ЛАБОРАТОРНА РОБОТА № 1

ПОПЕРЕДНЯ ОБРОБКА ТА КОНТРОЛЬОВАНА КЛАСИФІКАЦІЯ ДАНИХ

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

Завдання 1.

Код програми:

task1.1.py

```python
import numpy as np
from sklearn import preprocessing

input_data = np.array([[5.1, -2.9, 3.3],
                       [-1.2, 7.8, -6.1],
                       [3.9, 0.4, 2.1],
                       [7.3, -9.9, -4.5]])
# Бінаризація даних
data_binarized = preprocessing.Binarizer(threshold=2.1).transform(input_data)
print("\n Binarized data:\n", data_binarized)

# Виведення середнього значення та стандартного відхилення
print("\nBEFORE --> ")
print("Mean = ", input_data.mean(axis=0))
print("Std deviation = ", input_data.std(axis=0))

# Виключення середнього
data_scaled = preprocessing.scale(input_data)
print("\nAFTER --> ")
print("Mean = ", data_scaled.mean(axis=0))
print("Std deviation = ", data_scaled.std(axis=0))

# Масштабування MinMax
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)

# Нормалізація даних
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)
```

Рис 1.1 – лістинг програми

					ЖИТОМИРСЬКА ПОЛІТЕХНІКА.22.121.15.000 – Лр.1		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Прокопчук О.С			Звіт з лабораторної роботи №1	Літ.	Арк. 1 / Аркушів 17
Перевір.		Голенко М.Ю.					
Реценз.						ФІКТ, гр. ІПЗ-21-1(2)	
Н. Контр.							
Зав.каф.							

```
 Binarized data:
 [[1. 0. 1.]
 [0. 1. 0.]
 [1. 0. 0.]
 [1. 0. 0.]]

BEFORE -->
Mean =  [ 3.775 -1.15  -1.3  ]
Std deviation =  [3.12039661 6.36651396 4.0620192 ]

AFTER -->
Mean =  [1.11022302e-16 0.00000000e+00 2.77555756e-17]
Std deviation =  [1. 1. 1.]

Min max scaled data:
 [[0.74117647 0.39548023 1.        ]
 [0.         1.         0.        ]
 [0.6        0.5819209  0.87234043]
 [1.         0.         0.17021277]]

l1 normalized data:
 [[ 0.45132743 -0.25663717  0.2920354 ]
 [-0.0794702   0.51655629 -0.40397351]
 [ 0.609375    0.0625      0.328125  ]
 [ 0.33640553 -0.4562212  -0.20737327]]

l2 normalized data:
 [[ 0.75765788 -0.43082507  0.49024922]
 [-0.12030718  0.78199664 -0.61156148]
 [ 0.87690281  0.08993875  0.47217844]
 [ 0.55734935 -0.75585734 -0.34357152]]
```

Рис 1.2 – результат виконання

L1-нормалізація розподіляє значення рівномірно зосереджуючи увагу на сумарному впливі елементів. А L2 краще враховує відносну велечину елементів у векторі, шо робить її більш корисною для задач, де важлива геометрична структура даних.

Завдання 1.2.

task1.1.py task1.2.py

```python
import numpy as np
from sklearn import preprocessing

# Надання позначок вхідних даних
input_labels = ['red', 'black', 'red', 'green', 'black', 'yellow', 'white']

# Створення кодувальника та встановлення відповідності
# між мітками та числами
encoder = preprocessing.LabelEncoder()
encoder.fit(input_labels)

# Виведення відображення
print("\nLabel mapping:")
for i, item in enumerate(encoder.classes_):
    print(item, '-->', i)

# перетворення міток за допомогою кодувальника
test_labels = ['green', 'red', 'black']
encoded_values = encoder.transform(test_labels)
print("\nLabels =", test_labels )
print("Encoded values =", list (encoded_values))

# Декодування набору чисел за допомогою декодера
encoded_values = [3, 0, 4, 1]
decoded_list = encoder.inverse_transform(encoded_values)
print("\nEncoded values =", encoded_values)
print("Decoded labels =", list (decoded_list))
```

Рис 1.3 – лістинг програми

```
Label mapping:
black --> 0
green --> 1
red --> 2
white --> 3
yellow --> 4

Labels = ['green', 'red', 'black']
Encoded values = [np.int64(1), np.int64(2), np.int64(0)]

Encoded values = [3, 0, 4, 1]
Decoded labels = [np.str_('white'), np.str_('black'), np.str_('yellow'), np.str_('green')]
```

Рис 1.4 – результат виконання

Завдання 2.1.

У коді програми попереднього завдання поміняйте дані по рядках (значення змінної input_data) на значення відповідно варіанту таблиці 1 та виконайте операції: Бінарізації, Виключення середнього, Масштабування, Нормалізації.

```
import numpy as np
from sklearn import preprocessing

from pythonProject.JF import data_binarized

input_data = np.array([[-2.3, 3.9, -4.5],
                       [-5.3, -4.2, -1.3],
                       [5.2, -6.5, -1.1],
                       [-5.2, 2.6, -2.2]])

#Бінаризація даних
data_binarized = preprocessing.Binarizer(threshold=3.0).transform(input_data)
print("\n Binarized data:\n", data_binarized)

# Виведення середнього значення та стандартного відхилення
print("\nBEFORE --> ")
print("Mean = ", input_data.mean(axis=0))
print("Std deviation = ", input_data.std(axis=0))

# Виключення середнього
data_scaled = preprocessing.scale(input_data)
print("\nAFTER --> ")
print("Mean = ", data_scaled.mean(axis=0))
print("Std deviation = ", data_scaled.std(axis=0))

# Масштабування MinMax
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)

# Нормалізація даних
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)
```

