**ЛАБОРАТОРНА РОБОТА №2 ПОРІВНЯННЯ МЕТОДІВ КЛАСИФІКАЦІЇ ДАНИХ**

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

Хід роботи:

Завдання 2.1.

import numpy as np  
from sklearn import preprocessing  
from sklearn.svm import LinearSVC  
from sklearn.multiclass import OneVsOneClassifier  
from sklearn.model\_selection import train\_test\_split  
from sklearn.model\_selection import 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)  
scaller = preprocessing.MinMaxScaler(feature\_range=(0, 1))  
X = scaller.fit\_transform(X)  
classifier = OneVsOneClassifier(LinearSVC(random\_state=0))  
classifier.fit(X=X, y=Y)  
X\_train, X\_test, y\_train, y\_test \  
= train\_test\_split(X, Y, test\_size=0.2, random\_state=5)  
scaller = preprocessing.MinMaxScaler(feature\_range=(0, 1))  
X\_train = scaller.fit\_transform(X\_train)  
classifier.fit(X=X\_train, y=y\_train)  
y\_test\_pred = classifier.predict(X\_test)  
f1 = cross\_val\_score(classifier, X, Y, scoring="f1\_weighted", cv=3)  
accuracy\_values = cross\_val\_score(classifier, X, Y, scoring='accuracy', cv=3)  
print("Accuracy: " + str(round(100 \* accuracy\_values.mean(), 2)) + "%")  
precision\_values = cross\_val\_score(classifier, X, Y, scoring='precision\_weighted', cv=3)  
print("Precision: " + str(round(100 \* precision\_values.mean(), 2)) + "%")  
recall\_values = cross\_val\_score(classifier, X, Y, scoring='recall\_weighted', cv=3)  
print("Recall: " + str(round(100 \* recall\_values.mean(), 2)) + "%")  
f1\_values = cross\_val\_score(classifier, X, Y, scoring='f1\_weighted', cv=3)  
print("F1: " + str(round(100 \* f1\_values.mean(), 2)) + "%")  
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 = np.array([-1] \* len(input\_data))  
count = 0  
for i, item in enumerate(input\_data):  
 if item.isdigit():  
 input\_data\_encoded[i] = item  
 else:  
 input\_data\_encoded[i] = int(label\_encoder[count].transform([item]))  
 count += 1  
input\_data\_encoded = input\_data\_encoded.astype(int)  
input\_data\_encoded = [input\_data\_encoded]  
predicate\_class = classifier.predict(input\_data\_encoded)  
print(label\_encoder[-1].inverse\_transform(predicate\_class)[0])

Изображение выглядит как текст

Автоматически созданное описание

Рис. 1 - Результат виконання

Завдання 2.2.

Изображение выглядит как текст

Автоматически созданное описание

Рис. 2 - Поліноміальне ядро

Изображение выглядит как текст

Автоматически созданное описание

Рис. 3 - Гаусове ядро

Изображение выглядит как текст

Автоматически созданное описание

Рис. 4 - Сигмоїдальне ядро

Як бачимо, в даній ситуації краще за всього справляється RBF, має добру точність та швидкість. Сигмоїдне ядро не справилось так добре, відстає по швидкості.

Завдання 2.3.

