**Лабораторна робота № 1**

**ПОПЕРЕДНЯ ОБРОБКА ТА КОНТРОЛЬОВАНА КЛАСИФІКАЦІЯ ДАНИХ**

Мета роботи: використовуючи спеціалізовані бібліотеки та мову програмування Python дослідити попередню обробку та класифікацію даних

Хід роботи:

Для написання коду використовується середовище програмування PyCharm.

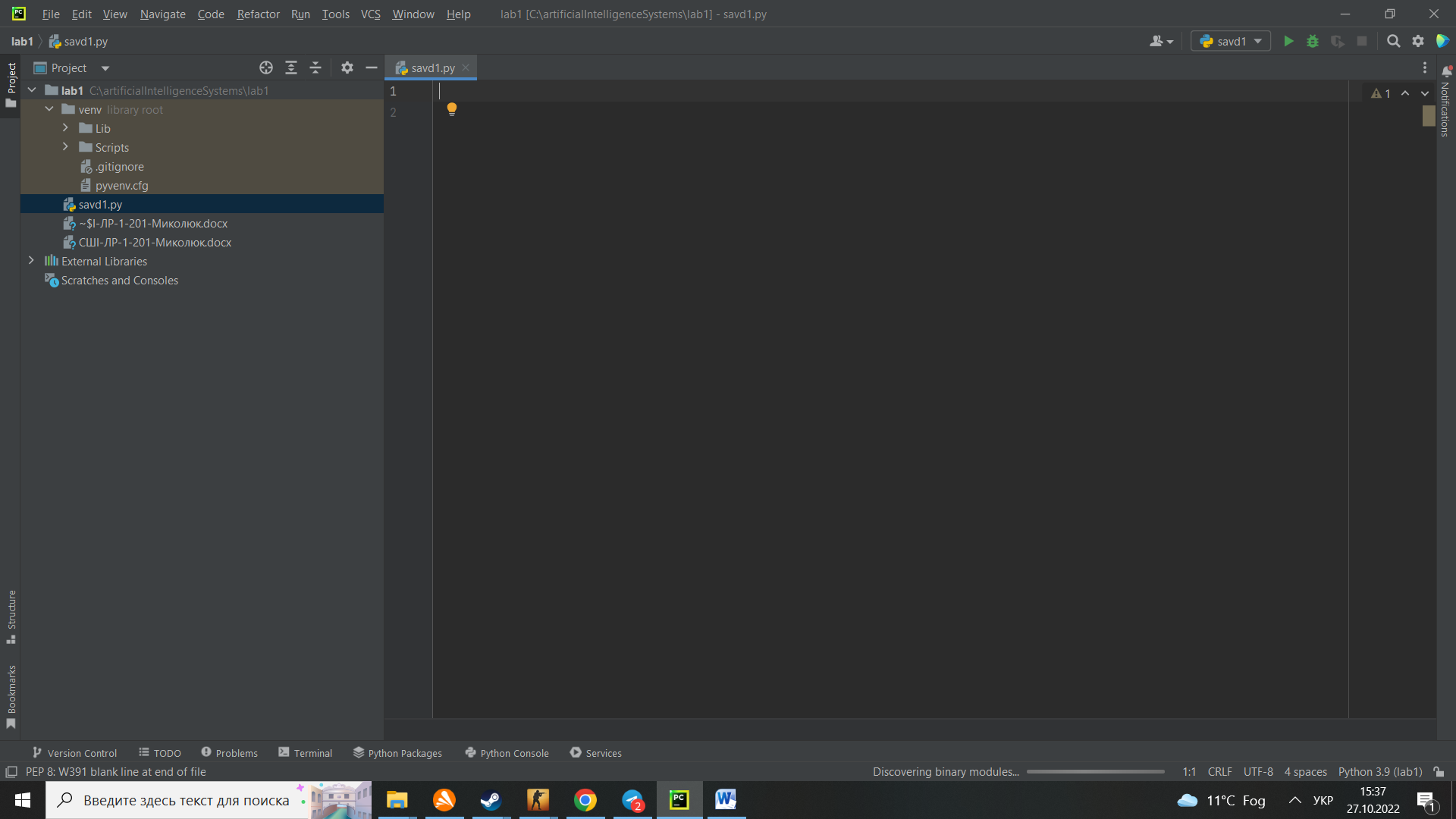


Рис 1. Середовище програмування PyCharm

**Завдання 1.**

import numpy as np  
from sklearn import preprocessing  
Input\_labels = ['red', 'blасk', 'red', 'green', 'Ьlack', 'yellow', 'white']  
encoder = preprocessing.LabelEncoder()  
encoder.fit(Input\_labels)  
print("\nLabel mapping:")  
for i, item in enumerate(encoder.classes\_):  
 print(item, '-->', i)  
test\_labels = ['green', 'red', 'blасk']  
encoded\_values = encoder.transform(test\_labels)  
print("\nLabels =", test\_labels)  
print("Encoded values =", list(encoded\_values))  
encoded\_values = [3, 0, 4, 1]  
decoded\_list = encoder.inverse\_transform(encoded\_values)  
print("\nEncoded values =", encoded\_values)  
print("Decoded labels =", list(decoded\_list))

Результат виконання програми зображено на рисунку 2.