Рис 2.1 – лістинг програми

15.	-2.3	3.9	-4.5	-5.3	-4.2	-1.3	5.2	-6.5	-1.1	-5.2	2.6	-2.2	3.0

```
 Binarized data:
 [[1. 0. 1.]
 [0. 1. 0.]
 [1. 0. 0.]
 [1. 0. 0.]]

BEFORE -->
Mean =  [ 3.775 -1.15  -1.3  ]
Std deviation =  [3.12039661 6.36651396 4.0620192 ]

AFTER -->
Mean =  [1.11022302e-16 0.00000000e+00 2.77555756e-17]
Std deviation =  [1. 1. 1.]

Min max scaled data:
 [[0.74117647 0.39548023 1.        ]
 [0.         1.         0.        ]
 [0.6        0.5819209  0.87234043]
 [1.         0.         0.17021277]]

l1 normalized data:
 [[ 0.45132743 -0.25663717  0.2920354 ]
 [-0.0794702   0.51655629 -0.40397351]
 [ 0.609375    0.0625      0.328125  ]
 [ 0.33640553 -0.4562212  -0.20737327]]

l2 normalized data:
 [[ 0.75765788 -0.43082507  0.49024922]
 [-0.12030718  0.78199664 -0.61156148]
 [ 0.87690281  0.08993875  0.47217844]
 [ 0.55734935 -0.75585734 -0.34357152]]
```

Рис 2.2 – результат виконання

Завдання 2.3. Класифікація логістичною регресією або логістичний класифікатор

```python
import numpy as np
from sklearn import linear_model
import matplotlib.pyplot as plt
from utilities import visualize_classifier

# Визначення зразка вхідних даних
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])

# Створення логістичного класифікатора
classifier = linear_model.LogisticRegression(solver='liblinear', C = 1)

# Тренування класифікатора
classifier.fit(X, y)

visualize_classifier(classifier, X, y)
```

Рис 2.3 – лістинг програми

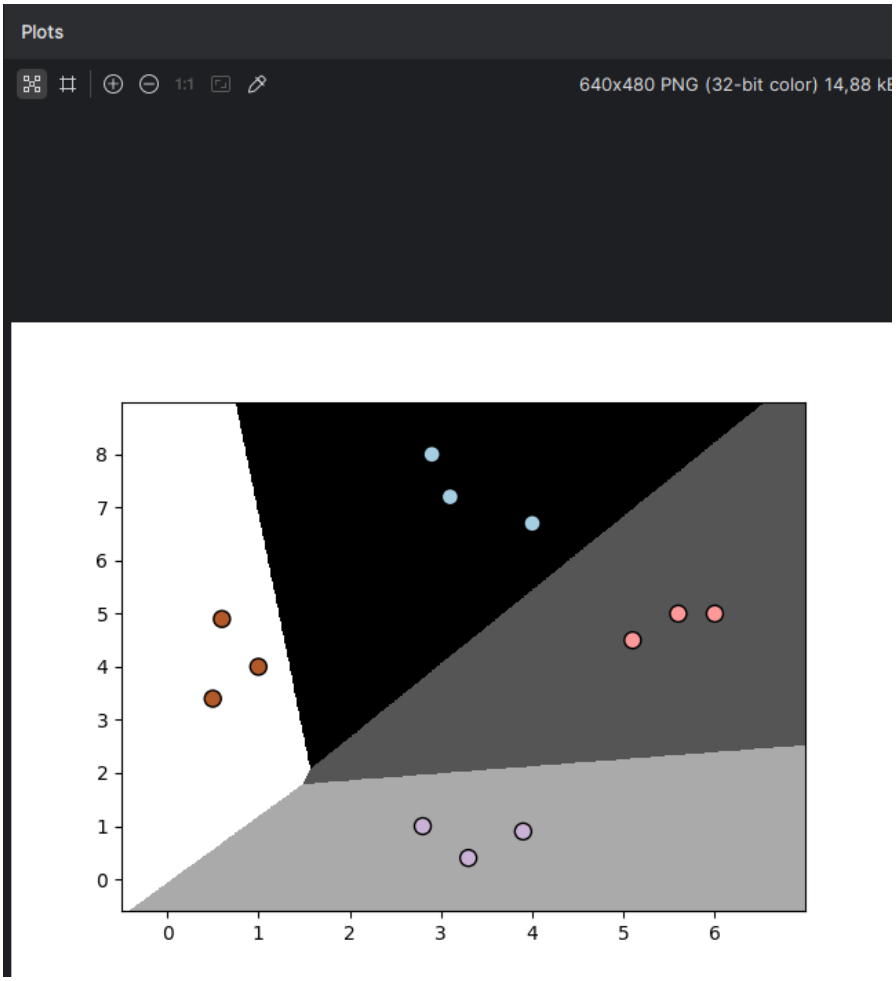

Рис 2.4 – результат виконання

Завдання 2.4. Класифікація наївним байєсовським класифікатором

```python
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

# Вхідний файл, який містить дані
input_file = 'data_multivar_nb.txt'

# Завантаження даних із вхідного файлу
data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1]

# Створення наївного байєсовського класифікатора
classifier = GaussianNB()

# Тренування класифікатора
classifier.fit(X, y)

# Прогнозування значень для тренувальних даних
y_pred = classifier.predict(X)

# Обчислення якості класифікатора
accuracy = 100.0 * (y == y_pred).sum() / X.shape[0]
print("Accuracy of Naive Bayes classifier =", round(accuracy, 2), "%")

# Візуалізація результатів роботи класифікатора
visualize_classifier(classifier, X, y)
```

