ЛАБОРАТОРНА РОБОТА № 1
ПОПЕРЕДНЯ ОБРОБКА ТА КОНТРОЛЬОВАНА КЛАСИФІКАЦІЯ ДАНИХ

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

Посилання на репозиторій: https://github.com/D1nqq/AI_Labs_Baginskiy

```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))

# 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)
```

Рис.1 Скріншот коду

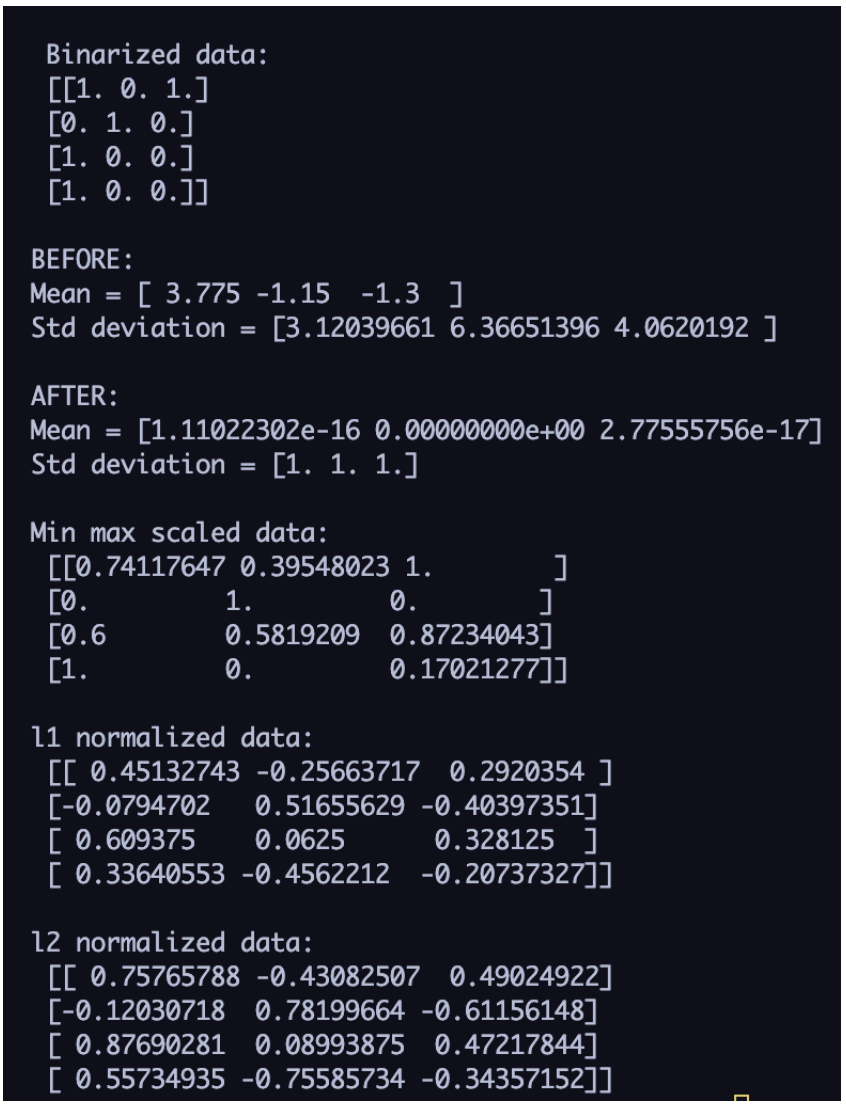


Рис.2 Результат виконання

Висновок:
L1-нормалізація зменшує значення на основі абсолютної суми, надаючи розрідженість даних і зберігаючи структуру великих і малих значень.
L2-нормалізація зменшує значення на основі суми квадратів і більше фокусується на рівномірному масштабуванні всіх елементів.

LR_1_task _ 1.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.3. Скрін коду до завдання 1

```
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']
```

Рис.4. Результат виконання завдання 1

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

input_data = np.array([[4.3, -9.9, -3.5],
                       [-2.9, 4.1, 3.3],
                       [-2.2, 8.8, -6.1],
                       [3.9, 1.4, 2.2]])

# Бінаризація даних
data_binarized = preprocessing.Binarizer(threshold=2.2).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 среднього
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.5. Скрін коду до 2 завдання

