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:

Завдання 1.

```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)
```

```
 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]]
```

Різниця між L1-нормалізацією та L2-нормалізацією.

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 значень.

L1 – менш чутлива до викидів, тому вважається більш надійною. Її використовують коли є підозра на наявність багатьох незначущих ознак у даних. Але якщо вирішувати завдання де викиди мають грати чутливу роль, тобто більш стабільні дані, краще використовувати L2-нормалізацію.

Завдання 2.1

```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 ) )
```

```
PS D:\proggers\AI> python LR_1_task_1.py

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']
```

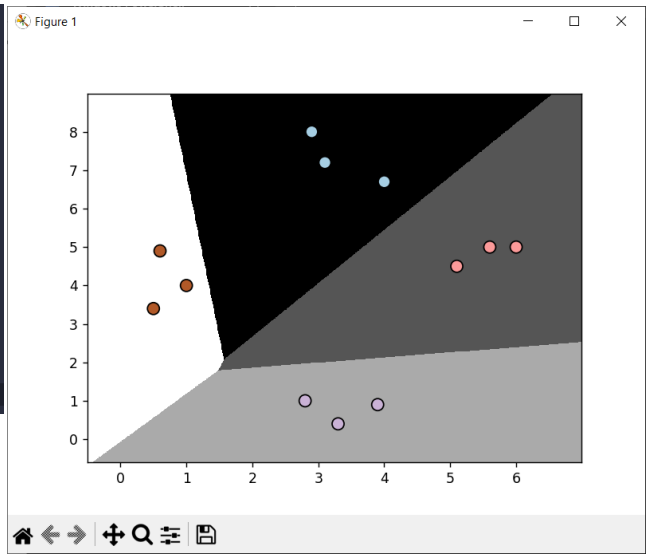
Завдання 2.2

```python
import numpy as np
from sklearn import preprocessing
input_data = np.array([[1.3, 3.9, 6.2],
 [4.9, 2.2, -4.3],
 [-2.6, 6.5, 4.6],
[-5.2, -3.4, -5.2]])

# Бінаризація даних
data_binarized = preprocessing.Binarizer(threshold=2.7).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)
```

```
PS D:\proggers\AI> python LR_1_task_2.py

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

BEFORE:
Mean = [-0.4    2.3    0.325]
Std deviation = [3.83601356 3.62973828 5.11633414]

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

Min max scaled data:
 [[0.64356436 0.73737374 1.        ]
 [1.         0.56565657 0.07894737]
 [0.25742574 1.         0.85964912]
 [0.         0.         0.        ]]

l1 normalized data:
 [[ 0.11403509  0.34210526  0.54385965]
 [ 0.42982456  0.19298246 -0.37719298]
 [-0.18978102  0.47445255  0.33576642]
 [-0.37681159 -0.24637681 -0.37681159]]

l2 normalized data:
 [[ 0.17475265  0.52425796  0.83343572]
 [ 0.71216718  0.31974853 -0.62496303]
 [-0.31038277  0.77595693  0.54913875]
 [-0.64182859 -0.41965715 -0.64182859]]
```