Рис 2.5 – лістинг програми

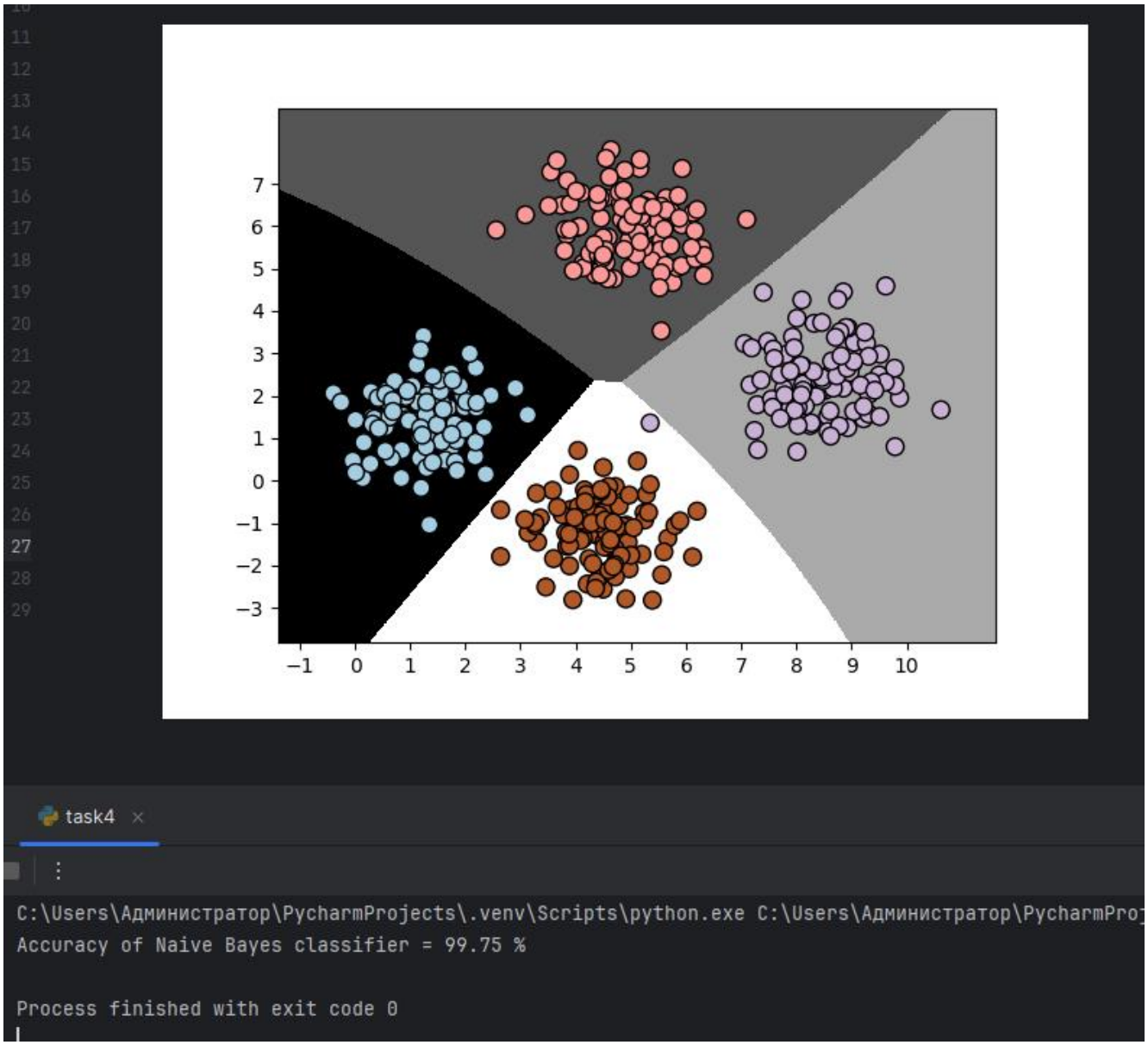

Рис 2.6 – результат виконання

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split, cross_val_score
from utilities import visualize_classifier
# Вхідний файл, який містить дані
input_file = 'data_multivar_nb.txt'
# Завантаження даних із вхідного файлу
data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1]
# Розбивка даних на навчальний та тестовий набори
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)
# Тренування класифікатора
classifier = GaussianNB()
classifier.fit(X, y)
# Прогнозування значень для тренувальних даних
y_pred = classifier.predict(X)
# Обчислення якості класифікатора
accuracy = 100.0 * (y_test == y_test_pred).sum() / X_test.shape[0]
print("Accuracy of the new classifier =", round(accuracy, 2), "%")
# Перевірка на крос-валідації
num_folds = 3
accuracy_values = cross_val_score(classifier, X, y, scoring='accuracy',cv=num_folds)
print("Accuracy: " + str(round(100 * accuracy_values.mean(), 2)) + "%")
precision_values = cross_val_score(classifier, X, y, scoring='precision_weighted', cv=num_folds)
print("Precision: " + str(round(100 * precision_values.mean(), 2)) + "%")
recall_values = cross_val_score(classifier, X, y, scoring='recall_weighted', cv=num_folds)