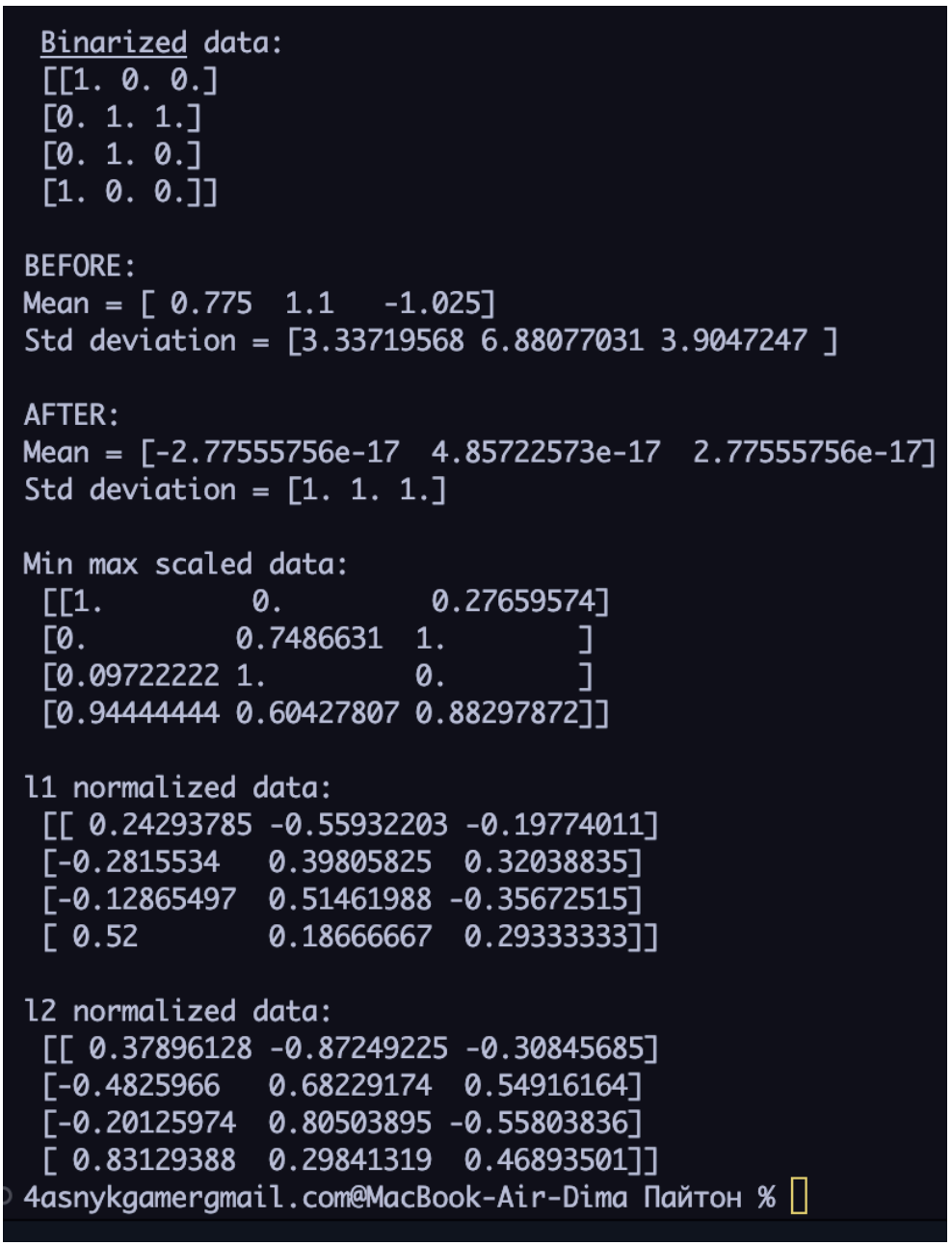


Рис.6. Результат виконання 2 завдання

```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.7. Скрін коду до 3-го завдання

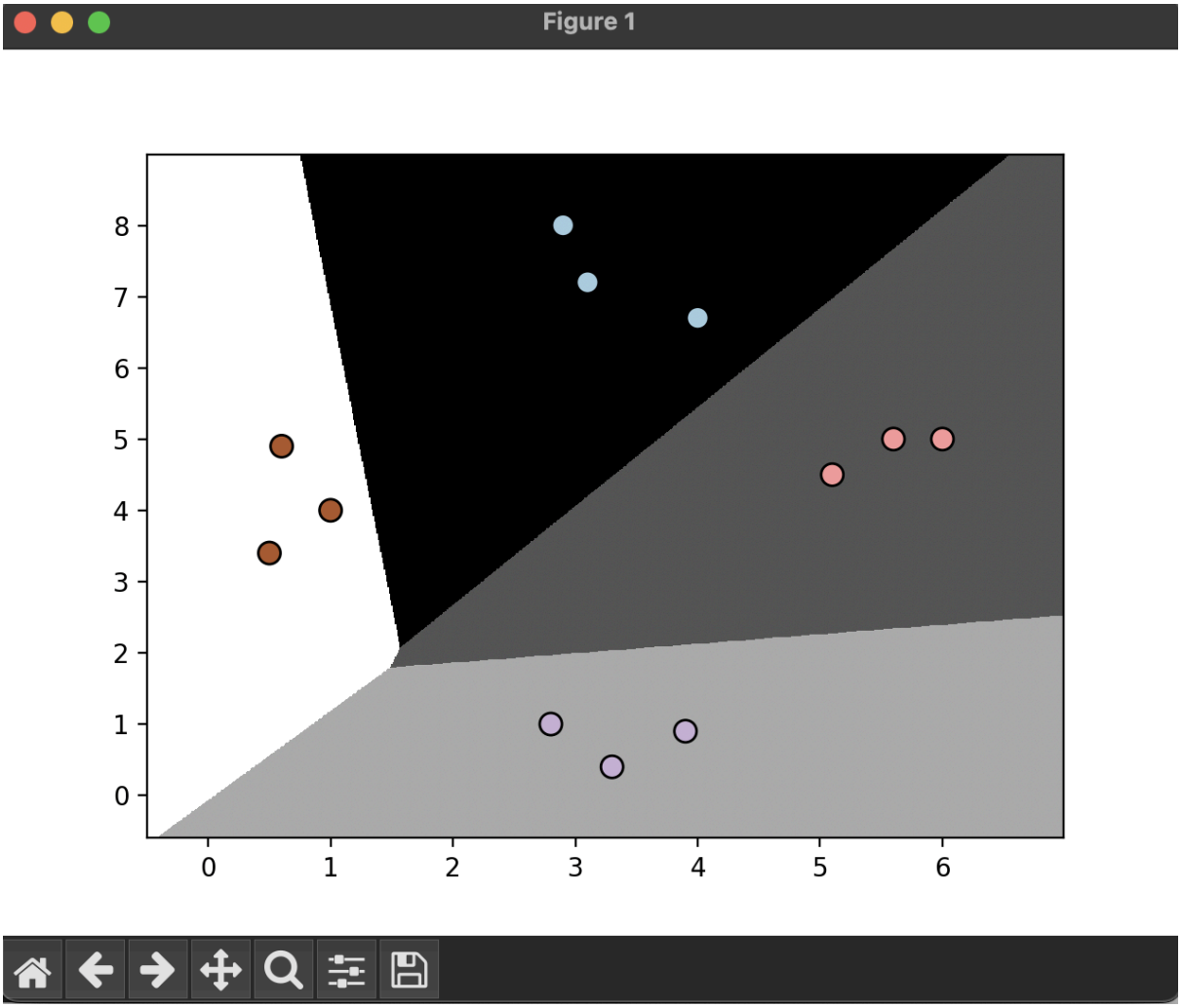
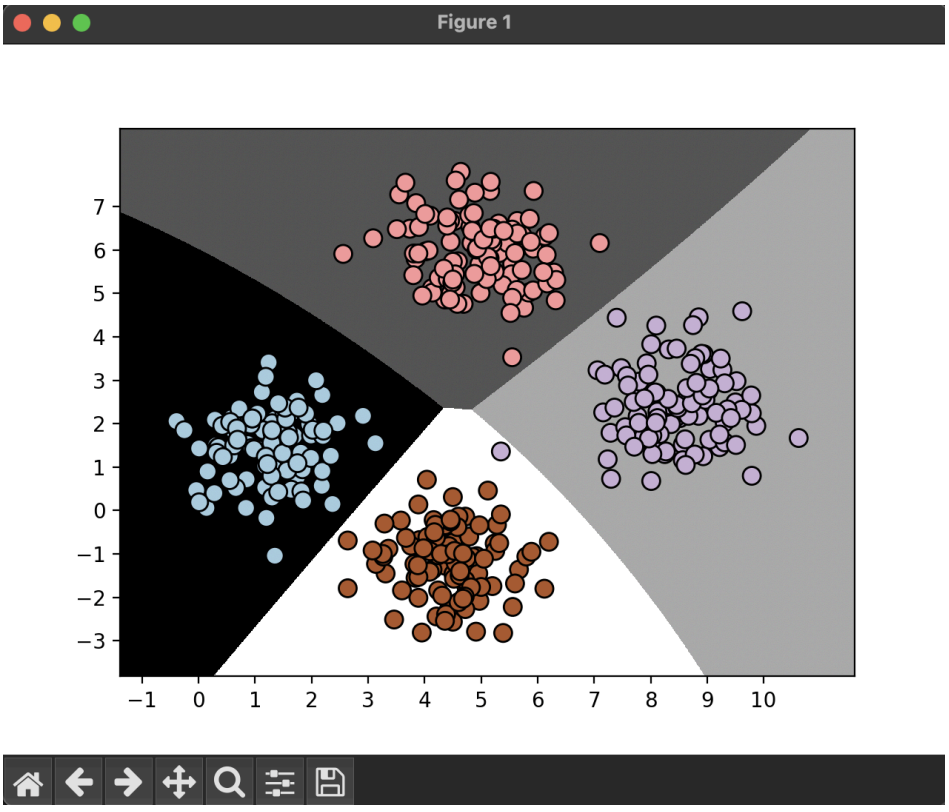


Рис.8. Результат виконання 3 завдання

```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.9. Скрін коду




```
Accuracy of Naive Bayes classifier = 99.75 %
```

Рис.10. Результат виконання завдання

```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]

# 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, random_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)
```

```python
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)) + "%")
```

