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

1.1. Бінарізація

Лістинг програми:

```
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.]]
```

Рисунок 1.1 – Результат роботи програми

1.2. Виключення середнього

Лістинг програми:

```
# 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 значення
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.]
```

Рисунок 1.2 – Результат роботи програми

1.3. Масштабування

Лістинг програми:

```
# 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]]
```

Рисунок 1.3 – Результат роботи програми

1.4. Нормалізація

Лістинг програми:

```
# 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]]
```

Рисунок 1.4 – Результат роботи програми

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 значень. Взагалі, техніка L1-нормалізації вважається більш надійною по порівняно з L2-нормалізацією, оскільки вона менш чутлива до викидів.

Дуже часто дані містять викиди. Ми хочемо використовувати безпечні методики, що дозволяють ігнорувати викиди у процесі обчислень. Якби ми вирішували завдання, в якому викиди грають важливу роль, то, ймовірно, найкращим вибором була б L2-нормалізація.

Дивлячись на отримані значення обох нормалізацій, можна зробити висновок, що значення у L1 менші ніж у L2.

1.5. Кодування міток

Лістинг програми:

```
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 =", 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:
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']
```

Рисунок 1.5 – Результат роботи програми

Завдання 2. Попередня обробка нових даних

У коді програми попереднього завдання поміняйте дані по рядках (значення змінної input_data) на значення відповідно варіанту таблиці 1.1 та виконайте операції: Бінарізації, Виключення середнього, Масштабування, Нормалізації. Варіант обирається відповідно номера за списком групи відповідно до таблиці 1.1.

Таблиця 1.1

№ варіанту	Значення змінної input_data												Поріг бінаризації
15.	-2.3	3.9	-4.5	-5.3	-4.2	-1.3	5.2	-6.5	-1.1	-5.2	2.6	-2.2	3.0

Лістинг програми:

```
        import numpy as np
from sklearn import preprocessing

input_data = np.array([[-2.3, 3.9, -4.5], [-5.3, -4.2, -1.3], [5.2, -6.5, -1.1],
[-5.2, 2.6, -2.2]])

# Бінаризація даних
data_binarized = preprocessing.Binarizer(threshold=3.0).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 значення
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:
 [[0. 1. 0.]
 [0. 0. 0.]
 [1. 0. 0.]
 [0. 0. 0.]]

BEFORE:
Mean = [-1.9   -1.05  -2.275]
Std deviation = [4.27258704 4.40028408 1.3497685 ]

AFTER:
Mean = [-2.77555756e-17  5.55111512e-17  2.09901541e-16]
Std deviation = [1. 1. 1.]

Min max scaled data:
 [[0.28571429 1.         0.        ]
 [0.         0.22115385 0.94117647]
 [1.         0.         1.        ]
 [0.00952381 0.875      0.67647059]]

l1 normalized data:
 [[-0.21495327  0.36448598 -0.42056075]
 [-0.49074074 -0.38888889 -0.12037037]
 [ 0.40625    -0.5078125  -0.0859375 ]
 [-0.52        0.26       -0.22      ]]

l2 normalized data:
 [[-0.36029981  0.61094315 -0.7049344 ]
 [-0.76965323 -0.60991388 -0.18878287]
 [ 0.61931099 -0.77413873 -0.13100809]
 [-0.83653629  0.41826814 -0.3539192 ]]
```

Рисунок 1.6 – Результат роботи програми

Завдання 3. Класифікація логістичною регресією або логістичний класифікатор

Лістинг програми:

```
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)
```

