Лабораторна робота №1

Тема: ПОПЕРЕДНЯ ОБРОБКА ТА КОНТРОЛЬОВАНА КЛАСИФІКАЦІЯ ДАНИХ

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

Хід роботи

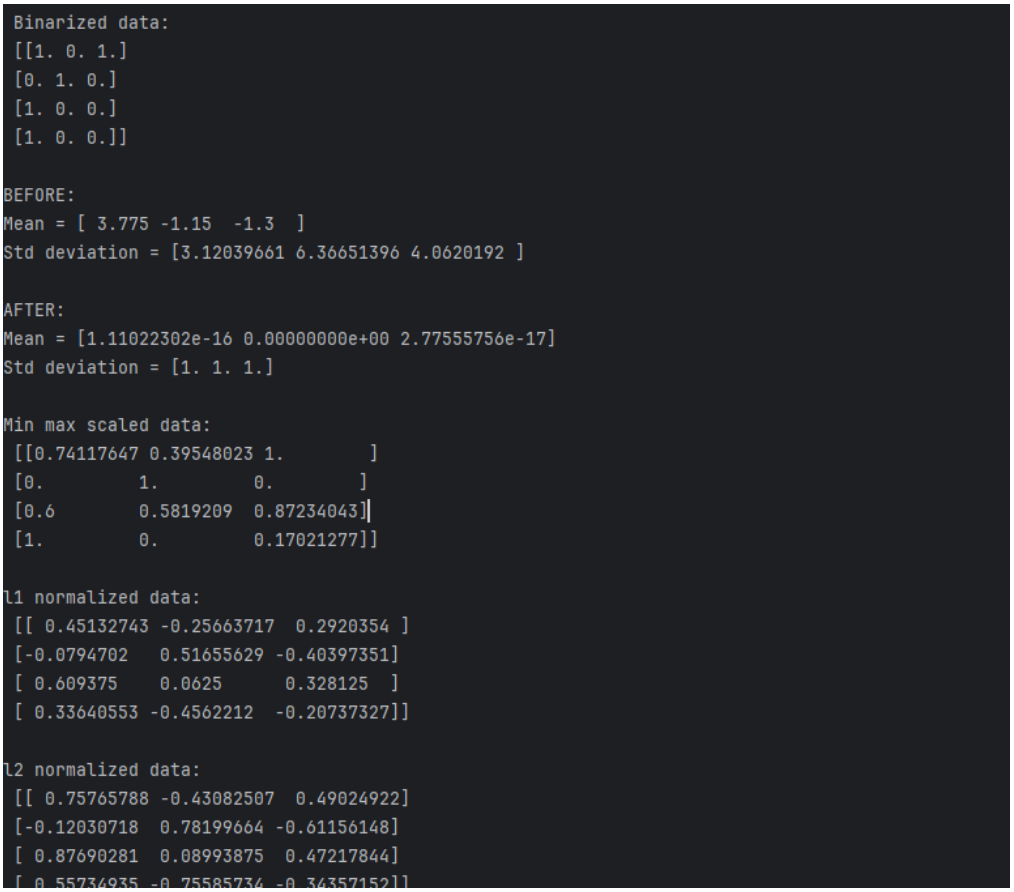
Репозиторій: https://github.com/enot1k666/labsOAI.git

Завдання 2.1: Попередня обробка даних.

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)

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

# Исключение среднего
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)
```

Результат роботи 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]]
```

Висновок: L1-нормалізація та L2-нормалізація - це методи нормалізації даних, які використовуються для перетворення векторів або матриць в одиничні вектори з різними підходами до обчислення норми (або довжини) вектора.

- У L1-нормалізації норма визначається як сума абсолютних значень всіх елементів вектора (або рядка матриці), що розраховується за формулою $||x||_1 = |x_1| + |x_2| + ... + |x_n|$. Цей метод чутливий до величини та знаку кожного окремого елемента вектора.
- У L2-нормалізації норма визначається як квадратний корінь з суми квадратів всіх елементів вектора (або рядка матриці), що розраховується за формулою $||x||_2 = \sqrt{(x_1^2 + x_2^2 + ... + x_n^2)}$. Цей метод менше чутливий до величини окремих елементів та акцентує загальну кількість енергії вектора.