print("Recall: " + str(round(100 * recall_values.mean(), 2)) + "%")
f1_values = cross_val_score(classifier, X, y, scoring='f1_weighted', cv=num_folds)
print("F1: " + str(round(100 * f1_values.mean(), 2)) + "%")
# Візуалізація роботи класифікатора
visualize_classifier(classifier_new, X_test, y_test)
```

Рис 2.7 – лістинг програми

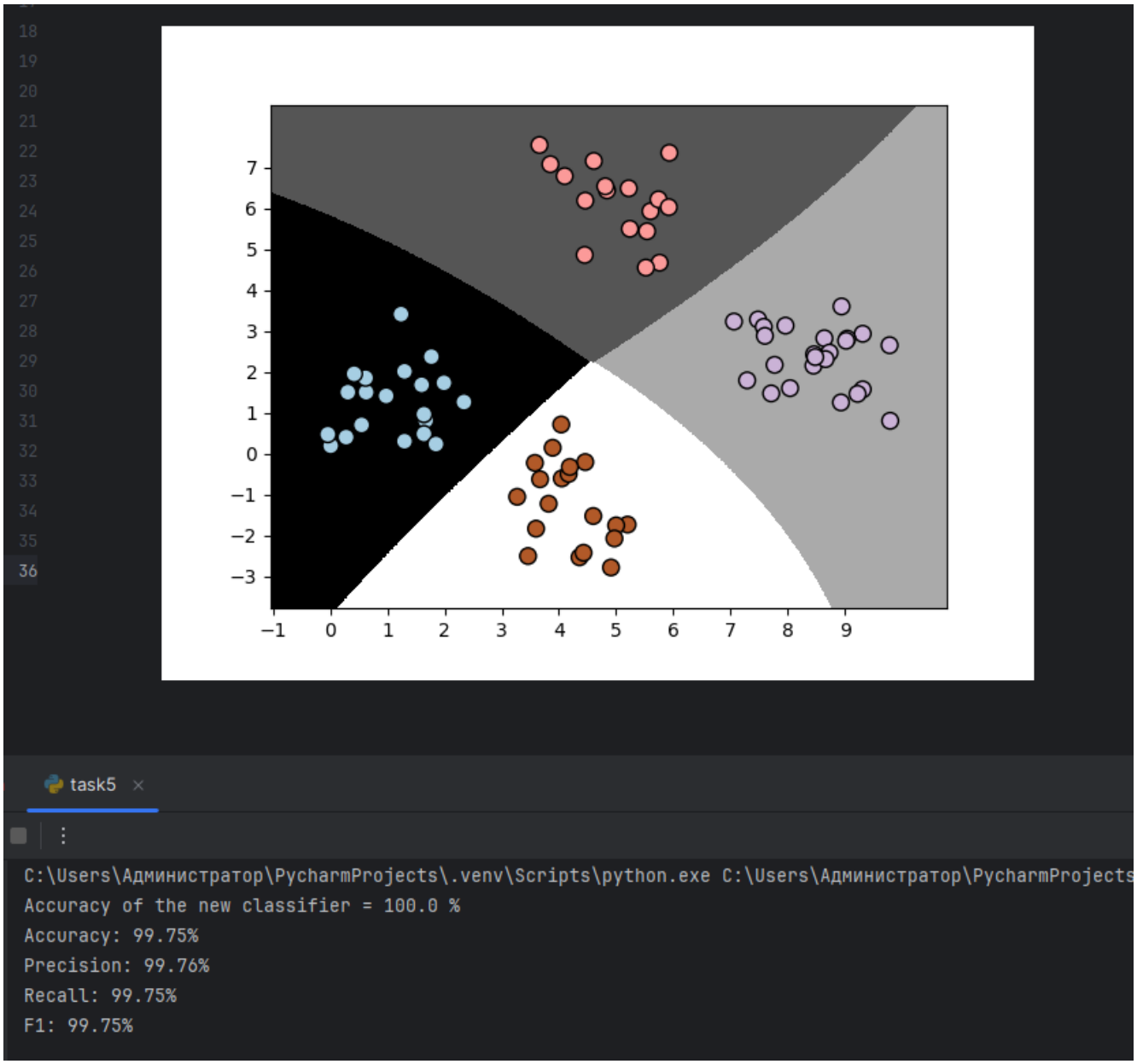

Рис 2.8 – результат виконання

Завдання 2.5. Вивчити метрики якості класифікації

```python
import pandas as pd
from sklearn.metrics import confusion_matrix, accuracy_score
import numpy as np

# Load dataset
df = pd.read_csv('data_metrics.csv')
df.head()

# Set threshold and make predictions
thresh = 0.5
df['predicted_RF'] = (df.model_RF >= thresh).astype('int')
df['predicted_LR'] = (df.model_LR >= thresh).astype('int')
df.head()

# Define helper functions for confusion matrix components
def find_TP(y_true, y_pred):  2 usages
    # 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):  2 usages
    # 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):  2 usages
    # 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):  2 usages
    # counts the number of true negatives (y_true = 0, y_pred = 0)
    return sum((y_true == 0) & (y_pred == 0))
# Testing the helper functions
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))