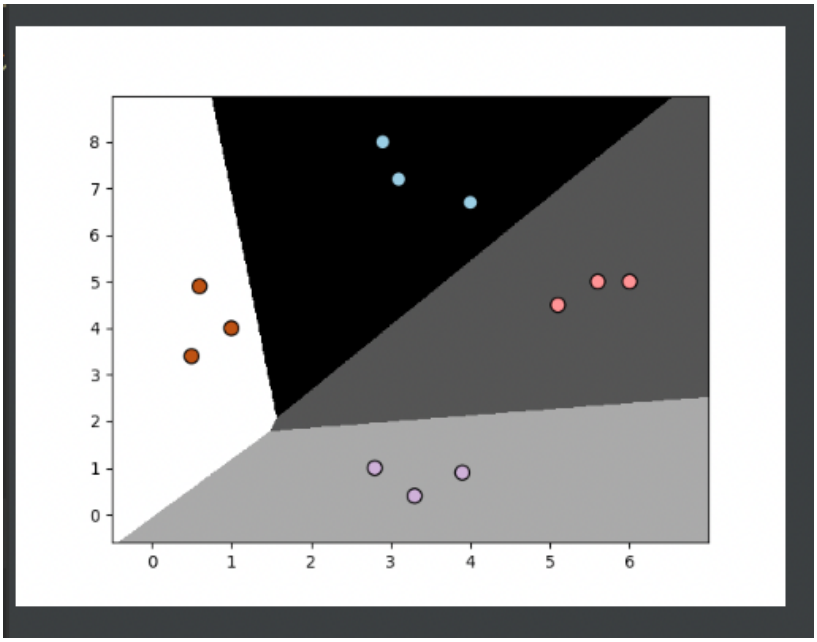

Рисунок 1.7 – Результат роботи програми у вигляді графіку

Завдання 4. Класифікація наївним байєсовським класифікатором

Лістинг програми:

```
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 * (np.array(y) == np.array(y_pred)).sum() / X.shape[0]
print("Accuracy of Naive Bayes classifier =", round(accuracy, 2), "%")
# Візуалізація результатів роботи класифікатора
visualize_classifier(classifier, X, y)
```

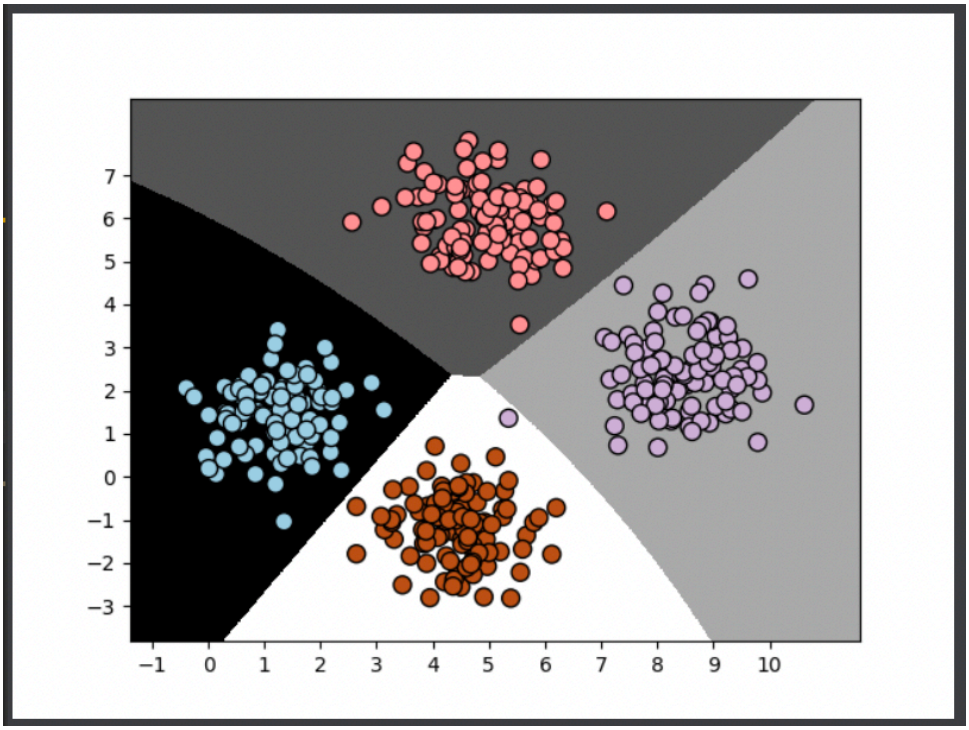

Рисунок 1.8 – Результат роботи програми у вигляді графіку

```
Accuracy of Naive Bayes classifier = 100.0 %
```

Рисунок 1.9 – Результат роботи програми

Лістинг програми:

```
# 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 * (np.array(y_test) == np.array(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)) + "%")
```

