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. Попередня обробка даних

```
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 сер знач та станд відхилення
print("\nBEFORE: ")
print("Mean = ", input_data.mean(axis=0))
print("Std deviation = ", input_data.std(axis=0))

# Виключення середнього
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_scaler_minmax = data_scaler_minmax.fit_transform(input_data)
print("\n Min max scaled data:\n", data_scaler_minmax)

# Нормалізація даних
data_normalized_l1 = preprocessing.normalize(input_data, norm='l1')
data_normalized_l2 = preprocessing.normalize(input_data, norm='l2')
print("\n l1 normalized data:\n", data_normalized_l1)
print("\n l2 normalized data:\n", data_normalized_l2)
```

```
C:\Users\Admin\AppData\Local\Microsoft\WindowsApps\python3.10.exe "D:/LabsPoli/AI/Lab1/Task 1/1.py"

 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]]

Process finished with exit code 0
```

Рис. 1

L1-нормалізація використовує метод найменших абсолютних відхилень, що забезпечує рівність 1 суми абсолютних значень в кожному ряду. L2-нормалізація використовує метод найменших квадратів, що забезпечує рівність 1 суми квадратів 4 значень.

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

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))
```

```
2 ×
C:\Users\Admin\AppData\Local\Microsoft\WindowsApps\python3.10.exe "D:/LabsPoli/AI/Lab1/Task 1/2.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']

Process finished with exit code 0
```

Рис. 2

Завдання 2. Попередня обробка нових даних

```
import numpy as np
from sklearn import preprocessing

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

# Бінаризація даних
data_binarized = preprocessing.Binarizer(threshold=2.0).transform(input_data)
print("\n Binarized data: \n", data_binarized)

# Виведення сер знач та станд відхилення
print("\nBEFORE: ")
print("Mean = ", input_data.mean(axis=0))
print("Std deviation = ", input_data.std(axis=0))

# Виключення середнього
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_scaler_minmax = data_scaler_minmax.fit_transform(input_data)
print("\n Min max scaled data:\n", data_scaler_minmax)

# Нормалізація даних
data_normalized_l1 = preprocessing.normalize(input_data, norm='l1')
data_normalized_l2 = preprocessing.normalize(input_data, norm='l2')
print("\n l1 normalized data:\n", data_normalized_l1)
print("\n l2 normalized data:\n", data_normalized_l2)
```

```
C:\Users\Admin\AppData\Local\Microsoft\WindowsApps\python3.10.exe D:/LabsPoli/AI/Lab1/task_2.py

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

BEFORE:
Mean =  [-0.3  2.3  0.2]
Std deviation =  [3.78219513 3.62973828 5.01547605]

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

 Min max scaled data:
 [[0.64356436 0.73737374 1.        ]
 [1.         0.56565657 0.07894737]
 [0.2970297  1.         0.81578947]
 [0.         0.         0.        ]]

 l1 normalized data:
 [[ 0.11403509  0.34210526  0.54385965]
 [ 0.42982456  0.19298246 -0.37719298]
 [-0.171875    0.5078125   0.3203125 ]
 [-0.37681159 -0.24637681 -0.37681159]]

 l2 normalized data:
 [[ 0.17475265  0.52425796  0.83343572]
 [ 0.71216718  0.31974853 -0.62496303]
 [-0.2752151   0.81313551  0.51290086]
 [-0.64182859 -0.41965715 -0.64182859]]

Process finished with exit code 0
```

Рис. 3

Завдання 3. Класифікація логістичною регресією або логістичний класифікатор

