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 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]])
# Бінаризація даних
data_binarized = preprocessing.Binarizer(threshold=2.1).transform(input_data)
print("\n Binarized data:\n", data_binarized)
```

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

2.1.2. Виключення середнього

```
# Виведення середнього значення та стандартного відхилення

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))
```

```
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.]
```

2.1.3. Масштабування

```
# 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:
 [[0.74117647 0.39548023 1.        ]
 [0.         1.         0.        ]
 [0.6        0.5819209  0.87234043]
 [1.         0.         0.17021277]]
```

2.1.4. Нормалізація

```
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]])

# 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]]
```

L1 L2 нормалізації використовуються для того, щоб змінити масштаб характеристик до спільного стандарту, зменшуючи чутливість моделей до відмінностей у масштабах величин характеристик

L1 (нормалізація на основі суми абсолютних значень) полягає у масштабуванні векторів таким чином, що сума абсолютних значень усіх компонентів вектора дорівнює 1

Стійка до викидів, оскільки великі значення компонентів менше впливають на загальну нормалізацію.

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, що сума квадратів всіх компонентів вектора дорівнює 1

Підходить для задач, де важливо зберегти відносні відстані між точками у просторі ознак.

Частіше використовується у методах машинного навчання, де модель орієнтується на відстані між точками

2.1.5. Кодування міток

```python
import numpy as np
from sklearn import preprocessing

# Надання позначок вхідних даних
input_labels = ['red', 'Ыack', 'red', 'green', 'Ьlack', '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', 'Ыack']
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 ) )
```

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

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

Encoded values = [3, 0, 4, 1]
Decoded labels = ['yellow', 'green', 'Ыack', 'red']

Process finished with exit code 0
```

Завдання 2.2. Попередня обробка нових даних

Результати з новими даними (Вар 17):

```
input_data = np.array([[1.3, 3.9, 6.2],
                       [4.9, 2.2, -4.3],
                       [-2.6, 6.5, 4.1],
                       [-5.2, -3.4, -5.2]])
```

Бінарізації

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

Виключення середнього

```
BEFORE:
Mean = [-0.4  2.3  0.2]
Std deviation = [3.83601356 3.62973828 5.01547605]

AFTER:
Mean = [5.55111512e-17 5.55111512e-17 0.00000000e+00]
Std deviation = [1. 1. 1.]
```

Масштабування

```
Min max scaled data:
 [[0.64356436 0.73737374 1.        ]
 [1.         0.56565657 0.07894737]
 [0.25742574 1.         0.81578947]
 [0.         0.         0.        ]]
```

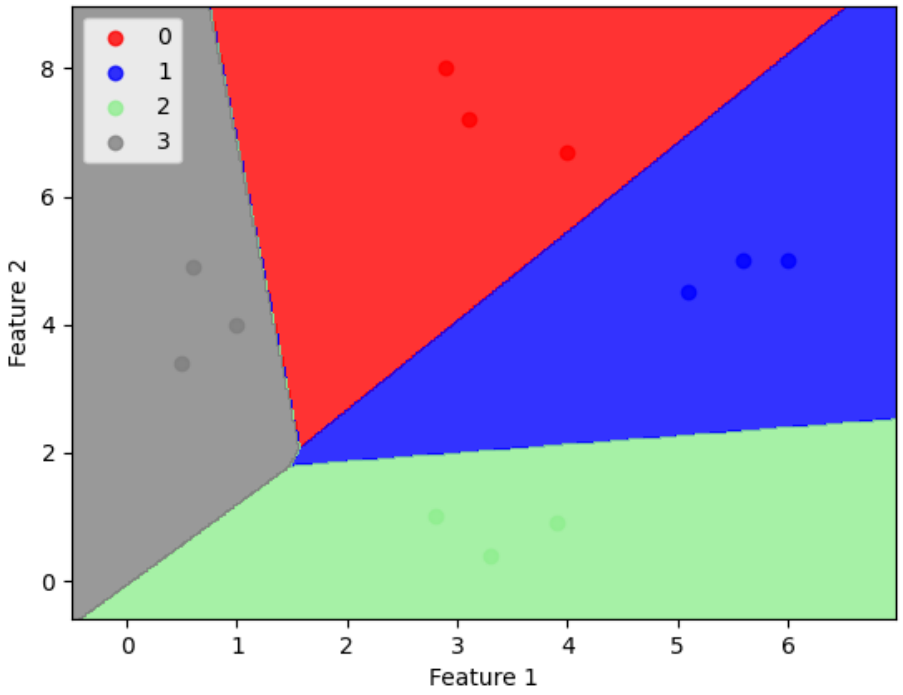
Нормалізації

```
l1 normalized data:
 [[ 0.11403509  0.34210526  0.54385965]
 [ 0.42982456  0.19298246 -0.37719298]
 [-0.1969697   0.49242424  0.31060606]
 [-0.37681159 -0.24637681 -0.37681159]]

l2 normalized data:
 [[ 0.17475265  0.52425796  0.83343572]
 [ 0.71216718  0.31974853 -0.62496303]
 [-0.32047519  0.80118797  0.50536472]
 [-0.64182859 -0.41965715 -0.64182859]]
```