import numpy as np  
from sklearn import preprocessing  
from sklearn.svm import SVC  
from sklearn.multiclass import OneVsOneClassifier  
from sklearn.model\_selection import train\_test\_split  
from sklearn.model\_selection import 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)  
scaller = preprocessing.MinMaxScaler(feature\_range=(0, 1))  
X = scaller.fit\_transform(X)  
classifier = SVC(kernel='sigmoid')  
classifier.fit(X=X, y=Y)  
X\_train, X\_test, y\_train, y\_test \  
= train\_test\_split(X, Y, test\_size=0.2, random\_state=5)  
scaller = preprocessing.MinMaxScaler(feature\_range=(0, 1))  
X\_train = scaller.fit\_transform(X\_train)  
classifier.fit(X=X\_train, y=y\_train)  
y\_test\_pred = classifier.predict(X\_test)  
f1 = cross\_val\_score(classifier, X, Y, scoring="f1\_weighted", cv=3)  
accuracy\_values = cross\_val\_score(classifier, X, Y, scoring='accuracy', cv=3)  
print("Accuracy: " + str(round(100 \* accuracy\_values.mean(), 2)) + "%")  
precision\_values = cross\_val\_score(classifier, X, Y, scoring='precision\_weighted', cv=3)  
print("Precision: " + str(round(100 \* precision\_values.mean(), 2)) + "%")  
recall\_values = cross\_val\_score(classifier, X, Y, scoring='recall\_weighted', cv=3)  
print("Recall: " + str(round(100 \* recall\_values.mean(), 2)) + "%")  
f1\_values = cross\_val\_score(classifier, X, Y, scoring='f1\_weighted', cv=3)  
print("F1: " + str(round(100 \* f1\_values.mean(), 2)) + "%")  
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 = np.array([-1] \* len(input\_data))  
count = 0  
for i, item in enumerate(input\_data):  
 if item.isdigit():  
 input\_data\_encoded[i] = item  
 else:  
 input\_data\_encoded[i] = int(label\_encoder[count].transform([item]))  
 count += 1  
input\_data\_encoded = input\_data\_encoded.astype(int)  
input\_data\_encoded = [input\_data\_encoded]  
predicate\_class = classifier.predict(input\_data\_encoded)  
print(label\_encoder[-1].inverse\_transform(predicate\_class)[0])

