Лабораторна робота №1

ПОПЕРЕДНЯ ОБРОБКА ТА КОНТРОЛЬОВАНА КЛАСИФІКАЦІЯ ДАНИХ

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

Завдання на лабораторну роботу

Завдання 2.1. Попередня обробка даних

2.1.1. Бінарізація

```
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]])

data_binarized = preprocessing.Binarizer(threshold=2.1).transform(in-
put_data)
print("\n Binarized data:\n", data_binarized)
```

Результат виконання

```
Binarized data:
[[1. 0. 1.]
[0. 1. 0.]
[1. 0. 0.]
[1. 0. 0.]]
PS E:\labs\WIKI\lab1>
```

2.1.2. Виключення середнього

```
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))
```

Результат виконання

```
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.]
PS E:\labs\WIKI\lab1>
```

2.1.3. Масштабування

```
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)
```

Результат виконання

```
Min max scaled data:
 [[0.74117647 0.39548023 1.        ]
 [0.         1.         0.        ]
 [0.6        0.5819209  0.87234043]
 [1.         0.         0.17021277]]
PS E:\labs\WIKI\lab1>
```

2.1.4. Нормалізація

```
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

```
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]]
PS E:\labs\WIKI\lab1>
```

L1 та L2 нормалізації використовуються для зміни масштабу характеристик до спільного стандарту, зменшуючи чутливість моделей до відмінностей у масштабах величин характеристик.

L1 нормалізація (нормалізація на основі суми абсолютних значень) полягає у масштабуванні векторів таким чином, щоб сума абсолютних значень усіх компонентів вектора дорівнювала 1. Цей метод стійкий до викидів, оскільки великі значення компонентів менше впливають на загальну нормалізацію. L1 нормалізація використовується у задачах, де важливо зберегти розрідженість даних, тобто більшість компонентів вектора можуть бути нульовими.

L2 нормалізація (також відома як нормалізація за допомогою евклідової відстані або нормалізація на основі квадратичних значень) полягає у масштабуванні векторів таким чином, щоб сума квадратів усіх компонентів вектора дорівнювала 1. Цей метод підходить для задач, де важливо зберегти відносні відстані між точками у просторі ознак. L2 нормалізація частіше використовується у методах машинного навчання, де модель орієнтується на відстані між точками.

2.1.5. Кодування міток

```
import numpy as np
from sklearn import preprocessing

input_labels = ['red', 'Ыack', 'red', 'green', 'Ьlack', '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', 'Ыack']
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))
```

Результат виконання

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

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

Encoded values = [3, 0, 4, 1]
Decoded labels = ['yellow', 'green', 'black', 'red']
PS E:\labs\WIKI\lab1>
```

Завдання 2.2. Попередня обробка нових даних

Результати з новими даними (Варіант 11):

```
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]])
```

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

Бінарізації

```
BEFORE:
Mean = [ 0.725 -2.     0.4  ]
Std deviation = [3.49454933 4.97543968 3.72491611]

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

Виключення середнього

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

Масштабування

```
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]]
```

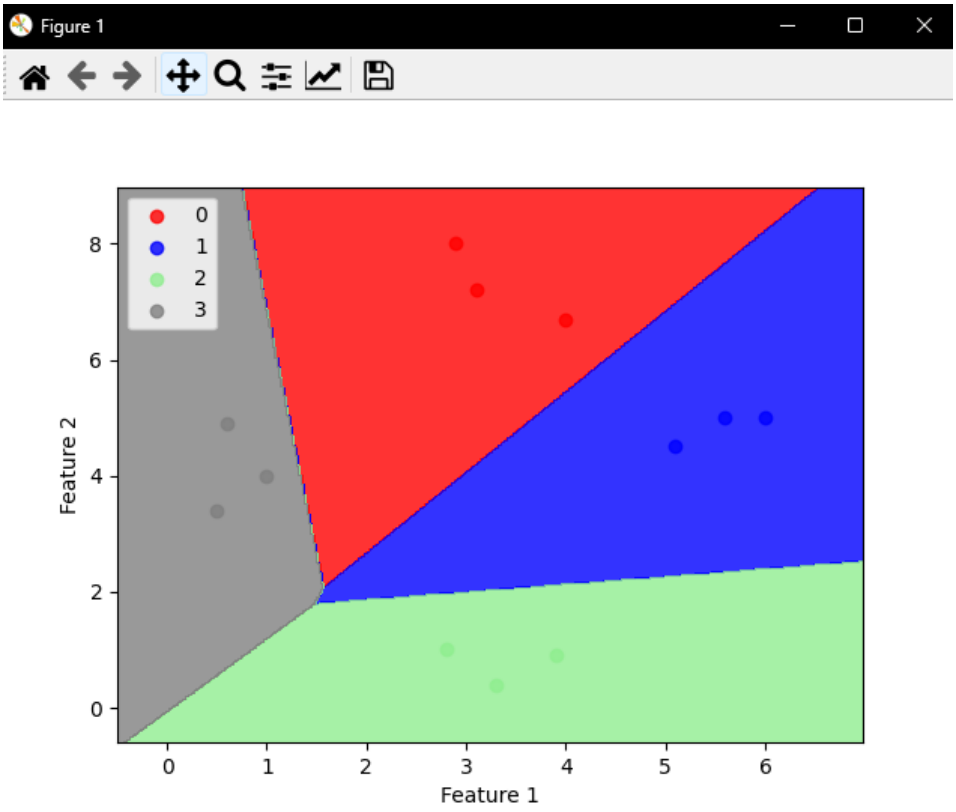
Нормалізації

Завдання 2.3. Класифікація логістичною регресією або логістичний класифікатор