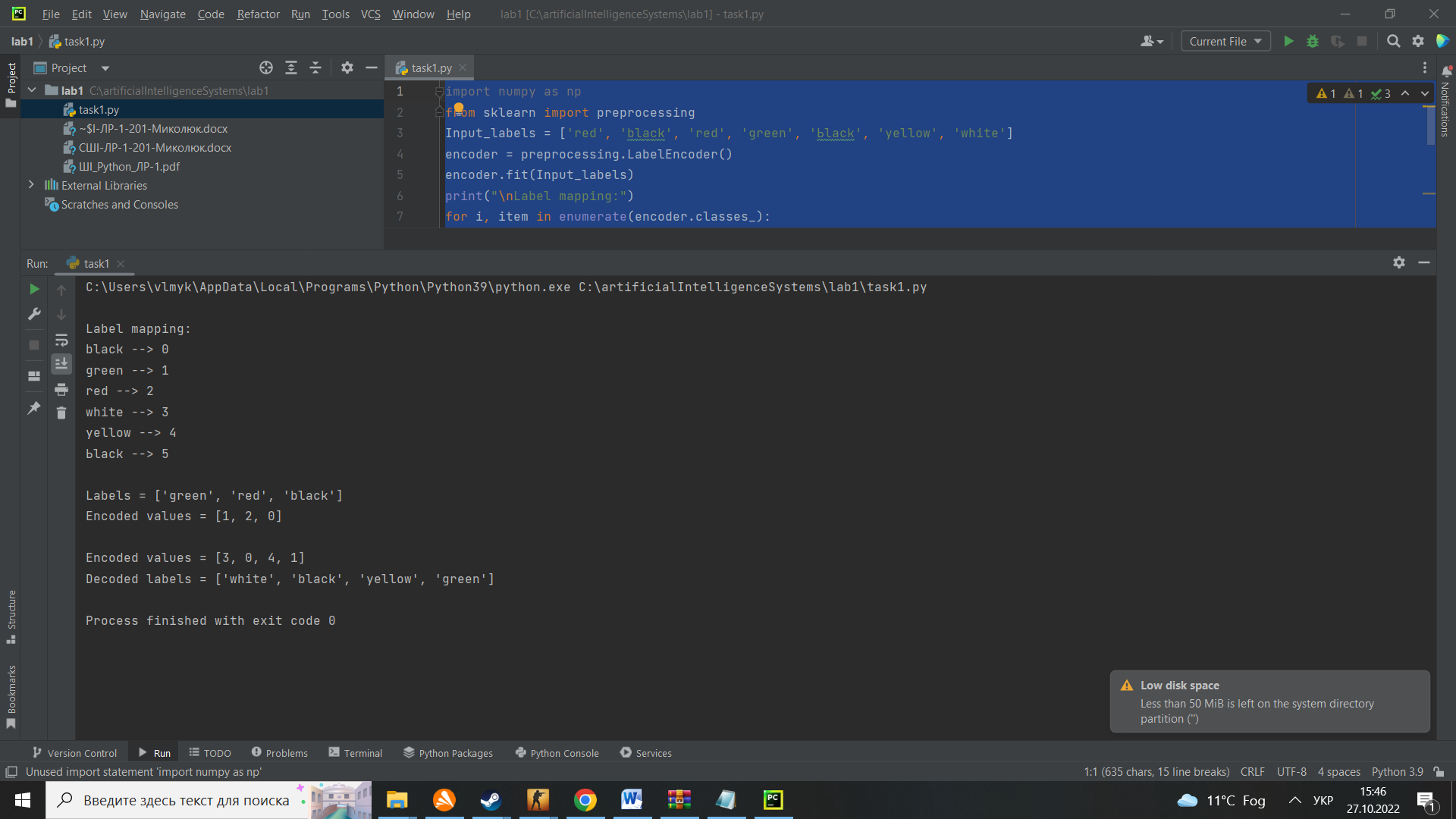
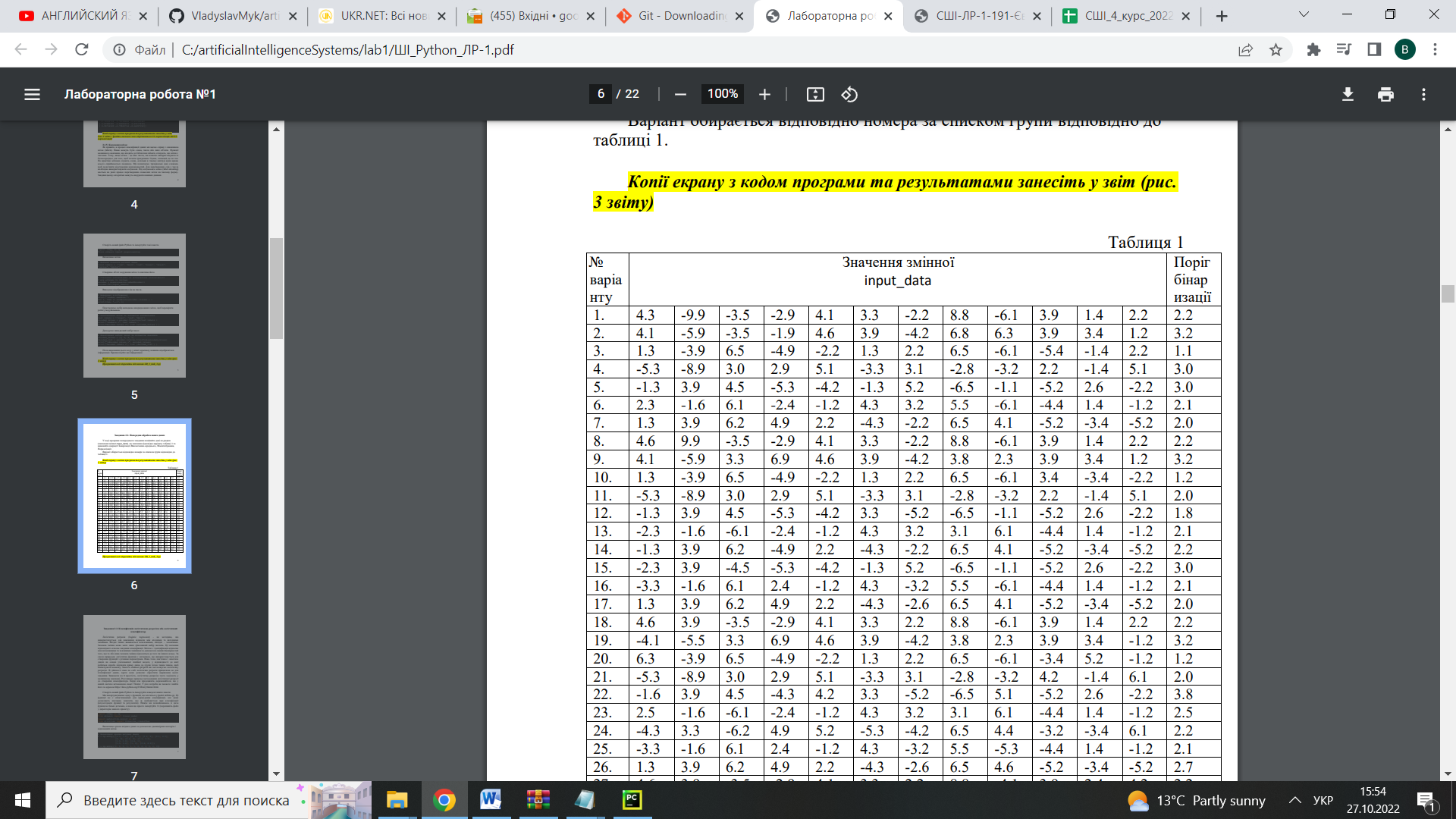


