Лабораторна робота №1

**з дисципліни Основи штучного інтелекту**
**студента групи ЗІПЗк-22-1**
**Перехватова Алевтина Олександрівна**
**дата виконання: 20.10.2023**

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

## Виконання роботи

### Завдання 2

**Дія1:**

```
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.1.1– Код програми

```
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 – Реакція програми на дію

Висновок: L1-нормалізація і L2-нормалізація є двома різними методами нормалізації векторів, які використовуються в машинному навчанні і статистиці. L1-нормалізація використовує метод найменших абсолютних відхилень (Least Absolute Deviations), що забезпечує рівність 1 суми абсолютних значень в кожному ряду. L2-нормалізація використовує метод найменших квадратів, що забезпечує рівність 1 суми квадратів 4 значень. Взагалі, техніка L1-нормалізації вважається більш надійною по порівняно з L2-нормалізацією, оскільки вона менш чутлива до викидів.

**Завдання 2.1.5**

**Дія1:**

```
import numpy as np
from sklearn import preprocessing
# Надання позначок вхідних даних
Input_labels = ['red', 'ыack', 'red', 'green', 'ыlack',
'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', 'ыack']
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.2.1– Код програми

```
Label mapping:
green --> 0
red --> 1
white --> 2
yellow --> 3
ыack --> 4
ыlack --> 5

Labels = ['green', 'red', 'ыack']
Encoded values = [0, 1, 4]

Encoded values = [3, 0, 4, 1]
Decoded labels = ['yellow', 'green', 'ыack', 'red']
```

Рис.2.2 – Реакція програми на дію

**Завдання 2.2**

**Дія1:**

```
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)
```

Рис.3.1– Код програми

```
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]]
```

Рис.3.2 – Реакція програми на дію

**Завдання 2.3**

**Дія1:**

```
import numpy as np
from sklearn import linear_model
!pip install matplotlib-venn
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.4.1– Код програми

```
Requirement already satisfied: matplotlib-venn in /usr/loc
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from m
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from m
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (fro
```

