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. - 2.1.4.

```
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(f"\nBinarized data:\n{data_binarized}")

# Виведення середнього значення та стандартного відхилення
print("\nBEFORE: ")
print(f"Mean = {input_data.mean(axis=0)}")
print(f"Std deviation = {input_data.std(axis=0)}")

# Исключение среднего
data_scaled = preprocessing.scale(input_data)
print("\nAFTER: ")
print(f"Mean = {data_scaled.mean(axis=0)}")
print(f"Std deviation = {data_scaled.std(axis=0)}")

# Масштабування MinMax
data_scaled_minmax = preprocessing.MinMaxScaler(feature_range=(0,1))
data_scaled_minmax = data_scaled_minmax.fit_transform(input_data)
print(f"\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(f"\nL1 normalized data:\n{data_normalized_l1}")
print(f"\nL2 normalized data:\n{data_normalized_l2}")
```

Результат виконання:

```
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]]
```

Рис.1 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]]
```

Рис.2 Результат виконання

Завдання 2.1.5.

```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(f"\nLabels = {test_labels}")
print(f"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(f"\nEncoded values = {encoded_values}")
print(f"Decoded labels = {list(decoded_list)}")
```

Результат виконання:

```
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']
```

Рис.3 Результат виконання

Завдання 2.2

6.	2.3	-1.6	6.1	-2.4	-1.2	4.3	3.2	5.5	-6.1	-4.4	1.4	-1.2	2.1

```
import numpy as np
from sklearn import preprocessing

input_data = np.array([[2.3, -1.6, 6.1],
                       [-2.4, -1.2, 4.3],
                       [3.2, 5.5, -6.1],
                        [-4.4, 1.4, -1.2]])
# Бінаризація даних
data_binarized = preprocessing.Binarizer(threshold=2.1).transform(input_data)
print(f"\nBinarized data:\n{data_binarized}")

# Виведення середнього значення та стандартного відхилення
print("\nBEFORE: ")
print(f"Mean = {input_data.mean(axis=0)}")
```

```
print(f"Std deviation = {input_data.std(axis=0)}")

# Исключение среднего
data_scaled = preprocessing.scale(input_data)
print("\nAFTER: ")
print(f"Mean = {data_scaled.mean(axis=0)}")
print(f"Std deviation = {data_scaled.std(axis=0)}")

# Масштабування MinMax
data_scaled_minmax = preprocessing.MinMaxScaler(feature_range=(0,1))
data_scaled_minmax = data_scaled_minmax.fit_transform(input_data)
print(f"\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(f"\nL1 normalized data:\n{data_normalized_l1}")
print(f"\nL2 normalized data:\n{data_normalized_l2}")
```

Результат виконання:

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

BEFORE:
Mean = [-0.325  1.025  0.775]
Std deviation = [3.17125764 2.82875856 4.79446295]

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

Min max scaled data:
[[0.88157895 0.         1.        ]
 [0.26315789 0.05633803 0.85245902]
 [1.         1.         0.        ]
 [0.         0.42253521 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.4 Результат виконання

Завдання 2.3

utilities.py

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

def visualize_classifier(classifier, X, y):
    # Define the minimum and maximum values for X and Y
    # that will be used in the mesh grid
    min_x, max_x = X[:, 0].min() - 1.0, X[:, 0].max() + 1.0
    min_y, max_y = X[:, 1].min() - 1.0, X[:, 1].max() + 1.0

    # Define the step size to use in plotting the mesh grid
    mesh_step_size = 0.01

    # Define the mesh grid of X and Y values
    x_vals, y_vals = np.meshgrid(np.arange(min_x, max_x, mesh_step_size),
np.arange(min_y, max_y, mesh_step_size))

    # Run the classifier on the mesh grid
    output = classifier.predict(np.c_[x_vals.ravel(), y_vals.ravel()])

    # Reshape the output array
    output = output.reshape(x_vals.shape)

    # Create a plot
    plt.figure()

    # Choose a color scheme for the plot
    plt.pcolormesh(x_vals, y_vals, output, cmap=plt.cm.gray)

    # Overlay the training points on the plot
    plt.scatter(X[:, 0], X[:, 1], c=y, s=75, edgecolors='black', linewidth=1,
cmap=plt.cm.Paired)

    # Specify the boundaries of the plot
    plt.xlim(x_vals.min(), x_vals.max())
    plt.ylim(y_vals.min(), y_vals.max())

    # Specify the ticks on the X and Y axes
    plt.xticks((np.arange(int(X[:, 0].min() - 1), int(X[:, 0].max() + 1), 1.0)))
    plt.yticks((np.arange(int(X[:, 1].min() - 1), int(X[:, 1].max() + 1), 1.0)))

    plt.show()
```

LR_1_task_3.py

```
import numpy as np
from sklearn import linear_model
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)
```

Результат виконання:

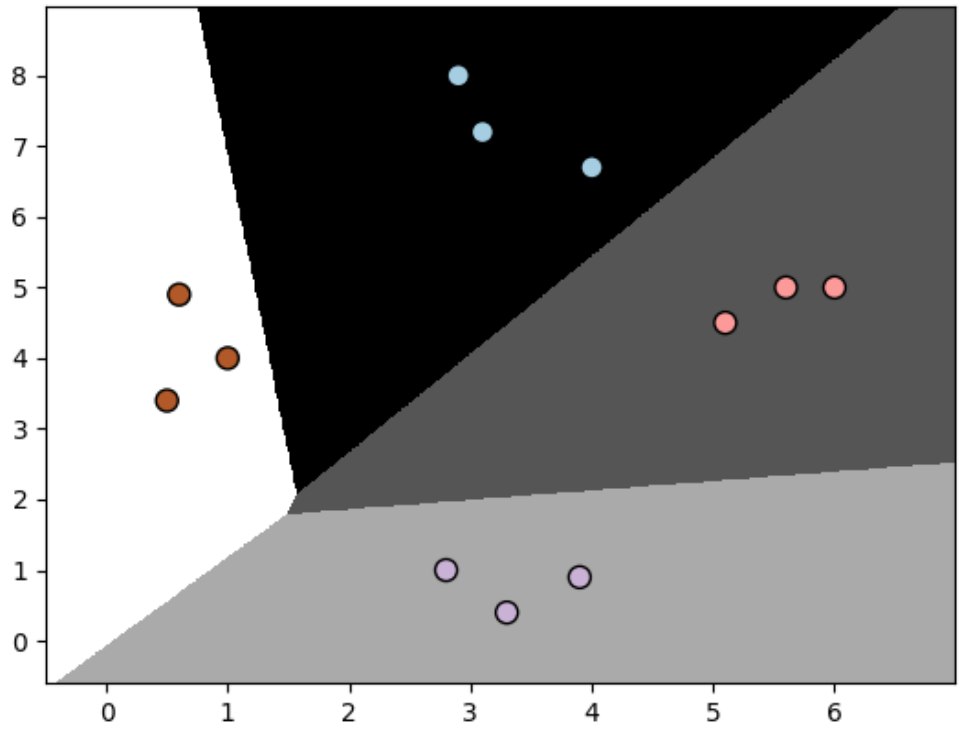

Рис.5 Результат виконання

Завдання 2.4

```
import numpy as np
from utilities import visualize_classifier
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split

# 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(f"Accuracy of Naive Bayes classifier = {round(accuracy,2)}%")

# Візуалізація результатів роботи класифікатора
visualize_classifier(classifier, X, y)
```

Результат виконання:

```
Python 3.9.5 (tags/v3.9.5:0a7dcbd, May  3 2021, 17:27:52) [MSC v.1928 64 bit (AMD64)]
Accuracy of Naive Bayes classifier = 99.75%
```

