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:

Посилання на репозиторій: https://github.com/SerhiiHrushevitskiy/AI_lab1.git

Завдання 1: Попередня обробка даних

Лістинг коду:

```
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))
# Исключение среднего
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)
```

Результат виконання коду:

```
 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]]

Process finished with exit code 0
```

Рис 1.1 Результат виконання коду

Відмінність між L1-нормалізацією та L2-нормалізацією полягає у їхньому підході до нормалізації даних у машинному навчанні. Обидва ці методи спрямовані на приведення даних до формату, який корисний для моделей машинного навчання:

- L1-нормалізація використовує абсолютні значення ознак і робить так, щоб сума абсолютних значень в кожному ряду дорівнювала 1. Вона менш чутлива до викидів.

- L2-нормалізація використовує квадрати значень ознак і робить так, щоб сума квадратів значень в кожному ряду дорівнювала 1. Вона більш чутлива до викидів, оскільки враховує квадрати.

Ці два методи впливають на дані та їхню чутливість до викидів по-різному, що може вплинути на результати обробки даних. Що важливо відзначити, L1-нормалізація може мати корисний побічний ефект, який приводить до того, що одне або більше вагових значень стають рівними 0.0. З іншого боку, L2-нормалізація обмежує вагові значення моделі, але часто призводить до повного обнулення цих значень.

Тому, може виникнути враження, що L1-нормалізація є більш вигідною ніж L2-нормалізація. Проте слід зауважити, що L1-нормалізацію не так просто використовувати з деякими алгоритмами машинного навчання, особливо з тими, які використовують чисельні методи для обчислення градієнту. У той час як L2-нормалізацію можна застосовувати з будь-яким типом алгоритму навчання.

Завдання 2: Кодування міток
Лістинг коду:

```
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 ) )
```

Результат виконання коду:

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

Labels = ['green', 'red', 'black']
Encoded values = [1, 2, 0]

Encoded values = [3, 0, 4, 1]
Decoded labels = ['white', 'black', 'yellow', 'green']

Process finished with exit code 0
```

Рис 2.1 Результат виконання коду

Завдання 3: Попередня обробка нових даних

Лістинг коду:

```
import numpy as np
from sklearn import preprocessing

input_data = np.array([[1.3, 3.9, 6.2],
                       [4.9, 2.2, -4.3],
                       [-2.2, 6.5, 4.1],
                       [-5.2, -3.4, -5.2]])

# Бінаризація даних
data_binarized = preprocessing.Binarizer(threshold=2.0).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))
# Исключение среднего
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)
```

Результат виконання коду:

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

BEFORE:
Mean = [-0.3  2.3  0.2]
Std deviation = [3.78219513 3.62973828 5.01547605]

AFTER:
Mean = [-5.55111512e-17  5.55111512e-17  0.00000000e+00]
Std deviation = [1. 1. 1.]

Min max scaled data:
 [[0.64356436 0.73737374 1.        ]
 [1.         0.56565657 0.07894737]
 [0.2970297  1.         0.81578947]
 [0.         0.         0.        ]]

l1 normalized data:
 [[ 0.11403509  0.34210526  0.54385965]
 [ 0.42982456  0.19298246 -0.37719298]
 [-0.171875    0.5078125   0.3203125 ]
 [-0.37681159 -0.24637681 -0.37681159]]

l2 normalized data:
 [[ 0.17475265  0.52425796  0.83343572]
 [ 0.71216718  0.31974853 -0.62496303]
 [-0.2752151   0.81313551  0.51290086]
 [-0.64182859 -0.41965715 -0.64182859]]

Process finished with exit code 0
```

Рис 3.1 Результат виконання коду

Завдання 4: Класифікація логістичною регресією або логістичний класифікатор

Лістинг коду:

```
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)
# Візуалізуємо результат роботи класифікатора
visualize_classifier(classifier, X, y)
```

Результат виконання коду:

