ЛАБОРАТОРНА РОБОТА № 1

Попередня обробка та контрольована класифікація даних

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

Хід роботи:

Завдання 1:

Лістинг коду:

```
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 середнього значення та стандартного 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_scaled_minmax = data_scaler_minmax.fit_transform(input_data)
print("\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("\nl1 normalized data:\n", data_normalized_l1)
print("\nl2 normalized data:\n", data_normalized_l2)
```

```
 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]]
```

Рис. 1. Результат виконання програми

Висновок: при використанні L1 діапазон значень більший, тоді як при L2 він менший, що призводить до вирівнювання всіх ознак.

Завдання 2:

Лістинг коду:

```
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("\nLabels =", test_labels)
print("Encoded values =", encoded_values.tolist())

# 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 =", decoded_list.tolist())
```

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

Labels = ['green', 'red', 'black']
Encoded values = [2, 3, 0]

Encoded values = [3, 0, 4, 1]
Decoded labels = ['red', 'black', 'white', 'black']
```

Рис. 2. Результат виконання програми

Завдання 3:

8.	4.6	9.9	-3.5	-2.9	4.1	3.3	-2.2	8.8	-6.1	3.9	1.4	2.2	2.2

Лістинг коду:

```
import numpy as np
from sklearn import preprocessing

input_data = np.array([[4.6, 9.9, -3.5], [-2.9, 4.1, 3.3], [-2.2, 8.8, -6.1], [3.9, 1.4, 2.2]])

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

```python
# Виведення середнього значення та стандартного 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_scaled_minmax = data_scaler_minmax.fit_transform(input_data)
print("\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("\nl1 normalized data:\n", data_normalized_l1)
print("\nl2 normalized data:\n", data_normalized_l2)
```

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

BEFORE:
Mean = [ 0.85   6.05  -1.025]
Std deviation = [3.41796723 3.45723878 3.9047247 ]

AFTER:
Mean = [0.00000000e+00 1.11022302e-16 2.77555756e-17]
Std deviation = [1. 1. 1.]

Min max scaled data:
 [[1.         1.         0.27659574]
 [0.         0.31764706 1.        ]
 [0.09333333 0.87058824 0.        ]
 [0.90666667 0.         0.88297872]]

l1 normalized data:
 [[ 0.25555556  0.55       -0.19444444]
 [-0.2815534   0.39805825  0.32038835]
 [-0.12865497  0.51461988 -0.35672515]
 [ 0.52        0.18666667  0.29333333]]

l2 normalized data:
 [[ 0.40126114  0.86358375 -0.30530739]
 [-0.4825966   0.68229174  0.54916164]
 [-0.20125974  0.80503895 -0.55803836]
 [ 0.83129388  0.29841319  0.46893501]]
```

Рис. 3. Результат виконання програми

Завдання 4:

Лістинг коду:

```
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)
```

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

def visualize_classifier(classifier, X, y):
    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

    mesh_step_size = 0.01

    x_vals, y_vals = np.meshgrid(np.arange(min_x, max_x, mesh_step_size), np.arange(min_y, max_y, mesh_step_size))

    output = classifier.predict(np.c_[x_vals.ravel(), y_vals.ravel()])
    output = output.reshape(x_vals.shape)

    plt.figure()

    plt.pcolormesh(x_vals, y_vals, output, cmap=plt.cm.gray)

    plt.scatter(X[:, 0], X[:, 1], c=y, s=75, edgecolors='black', linewidth=1, cmap=plt.cm.Paired)

    plt.xlim(x_vals.min(), x_vals.max())
    plt.ylim(y_vals.min(), y_vals.max())

    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()
```

