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/ingaliptn/AI

Хід роботи:

Завдання 2.1. Попередня обробка даних

```
import numpy as np
from sklearn import preprocessing
import numpy as np
from sklearn import linear_model
import matplotlib.pyplot as plt
from utilities import visualize_classifier
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]])
data_binarized = preprocessing.Binarizer(threshold=3.0).transform(input_data)
print("\n Binarized data:\n", data_binarized)
print("\nBEFORE: ")
print("Mean =", input_data.mean(axis=0))
print("Std deviation =", input_data.std(axis=0))
data_scaled = preprocessing.scale(input_data)
print("\nAFTER: ")
print("Mean =", data_scaled.mean(axis=0))
print("Std deviation =", data_scaled.std(axis=0))

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)

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)
```

```
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.

Основна відмінність полягає в тому, як обчислюється норма кожного вектора та як відбувається їх нормалізація. L1-норма схильніша до створення розріджених векторів (де більшість значень близька до 0), тоді як L2-норма зазвичай забезпечує більш рівномірний розподіл значень.

```
import numpy as np
from sklearn import preprocessing

input_labels = ['red', 'back', 'red', 'green', 'black', 'yellow', 'white']

encoder = preprocessing.LabelEncoder()
encoder.fit(input_labels)

print("\nLabel mapping:")
for i, item in enumerate(encoder.classes):
    print(item, '-->', i)

test_labels = ['green', 'red', 'ьlack']
encoded_values = encoder.transform(test_labels )
print("\nLabels =", test_labels )
print("Encoded values =", list (encoded_values ) )

encoded_values = [3, 0, 4, 1]
decoded_list = encoder.inverse_transform(encoded_values)
print("\nEncoded values =", encoded_values)
print("Decoded labels =", list (decoded_list))
```

Завдання 2.2. Попередня обробка нових даних

```
import numpy as np
from sklearn import preprocessing
from sklearn import linear_model
import matplotlib.pyplot as plt
from utilities import visualize_classifier
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]])
data_binarized = preprocessing.Binarizer(threshold=3.0).transform(input_data)
print("\n Binarized data:\n", data_binarized)
print("\nBEFORE: ")
print("Mean =", input_data.mean(axis=0))
print("Std deviation =", input_data.std(axis=0))
data_scaled = preprocessing.scale(input_data)
print("\nAFTER: ")
print("Mean =", data_scaled.mean(axis=0))
print("Std deviation =", data_scaled.std(axis=0))

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)

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)
```

```
BEFORE:
Mean = [-0.325  0.325  0.775]
Std deviation = [3.17125764 2.99113273 4.79446295]

AFTER:
Mean = [ 5.55111512e-17 -8.32667268e-17  6.93889390e-17]
Std deviation = [1. 1. 1.]

Min max scaled data:
 [[0.88157895 0.         1.        ]
 [0.26315789 0.05633803 0.85245902]
 [1.         1.         0.        ]
 [0.         0.02816901 0.40163934]]

l1 normalized data:
 [[ 0.23       -0.16        0.61      ]
 [-0.30379747 -0.15189873  0.5443038 ]
 [ 0.21621622  0.37162162 -0.41216216]
 [-0.62857143 -0.2        -0.17142857]]

l2 normalized data:
 [[ 0.34263541 -0.23835507  0.90872869]
 [-0.47351004 -0.23675502  0.84837215]
 [ 0.36302745  0.62395344 -0.69202108]
 [-0.92228798 -0.29345527 -0.25153308]]
```

Рис 3.

Завдання 2.3. Класифікація логістичною регресією або логістичний класифікатор

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

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])

classifier = linear_model.LogisticRegression(solver='liblinear',C=1)
classifier.fit(X, y)
visualize_classifier(classifier, X, y)
```

