**Лабораторна робота 1**

**ПОПЕРЕДНЯ ОБРОБКА ТА КОНТРОЛЬОВАНА КЛАСИФІКАЦІЯ ДАНИХ**

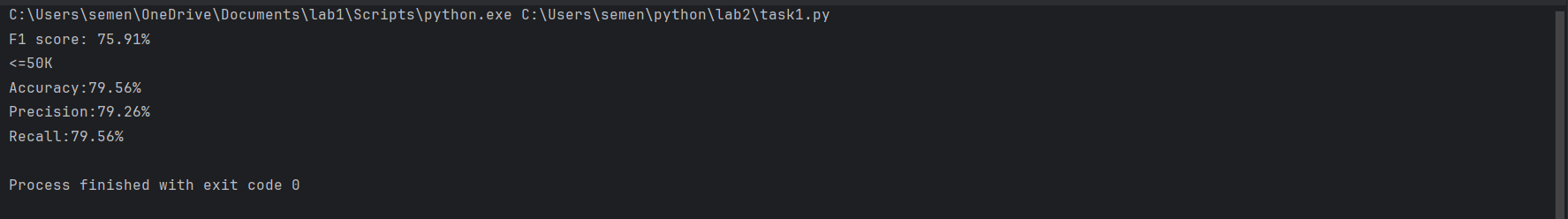
Завдання 2.1. Попередня обробка даних

Код програми

import numpy as np  
from sklearn import preprocessing  
from sklearn.metrics import precision\_score, recall\_score, accuracy\_score  
from sklearn.svm import LinearSVC  
from sklearn.multiclass import OneVsOneClassifier  
from sklearn.model\_selection import cross\_val\_score, train\_test\_split  
  
# Load data  
input\_file = '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:



Тестова точка належить до класу з доходом нижче рівне 50000 (з ймовірністю близько 79%).

Завдання 2.2. Порівняння якості класифікаторів SVM з нелінійними ядрами

Поліноміальне ядро:

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

Гаусове ядро:

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

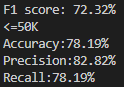
Сигмоїдальне ядро:

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

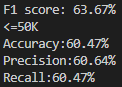
Результат виконання:

Поліноміальне ядро: код так і не виконався

Гаусове ядро:



Сигмоїдальне ядро:



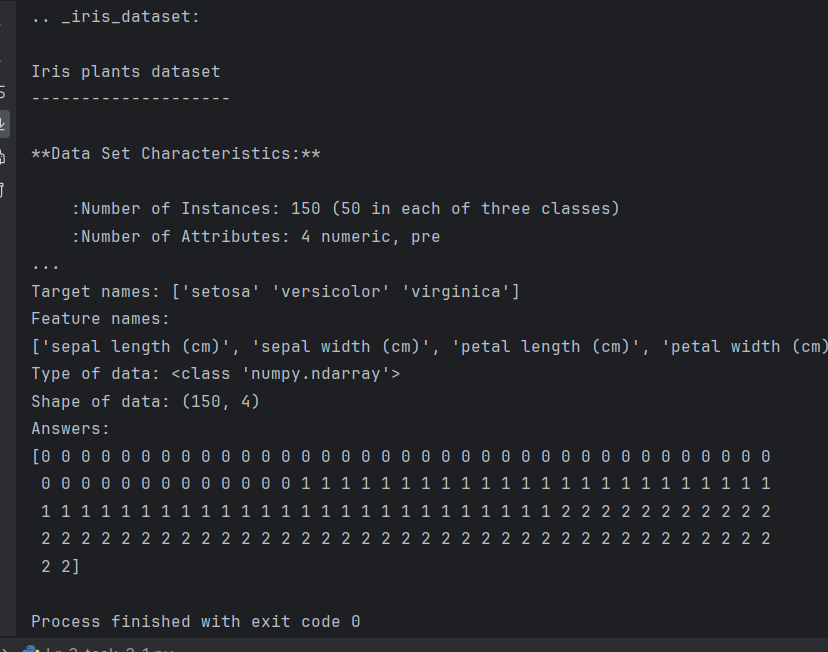
В середньому класифікатори з лінійним ядром та гаусовим ядром мають схожі показники, проте класифікатор з гаусовим ядром має помітно кращу точність (precision), тому якщо важливий саме цей показник, то це ядро найкраще.