# Confusion matrix implementation
def find_conf_matrix_values(y_true, y_pred):  4 usages
    # 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 prokopchuk_confusion_matrix(y_true, y_pred):  2 usages
    TP, FN, FP, TN = find_conf_matrix_values(y_true, y_pred)
    return np.array([[TN, FP], [FN, TP]])
```

Рис 2.9 – лістинг програми

```
# Compare custom confusion matrix with sklearn's
assert np.array_equal(prokopchuk_confusion_matrix(df.actual_label.values, df.predicted_RF.values),
 confusion_matrix(df.actual_label.values,df.predicted_RF.values)), 'my_confusion_matrix() is not correct for RF'
assert np.array_equal(prokopchuk_confusion_matrix(df.actual_label.values, df.predicted_LR.values),
 confusion_matrix(df.actual_label.values, df.predicted_LR.values)), 'my_confusion_matrix() is not correct for LR'

# Calculate accuracy
accuracy_rf = accuracy_score(df.actual_label.values,
df.predicted_RF.values)
print(f'Accuracy (Random Forest): {accuracy_rf}')

def prokopchuk_accuracy_score(y_true, y_pred):  5 usages
# calculates the fraction of samples
    TP,FN,FP,TN = find_conf_matrix_values(y_true,y_pred)
    return (TP + TN) / (TP + TN + FP + FN)
assert prokopchuk_accuracy_score(df.actual_label.values,
df.predicted_RF.values) == accuracy_score(df.actual_label.values,
df.predicted_RF.values), 'my_accuracy_score failed on RF'

assert prokopchuk_accuracy_score(df.actual_label.values,
df.predicted_LR.values) == accuracy_score(df.actual_label.values,
df.predicted_LR.values), 'my_accuracy_score failed on LR'

print('Accuracy LR: %.3f'%(prokopchuk_accuracy_score(df.actual_label.values,
df.predicted_LR.values)))

from sklearn.metrics import recall_score
recall_score(df.actual_label.values, df.predicted_RF.values)

def prokopchuk_recall_score(y_true, y_pred):  7 usages
# calculates the fraction of positive samples predicted correctly
    TP,FN,FP,TN = find_conf_matrix_values(y_true,y_pred)
    return TP / (TP + FN) if (TP + FN) != 0 else 0
assert prokopchuk_recall_score(df.actual_label.values,
df.predicted_RF.values) == recall_score(df.actual_label.values,
df.predicted_RF.values), 'prokopchuk_accuracy_score failed on RF'
assert prokopchuk_recall_score(df.actual_label.values,
df.predicted_LR.values) == recall_score(df.actual_label.values,
df.predicted_LR.values), 'prokopchuk_accuracy_score failed on LR'

print('Recall RF: %.3f'%(prokopchuk_recall_score(df.actual_label.values, df.predicted_RF.values)))
print('Recall LR: %.3f'%(prokopchuk_recall_score(df.actual_label.values, df.predicted_LR.values)))
```

Рис 2.10 – лістинг програми

```
from sklearn.metrics import precision_score
precision_score(df.actual_label.values, df.predicted_RF.values)

def prokopchuk_precision_score(y_true, y_pred):  7 usages
 # 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) if (TP + FP) != 0 else 0

assert prokopchuk_precision_score(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 prokopchuk_precision_score(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'%(prokopchuk_precision_score(df.actual_label.values, df.predicted_RF.values)))
print('Precision LR: %.3f'%(prokopchuk_precision_score(df.actual_label.values, df.predicted_LR.values)))
```

Рис 2.11 – лістинг програми

```
from sklearn.metrics import f1_score
f1_score(df.actual_label.values, df.predicted_RF.values)

def prokopchuk_f1_score(y_true, y_pred):  6 usages
    # calculates the F1 score
    recall = prokopchuk_recall_score(y_true,y_pred)
    precision = prokopchuk_precision_score(y_true,y_pred)
    return 2 * (precision * recall) / (precision + recall) if (precision + recall) != 0 else 0
assert np.isclose(prokopchuk_f1_score(df.actual_label.values, df.predicted_RF.values), f1_score(df.actual_label.values,
df.predicted_RF.values)), 'prokopchuk_f1_score failed on RF'
assert np.isclose(prokopchuk_f1_score(df.actual_label.values, df.predicted_LR.values), f1_score(df.actual_label.values,
df.predicted_LR.values)), 'prokopchuk_f1_score failed on LR'
print('F1 RF: %.3f'%(prokopchuk_f1_score(df.actual_label.values, df.predicted_RF.values)))
print('F1 LR: %.3f'%(prokopchuk_f1_score(df.actual_label.values, df.predicted_LR.values)))
print('scores with threshold = 0.5')
print('Accuracy RF: %.3f'%(prokopchuk_accuracy_score(df.actual_label.values, df.predicted_RF.values)))
print('Recall RF: %.3f'%(prokopchuk_recall_score(df.actual_label.values,
df.predicted_RF.values)))
print('Precision RF: %.3f'%(prokopchuk_precision_score(df.actual_label.values, df.predicted_RF.values)))
print('F1 RF: %.3f'%(prokopchuk_f1_score(df.actual_label.values, df.predicted_RF.values)))

print('')
print('scores with threshold = 0.25')
print('Accuracy RF: %.3f'%(prokopchuk_accuracy_score(df.actual_label.values, (df.model_RF >= 0.25).astype('int').values)))
print('Recall RF: %.3f'%(prokopchuk_recall_score(df.actual_label.values, (df.model_RF >= 0.25).astype('int').values)))
print('Precision RF: %.3f'%(prokopchuk_precision_score(df.actual_label.values, (df.model_RF >= 0.25).astype('int').values)))
print('F1 RF: %.3f'%(prokopchuk_f1_score(df.actual_label.values, (df.model_RF >= 0.25).astype('int').values)))

from sklearn.metrics import roc_curve
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)

import matplotlib.pyplot as plt
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()
```

Рис 2.12 – лістинг програми