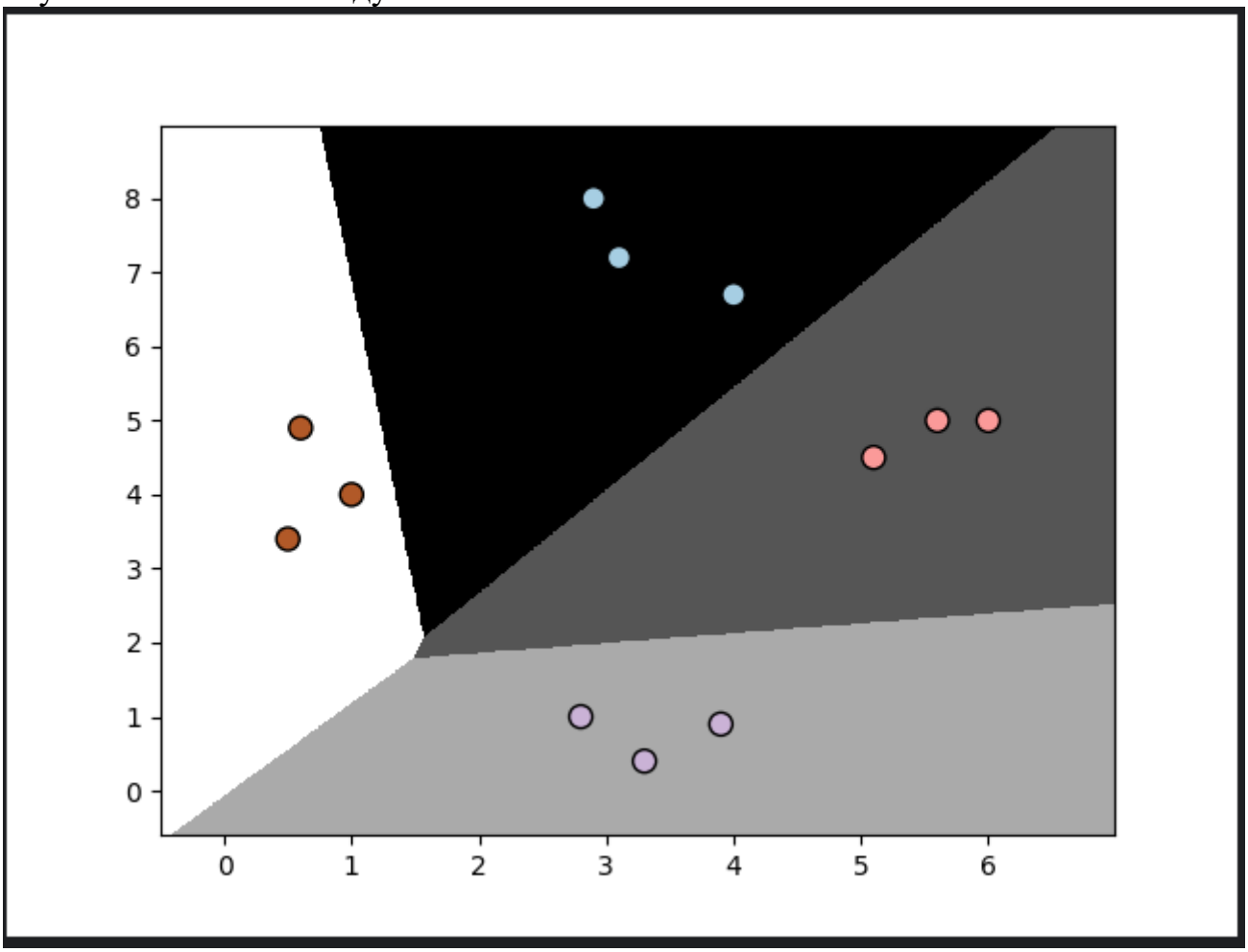

Рис 4.1 Результат виконання коду

Завдання 5: Класифікація наївним байєсовським класифікатором

Лістинг коду:

```
import numpy as np
import matplotlib.pyplot as plt

from sklearn.naive_bayes import GaussianNB
import sklearn.model_selection
from utilities import visualize_classifier

train_test_split = sklearn.model_selection.train_test_split
cross_val_score = sklearn.model_selection.cross_val_score
# 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)


# Розбивка даних на навчальний та тестовий набори
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran-
dom_state=3)
classifier_new = GaussianNB()
classifier_new.fit(X_train, y_train)
y_test_pred = classifier_new.predict(X_test)

# 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),
"%")
# Візуалізація роботи класифікатора
visualize_classifier(classifier_new, X_test, y_test)

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)) + "%")
```