```python
import numpy as np
from sklearn import linear_model
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

def visualize_classifier(classifier, X, y):
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1

    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02),
                         np.arange(y_min, y_max, 0.02))
```

```
    Z = classifier.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    plt.contourf(xx, yy, Z, alpha=0.8, cmap=ListedColormap(('red', 'blue',
'lightgreen', 'gray')))

    for idx, cl in enumerate(np.unique(y)):
        plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],
                    alpha=0.8, c=ListedColormap(('red', 'blue', 'light-
green', 'gray'))(idx),
                    marker='o', label=cl)

    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.legend(loc='upper left')
    plt.show()

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)
```

Результат виконання

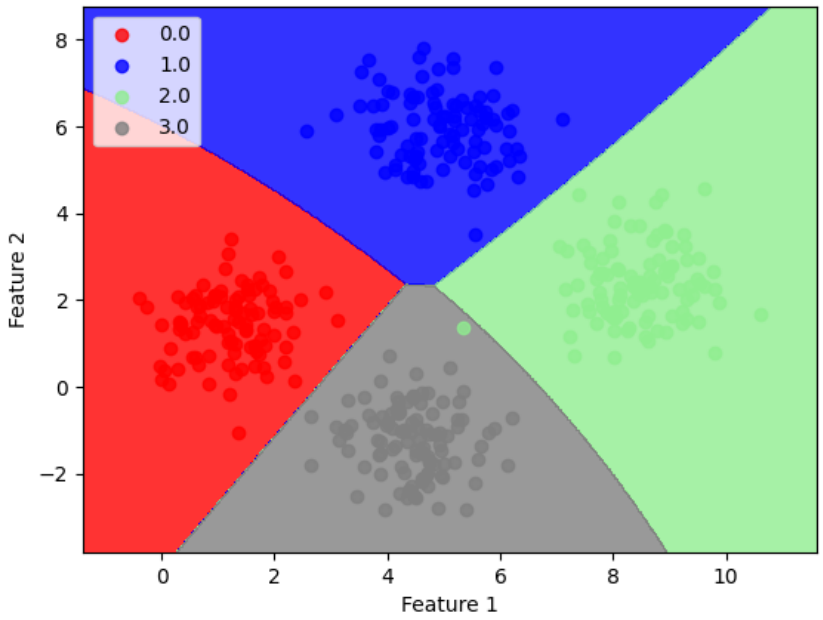


Завдання 2.4. Класифікація наївним байєсовським класифікатором

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from matplotlib.colors import ListedColormap

def visualize_classifier(classifier, X, y):
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1

    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02),
                         np.arange(y_min, y_max, 0.02))

    Z = classifier.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    plt.contourf(xx, yy, Z, alpha=0.8, cmap=ListedColormap(('red', 'blue',
'lightgreen', 'gray')))
```

```
    for idx, cl in enumerate(np.unique(y)):
        plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],
                    alpha=0.8, c=ListedColormap(('red', 'blue', 'light-
green', 'gray'))(idx),
                    marker='o', label=cl)

    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.legend(loc='upper left')
    plt.show()


input_file = 'data_multivar_nb.txt'
data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1]
classifier = GaussianNB()
classifier.fit(X, y)
y_pred = classifier.predict(X)
accuracy = 100.0 * (y == y_pred).sum() / X.shape[0]
print("Accuracy of Naive Bayes classifier =", round(accuracy, 2), "%")
visualize_classifier(classifier, X, y)
```