```
from sklearn.metrics import roc_auc_score
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)

import matplotlib.pyplot as plt
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()
```

Рис 2.13 – лістинг програми

```
TP: 5047
FN: 2832
FP: 2360
TN: 5519
Accuracy (Random Forest): 0.6705165630156111
Accuracy LR: 0.616
Recall RF: 0.641
Recall LR: 0.543
Precision RF: 0.681
Precision LR: 0.636
F1 RF: 0.660
F1 LR: 0.586
scores with threshold = 0.5
Accuracy RF: 0.671
Recall RF: 0.641
Precision RF: 0.681
F1 RF: 0.660

scores with threshold = 0.25
Accuracy RF: 0.502
Recall RF: 1.000
Precision RF: 0.501
F1 RF: 0.668
AUC RF:0.738
AUC LR:0.666
```

Рис 2.14 – результат виконання

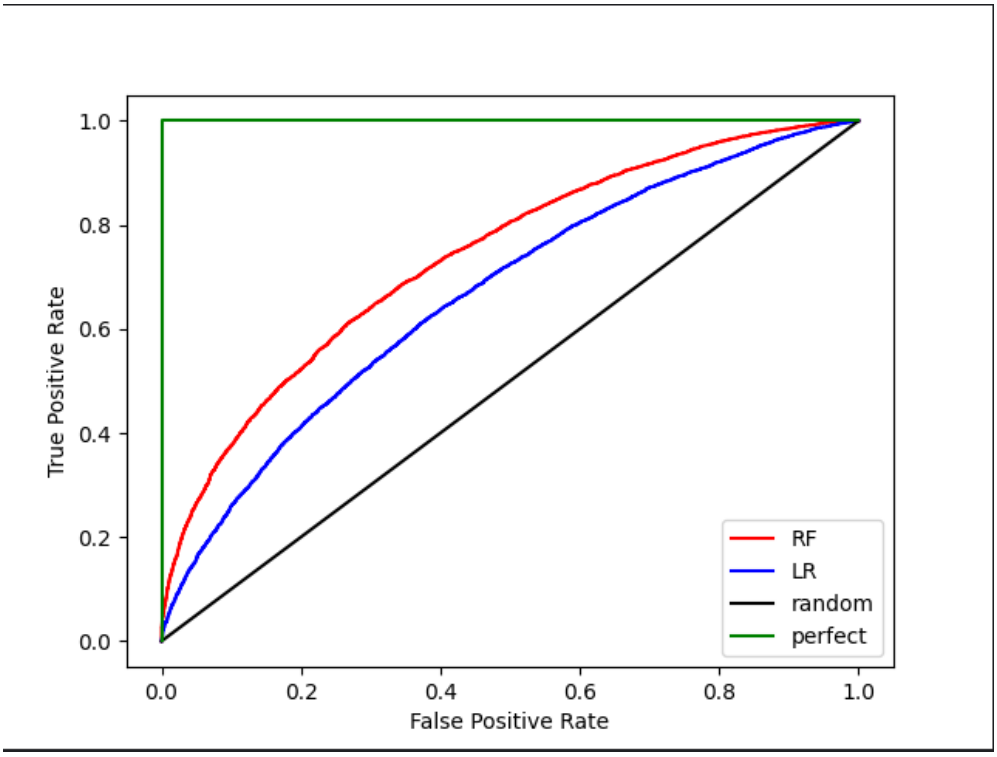

Рис 2.15 – результат виконання

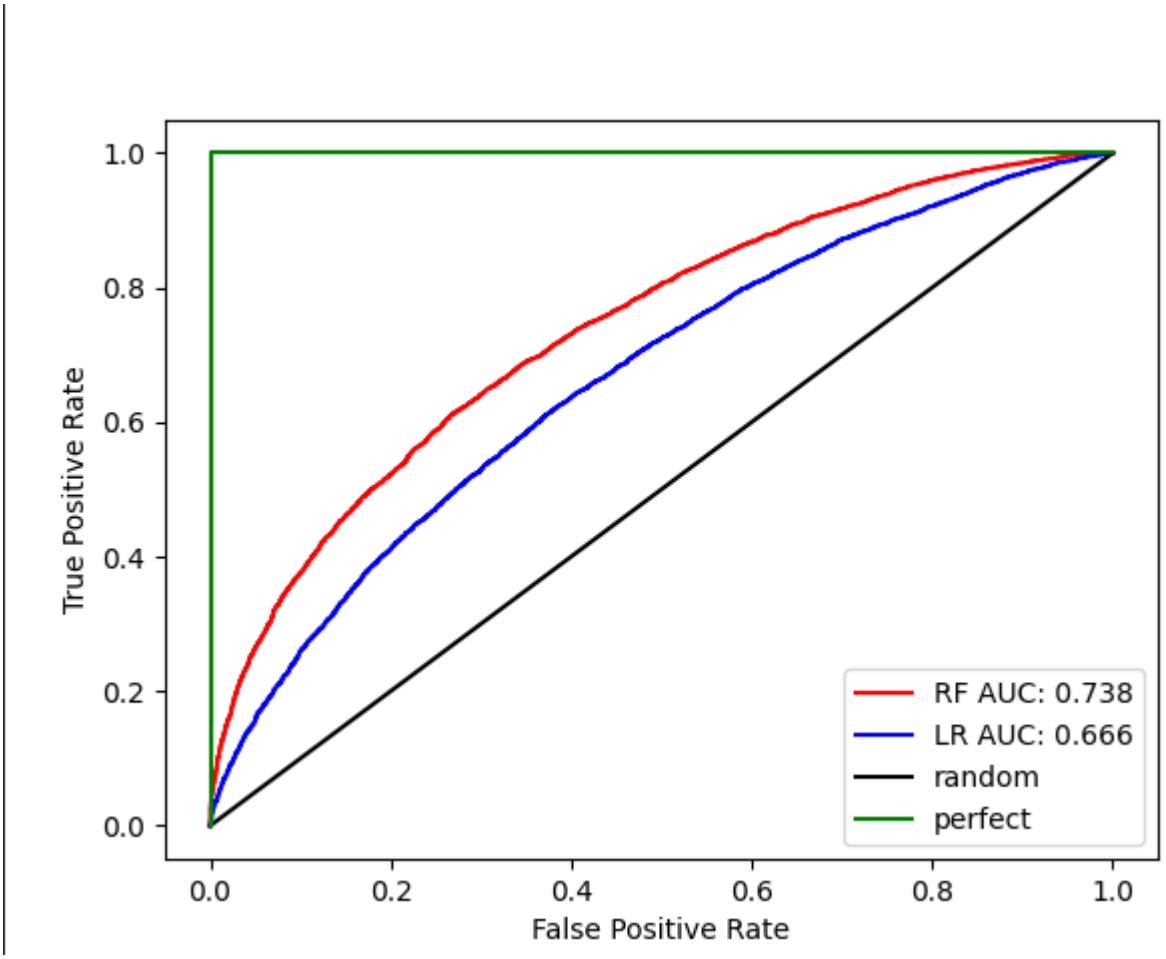

Рис 2.16 – результат виконання

Random Forest (RF) демонструє вищі показники повноти та F1-міри, що вказує на його здатність ефективніше виявляти позитивні приклади, навіть якщо це супроводжується збільшенням кількості хибнопозитивних результатів. Це робить RF оптимальним вибором для завдань, де пріоритетом є виявлення максимальної кількості позитивних випадків, наприклад, у сфері виявлення шахрайства або діагностики захворювань, де критично важливо не пропустити позитивні випадки. Натомість, Logistic Regression (LR) має вищі показники прецизійності, точності та AUC, що робить її більш придатною для ситуацій, де необхідно мінімізувати хибнопозитивні передбачення. Це підходить для завдань, у яких важлива висока точність кожного передбачення, наприклад, у фінансовій або юридичній сфері. Таким чином, вибір моделі залежить від того, що є пріоритетом: максимізація виявлення позитивних випадків (RF) чи зменшення кількості хибнопозитивних передбачень (LR).

Завдання 2.6