```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)
```

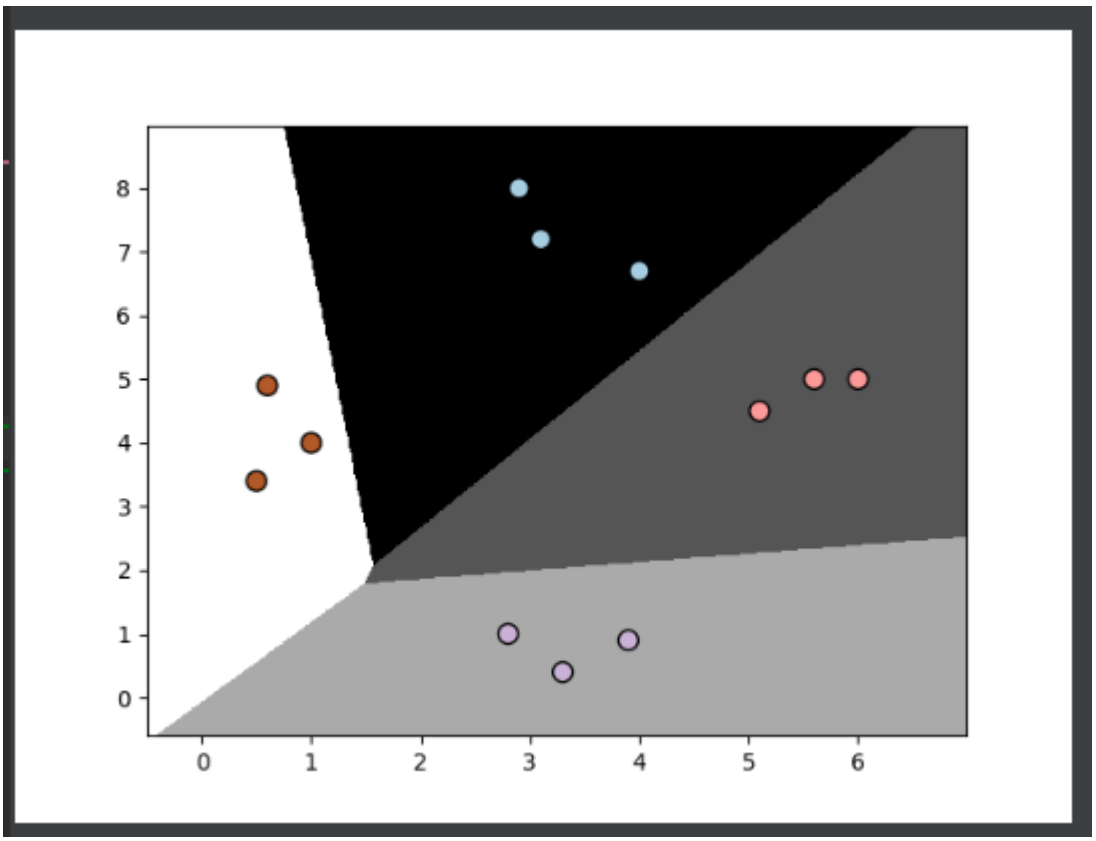


Рис. 4

Завдання 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, cross_val_score
from utilities import visualize_classifier

# 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)) + "%")
```

```
LR_1_task_4
C:\Users\Admin\AppData\Local\Microsoft\WindowsApps\python3.10.exe D:/LabsPoli/AI/Lab1/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%

Process finished with exit code 0
```