Рис.6 Результат виконання

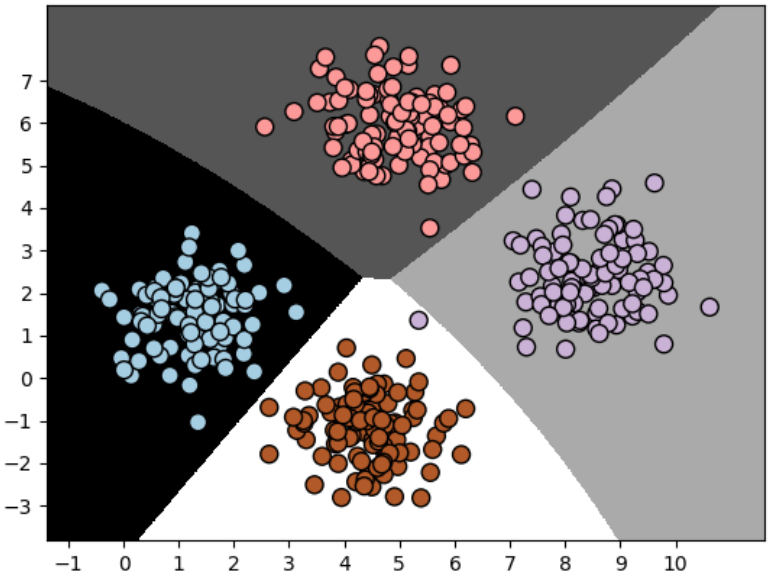

Рис.7 Результат виконання

```
import numpy as np
from utilities import visualize_classifier
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import 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(f"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(f"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='precision_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)) + "%")
```

```
Accuracy of Naive Bayes classifier = 99.75%
Accuracy of the new classifier = 100.0%
Accuracy: 99.75%
Precision: 99.76%
Recall: 99.76%
F1: 99.75%
```

Рис.8 Результат виконання

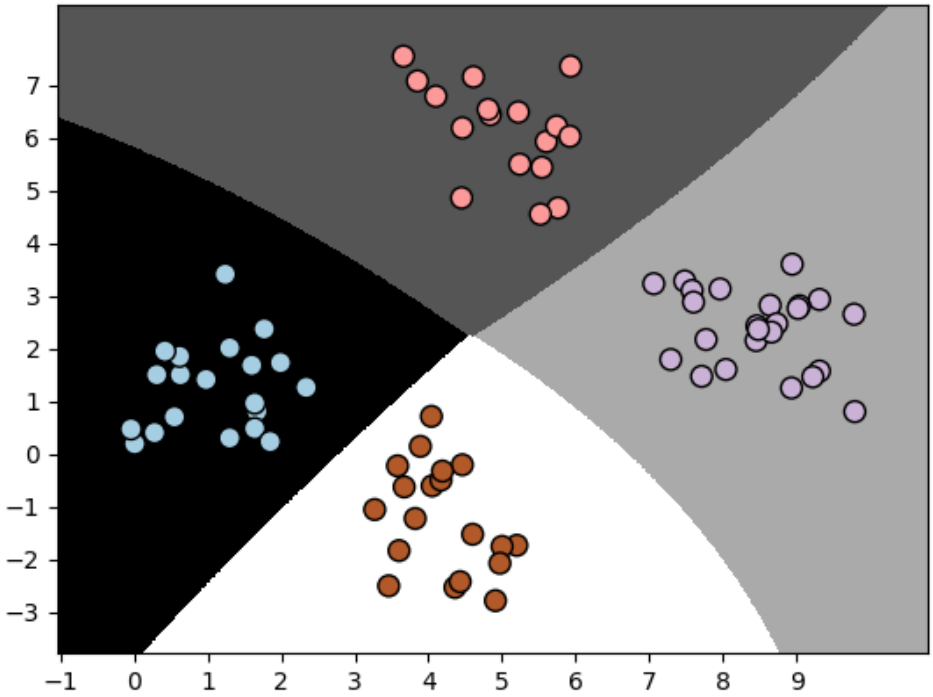

Рис.9 Результат виконання

Після того як ми виконали перехресну перевірку та розбили дані на тестові та тренувальні точність підвищилась до 100%.