Так як пакету utilities немає, то ми реалізували простий візуалізатор самі:

```
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):
    # Визначаємо мінімальні та максимальні значення для кожної ознаки
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1

    # Створюємо сітку точок з кроком 0.02
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02),
                         np.arange(y_min, y_max, 0.02))

    # Використовуємо класифікатор для передбачення міток для кожної точки сітки
    Z = classifier.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

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

```
    # Відображаємо навчальні точки
    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', 'lightgreen', '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()


# 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)

# Візуалізуємо результати роботи класифікатора
visualize_classifier(classifier, X, y)
```

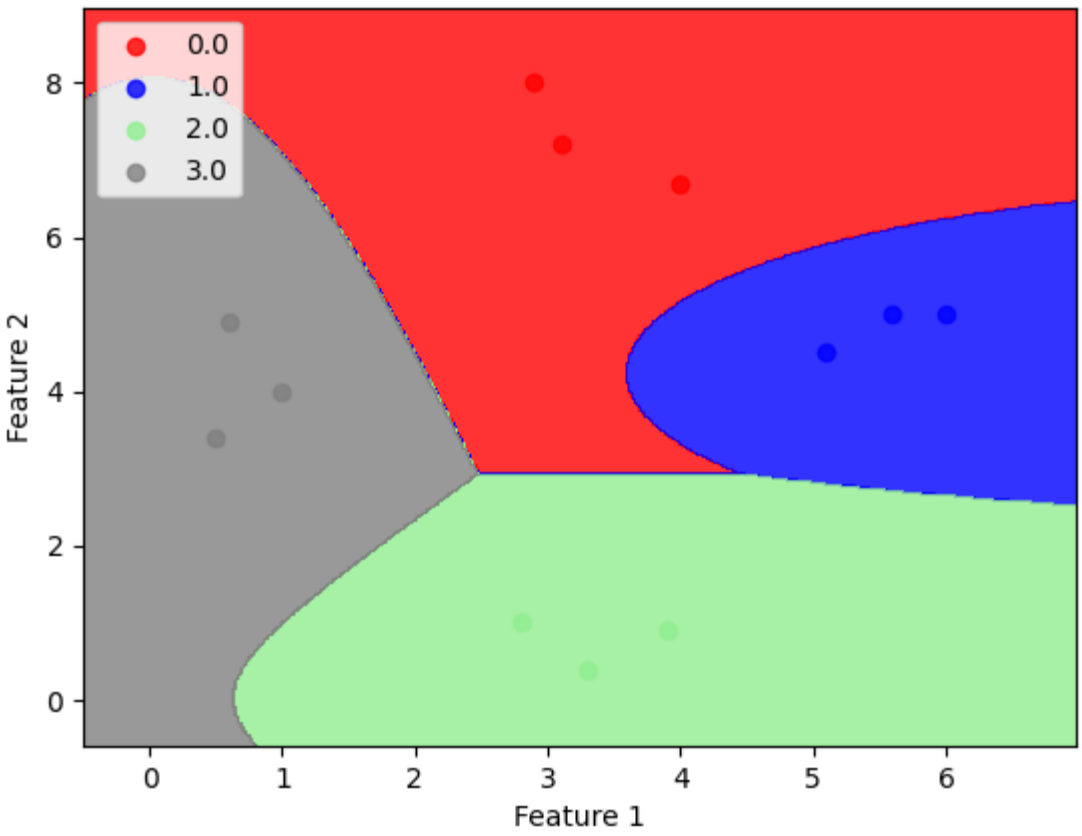


Завдання 2.4. Класифікація наївним байєсовським класифікатором