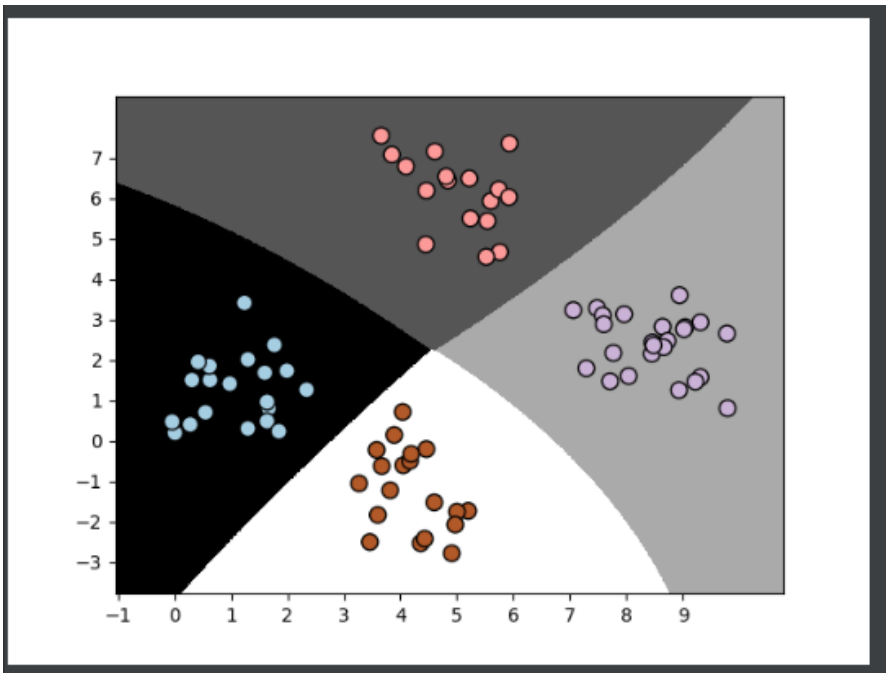


Рис. 5

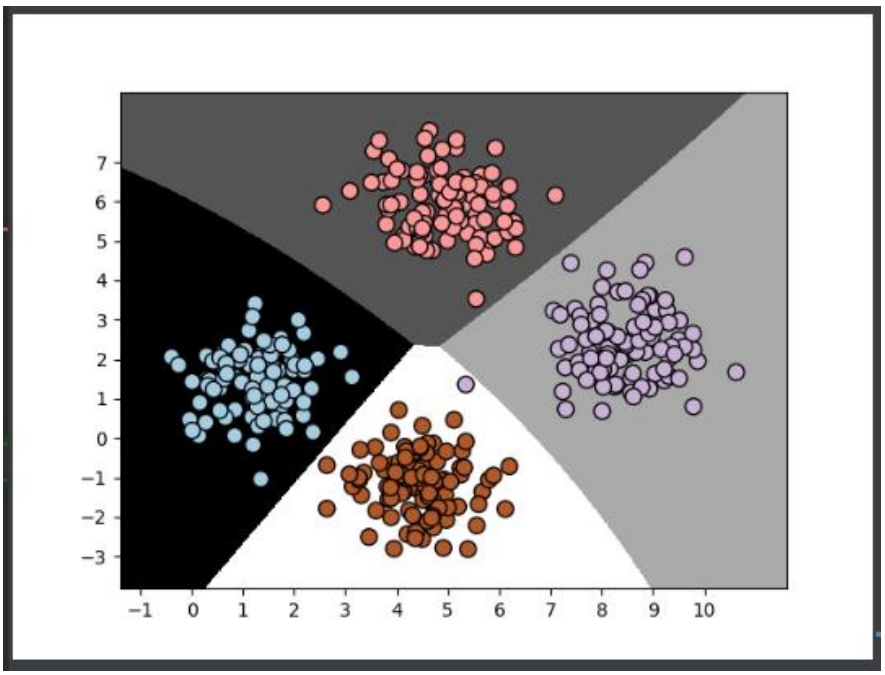
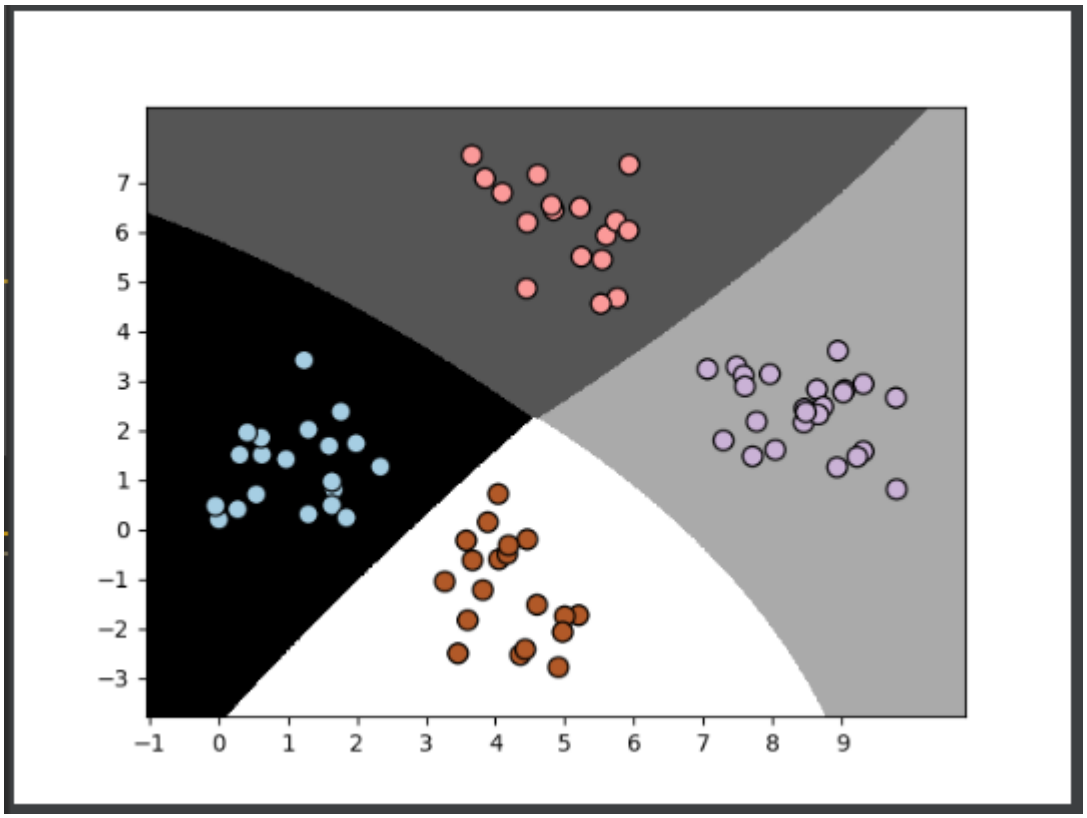