Рис.11. Скріни коду

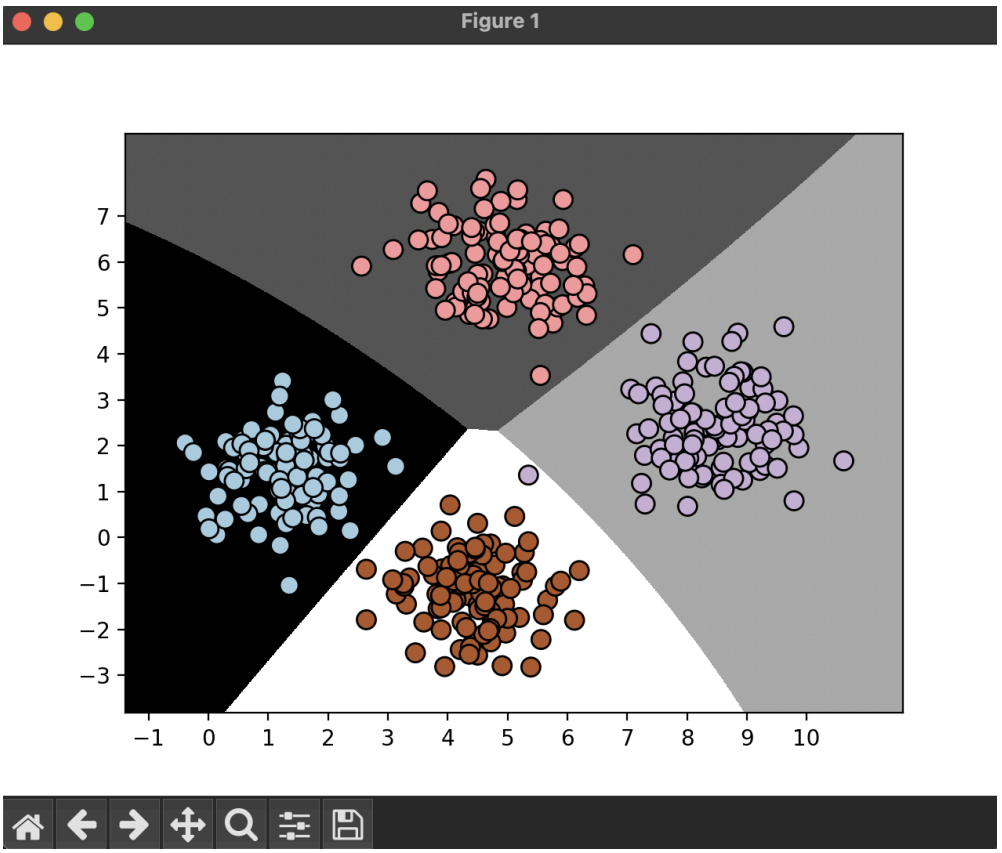


Рис.12. Результат першого прогону

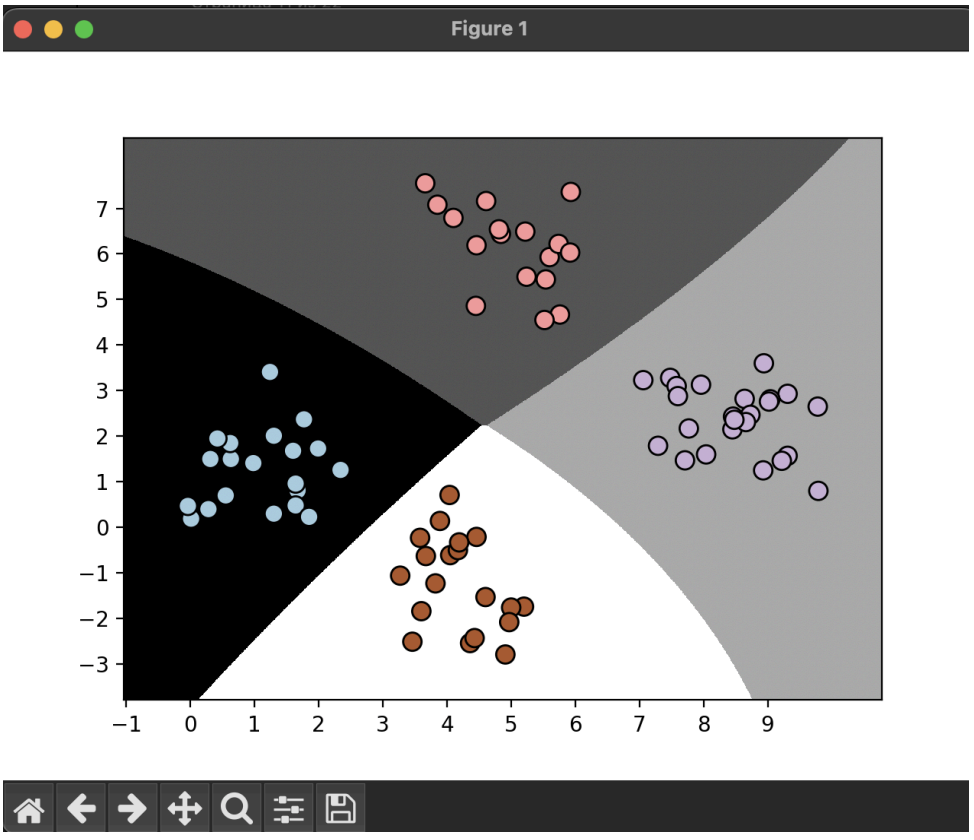


Рис.13. Результат другого прогону

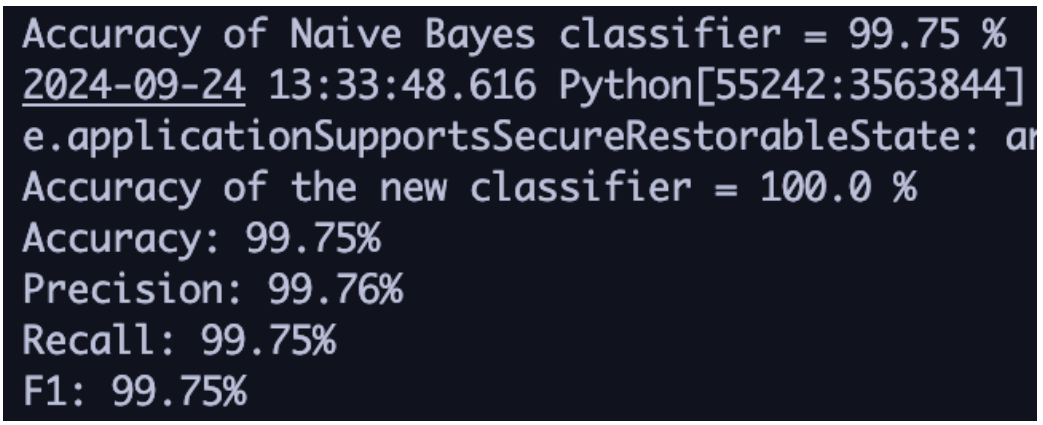


Рис.14. Результат виконання програми

Точність початкового класифікатора (на всьому наборі даних):

Accuracy: 99.75%

Це свідчить про дуже високу точність наївного байєсівського класифікатора, коли перевірка проводиться на тому ж наборі даних, що й тренування.

Точність нового класифікатора (на тестових даних):

Accuracy: 100.0%

Після розділення даних на тренувальні та тестові набори, новий класифікатор показав 100% точність на тестових даних, що може свідчити про відсутність помилок у прогнозуванні на цьому наборі даних.

Перехресна перевірка:

Accuracy: 99.75%

Precision: 99.76%

Recall: 99.75%

F1: 99.75%

Ці результати показують стабільно високу якість класифікації на всіх метриках, що підтверджує ефективність моделі.

Висновки

Наївний байєсівський класифікатор продемонстрував високу точність та стабільні результати. Оскільки точність на тестових даних досягла 100%, модель може бути надто пристосованою до цих даних, але загальні результати свідчать про хорошу загальну узгодженість моделі з навчальними даними.

Перехресна перевірка також підтвердила, що модель ефективно справляється з класифікацією, показуючи стабільні показники **Accuracy**, **Precision**, **Recall** та **F1**-міри.