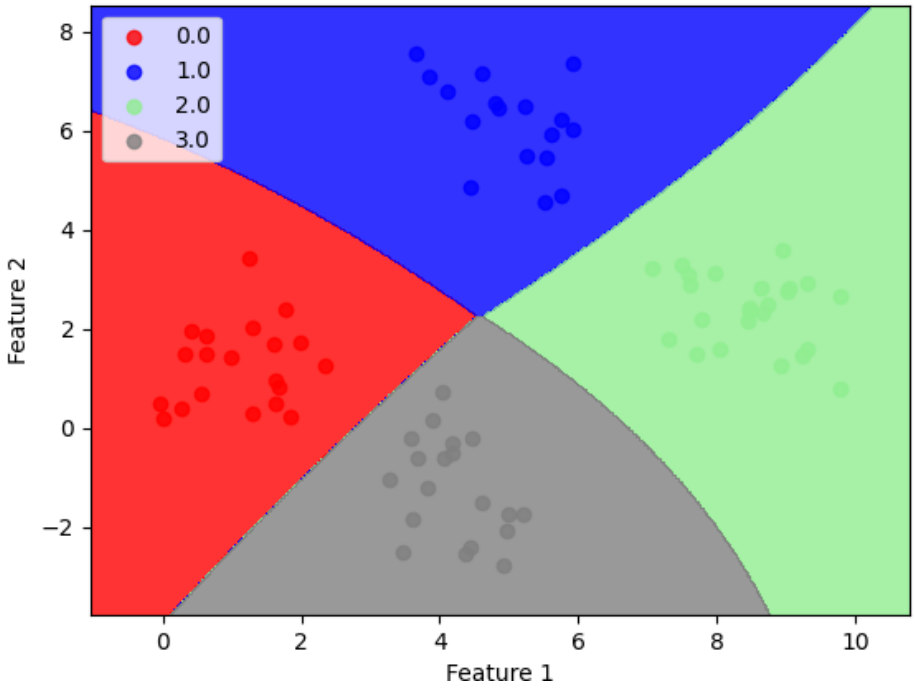
Результат виконання



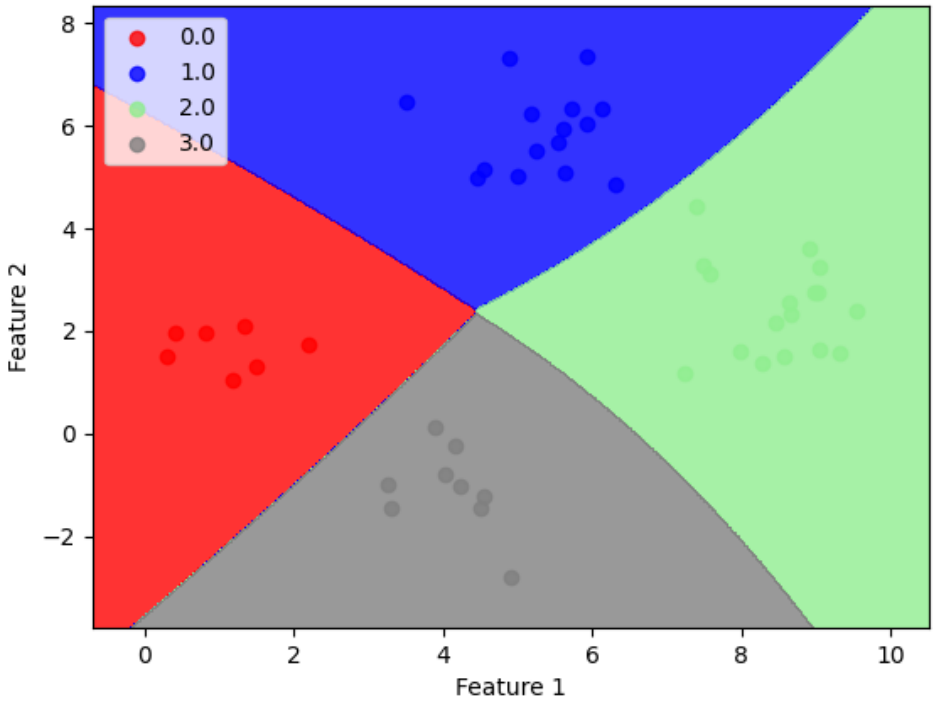
Код, в який додано перехресну перевірку, розбивку даних на тренувальний і тестовий набори, а також обчислення якості, точності, повноти та F1-міри класифікатора

```
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 matplotlib.colors import ListedColormap

def visualize_classifier(classifier, X, y):
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1

    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02),
                         np.arange(y_min, y_max, 0.02))

    Z = classifier.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    plt.contourf(xx, yy, Z, alpha=0.8, cmap=ListedColormap(('red', 'blue',
'lightgreen', 'gray')))

    for idx, cl in enumerate(np.unique(y)):
        plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],
                    alpha=0.8, c=ListedColormap(('red', 'blue', 'light-
green', 'gray'))(idx),
                    marker='o', label=cl)

    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.legend(loc='upper left')
    plt.show()

input_file = 'data_multivar_nb.txt'

data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1]
```

```
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)

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

visualize_classifier(classifier_new, X_test, y_test)

num_folds = 3

accuracy_values = cross_val_score(classifier_new, X, y, scoring='accuracy',
cv=num_folds)
print("Accuracy: " + str(round(100 * accuracy_values.mean(), 2)) + "%")
precision_values = cross_val_score(classifier_new, X, y, scoring='preci-
sion_weighted', cv=num_folds)
print("Precision: " + str(round(100 * precision_values.mean(), 2)) + "%")
recall_values = cross_val_score(classifier_new, X, y, scoring='re-
call_weighted', cv=num_folds)
print("Recall: " + str(round(100 * recall_values.mean(), 2)) + "%")
f1_values = cross_val_score(classifier_new, X, y, scoring='f1_weighted',
cv=num_folds)
print("F1: " + str(round(100 * f1_values.mean(), 2)) + "%")
```