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split, cross_val_score
from utilities import visualize_classifier

# Вхідний файл, який містить дані
input_file = 'data_multivar_nb.txt'

# Завантаження даних із вхідного файлу
data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1]

# Розбивка даних на навчальний та тестовий набори
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=3)

# Наївний Байєсівський Класифікатор
classifier_nb = GaussianNB()
classifier_nb.fit(X_train, y_train)
y_test_pred_nb = classifier_nb.predict(X_test)

# Прогнозування та обчислення якості для NB
accuracy_nb = 100.0 * (y_test == y_test_pred_nb).sum() / X_test.shape[0]
print("Accuracy of the Naive Bayes classifier =", round(accuracy_nb, 2), "%")

# Крос-валідація для NB
num_folds = 3
accuracy_values_nb = cross_val_score(classifier_nb, X, y, scoring='accuracy', cv=num_folds)
print("Naive Bayes Accuracy: " + str(round(100 * accuracy_values_nb.mean(), 2)) + "%")
precision_values_nb = cross_val_score(classifier_nb, X, y,scoring='precision_weighted', cv=num_folds)
print("Naive Bayes Precision: " + str(round(100 * precision_values_nb.mean(), 2)) + "%")
recall_values_nb = cross_val_score(classifier_nb, X, y, scoring='recall_weighted', cv=num_folds)
print("Naive Bayes Recall: " + str(round(100 * recall_values_nb.mean(), 2)) + "%")
f1_values_nb = cross_val_score(classifier_nb, X, y, scoring='f1_weighted', cv=num_folds)
print("Naive Bayes F1: " + str(round(100 * f1_values_nb.mean(), 2)) + "%")

# Візуалізація результатів NB
visualize_classifier(classifier_nb, X_test, y_test)

# Support Vector Machine (SVM) Класифікатор
classifier_svm = SVC(kernel='linear', random_state=3)
classifier_svm.fit(X_train, y_train)
y_test_pred_svm = classifier_svm.predict(X_test)
```