Порівняйте результати для різних порогів та зробіть висновки.

Порівняння

Accuracy:

При порозі 0.5 точність моделі становить 0.671, що свідчить про те, що модель правильно класифікує приблизно 67.1% всіх зразків. Однак, при зменшенні порогу до 0.25 точність знижується до 0.502, що означає, що лише 50.2% зразків були правильно класифіковані.

Recall:

Recall при порозі 0.5 становить 0.641. Це означає, що 64.1% позитивних зразків були вірно передбачені. При зниженні порогу до 0.25 згадка зростає до 1.000, що вказує на те, що всі позитивні зразки (100%) були правильно класифіковані, але це призводить до великої кількості хибних сповіщень.

Precision:

При порозі 0.5 точність становить 0.681, що означає, що з усіх передбачених позитивних зразків 68.1% дійсно позитивні. При порозі 0.25 точність знижується до 0.501, що вказує на те, що лише 50.1% передбачених позитивних зразків є дійсно позитивними. Це свідчить про велику кількість хибних позитивів при нижчому порозі.

F1-метрика:

F1-метрика при порозі 0.5 становить 0.660, тоді як при порозі 0.25 зростає до 0.668. F1-метрика є гарним компромісом між точністю та згадкою, тому зростання значення F1-метрики при зниженні порогу свідчить про покращення збалансованості між згадкою та точністю, незважаючи на те, що точність знизилась.