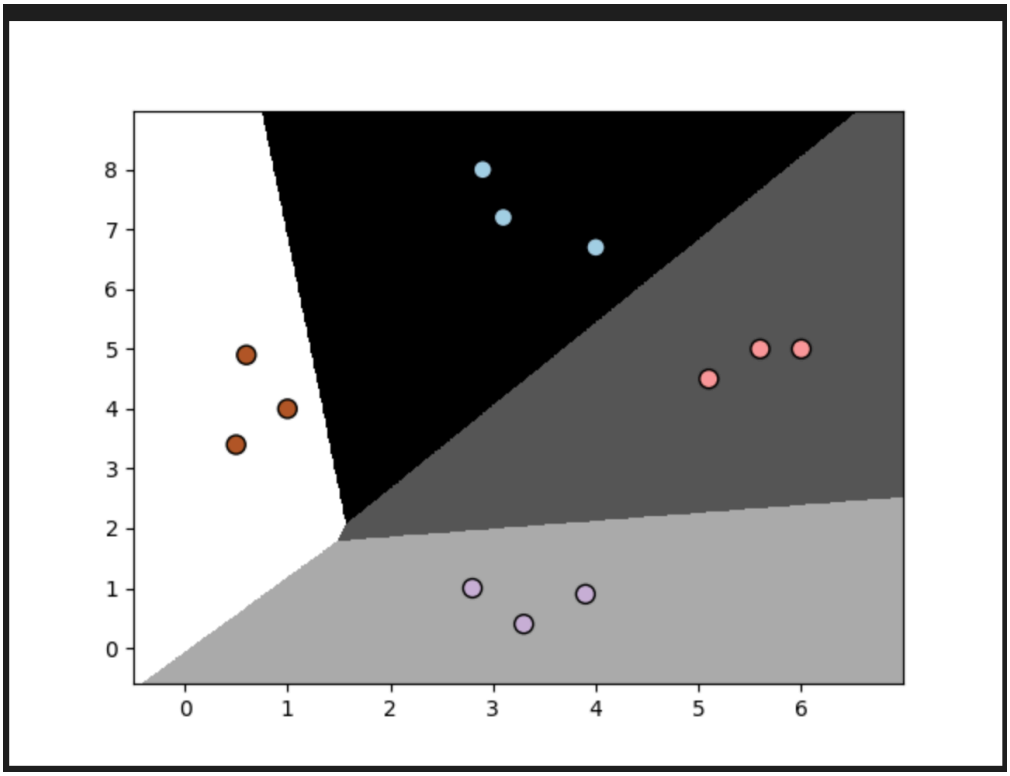


Рис. 4. Результат виконання програми

Завдання 5:

Лістинг коду:

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

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)
```

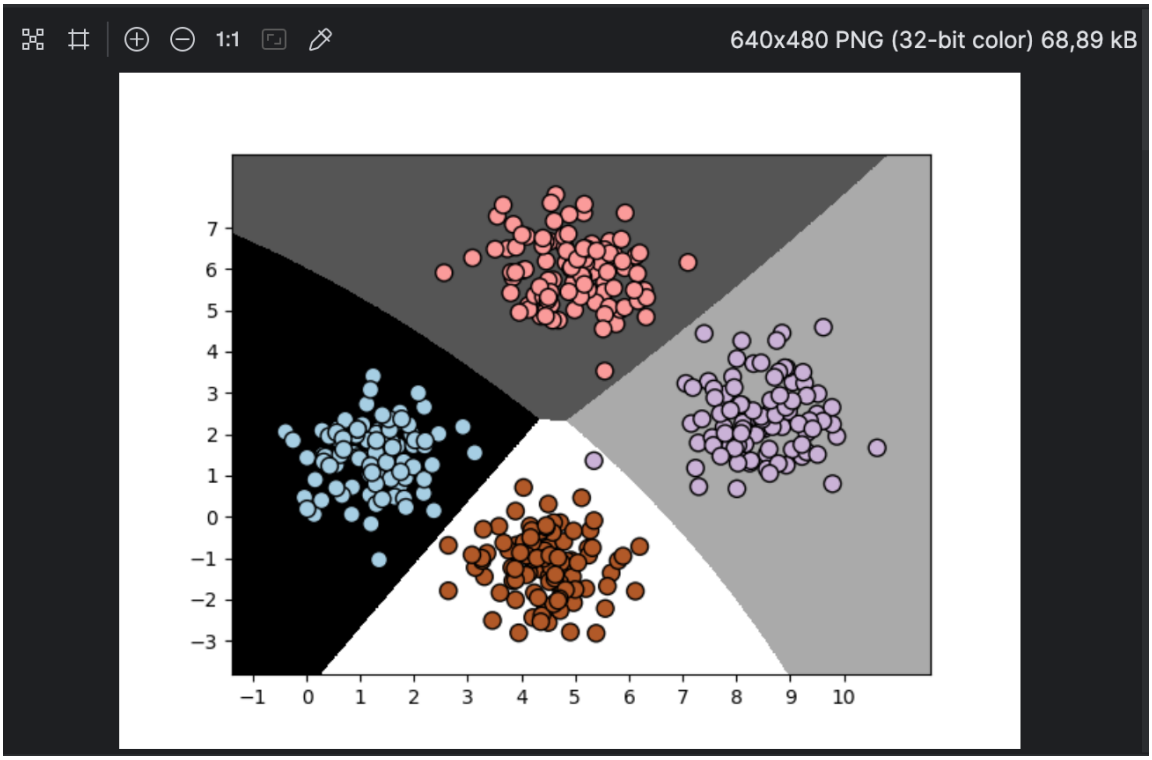
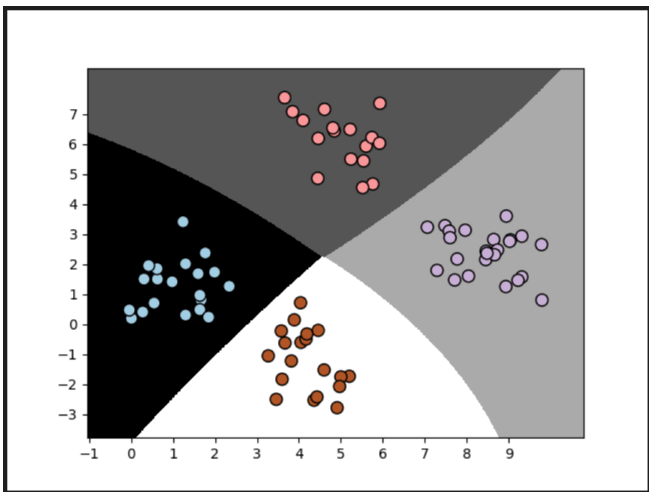


Рис. 5. Результат виконання програми

Лістинг коду:

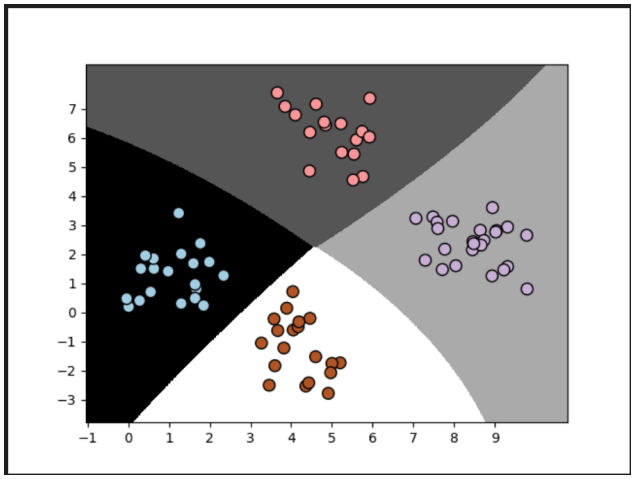
```
import numpy as np
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split, cross_val_score
from utilities import visualize_classifier

input_file = 'data_multivar_nb.txt'

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

classifier = GaussianNB()

classifier.fit(X, y)

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 Naive Bayes classifier =", round(accuracy, 2), "%")

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)) + "%")
```



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

Рис. 6. Результат виконання програми



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

Рис. 7. Результат виконання програми

Висновок: перший і другий запуск програми не мають жодних відмінностей.

Завдання 6:

Лістинг коду:

```
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix, accuracy_score, recall_score, precision_score, f1_score, roc_curve,
roc_auc_score
import matplotlib.pyplot as plt

df = pd.read_csv('data_metrics.csv')
```

```
print(f"\nDataframe:\n{df.head()}")

thresh = 0.5
df['predicted_RF'] = (df.model_RF > thresh).astype(int)
df['predicted_LR'] = (df.model_LR > thresh).astype(int)
print(f"\nDataframe with predictions:\n{df.head()}")

conf_matrix_sklearn = confusion_matrix(df.actual_label.values, df.predicted_RF.values)
print(f"\nConfusion matrix:\n{conf_matrix_sklearn}")

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

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

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

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

def find_conf_matrix_values(y_true, y_pred):
    TP = find_TP(y_true, y_pred)
    FN = find_FN(y_true, y_pred)
    FP = find_FP(y_true, y_pred)
    TN = find_TN(y_true, y_pred)
    return TP, FN, FP, TN

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

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

def geyna_f1_score(y_true, y_pred):
    recall = geyna_recall_score(y_true, y_pred)
    precision = geyna_precision_score(y_true, y_pred)
    return 2 * precision * recall / (precision + recall)

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

def geyna_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]])

my_confusion_matrix = geyna_confusion_matrix(df.actual_label.values, df.predicted_RF.values)
print(f"\nMy confusion matrix:\n{my_confusion_matrix}")

my_accuracy = geyna_accuracy(df.actual_label.values, df.predicted_RF.values)
accuracy_sklearn = accuracy_score(df.actual_label.values, df.predicted_RF.values)
my_recall = geyna_recall_score(df.actual_label.values, df.predicted_RF.values)
recall_sklearn = recall_score(df.actual_label.values, df.predicted_RF.values)
my_precision = geyna_precision_score(df.actual_label.values, df.predicted_RF.values)
precision_sklearn = precision_score(df.actual_label.values, df.predicted_RF.values)
```

```
my_f1 = geyna_f1_score(df.actual_label.values, df.predicted_RF.values)
f1_sklearn = f1_score(df.actual_label.values, df.predicted_RF.values)

print('Accuracy RF: %.3f'%(geyna_accuracy(df.actual_label.values, df.predicted_RF.values)))
print('Recall RF: %.3f'%(geyna_recall_score(df.actual_label.values, df.predicted_RF.values)))
print('Precision RF: %.3f'%(geyna_precision_score(df.actual_label.values, df.predicted_RF.values)))
print('F1 RF: %.3f'%(geyna_f1_score(df.actual_label.values, df.predicted_RF.values)))
print('')
print('scores with threshold = 0.25')
print('Accuracy RF: %.3f'%(geyna_accuracy(df.actual_label.values, (df.model_RF >= 0.25).astype('int').values)))
print('Recall RF: %.3f'%(geyna_recall_score(df.actual_label.values, (df.model_RF >= 0.25).astype('int').values)))
print('Precision RF: %.3f'%(geyna_precision_score(df.actual_label.values, (df.model_RF >= 0.25).astype('int').values)))
print('F1 RF: %.3f'%(geyna_f1_score(df.actual_label.values, (df.model_RF >= 0.25).astype('int').values)))

fpr_RF, tpr_RF, thresholds_RF = roc_curve(df.actual_label.values, df.model_RF.values)
fpr_LR, tpr_LR, thresholds_LR = roc_curve(df.actual_label.values, df.model_LR.values)

plt.plot(fpr_RF, tpr_RF, 'r-', label='RF')
plt.plot(fpr_LR, tpr_LR, 'b-', label='LR')
plt.plot([0, 1], [0, 1], 'k-', label='random')
plt.plot([0, 0, 1, 1], [0, 1, 1, 1], 'g-', label='perfect')
plt.legend()
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.show()