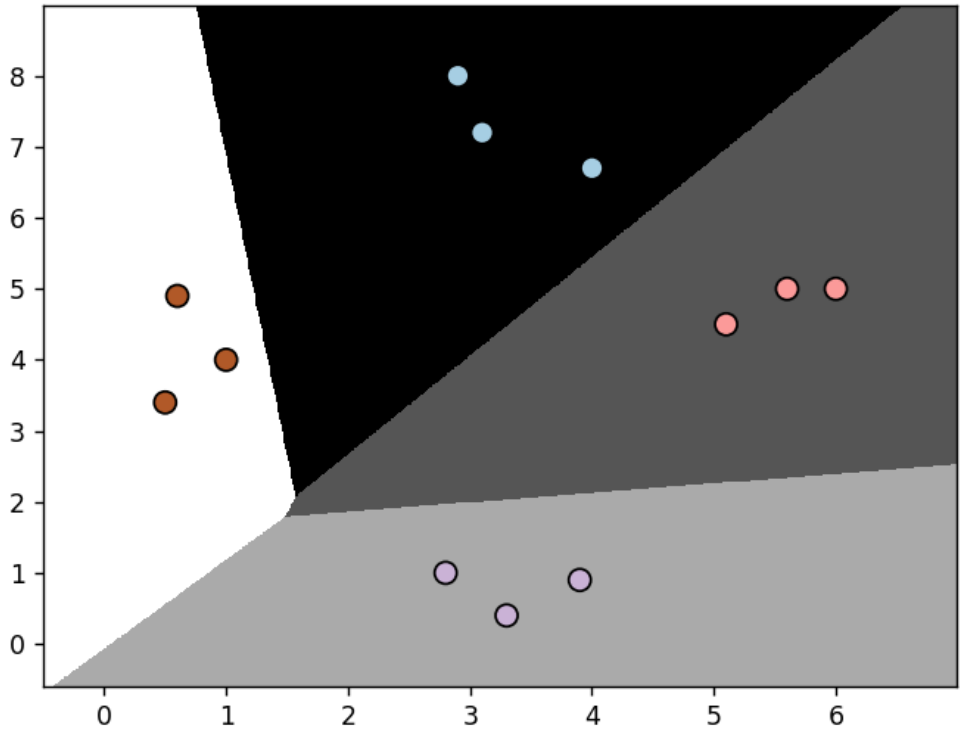


Рис 4.

Завдання 2.4. Класифікація наївним байєсовським класифікатором

```
import numpy as np
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, classification_report

data = np.loadtxt('data_multivar_nb.txt', delimiter=',')

X = data[:, :-1]
y = data[:, -1]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

svm_model = SVC(kernel='linear')
svm_model.fit(X_train, y_train)

svm_predictions = svm_model.predict(X_test)
```

```
print("Support Vector Machine (SVM) Classification Report:")
print(classification_report(y_test, svm_predictions))
print("Accuracy:", accuracy_score(y_test, svm_predictions))

nb_model = GaussianNB()
nb_model.fit(X_train, y_train)

nb_predictions = nb_model.predict(X_test)

print("\nNaive Bayes Classification Report:")
print(classification_report(y_test, nb_predictions))
print("Accuracy:", accuracy_score(y_test, nb_predictions))
```