Результат виконання коду:

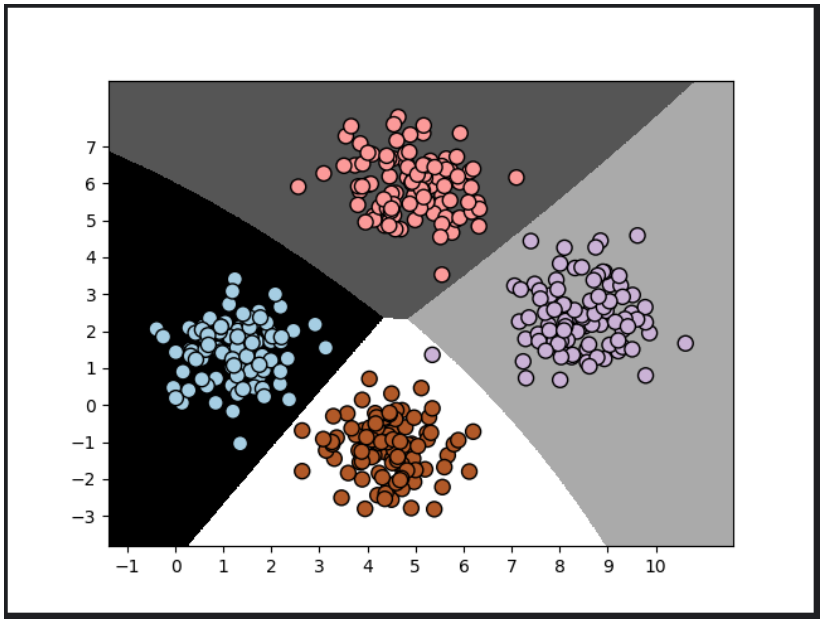

Рис 5.1 Результат виконання коду (1)

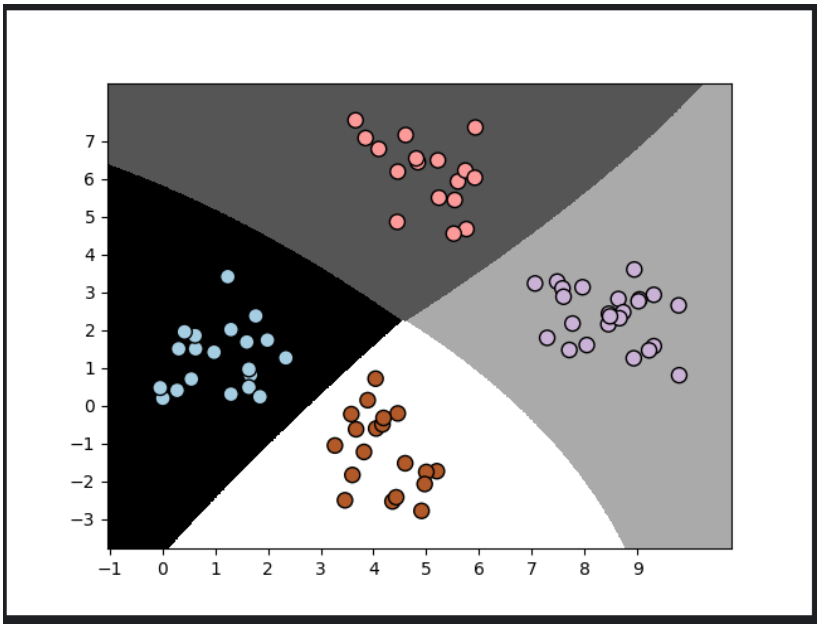

Рис 5.2 Результат виконання коду (2)

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

Process finished with exit code 0
```

Рис 5.3 Результат виконання коду (3)

Завдання 6: Класифікація наївним байєсовським класифікатором
Лістинг коду:

```
import pandas as pd

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()
from sklearn.metrics import confusion_matrix

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))
import numpy as np


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 Hrushevitskiy_confusion_matrix(y_true, y_pred):
    TP, FN, FP, TN = find_conf_matrix_values(y_true, y_pred)
    return np.array([[TN, FP], [FN, TP]])


print(Hrushevitskiy_confusion_matrix(df.actual_label.values,
df.predicted_RF.values))
assert np.array_equal(Hrushevitskiy_confusion_matrix(df.actual_label.values,
df.predicted_RF.values),
                      confusion_matrix(df.actual_label.values,
                                       df.predicted_RF.values)),
```