Завдання 2.5

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

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
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score

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()

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))


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 saukh_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]])


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

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

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


def saukh_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 saukh_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
RF'
assert saukh_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' % (saukh_accuracy_score(df.actual_label.values,
df.predicted_RF.values)))

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


def saukh_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)


assert saukh_recall_score(df.actual_label.values, df.predicted_RF.values) ==
recall_score(df.actual_label.values,

df.predicted_RF.values), 'saukh_accuracy_score failed on RF'
assert saukh_recall_score(df.actual_label.values, df.predicted_LR.values) ==
recall_score(df.actual_label.values,

df.predicted_LR.values), 'saukh_accuracy_score failed on LR'

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

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


def saukh_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)


assert saukh_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 saukh_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' % (saukh_precision_score(df.actual_label.values,
df.predicted_RF.values)))
print('Precision LR: %.3f' % (saukh_precision_score(df.actual_label.values,
df.predicted_LR.values)))

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


def saukh_f1_score(y_true, y_pred):  # calculates the F1 score
    recall = saukh_recall_score(y_true, y_pred)
    precision = saukh_precision_score(y_true, y_pred)
    return 2 * (precision * recall) / (precision + recall)


assert saukh_f1_score(df.actual_label.values, df.predicted_RF.values) ==
```

```
f1_score(df.actual_label.values,

df.predicted_RF.values), 'my_accuracy_score failed on RF'
assert saukh_f1_score(df.actual_label.values, df.predicted_LR.values) ==
f1_score(df.actual_label.values,

df.predicted_LR.values), 'my_accuracy_score failed on LR'

print('F1 RF: %.3f' % (saukh_f1_score(df.actual_label.values,
df.predicted_RF.values)))
print('F1 LR: %.3f' % (saukh_f1_score(df.actual_label.values,
df.predicted_LR.values)))
print('scores with threshold = 0.5')

print('Accuracy RF: % .3f' % (saukh_accuracy_score(df.actual_label.values,
df.predicted_RF.values)))
print('Recall RF: %.3f' % (saukh_recall_score(df.actual_label.values,
df.predicted_RF.values)))
print('Precision RF: % .3f' % (saukh_precision_score(df.actual_label.values,
df.predicted_RF.values)))
print('F1 RF: %.3f' % (saukh_f1_score(df.actual_label.values,
df.predicted_RF.values)))
print('')

threshold = 0.25

print(f'Scores with threshold = {threshold}')
print('Accuracy RF: % .3f' % (saukh_accuracy_score(df.actual_label.values,
(df.model_RF >= threshold).astype('int').values)))
print('Recall RF: %.3f' % (saukh_recall_score(df.actual_label.values, (df.model_RF
>= threshold).astype('int').values)))
print('Precision RF: %.3f' % (saukh_precision_score(df.actual_label.values,
(df.model_RF >= threshold).astype('int').values)))
print('F1 RF: %.3f' % (saukh_f1_score(df.actual_label.values, (df.model_RF >=
threshold).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()
```

Результат виконання:

```
[[5519 2360]
 [2832 5047]]
TP: 5047
FN: 2832
FP: 2360
TN: 5519
0.6705165630156111
Accuracy RF:0.671
0.6405635232897576
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
```

Рис.10 Результат виконання

```
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
```

Рис.11 Результат виконання для порогу 0.25

```
Scores with threshold = 0.1
Accuracy RF:  0.500
Recall RF: 1.000
Precision RF: 0.500
F1 RF: 0.667
AUC RF:0.738
AUC LR:0.666
```

Рис.12 Результат виконання для порогу 0.10