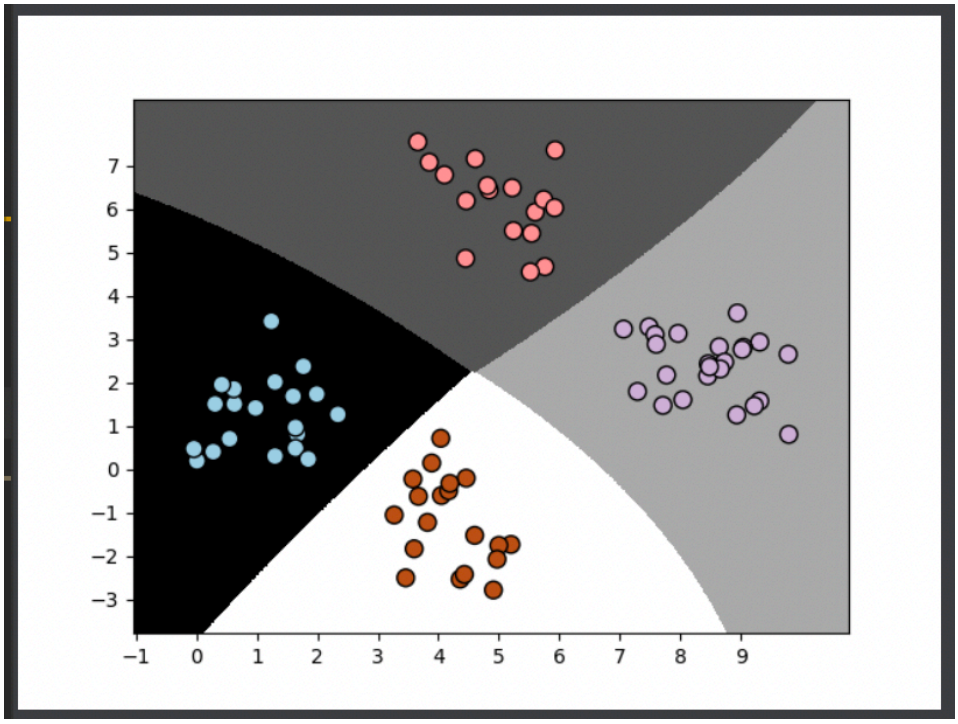

Рисунок 2.1 – Результат роботи програми у вигляді графіку

```
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%
```

Рисунок 2.2 – Результат роботи програми

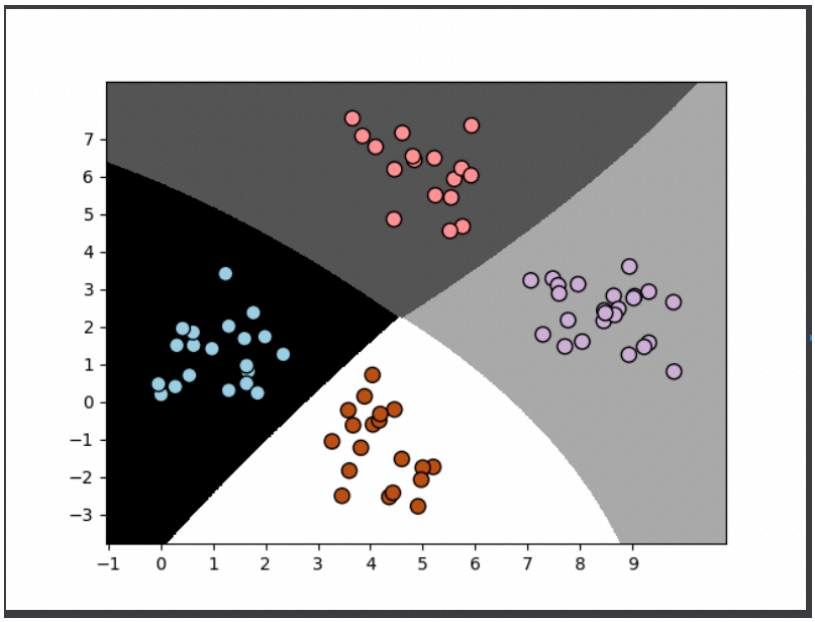

Рисунок 2.3 – Зображення результатів класифікації другого прогону

Бачимо, що результат не змінився.