Рис. 6



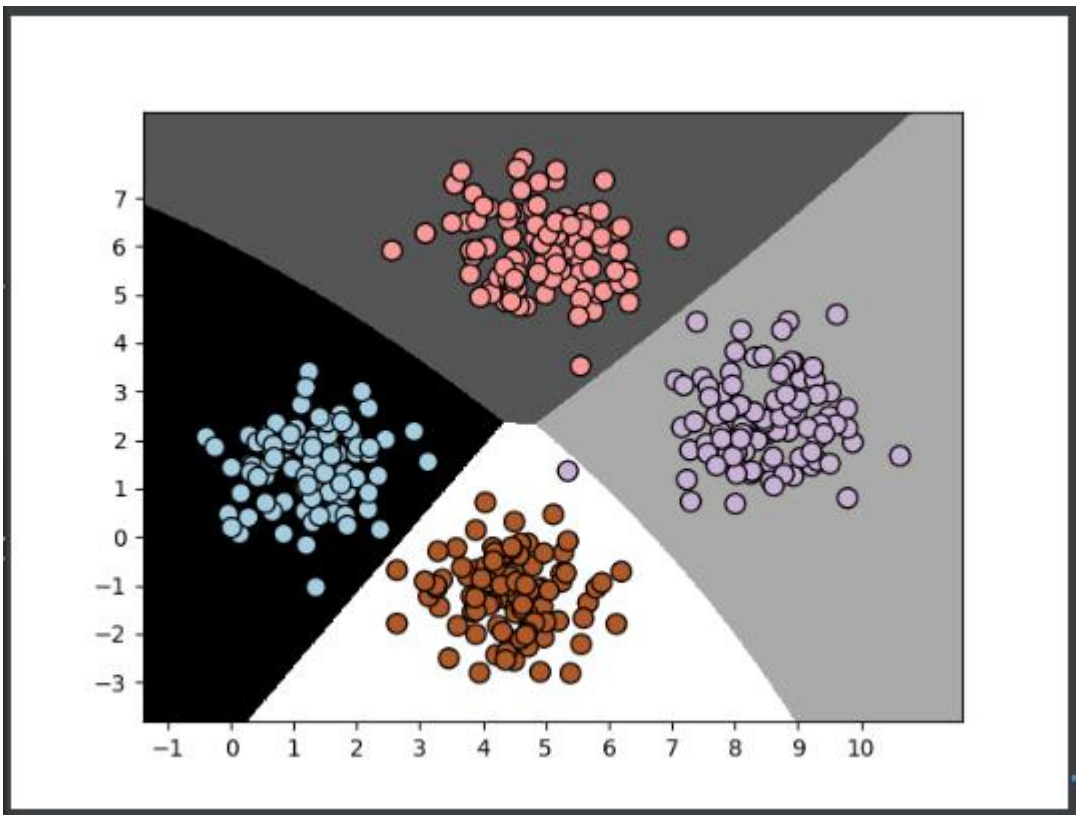


Рис. 7

Завдання 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 recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import roc_curve
import matplotlib.pyplot as plt

