

## ЛАБОРАТОРНА РОБОТА №1

### Тема: ПОПЕРЕДНЯ ОБРОБКА ТА КОНТРОЛЬОВАНА КЛАСИФІКАЦІЯ ДАНИХ

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

#### Посилання на Git:

#### Хід роботи:

##### Завдання 2.1. Попередня обробка даних

#### Лістинг програми:

```
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)
```

					ДУ «Житомирська політехніка».20.121.11.000 – Лр 1			
Змн.	Арк.	№ докум.	Підпис	Дата	Звіт з лабораторної роботи  ФІКТ Гр. ІПЗ-20-1[1]			
Розроб.		Зима Д.А.						
Перевір.		Голенко М.Ю.						
Керівник								
Н. контр.								
Зав. каф.								
					Літ.	Арк.	Аркушів	
						1	12	

### Результат виконання програми:

```
C:\Users\1\AppData\Local\Programs\Python\Python310\python.exe

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.2.1.1** Результат виконання програми

### Різниця L1-нормалізації від L2-нормалізації:

На цих двох рівнях нормалізації використовують різні методи. Наприклад, на рівні L1 застосовується метод найменших абсолютних відхилень, який гарантує, що сума абсолютних значень в кожному рядку становитиме 1. З іншого боку, на рівні L2 використовується метод найменших квадратів, який забезпечує рівність 1 суми квадратів значень. Загалом, L1-нормалізація вважається більш надійною порівняно з L2-нормалізацією, оскільки вона менше чутлива до викидів.

		Зима Д.А.			ДУ «Житомирська політехніка».20.121.11.000 – Лр1	Арк.
		Голенко М.Ю.				2
Змн.	Арк.	№ докум.	Підпис	Дата		

### Лістинг програми:

```
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 програми:

```
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.2.1.2** Результат виконання програми

		Зима Д.А.			ДУ «Житомирська політехніка».20.121.11.000 – Лр1	Арк.
		Голенко М.Ю.				3
Змн.	Арк.	№ докум.	Підпис	Дата		

## Завдання 2.2. Попередня обробка нових даних

№	Значення змінної input_data												Поріг бінаризації
11.	-5.3	-8.9	3.0	2.9	5.1	-3.3	3.1	-2.8	-3.2	2.2	-1.4	5.1	2.0

### Лістинг програми:

```
import numpy as np
from sklearn import preprocessing

input_data = np.array([[ -5.3, -8.9, 3.0],
                       [2.9, 5.1, -3.3],
                       [3.1, -2.8, -3.2],
                       [2.2, -1.4, 5.1]])

# Бінаризація даних
data_binarized = preprocessing.Binarizer(threshold=2.0).transform(input_data)
print("\n Binarized data:\n", data_binarized)

# 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)
```

### Результат виконання програми:

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

AFTER:
Mean = [-2.77555756e-17 -2.42861287e-17  0.00000000e+00]
Std deviation = [1. 1. 1.]

Min max scaled data:
[[0.         0.         0.75        ]
 [0.97619048  1.         0.         ]
 [1.         0.43571429  0.01190476]
 [0.89285714  0.53571429  1.         ]]

l1 normalized data:
[[-0.30813953 -0.51744186  0.1744186 ]
 [ 0.25663717  0.45132743 -0.2920354 ]
 [ 0.34065934 -0.30769231 -0.35164835]
 [ 0.25287356 -0.16091954  0.5862069 ]]

l2 normalized data:
[[-0.49145755 -0.82527777  0.27818352]
 [ 0.43082507  0.75765788 -0.49024922]
 [ 0.58911518 -0.53210404 -0.6081189 ]
 [ 0.38407812 -0.24441335  0.89036291]]
```

Рис.2.2.1 Результат виконання програми

		Зима Д.А.			ДУ «Житомирська політехніка».20.121.11.000 – Лр 1	Арк.
		Голенко М.Ю.				4
Змн.	Арк.	№ докум.	Підпис	Дата		

### Завдання 2.3. Класифікація логістичною регресією або логістичний класифікатор

#### Лістинг програми:

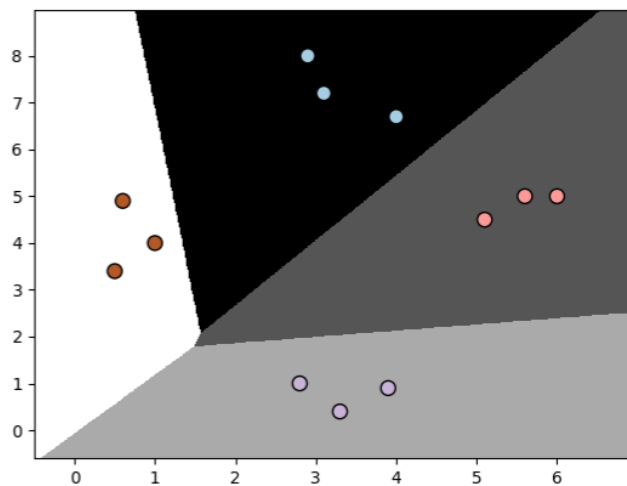
```
import numpy as np
from sklearn import linear_model
import matplotlib.pyplot as plt
from utilities import visualize_classifier

# 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])

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

# Тренування класифікатора
classifier.fit(X, y)
visualize_classifier(classifier, X, y)
```

#### Результат виконання програми:



**Рис.2.3.1** Результат виконання програми

		Зима Д.А.			ДУ «Житомирська політехніка».20.121.11.000 – Лр1	Арк.
		Голенко М.Ю.				5
Змн.	Арк.	№ докум.	Підпис	Дата		

## Завдання 2.4. Класифікація наївним байєсовським класифікатором

### Лістинг програми:

```
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)

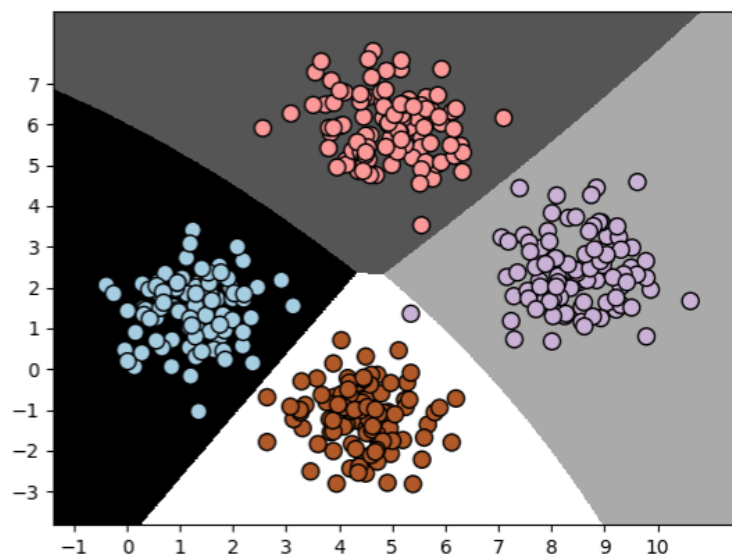
# Прогнозування значень для тренувальних даних
y_pred = classifier.predict(X)

# 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)
```

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

### Результат виконання програми:



**Рис.2.4.1** Результат виконання програми

Оскільки попередній метод обчислення є не надійним, ми виконаємо перехресну перевірку та зробимо ще один прогін. Додамо до минулого коду наступне:

		Зима Д.А.			ДУ «Житомирська політехніка».20.121.11.000 – Лр1	Арк.
		Голенко М.Ю.				6
Змн.	Арк.	№ докум.	Підпис	Дата		

### Лістинг програми:

```
# 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)) + "%")
```

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

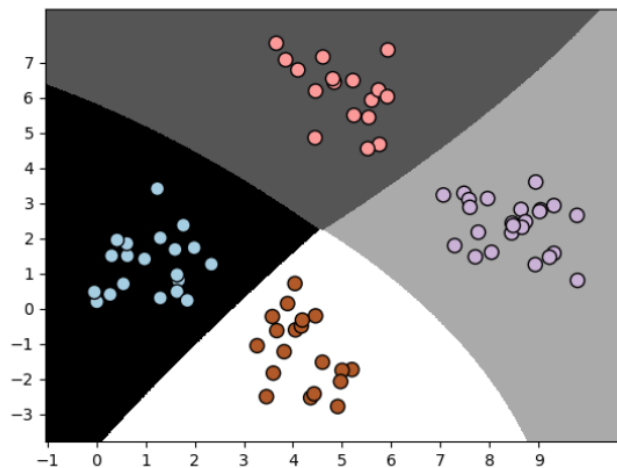
**Accuracy: 99.75%**

**Precision: 99.76%**

**Recall: 99.75%**

**F1: 99.75%**

### Результат виконання програми:



**Рис.2.4.2** Результат виконання програми

Порівнюючи два зображення ми бачимо, що результат другої класифікації є більш коректним й точним, також має менше міток.

		Зима Д.А.			ДУ «Житомирська політехніка».20.121.11.000 – Пр1	Арк.
		Голенко М.Ю.				7
Змн.	Арк.	№ докум.	Підпис	Дата		

## Завдання 2.5. Вивчити метрики класифікації

### Лістинг програми:

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

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()

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()

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

def boyko_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 boyko_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 boyko_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 boyko_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:', boyko_find_TP(df.actual_label.values, df.predicted_RF.values))
print('FN:', boyko_find_FN(df.actual_label.values, df.predicted_RF.values))
print('FP:', boyko_find_FP(df.actual_label.values, df.predicted_RF.values))
print('TN:', boyko_find_TN(df.actual_label.values, df.predicted_RF.values))
def find_conf_matrix_values(y_true, y_pred):
    # calculate TP, FN, FP, TN
    TP = boyko_find_TP(y_true, y_pred)
    FN = boyko_find_FN(y_true, y_pred)
    FP = boyko_find_FP(y_true, y_pred)
    TN = boyko_find_TN(y_true, y_pred)
    return TP, FN, FP, TN
def boyko_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]])

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

assert np.array_equal(boyko_confusion_matrix(df.actual_label.values, df.predicted_RF.values), confusion_matrix(df.actual_label.values, df.predicted_RF.values)), 'boyko_confusion_matrix() is not correct for RF'
assert np.array_equal(boyko_confusion_matrix(df.actual_label.values, df.predicted_LR.values), confusion_matrix(df.actual_label.values, df.predicted_LR.values)), 'boyko_confusion_matrix() is not correct for LR'

from sklearn.metrics import accuracy_score
accuracy_score(df.actual_label.values, df.predicted_RF.values)
def boyko_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 boyko_accuracy_score(df.actual_label.values, df.predicted_RF.values) == accuracy_score(df.actual_label.values, df.predicted_RF.values), 'boyko_accuracy_score failed on RF'
assert boyko_accuracy_score(df.actual_label.values, df.predicted_LR.values) == accuracy_score(df.actual_label.values, df.predicted_LR.values), 'boyko_accuracy_score failed on LR'
print('Accuracy LR: %.3f%(boyko_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 boyko_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 boyko_recall_score(df.actual_label.values, df.predicted_RF.values) == recall_score(df.actual_label.values, df.predicted_RF.values), 'my_accuracy_score failed on RF'
assert boyko_recall_score(df.actual_label.values, df.predicted_LR.values) == recall_score(df.actual_label.values, df.predicted_LR.values), 'my_accuracy_score failed on LR'
print('Recall RF: %.3f%(boyko_recall_score(df.actual_label.values, df.predicted_RF.values)))
print('Recall LR: %.3f%(boyko_recall_score(df.actual_label.values, df.predicted_LR.values)))
```

		Зима Д.А.			ДУ «Житомирська політехніка».20.121.11.000 – Лр1	Арк.
		Голенко М.Ю.				8
Змн.	Арк.	№ докум.	Підпис	Дата		



```

from sklearn.metrics import precision_score
precision_score(df.actual_label.values, df.predicted_RF.values)
def boyko_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 boyko_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 boyko_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'%(boyko_precision_score(df.actual_label.values, df.predicted_RF.values)))
print('Precision LR: %.3f'%(boyko_precision_score(df.actual_label.values, df.predicted_LR.values)))

from sklearn.metrics import f1_score
f1_score(df.actual_label.values, df.predicted_RF.values)
def boyko_f1_score(y_true, y_pred):
    # calculates the F1 score
    recall = boyko_recall_score(y_true, y_pred)
    precision = boyko_precision_score(y_true, y_pred)
    return 2*(precision*recall)/(precision+recall)

assert boyko_f1_score(df.actual_label.values, df.predicted_RF.values) == f1_score(df.actual_label.values, df.predicted_RF.values),
'boyko_accuracy_score failed on RF'
assert boyko_f1_score(df.actual_label.values, df.predicted_LR.values) == f1_score(df.actual_label.values, df.predicted_LR.values),
'boyko_accuracy_score failed on LR'
print('F1 RF: %.3f'%(boyko_f1_score(df.actual_label.values, df.predicted_RF.values)))
print('F1 LR: %.3f'%(boyko_f1_score(df.actual_label.values, df.predicted_LR.values)))

print('scores with threshold = 0.5')
print('Accuracy RF: %.3f'%(boyko_accuracy_score(df.actual_label.values, df.predicted_RF.values)))
print('Recall RF: %.3f'%(boyko_recall_score(df.actual_label.values, df.predicted_RF.values)))
print('Precision RF: %.3f'%(boyko_precision_score(df.actual_label.values, df.predicted_RF.values)))
print('F1 RF: %.3f'%(boyko_f1_score(df.actual_label.values, df.predicted_RF.values)))
print("")
print('scores with threshold = 0.25')
print('Accuracy RF: %.3f'%(boyko_accuracy_score(df.actual_label.values, (df.model_RF >= 0.25).astype('int').values)))
print('Recall RF: %.3f'%(boyko_recall_score(df.actual_label.values, (df.model_RF >= 0.25).astype('int').values)))
print('Precision RF: %.3f'%(boyko_precision_score(df.actual_label.values, (df.model_RF >= 0.25).astype('int').values)))
print('F1 RF: %.3f'%(boyko_f1_score(df.actual_label.values, (df.model_RF >= 0.25).astype('int').values)))

```

#### Результат виконання програми:

```

TP: 5047
FN: 2832
FP: 2360
TN: 5519
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

```

Рис.2.5.1 Результат виконання програми

		Зима Д.А.			ДУ «Житомирська політехніка».20.121.11.000 – Лр1	Арк.
		Голенко М.Ю.				9
Змн.	Арк.	№ докум.	Підпис	Дата		

На основі поданих результатів для різних порогів можна зробити висновок:

**Поріг 0.5:**

Accuracy (точність) RF: 0.671 - Це означає, що модель правильно класифікувала 67.1% випадків.

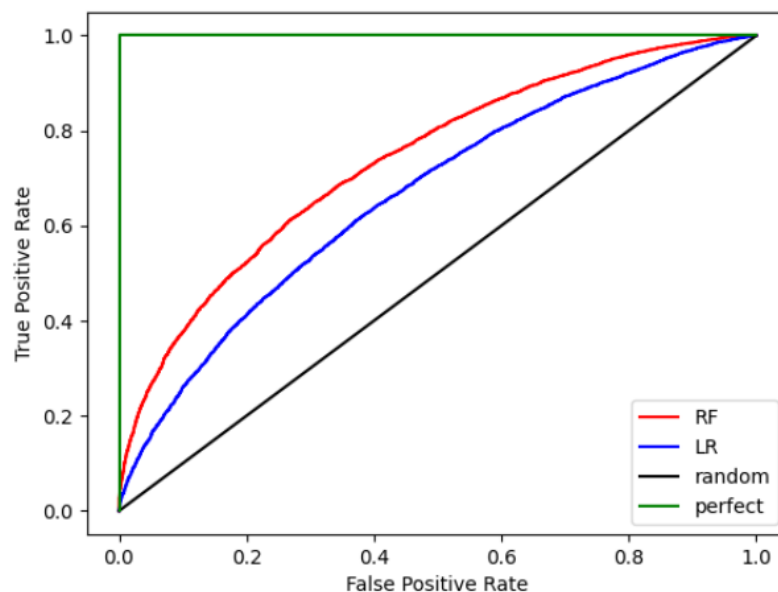
- Recall (повнота) RF: 0.641 - Модель виявила 64.1% усіх позитивних випадків.
- Precision (точність) RF: 0.681 - З 68.1% випадків, які модель відзначила як позитивні, вони дійсно були позитивними.
- F1 RF: 0.660 - F1-мера об'єднує як Recall, так і Precision в одну метрику, і вона становить середнє гармонічне між ними. У вас F1 дорівнює 0.660.

**Поріг 0.25:**

- Accuracy (точність) RF: 0.502 - Тут модель має меншу точність, вона правильно класифікувала тільки 50.2% випадків.
- Recall (повнота) RF: 1.000 - Модель виявила всі можливі позитивні випадки, що свідчить про те, що вона не пропускає жодного позитивного випадку.
- Precision (точність) RF: 0.501 - Проте, точність моделі при цьому порозі дуже низька, всього 50.1% випадків, які модель відзначила як позитивні, дійсно були позитивними.
- F1 RF: 0.668 - F1-мера знову вказує на баланс між точністю і повнотою, і в цьому випадку вона є вищою, ніж при порозі 0.5.

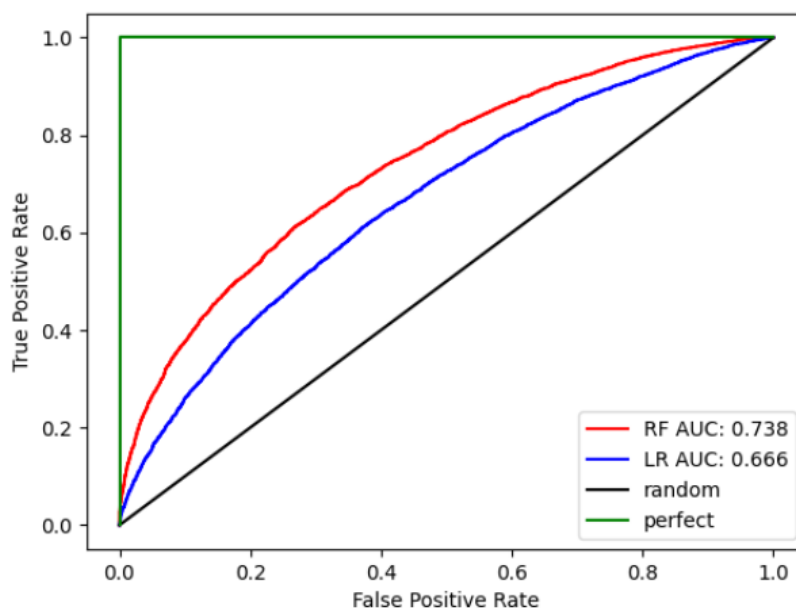
**Висновок:**

- За порогом 0.5 модель має кращу точність, але меншу повноту.
- За порогом 0.25 модель має високу повноту, але низьку точність.
- Вибір порогу залежить від конкретних вимог задачі. Якщо важливо уникнути пропусків позитивних випадків, може бути кращим вибором поріг 0.25. Якщо важлива точність класифікації, то краще залишити поріг на рівні 0.5.



**Рис.2.5.2** криві ROC для кожної моделі

		Зима Д.А.			ДУ «Житомирська політехніка».20.121.11.000 – Лр1	Арк.
		Голенко М.Ю.				10
Змн.	Арк.	№ докум.	Підпис	Дата		



**Рис.2.5.3** Додали AUC

**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 програми:

```
import numpy as np
from sklearn import svm
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 класифікатора машини опорних векторів
classifier = svm.SVC()

# 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 = svm.SVC()
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 Support Vector Machine 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)) + "%")
```

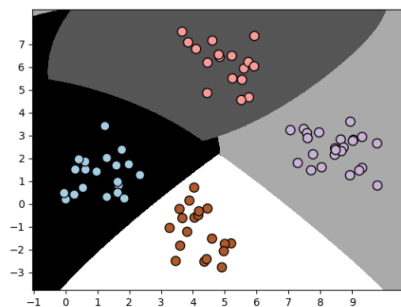
		Зима Д.А.			ДУ «Житомирська політехніка».20.121.11.000 – Лр1	Арк.
		Голенко М.Ю.				11
Змн.	Арк.	№ докум.	Підпис	Дата		

```
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 програми:

```
Accuracy of Support Vector Machine classifier = 100.0 %
Accuracy: 99.75%
Precision: 99.76%
Recall: 99.75%
F1: 99.75%
```

**Рис.2.6.1** Результат виконання програми



**Рис.2.6.2** Результат виконання програми

Порівнюючи результати моєї програми класифікації з результатами наївного баєсового класифікатора, можна відзначити, що точність в баєсового класифікатора складає 99.75%, тоді як у моєї програми точність становить 100%. З цього можна зробити висновок, що моя програма виявляється переважнішою за метод байєсового класифікатора, принаймні за цим показником.

**Висновок:** Навчився використовувати спеціалізовані бібліотеки та мову програмування Python, також дослідив попередню обробку та класифікацію даних.

		Зима Д.А.			ДУ «Житомирська політехніка».20.121.11.000 – Пр1	Арк.
		Голенко М.Ю.				12
Змн.	Арк.	№ докум.	Підпис	Дата		