Завдання 5. Вивчити метрики якості класифікації

Лістинг програми:

```
import pandas as pd
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()

from sklearn.metrics import confusion_matrix
```

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


def sivchenko_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 sivchenko_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 sivchenko_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 sivchenko_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:', sivchenko_find_TP(df.actual_label.values, df.predicted_RF.values))
print('FN:', sivchenko_find_FN(df.actual_label.values, df.predicted_RF.values))
print('FP:', sivchenko_find_FP(df.actual_label.values, df.predicted_RF.values))
print('TN:', sivchenko_find_TN(df.actual_label.values, df.predicted_RF.values))

import numpy as np


def sivchenko_find_conf_matrix_values(y_true, y_pred):  # calculate TP, FN, FP, TN
    TP = sivchenko_find_TP(y_true, y_pred)
    FN = sivchenko_find_FN(y_true, y_pred)
    FP = sivchenko_find_FP(y_true, y_pred)
    TN = sivchenko_find_TN(y_true, y_pred)
    return TP, FN, FP, TN


def sivchenko_confusion_matrix(y_true, y_pred):
    TP, FN, FP, TN = sivchenko_find_conf_matrix_values(y_true, y_pred)
    return np.array([[TN, FP], [FN, TP]])


print('Confusion matrix:\n', sivchenko_confusion_matrix(df.actual_label.values,
df.predicted_RF.values))

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

from sklearn.metrics import accuracy_score
print('Accuracy score: ', accuracy_score(df.actual_label.values,
df.predicted_RF.values))
```

```
def sivchenko_accuracy_score(y_true, y_pred):  # calculates the fraction of
samples
    TP, FN, FP, TN = sivchenko_find_conf_matrix_values(y_true, y_pred)
    return (TP + TN)/(TP + TN + FP + FN)


print('Accuracy score function: ',
sivchenko_accuracy_score(df.actual_label.values, df.predicted_RF.values))


assert sivchenko_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 sivchenko_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' % (sivchenko_accuracy_score(df.actual_label.values,
df.predicted_RF.values)))
print('Accuracy LR: %.3f' % (sivchenko_accuracy_score(df.actual_label.values,
df.predicted_LR.values)))


from sklearn.metrics import recall_score
print('Recall score: ', recall_score(df.actual_label.values,
df.predicted_RF.values))


def sivchenko_recall_score(y_true, y_pred):  # calculates the fraction of positive
samples predicted correctly
    TP, FN, FP, TN = sivchenko_find_conf_matrix_values(y_true, y_pred)
    return TP/(TP + FN)


print('Recall score function: ', sivchenko_recall_score(df.actual_label.values,
df.predicted_RF.values))

assert sivchenko_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 sivchenko_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' % (sivchenko_recall_score(df.actual_label.values,
df.predicted_RF.values)))
print('Recall LR: %.3f' % (sivchenko_recall_score(df.actual_label.values,
df.predicted_LR.values)))


from sklearn.metrics import precision_score
print('Precision score: ', precision_score(df.actual_label.values,
df.predicted_RF.values))


def sivchenko_precision_score(y_true, y_pred):  # calculates the fraction of
predicted positives sample that are actually positive
    TP, FN, FP, TN = sivchenko_find_conf_matrix_values(y_true, y_pred)
    return TP/(TP + FP)


print('Precision score function: ',
sivchenko_precision_score(df.actual_label.values, df.predicted_RF.values))
```

```
assert sivchenko_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 sivchenko_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' % (sivchenko_precision_score(df.actual_label.values,
df.predicted_RF.values)))
print('Precision LR: %.3f' % (sivchenko_precision_score(df.actual_label.values,
df.predicted_LR.values)))


from sklearn.metrics import f1_score
print('f1 score: ', f1_score(df.actual_label.values, df.predicted_RF.values))


def sivchenko_f1_score(y_true, y_pred): # calculates the F1 score
    recall = sivchenko_recall_score(y_true, y_pred)
    precision = sivchenko_precision_score(y_true, y_pred)
    return (2 * (precision * recall))/(precision + recall)


print('f1 score function: ', f1_score(df.actual_label.values,
df.predicted_RF.values))

assert sivchenko_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 sivchenko_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' % (sivchenko_f1_score(df.actual_label.values,
df.predicted_RF.values)))
print('F1 LR: %.3f' % (sivchenko_f1_score(df.actual_label.values,
df.predicted_LR.values)))