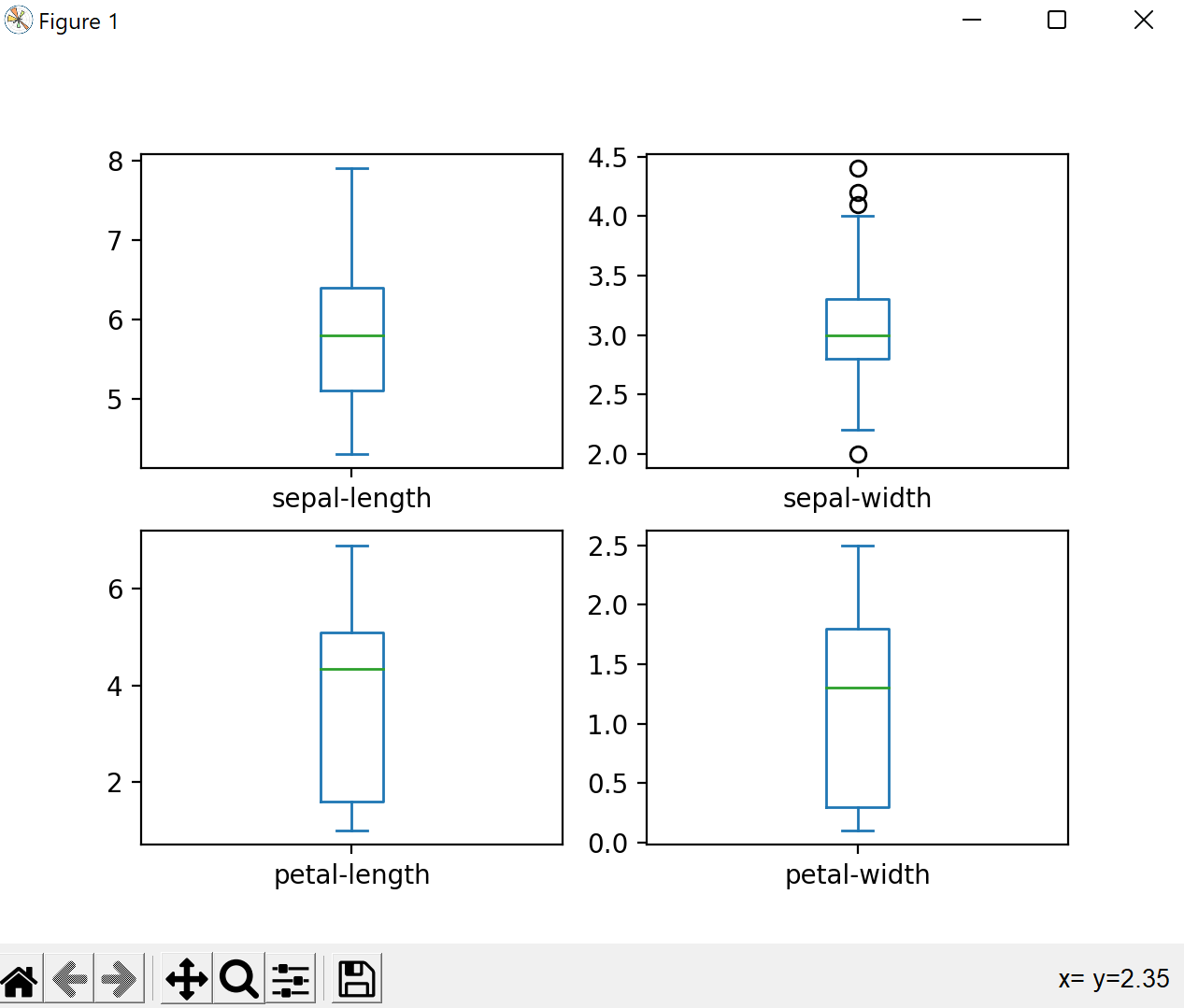


Рис. 5 - Діаграма розмаху

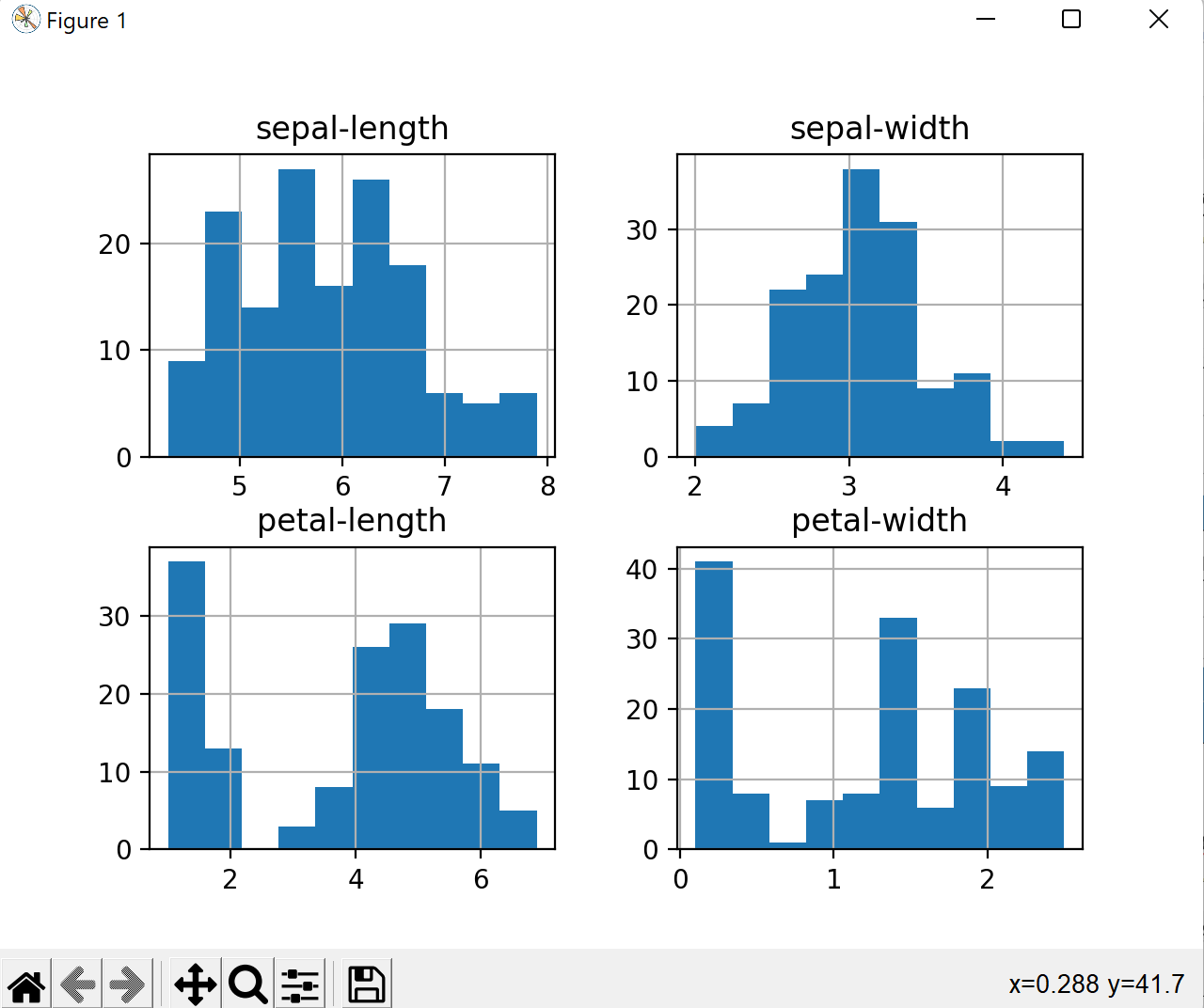


Рис. 6 - Діаграми розподілу атрибутів

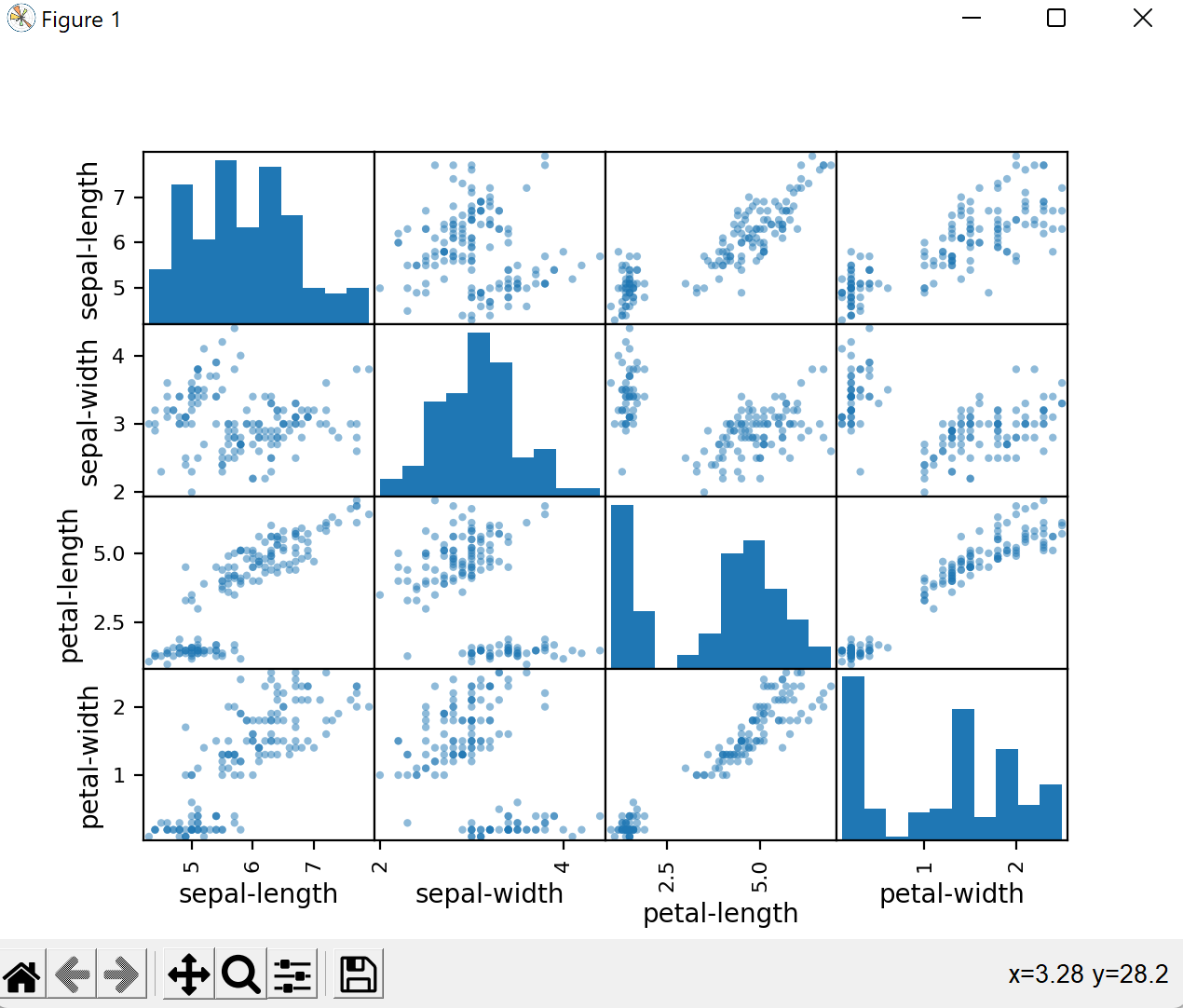


Рис. 7 - Діаграми розсіювання

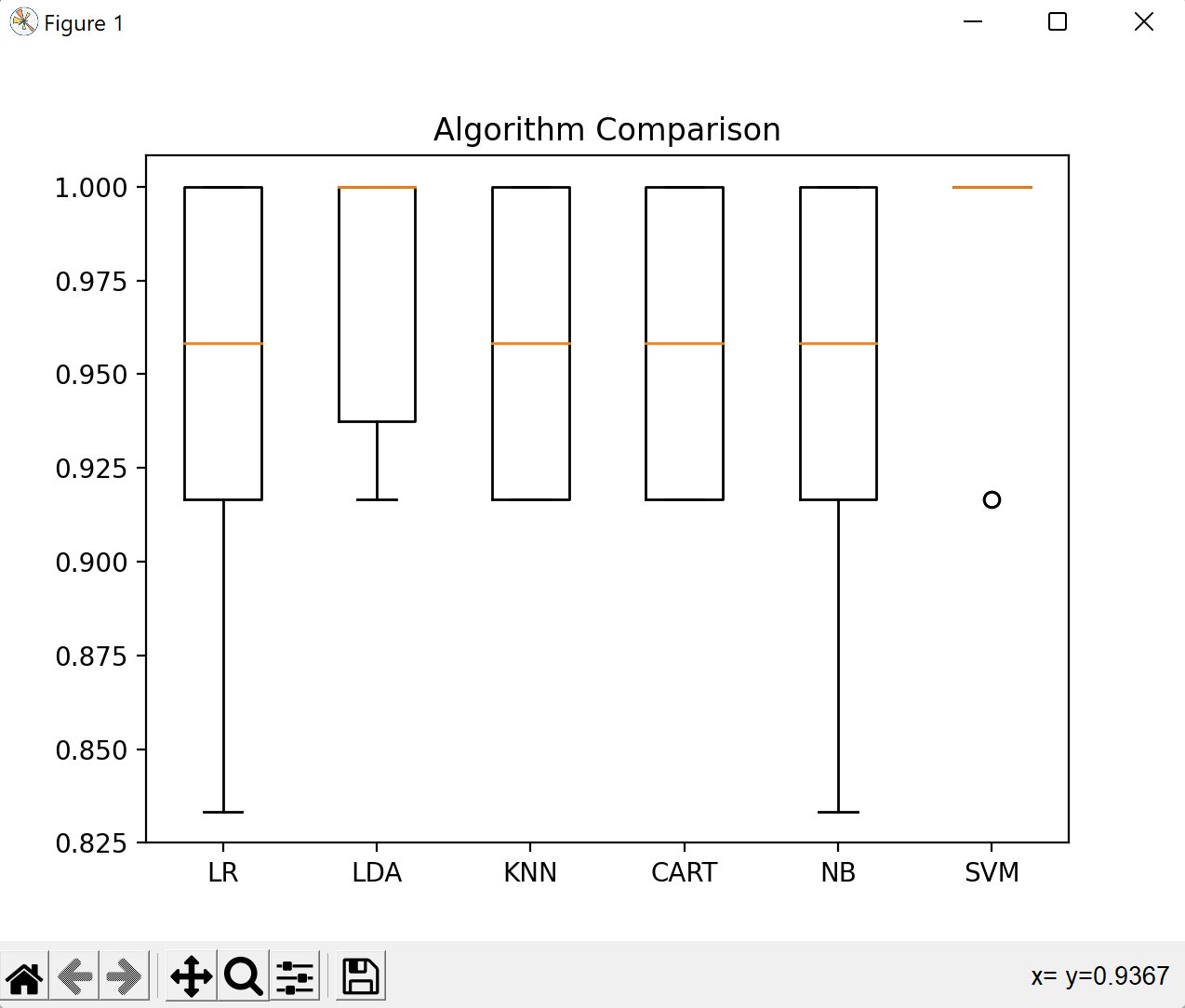


Рис. 8 - Порівняння алгоритмів

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Рис. 9 - Результат виконання

Як бачимо, краще за всіх показала себе модель лінійного дискримінантного аналізу. Квітка з кроку 8 належить до класу Iris-setosa.

Завдання 2.4.

import numpy as np  
from sklearn import preprocessing  
from sklearn.svm import SVC  
from sklearn.model\_selection import train\_test\_split  
from sklearn.model\_selection import cross\_val\_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  
  
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)  
scaller = preprocessing.MinMaxScaler(feature\_range=(0, 1))  
X = scaller.fit\_transform(X)  
  
#classifier = LogisticRegression(solver='liblinear', multi\_class='ovr')  
  
#classifier = LogisticRegression(solver='liblinear', multi\_class='ovr')  
#classifier = LinearDiscriminantAnalysis()  
#classifier = KNeighborsClassifier()  
#classifier = DecisionTreeClassifier()  
#classifier = GaussianNB()  
classifier = SVC(gamma='auto')  
  
