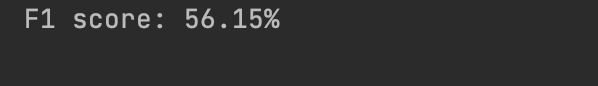
**ЛАБОРАТОРНА РОБОТА № 2**

**ПОРІВНЯННЯ МЕТОДІВ КЛАСИФІКАЦІЇ ДАНИХ**

**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 OneVsOneClassifier  
#from sklearn import cross\_validation  
from sklearn.model\_selection import train\_test\_split, cross\_val\_score  
  
  
input\_file = '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 = OneVsOneClassifier(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 = OneVsOneClassifier(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  
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]))  
 count += 1  
  
input\_data\_encoded = np.array(input\_data\_encoded)  
  
  
predicted\_class = classifier.predict(input\_data\_encoded)  
print(label\_encoder[-1].inverse\_transform(predicted\_class)[0])



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

**ядрами**

Найкраще підходить для нашого завдання.

1. Поліноміальне ядро. У разі поліноміального ядра ви також повинні передати значення для параметра degree класу SVC. Це переважно ступінь многочлена. Фактично у попередньому коді вам необхідно замінити лінійний параметр на: KernelSVC(kernel='poly', degree=8):

Не забудьте імпортувати відповідну функцію з бібліотеки.Вся решта коду повинна працювати.

2. Гаусове ядро. Ми можемо використовувати гаусове ядро для реалізації kernel SVM: KernelSVC(kernel='rbf')

Щоб використовувати ядро Гауса, ви повинні вказати 'rbf' як значення параметра ядра класу SVC.

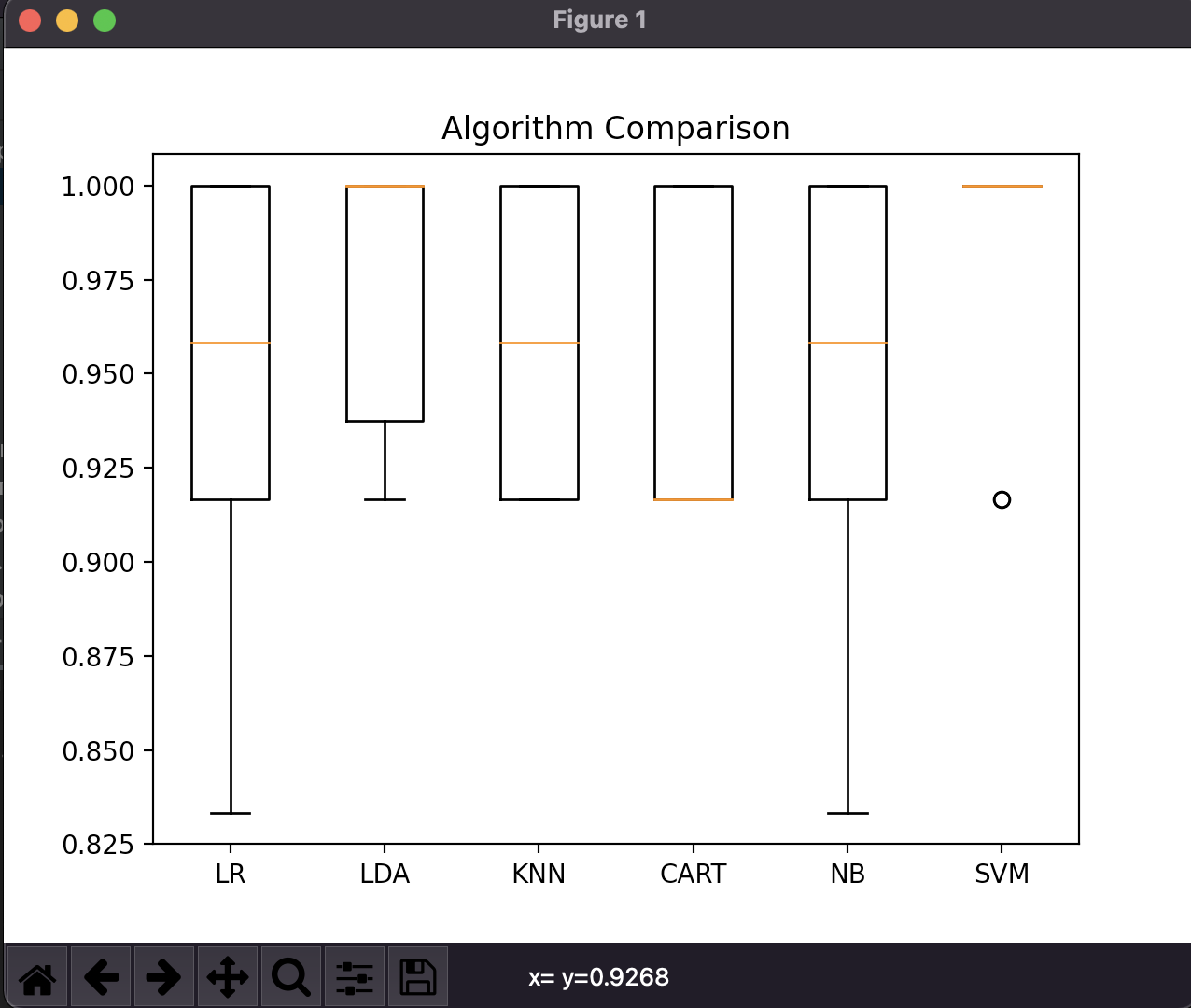
3. Сигмоїдальне ядро. Щоб використовувати сигмоїдальне ядро, ви повинні вказати 'sigmoid' як значення для параметра kernel класу SVC .