Рис 2. Виконання програми

**Завдання 2.**

Згідно варіанту 10.



import numpy as np  
from sklearn import preprocessing  
input\_data = np.array([[1.3, -3.9, 6.5],  
 [-4.9, -2.2, 1.3],  
 [2.2, 6.5, -6.1],  
 [3.4, -3.4, -2.2]])  
data\_binarized = preprocessing.Binarizer(threshold=1.2).transform(input\_data)  
print("\n Binarized data:\n", data\_binarized)  
print("\nBEFORE: ")  
print("Mean =", input\_data.mean(axis=0))  
print("Std deviation =", input\_data.std(axis=0))  
data\_scaled = preprocessing.scale(input\_data)  
print("\nAFTER: ")  
print("Mean =", data\_scaled.mean(axis=0))  
print("Std deviation =", data\_scaled.std(axis=0))  
data\_scaler\_minmax = preprocessing.MinMaxScaler(feature\_range=(0, 1))  
data\_scaled\_minmax = data\_scaler\_minmax.fit\_transform(input\_data)  
print("\nMin max scaled data:\n", data\_scaled\_minmax)  
data\_normalized\_l1 = preprocessing.normalize(input\_data, norm='l1')  
data\_normalized\_l2 = preprocessing.normalize(input\_data, norm='l2')  
print("\nl1 normalized data:\n", data\_normalized\_l1)  
print("\nl2 normalized data:\n", data\_normalized\_l2)

Результат виконання програми зображено на рисунку 3.

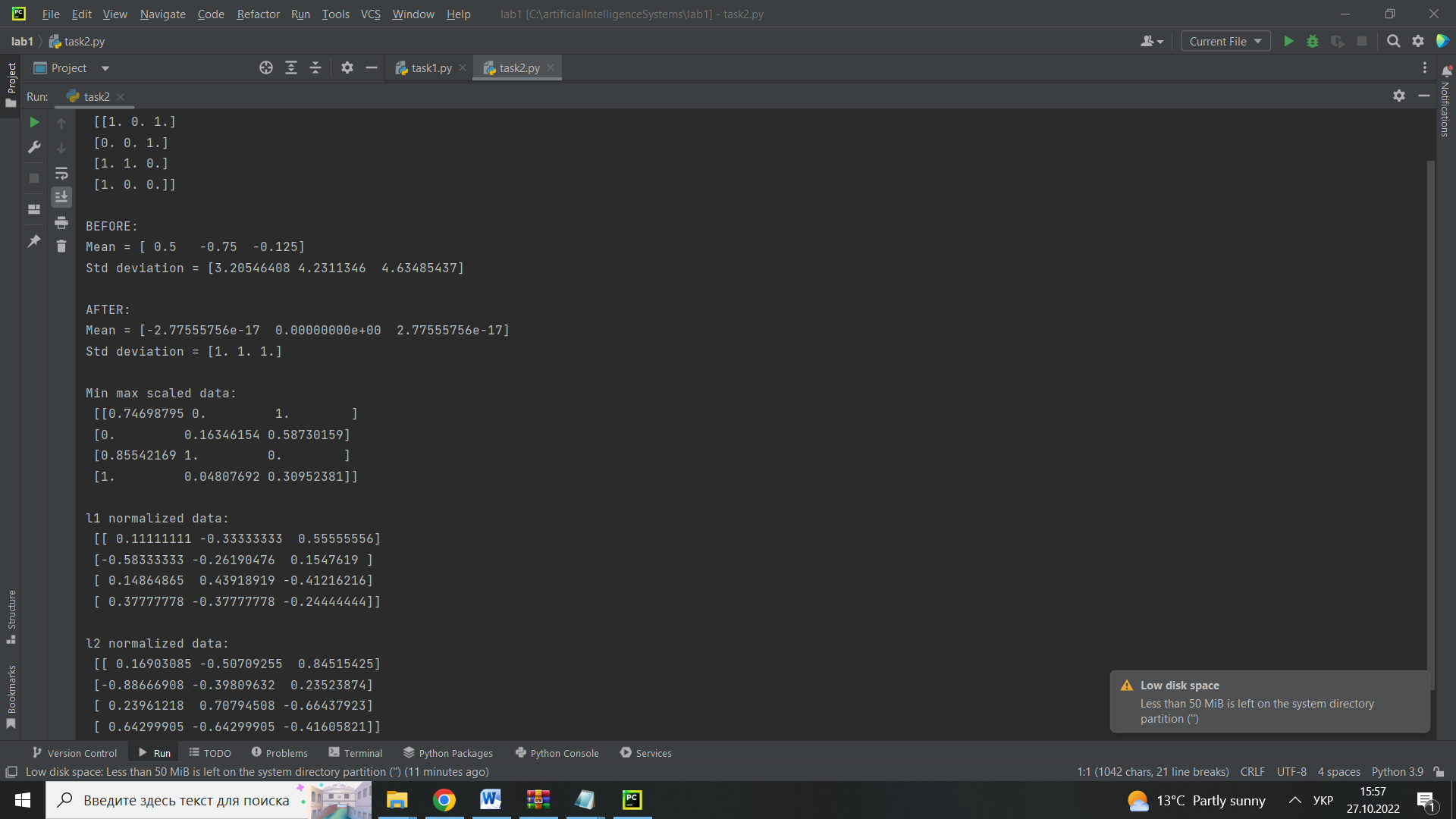


Рис 3. Виконання програми

**Завдання 3.** **Класифікація логістичною регресією або логістичний класифікатор**

Виникли проблеми з package utilities. Було завантажено цей package з

<https://github.com/PacktPublishing/Artificial-Intelligence-with-Python/tree/master/Chapter%2002/code>