Завдання 2.1.5: Кодування міток.

Лістинг програми:

```
import numpy as np
from sklearn import preprocessing

# Надання позначок вхідних даних
input_labels = ['red', 'Back', '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', 'Back']
encoded_values = encoder.transform(test_labels )
print("\nLabels =", test_labels )
print("Encoded values =", list (encoded_values ) )
```

Результат виконання програми:

```
C:\Users\skoki\PycharmProjects\labsOAI\.venv\Scripts\python.exe C:\Users\skoki\PycharmProjects\labsOAI\lab1\task2.py

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

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

Висновок: Створюється об'єкт кодувальника типу LabelEncoder, який визначається списком input_labels, що містить категоріальні мітки. Застосовуючи метод .fit(input_labels), кодувальник "навчається" на цих вхідних мітках і встановлює відповідність між мітками та числами. Після цього виводиться відображення цих міток на числа у консоль за допомогою циклу, що показує числові значення, призначені кожній унікальній мітці зі списку input_labels.

Далі проводиться перевірка кодувальника за допомогою трьох тестових міток. Ці мітки трансформуються на числові значення за допомогою кодувальника через метод .transform(test_labels). Результат цього процесу вказує, які числові коди відповідають кожній з тестових міток, що допомагає зрозуміти, як кодувальник перетворює категоріальні дані у числовий формат.

Завдання 2.2: Попередня обробка нових даних
Варіант 3:

№ варіанту	Значення змінної input_data												Поріг бінаризації
18.	4.6	3.9	-3.5	-2.9	4.1	3.3	2.2	8.8	-6.1	3.9	1.4	2.2	2.2

Лістинг програми:

```
import numpy as np
from sklearn import preprocessing

input_data = np.array(
    [[4.6, 3.9, -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=2.2).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))
# Исключение среднего
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)
```

Результат виконання програми:

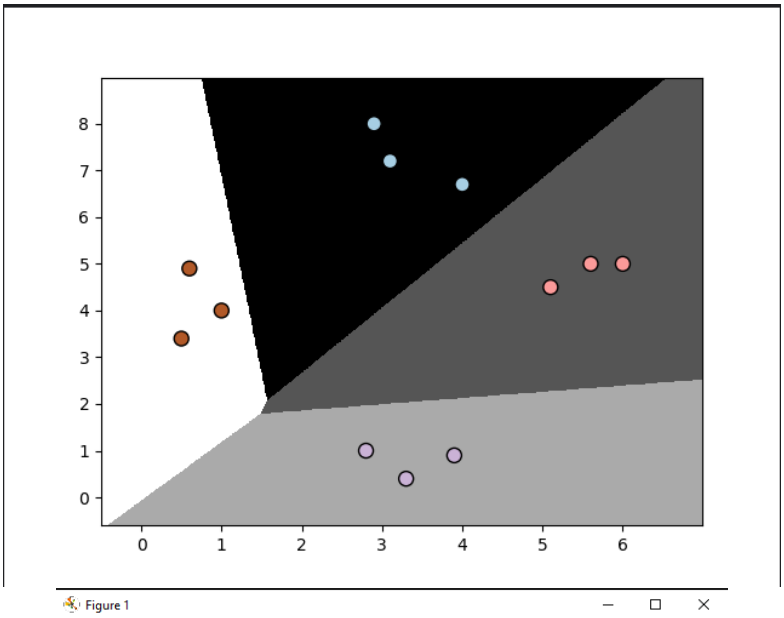
```
C:\Users\skoki\PycharmProjects\labsOAI\.venv\Scripts\python.exe C:\Users\skoki\PycharmProjects\labsOAI\lab1\task3.py

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

BEFORE:
Mean = [ 1.95   4.55  -1.025]
Std deviation = [2.93300188 2.67441582 3.9047247 ]