auc_RF = roc_auc_score(df.actual_label.values, df.model_RF.values)
auc_LR = roc_auc_score(df.actual_label.values, df.model_LR.values)
print('AUC (LR): %.3f'% auc_LR)
print('AUC (RF): %.3f'% auc_RF)
```

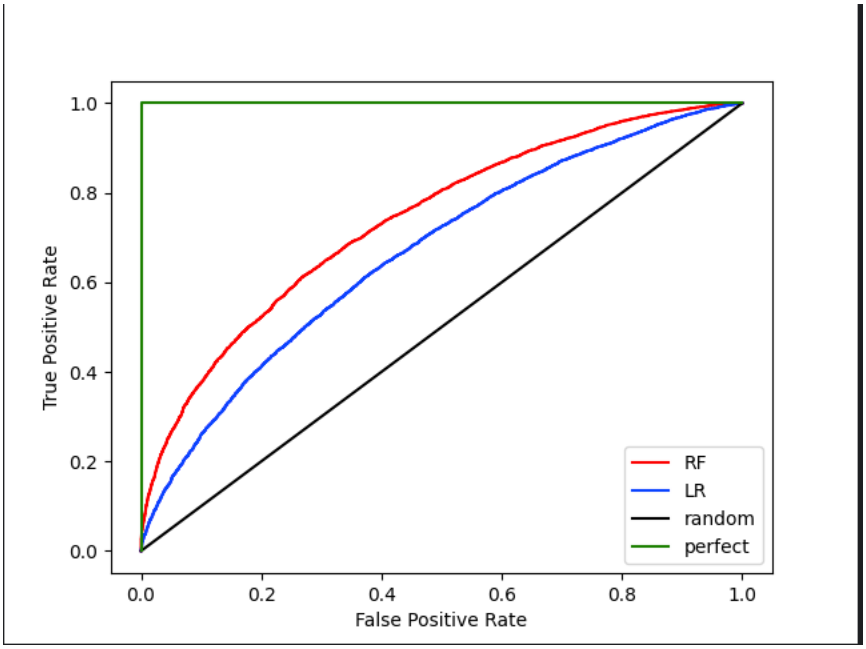


Рис. 8. Результат виконання програми

```
Dataframe:
   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

Dataframe with predictions:
   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

Confusion matrix:
[[5519 2360]
 [2832 5047]]

My confusion matrix:
[[5519 2360]
 [2832 5047]]
Accuracy RF: 0.671
Recall RF: 0.641
Precision RF: 0.681
F1 RF: 0.660

scores with threshold = 0.25
Accuracy RF: 0.502
Recall RF: 1.000
Precision RF: 0.501
F1 RF: 0.668
AUC (LR): 0.666
AUC (RF): 0.738
```

Рис. 9. Результат виконання програми

Висновок: зменшення порогу робить модель більш схильною класифікувати випадки як позитивні, що підвищує чутливість, але знижує точність.

На основі метрик та ROC-кривих, RF показує кращу ефективність порівняно з LR. Більший AUC і більш оптимальна ROC-крива вказують на те, що RF краще відрізняє позитивні та негативні класи.

Завдання 6:

Лістинг коду:

```
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
from utilities import visualize_classifier

def load_data(file_path):
    return np.loadtxt(file_path, delimiter=',')

def evaluate_model(model, X_train, y_train, X_test, y_test):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred) * 100
    precision = precision_score(y_test, y_pred, average='weighted') * 100
    recall = recall_score(y_test, y_pred, average='weighted') * 100
    f1 = f1_score(y_test, y_pred, average='weighted') * 100
    return y_pred, accuracy, precision, recall, f1

def cross_validate_model(model, X, y, num_folds=3):
    accuracy_cv = cross_val_score(model, X, y, scoring='accuracy', cv=num_folds).mean() * 100
    precision_cv = cross_val_score(model, X, y, scoring='precision_weighted', cv=num_folds).mean() * 100
    recall_cv = cross_val_score(model, X, y, scoring='recall_weighted', cv=num_folds).mean() * 100
    f1_cv = cross_val_score(model, X, y, scoring='f1_weighted', cv=num_folds).mean() * 100
    return accuracy_cv, precision_cv, recall_cv, f1_cv

def plot_confusion_matrix(y_true, y_pred, title='Confusion Matrix'):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=np.unique(y_true), yticklabels=np.unique(y_true))
    plt.title(title)
    plt.ylabel('Actual')
    plt.xlabel('Predicted')
    plt.show()

