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 2.1. Попередня обробка даних

```python
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)
```

Рис. 1. Код програми

```
 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. 2. Результат виконання програми

Різниця між L1 та L2 нормалізаціями:

в L1-нормалізації використовується метод найменших абсолютних відхилень, а в L2-нормалізації – метод найменших квадратів. Розглянемо на прикладі першого рядка масиву:

L1-нормалізація (сума абсолютних значень елементів повинна бути рівна нулю): |0.45132743| + |-0.25663717| + |0.2920354| = 1

L2-нормалізація (сума квадратів значень елементів повинна бути рівна нулю): 0.75765788^2 + (-0.43082507)^2 + (0.49024922)^2 = 1

Кодування міток

```
# 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))
```

Рис. 3. Код програми

```
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. 4. Результат виконання програми

Бачимо, що кожній мітці присвоїлось число (як індекс масиву). Потім для відповідних міток виводиться їх закодоване значення (green = 1, red = 2, black = 0). Так само робиться декодування для міток по їх закодованому значенню (3 = white і т.д.).

Завдання 2.2. Попередня обробка даних

№ варіанту	Значення змінної input_data											Поріг бінаризації
19.	-4.1	-5.5	3.3	6.9	4.6	3.9	-4.2	3.8	2.3	3.9	3.4	-1.2	3.2

Рис. 5. Завдання за варіантом

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

input_data = np.array([[-4.1, -5.5, 3.3], [6.9, 4.6, 3.9], [-4.2, 3.8, 2.3], [3.9, 3.4, -1.2]])

# Бінаризація даних
data_binarized = preprocessing.Binarizer(threshold=3.2).transform(input_data)
print("\n Binarized data:\n", data_binarized)

# Виключення середнього
data_scaled = preprocessing.scale(input_data)
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)
```

Рис. 6. Код програми

```
 Binarized data:
 [[0. 0. 1.]
 [1. 1. 1.]
 [0. 1. 0.]
 [1. 1. 0.]]
Mean = [ 2.77555756e-17  4.16333634e-17 -1.66533454e-16]
Std deviation = [1. 1. 1.]
```

Рис. 7. Бінаризація і виключення середнього

```
Min max scaled data:
 [[0.00900901 0.         0.88235294]
 [1.         1.         1.        ]
 [0.         0.92079208 0.68627451]
 [0.72972973 0.88118812 0.        ]]

L1 normalized data:
 [[-0.31782946 -0.42635659  0.25581395]
 [ 0.44805195  0.2987013   0.25324675]
 [-0.40776699  0.36893204  0.22330097]
 [ 0.45882353  0.4        -0.14117647]]

L2 normalized data:
 [[-0.53858858 -0.72249687  0.43349812]
 [ 0.75294136  0.50196091  0.42557555]
 [-0.68704905  0.6216158   0.37624114]
 [ 0.73428231  0.64014355 -0.22593302]]
```

Рис. 8. Масштабування та нормалізація

Завдання 2.3. Класифікація логістичною регресією або логістичний класифікатор

```python
import numpy as np
from sklearn import linear_model
from Task.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. 9. Код програми