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()

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 kyiashenko_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]])


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

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


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

```
print('precision_score:', precision_score(df.actual_label.values,
df.predicted_RF.values))

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


def kyiashenko_recall_score(y_true, y_pred):
    # calculates the fraction of positive samples predictedcorrectly
    TP, FN, FP, TN = find_conf_matrix_values(y_true, y_pred)
    return TP / (TP + FN)


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

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

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


def kyiashenko_precision_score(y_true, y_pred):
    # calculates the fraction of predicted positives samplesthat are actually
positive
    TP, FN, FP, TN = find_conf_matrix_values(y_true, y_pred)
    return TP / (TP + FP)


assert kyiashenko_precision_score(df.actual_label.values,
df.predicted_RF.values) == precision_score(
    df.actual_label.values, df.predicted_RF.values),
'kyiashenko_precision_score failed on RF'
assert kyiashenko_precision_score(df.actual_label.values,
df.predicted_LR.values) == precision_score(
    df.actual_label.values, df.predicted_LR.values),
'kyiashenko_precision_score failed on LR'
print('Precision RF:% .3f' %
(kyiashenko_precision_score(df.actual_label.values, df.predicted_RF.values)))
print('Precision LR:% .3f' %
(kyiashenko_precision_score(df.actual_label.values, df.predicted_LR.values)))

print('f1_score:', f1_score(df.actual_label.values, df.predicted_RF.values))


def kyiashenko_f1_score(y_true, y_pred):
    # calculates the F1 score
    recall = kyiashenko_recall_score(y_true, y_pred)
    precision = kyiashenko_precision_score(y_true, y_pred)
    return (2 * (precision * recall)) / (precision + recall)


assert kyiashenko_f1_score(df.actual_label.values, df.predicted_RF.values) ==
f1_score(df.actual_label.values,

df.predicted_RF.values), 'kyiashenko_f1_score failed on RF'
assert kyiashenko_f1_score(df.actual_label.values, df.predicted_LR.values) ==
f1_score(df.actual_label.values,

df.predicted_LR.values), 'kyiashenko_f1_score failed on LR'
print('F1 RF: %.3f' % (kyiashenko_f1_score(df.actual_label.values,
```

```
df.predicted_RF.values)))
print('F1 LR: %.3f' % (kyiashenko_f1_score(df.actual_label.values,
df.predicted_LR.values)))

print('scores with threshold = 0.5')
print('Accuracy RF:%.3f' % (kyiashenko_accuracy_score(df.actual_label.values,
df.predicted_RF.values)))
print('Recall RF: %.3f' % (kyiashenko_recall_score(df.actual_label.values,
df.predicted_RF.values)))
print('Precision RF:%.3f' %
(kyiashenko_precision_score(df.actual_label.values, df.predicted_RF.values)))
print('F1 RF: %.3f' % (kyiashenko_f1_score(df.actual_label.values,
df.predicted_RF.values)))
print('')
print('scores with threshold = 0.25')
print('Accuracy RF:%.3f' % (
    kyiashenko_accuracy_score(df.actual_label.values, (df.model_RF >=
0.25).astype('int').values)))
print('Recall RF: %.3f' % (kyiashenko_recall_score(df.actual_label.values,
(df.model_RF >= 0.25).astype('int').values)))
print('Precision RF:%.3f' % (
    kyiashenko_precision_score(df.actual_label.values, (df.model_RF >=
0.25).astype('int').values)))
print('F1 RF: %.3f' % (kyiashenko_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()
```

```
C:\Users\Admin\AppData\Local\Microsoft\WindowsApps\python3.10.exe D:/LabsPoli/AI/Lab1/LR_1_task_5.py
TP: 5047
FN: 2832
FP: 2360
TN: 5519
kyiashenko_confusion_matrix: [[5519 2360]
 [2832 5047]]
accuracy_score: 0.6705165630156111
Accuracy RF:0.671
precision_score: 0.681382476036182
recall_score: 0.6405635232897576
Recall RF: 0.641
Precision RF: 0.681
Precision LR: 0.636
f1_score: 0.660342797330891
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

Process finished with exit code 0
```