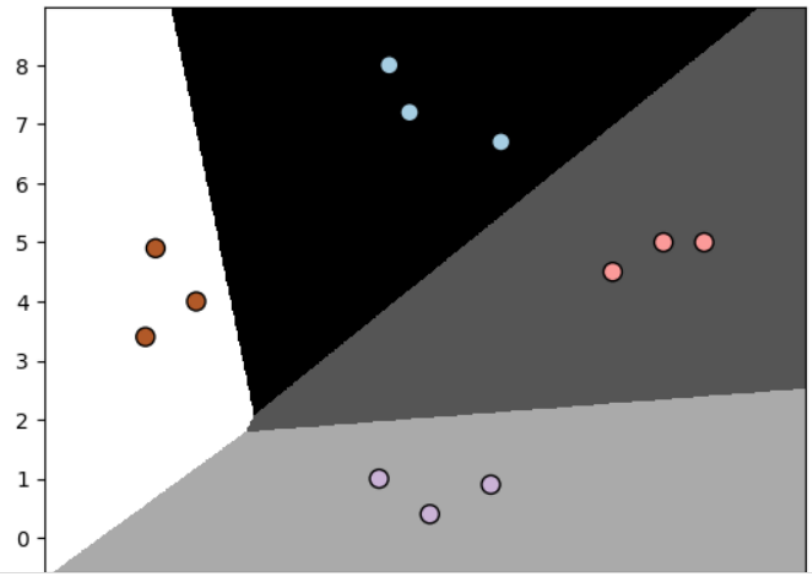



Рис.4.2 – Реакція програми на дію

**Завдання 2.4**

**Дія1:**

```
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.5.1– Код програми

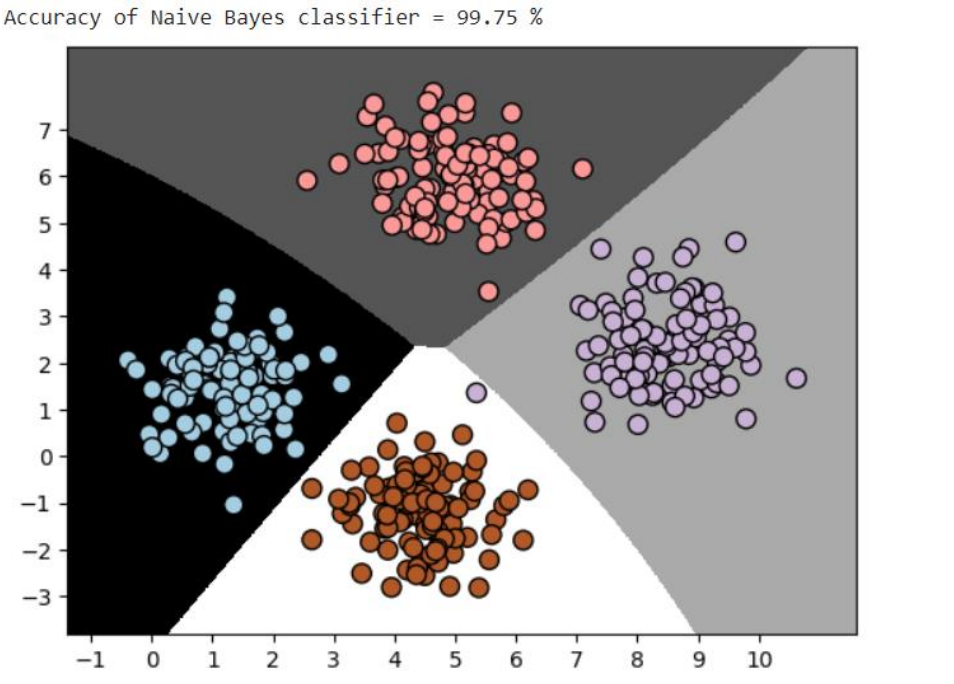



Рис.5.2 – Реакція програми на дію

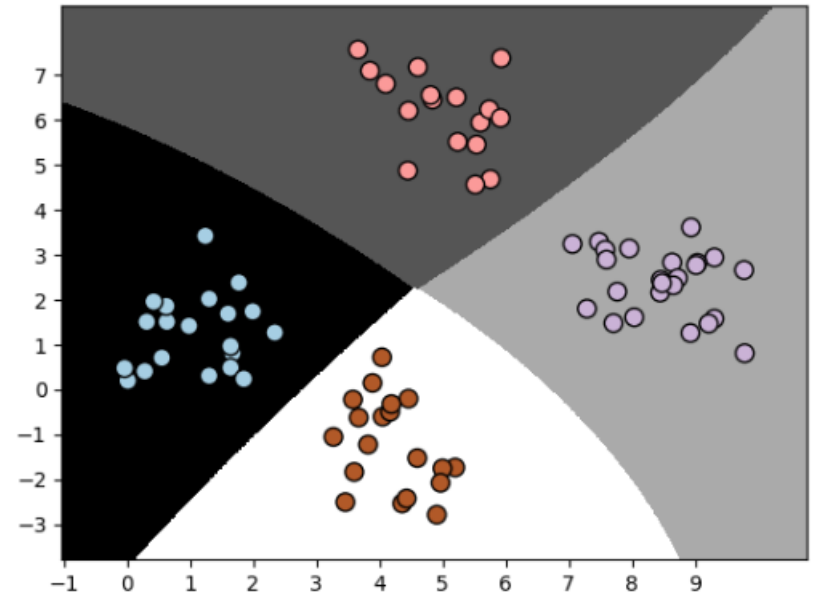
**Дія1:**

```
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)
# 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)
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.6.1– Код програми

```
Accuracy of the new classifier = 100.0 %
```




```
Accuracy: 99.75%
Precision: 99.76%
Recall: 99.75%
F1: 99.75%
```

Рис.6.2 – Реакція програми на дію

confusion_matrix

```python
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')
print(df.head())

from sklearn.metrics import confusion_matrix
rf_cm = confusion_matrix(df['actual_label'], df['predicted_RF'])
def perehvatova_confusion_matrix(y_true, y_pred):
   TP = sum((y_true.astype(int) == 1) & (y_pred.astype(int) == 1))
   FN = sum((y_true == 1) & (y_pred == 0))
   FP = sum((y_true == 0) & (y_pred == 1))
   TN = sum((y_true.astype(int) == 0) & (y_pred.astype(int) == 0))

   return np.array([[TP, FP], [FN, TN]], dtype=int)



print(rf_cm)
print(custom_cm)
```