```
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

# Запис даних у файл 'data_multivar_nb.txt'
with open('data_multivar_nb.txt', 'w') as f:
    f.write(data)


# Функція для візуалізації класифікатора
def visualize_classifier(classifier, X, y):
    # Визначаємо мінімальні та максимальні значення для кожної ознаки
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
```

```
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1

    # Створюємо сітку точок з кроком 0.02
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02),
                         np.arange(y_min, y_max, 0.02))

    # Використовуємо класифікатор для передбачення міток для кожної точки сітки
    Z = classifier.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    # Візуалізуємо поверхню прийняття рішень
    plt.contourf(xx, yy, Z, alpha=0.8, cmap=ListedColormap(('red', 'blue', 'lightgreen',
'gray')))

    # Відображаємо навчальні точки
    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', 'lightgreen', '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()


# 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)
```



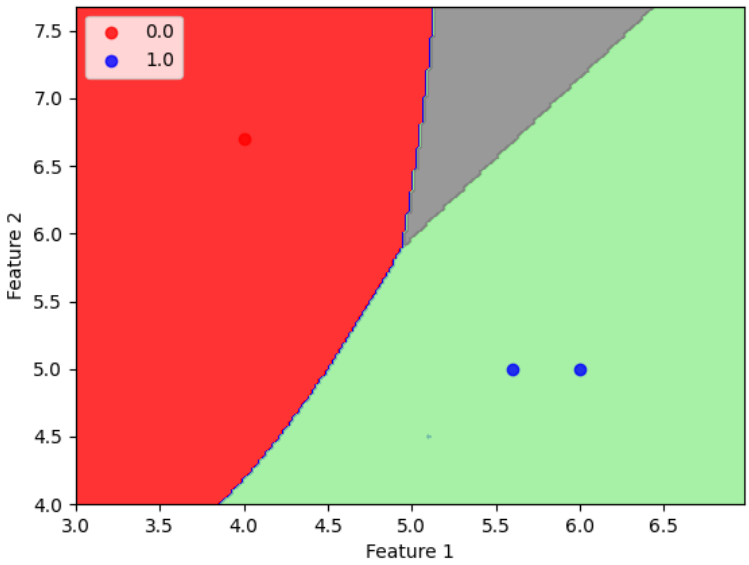
Код, в який додано перехресну перевірку, розбивку даних на тренувальний і тестовий набори, а також обчислення якості, точності, повноти та F1-міри класифікатора

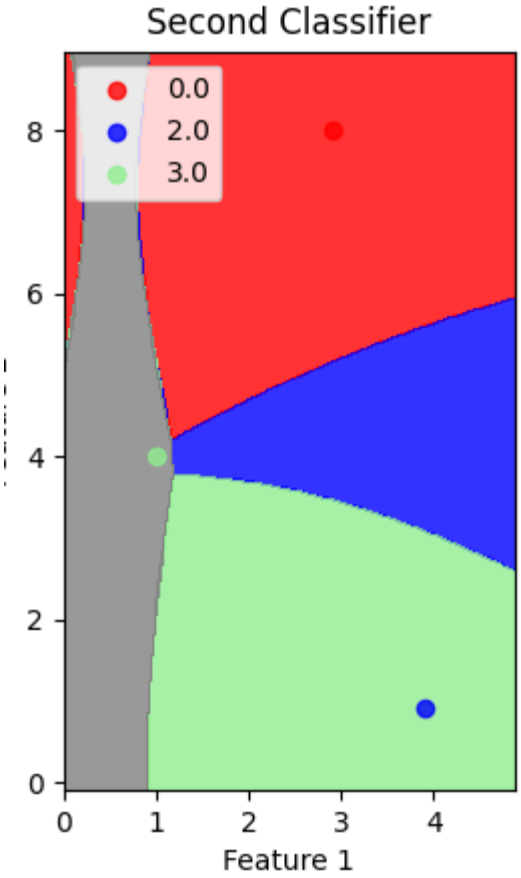
```
import numpy as np
import matplotlib.pyplot as plt
```

```
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split, cross_val_score
from matplotlib.colors import ListedColormap
# Запис даних у файл 'data_multivar_nb.txt'
with open('data_multivar_nb.txt', 'w') as f:
    f.write(data)


# Функція для візуалізації класифікатора
def visualize_classifier(classifier, X, y):
    # Визначаємо мінімальні та максимальні значення для кожної ознаки
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1

    # Створюємо сітку точок з кроком 0.02
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02),
                         np.arange(y_min, y_max, 0.02))