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

Лістинг коду для ознайомлення зі структурою даних:

from sklearn.datasets import load\_iris  
  
iris\_dataset = load\_iris()  
  
print("Keys of iris\_dataset: \n{}".format(iris\_dataset.keys()))  
print(iris\_dataset['DESCR'][:193] + "\n...")  
print("Target names: {}".format(iris\_dataset['target\_names']))  
print("Feature names: \n{}".format(iris\_dataset['feature\_names']))  
print("Type of data: {}".format(type(iris\_dataset['data'])))  
print("Shape of data: {}".format(iris\_dataset['data'].shape))  
print("Answers: \n{}".format(iris\_dataset['target']))

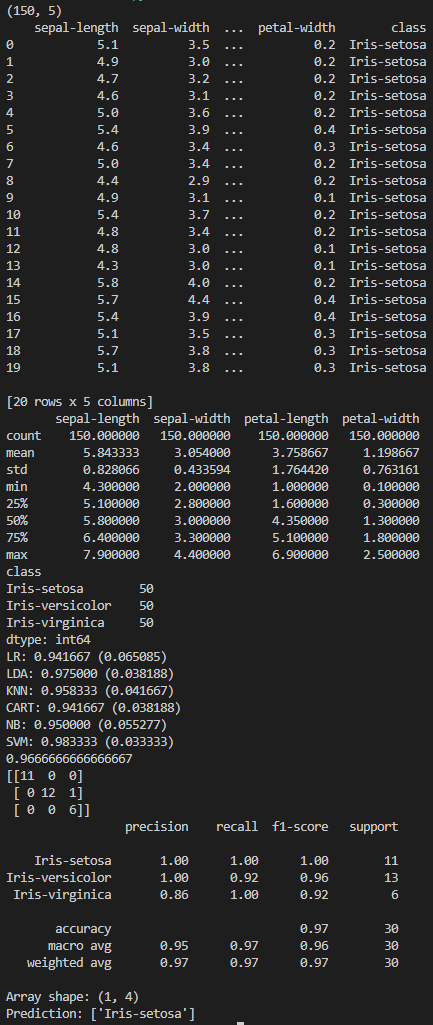
результат:

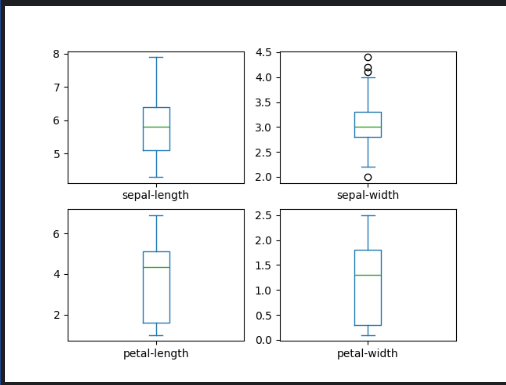


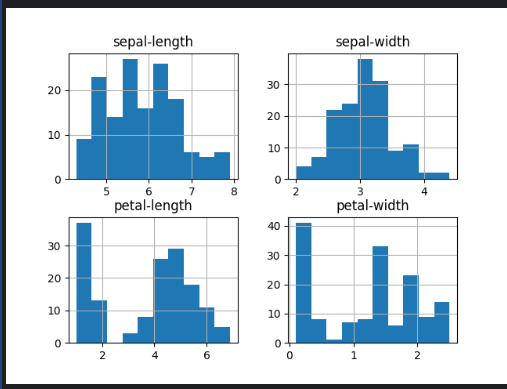
3.2

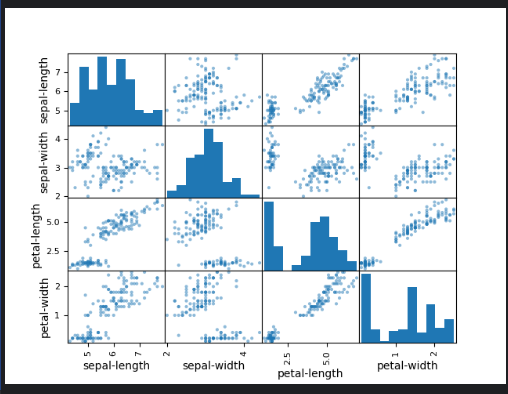
from pandas import read\_csv  
from pandas.plotting import scatter\_matrix  
from matplotlib import pyplot  
from sklearn.model\_selection import train\_test\_split  
from sklearn.model\_selection import cross\_val\_score  
from sklearn.model\_selection import StratifiedKFold  
from sklearn.metrics import classification\_report  
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  
from sklearn.datasets import load\_iris  
import numpy as np  
  