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

Min max scaled data:
 [[1.         0.33783784 0.27659574]
 [0.         0.36486486 1.        ]
 [0.68       1.         0.        ]
 [0.90666667 0.         0.88297872]]

l1 normalized data:
 [[ 0.38333333  0.325      -0.29166667]
 [-0.2815534   0.39805825  0.32038835]
 [ 0.12865497  0.51461988 -0.35672515]
 [ 0.52        0.18666667  0.29333333]]

l2 normalized data:
 [[ 0.65970588  0.55931585 -0.50195013]
 [-0.4825966   0.68229174  0.54916164]
 [ 0.20125974  0.80503895 -0.55803836]
 [ 0.83129388  0.29841319  0.46893501]]
```

Завдання 2.3: 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)
```

Результат виконання програми:



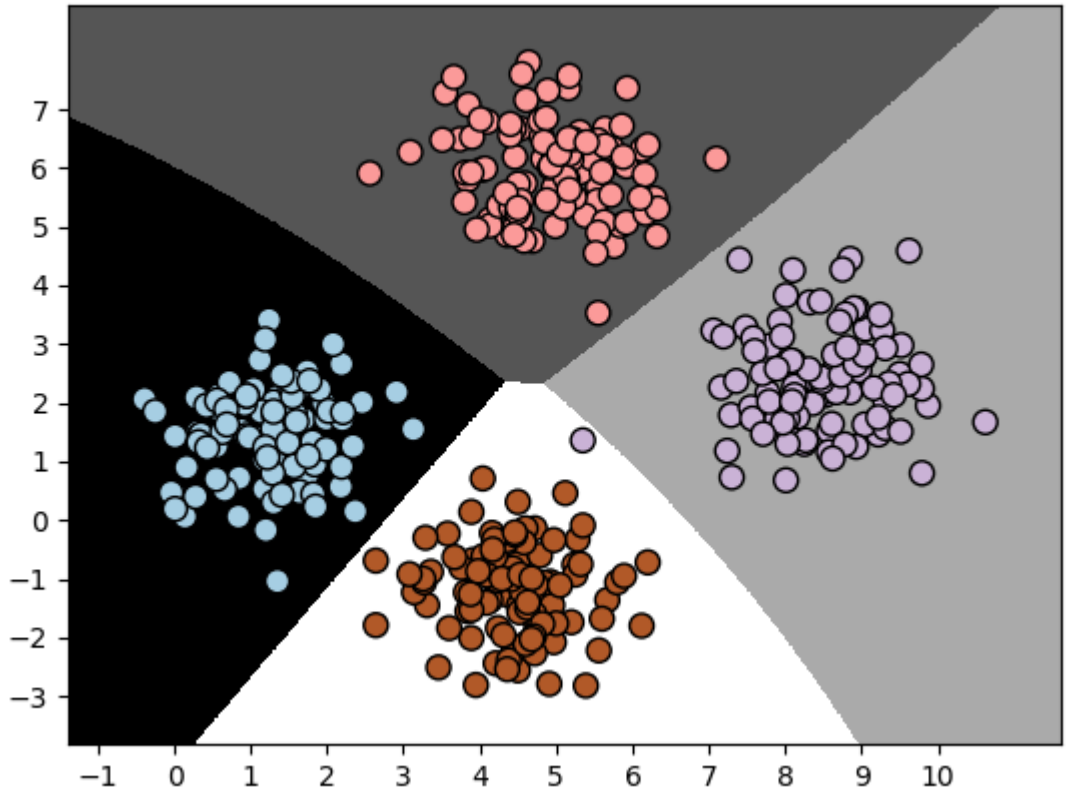
Завдання 2.4: Класифікація наївним байєсовським класифікатором.

