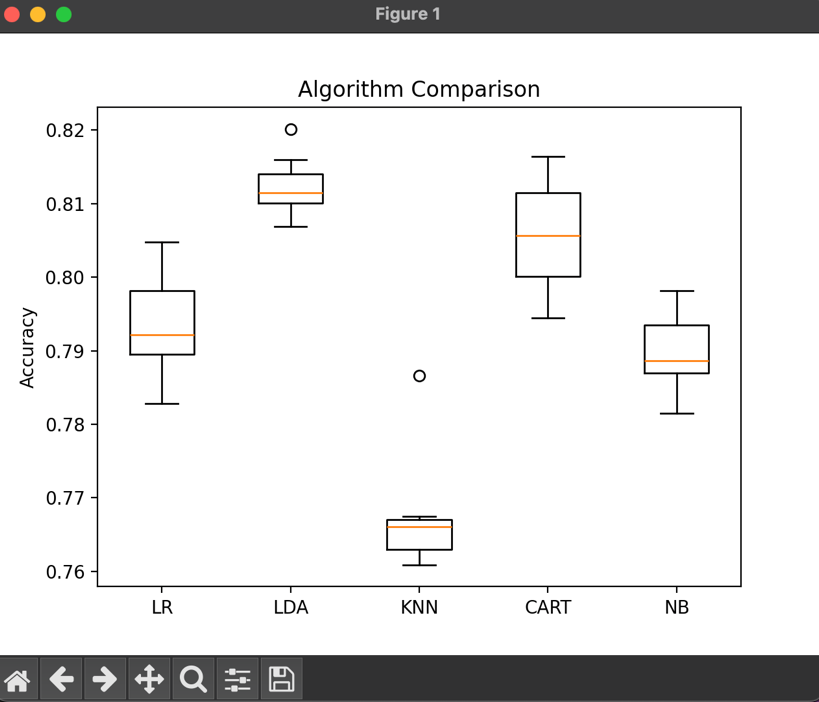
plt.boxplot(results, labels=names)

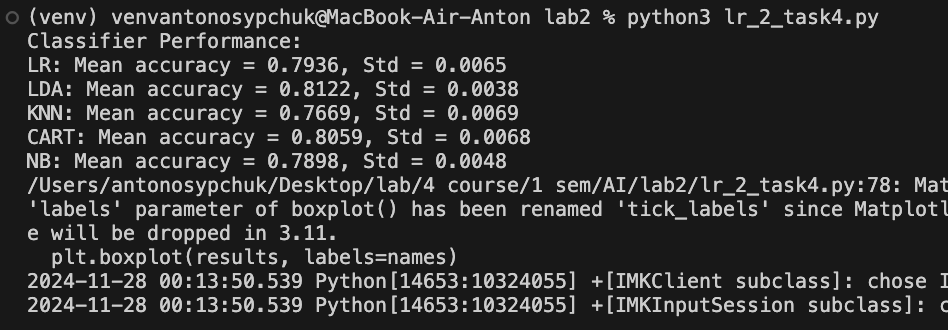
plt.title('Algorithm Comparison')

plt.ylabel('Accuracy')

plt.show()

Результат виконання:





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

Лістинг програми:

import seaborn as sns

import numpy as np

from io import BytesIO

from matplotlib import pyplot as plt

from sklearn.datasets import load\_iris

from sklearn.linear\_model import RidgeClassifier

from sklearn.metrics import confusion\_matrix

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

from sklearn import metrics

iris = load\_iris()

X, y = iris.data, iris.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=0)

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

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

ypred = clf.predict(X\_test)

print('Accuracy:', np.round(metrics.accuracy\_score(y\_test, ypred), 4))

print('Precision:', np.round(metrics.precision\_score(y\_test, ypred, average='weighted'), 4))

print('Recall:', np.round(metrics.recall\_score(y\_test, ypred, average='weighted'), 4))

print('F1 Score:', np.round(metrics.f1\_score(y\_test, ypred, average='weighted'), 4))

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

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

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

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

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

plt.xlabel('True Label')