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)  
  
# Head  
print(dataset.head(20))  
  
# Descriptions  
print(dataset.describe())  
  
# Class distribution  
print(dataset.groupby('class').size())  
  
# Box and whisker plots  
dataset.plot(kind='box', subplots=True, layout=(2, 2), sharex=False, sharey=False)  
pyplot.show()  
  
# Histograms  
dataset.hist()  
pyplot.show()  
  
# Scatter plot matrix  
scatter\_matrix(dataset)  
pyplot.show()  
  
# Split-out validation dataset  
array = dataset.values  
  
# All rows, columns 0-4  
X = array[:, 0:4]  
  
# All rows, column 4  
y = array[:, 4]  
  
# 80% training, 20% validation  
X\_train, X\_validation, Y\_train, Y\_validation = train\_test\_split(X, y, test\_size=0.20, random\_state=1)  
  
# Test options and evaluation metric  
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')))  
  
# Evaluate each model in turn  
results = []  
names = []  
  
for name, model in models:  
 # 10-fold cross validation  
 kfold = StratifiedKFold(n\_splits=10, random\_state=1, shuffle=True)  
  
 # Evaluate model on 10 different splits of the dataset  
 cv\_results = cross\_val\_score(model, X\_train, Y\_train, cv=kfold, scoring='accuracy')  
  
 # Store results  
 results.append(cv\_results)  
 names.append(name)  
  
 # Summarize performance  
 print('%s: %f (%f)' % (name, cv\_results.mean(), cv\_results.std()))  
  
# Compare algorithms  
pyplot.boxplot(results, labels=names)  
pyplot.title('Algorithm Comparison')  
pyplot.show()  
  
# Make predictions on validation dataset  
model = SVC(gamma='auto')  
model.fit(X\_train, Y\_train)  
predictions = model.predict(X\_validation)  
  
# Evaluate predictions  
print(accuracy\_score(Y\_validation, predictions))  
print(confusion\_matrix(Y\_validation, predictions))  
print(classification\_report(Y\_validation, predictions))  
  
# Load iris dataset  
iris\_dataset = load\_iris()  
  
# Create a NumPy array with the new iris data  
new\_data = np.array([[5.0, 2.9, 1.0, 0.2]])  
  
# Get the shape of the array  
shape = new\_data.shape  
  
# Print the shape of the array  
print('Array shape:', shape)  
  
# Define KNN model  
model = SVC(gamma='auto')  
  
# Fit the model to the training data  
model.fit(X\_train, Y\_train)  
  
# Make predictions on new data  
new\_prediction = model.predict(new\_data)  
  
# Print the prediction  
print('Prediction:', new\_prediction)

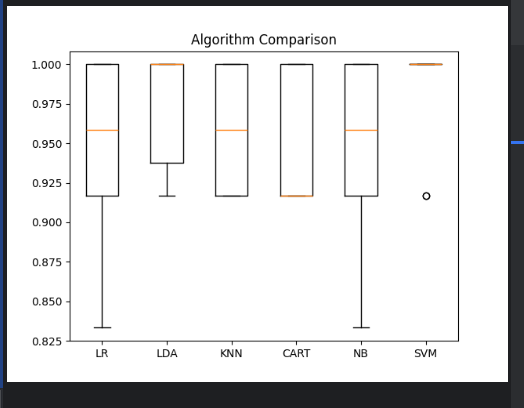
Результат:









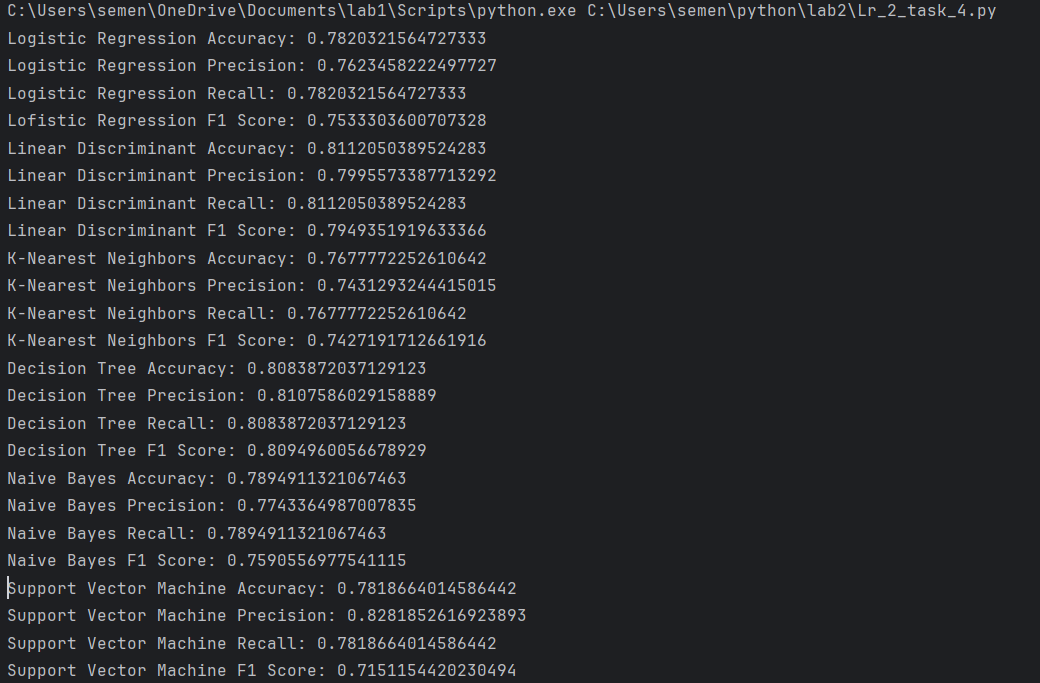


**Завдання 2.4.** Порівняння якості класифікаторів для набору даних завдання 2.1.

Лістинг:

import numpy as np  
from sklearn import preprocessing  
from sklearn.multiclass import OneVsOneClassifier, OneVsRestClassifier  
from sklearn.model\_selection import train\_test\_split  
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, f1\_score, precision\_score, recall\_score  
  
# Load data  
input\_file = '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)  
  
# Cross validation  
X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=5)  
  
# Train and evaluate the logistic regression model  
lr\_model = LogisticRegression(max\_iter=10000)  
lr\_model.fit(X\_train, Y\_train)  
lr\_predictions = lr\_model.predict(X\_test)  
  
print('Logistic Regression Accuracy:', accuracy\_score(Y\_test, lr\_predictions))  
print('Logistic Regression Precision:', precision\_score(Y\_test, lr\_predictions, average='weighted'))  
print('Logistic Regression Recall:', recall\_score(Y\_test, lr\_predictions, average='weighted'))  
print('Lofistic Regression F1 Score:', f1\_score(Y\_test, lr\_predictions, average='weighted'))  
  
# Train and evaluate the linear discriminant analysis model  
lda\_model = LinearDiscriminantAnalysis()  
lda\_model.fit(X\_train, Y\_train)  
lda\_predictions = lda\_model.predict(X\_test)  
  
print('Linear Discriminant Accuracy:', accuracy\_score(Y\_test, lda\_predictions))  
print('Linear Discriminant Precision:', precision\_score(Y\_test, lda\_predictions, average='weighted'))  
print('Linear Discriminant Recall:', recall\_score(Y\_test, lda\_predictions, average='weighted'))  
print('Linear Discriminant F1 Score:', f1\_score(Y\_test, lda\_predictions, average='weighted'))  
  