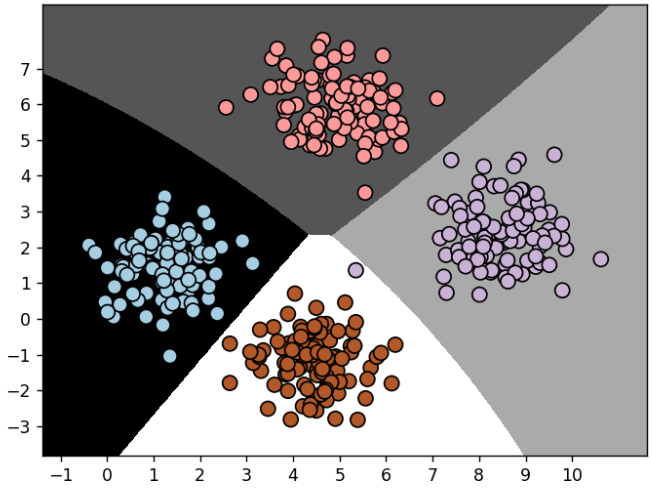


Рис 5

```
import numpy as np
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, classification_report

data = np.loadtxt('data_multivar_nb.txt', delimiter=',')

X = data[:, :-1]
y = data[:, -1]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

svm_model = SVC(kernel='linear')
svm_model.fit(X_train, y_train)

svm_predictions = svm_model.predict(X_test)

print("Support Vector Machine (SVM) Classification Report:")
print(classification_report(y_test, svm_predictions))
print("Accuracy:", accuracy_score(y_test, svm_predictions))
```

```
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)

nb_predictions = nb_model.predict(X_test)

print("\nNaive Bayes Classification Report:")
print(classification_report(y_test, nb_predictions))
print("Accuracy:", accuracy_score(y_test, nb_predictions))
```

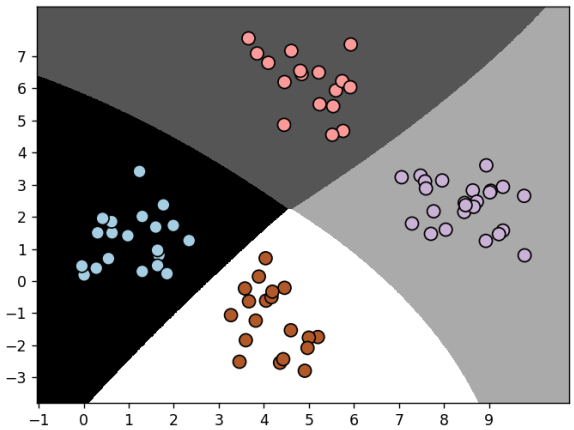


Рис 6.

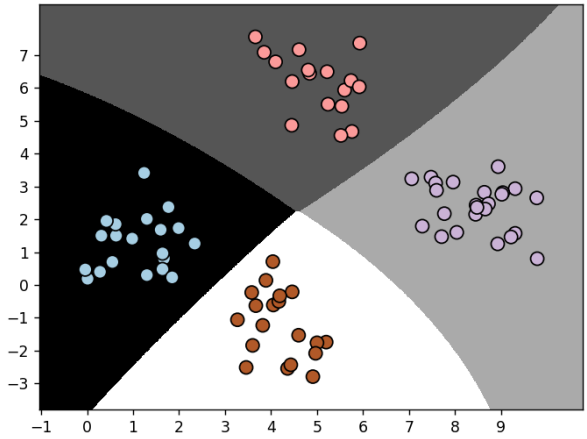


Рис 7.

```
D:\Labs\OWI\AILab1\venv\Scripts\python.exe D:/Labs/OWI/AILab1/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%
```

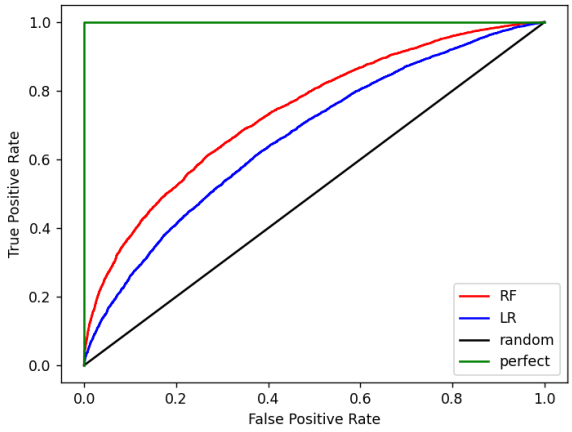
Завдання 2.5. Вивчити метрики якості класифікації