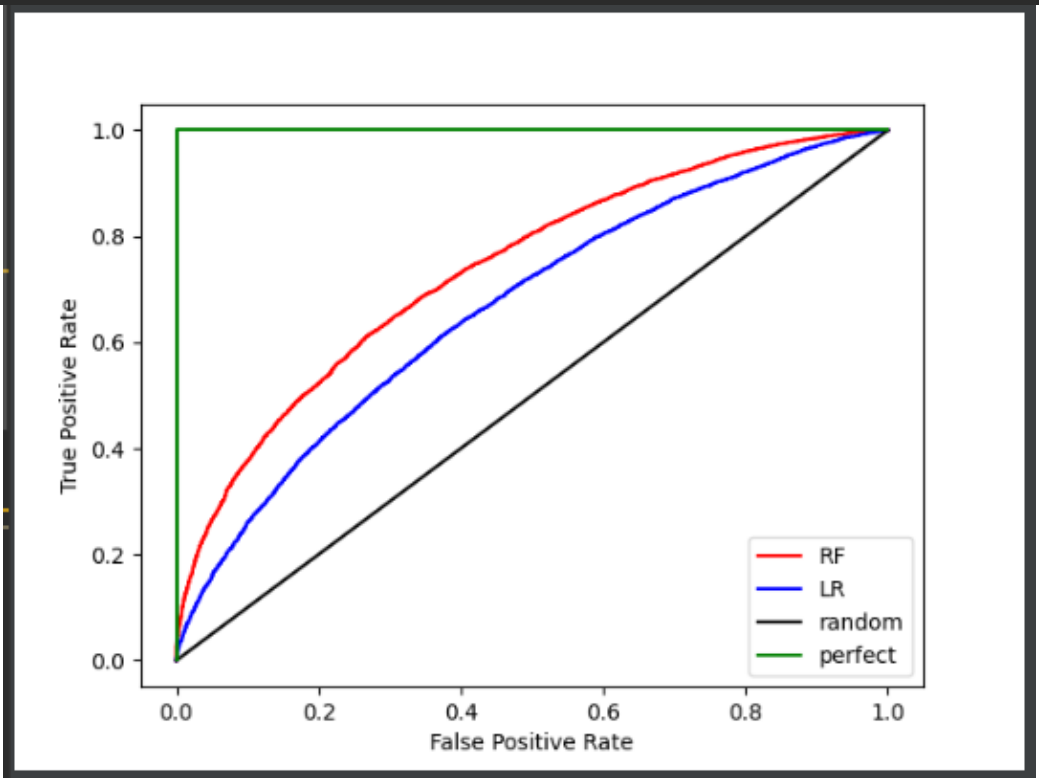


Рис. 8

RF модель показала себе більш точною.