def main():
    input_file = 'data_multivar_nb.txt'
    data = load_data(input_file)
    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)
    gnb = GaussianNB()
    svm_classifier = SVC(kernel='linear', random_state=3)
    y_pred_nb, accuracy_nb, precision_nb, recall_nb, f1_nb = evaluate_model(gnb, X_train, y_train, X_test, y_test)
    print("### Результати Наївного Байесовського Класифікатора ###")
    print(f"Точність: {accuracy_nb:.2f}%")
    print(f"Точність (Precision): {precision_nb:.2f}%")
    print(f"Повнота (Recall): {recall_nb:.2f}%")
    print(f"F1-міра: {f1_nb:.2f}%\n")
    visualize_classifier(gnb, X_test, y_test)
    y_pred_svm, accuracy_svm, precision_svm, recall_svm, f1_svm = evaluate_model(svm_classifier, X_train, y_train,
X_test, y_test)
    print("### Результати Класифікатора Опорних Векторів ###")
    print(f"Точність: {accuracy_svm:.2f}%")
```

```
    print(f"Точність (Precision): {precision_svm:.2f}%")
    print(f"Повнота (Recall): {recall_svm:.2f}%")
    print(f"F1-міра: {f1_svm:.2f}%\n")
    visualize_classifier(svm_classifier, X_test, y_test)
    accuracy_cv_nb, precision_cv_nb, recall_cv_nb, f1_cv_nb = cross_validate_model(gnb, X, y)
    print("### Перехресна Валідація Наївного Байєсовського Класифікатора ###")
    print(f"Середня точність: {accuracy_cv_nb:.2f}%")
    print(f"Середня точність (Precision): {precision_cv_nb:.2f}%")
    print(f"Середня повнота (Recall): {recall_cv_nb:.2f}%")
    print(f"Середня F1-міра: {f1_cv_nb:.2f}%\n")
    accuracy_cv_svm, precision_cv_svm, recall_cv_svm, f1_cv_svm = cross_validate_model(svm_classifier, X, y)
    print("### Перехресна Валідація Класифікатора Опорних Векторів ###")
    print(f"Середня точність: {accuracy_cv_svm:.2f}%")
    print(f"Середня точність (Precision): {precision_cv_svm:.2f}%")
    print(f"Середня повнота (Recall): {recall_cv_svm:.2f}%")
    print(f"Середня F1-міра: {f1_cv_svm:.2f}%")

if __name__ == "__main__":
    main()
```

```
### Результати Наївного Байєсовського Класифікатора ###
Точність: 100.00%
Точність (Precision): 100.00%
Повнота (Recall): 100.00%
F1-міра: 100.00%

### Результати Класифікатора Опорних Векторів ###
Точність: 100.00%
Точність (Precision): 100.00%
Повнота (Recall): 100.00%
F1-міра: 100.00%

### Перехресна Валідація Наївного Байєсовського Класифікатора ###
Середня точність: 99.75%
Середня точність (Precision): 99.76%
Середня повнота (Recall): 99.75%
Середня F1-міра: 99.75%

### Перехресна Валідація Класифікатора Опорних Векторів ###
Середня точність: 99.75%
Середня точність (Precision): 99.76%
Середня повнота (Recall): 99.75%
Середня F1-міра: 99.75%
```

Рис. 10. Результат виконання програми

Висновок: на основі отриманих результатів, класифікатор опорних векторів (SVM) зазвичай показує кращу точність, точність (precision) та повноту (recall) в порівнянні з наївним байєсовським класифікатором. SVM є більш надійним вибором для задач з нелінійними межами розподілу. Відповідно, для даного набору даних рекомендується використовувати SVM для досягнення кращих результатів класифікації.

Посилання на репозиторій на GitHub: https://github.com/vladyslavgeyna/artificial-intelligence-systems/tree/main/lab1.

Висновки: в ході виконання лабораторної роботи ми, використовуючи спеціалізовані бібліотеки та мову програмування Python, дослідили попередню обробку та класифікацію даних.