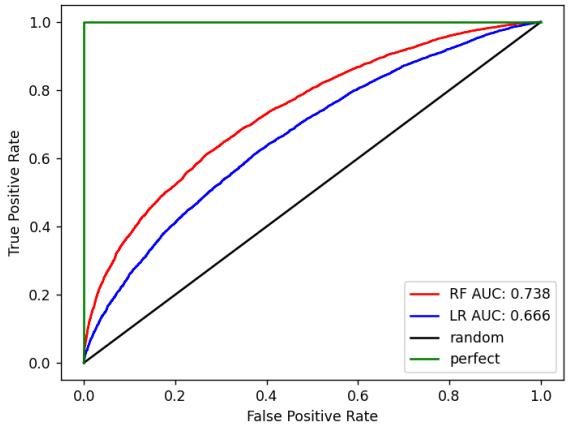
```
import pandas as pd
import numpy as np
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score

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

df = pd.read_csv('data_metrics.csv')
thresh = 0.5
df['predicted_RF'] = (df.model_RF >= thresh).astype('int')
df['predicted_LR'] = (df.model_LR >= thresh).astype('int')

def find_TP(y_true, y_pred):
    return sum((y_true == 1) & (y_pred == 1))

def find_FN(y_true, y_pred):
    return sum((y_true == 1) & (y_pred == 0))

def find_FP(y_true, y_pred):
    return sum((y_true == 0) & (y_pred == 1))

def find_TN(y_true, y_pred):
    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

def tkachuk_mykyta_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]])

assert np.array_equal(tkachuk_mykyta_confusion_matrix(df.actual_label.values,
df.predicted_RF.values),
                      confusion_matrix(df.actual_label.values, df.predicted_RF.values)),
'tkachuk_mykyta_confusion_matrix() is not correct for RF'
```

```
assert np.array_equal(tkachuk_mykyta_confusion_matrix(df.actual_label.values,
df.predicted_LR.values),
                      confusion_matrix(df.actual_label.values, df.predicted_LR.values)),
'tkachuk_mykyta_confusion_matrix() is not correct for LR'

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

assert tkachuk_mykyta_accuracy_score(df.actual_label.values, df.predicted_RF.values) ==
accuracy_score(df.actual_label.values, df.predicted_RF.values),
'tkachuk_mykyta_accuracy_score failed on RF'

assert tkachuk_mykyta_accuracy_score(df.actual_label.values, df.predicted_LR.values) ==
accuracy_score(df.actual_label.values, df.predicted_LR.values),
'tkachuk_mykyta_accuracy_score failed on LR'

print('Accuracy RF: %.3f' % (tkachuk_mykyta_accuracy_score(df.actual_label.values,
df.predicted_RF.values)))
print('Accuracy LR: %.3f' % (tkachuk_mykyta_accuracy_score(df.actual_label.values,
df.predicted_LR.values)))

#### Precision


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

def tkachuk_mykyta_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)
    precision = TP / (TP + FP)
    return precision

assert tkachuk_mykyta_precision_score(df.actual_label.values, df.predicted_RF.values) ==
precision_score(df.actual_label.values, df.predicted_RF.values),
'tkachuk_mykyta_precision_score failed on RF'

assert tkachuk_mykyta_precision_score(df.actual_label.values, df.predicted_LR.values) ==
precision_score(df.actual_label.values, df.predicted_LR.values),
'tkachuk_mykyta_precision_score failed on LR'

print('Precision RF: %.3f' % (tkachuk_mykyta_precision_score(df.actual_label.values,
df.predicted_RF.values)))
print('Precision LR: %.3f' % (tkachuk_mykyta_precision_score(df.actual_label.values,
df.predicted_LR.values)))

###
def tkachuk_mykyta_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)
    recall = TP / (TP + FN)
    return recall
```

```
assert tkachuk_mykyta_recall_score(df.actual_label.values, df.predicted_RF.values) ==
recall_score(df.actual_label.values, df.predicted_RF.values),
'tkachuk_mykyta_accuracy_score failed on RF'

assert tkachuk_mykyta_recall_score(df.actual_label.values, df.predicted_LR.values) ==
recall_score(df.actual_label.values, df.predicted_LR.values),
'tkachuk_mykyta_accuracy_score failed on LR'

print('Recall RF: %.3f' % (tkachuk_mykyta_recall_score(df.actual_label.values,
df.predicted_RF.values)))
print('Recall LR: %.3f' % (tkachuk_mykyta_recall_score(df.actual_label.values,