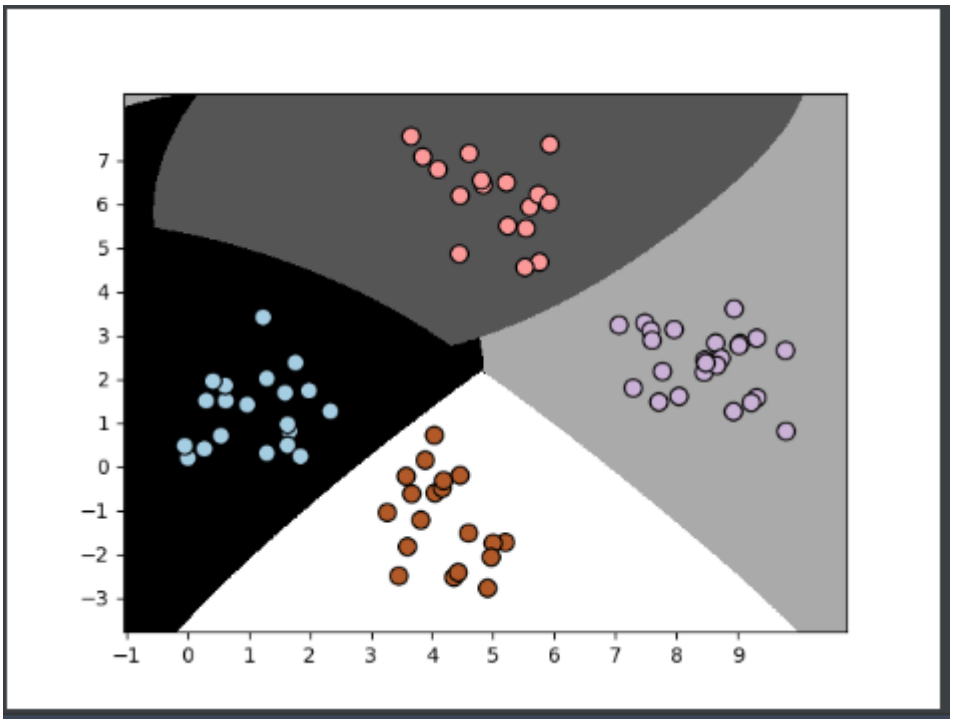
Завдання 6

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.

```
import numpy as np
from sklearn import svm
from sklearn.model_selection import train_test_split, cross_val_score
from utilities import visualize_classifier

# 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 = svm.SVC()

# 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 svc 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 = svm.SVC()
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)) + "%")
```

```
C:\Users\Admin\AppData\Local\Microsoft\WindowsApps\python3.10.exe D:/LabsPoli/AI/Lab1/LR_1_task_6.py
Accuracy of svc classifier = 99.75 %
Accuracy of the new classifier = 100.0 %
Accuracy: 99.75%
Precision: 99.76%
Recall: 99.75%
F1: 99.75%

Process finished with exit code 0
```



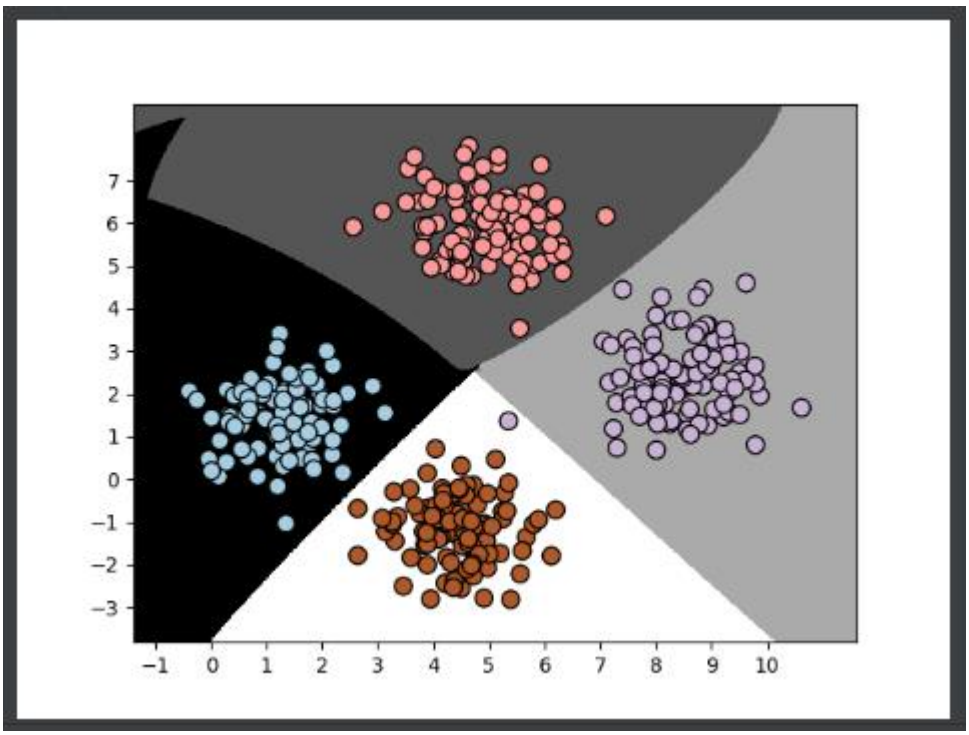


Рис. 9

Метод наївного байєсівського класифікатора спрацював точніше і швидше.

Посилання на Git: https://github.com/Grum74/AI

Висновок

Я, використовуючи спеціалізовані бібліотеки та мову програмування Python дослідив попередню обробку та класифікацію даних.