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/SMTH666/OAI-Lab**

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:

```
C:\Users\toxa1\PycharmProjects\lab01\venv\Scripts\python.exe C:\Users\toxa1\PycharmProjects\lab01\task2.1.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
```

Завдання 2.1.5: Кодування міток.

Лістинг програми:

```python
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:

```
PS C:\AI_labs> & E:/Users/madma/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/AI_labs/lab1/LR_1_task_1.py

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

Labels = ['green', 'red', 'Back']
Encoded values = [2, 3, 0]
PS C:\AI_labs>
```

Рисунок 2

Завдання 2.2: Попередня обробка нових даних

Варіант 23:

23.	2.5	-1.6	-6.1	-2.4	-1.2	4.3	3.2	3.1	6.1	-4.4	1.4	-1.2	2.5

Лістинг програми:

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

input_data = np.array(
    [[2.5, -1.6, -6.1], [-2.4, -1.2, 4.3], [3.2, 3.1, 6.1], [-4.4, 1.4, -1.2]]
)

# Бінаризація даних
data_binarized = preprocessing.Binarizer(threshold=2.5).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\toxa1\PycharmProjects\lab01\venv\Scripts\python.exe C:\Users\toxa1\PycharmProjects\lab01\task2.2.py

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

BEFORE:
Mean = [-0.275  0.425  0.775]
Std deviation = [3.21354555 1.92662269 4.79446295]

AFTER:
Mean = [ 0.00000000e+00 -5.55111512e-17 -4.16333634e-17]
Std deviation = [1. 1. 1.]

Min max scaled data:
 [[0.90789474 0.         0.        ]
 [0.26315789 0.08510638 0.85245902]
 [1.         1.         1.        ]
 [0.         0.63829787 0.40163934]]

l1 normalized data:
 [[ 0.24509804 -0.15686275 -0.59803922]
 [-0.30379747 -0.15189873  0.5443038 ]
 [ 0.25806452  0.25        0.49193548]
 [-0.62857143  0.2        -0.17142857]]

l2 normalized data:
 [[ 0.36852479 -0.23585586 -0.89920048]
 [-0.47351004 -0.23675502  0.84837215]
 [ 0.42362745  0.41038909  0.80753983]
 [-0.92228798  0.29345527 -0.25153308]]

Process finished with exit code 0
```

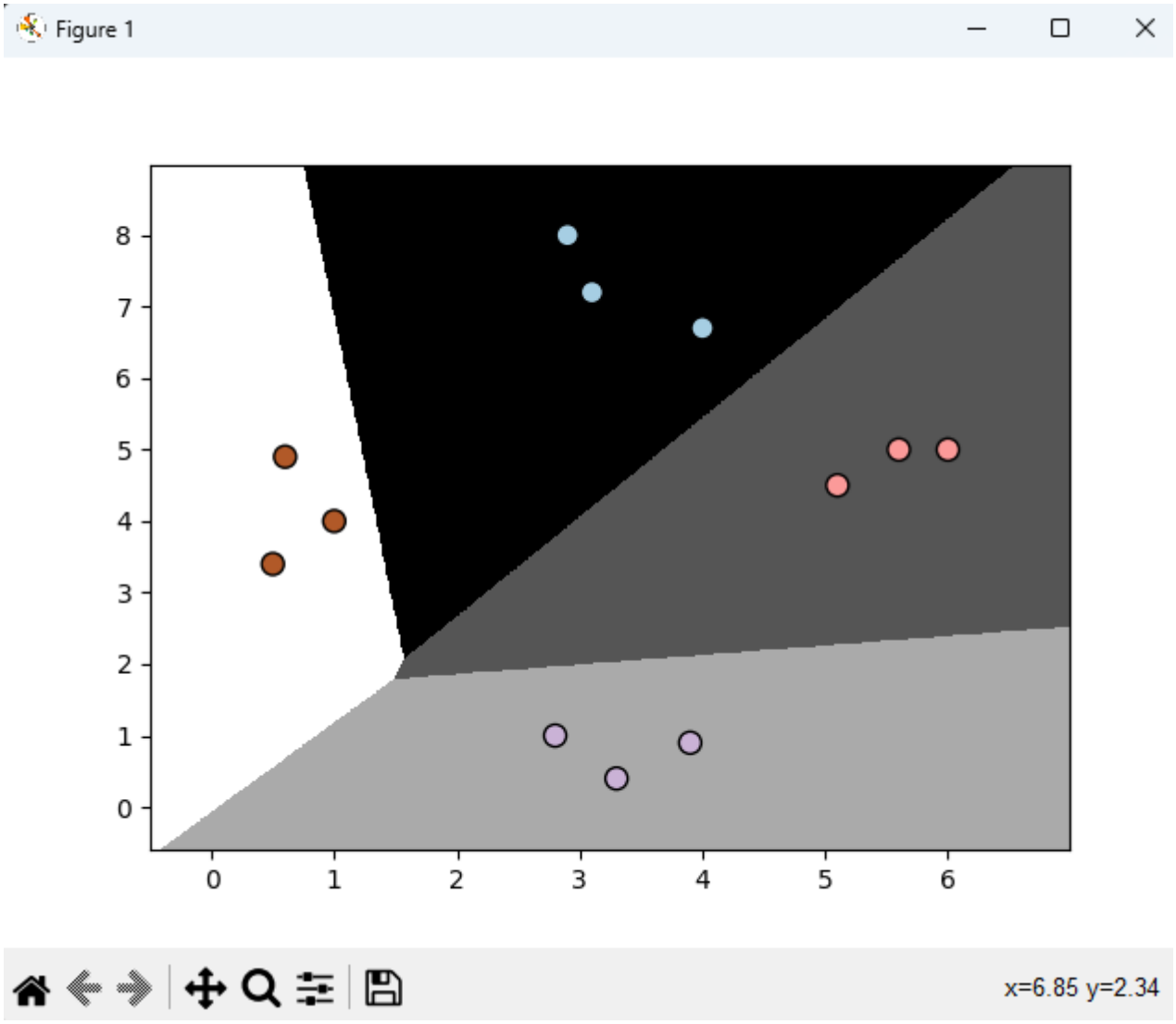
Завдання 2.3: Класифікація логістичною регресією або логістичний класифікатор