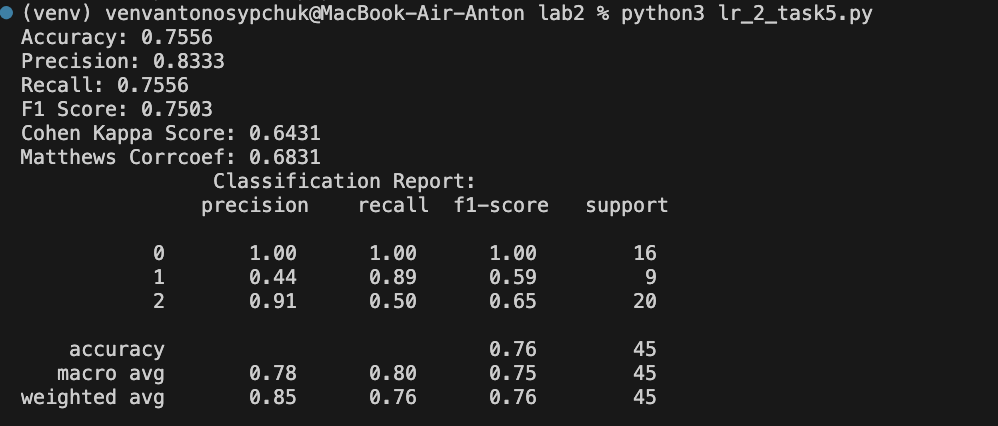
plt.ylabel('Predicted Label')

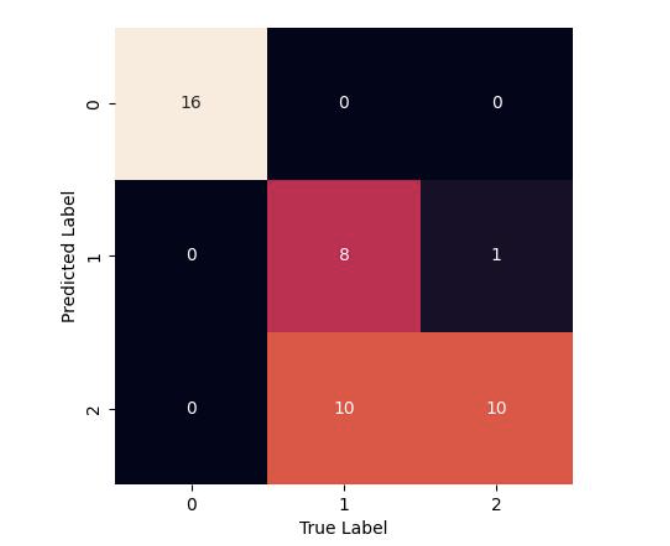
plt.savefig("Confusion.jpg")

f = BytesIO()

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

Результат виконання:





**Налаштування класифікатора Ridge**

tol=1e-2: Допуск для критерію зупинки. Це визначає точність оптимізації. Якщо зміна втраченого значення (loss) між ітераціями менша за це значення, оптимізація припиняється.

solver='sag': Алгоритм оптимізації "Stochastic Average Gradient Descent". Цей метод добре працює з великими наборами даних та ітеративно зменшує похибку, обробляючи міні-батчі даних.

RidgeClassifier є варіантом лінійного класифікатора, який додає регуляризацію L2 для боротьби з переобученням. Він підходить для класифікації задач, які вимагають балансування між простотою моделі та точністю.

**Показники якості та результати**

Accuracy: Частка правильно класифікованих прикладів.

Precision (зважена середня): Частка правильно передбачених позитивних класів серед усіх передбачень для кожного класу.

Recall (зважена середня): Частка правильно передбачених позитивних класів серед усіх фактичних позитивних зразків.

F1 Score (зважена середня): Гармонійне середнє Precision та Recall. Використовується для балансування між ними.

Cohen Kappa Score: Міра узгодженості між передбаченнями та реальними мітками, яка враховує випадкову згоду.

Matthews Corrcoef: Кореляційний коефіцієнт між передбаченнями та фактичними мітками. Він враховує дисбаланс у даних.

**Пояснення зображення Confusion.jpg**

Істинний клас 0 був правильно передбачений для 16 зразків. Істинний клас 1 був передбачений як 2 для 10 зразків, що вказує на значний confusion між цими класами. Матриця дозволяє зрозуміти, які класи модель плутає найчастіше.

**Пояснення коефіцієнтів**

Коефіцієнт Коена Каппа (Cohen Kappa Score) – це статистичний показник узгодженості між передбаченнями моделі та істинними мітками, з урахуванням випадкової згоди.

Коефіцієнт кореляції Метьюза (Matthews Correlation Coefficient, MCC) – це міра якості класифікації, яка враховує всі чотири значення матриці плутанини (TP, TN, FP, FN). MCC підходить для задач із дисбалансом даних.