print('scores with threshold = 0.5')
print('Accuracy RF: %.3f' % (sivchenko_accuracy_score(df.actual_label.values,
df.predicted_RF.values)))
print('Recall RF: %.3f' % (sivchenko_recall_score(df.actual_label.values,
df.predicted_RF.values)))
print('Precision RF: %.3f' % (sivchenko_precision_score(df.actual_label.values,
df.predicted_RF.values)))
print('F1 RF: %.3f' % (sivchenko_f1_score(df.actual_label.values,
df.predicted_RF.values)))
print('')
print('scores with threshold = 0.25')
print('Accuracy RF: %.3f' % (sivchenko_accuracy_score(df.actual_label.values,
(df.model_RF >= 0.25).astype('int').values)))
print('Recall RF: %.3f' % (sivchenko_recall_score(df.actual_label.values,
(df.model_RF >= 0.25).astype('int').values)))
print('Precision RF: %.3f' % (sivchenko_precision_score(df.actual_label.values,
(df.model_RF >= 0.25).astype('int').values)))
print('F1 RF: %.3f' % (sivchenko_f1_score(df.actual_label.values, (df.model_RF >=
0.25).astype('int').values)))


from sklearn.metrics import roc_curve
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)
```

```
import matplotlib.pyplot as plt
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()


from sklearn.metrics import roc_auc_score
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)

import matplotlib.pyplot as plt
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()
```

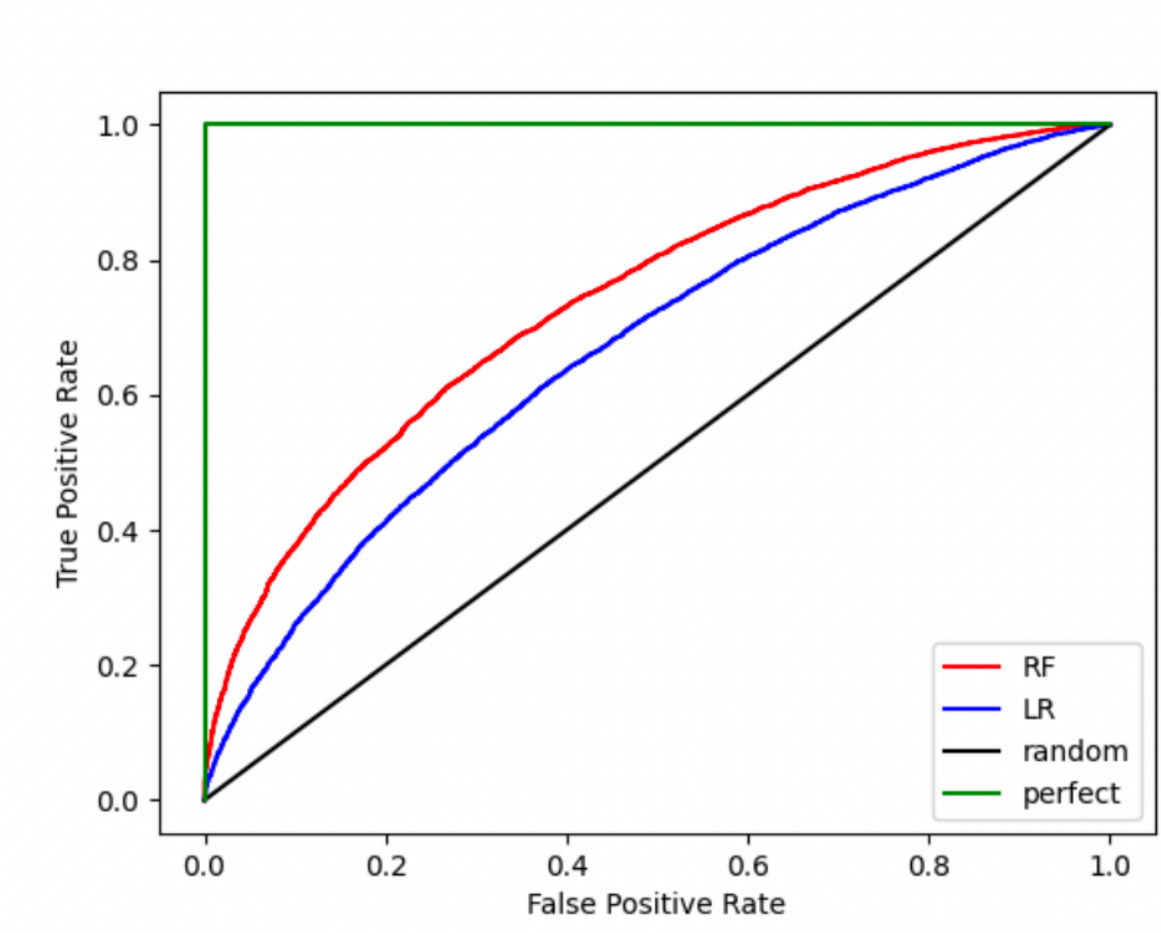

Рисунок 2.4 – Результат роботи програми у вигляді графіку