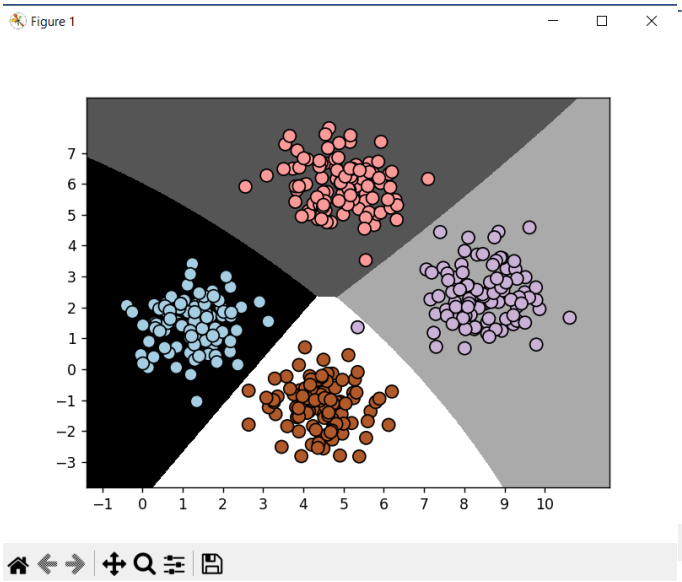
Завдання 2.3

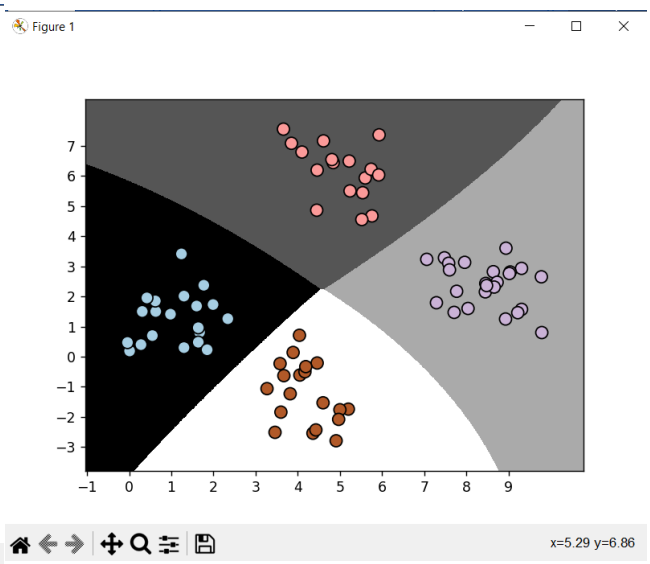
```python
import numpy as np
from sklearn import linear_model
import matplotlib.pyplot as plt
from utilities import visualize_classifier

# 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)
```



Завдання 2.4

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






```
PS D:\proggers\AI> python LR_1_task_4.py
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%
```

Перший класифікатор навчається на всьому наборі даних, тоді як другий - на розділених навчальних і тестових даних. З рисунків можна побачити різницю: другий класифікатор точніше.

Завдання 2.5

```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')
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)
```