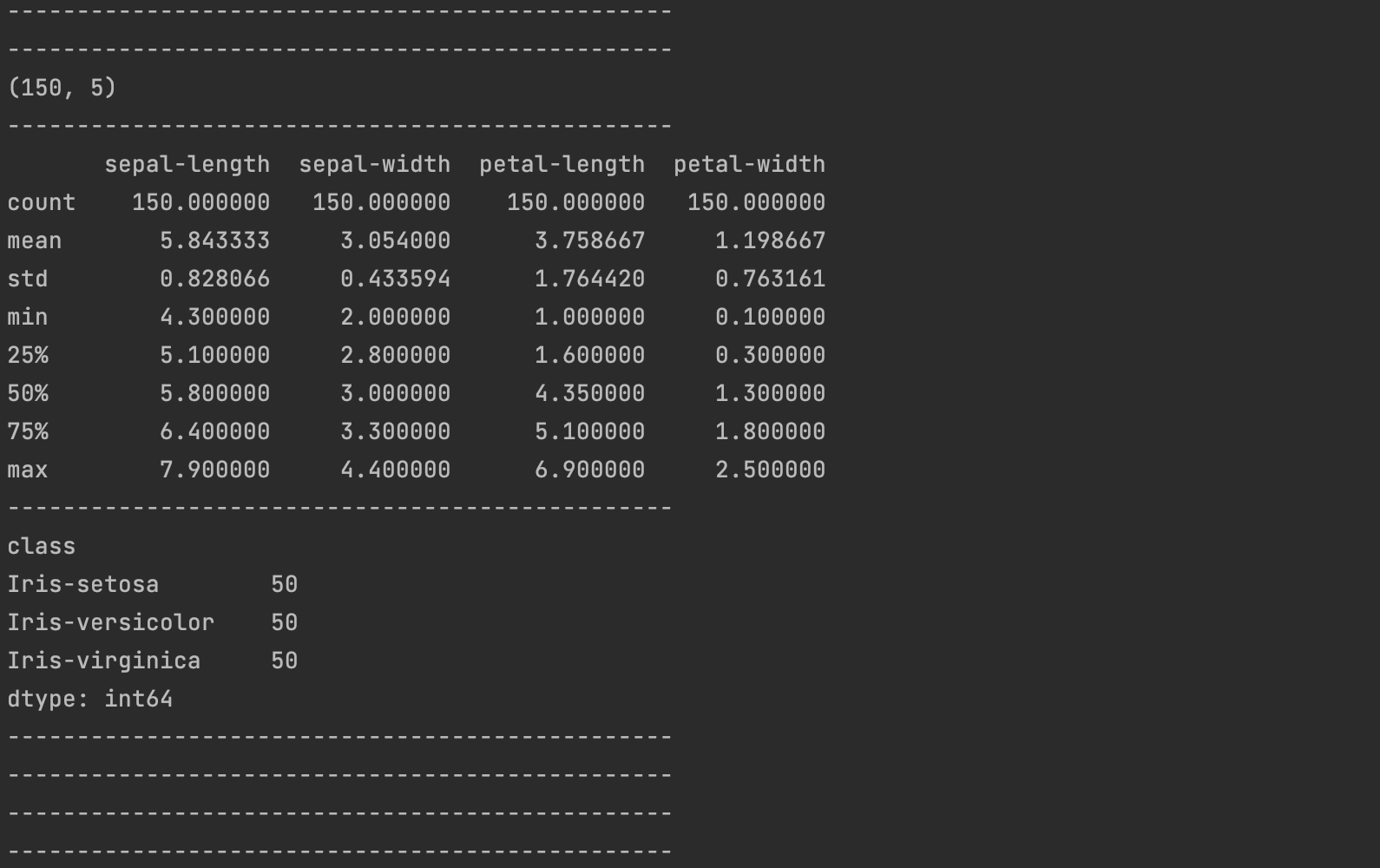
KernelSVC(kernel='sigmoid')