Код програми:

import numpy as np  
from sklearn import linear\_model  
import matplotlib.pyplot as plt  
from utilities import visualize\_classifier  
X = np.array([[3.1, 7.2], [4, 6.7], [2.9, 8], [5.1, 4.5],  
 [6, 5], [5.6, 5], [3.3, 0.4],  
 [3.9, 0.9], [2.8, 1],  
 [0.5, 3.4], [1, 4], [0.6, 4.9]])  
y = np.array([0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 3])  
classifier = linear\_model.LogisticRegression(solver='liblinear', C=1)  
classifier.fit(X, y)  
visualize\_classifier(classifier, X, y)

Результат роботи програми зображено на рисунку 4.

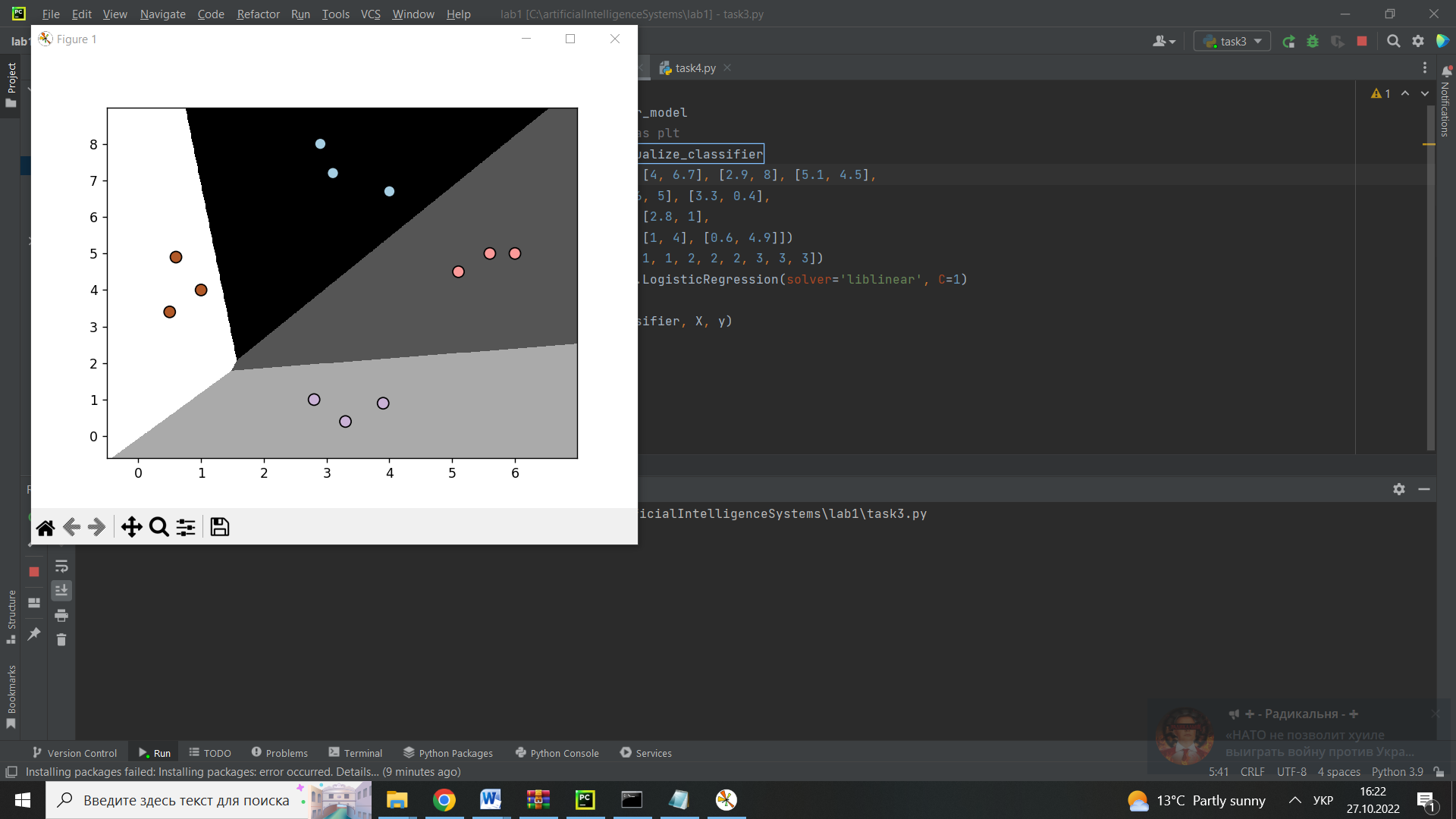


Рис 4. Результат роботи програми

**Завдання 4. Класифікація наївним байєсовським класифікатором**

import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.naive\_bayes import GaussianNB  
from sklearn.model\_selection import train\_test\_split  
  