```
TP: 5047
FN: 2832
FP: 2360
TN: 5519
Confusion matrix:
 [[5519 2360]
 [2832 5047]]
Accuracy score:  0.6705165630156111
Accuracy score function:  0.6705165630156111
Accuracy RF: 0.671
Accuracy LR: 0.616
Recall score:  0.6405635232897576
Recall score function:  0.6405635232897576
Recall RF: 0.641
Recall LR: 0.543
Precision score:  0.681382476036182
Precision score function:  0.681382476036182
Precision RF: 0.681
Precision LR: 0.636
f1 score:  0.660342797330891
f1 score function:  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
```

Рисунок 2.5 – Результат роботи програми

Лістинг програми:

```
from sklearn.metrics import roc_auc_score
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)

import matplotlib.pyplot as plt
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()
```

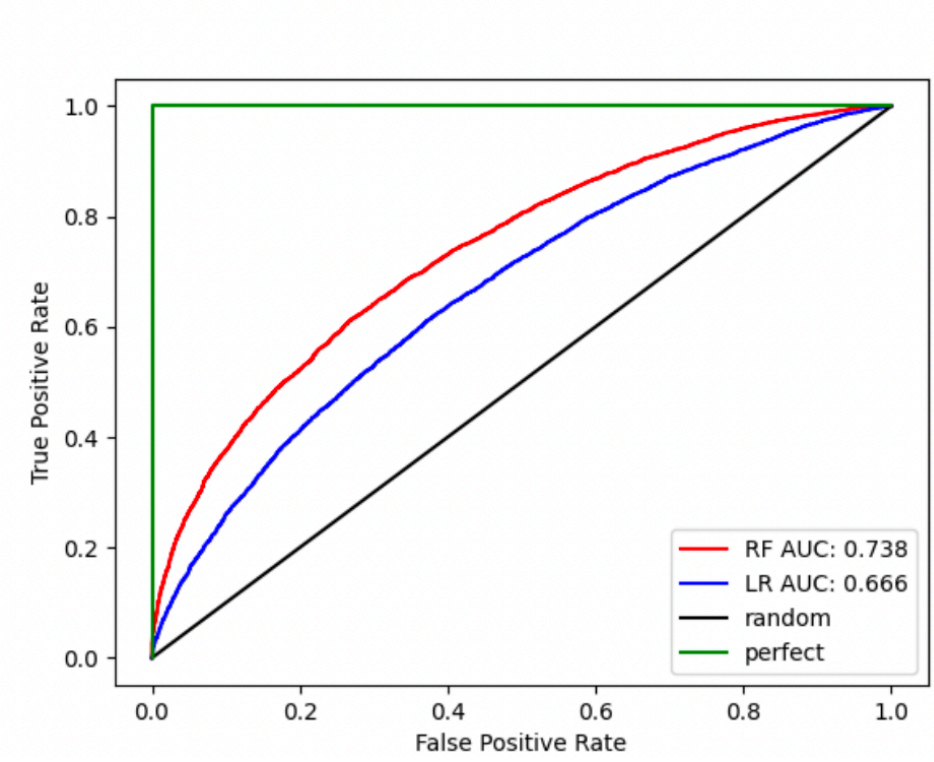

Рисунок 2.6 – Результат роботи програми у вигляді графіку