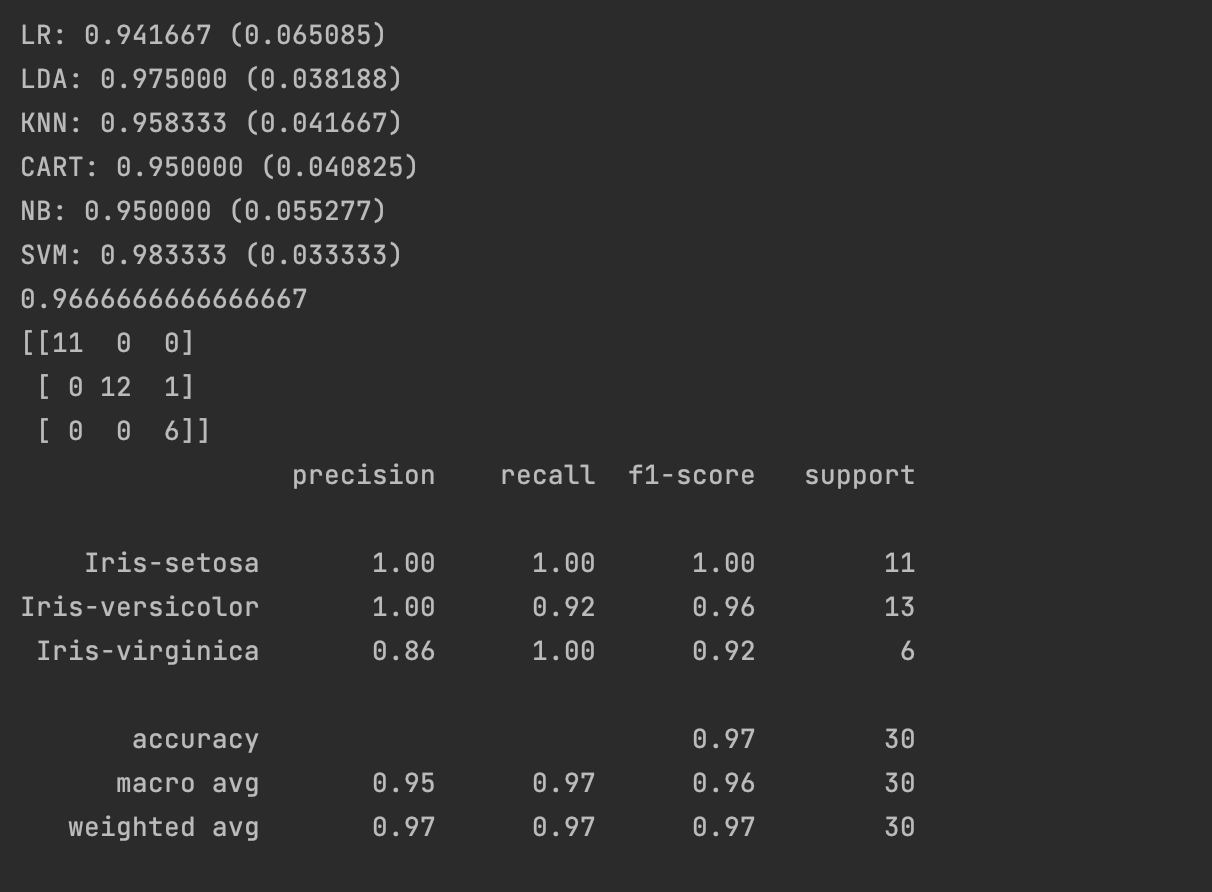
import numpy as np

import matplotlib.pyplot as plt  
from sklearn import preprocessing  
from sklearn.model\_selection import train\_test\_split, cross\_val\_score  
from sklearn.svm import LinearSVC  
from sklearn.multiclass import OneVsOneClassifier  
  