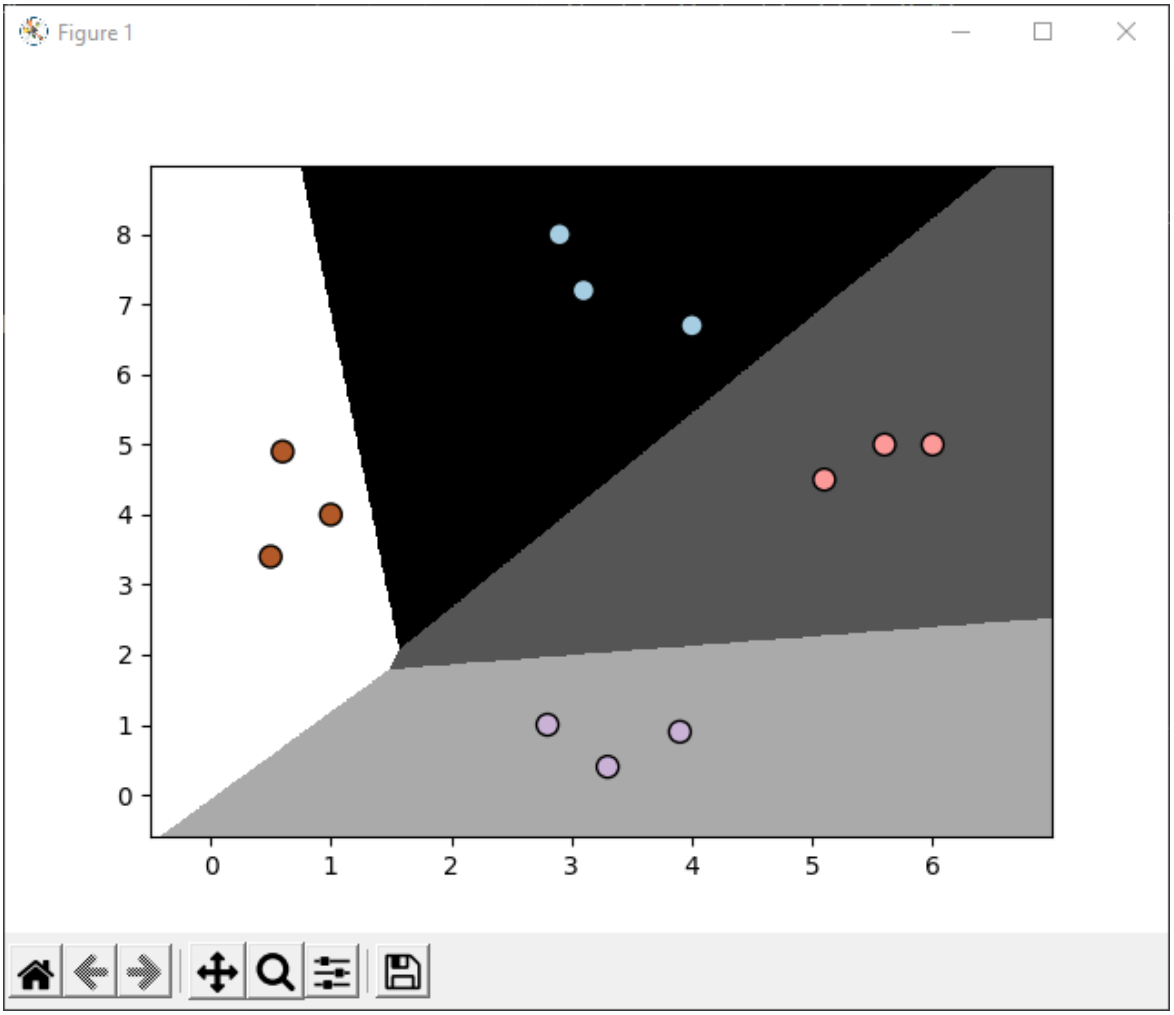


Рис. 10. Результат виконання програми

Завдання 2.4. Класифікація наївним байєсовським класифікатором

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

# Вхідний файл, який містить дані
input_file = 'Task/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)
```

Рис. 11. Код програми

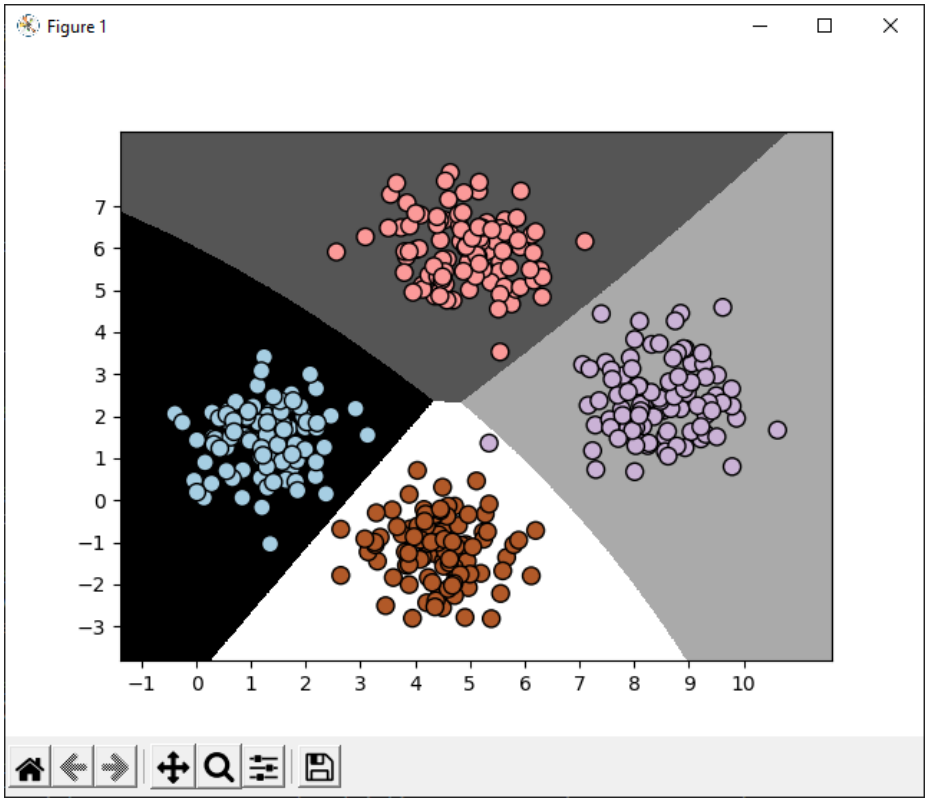
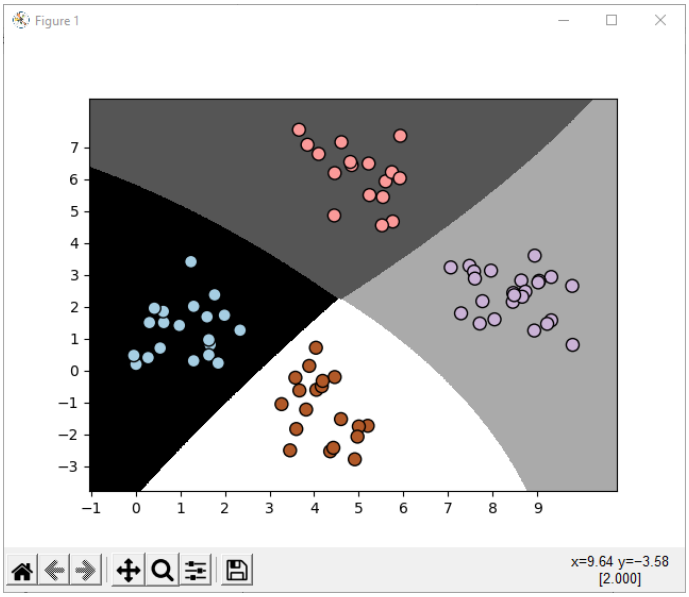


Рис. 12. Результат виконання програми

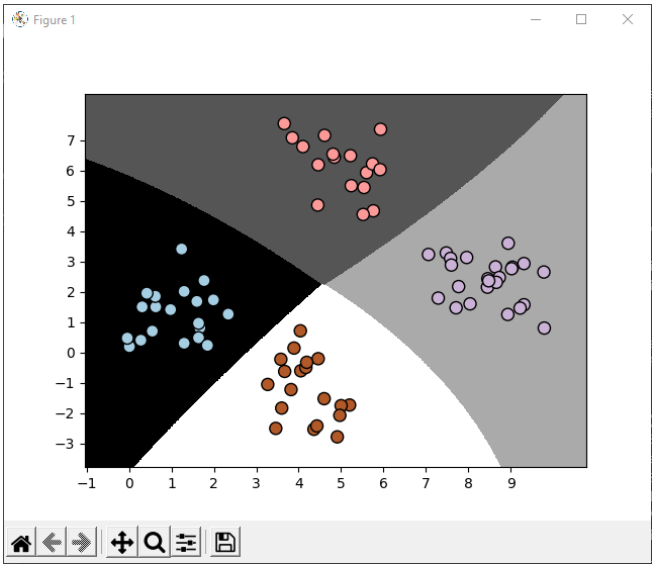
```
# Розбивка даних на навчальний та тестовий набори
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=3)
classifier_new = GaussianNB()
classifier_new.fit(X_train, y_train)
y_test_pred = classifier_new.predict(X_test)

# Обчислення якості класифікатора
accuracy = 100.0 * (y_test == y_test_pred).sum() / X_test.shape[0]
print("Accuracy of the new classifier =", round(accuracy, 2), "%")

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)) + "%")

# Візуалізація роботи класифікатора
visualize_classifier(classifier_new, X_test, y_test)
```

Рис. 13. Доданий код




```
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%
```

Рис. 14. Результат виконання оновленої програми