Рис.7.1– Код програми

```
   actual_label  model_RF  model_LR  predicted_RF  predicted_LR
0             1  0.639816  0.531904             1             1
1             0  0.490993  0.414496             0             0
2             1  0.623815  0.569883             1             1
3             1  0.506616  0.443674             1             0
4             0  0.418302  0.369532             0             0
[[5519 2360]
 [2832 5047]]
[[5047 2360]
 [2832 5519]]
```

Рис.7.2 – Реакція програми на дію

accuracy_score

```python
from sklearn.metrics import accuracy_score

def find_conf_matrix_values(y_true, y_pred):
    TP, FN, FP, TN = 0, 0, 0, 0
    for i in range(len(y_true)):
        if y_true[i] == 1 and y_pred[i] == 1:
            TP += 1
        elif y_true[i] == 1 and y_pred[i] == 0:
            FN += 1
        elif y_true[i] == 0 and y_pred[i] == 1:
            FP += 1
        else:
            TN += 1
    return TP, FN, FP, TN

def perehv_accuracy_score(y_true, y_pred):
    TP, FN, FP, TN = find_conf_matrix_values(y_true, y_pred)
    accuracy = (TP + TN) / (TP + TN + FP + FN)
    return accuracy

accuracy_RF = accuracy_score(df.actual_label.values, df.predicted_RF.values)

assert perehv_accuracy_score(df.actual_label.values, df.predicted_RF.values) == accuracy_RF
assert perehv_accuracy_score(df.actual_label.values, df.predicted_LR.values) == accuracy_score(df.actual_label.values, df.predicted_LR.values)

print('Accuracy RF: %.3f' % (accuracy_RF))
print('Accuracy LR: %.3f' % (accuracy_score(df.actual_label.values, df.predicted_LR.values)))
```

Рис.8.1– Код програми

```
[2832 5519]]
Accuracy RF: 0.671
Accuracy LR: 0.616
```

Рис.8.2 – Реакція програми на дію

## recall_score

```
from sklearn.metrics import recall_score

def find_conf_matrix_values(y_true, y_pred):
    TP, FN, FP, TN = 0, 0, 0, 0
    for i in range(len(y_true)):
        if y_true[i] == 1 and y_pred[i] == 1:
            TP += 1
        elif y_true[i] == 1 and y_pred[i] == 0:
            FN += 1
        elif y_true[i] == 0 and y_pred[i] == 1:
            FP += 1
        else:
            TN += 1
    return TP, FN, FP, TN

def perehv_recall_score(y_true, y_pred):
    TP, FN, FP, TN = find_conf_matrix_values(y_true, y_pred)
    recall = TP / (TP + FN)  # Recall calculation
    return recall

recall_RF = recall_score(df.actual_label.values, df.predicted_RF.values)

assert perehv_recall_score(df.actual_label.values, df.predicted_RF.values) == recall_RF, 'perehv_recall_score failed on RF'
assert perehv_recall_score(df.actual_label.values, df.predicted_LR.values) == recall_score(df.actual_label.values, df.predicted_LR.values), 'perehv_recall_score failed on LR'

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

Рис.9.1– Код програми

```
Recall RF: 0.641
Recall LR: 0.543
```

Рис.9.2 – Реакція програми на дію

## precision_score

```
from sklearn.metrics import precision_score

def find_conf_matrix_values(y_true, y_pred):
    TP, FN, FP, TN = 0, 0, 0, 0
    for i in range(len(y_true)):
        if y_true[i] == 1 and y_pred[i] == 1:
            TP += 1
        elif y_true[i] == 1 and y_pred[i] == 0:
            FN += 1
        elif y_true[i] == 0 and y_pred[i] == 1:
            FP += 1
        else:
            TN += 1
    return TP, FN, FP, TN

def perehv_precision_score(y_true, y_pred):
    TP, FN, FP, TN = find_conf_matrix_values(y_true, y_pred)
    precision = TP / (TP + FP)  # Precision calculation
    return precision

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

assert perehv_precision_score(df.actual_label.values, df.predicted_RF.values) == precision_RF, 'perehv_precision_score failed on RF'
assert perehv_precision_score(df.actual_label.values, df.predicted_LR.values) == precision_score(df.actual_label.values, df.predicted_LR.values), 'perehv_precision_score failed on LR'

print('Precision RF: %.3f' % precision_RF)
print('Precision LR: %.3f' % perehv_precision_score(df.actual_label.values, df.predicted_LR.values))
```

Рис.10.1– Код програми