from utilities import visualize\_classifier  
input\_file = 'data\_multivar\_nb.txt'  
data = np.loadtxt(input\_file, delimiter=',')  
X, y = data[:, :-1], data[:, -1]  
classifier = GaussianNB()  
classifier.fit(X, y)  
y\_pred = classifier.predict(X)  
accuracy = 100.0 \* (y == y\_pred).sum() / X.shape[0]  
print("Accuracy of Naive Bayes classifier =", round(accuracy, 2), "%")  
visualize\_classifier(classifier, X, y)  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(  
 X, y, test\_size=0.2, random\_state=3)  
classifier\_new = GaussianNB()  
classifier\_new.fit(X\_train, y\_train)  
y\_test\_pred = classifier\_new.predict(X\_test)  
accuracy = 100.0 \* (y\_test == y\_test\_pred).sum() / X\_test.shape[0]  
print("Accuracy of the new classifier =", round(accuracy, 2), "%")  
visualize\_classifier(classifier\_new, X\_test, y\_test)  
num\_folds = 3  
accuracy\_values = train\_test\_split.cross\_val\_score(classifier, X, y, scoring='accuracy', cv=num\_folds)  
print("Accuracy: " + str(round(100 \* accuracy\_values.mean(), 2)) + "%")  
precision\_values = train\_test\_split.cross\_val\_score(classifier, X, y,  
 scoring='precision\_weighted', cv=num\_folds)  
print("Precision: " + str(round(100 \* precision\_values.mean(), 2)) + "%")  
recall\_values = train\_test\_split.cross\_val\_score(classifier, X, y, scoring='recall\_weighted',  
 cv=num\_folds)  
print("Recall: " + str(round(100 \* recall\_values.mean(), 2)) + "%")  
f1\_values = train\_test\_split.cross\_val\_score(classifier, X, y, scoring='f1\_weighted',  
 cv=num\_folds)  
print("F1: " + str(round(100 \* f1\_values.mean(), 2)) + "%")

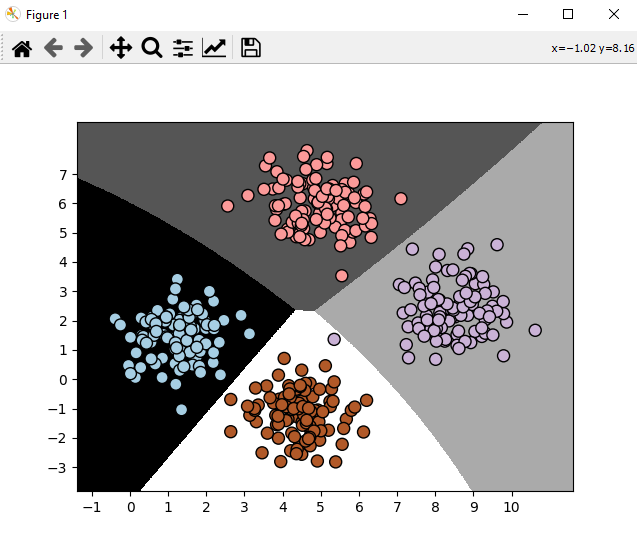


Рис 5. Результат виконання програми

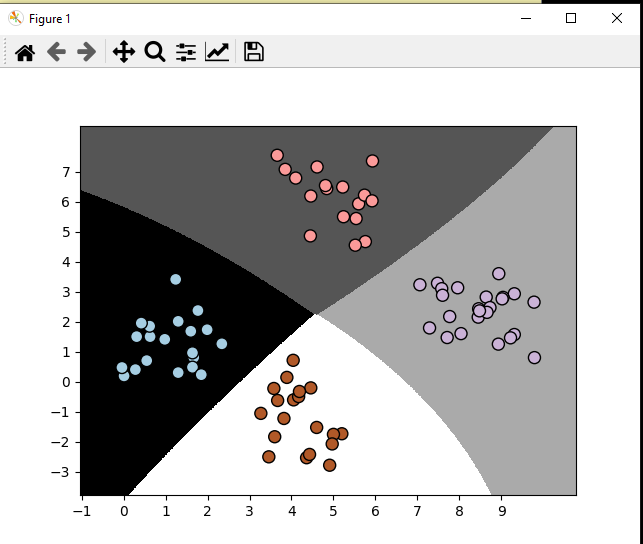


Рис 6. Результат виконання програми

**Завдання 5. Вивчити метрики якості класифікації**

import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.metrics import confusion\_matrix  
from sklearn.metrics import accuracy\_score  
from sklearn.metrics import recall\_score  
from sklearn.metrics import precision\_score  
from sklearn.metrics import f1\_score  
from sklearn.metrics import roc\_curve  
from sklearn.metrics import roc\_auc\_score  
df = pd.read\_csv('data\_metrics.csv')  
df.head()  
thresh = 0.5  
df['predicted\_RF'] = (df.model\_RF >= 0.5).astype('int')  
df['predicted\_LR'] = (df.model\_LR >= 0.5).astype('int')  
df.head()  
print(confusion\_matrix(df.actual\_label.values, df.predicted\_RF.values))  
  
  
def find\_TP(y\_true, y\_pred):  
 # counts the number of true positives (y\_true = 1, y\_pred = 1)  
 return sum((y\_true == 1) & (y\_pred == 1))  
  
  
def find\_FN(y\_true, y\_pred):  
  
 # counts the number of false negatives (y\_true = 1, y\_pred = 0)  
 return sum((y\_true == 1) & (y\_pred == 0))  
  
  
def find\_FP(y\_true, y\_pred):  
  
 # counts the number of false positives (y\_true = 0, y\_pred = 1)  
 return sum((y\_true == 0) & (y\_pred == 1))  
  
  
def find\_TN(y\_true, y\_pred):  
 # counts the number of true negatives (y\_true = 0, y\_pred = 0)  
 return sum((y\_true == 0) & (y\_pred == 0))  
  
  
print('TP:', find\_TP(df.actual\_label.values,  
 df.predicted\_RF.values))  
print('FN:', find\_FN(df.actual\_label.values,  
 df.predicted\_RF.values))  
print('FP:', find\_FP(df.actual\_label.values,  
 df.predicted\_RF.values))  
print('TN:', find\_TN(df.actual\_label.values,  
 df.predicted\_RF.values))  
  
  
def find\_conf\_matrix\_values(y\_true, y\_pred):  
 # calculate TP, FN, FP, TN  
 TP = find\_TP(y\_true, y\_pred)  
 FN = find\_FN(y\_true, y\_pred)  
 FP = find\_FP(y\_true, y\_pred)  
 TN = find\_TN(y\_true, y\_pred)  
 return TP, FN, FP, TN  
  
  
def MyConfusionMatrix(y\_true, y\_pred):  
 TP, FN, FP, TN = find\_conf\_matrix\_values(y\_true, y\_pred)  
 return np.array([[TN, FP], [FN, TP]])  
  