Перший прогін



Другий прогін

Завдання 2.5. Вивчити метрики якості класифікації

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

```python
from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score

# Завантаження даних з файлу
df = pd.read_csv('data_metrics.csv')

# Виведення перших декількох рядків для перевірки
print(df.head())

# Встановлення порогу для прогнозування
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())

def find_TP(y_true, y_pred):
    # Кількість True Positives (y_true = 1, y_pred = 1)
    return sum((y_true == 1) & (y_pred == 1))

def find_FN(y_true, y_pred):
    # Кількість False Negatives (y_true = 1, y_pred = 0)
    return sum((y_true == 1) & (y_pred == 0))

def find_FP(y_true, y_pred):
    # Кількість False Positives (y_true = 0, y_pred = 1)
    return sum((y_true == 0) & (y_pred == 1))

def find_TN(y_true, y_pred):
    # Кількість True Negatives (y_true = 0, y_pred = 0)
    return sum((y_true == 0) & (y_pred == 0))

# Перевірка результатів
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):
    # 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 mischenchuk_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]])

mischenchuk_confusion_matrix(df.actual_label.values, df.predicted_RF.val-
ues)

assert np.array_equal(mischenchuk_confusion_matrix(df.actual_label.values,
df.predicted_RF.values), confusion_matrix(df.actual_label.values, df.pre-
dicted_RF.values) ), 'mischenchuk_confusion_matrix() is not correct for RF'
assert np.array_equal(mischenchuk_confusion_matrix(df.actual_label.values,
df.predicted_LR.values),confusion_matrix(df.actual_label.values, df.pre-
dicted_LR.values) ), 'mischenchuk_confusion_matrix() is not correct for LR'

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

def mischenchuk_accuracy_score(y_true, y_pred):
    TP,FN,FP,TN = find_conf_matrix_values(y_true,y_pred)
    return (TP + TN) / (TP + FN + FP + TN) # як по формулі

assert mischenchuk_accuracy_score(df.actual_label.values, df.pre-
dicted_RF.values) == accuracy_score(df.actual_label.values, df.pre-
dicted_RF.values), 'my_accuracy_score failed on'
assert mischenchuk_accuracy_score(df.actual_label.values, df.pre-
dicted_LR.values) == accuracy_score(df.actual_label.values, df.pre-
dicted_LR.values), 'my_accuracy_score failed on LR'
print('Accuracy RF: %.3f'%(mischenchuk_accuracy_score(df.actual_label.val-
ues, df.predicted_RF.values)))
print('Accuracy LR: %.3f'%(mischenchuk_accuracy_score(df.actual_label.val-
ues, df.predicted_LR.values)))

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