Рис 2.17 – лістинг програми

```python
# Прогнозування та обчислення якості для SVM
accuracy_svm = 100.0 * (y_test == y_test_pred_svm).sum() / X_test.shape[0]
print("\nAccuracy of the SVM classifier =", round(accuracy_svm, 2), "%")

# Крос-валідація для SVM
accuracy_values_svm = cross_val_score(classifier_svm, X, y, scoring='accuracy', cv=num_folds)
print("SVM Accuracy: " + str(round(100 * accuracy_values_svm.mean(), 2)) + "%")
precision_values_svm = cross_val_score(classifier_svm, X, y, scoring='precision_weighted', cv=num_folds)
print("SVM Precision: " + str(round(100 * precision_values_svm.mean(), 2)) + "%")
recall_values_svm = cross_val_score(classifier_svm, X, y, scoring='recall_weighted', cv=num_folds)
print("SVM Recall: " + str(round(100 * recall_values_svm.mean(), 2)) + "%")
f1_values_svm = cross_val_score(classifier_svm, X, y, scoring='f1_weighted', cv=num_folds)
print("SVM F1: " + str(round(100 * f1_values_svm.mean(), 2)) + "%")

# Візуалізація результатів SVM
visualize_classifier(classifier_svm, X_test, y_test)

# Порівняння результатів
print("\nComparison between Naive Bayes and SVM:")
print(f"Naive Bayes Accuracy: {round(accuracy_nb, 2)}%, SVM Accuracy: {round(accuracy_svm, 2)}%")
print(f"Naive Bayes F1: {round(100 * f1_values_nb.mean(), 2)}%, SVM F1: {round(100 * f1_values_svm.mean(), 2)}%")
```

Рис 2.18 – лістинг програми

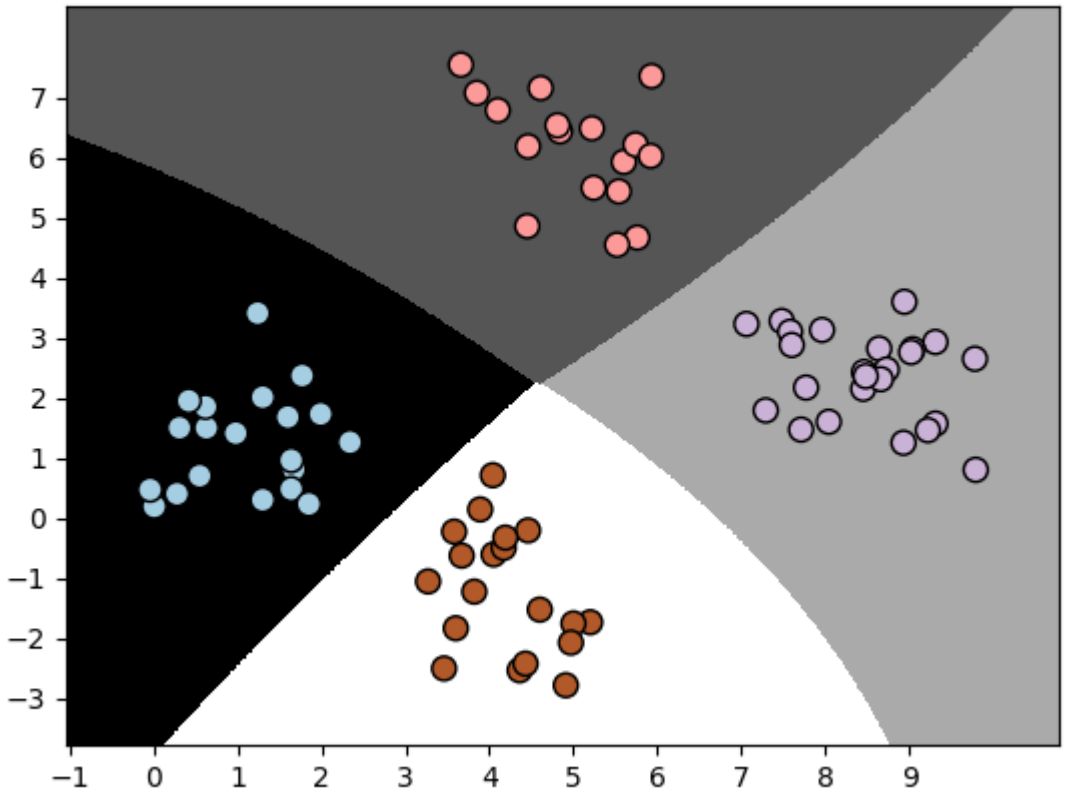

Рис 2.19 – результат виконання

```
Accuracy of the Naive Bayes classifier = 100.0 %
Naive Bayes Accuracy: 99.75%
Naive Bayes Precision: 99.76%
Naive Bayes Recall: 99.75%
Naive Bayes F1: 99.75%

Accuracy of the SVM classifier = 100.0 %
SVM Accuracy: 99.75%
SVM Precision: 99.76%
SVM Recall: 99.75%
SVM F1: 99.75%

Comparison between Naive Bayes and SVM:
Naive Bayes Accuracy: 100.0%, SVM Accuracy: 100.0%
Naive Bayes F1: 99.75%, SVM F1: 99.75%
```

Рис 2.20 – результат виконання

Висновок: У ході виконання лабораторної роботи було усвідомлено, що підготовка даних є надзвичайно важливим етапом у процесі машинного навчання, оскільки вона істотно впливає на ефективність моделі. Застосування правильних методів нормалізації та масштабування сприяє покращенню результатів класифікації. Також вдалося поглибити знання щодо аналізу даних завдяки роботі з бібліотеками Python. Ця лабораторна робота наочно продемонструвала ключову роль підготовки даних перед їх використанням для навчання моделей.

Посилання на репозиторій: