

ЛАБОРАТОРНА РОБОТА № 1

ПОПЕРЕДНЯ ОБРОБКА ТА КОНТРОЛЬОВАНА КЛАСИФІКАЦІЯ ДАНИХ

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

GitHub: https://github.com/MidorayUn/AI_CS_20_1/tree/main

Хід роботи:

Нормалізація даних:

```
# 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)
```



```
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.Результат нормалізації даних.

L1-нормалізація та L2-нормалізація - це два різні методи нормалізації даних, які використовуються для створення вхідних даних з одиничною нормою (довжиною) вектора.

L1-нормалізація, також відома як "манхеттенська нормалізація", виконує обчислення суми абсолютних значень кожного рядка у матриці даних і потім ділить кожне значення у рядку на цю суму.

L2-нормалізація, також відома як "Євклідова нормалізація", обчислює

					ДУ «Житомирська політехніка».22.122.5.000 – Лр1			
Змн.	Арк.	№ докум.	Підпис	Дата	Звіт з лабораторної роботи	Лім.	Арк.	Аркушів
Розроб.		Ищук О.Ю.						
Перевір.		Голенко М.Ю.					1	
Керівник						ФІКТ Гр. КН-20-1(1)		
Н. контр.								
Зав. каф.								

Євклідову норму (квадратний корінь з суми квадратів значень) кожного рядка у матриці даних і потім ділить кожне значення у рядку на цю норму. Це призводить до різних результатів для однакових вхідних даних, і в даному випадку абсолютні значення при L2-нормалізації є більшими.

Завдання 1

```
[ ] 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)
```

▼ LabelEncoder
LabelEncoder()

Рис.2. Результат кодувальника.

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

Label mapping:
black --> 0
green --> 1
red --> 2
white --> 3
yellow --> 4
black --> 5

[ ] # 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 ) )

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

Рис.3. Результат відображення та перетворення міток.

		Ицук О.Ю..			ДУ «Житомирська політехніка».22.122.5.000 – Лр1	Арк.
		Голенко М.Ю.				2
Змн.	Арк.	№ докум.	Підпис	Дата		

```

# 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) )

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

```

Рис.4. Результат декодування.

Завдання 2

```

import numpy as np
from sklearn import preprocessing

[3] #task 1
input_data = np.array([[-1.3,3.9,4.5],
                        [-5.3, -4.2, -1.3],
                        [5.2, -6.5, -1.1],
                        [-5.2, 2.6, 2.2]])

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

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

```

Рис.5. Результат зміни початкових даних та їх бінаризації.

```

+ Код + Текст
[5] # Виведення середнього значення та стандартного відхилення
print("\nBEFORE: ")
print("Mean =", input_data.mean(axis=0))
print("Std deviation =", input_data.std(axis=0))

BEFORE:
Mean = [-1.65 -1.05 1.075]
Std deviation = [4.27112397 4.40028408 2.4169971 ]

[6] #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))

AFTER:
Mean = [-2.77555756e-17 5.55111512e-17 -4.16333634e-17]
Std deviation = [1. 1. 1.]

```

Рис.6. Результат виключення середнього.

```

[7] # 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)

Min max scaled data:
[[0.38095238 1. 1. ]
 [0. 0.22115385 0. ]
 [1. 0. 0.03448276]
 [0.00952381 0.875 0.60344828]]

[8] # 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)

l1 normalized data:
[[-0.13402062 0.40206186 0.46391753]
 [-0.49074074 -0.38888889 -0.12037037]
 [ 0.40625 -0.5078125 -0.0859375 ]
 [-0.52 0.26 0.22 ]]

l2 normalized data:
[[-0.21328678 0.63986035 0.7383004 ]
 [-0.76965323 -0.60991388 -0.18878287]
 [ 0.61931099 -0.77413873 -0.13100809]
 [-0.83653629 0.41826814 0.3539192 ]]

```

Рис.7. Результат масштабування та нормалізації даних.

Завдання 3

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

[6] # 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])

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

[8] # Тренування класифікатора
    classifier.fit(x, y)
```

▼ LogisticRegression
LogisticRegression(C=1, solver='liblinear')

Рис.8. Результат створення та тренування логістичного класифікатора.

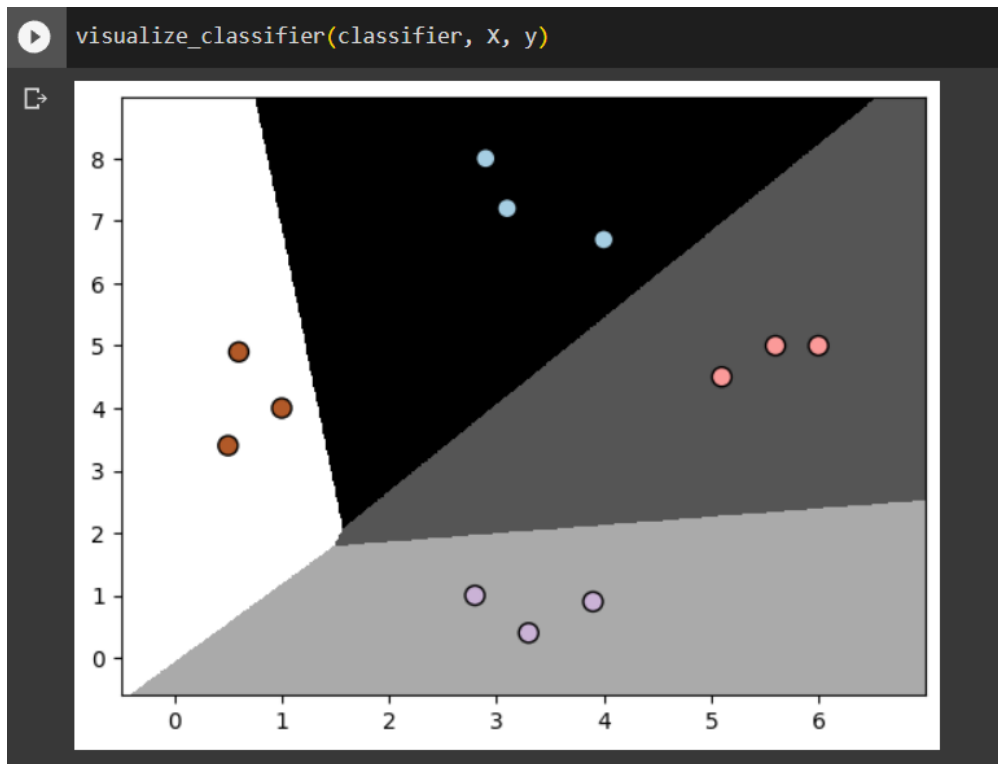


Рис.9. Візуалізація роботи класифікатора.

Завдання 4

```
[1] 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'

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

[4] # Створення наївного байєсовського класифікатора
classifier = GaussianNB()

[5] # Тренування класифікатора
classifier.fit(X, y)
```

▼ GaussianNB
GaussianNB()

Рис.10. Створення та тренування класифікатора.

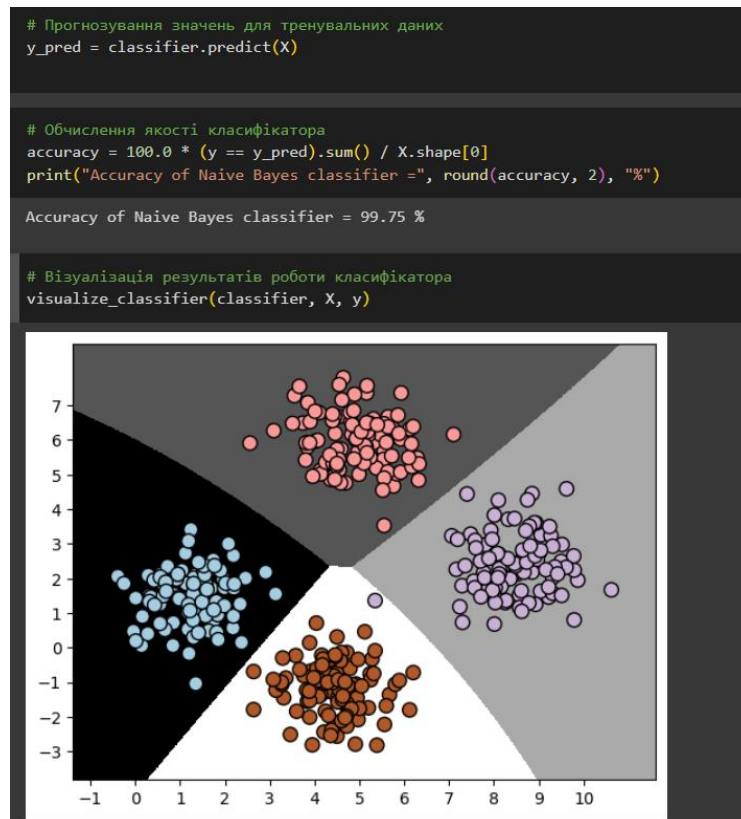


Рис.11. Прогнозування значень, обчислення якості та візуалізація результатів створеного класифікатора.

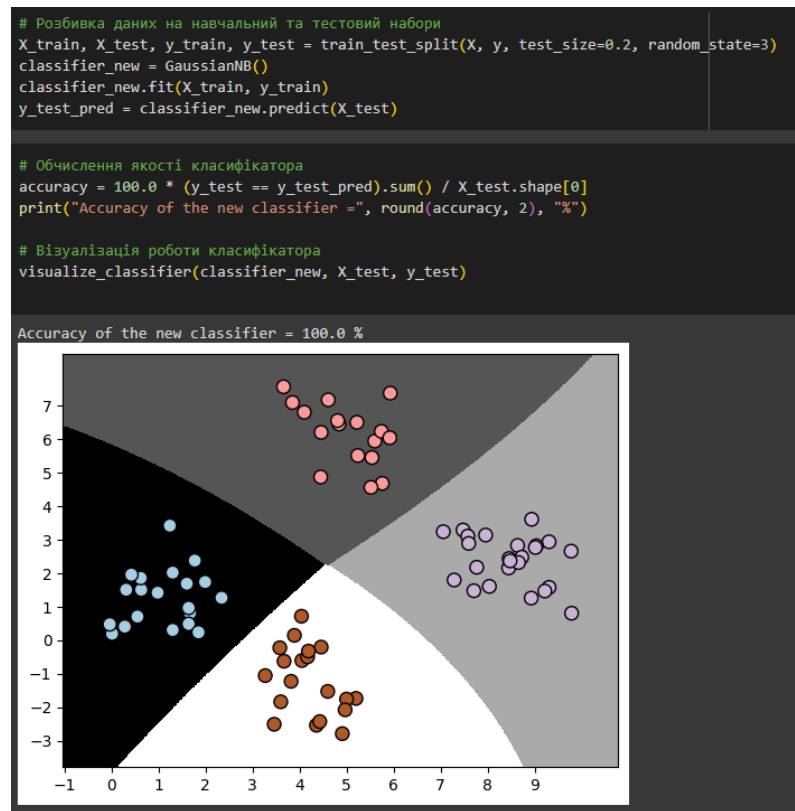


Рис.12. Результат розбивки набору даних, обчислення якості та візуалізація результатів роботи створеного класифікатора.

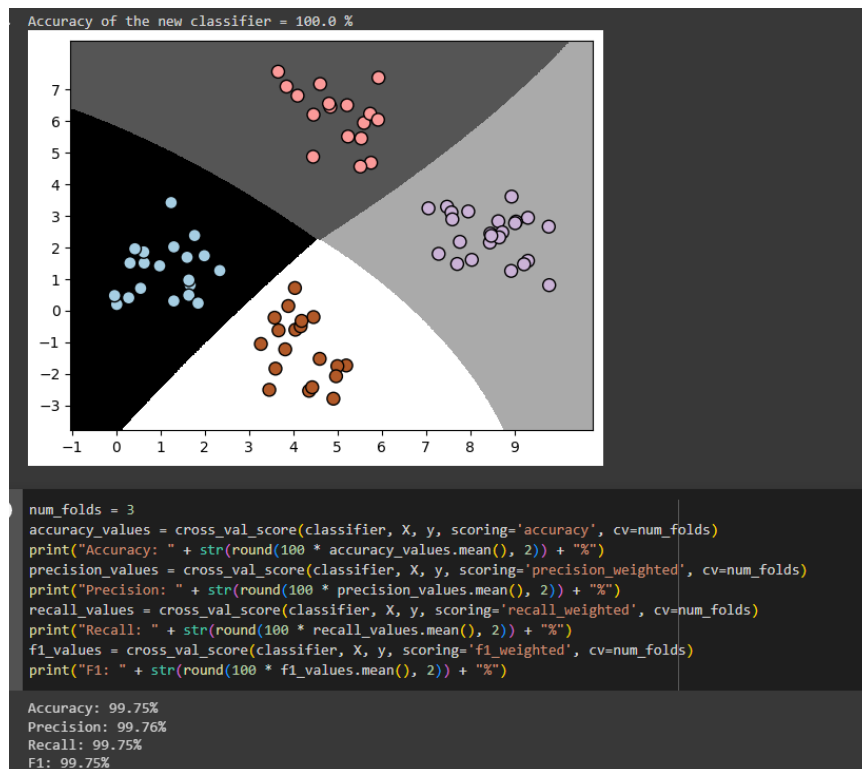


Рис.13. Результат класифікатора при повторному прогоні.

		Іщук О.Ю..			ДУ «Житомирська політехніка».22.122.5.000 – Лр1	Арк.
		Голенко М.Ю.				7
Змн.	Арк.	№ докум.	Підпис	Дата		

При подальшому виконанні коду ми спостерігаємо отримання того ж результату. Це стає можливим завдяки використанню випадкового розподілу даних між навчальним та тестовим наборами з використанням функції `train_test_split` з фіксованим параметром `random_state` (3).

Цей параметр задає початковий стан генератора випадкових чисел, і якщо він залишається незмінним між кількома виконаннями коду, то поділ даних залишається сталим. В результаті модель навчається і тестується на тих самих наборах даних, що призводить до однакового результату при кожному запуску.

Завдання 5

```
import pandas as pd
df = pd.read_csv('data_metrics.csv')
df.head()
```

	actual_label	model_RF	model_LR
0	1	0.639816	0.531904
1	0	0.490993	0.414496
2	1	0.623815	0.569883
3	1	0.506616	0.443674
4	0	0.418302	0.369532

```
[5] 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()
```

	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

Рис.14. Завантаження набору даних та визначення граничного значення.


```
[7] def find_TP(y_true, y_pred):
    # Підраховує кількість true positives (y_true = 1, y_pred = 1)
    return sum((y_true == 1) & (y_pred == 1))

def find_FN(y_true, y_pred):
    # Підраховує кількість false negatives (y_true = 1, y_pred = 0)
    return sum((y_true == 1) & (y_pred == 0))

def find_FP(y_true, y_pred):
    # Підраховує кількість false positives (y_true = 0, y_pred = 1)
    return sum((y_true == 0) & (y_pred == 1))

def find_TN(y_true, y_pred):
    # Підраховує кількість 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))
```

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

Рис.15. Результат функцій для перевірки confusion_matrix.

```
[ ] 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 ishchuk_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]])

ishchuk_confusion_matrix(df.actual_label.values, df.predicted_RF.values)
```

array([[5519, 2360],
[2832, 5047]])

Рис.16. Результат ф-ції ishchuk_confusion_matrix та перевірка роботи.

```
[ ] from sklearn.metrics import accuracy_score
accuracy_score(df.actual_label.values, df.predicted_RF.values)

0.6705165630156111

[ ] def ishchuk_accuracy_score(y_true, y_pred):
    TN, FP, FN, TP = confusion_matrix(y_true, y_pred).ravel()
    accuracy = (TP + TN) / (TP + TN + FP + FN)

    return accuracy

assert ishchuk_accuracy_score(df.actual_label.values, df.predicted_LR.values) == accuracy_score(df.actual_label.values, df.predicted_RF.values)
print('Accuracy RF: %.3f' % (ishchuk_accuracy_score(df.actual_label.values, df.predicted_RF.values)))
print('Accuracy LR: %.3f' % (ishchuk_accuracy_score(df.actual_label.values, df.predicted_LR.values)))

Accuracy RF: 0.671
Accuracy LR: 0.616
```

Рис.17. Результат ф-ції accuracy_score.

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

0.6405635232897576

[ ] def ishchuk_recall_score(y_true, y_pred):
    TP,FN,FP,TN = find_conf_matrix_values(y_true,y_pred)
    accuracy = (TP) / (TP + FN)
    return accuracy
assert ishchuk_recall_score(df.actual_label.values, df.predicted_LR.values) == recall_score(df.actual_label.values, df.predicted_LR.values)
print('Recall RF: %.3f'%(ishchuk_recall_score(df.actual_label.values, df.predicted_RF.values)))
print('Recall LR: %.3f'%(ishchuk_recall_score(df.actual_label.values, df.predicted_LR.values)))

Recall RF: 0.641
Recall LR: 0.543

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

0.681382476036182

[ ] def ishchuk_precision_score(y_true, y_pred):
    TP,FN,FP,TN = find_conf_matrix_values(y_true,y_pred)
    accuracy = accuracy = (TP) / (TP + FP)
    return accuracy
assert ishchuk_precision_score(df.actual_label.values, df.predicted_LR.values) == precision_score(df.actual_label.values, df.predicted_LR.values)
print('Precision RF: %.3f'%(ishchuk_precision_score(df.actual_label.values, df.predicted_RF.values)))
print('Precision LR: %.3f'%(ishchuk_precision_score(df.actual_label.values, df.predicted_LR.values)))

Precision RF: 0.681
Precision LR: 0.636
```

Рис.18. Результат ф-ції recall_score та precision_score.

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

0.660342797330891

[ ] def ishchuk_f1_score(y_true, y_pred):
    recall = ishchuk_recall_score(y_true,y_pred)
    precision = ishchuk_precision_score(y_true,y_pred)
    f1 = 2 * (precision * recall) / (precision + recall)
    return f1
assert ishchuk_f1_score(df.actual_label.values, df.predicted_LR.values) == f1_score(df.actual_label.values, df.predicted_LR.values)
print('F1 RF: %.3f'%(ishchuk_f1_score(df.actual_label.values, df.predicted_RF.values)))
print('F1 LR: %.3f'%(ishchuk_f1_score(df.actual_label.values, df.predicted_LR.values)))

F1 RF: 0.660
F1 LR: 0.586
```

Рис.19. Результат ф-ції f1_score.

		Іщук О.Ю..			ДУ «Житомирська політехніка».22.122.5.000 – Лр1	Арк.
		Голенко М.Ю.				10
Змн.	Арк.	№ докум.	Підпис	Дата		

```
[ ] print('scores with threshold = 0.5')
print('Accuracy RF: %.3f'%(ishchuk_accuracy_score(df.actual_label.values, df.predicted_RF.values)))
print('Recall RF: %.3f'%(ishchuk_recall_score(df.actual_label.values, df.predicted_RF.values)))
print('Precision RF: %.3f'%(ishchuk_precision_score(df.actual_label.values, df.predicted_RF.values)))
print('F1 RF: %.3f'%(ishchuk_f1_score(df.actual_label.values, df.predicted_RF.values)))
print('')
print('scores with threshold = 0.25')
print('Accuracy RF: %.3f'%(ishchuk_accuracy_score(df.actual_label.values, (df.model_RF >= 0.25).astype('int').values)))
print('Recall RF: %.3f'%(ishchuk_recall_score(df.actual_label.values, (df.model_RF >= 0.25).astype('int').values)))
print('Precision RF: %.3f'%(ishchuk_precision_score(df.actual_label.values, (df.model_RF >= 0.25).astype('int').values)))
print('F1 RF: %.3f'%(ishchuk_f1_score(df.actual_label.values, (df.model_RF >= 0.25).astype('int').values)))
```

```
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.20. Результати при різних порогах.

Зменшення порогу до 0.25 призвело до збільшення повності (Recall), що означає, що модель розпізнала більше позитивних прикладів. Проте це також призвело до зменшення точності (Precision), що означає більше помилкових позитивних класифікацій, коли модель вважає негативний приклад за позитивний.

З порогом 0.5 точність зберігається на вищому рівні, і модель робить менше помилкових позитивних класифікацій. Це робить F1-показник більш збалансованим між точністю і повністю.

Вибір оптимального порогу залежить від конкретного використання моделі і важливості точності та повності в конкретному контексті. Якщо важливо уникнути пропусків позитивних прикладів (важлива повнота), то може бути розумним знизити поріг. Однак це також призведе до більшої кількості помилкових позитивних класифікацій. Якщо важлива точність і уникнення надмірної кількості помилкових позитивних класифікацій, то краще залишити поріг на вищому рівні, як 0.5.

		Іщук О.Ю..			ДУ «Житомирська політехніка».22.122.5.000 – Лр1	Арк.
		Голенко М.Ю.				11
Змн.	Арк.	№ докум.	Підпис	Дата		

```
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')
```

```
Text(0, 0.5, 'True Positive Rate')
```

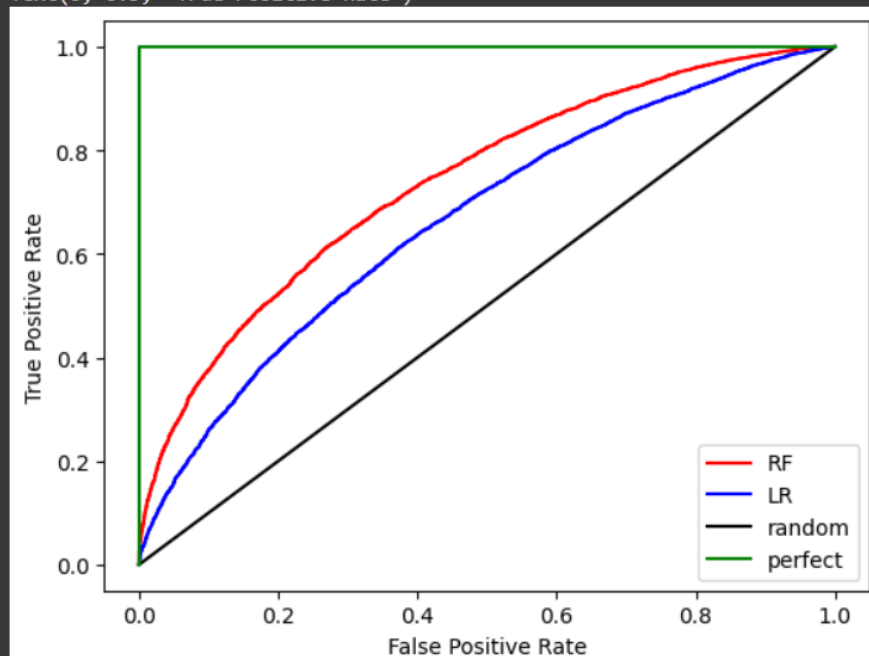


Рис.21. Результат побудови ROC кривої для кожної моделі.

```
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)
```

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

Рис.22. Метрика площі під кривою.

		Ицук О.Ю..			ДУ «Житомирська політехніка».22.122.5.000 – Лр1	Арк.
		Голенко М.Ю.				12
Змн.	Арк.	№ докум.	Підпис	Дата		

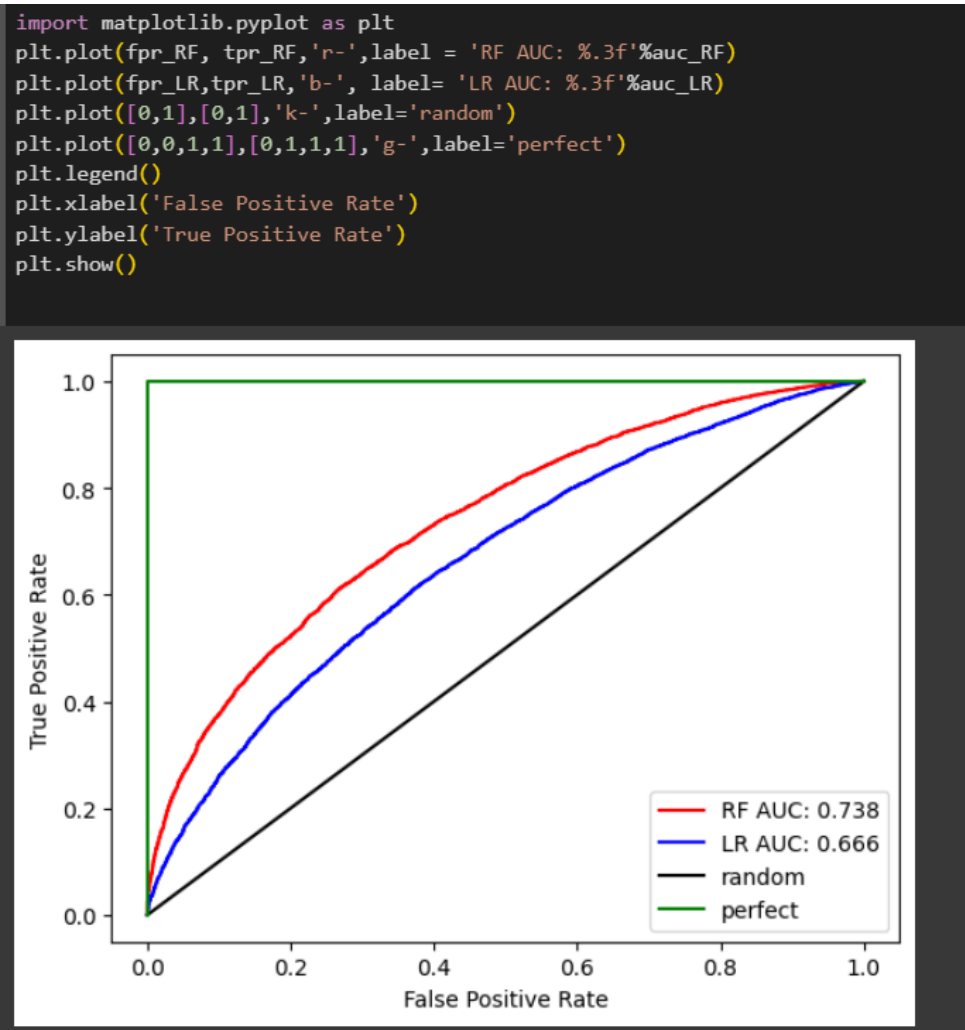


Рис.21. Результат побудови ROC кривої для кожної моделі з AUC.

Зазвичай, більше значення площі під ROC-кривою (AUC) свідчить про вищу якість класифікатора. У нашому випадку модель випадкового лісу (RF) демонструє кращу продуктивність, оскільки її значення AUC є вищим. Крім того, при використанні порогу 0.5, модель RF також виявляється точнішою, має вищу повноту та вищий F1-показник, порівняно з моделлю логістичної регресії (LR).

Завдання 6

		Іщук О.Ю..			ДУ «Житомирська політехніка».22.122.5.000 – Лр1	Арк.
		Голенко М.Ю.				13
Змн.	Арк.	№ докум.	Підпис	Дата		

```

# Розбиття 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)

# Ініціалізуємо модель SVC
svm_model = SVC(kernel='linear') # Використовуємо лінійне ядро для спрощення прикладу

# Тренуємо модель на навчальних даних
svm_model.fit(X_train, y_train)

# Проводимо класифікацію на тестових даних
y_pred = svm_model.predict(X_test)

classification_rep = classification_report(y_test, y_pred)

# Обчислення якості класифікатора
accuracy = 100.0 * (y_test == y_pred).sum() / X_test.shape[0]
print("Accuracy of the new classifier =", round(accuracy, 2), "%")