```
Precision RF: 0.681
Precision LR: 0.636
```

Рис.10.2 – Реакція програми на дію

f1_score

```python
from sklearn.metrics import f1_score

def perehv_f1_score(y_true, y_pred):
    recall = perehv_recall_score(y_true, y_pred)
    precision = perehv_precision_score(y_true, y_pred)
    f1 = 2 * (precision * recall) / (precision + recall)  # F1 score calculation
    return f1

f1_RF = f1_score(df.actual_label.values, df.predicted_RF.values)

assert perehv_f1_score(df.actual_label.values, df.predicted_RF.values) == f1_RF, 'perehv_f1_score failed on RF'
assert perehv_f1_score(df.actual_label.values, df.predicted_LR.values) == f1_score(df.actual_label.values, df.predicted_LR.values), 'perehv_f1_score failed on LR'

print('F1 RF: %.3f' % f1_RF)
print('F1 LR: %.3f' % perehv_f1_score(df.actual_label.values, df.predicted_LR.values))

# Scores with threshold = 0.5
print('Accuracy RF: %.3f' % perehv_accuracy_score(df.actual_label.values, df.predicted_RF.values))
print('Recall RF: %.3f' % perehv_recall_score(df.actual_label.values, df.predicted_RF.values))
print('Precision RF: %.3f' % perehv_precision_score(df.actual_label.values, df.predicted_RF.values))
print('F1 RF: %.3f' % perehv_f1_score(df.actual_label.values, df.predicted_RF.values))
print('')

# Scores with threshold = 0.25
print('Accuracy RF: %.3f' % perehv_accuracy_score(df.actual_label.values, (df.model_RF >= 0.25).astype('int').values))
print('Recall RF: %.3f' % perehv_recall_score(df.actual_label.values, (df.model_RF >= 0.25).astype('int').values))
print('Precision RF: %.3f' % perehv_precision_score(df.actual_label.values, (df.model_RF >= 0.25).astype('int').values))
print('F1 RF: %.3f' % perehv_f1_score(df.actual_label.values, (df.model_RF >= 0.25).astype('int').values))
```

Рис.11.1– Код програми

```
F1 RF: 0.660
F1 LR: 0.586
Accuracy RF: 0.671
Recall RF: 0.641
Precision RF: 0.681
F1 RF: 0.660

Accuracy RF: 0.502
Recall RF: 1.000
Precision RF: 0.501
F1 RF: 0.668
```

Рис.11.2 – Реакція програми на дію

**Висновки:**

- Коли використовується поріг 0.5:
  - Розраховуються та виводяться точність, покриття, точність та F1-оцінка для класифікатора Random Forest.
- При використанні меншого порогу 0.25:
  - Генеруються нові передбачення на основі порогу.
  - Розраховуються та виводяться точність, покриття, точність та F1-оцінка для класифікатора Random Forest з новими передбаченнями.

roc_curve

```
from sklearn.metrics import roc_curve
import matplotlib.pyplot as plt

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()
```

Рис.12.1– Код програми

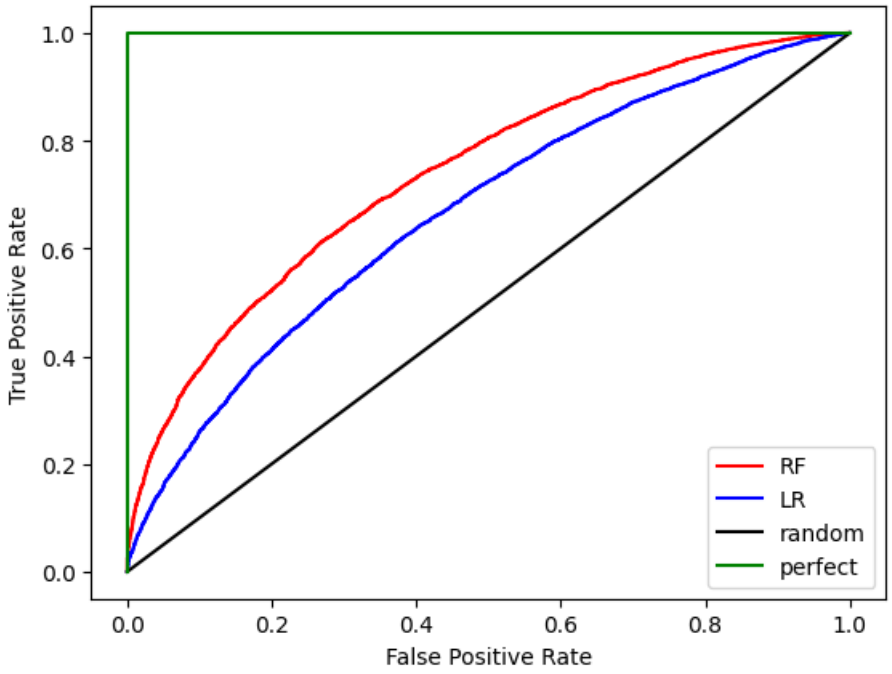



Рис.12.2 – Реакція програми на дію

roc_auc_score