    # Використовуємо класифікатор для передбачення міток для кожної точки сітки
    Z = classifier.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    # Візуалізуємо поверхню прийняття рішень
    plt.contourf(xx, yy, Z, alpha=0.8, cmap=ListedColormap(('red', 'blue', 'lightgreen',
'gray')))

    # Відображаємо навчальні точки
    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', 'lightgreen', '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()


# 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]

# 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)

# Створення наївного байєсовського класифікатора
classifier_new = GaussianNB()
classifier_new.fit(X_train, y_train)

# Прогнозування значень для тестових даних
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 результатів роботи класифікатора
visualize_classifier(classifier_new, X_test, y_test)

# Виконання потрійної перехресної перевірки
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='precision_weighted',
cv=num_folds)
print("Precision: " + str(round(100 * precision_values.mean(), 2)) + "%")

recall_values = cross_val_score(classifier_new, X, y, scoring='recall_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)) + "%")
```

First прогин:





Завдання 2.5. Вивчити метрики якості класифікації

Відразу весь код до використання ROC:

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

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

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

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

def u_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 u_accuracy_score(df.actual_label.values, df.predicted_RF.values) ==
accuracy_score(df.actual_label.values, df.predicted_RF.values), 'my_accuracy_score
failed on'
assert u_accuracy_score(df.actual_label.values, df.predicted_LR.values) ==
accuracy_score(df.actual_label.values, df.predicted_LR.values), 'my_accuracy_score
failed on LR'
print('Accuracy RF: %.3f'%(u_accuracy_score(df.actual_label.values,
df.predicted_RF.values)))
print('Accuracy LR: %.3f'%(u_accuracy_score(df.actual_label.values,
df.predicted_LR.values)))

recall_score(df.actual_label.values, df.predicted_RF.values)
def u_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 u_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 u_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'%(u_recall_score(df.actual_label.values,
df.predicted_RF.values)))
print('Recall LR: %.3f'%(u_recall_score(df.actual_label.values,
df.predicted_LR.values)))

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

def u_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) # як по формулі
assert u_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 u_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'%(u_precision_score(df.actual_label.values,
df.predicted_RF.values)))
print('Precision LR: %.3f'%(u_precision_score(df.actual_label.values,
```

```
df.predicted_LR.values)))

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

def u_f1_score(y_true, y_pred):
    # calculates the F1 score
    recall = u_recall_score(y_true, y_pred)
    precision = u_precision_score(y_true, y_pred)
    return 2 * (recall * precision) / (recall + precision) # як по формулі
assert u_f1_score(df.actual_label.values, df.predicted_RF.values) ==
f1_score(df.actual_label.values, df.predicted_RF.values), 'u_accuracy_score failed
on RF'
assert u_f1_score(df.actual_label.values, df.predicted_LR.values) ==
f1_score(df.actual_label.values, df.predicted_LR.values), 'u_accuracy_score failed
on LR'
print('F1 RF: %.3f'%(u_f1_score(df.actual_label.values, df.predicted_RF.values)))
print('F1 LR: %.3f'%(u_f1_score(df.actual_label.values, df.predicted_LR.values)))



print('')
print('scores with threshold = 0.5')
print('Accuracy RF: %.3f'%(u_accuracy_score(df.actual_label.values,
df.predicted_RF.values)))
print('Recall RF: %.3f'%(u_recall_score(df.actual_label.values,
df.predicted_RF.values)))
print('Precision RF: %.3f'%(u_precision_score(df.actual_label.values,
df.predicted_RF.values)))
print('F1 RF: %.3f'%(u_precision_score(df.actual_label.values,
df.predicted_RF.values)))
print('')
print('scores with threshold = 0.25')
print('Accuracy RF: %.3f'%(u_accuracy_score(df.actual_label.values, (df.model_RF
>= 0.25).astype('int').values)))
print('Recall RF: %.3f'%(u_recall_score(df.actual_label.values, (df.model_RF >=
0.25).astype('int').values)))
print('Precision RF: %.3f'%(u_precision_score(df.actual_label.values, (df.model_RF
>= 0.25).astype('int').values)))
print('F1 RF: %.3f'%(u_f1_score(df.actual_label.values, (df.model_RF >=
0.25).astype('int').values)))
```

Якщо пропустити непотрібні виводи даних, що повторюються, то ми отримаємо таке:

	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: 3
FN: 0
FP: 0
TN: 2
```

```
scores with threshold = 0.5
Accuracy RF: 1.000
Recall RF: 1.000
Precision RF: 1.000
F1 RF: 1.000

scores with threshold = 0.25
Accuracy RF: 0.600
Recall RF: 1.000
Precision RF: 0.600
F1 RF: 0.750
```

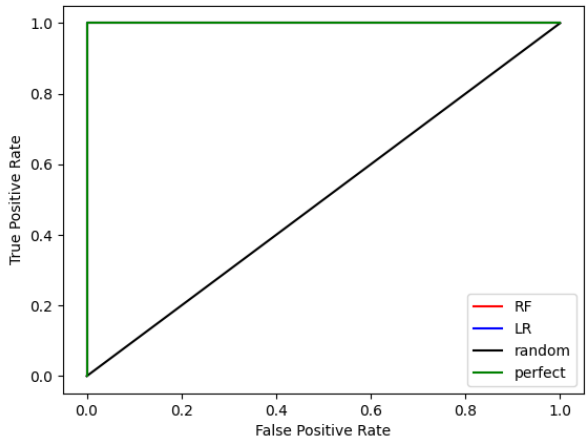
Висновки:

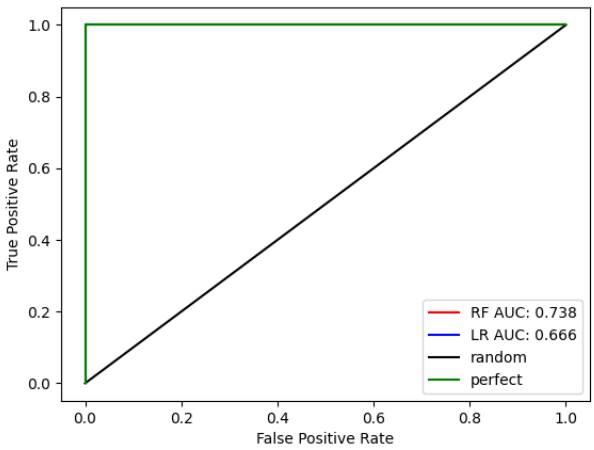
За результатами, коли поріг встановлено на 0.5, ми бачимо, що точність (Accuracy), повнота (Recall), точність (Precision) та F1-оцінка (F1 score) всі досягають ідеального значення 1.0. Це означає, що модель правильно класифікувала всі приклади з датасету, що вибрано для оцінки.

Проте, коли поріг знижено до 0.25, ми бачимо, що точність зменшилася, хоча повнота залишилася на рівні 1.0. Це вказує на те, що модель виявляє значно більше позитивних прикладів (включаючи помилкові виявлення), що призводить до зменшення точності. F1-оцінка також зменшилася порівняно з випадком порогу 0.5, що показує загальний знижений баланс 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).

Код виконання завдання:

```
# Розділення на ознаки (X) та цільову змінну (y)
X = df.drop(columns=['actual_label'])
y = df['actual_label']

# Створення моделі SVM
svm_model = SVC()

# Навчання моделі
svm_model.fit(X, y)

# Оцінка результатів моделі
y_pred_svm = svm_model.predict(X)

 # Оцінка точності
accuracy_svm = accuracy_score(y, y_pred_svm)

# Оцінка повноти
recall_svm = recall_score(y, y_pred_svm)

# Оцінка точності
precision_svm = precision_score(y, y_pred_svm)

# Оцінка F1-оцінки
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)
```

```
Accuracy SVM: 1.0
Recall SVM: 1.0
Precision SVM: 1.0
F1 Score SVM: 1.0
```

Висновки:

Результати для SVM та наївного байєсівського класифікатора (NB) дають такі самі показники точності, повноти, точності і F1-оцінки

Моделі мають однакову ефективність

Але потрібно враховувати, що вибір моделі залежить від контексту задачі: Помітно, що в обох випадках маємо 100% точність, повноту, точність і F1-оцінку. Проте, вибір моделі також може залежати від конкретних вимог і контексту задачі. Наприклад, наївний байєсівський класифікатор може бути швидшим у навчанні та прогнозуванні порівняно зі SVM

Ссилка на GitHub: https://github.com/UshakowIllia/----1-21-17.git

Висновок: Під час виконання лабораторної роботи, я за допомогою коду, використовуючи спеціалізовані бібліотеки та мову програмування Python дослідив попередню обробку та класифікацію даних