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.

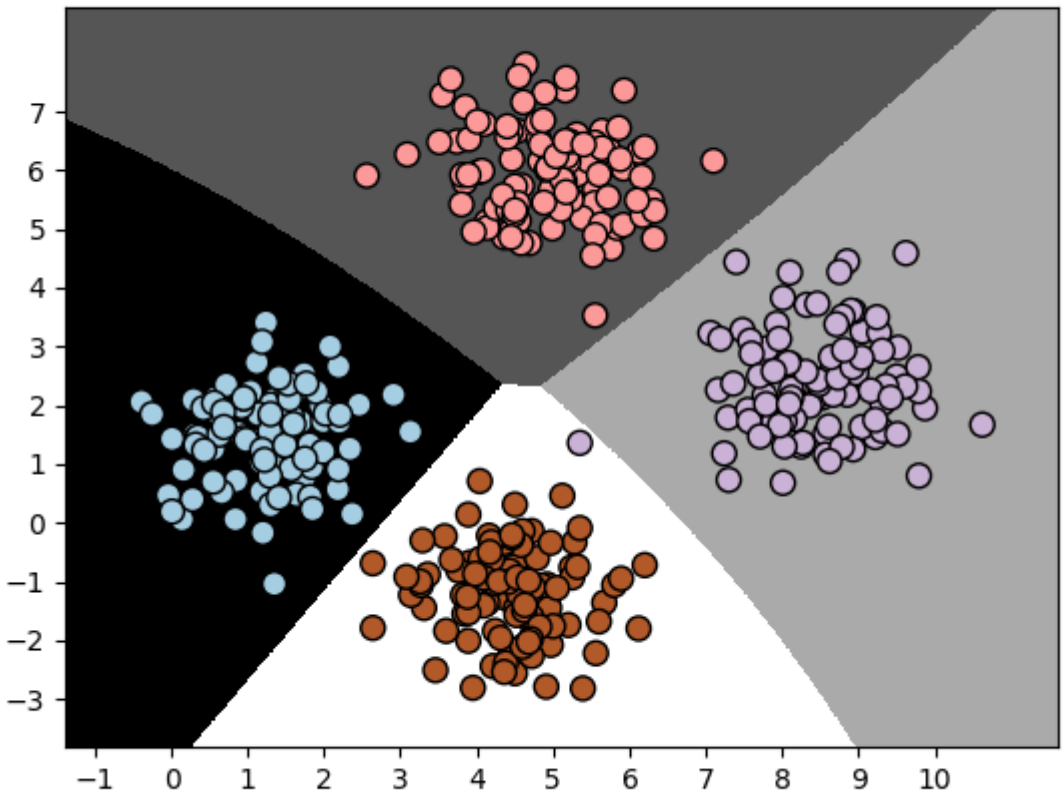
Лістинг програми:

```
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 = "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\toxa1\PycharmProjects\lab01\venv\Scripts\python.exe C:\Users\toxa1\PycharmProjects\lab01\task2.4.py
Accuracy of Naive Bayes classifier = 99.75 %

Process finished with exit code 0
```