```
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

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()
```

Рис.13.1– Код програми

```
AUC RF: 0.738
AUC LR: 0.666
```

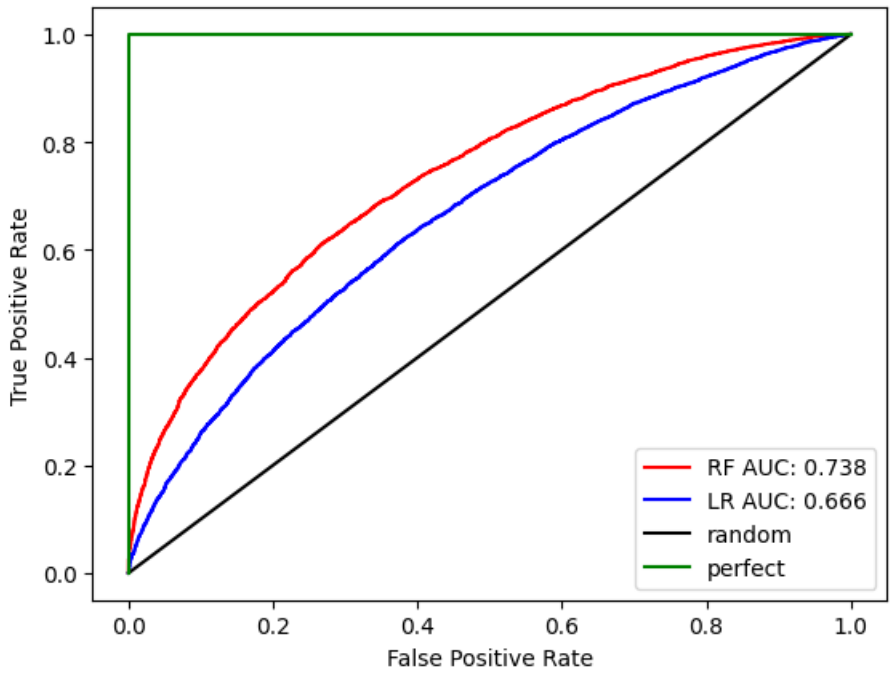



Рис.13.2 – Реакція програми на дію

**Завдання 2.6**

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

df = pd.read_csv('data_multivar_nb.txt', names=['x1', 'x2', 'y'])

X = df[['x1', 'x2']].to_numpy()
y = df['y'].to_numpy()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)

svm = SVC()
svm.fit(X_train, y_train)
y_pred = svm.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')


print(f'Accuracy: {accuracy:.3f}')
print(f'Precision: {precision:.3f}')
print(f'Recall: {recall:.3f}')
print(f'F1: {f1:.3f}')
```

Рис.14.1– Код програми

```
Accuracy: 0.990
Precision: 0.991
Recall: 0.990
F1: 0.990
```

Рис.14.2 – Реакція програми на дію

**Висновки**:

Показники якості класифікації SVM і наївного байєсівського класифікатора для цього набору даних практично однакові. Однак, SVM трохи перевершує наївний байєсівський класифікатор за точністю та F-мірою.

Крім того, SVM є більш гнучкою моделлю, ніж наївний байєсівський класифікатор. SVM може адаптуватися до більш складних розподілів даних, ніж наївний байєсівський класифікатор, який припускає, що класи розподілені нормально.

Загалом, SVM є більш кращою моделлю класифікації, ніж наївний байєсівський класифікатор. SVM має більш високі показники якості класифікації і є більш гнучкою моделлю.