classifier.fit(X=X, y=Y)  
X\_train, X\_test, y\_train, y\_test \  
 = train\_test\_split(X, Y, test\_size=0.2, random\_state=5)  
scaller = preprocessing.MinMaxScaler(feature\_range=(0, 1))  
X\_train = scaller.fit\_transform(X\_train)  
classifier.fit(X=X\_train, y=y\_train)  
y\_test\_pred = classifier.predict(X\_test)  
f1 = cross\_val\_score(classifier, X, Y, scoring="f1\_weighted", cv=3)  
accuracy\_values = cross\_val\_score(classifier, X, Y, scoring='accuracy', cv=3)  
print("Accuracy: " + str(round(100 \* accuracy\_values.mean(), 2)) + "%")  
precision\_values = cross\_val\_score(  
 classifier, X, Y, scoring='precision\_weighted', cv=3)  
print("Precision: " + str(round(100 \* precision\_values.mean(), 2)) + "%")  
recall\_values = cross\_val\_score(  
 classifier, X, Y, scoring='recall\_weighted', cv=3)  
print("Recall: " + str(round(100 \* recall\_values.mean(), 2)) + "%")  
f1\_values = cross\_val\_score(classifier, X, Y, scoring='f1\_weighted', cv=3)  
print("F1: " + str(round(100 \* f1\_values.mean(), 2)) + "%")  
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 = np.array([-1] \* len(input\_data))  
count = 0  
for i, item in enumerate(input\_data):  
 if item.isdigit():  
 input\_data\_encoded[i] = item  
 else:  
 input\_data\_encoded[i] = int(label\_encoder[count].transform([item]))  
 count += 1  
input\_data\_encoded = input\_data\_encoded.astype(int)  
input\_data\_encoded = [input\_data\_encoded]  
predicate\_class = classifier.predict(input\_data\_encoded)  
print(label\_encoder[-1].inverse\_transform(predicate\_class)[0])

Изображение выглядит как текст

Автоматически созданное описание

Рис.10 - Точність класифікатора LRИзображение выглядит как текст

Автоматически созданное описание

Рис. 11 - Точність класифікатора LDA

Изображение выглядит как текст

Автоматически созданное описание

Рис. 12 - Точність класифікатора KNNИзображение выглядит как текст

Автоматически созданное описание

Рис. 13 - Точність класифікатора CARTИзображение выглядит как текст

Автоматически созданное описание

Рис. 14 - Точність класифікатора NBИзображение выглядит как текст

Автоматически созданное описание

Рис. 15 - Точність класифікатора SVM

Завдання 2.5.

import numpy as np  
import seaborn as sns  
from sklearn.datasets import load\_iris  
from sklearn.linear\_model import RidgeClassifier  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import confusion\_matrix  
from io import BytesIO  
import matplotlib.pyplot as plt  
from sklearn import metrics  
sns.set()  
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)  
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))  
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")

Изображение выглядит как текст

Автоматически созданное описание

Рис. 16 - Результат виконання

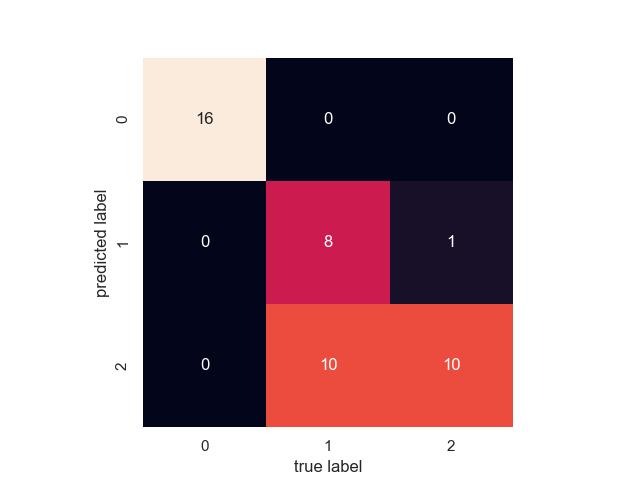


Рис. 17 - Матриця невідповідності

Матриця невідповідності – це таблиця особливого компонування, що дає можливість унаочнювати продуктивність алгоритму. Кожен з рядків цієї матриці представляє зразки прогнозованого класу, тоді як кожен зі стовпців представляє зразки справжнього класу (або навпаки).

Коефіцієнт каппа Коена статистика, яка використовується для вимірювання надійності між експертами для якісних пунктів.

Кореляції Метьюза – використовується в машинному навчанні, як міра якості бінарних мультикласних класифікацій.

Висновки: Після виконання лаби навчився використовувати спеціалізовані бібліотеки та мову програмування Python дослідив різні методи класифікації даних та навчився їх порівнювати.

Git: https://github.com/pvoitko/II