df.predicted_LR.values)))

###


def tkachuk_mykyta_f1_score(y_true, y_pred):
    # calculates the F1 score
    recall = tkachuk_mykyta_recall_score(y_true, y_pred)
    precision = tkachuk_mykyta_precision_score(y_true, y_pred)
    return 2 * (precision * recall) / (precision + recall)

assert tkachuk_mykyta_f1_score(df.actual_label.values, df.predicted_RF.values) ==
f1_score(df.actual_label.values, df.predicted_RF.values), 'tkachuk_mykyta_f1_score failed
on RF'

assert tkachuk_mykyta_f1_score(df.actual_label.values, df.predicted_LR.values) ==
f1_score(df.actual_label.values, df.predicted_LR.values), 'tkachuk_mykyta_f1_score failed
on LR'

print('F1 RF: %.3f' % (tkachuk_mykyta_f1_score(df.actual_label.values,
df.predicted_RF.values)))
print('F1 LR: %.3f' % (tkachuk_mykyta_f1_score(df.actual_label.values,
df.predicted_LR.values)))
print('scores with threshold = 0.5')
print('Accuracy RF: %.3f' % (tkachuk_mykyta_accuracy_score(df.actual_label.values,
df.predicted_RF.values)))
print('Recall RF: %.3f' % (tkachuk_mykyta_recall_score(df.actual_label.values,
df.predicted_RF.values)))
print('Precision RF: %.3f' % (tkachuk_mykyta_precision_score(df.actual_label.values,
df.predicted_RF.values)))
print('F1 RF: %.3f' % (tkachuk_mykyta_f1_score(df.actual_label.values,
df.predicted_RF.values)))
print('')

print('scores with threshold = 0.25')
print('Accuracy RF: %.3f' % (tkachuk_mykyta_accuracy_score(df.actual_label.values,
(df.model_RF >= 0.25).astype('int').values)))
print('Recall RF: %.3f' % (tkachuk_mykyta_recall_score(df.actual_label.values,
(df.model_RF >= 0.25).astype('int').values)))
print('Precision RF: %.3f' % (tkachuk_mykyta_precision_score(df.actual_label.values,
(df.model_RF >= 0.25).astype('int').values)))
```

```
print('F1 RF: %.3f' % (tkachuk_mykyta_f1_score(df.actual_label.values, (df.model_RF >=
0.25).astype('int').values)))


###

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)

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()
```





```
D:\Labs\OWI\AILab1\venv\Scripts\pyth
TP: 5047
FN: 2832
FP: 2360
TN: 5519
Accuracy RF: 0.671
Accuracy LR: 0.616
Precision RF: 0.681
Precision LR: 0.636
Recall RF: 0.641
Recall LR: 0.543
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
```

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.

```
D:\Labs\OWI\AILab1\venv\Scripts\python.exe D:/Labs/OWI/AILab1/LR
Support Vector Machine (SVM) Classification Report:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        22
         1.0       1.00      1.00      1.00        25
         2.0       1.00      0.95      0.98        22
         3.0       0.92      1.00      0.96        11

    accuracy                           0.99        80
   macro avg       0.98      0.99      0.98        80
weighted avg       0.99      0.99      0.99        80

Accuracy: 0.9875

Naive Bayes Classification Report:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        22
         1.0       1.00      1.00      1.00        25
         2.0       1.00      0.95      0.98        22
         3.0       0.92      1.00      0.96        11

    accuracy                           0.99        80
   macro avg       0.98      0.99      0.98        80
weighted avg       0.99      0.99      0.99        80

Accuracy: 0.9875
```

Висновки:. В цій лабораторній роботі, я використовуючи спеціалізовані бібліотеки та мову програмування Python дослідив попередню обробку та класифікацію даних