```
'Hrushevitskiy_confusion_matrix() is not correct for RF'
assert np.array_equal(Hrushevitskiy_confusion_matrix(df.actual_label.values,
df.predicted_LR.values),
                      confusion_matrix(df.actual_label.values,
                                       df.predicted_LR.values)),
'Hrushevitskiy_confusion_matrix() is not correct for LR'
from sklearn.metrics import accuracy_score

print(accuracy_score(df.actual_label.values, df.predicted_RF.values))


def Hrushevitskiy_accuracy_score(y_true, y_pred):
    # 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 Hrushevitskiy_accuracy_score(df.actual_label.values,
df.predicted_RF.values) == accuracy_score(
    df.actual_label.values,
    df.predicted_RF.values), 'Hrushevitskiy_accuracy_score failed on RF'
assert Hrushevitskiy_accuracy_score(df.actual_label.values,
                                    df.predicted_LR.values) == accura-
cy_score(df.actual_label.values,

df.predicted_LR.values), 'Hrushevitskiy_accuracy_score failed on LR'
print('Accuracy RF:%.3f' % (Hrushevitskiy_accuracy_score(df.actual_label.values,
df.predicted_RF.values)))
print('Accuracy LR:%.3f' % (Hrushevitskiy 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 Hrushevitskiy_recall_score(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)


assert Hrushevitskiy_recall_score(df.actual_label.values, df.predicted_RF.values)
== recall_score(
    df.actual_label.values,
    df.predicted_RF.values), 'Hrushevitskiy_recall_score failed on RF'
assert Hrushevitskiy_recall_score(df.actual_label.values, df.predicted_LR.values)
== recall_score(
    df.actual_label.values, df.predicted_LR.values), 'Hrushevitskiy_recall_score
failed on LR'
print('Recall RF: %.3f' % (Hrushevitskiy_recall_score(df.actual_label.values,
df.predicted_RF.values)))
print('Recall LR: %.3f' % (Hrushevitskiy_recall_score(df.actual_label.values,
df.predicted_LR.values)))
from sklearn.metrics import precision_score

precision_score(df.actual_label.values, df.predicted_RF.values)


def Hrushevitskiy_precision_score(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)


assert Hrushevitskiy_precision_score(df.actual_label.values,
df.predicted_RF.values) == precision_score(
    df.actual_label.values, df.predicted_RF.values),
'Hrushevitskiy_precision_score failed on RF'
assert Hrushevitskiy_precision_score(df.actual_label.values,
df.predicted_LR.values) == precision_score(
    df.actual_label.values, df.predicted_LR.values),
'Hrushevitskiy_precision_score failed on LR'
print('Precision RF:%.3f' % (Hrushevitskiy_precision_score(df.actual_label.values,
df.predicted_RF.values)))
from sklearn.metrics import f1_score

f1_score(df.actual_label.values, df.predicted_RF.values)


def Hrushevitskiy_f1_score(y_true, y_pred):
    # calculates the F1 score
    recall = Hrushevitskiy_recall_score(y_true, y_pred)
    precision = Hrushevitskiy_precision_score(y_true, y_pred)
    return (2 * (precision * recall)) / (precision + recall)


assert Hrushevitskiy_f1_score(df.actual_label.values, df.predicted_RF.values) ==
f1_score(df.actual_label.values,

df.predicted RF.values), 'Hrushevitskiy f1 score failed on RF'
assert Hrushevitskiy_f1_score(df.actual_label.values, df.predicted_LR.values) ==
f1_score(df.actual_label.values,

df.predicted_LR.values), 'Hrushevitskiy_f1_score failed on LR'
print('F1 RF: %.3f' % (Hrushevitskiy_f1_score(df.actual_label.values,
df.predicted_RF.values)))
print('F1 LR: %.3f' % (Hrushevitskiy_f1_score(df.actual_label.values,
df.predicted_LR.values)))