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
from utilities import visualize_classifier

# Вхідний файл, який містить дані
input_file = "lab1/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)
```

Результат виконання програми:

```
C:\Users\skoki\PycharmProjects\labsOAI\.venv\Scripts\python.exe C:\Users\skoki\PycharmProjects\labsOAI\lab1\task5.py
Accuracy of Naive Bayes classifier = 99.75 %
```

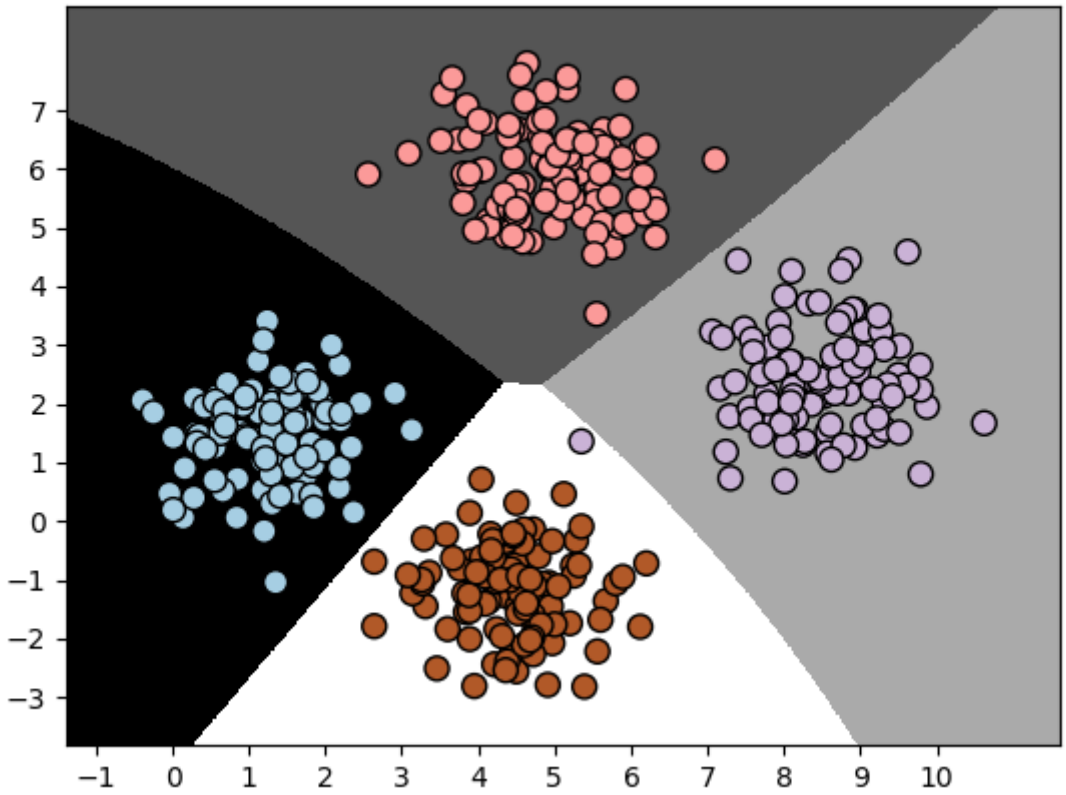


Лістинг програми після розбиття даних на навчальний та тестовий набори:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from utilities import visualize_classifier
# Вхідний файл, який містить дані
input_file = "lab1/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)
# 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)
```

Результат виконання програми:



Результати класифікації:

```
C:\Users\skoki\PycharmProjects\labsOAI\.venv\Scripts\python.exe C:\Users\skoki\PycharmProjects\labsOAI\lab1\task6.py
Accuracy of the new classifier = 100.0 %
Accuracy: 99.75%
Precision: 99.76%
Recall: 99.75%
F1: 99.75%
```

Висновок: Розділення набору даних на навчальну і тестову вибірки є ключовим етапом для оцінки ефективності моделі на нових даних, які раніше не були використані для її навчання. Це дозволяє визначити, наскільки добре модель узагальнює дані та уникає перенавчання. Додатково, модифікований підхід використовує кросвалідацію, що дозволяє отримати більш об'єктивні метрики ефективності моделі. Це досягається оцінкою моделі на кількох різних підвибірках даних, що допомагає зменшити вплив випадковості при розділенні на навчальний і тестовий набори. Такий підхід сприяє більш надійній оцінці здатності моделі до узагальнення та зниженню можливості виникнення викривлених результатів.