# Виводимо результати
print("Classification Report:\n", classification_rep)

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

```

Рис.22. Класифікація даних за допомогою SVC.

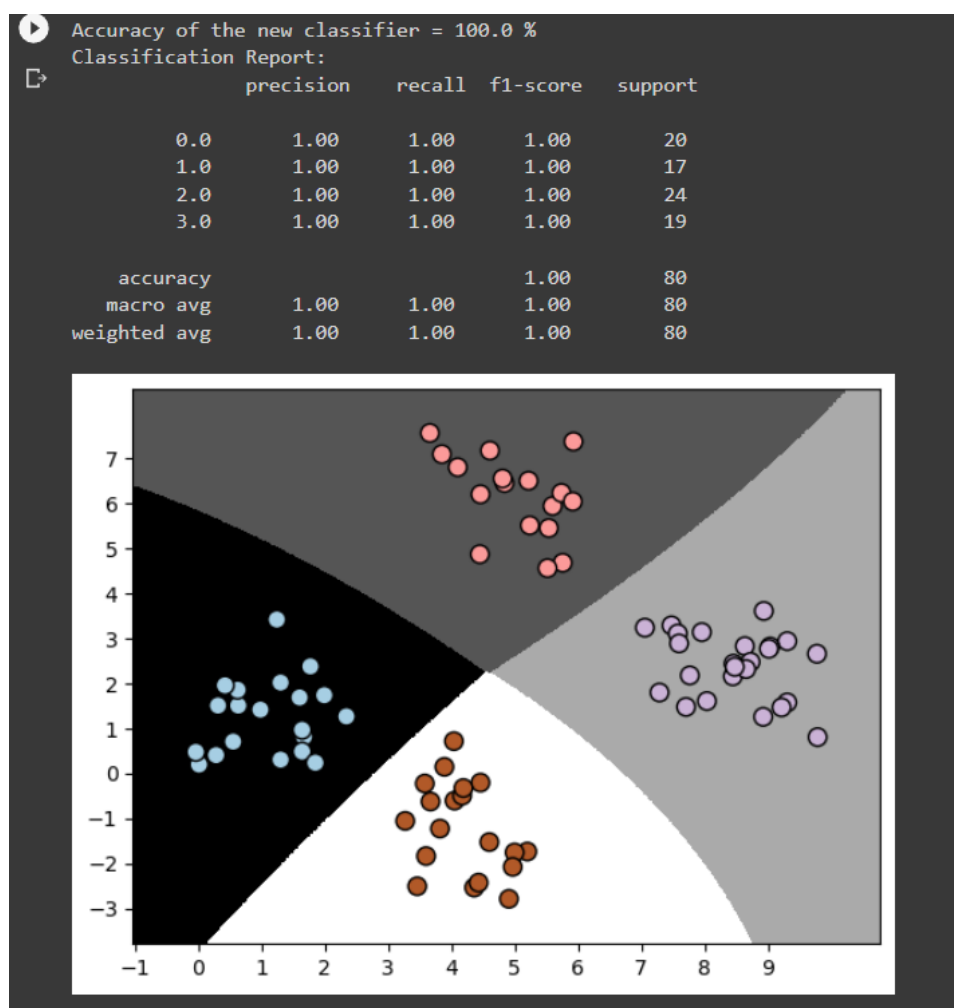


Рис.23. Результат роботи.

З огляду на отримані результати, які показують ідеальну точність (Ассигасу = 100%) для обох моделей, а саме байєсівського класифікатора і методу опорних векторів (SVC), важливо враховувати наступне. Байєсівський класифікатор, зокрема Gaussian Naive Bayes, вважається менш складним та обчислювально ефективним методом, в той час як метод опорних векторів (SVC) використовує

більш складні обчислення і може давати кращі результати, коли дані мають складну структуру або розподіл.

В нашому випадку обидві моделі успішно справляються із поставленою задачею класифікації, і вибір між ними буде залежати від конкретних вимог та умов.

		Іщук О.Ю..			ДУ «Житомирська політехніка».22.122.5.000 – Лр1	Арк.
		Голенко М.Ю.				15
Змн.	Арк.	№ докум.	Підпис	Дата		