```
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%
```

Рис. 15. Результат повторного виконання

Результат першого та повторного виконань є ідентичними, отже, можна зробити висновок, що класифікація є точною.

Завдання 2.5. Вивчити метрики якості класифікації

```
scores with threshold = 0.5
Accuracy RF: 0.671
Recall RF: 0.641
Precision RF: 0.681
F1 RF: 0.660
```

Рис. 16. Результати при порозі 0.5

```
scores with threshold = 0.25
Accuracy RF: 0.502
Recall RF: 1.000
Precision RF: 0.501
F1 RF: 0.668
```

Рис. 17. Результати при порозі 0.25

Можемо зробити висновок, що при нижчому порозі акуратність зменшилась, чутливість збільшилась до максимальної, точність зменшилась, а оцінка майже не змінилась

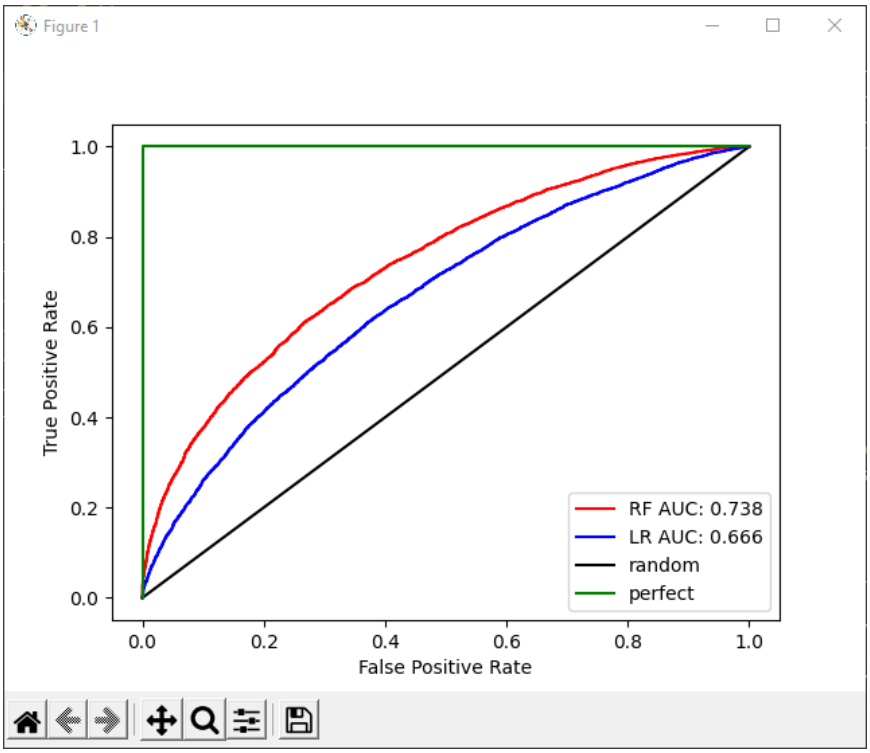


Рис. 18. Крива ROC з урахуванням AUC

З графіку можна зрозуміти, що модель RF є більш оптимальною, адже знаходиться далі від чорної лінії (випадковість) та ближче до зеленої (правильність вибору)

Завдання 2.6. 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.

Якість класифікації при SVM = 100%; при байєсівському класифікаторі = 99.75%. Але при використанні наївного байєсівського класифікатора велику роль відіграє значення порогу, при зміні якого змінюються і результати роботи. Тому хоча значення 99.75% дуже близьке до 100%, але краще використовувати модель SVM, яка працює без залежності від певних встановлених значень.

```python
import numpy as np
from sklearn import svm
from sklearn.metrics import recall_score, classification_report, confusion_matrix
from sklearn.model_selection import train_test_split

input_file = 'Task/data_multivar_nb.txt'
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, test_size=0.2)
model = svm.SVC(kernel='linear', C=1.0)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
print(recall_score(y_test, y_pred, average=None))
```

Рис. 19. Код програми

```
[[28  0  0  0]
 [ 0 11  0  0]
 [ 0  0 22  0]
 [ 0  0  0 19]]
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        28
         1.0       1.00      1.00      1.00        11
         2.0       1.00      1.00      1.00        22
         3.0       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

[1. 1. 1. 1.]
```

Рис. 20. Результат виконання програми

Висновок: в ході виконання лабораторної роботи було досліджено попередню обробку та класифікацію даних для подальшого їх використання за допомогою мови програмування Python.