```
Scores with threshold = 0.75
Accuracy RF:  0.512
Recall RF: 0.025
Precision RF: 0.995
F1 RF: 0.049
AUC RF:0.738
AUC LR:0.666
```

Рис.13 Результат виконання для порогу 0.75

Висновки: в результаті збільшення порогу, F1 міра зменшується.

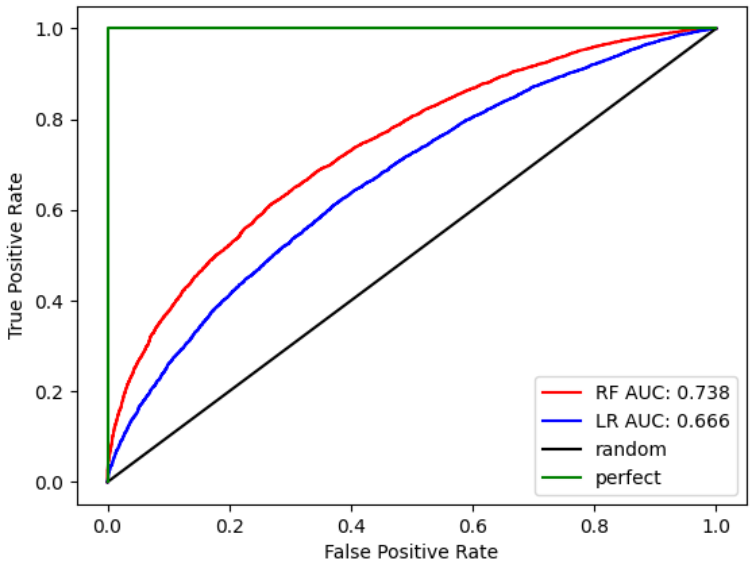

Рис.14 ROC - крива

Як бачимо з графіку RF модель має більшу точність, аніж LR модель. Звісно, що можуть бути ситуації, коли LR має переваги перед RF, але я

думаю, що важливіше, що слід враховувати, — це складність моделі.

Завдання 2.6.

```
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn import metrics

# Вхідний файл, який містить дані
from utilities import visualize_classifier
input_file = 'data_multivar_nb.txt'
# Завантаження даних із вхідного файлу
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.astype(int),
test_size=0.2, random_state=3)

cls = svm.SVC(kernel='linear')
cls.fit(X_train, y_train)
pred = cls.predict(X_test)
print("Accuracy:", metrics.accuracy_score(y_test, y_pred=pred))

print("Precision: ", metrics.precision_score(y_test, y_pred=pred,
average='macro'))

print("Recall", metrics.recall_score(y_test, y_pred=pred, average='macro'))
print(metrics.classification_report(y_test, y_pred=pred))

visualize_classifier(cls, X_test, y_test)
```

Результат виконання:

```
Accuracy: 1.0
Precision:  1.0
Recall 1.0
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        20
           1       1.00      1.00      1.00        17
           2       1.00      1.00      1.00        24
           3       1.00      1.00      1.00        19

    accuracy                           1.00        80
   macro avg       1.00      1.00      1.00        80
weighted avg       1.00      1.00      1.00        80
```

Рис. 15 Результат виконання

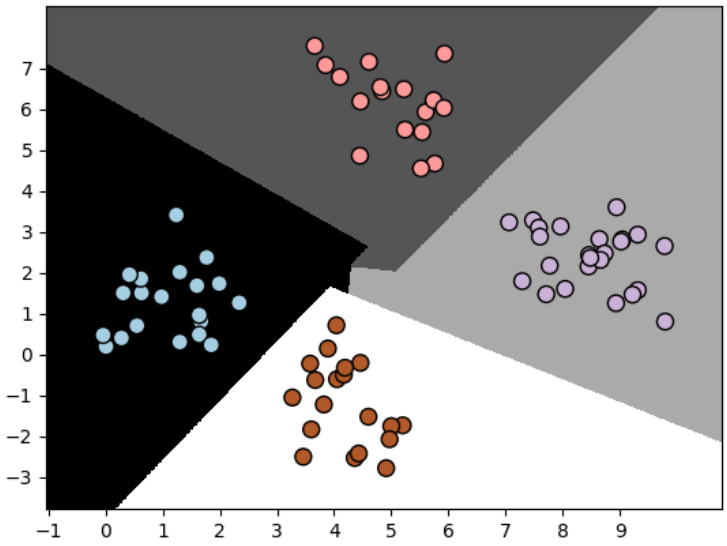

Рис. 16 Результат виконання

Результат порівняння: Наївний класифікатор байесовського і метод опорних векторів (SVM) мають різні параметри, включаючи вибір функції ядра для кожного з них. Обидва алгоритми є дуже чутливими до оптимізації параметрів, тобто вибір різних параметрів може суттєво змінити їхній вихід. Отже, якщо результат показує, що NBC працює краще, ніж SVM, це вірно тільки для вибраних параметрів. Тим не менш, за інших параметрів можна виявити, що SVM працює краще.