  
MyConfusionMatrix(df.actual\_label.values,  
 df.predicted\_RF.values)  
assert np.array\_equal(MyConfusionMatrix(df.actual\_label.values, df.predicted\_RF.values), confusion\_matrix(  
 df.actual\_label.values, df.predicted\_RF.values)), 'MyConfusionMatrix() is not correct for RF'  
assert np.array\_equal(MyConfusionMatrix(df.actual\_label.values,  
 df.predicted\_LR.values),  
 confusion\_matrix(df.actual\_label.values,  
 df.predicted\_LR.values)), 'MyConfusionMatrix() is not correct for LR'  
print(accuracy\_score(df.actual\_label.values, df.predicted\_RF.values))  
# calculates the fraction of samples  
  
def MyAccuracyScore(y\_true, y\_pred):  
 TP, FN, FP, TN = find\_conf\_matrix\_values(y\_true, y\_pred)  
 return (TP + TN) / (TP + TN + FP + FN)  
  
  
MyAccuracyScore(df.actual\_label.values,  
 df.predicted\_RF.values) == accuracy\_score(df.actual\_label.values,  
 df.predicted\_RF.values),  
'MyAccuracyScore failed on'  
assert MyAccuracyScore(df.actual\_label.values,  
 df.predicted\_LR.values) == accuracy\_score(df.actual\_label.values,  
 df.predicted\_LR.values), 'MyAccuracyScore failed on LR'  
print('Accuracy RF: % .3f ' % (MyAccuracyScore(df.actual\_label.values,  
 df.predicted\_RF.values)))  
print(recall\_score(df.actual\_label.values, df.predicted\_RF.values))  
f1\_score(df.actual\_label.values, df.predicted\_RF.values)  
  
print(recall\_score(df.actual\_label.values, df.predicted\_RF.values))  
  
  
def MyRecallScore(y\_true, y\_pred):  
 # calculates the fraction of positive samples predicted correctly  
 TP, FN, FP, TN = find\_conf\_matrix\_values(y\_true, y\_pred)  
 return TP / (TP + FN)  
  
  
MyRecallScore(df.actual\_label.values, df.predicted\_RF.values) == recall\_score(df.actual\_label.values,  
 df.predicted\_RF.values), 'MyAccuracyScore failed on RF'  
assert MyRecallScore(df.actual\_label.values,  
 df.predicted\_LR.values) == recall\_score(df.actual\_label.values,  
 df.predicted\_LR.values), 'MyAccuracyScore failed on LR'  
  
print('Recall RF: %.3f' % (MyRecallScore(df.actual\_label.values,  
 df.predicted\_RF.values)))  
print('Recall LR: %.3f' % (MyRecallScore(df.actual\_label.values,  
 df.predicted\_LR.values)))  
precision\_score(df.actual\_label.values, df.predicted\_RF.values)  
  
  
def MyPrecisionScore(y\_true, y\_pred):  
 # calculates the fraction of predicted positives samples that are actually positive  
 TP, FN, FP, TN = find\_conf\_matrix\_values(y\_true, y\_pred)  
 return TP / (TP + FP)  
  
  
MyPrecisionScore(df.actual\_label.values, df.predicted\_RF.values) == precision\_score(df.actual\_label.values,  
 df.predicted\_RF.values), 'my\_accuracy\_score failed on RF'  
assert MyPrecisionScore(df.actual\_label.values, df.predicted\_LR.values) == precision\_score(  
 df.actual\_label.values,  
 df.predicted\_LR.values), 'my\_accuracy\_score failed on LR'  
print('Precision RF: %.3f' % (MyPrecisionScore(df.actual\_label.values,  
 df.predicted\_RF.values)))  
print('Precision LR: %.3f' % (MyPrecisionScore(df.actual\_label.values,  
 df.predicted\_LR.values)))  
f1\_score(df.actual\_label.values, df.predicted\_RF.values)  
  
  
def my\_f1\_score(y\_true, y\_pred):  
 # calculates the F1 score  
 recall = MyRecallScore(y\_true, y\_pred)  
 precision = MyPrecisionScore(y\_true, y\_pred)  
 return 2 \* (precision \* recall) / (precision + recall)  
  
  
my\_f1\_score(df.actual\_label.values, df.predicted\_RF.values) == f1\_score(df.actual\_label.values,  
 df.predicted\_RF.values), 'my\_accuracy\_score failed on RF'  
assert my\_f1\_score(df.actual\_label.values, df.predicted\_LR.values) == f1\_score(df.actual\_label.values,  
 df.predicted\_LR.values), 'my\_accuracy\_score failed on LR'  
print('F1 RF: %.3f' % (my\_f1\_score(df.actual\_label.values, df.predicted\_RF.values)))  
print('F1 LR: %.3f' % (my\_f1\_score(df.actual\_label.values, df.predicted\_LR.values)))  
print('scores with threshold = 0.5')  
print('Accuracy RF: % .3f' % (MyAccuracyScore(df.actual\_label.values,  
 df.predicted\_RF.values)))  
print('Recall RF: %.3f' % (MyRecallScore(df.actual\_label.values,  
 df.predicted\_RF.values)))  
print('Precision RF: % .3f' % (MyPrecisionScore(df.actual\_label.values,  
 df.predicted\_RF.values)))  
print('F1 RF: %.3f' % (my\_f1\_score(df.actual\_label.values, df.predicted\_RF.values)))  
  