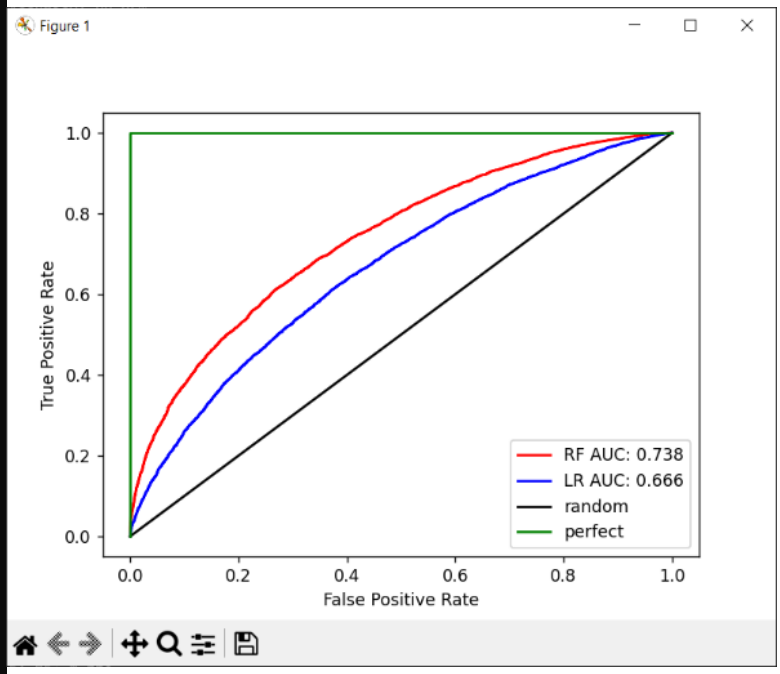
```
 return sum((y_true == 1) & (y_pred == 1))
def find_FN(y_true, y_pred):
 # counts the number of false negatives (y_true = 1, y_pred = 0)
 return sum((y_true == 1) & (y_pred == 0))
def find_FP(y_true, y_pred):
 # counts the number of false positives (y_true = 0, y_pred = 1)
 return sum((y_true == 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 shcherback_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]])

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

assert np.array_equal(shcherback_confusion_matrix(df.actual_label.values, df.pre-
dicted_RF.values),confusion_matrix(df.actual_label.values,
df.predicted_RF.values)), 'shcherback_confusion_matrix() is not correct for RF'
assert np.array_equal(shcherback_confusion_matrix(df.actual_label.values, df.pre-
dicted_LR.values),confusion_matrix(df.actual_label.values,
df.predicted_LR.values) ), 'shcherback_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 shcherback_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 shcherback_accuracy_score(df.actual_label.values,
df.predicted_RF.values) ==accuracy_score(df.actual_label.values, df.predicted_RF.val-
ues), 'shcherback_accuracy_score failed on RF'
assert shcherback_accuracy_score(df.actual_label.values,
df.predicted_LR.values) == accuracy_score(df.actual_label.values, df.pre-
dicted_LR.values), 'shcherback_accuracy_score failed on LR'
print('Accuracy RF: %.3f'%(shcherback_accuracy_score(df.actual_label.values, df.pre-
dicted_RF.values)))
print('Accuracy LR: %.3f'%(shcherback_accuracy_score(df.actual_label.values, df.pre-
dicted_LR.values)))

from sklearn.metrics import recall_score
# print(recall_score(df.actual_label.values, df.predicted_RF.values))

def shcherback_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)
shcherback_recall_score(df.actual_label.values,
df.predicted_RF.values) == recall_score(df.actual_label.values, df.predicted_RF.val-
ues),'shcherback_recall_score failed on RF'
assert shcherback_recall_score(df.actual_label.values,
df.predicted_LR.values) == recall_score(df.actual_label.values,df.predicted_LR.val-
ues), 'shcherback_recall_score failed on LR'
print('Recall RF: %.3f'%(shcherback_recall_score(df.actual_label.values, df.pre-
dicted_RF.values)))
print('Recall LR: %.3f'%(shcherback_recall_score(df.actual_label.values, df.pre-
dicted_LR.values)))


from sklearn.metrics import precision_score
# print(precision_score(df.actual_label.values,df.predicted_RF.values))

def shcherback_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)
shcherback_precision_score(df.actual_label.values,
df.predicted_RF.values) == precision_score(df.actual_label.values, df.pre-
dicted_RF.values), 'shcherback_precision_score failed on RF'
assert shcherback_precision_score(df.actual_label.values,
df.predicted_LR.values) == precision_score(df.actual_label.values, df.pre-
dicted_LR.values), 'shcherback_precision_score failed on LR'
print('Precision RF: %.3f'%(shcherback_precision_score(df.actual_label.values,
df.predicted_RF.values)))
print('Precision LR: %.3f'%(shcherback_precision_score(df.actual_label.values,
df.predicted_LR.values)))

from sklearn.metrics import f1_score
# print(f1_score(df.actual_label.values, df.predicted_LR.values))
```

```
def shcherback_f1_score(y_true, y_pred):
 # calculates the F1 score
 recall = shcherback_recall_score(y_true,y_pred)
 precision = shcherback_precision_score(y_true,y_pred)
 return 2*(precision*recall)/(precision+recall)
shcherback_f1_score(df.actual_label.values,
df.predicted_RF.values) == f1_score(df.actual_label.values, df.predicted_RF.values),
'shcherback_f1_score failed on RF'
assert shcherback_f1_score(df.actual_label.values,
df.predicted_LR.values) == f1_score(df.actual_label.values, df.predicted_LR.values),
'shcherback_f1_score failed on LR'
print('F1 RF: %.3f'%(shcherback_f1_score(df.actual_label.values, df.predicted_RF.val-
ues)))
print('F1 LR: %.3f'%(shcherback_f1_score(df.actual_label.values, df.predicted_LR.val-
ues)))



print('scores with threshold = 0.5')
print('Accuracy RF: %.3f'%(shcherback_accuracy_score(df.actual_label.values,df.pre-
dicted_RF.values)))
print('Recall RF: %.3f'%(shcherback_recall_score(df.actual_label.values,df.pre-
dicted_RF.values)))
print('Precision RF:%.3f'%(shcherback_precision_score(df.actual_label.values,df.pre-
dicted_RF.values)))
print('F1 RF: %.3f'%(shcherback_f1_score(df.actual_label.values,df.predicted_RF.val-
ues)))
print('')
print('scores with threshold = 0.25')
print('Accuracy RF:%.3f'%(shcherback_accuracy_score(df.actual_label.values,
(df.model_RF >=0.25).astype('int').values)))
print('Recall RF: %.3f'%(shcherback_recall_score(df.actual_label.values,(df.model_RF
>= 0.25).astype('int').values)))
print('Precision RF:%.3f'%(shcherback_precision_score(df.actual_label.values,
(df.model_RF>= 0.25).astype('int').values)))
print('F1 RF: %.3f'%(shcherback_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)


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()
```

```
PS D:\proggers\AI> python LR_1_task_5.py
[[5519 2360]
 [2832 5047]]
TP: 5047
FN: 2832
FP: 2360
TN: 5519
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: 0.660
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
```