print('scores with threshold = 0.5')
print('Accuracy RF:%.3f' % (Hrushevitskiy_accuracy_score(df.actual_label.values,
df.predicted_RF.values)))
print('Recall RF: %.3f' % (Hrushevitskiy_recall_score(df.actual_label.values,
df.predicted_RF.values)))
print('Precision RF:%.3f' % (Hrushevitskiy_precision_score(df.actual_label.values,
df.predicted_RF.values)))
print('F1 RF: %.3f' % (Hrushevitskiy_f1_score(df.actual_label.values,
df.predicted_RF.values)))
print('')
print('scores with threshold = 0.25')
print('Accuracy RF:%.3f' % (
    Hrushevitskiy_accuracy_score(df.actual_label.values, (df.model_RF >=
0.25).astype('int').values)))
print('Recall RF: %.3f' % (
    Hrushevitskiy_recall_score(df.actual_label.values, (df.model_RF >=
0.25).astype('int').values)))
print('Precision RF:%.3f' % (
    Hrushevitskiy_precision_score(df.actual_label.values, (df.model_RF >=
0.25).astype('int').values)))
print('F1 RF: %.3f' % (Hrushevitskiy_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()
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()
```

Результат виконання коду:

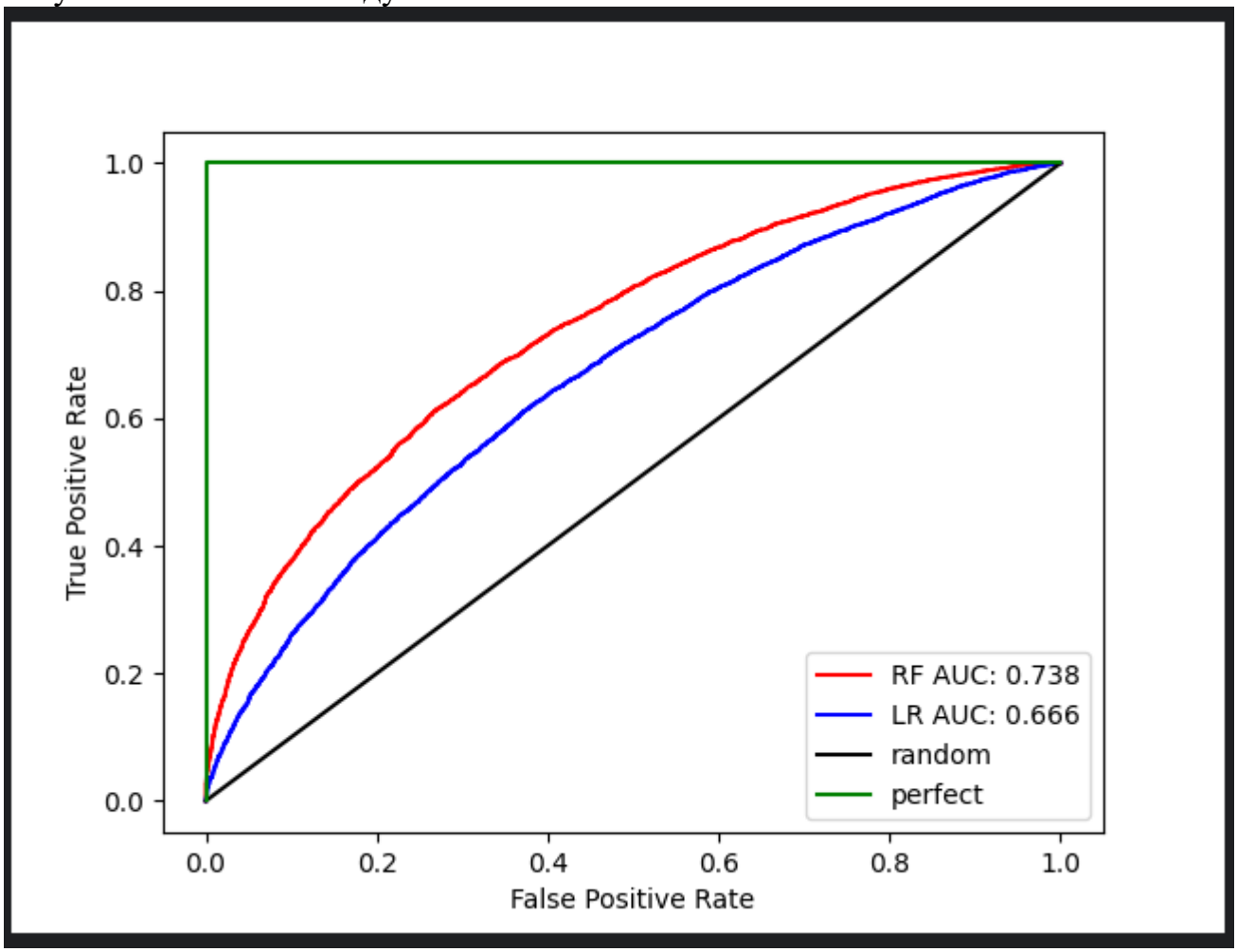

Рис 6.1 Результат виконання коду (1)