print('')  
print('scores with threshold = 0.25')  
print('Accuracy RF: % .3f' % (  
 MyAccuracyScore(df.actual\_label.values, (df.model\_RF >=  
 0.25).astype('int').values)))  
print('Recall RF: %.3f' % (MyRecallScore(df.actual\_label.values, (df.model\_RF >=  
 0.25).astype('int').values)))  
print('Precision RF: %.3f' % (  
 MyPrecisionScore(df.actual\_label.values, (df.model\_RF >=  
 0.25).astype('int').values)))  
print('F1 RF: %.3f' % (my\_f1\_score(df.actual\_label.values, (df.model\_RF >=  
 0.25).astype('int').values)))  
fpr\_RF, tpr\_RF, thresholds\_RF = roc\_curve(df.actual\_label.values,  
 df.model\_RF.values)  
fpr\_LR, tpr\_LR, thresholds\_LR = roc\_curve(  
 df.actual\_label.values, df.model\_LR.values)  
plt.plot(fpr\_RF, tpr\_RF, 'r-', label='RF')  
plt.plot(fpr\_LR, tpr\_LR, 'b-', label='LR')  
plt.plot([0, 1], [0, 1], 'k-', label='random')  
plt.plot([0, 0, 1, 1], [0, 1, 1, 1], 'g-', label='perfect')  
plt.legend()  
plt.xlabel('False Positive Rate')  
plt.ylabel('True Positive Rate')  
plt.show()  
auc\_RF = roc\_auc\_score(df.actual\_label.values, df.model\_RF.values)  
auc\_LR = roc\_auc\_score(df.actual\_label.values, df.model\_LR.values)  
print('AUC RF:%.3f' % auc\_RF)  
print('AUC LR:%.3f' % auc\_LR)  
plt.plot(fpr\_RF, tpr\_RF, 'r-', label='RF AUC: %.3f' % auc\_RF)  
plt.plot(fpr\_LR, tpr\_LR, 'b-', label='LR AUC: %.3f' % auc\_LR)  
plt.plot([0, 1], [0, 1], 'k-', label='random')  
plt.plot([0, 0, 1, 1], [0, 1, 1, 1], 'g-', label='perfect')  
plt.legend()  
plt.xlabel('False Positive Rate')  
plt.ylabel('True Positive Rate')  
plt.show()

Результат роботи програми зображено на рисунку 7.

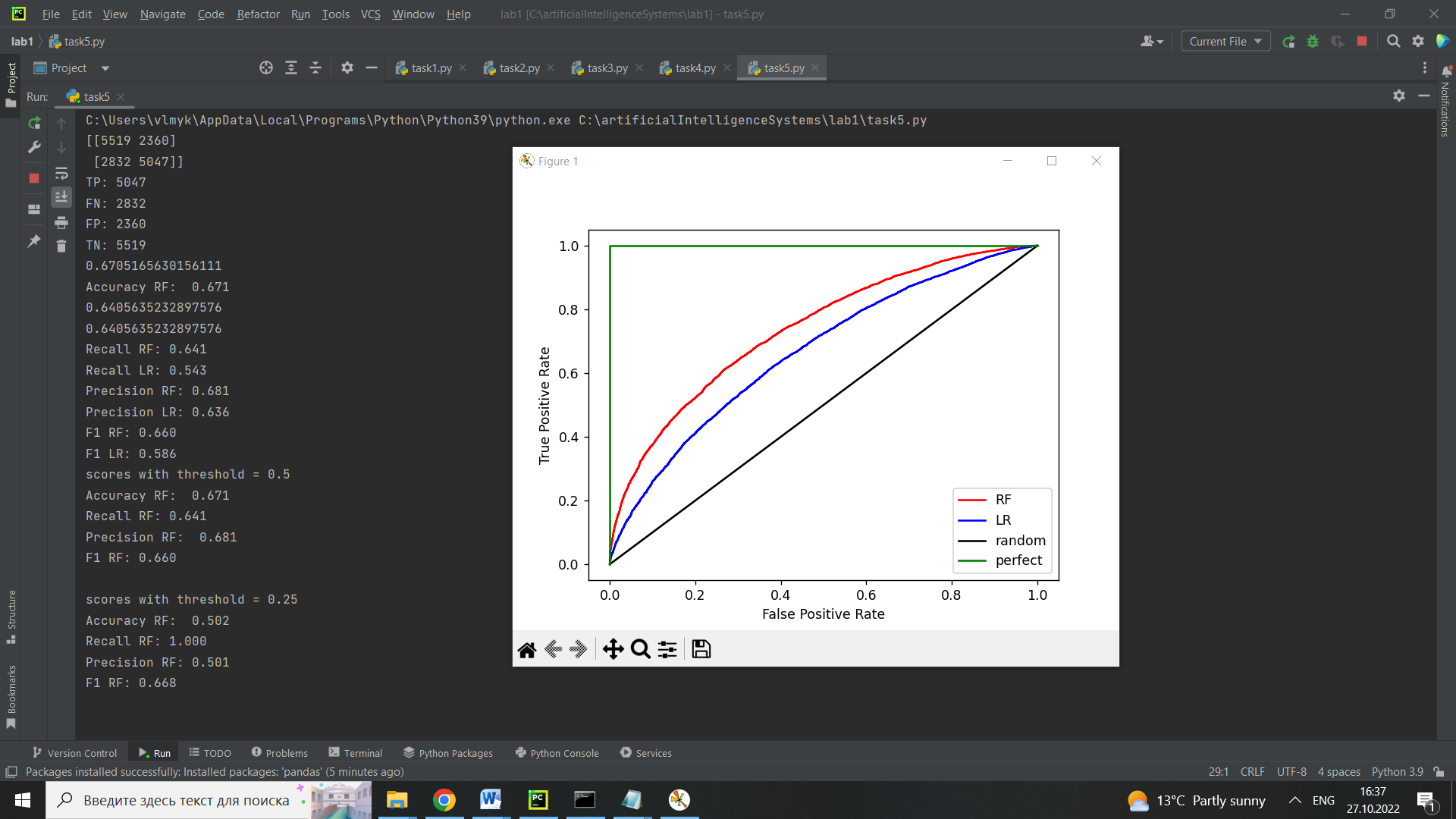


Рис 7. Результат роботи програми

**Завдання 6. Розробіть програму класифікації даних в файлі data\_multivar\_nb.txt за допомогою машини опорних векторів (Support Vector Machine - SVМ). Розрахуйте показники якості класифікації. Порівняйте їх з показниками наївного байєсівського класифікатора. Зробіть висновки яку модель класифікації краще обрати і чому.**