  
  
input\_file = '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 = OneVsOneClassifier(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 = OneVsOneClassifier(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  
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]))  
 count += 1  
  
input\_data\_encoded = np.array(input\_data\_encoded)  
  
  
predicted\_class = classifier.predict(input\_data\_encoded)  
print(label\_encoder[-1].inverse\_transform(predicted\_class)[0])

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





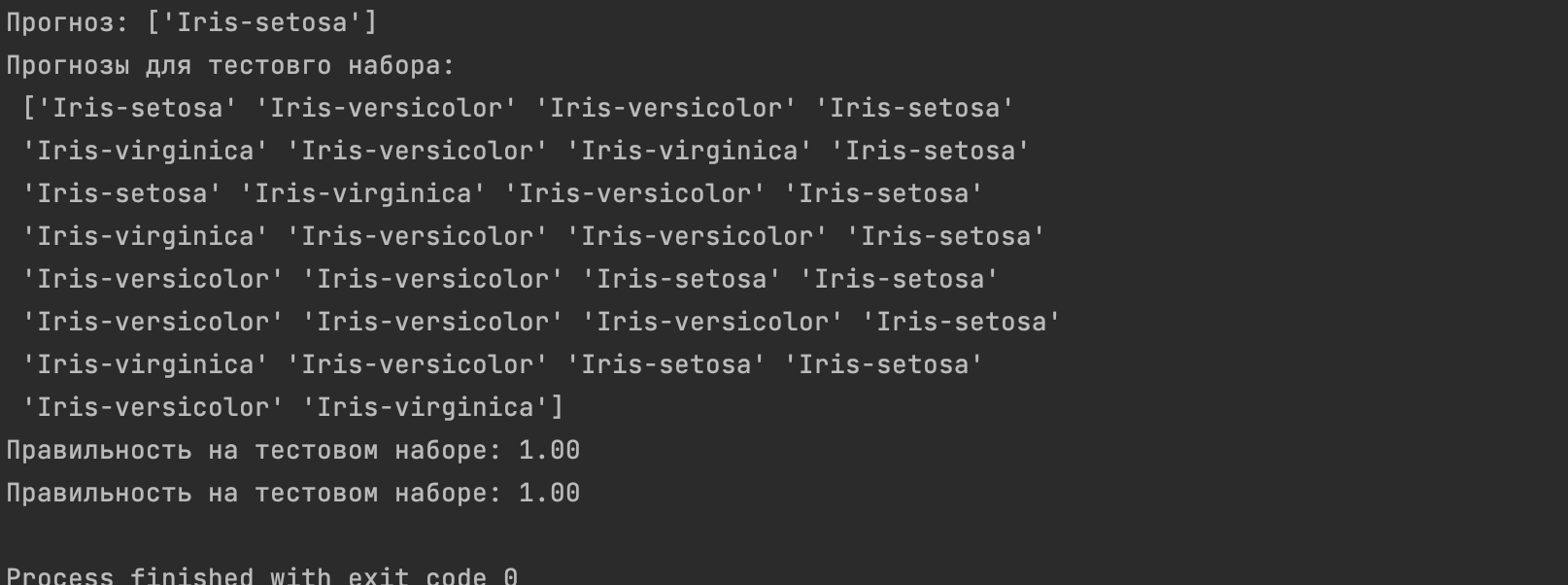


from pandas import read\_csv

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

print("------------------------------------------------")  
  
# Load dataset  
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("------------------------------------------------")  
print(dataset.shape)  
  