```
[[5519 2360]
 [2832 5047]]
TP: 5047
FN: 2832
FP: 2360
TN: 5519
[[5519 2360]
 [2832 5047]]
0.6705165630156111
Accuracy RF:0.671
Accuracy LR:0.616
Recall RF: 0.641
Recall LR: 0.543
Precision RF:0.681
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

Process finished with exit code 0
```

Рис 6.2 Результат виконання коду (2)

Після аналізу результатів для різних порогів було зроблено наступні спостереження:

- Зниження порогу призводить до збільшення значення відгуку (Recall), але при цьому знижується точність (Precision).
- У випадку високого порогу, модель, як правило, має більш високу точність, але одночасно менший відгук.
- F1-показник може виявитися корисним показником, оскільки він враховує баланс між точністю та відгуком при різних порогах.

Завдання 7: Розробіть програму класифікації даних в файлі data_multivar_nb.txt за допомогою машини опорних векторів (Support Vector Machine - SVM). Розрахуйте показники якості класифікації. Порівняйте їх з показниками наївного байєсівського класифікатора. Зробіть висновки яку модель класифікації краще обрати і чому.

Лістинг коду:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score

# Зчитайте дані з файлу
data = pd.read_csv('data_multivar_nb.txt', header=None)
X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values

# Розділіть дані на навчальний та тестовий набори
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran-
dom_state=42)

# Навчання SVM
svm_classifier = SVC(kernel='linear')
svm_classifier.fit(X_train, y_train)
y_pred_svm = svm_classifier.predict(X_test)

# Навчання наївного байєсівського класифікатора
nb_classifier = GaussianNB()
nb_classifier.fit(X_train, y_train)
y_pred_nb = nb_classifier.predict(X_test)

# Оцінка SVM
accuracy_svm = accuracy_score(y_test, y_pred_svm)
precision_svm = precision_score(y_test, y_pred_svm, average='weighted')
recall_svm = recall_score(y_test, y_pred_svm, average='weighted')
f1_svm = f1_score(y_test, y_pred_svm, average='weighted')

# Оцінка наївного байєсівського класифікатора
accuracy_nb = accuracy_score(y_test, y_pred_nb)
precision_nb = precision_score(y_test, y_pred_nb, average='weighted')
recall_nb = recall_score(y_test, y_pred_nb, average='weighted')
f1_nb = f1_score(y_test, y_pred_nb, average='weighted')

# Виведення результатів
print("SVM:")
print("Accuracy:", accuracy_svm)
```

```
print("Precision:", precision_svm)
print("Recall:", recall_svm)
print("F1 Score:", f1_svm)

print("\nNaive Bayes:")
print("Accuracy:", accuracy_nb)
print("Precision:", precision_nb)
print("Recall:", recall_nb)
print("F1 Score:", f1_nb)
```

Результат виконання коду:

```
SVM:
Accuracy: 0.9875
Precision: 0.9885416666666667
Recall: 0.9875
F1 Score: 0.9876263902932255

Naive Bayes:
Accuracy: 0.9875
Precision: 0.9885416666666667
Recall: 0.9875
F1 Score: 0.9876263902932255

Process finished with exit code 0
```

Рис 7.1 Результат виконання

Результати показують, що обидва класифікатори є прийнятними для вашого завдання, і ви можете вибрати той, який найкраще відповідає вашим конкретним потребам і обмеженням.

Висновок : під час виконання лабораторної роботи використовуючи спеціалізовані бібліотеки та мову програмування Python дослідив попередню обробку та класифікацію даних.