```
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
```

Рисунок 2.7 – Результат роботи програми

Як можна побачити на рисунку 2.7, площа під кривою для моделі RF (AUC = 0,738) краще, ніж LR (AUC = 0,666).

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.

Лістинг програми:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import SVC
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 = SVC()

# Тренування класифікатора
classifier.fit(X, y)

# Прогнозування значень для тренувальних даних
y_pred = classifier.predict(X)

# Обчислення якості класифікатора
accuracy = 100.0 * (np.array(y) == np.array(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 = SVC()
classifier_new.fit(X_train, y_train)
y_test_pred = classifier_new.predict(X_test)

# Обчислення якості класифікатора
accuracy = 100.0 * (np.array(y_test) == np.array(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)) + "%")
```

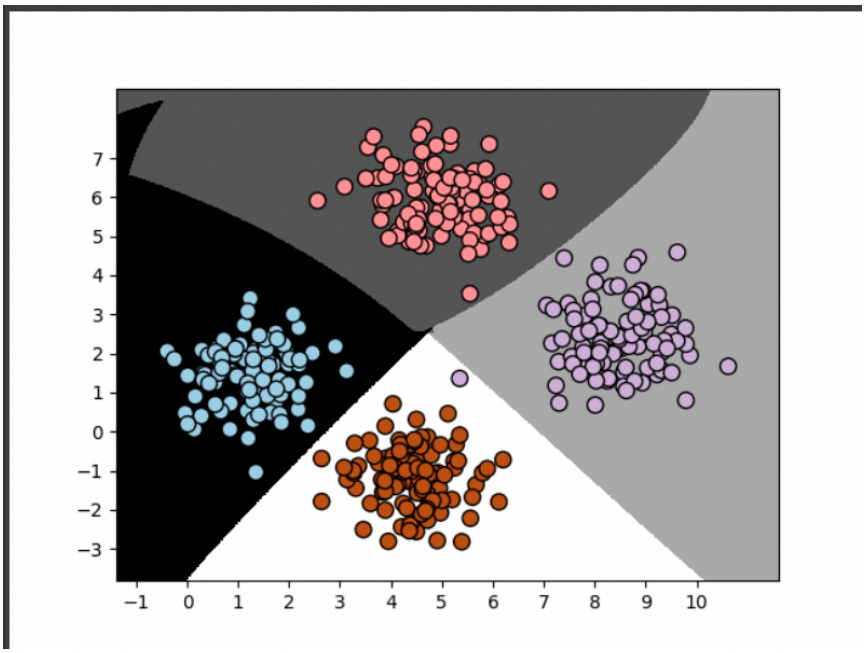

Рисунок 2.8 – Результат роботи програми у вигляді графіку

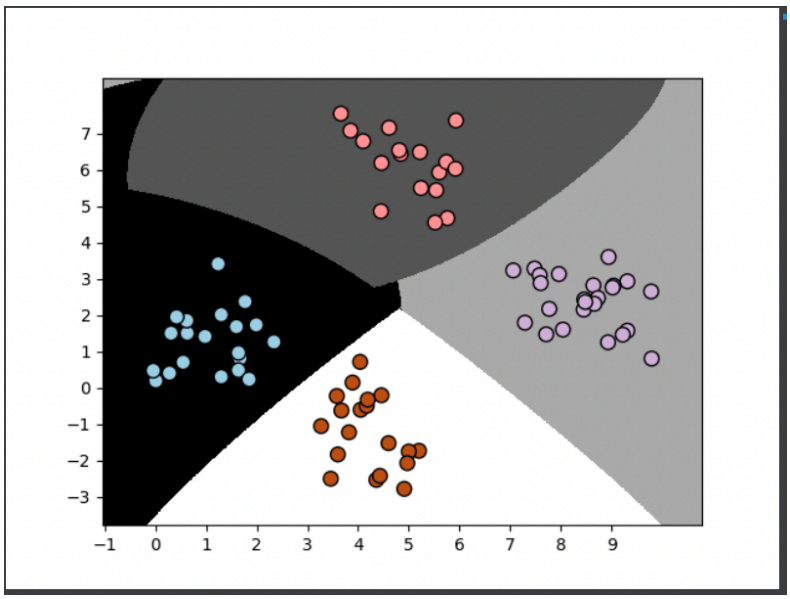

Рисунок 2.9 – Результат роботи програми у вигляді графіку

```
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%
```

Рисунок 3.1 – Результат роботи програми

Посилання на GitHub: https://github.com/SoylerProfile/SHI

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