# Train and evaluate the k-nearest neighbors model  
knn\_model = KNeighborsClassifier()  
knn\_model.fit(X\_train, Y\_train)  
knn\_predictions = knn\_model.predict(X\_test)  
  
print('K-Nearest Neighbors Accuracy:', accuracy\_score(Y\_test, knn\_predictions))  
print('K-Nearest Neighbors Precision:', precision\_score(Y\_test, knn\_predictions, average='weighted'))  
print('K-Nearest Neighbors Recall:', recall\_score(Y\_test, knn\_predictions, average='weighted'))  
print('K-Nearest Neighbors F1 Score:', f1\_score(Y\_test, knn\_predictions, average='weighted'))  
  
# Train and evaluate the decision tree model  
cart\_model = DecisionTreeClassifier()  
cart\_model.fit(X\_train, Y\_train)  
cart\_predictions = cart\_model.predict(X\_test)  
  
print('Decision Tree Accuracy:', accuracy\_score(Y\_test, cart\_predictions))  
print('Decision Tree Precision:', precision\_score(Y\_test, cart\_predictions, average='weighted'))  
print('Decision Tree Recall:', recall\_score(Y\_test, cart\_predictions, average='weighted'))  
print('Decision Tree F1 Score:', f1\_score(Y\_test, cart\_predictions, average='weighted'))  
  
# Train and evaluate the naive bayes model  
nb\_model = GaussianNB()  
nb\_model.fit(X\_train, Y\_train)  
nb\_predictions = nb\_model.predict(X\_test)  
  
print('Naive Bayes Accuracy:', accuracy\_score(Y\_test, nb\_predictions))  
print('Naive Bayes Precision:', precision\_score(Y\_test, nb\_predictions, average='weighted'))  
print('Naive Bayes Recall:', recall\_score(Y\_test, nb\_predictions, average='weighted'))  
print('Naive Bayes F1 Score:', f1\_score(Y\_test, nb\_predictions, average='weighted'))  
  
# Train and evaluate the support vector machine model  
svm\_model = SVC()  
svm\_model.fit(X\_train, Y\_train)  
svm\_predictions = svm\_model.predict(X\_test)  
  
print('Support Vector Machine Accuracy:', accuracy\_score(Y\_test, svm\_predictions))  
print('Support Vector Machine Precision:', precision\_score(Y\_test, svm\_predictions, average='weighted'))  
print('Support Vector Machine Recall:', recall\_score(Y\_test, svm\_predictions, average='weighted'))  
print('Support Vector Machine F1 Score:', f1\_score(Y\_test, svm\_predictions, average='weighted'))

Результат:

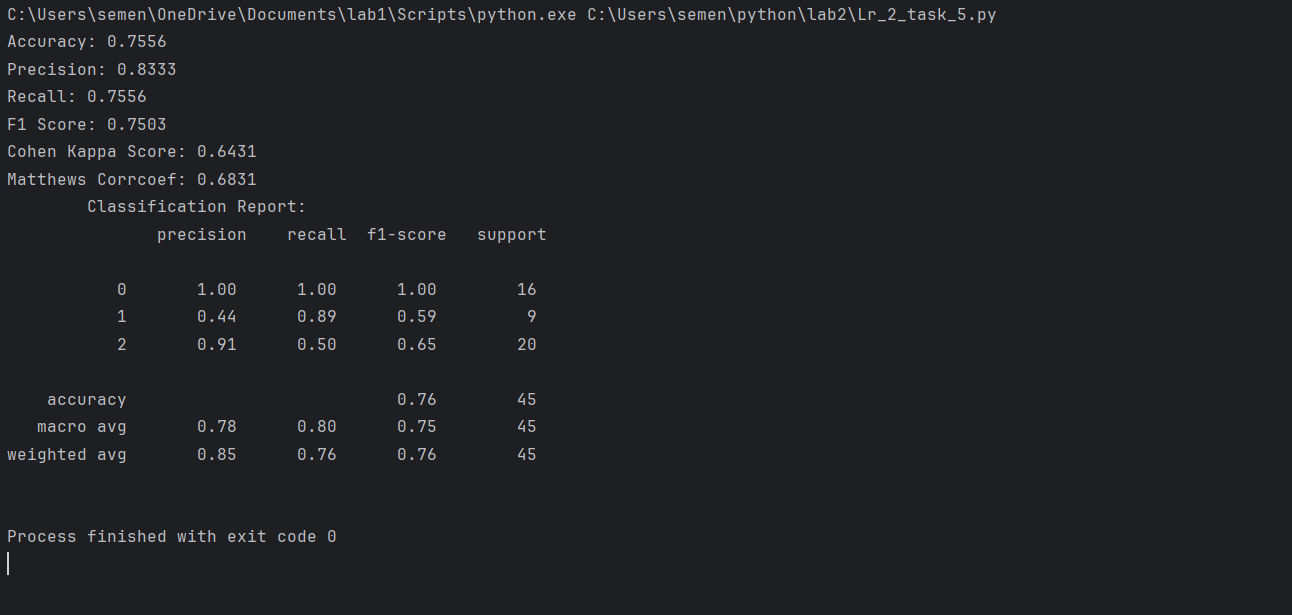


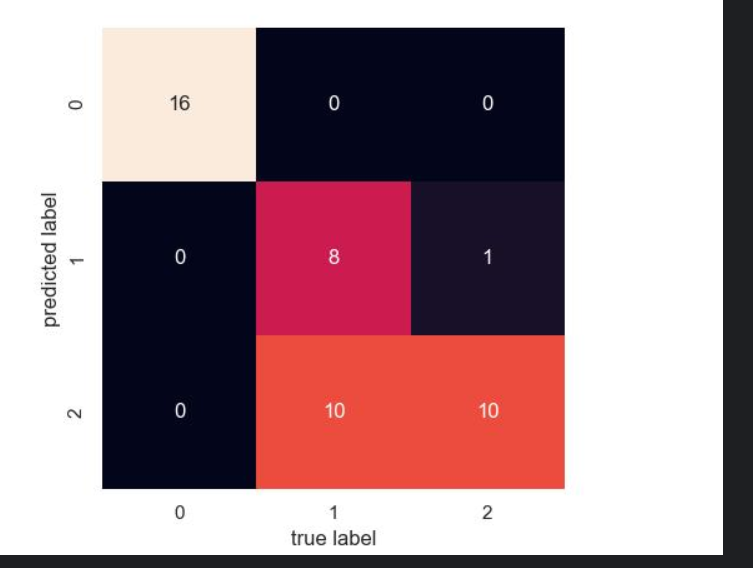
**Завдання 2.5.** Класифікація даних лінійним класифікатором Ridge.

Лістинг:

import numpy as np  
from sklearn.datasets import load\_iris  
from sklearn.linear\_model import RidgeClassifier  
from sklearn.metrics import confusion\_matrix  
from io import BytesIO  
import seaborn as sns  
from sklearn.model\_selection import train\_test\_split; sns.set()  
import matplotlib.pyplot as plt  
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")  
  
# Save SVG in a fake file object.  
f = BytesIO()  
plt.savefig(f, format = "svg")

Результат





Функція RidgeClassifier має два параметри tol та solver. Параметр tol це невелике позитивне число, яке визначає критерій зупинки для алгоритму оптимізації. У цьому коді для параметра tol встановлено значення 1e-2, що означає, що алгоритм оптимізації зупиниться, коли зміна цільової функції буде меншою за 0,01. Параметр solver визначає алгоритм для оптимізації. Це може бути одне з чотирьох значень: «auto», «svd», «cholesky» або «sag». У цьому коді параметр дорівнює значенню «sag», що означає використання алгоритму стохастичного середнього градієнтного спаду.

В програмі використовуються такі показники якості:

Точність (Accuracy) - 0.7556

Точність (Precision) - 0.8333

Показник повноти (Recall) - 0.7556

Оцінка F1 (F1 Score) - 0.7503

Оцінка Коена Каппа (Cohen Kappa Score) - 0.6431

Коефіцієнт кореляції Метьюза (Matthews Corr coef) - 0.6831

Оцінка Коена Каппа вимірює збіг між прогнозованими та фактичними мітками, враховуючи можливість випадкового збігу.

Коефіцієнт кореляції Метьюза вимірює кореляцію між прогнозованими та фактичними мітками, беручи до уваги можливість випадкового збігу.

Обидва методи вимірюють схожий показник, проте різними способами.

Зображення є візуальним представленням відношення кількості передбачених класів та правильних класів.