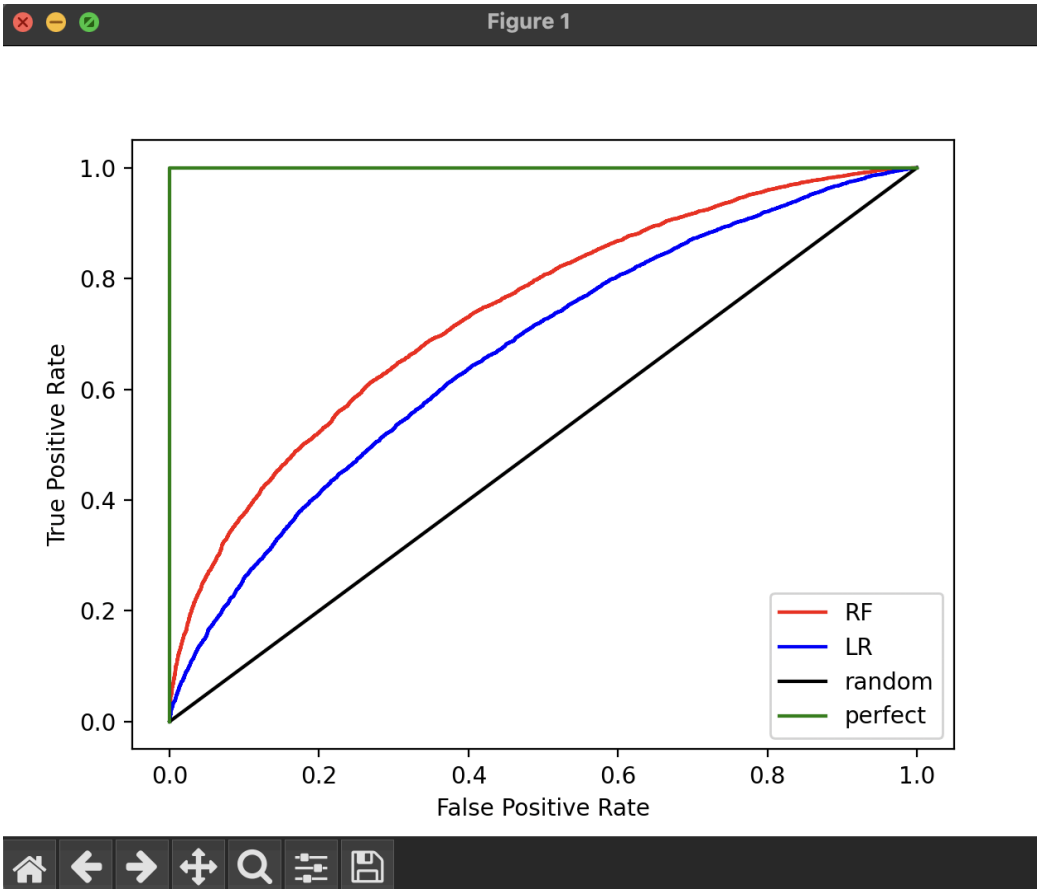
Висновок:

Зменшення порогу з 0.5 до 0.25 призводить до збільшення **Recall**, але за рахунок значного зниження точності. Модель стає "агресивнішою" у виявленні позитивних випадків, але також збільшує кількість хибних позитивів, що може бути неприйнятним у деяких контекстах. Вибір оптимального порогу залежить від конкретних вимог задачі: якщо важливіше виявити якомога більше позитивних випадків, може бути доцільно зменшити поріг. Якщо ж важливіше уникнути хибних сповіщень, то вищий поріг може бути кращим вибором.

```
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
```