Порівнявши результати для різних порогів, можна зробити висновки: Зменшення порогу призводить до зменшення точності та збільшення чутливості (recall).

Завдання 2.6

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

# Зчитуємо дані з файлу
data = pd.read_csv('data_multivar_nb.txt', header=None, names=['feature1', 'fea-
ture2', 'label'])

# Розділяємо дані на ознаки і цільову змінну
X = data[['feature1', 'feature2']]
y = data['label']

# Розділяємо дані на навчальну та тестову вибірки
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran-
dom_state=42)

# Навчаємо SVM модель
svm_classifier = svm.SVC(kernel='linear')
svm_classifier.fit(X_train, y_train)

# Навчаємо наївний байєсівський класифікатор
naive_bayes_classifier = naive_bayes.GaussianNB()
naive_bayes_classifier.fit(X_train, y_train)

# Прогнозуємо класи для тестових даних
svm_predictions = svm_classifier.predict(X_test)
naive_bayes_predictions = naive_bayes_classifier.predict(X_test)

# Оцінюємо якість SVM моделі
svm_accuracy = accuracy_score(y_test, svm_predictions)
svm_precision = precision_score(y_test, svm_predictions, average='weighted')
svm_recall = recall_score(y_test, svm_predictions, average='weighted')
svm_f1 = f1_score(y_test, svm_predictions, average='weighted')

# Оцінюємо якість наївного байєсівського класифікатора
naive_bayes_accuracy = accuracy_score(y_test, naive_bayes_predictions)
naive_bayes_precision = precision_score(y_test, naive_bayes_predictions, aver-
age='weighted')
naive_bayes_recall = recall_score(y_test, naive_bayes_predictions, aver-
age='weighted')
naive_bayes_f1 = f1_score(y_test, naive_bayes_predictions, average='weighted')

# Виводимо результати
print("SVM Метрики:")
print(f"Акуратність: {svm_accuracy}")
print(f"Точність: {svm_precision}")
print(f"Чутливість: {svm_recall}")
```

```python
print(f"F1-показник: {svm_f1}")

print("\nНаївний байєсівський класифікатор Метрики:")
print(f"Акуратність: {naive_bayes_accuracy}")
print(f"Точність: {naive_bayes_precision}")
print(f"Чутливість: {naive_bayes_recall}")
print(f"F1-показник: {naive_bayes_f1}")
```

```
PS D:\proggers\AI> python LR_1_task_6.py
SVM Метрики:
Акуратність: 0.9875
Точність: 0.9885416666666667
Чутливість: 0.9875
F1-показник: 0.9876263902932255

Наївний байєсівський класифікатор Метрики:
Акуратність: 0.9875
Точність: 0.9885416666666667
Чутливість: 0.9875
F1-показник: 0.9876263902932255
```

Github: ***https://github.com/mtvi/ipz202_Shcherback***

Висновки: використовуючи спеціалізовані бібліотеки та мову програмування Python дослідили попередню обробку та класифікацію даних.