import numpy as np  
from sklearn.model\_selection import train\_test\_split  
from sklearn import svm  
from sklearn import metrics  
from utilities import visualize\_classifier  
input\_file = 'data\_multivar\_nb.txt'  
data = np.loadtxt(input\_file, delimiter=',')  
X, y = data[:, :-1], data[:, -1]  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y.astype(int), test\_size=0.2, random\_state=3)  
cls = svm.SVC(kernel='linear')  
cls.fit(X\_train, y\_train)  
pred = cls.predict(X\_test)  
print("Accuracy:", metrics.accuracy\_score(y\_test, y\_pred=pred))  
print("Precision: ", metrics.precision\_score(y\_test, y\_pred=pred, average='macro'))  
print("Recall", metrics.recall\_score(y\_test, y\_pred=pred, average='macro'))  
print(metrics.classification\_report(y\_test, y\_pred=pred))  
visualize\_classifier(cls, X\_test, y\_test)

Результат роботи програми зображено на рисунку 8.

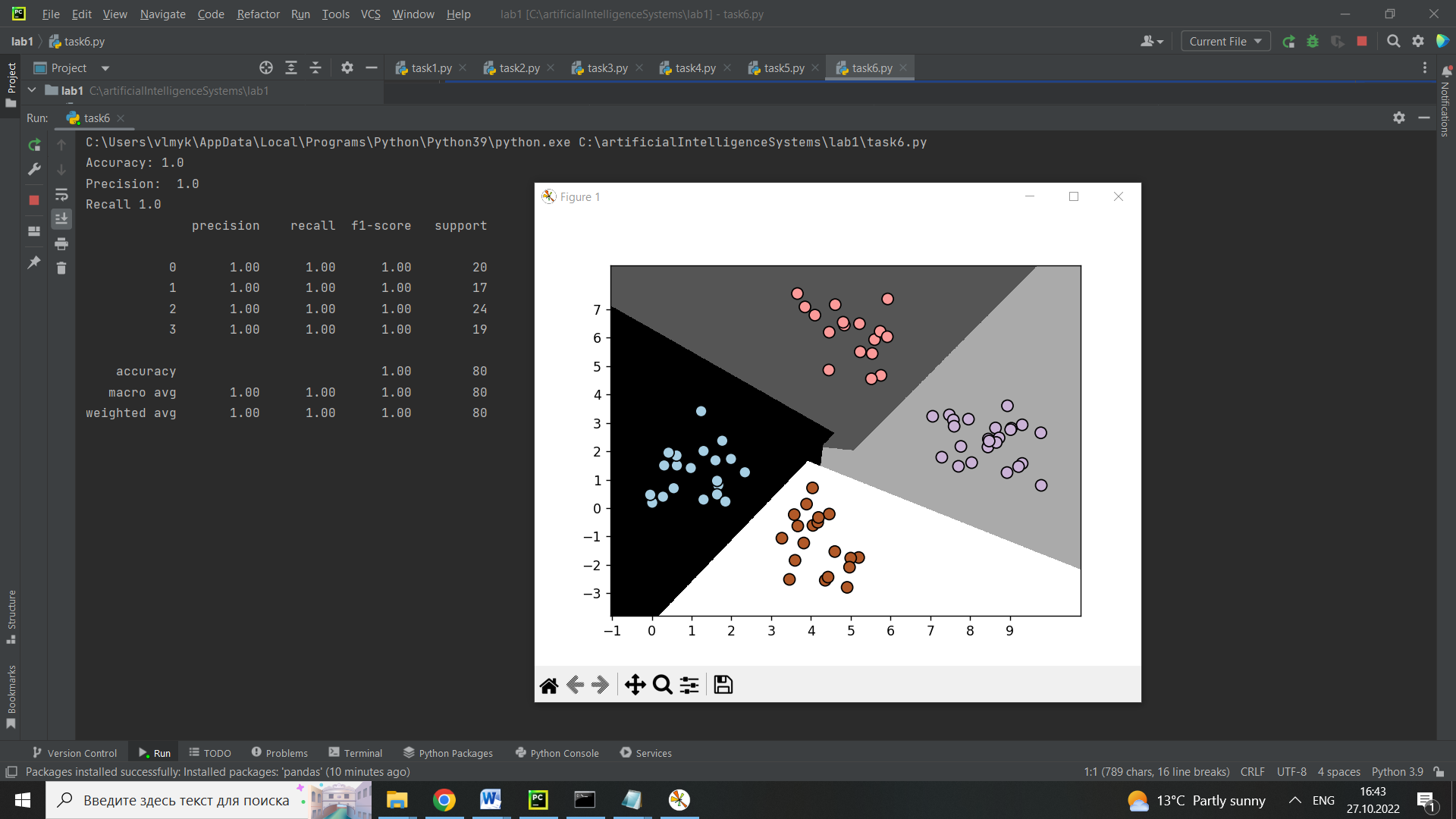


Рис 8. Результат роботи програми

Результат показує, що NBC працює краще, ніж SVM, це вірно тільки для відповідних параметрів, але за інших параметрів можна виявити, що SVM працює краще.

Висновки: в ході виконання лабораторної роботи ми навчилися використовувати спеціалізовані бібліотеки та мову програмування Python для дослідження попередньої обробки та класифікації даних.