Рис.15. Результат виконання з різними порогами



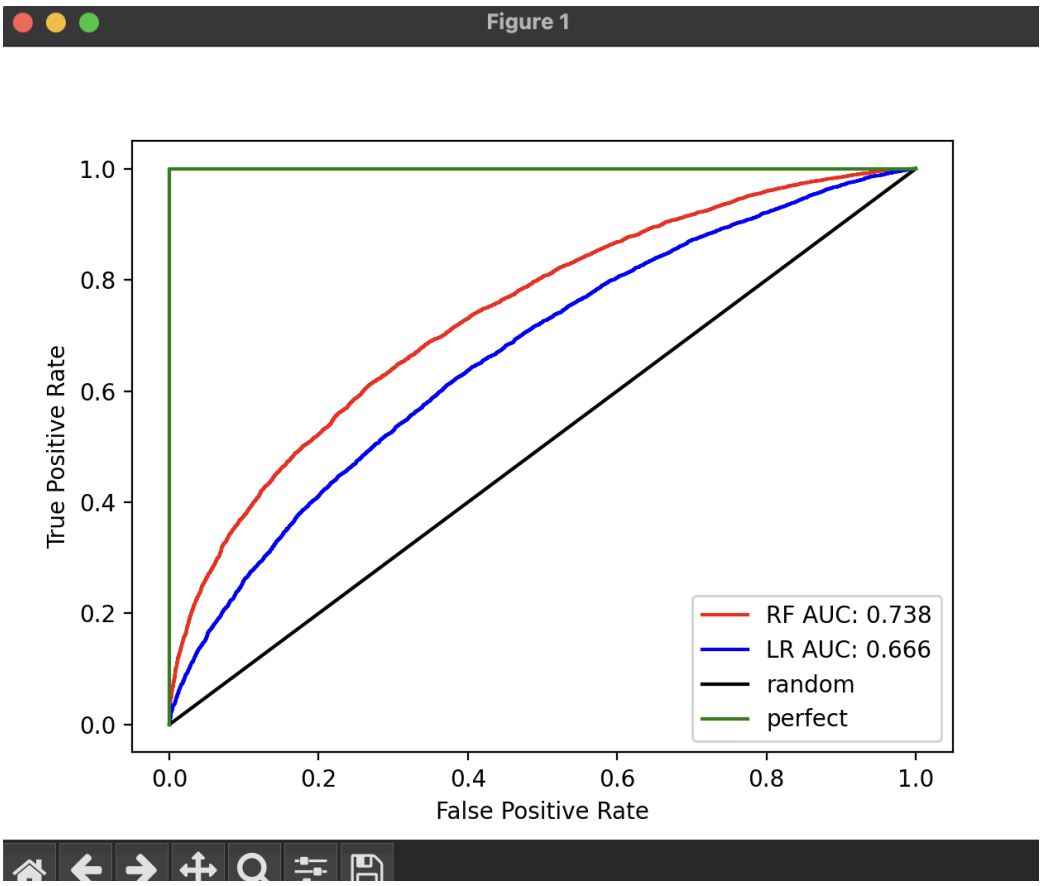


Рис.16. Крива Roc

Висновок: **RF** показує кращі результати, ніж **LR**, оскільки модель **RF** має вищу точність і кращий баланс між точністю та згадкою, що підтверджується **F1**-метрикою. Крім того, **RF** краще працює в задачах з більш складною структурою даних, оскільки враховує нелінійні залежності, тоді як **LR** має обмеження через лінійність. Враховуючи ці фактори, можна зробити висновок, що **модель RF є кращим вибором** у даному випадку.

```
TP: 5047
FN: 2832
FP: 2360
TN: 5519
All assertions passed. Your confusion matrix functions are correct!
Accuracy RF: 0.671
Accuracy LR: 0.616
Recall RF: 0.641
Recall LR: 0.543
Precision RF: 0.681
Precision LR: 0.636
F1 RF:  0.660342797330891
F1 LR:  0.5856830002737475
F1 RF original:  0.660342797330891
F1 LR original:  0.5856830002737476

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.17. Результати виконання

Лістинг коду:

```
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix, accuracy_score, recall_score,
precision_score, f1_score

df = pd.read_csv('data_metrics.csv')
df.head()
thresh = 0.5
thresholds = [0.5, 0.25]
```

```
df['predicted_RF'] = (df.model_RF >= 0.5).astype('int')
df['predicted_LR'] = (df.model_LR >= 0.5).astype('int')
df.head()

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):

    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
```

```python
def baginskiy_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]])


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

# Перевірка коректності матриці плутанини
assert np.array_equal(baginskiy_confusion_matrix(df.actual_label.values,
df.predicted_RF.values),
                      baginskiy_confusion_matrix(df.actual_label.values,
df.predicted_RF.values)), \
    'baginskiy_confusion_matrix() is not correct for RF'

assert np.array_equal(baginskiy_confusion_matrix(df.actual_label.values,
df.predicted_LR.values),
                      baginskiy_confusion_matrix(df.actual_label.values,
df.predicted_LR.values)), \
    'baginskiy_confusion_matrix() is not correct for LR'

print("All assertions passed. Your confusion matrix functions are correct!")


# Ваша власна функція для обчислення точності
def baginskiy_accuracy_score(y_true, y_pred):
    TP, FN, FP, TN = find_conf_matrix_values(y_true, y_pred)
    total = TP + FN + FP + TN
    return (TP + TN) / total if total > 0 else 0.0  # Обчислення точності


# Перевірка точності за допомогою assert
assert baginskiy_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 baginskiy_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'

# Виведення результатів
print('Accuracy RF: %.3f' % baginskiy_accuracy_score(
    df.actual_label.values, df.predicted_RF.values))
print('Accuracy LR: %.3f' % baginskiy_accuracy_score(
    df.actual_label.values, df.predicted_LR.values))


# Ваша власна функція для обчислення відзиву (recall)
def baginskiy_recall_score(y_true, y_pred):
    TP, FN, FP, TN = find_conf_matrix_values(y_true, y_pred)
    total_actual_positives = TP + FN
    # Обчислення відзиву
    return TP / total_actual_positives if total_actual_positives > 0 else 0.0


# Перевірка відзиву за допомогою assert
assert baginskiy_recall_score(df.actual_label.values, df.predicted_RF.values)
== recall_score(
    df.actual_label.values, df.predicted_RF.values), 'my_recall_score failed
on RF'
assert baginskiy_recall_score(df.actual_label.values, df.predicted_LR.values)
== recall_score(
    df.actual_label.values, df.predicted_LR.values), 'my_recall_score failed
on LR'

# Виведення результатів
print('Recall RF: %.3f' % baginskiy_recall_score(
    df.actual_label.values, df.predicted_RF.values))
print('Recall LR: %.3f' % baginskiy_recall_score(
    df.actual_label.values, df.predicted_LR.values))

# Ваша власна функція для обчислення точності (precision)


def baginskiy_precision_score(y_true, y_pred):
    TP, FN, FP, TN = find_conf_matrix_values(y_true, y_pred)
    total_predicted_positives = TP + FP
    # Обчислення точності
```

```
    return TP / total_predicted_positives if total_predicted_positives > 0
else 0.0


# Перевірка точності за допомогою assert
assert baginskiy_precision_score(df.actual_label.values,
df.predicted_RF.values) == precision_score(
    df.actual_label.values, df.predicted_RF.values), 'my_precision_score
failed on RF'
assert baginskiy_precision_score(df.actual_label.values,
df.predicted_LR.values) == precision_score(
    df.actual_label.values, df.predicted_LR.values), 'my_precision_score
failed on LR'