# print(dataset.head(20))  
  
# descriptions  
print("------------------------------------------------")  
print(dataset.describe())  
  
# classdistribution  
print("------------------------------------------------")  
print(dataset.groupby('class').size())  
  
# boxand whisker plots  
print("------------------------------------------------")  
# dataset.plot(kind='box', subplots=True, layout=(2,2), sharex=False, sharey=False)  
# pyplot.show()  
  
print("------------------------------------------------")  
# histograms  
# dataset.hist()  
# pyplot.show()  
  
print("------------------------------------------------")  
# scatter plot matrix  
# scatter\_matrix(dataset)  
# pyplot.show()  
  
print("------------------------------------------------")  
# Split-out validation dataset  
array = dataset.values  
X = array[:, 0:4]  
y = array[:, 4]  
X\_train, X\_validation, Y\_train, Y\_validation = train\_test\_split(X, y, test\_size=0.20, random\_state=1)  
  
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:  
 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()))  
  
# 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))

ОТРИМАННЯ ПРОГНОЗУ (ЗАСТОСУВАННЯ МОДЕЛІ ДЛЯ ПЕРЕДБАЧЕННЯ)



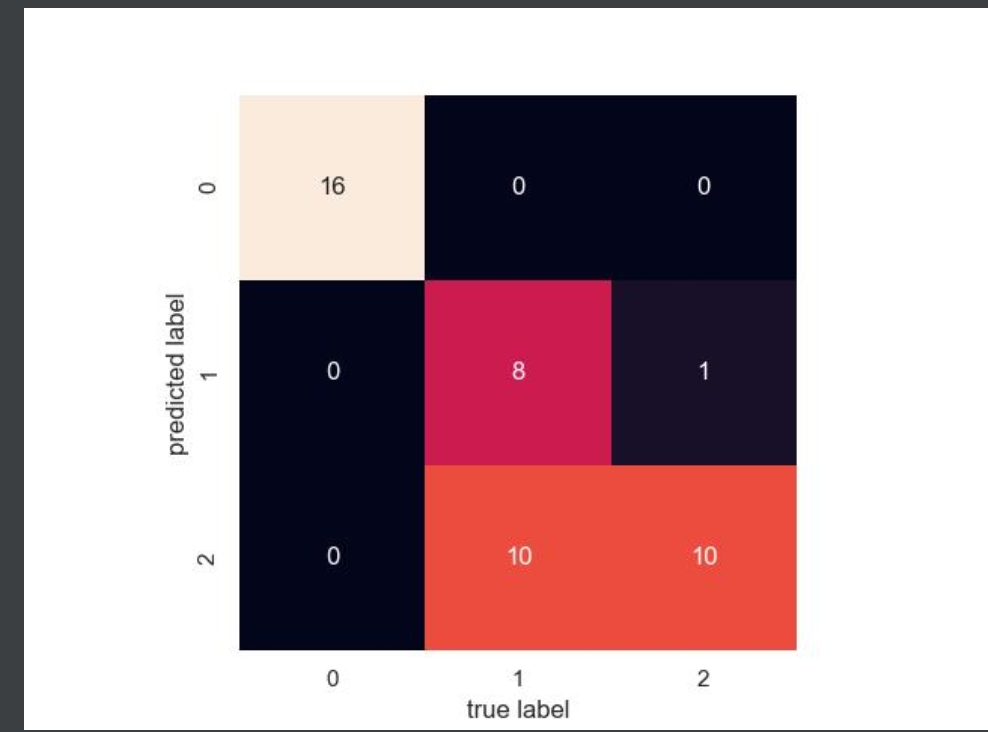
from pandas import read\_csv  
from pandas.plotting import scatter\_matrix  
from matplotlib import pyplot  
from sklearn.model\_selection import train\_test\_split  
from sklearn.model\_selection import cross\_val\_score  
from sklearn.model\_selection import StratifiedKFold  
from sklearn.metrics import 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  
  
print("------------------------------------------------")  
  
# Load dataset  
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("------------------------------------------------")  
print(dataset.shape)  
  
# print(dataset.head(20))  
  
# descriptions  
print("------------------------------------------------")  
print(dataset.describe())  
  
# classdistribution  
print("------------------------------------------------")  
print(dataset.groupby('class').size())  
  
# boxand whisker plots  
print("------------------------------------------------")  
# dataset.plot(kind='box', subplots=True, layout=(2,2), sharex=False, sharey=False)  
# pyplot.show()  
  
print("------------------------------------------------")  
# histograms  
# dataset.hist()  
# pyplot.show()  
  
print("------------------------------------------------")  
# scatter plot matrix  
# scatter\_matrix(dataset)  
# pyplot.show()  
  
print("------------------------------------------------")  
# Split-out validation dataset  
array = dataset.values  
X = array[:, 0:4]  
y = array[:, 4]  
X\_train, X\_validation, Y\_train, Y\_validation = train\_test\_split(X, y, test\_size=0.20, random\_state=1)  
  