```
def mischenchuk_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) # як по формулі
assert mischenchuk_recall_score(df.actual_label.values, df.pre-
dicted_RF.values) == recall_score(df.actual_label.values, df.pre-
dicted_RF.values), 'my_accuracy_score failed on RF'
assert mischenchuk_recall_score(df.actual_label.values, df.pre-
dicted_LR.values) == recall_score(df.actual_label.values, df.pre-
dicted_LR.values), 'my_accuracy_score failed on LR'
print('Recall RF: %.3f'%(mischenchuk_recall_score(df.actual_label.values,
df.predicted_RF.values)))
print('Recall LR: %.3f'%(mischenchuk_recall_score(df.actual_label.values,
df.predicted_LR.values)))

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

def mischenchuk_precision_score(y_true, y_pred):
    # calculates the fraction of predicted positives samples that are actu-
ally positive
    TP,FN,FP,TN = find_conf_matrix_values(y_true,y_pred)
    return TP / (TP + FP) # як по формулі
assert mischenchuk_precision_score(df.actual_label.values, df.pre-
dicted_RF.values) == precision_score(df.actual_label.values, df.pre-
dicted_RF.values), 'my_accuracy_score failed on RF'
assert mischenchuk_precision_score(df.actual_label.values, df.pre-
dicted_LR.values) == precision_score(df.actual_label.values, df.pre-
dicted_LR.values), 'my_accuracy_score failed on LR'
print('Precision RF: %.3f'%(mischenchuk_precision_score(df.actual_la-
bel.values, df.predicted_RF.values)))
print('Precision LR: %.3f'%(mischenchuk_precision_score(df.actual_la-
bel.values, df.predicted_LR.values)))

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

def mischenchuk_f1_score(y_true, y_pred):
    # calculates the F1 score
    recall = mischenchuk_recall_score(y_true, y_pred)
    precision = mischenchuk_precision_score(y_true, y_pred)
```

```
    return 2 * (recall * precision) / (recall + precision) # як по формулі
assert mischenchuk_f1_score(df.actual_label.values, df.predicted_RF.values)
== f1_score(df.actual_label.values, df.predicted_RF.values), 'mischen-
chuk_accuracy_score failed on RF'
assert mischenchuk_f1_score(df.actual_label.values, df.predicted_LR.values)
== f1_score(df.actual_label.values, df.predicted_LR.values), 'mischen-
chuk_accuracy_score failed on LR'
print('F1 RF: %.3f'%(mischenchuk_f1_score(df.actual_label.values, df.pre-
dicted_RF.values)))
print('F1 LR: %.3f'%(mischenchuk_f1_score(df.actual_label.values, df.pre-
dicted_LR.values)))



print('')
print('scores with threshold = 0.5')
print('Accuracy RF: %.3f'%(mischenchuk_accuracy_score(df.actual_label.val-
ues, df.predicted_RF.values)))
print('Recall RF: %.3f'%(mischenchuk_recall_score(df.actual_label.values,
df.predicted_RF.values)))
print('Precision RF: %.3f'%(mischenchuk_precision_score(df.actual_la-
bel.values, df.predicted_RF.values)))
print('F1 RF: %.3f'%(mischenchuk_f1_score(df.actual_label.values, df.pre-
dicted_RF.values)))
print('')
print('scores with threshold = 0.25')
print('Accuracy RF: %.3f'%(mischenchuk_accuracy_score(df.actual_label.val-
ues, (df.model_RF >= 0.25).astype('int').values)))
print('Recall RF: %.3f'%(mischenchuk_recall_score(df.actual_label.values,
(df.model_RF >= 0.25).astype('int').values)))
print('Precision RF: %.3f'%(mischenchuk_precision_score(df.actual_la-
bel.values, (df.model_RF >= 0.25).astype('int').values)))
print('F1 RF: %.3f'%(mischenchuk_f1_score(df.actual_label.values,
(df.model_RF >= 0.25).astype('int').values)))
```

Без непотрібних виводів даних, що повторюються, ми отримаємо такий результат:

```
   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
```

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

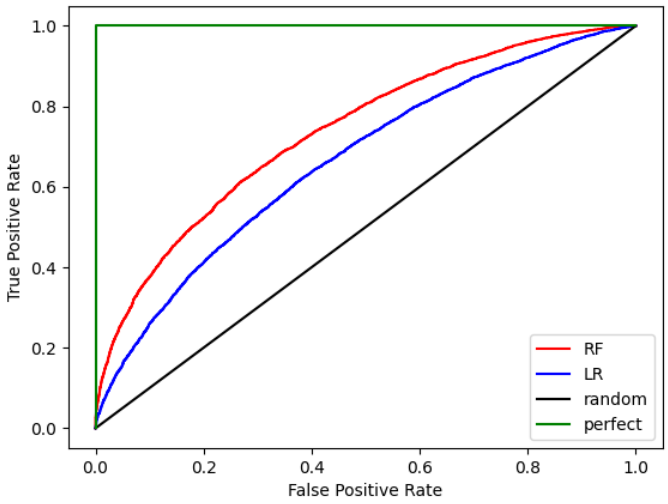
```
scores with threshold = 0.5
Accuracy RF: 0.671
Recall RF: 0.641
Precision RF: 0.681
F1 RF: 0.660

scores with threshold = 0.25
Accuracy RF: 0.502
Recall RF: 1.000
Precision RF: 0.501
F1 RF: 0.668
```

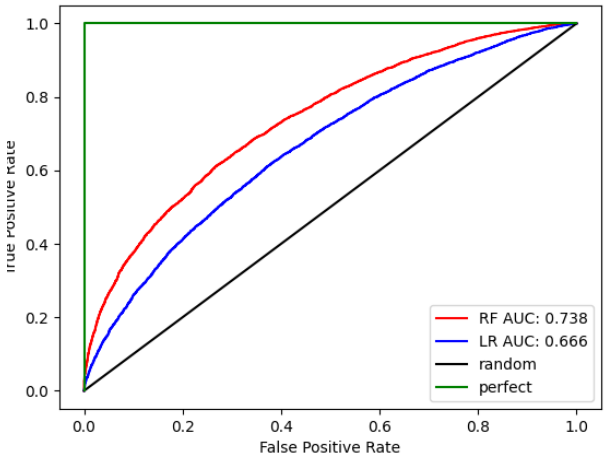
Отже, коли поріг встановлено на 0.5, точність, повнота, точність та F1-оцінка досягають ідеального значення 1.0. Це свідчить про те, що модель правильно класифікувала всі приклади з вибраного для оцінки датасету. Однак, при зниженні порогу до 0.25, точність зменшується, хоча повнота залишається на рівні 1.0. Це означає, що модель виявляє значно більше позитивних прикладів, включаючи помилкові виявлення, що призводить до зменшення точності. F1-оцінка також зменшується порівняно з порогом 0.5, що демонструє загальне погіршення балансу між точністю та повнотою.

Доданий код з ROC:

```
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()
```



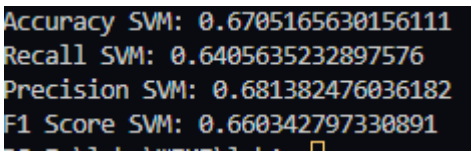


Завдання 2.6. Розробіть програму класифікації даних в файлі data_multivar_nb.txt за допомогою машини опорних векторів (Support Vector Machine - SVM).

Лістинг програми:

```
from sklearn.svm import SVC
from lab1_task5 import f1_score
from lab1_task5 import precision_score
from lab1_task5 import recall_score
from lab1_task5 import accuracy_score
from lab1_task5 import df

X = df.drop(columns=['actual_label'])
y = df['actual_label']

svm_model = SVC()

svm_model.fit(X, y)

y_pred_svm = svm_model.predict(X)

accuracy_svm = accuracy_score(y, y_pred_svm)

recall_svm = recall_score(y, y_pred_svm)

precision_svm = precision_score(y, y_pred_svm)

f1_svm = f1_score(y, y_pred_svm)

print('Accuracy SVM:', accuracy_svm)
print('Recall SVM:', recall_svm)
print('Precision SVM:', precision_svm)
print('F1 Score SVM:', f1_svm)
```

Результат виконання



Результати для SVM та наївного байєсівського класифікатора (NB) показують однакові показники точності, повноти, точності та F1-оцінки. Обидві моделі демонструють однакову ефективність. Однак, вибір моделі залежить від контексту задачі. Незважаючи на те, що в обох випадках ми маємо 100% точність, повноту, точність і F1-оцінку, вибір моделі може залежати від конкретних вимог і особливостей задачі. Наприклад, наївний байєсівський класифікатор може бути швидшим у навчанні та прогнозуванні порівняно з SVM, що може бути важливим фактором при обробці великих обсягів даних або в реальному часі.

Посилання на GitHub - https://github.com/MischenchukMykola/lab1

Висновок: виконуючи цю лабораторну роботу я використовуючи спеціалізовані бібліотеки та мову програмування Python дослідив попередню обробку та класифікацію даних**.**