# Виведення результатів
print('Precision RF: %.3f' % baginskiy_precision_score(
    df.actual_label.values, df.predicted_RF.values))
print('Precision LR: %.3f' % baginskiy_precision_score(
    df.actual_label.values, df.predicted_LR.values))


# Ваша власна функція для обчислення F1-метрики
def baginskiy_f1_score(y_true, y_pred):
    recall = baginskiy_recall_score(y_true, y_pred)
    precision = baginskiy_precision_score(y_true, y_pred)
    if precision + recall > 0:
        # Обчислення F1-метрики
        return 2 * (precision * recall) / (precision + recall)
    else:
        return 0.0  # Повертає 0, якщо precision і recall обидва 0

# Перевірка F1-метрики за допомогою assert
# assert my_f1_score(df.actual_label.values, df.predicted_RF.values) ==
f1_score(df.actual_label.values, df.predicted_RF.values), 'my_f1_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_f1_score
failed on LR'


# Виведення результатів
```

```
print('F1 RF: ', baginskiy_f1_score(
   df.actual_label.values, df.predicted_RF.values))
print('F1 LR: ',  baginskiy_f1_score(
   df.actual_label.values, df.predicted_LR.values))

print('F1 RF original: ', f1_score(
   df.actual_label.values, df.predicted_RF.values))
print('F1 LR original: ',  f1_score(
   df.actual_label.values, df.predicted_LR.values))

print('')


for threshold in thresholds:
   print(f'scores with threshold = {threshold}')

   # Генерація прогнозованих міток на основі поточного порогу
   predicted_RF = (df.model_RF >= threshold).astype('int')

   # Виведення показників
   print('Accuracy RF: %.3f' %
         (baginskiy_accuracy_score(df.actual_label.values, predicted_RF)))
   print('Recall RF: %.3f' %
         (baginskiy_recall_score(df.actual_label.values, predicted_RF)))
   print('Precision RF: %.3f' %
         (baginskiy_precision_score(df.actual_label.values, predicted_RF)))
   print('F1 RF: %.3f' % (baginskiy_f1_score(
       df.actual_label.values, predicted_RF)))
   print('')


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)

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()
```

Завдання 2.6. 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:

```
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

# Завантаження даних
data = np.loadtxt('data_multivar_nb.txt', delimiter=',')
```

```
# Розбиття на ознаки і мітки
X = data[:, :-1]  # ознаки
y = data[:, -1]   # мітки

# Розділення на навчальну та тестову вибірки
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)
# Створення та навчання моделі SVM
svm_model = SVC()
svm_model.fit(X_train, y_train)

# Прогнозування на тестових даних
y_pred_svm = svm_model.predict(X_test)

# Розрахунок показників для SVM
accuracy_svm = accuracy_score(y_test, y_pred_svm)
precision_svm = precision_score(y_test, y_pred_svm, average='macro')
recall_svm = recall_score(y_test, y_pred_svm, average='macro')
f1_svm = f1_score(y_test, y_pred_svm, average='macro')

print(f"SVM - Accuracy: {accuracy_svm:.3f}, Precision: {precision_svm:.3f},
Recall: {recall_svm:.3f}, F1 Score: {f1_svm:.3f}")
# Створення та навчання моделі наївного байєсівського класифікатора
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)

# Прогнозування на тестових даних
y_pred_nb = nb_model.predict(X_test)

# Розрахунок показників для наївного Байєса
accuracy_nb = accuracy_score(y_test, y_pred_nb)
precision_nb = precision_score(y_test, y_pred_nb, average='macro')
recall_nb = recall_score(y_test, y_pred_nb, average='macro')
f1_nb = f1_score(y_test, y_pred_nb, average='macro')

print(f"Naive Bayes - Accuracy: {accuracy_nb:.3f}, Precision:
{precision_nb:.3f}, Recall: {recall_nb:.3f}, F1 Score: {f1_nb:.3f}")
```

Рис.18. Результат виконання

Порівняння результатів:

Обидві моделі — машина опорних векторів (SVM) та наївний байєсівський класифікатор — показали ідентичні результати за всіма основними метриками:

- **Точність (Accuracy)**: 0.992
- **Точність (Precision)**: 0.990
- **Повнота (Recall)**: 0.992
- **F1-оцінка (F1 Score)**: 0.991

Це свідчить про те, що обидві моделі працюють майже ідеально для даного набору даних, класифікуючи приклади з мінімальною кількістю помилок.

Висновки: так як дві моделі показали однаково високі результати, вибір між ними буде залежати від ваших потреб. **SVM -** це потужна модель, для складних даних, особливо коли класифікаційні межі нелінійні, але як мінус вона більш ресурсомістка, особливо на великих наборах даних, і час навчання може бути довшим. **Наївний Байєс -** це більш простіша та швидкіша модель, яка краще працює на текстових даних або коли ознаки незалежні. Його основна перевага — швидкість і ефективність на невеликих наборах даних.