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:  
 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()))  
  
# 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))  
  
from sklearn.neighbors import KNeighborsClassifier  
import numpy as np  
knn = KNeighborsClassifier(n\_neighbors=1)  
  
knn.fit(X\_train, Y\_train)  
X\_new = np.array([[5, 2.9, 1, 0.2]])  
print("Форма массива X\_new: {}".format(X\_new.shape))  
prediction = knn.predict(X\_new)  
print("Прогноз: {}".format(prediction))  
  
  
y\_pred = knn.predict(X\_validation)  
print("Прогнозы для тестовго набора:\n {}".format(y\_pred))  
  
print("Правильность на тестовом наборе: {:.2f}".format(np.mean(y\_pred == Y\_validation)))  
  
print("Правильность на тестовом наборе: {:.2f}".format(knn.score(X\_validation, Y\_validation)))

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

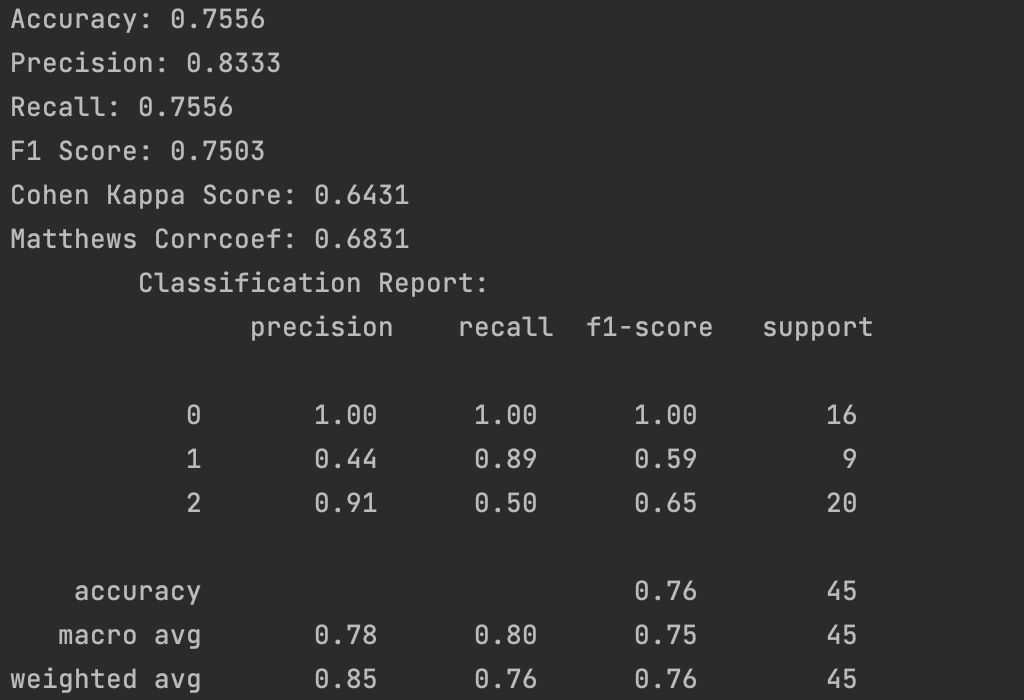
**завдання 2.1**



**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  
  
iris = load\_iris()  
X, y = iris.data, iris.target  
Xtrain, Xtest, ytrain, ytest = train\_test\_split(X, y, test\_size = 0.3,  
random\_state = 0)  
clf = RidgeClassifier(tol = 1e-2, solver = "sag")  
clf.fit(Xtrain,ytrain)  
ypred = clf.predict(Xtest)  
from sklearn import metrics  
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))  
print('\t\tClassification Report:\n',  
metrics.classification\_report(ypred,ytest))  
from sklearn.metrics import confusion\_matrix  
from io import BytesIO #neded for plot  
import seaborn as sns; sns.set()  
import matplotlib.pyplot as plt  
mat = confusion\_matrix(ytest, 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")



Висновок: Отже я, використовуючи спеціалізовані бібліотеки та мову програмування Python дослідив різні методи класифікації даних та навчився їх порівнювати