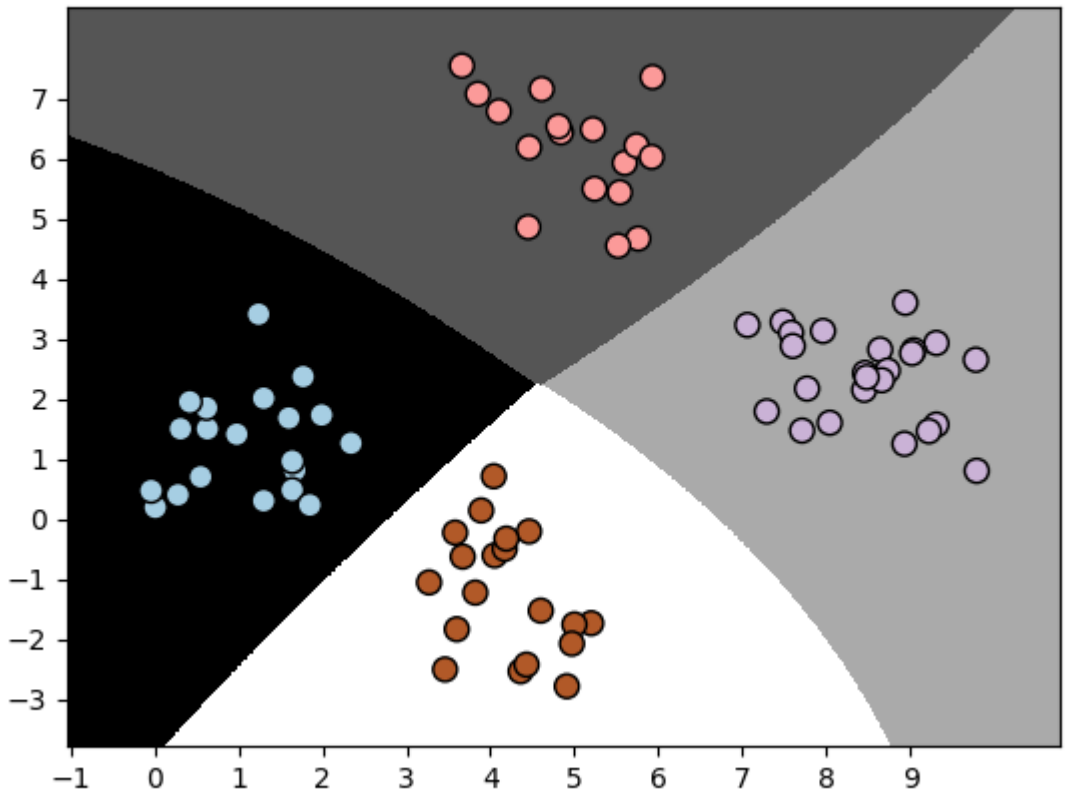


Лістинг програми після розбиття даних на навчальний та тестовий набори:

```
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\toxa1\PycharmProjects\lab01\venv\Scripts\python.exe C:\Users\toxa1\PycharmProjects\lab01\LR_1_task_4.py
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
```

Висновок: Розподіл даних на навчальні та тестові набори дозволяє оцінити ефективність моделі на невидимих для неї даних. Це сприяє визначенню того, наскільки добре модель узагальнює і уникає перенавчання. Крім того, використання скоригованого методу з крос-валідацією дозволяє отримати більш об'єктивні показники ефективності моделі, оскільки вони оцінюються на декількох різних підвибірках даних. Це сприяє зменшенню впливу випадковості при поділі на тренувальний та тестовий набори.

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.

Лістинг програми:

```python
import numpy as np
import pandas as pd
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score
from sklearn.preprocessing import StandardScaler

# Завантаження даних з файлу
data = pd.read_csv(
    "data_multivar_nb.txt", header=None, names=["Feature1", "Feature2", "Target"]
)

# Розділення даних на ознаки і мітки класів
X = data[["Feature1", "Feature2"]]
y = data["Target"]

# Розділення 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=42
)

# Нормалізація даних для SVM
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Навчання моделі SVM
svm_model = SVC(kernel="linear", C=1.0, random_state=42)
svm_model.fit(X_train_scaled, y_train)

# Прогноз на тестовому наборі для SVM
svm_predictions = svm_model.predict(X_test_scaled)

# Навчання та оцінка наївного байєсівського класифікатора
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)
nb_predictions = nb_model.predict(X_test)

# Функція для обчислення показників якості класифікації
def calculate_metrics(y_true, y_pred):
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred, average="weighted")
    recall = recall_score(y_true, y_pred, average="weighted")
    f1 = f1_score(y_true, y_pred, average="weighted")
    return accuracy, precision, recall, f1

# Розрахунок показників для SVM
svm_accuracy, svm_precision, svm_recall, svm_f1 = calculate_metrics(
    y_test, svm_predictions
)

# Розрахунок показників для наївного байєсівського класифікатора
nb_accuracy, nb_precision, nb_recall, nb_f1 = calculate_metrics(y_test, nb_predic-
tions)

# Виведення результатів
print("Результати для моделі SVM:")
print("Accuracy:", svm_accuracy)
print("Precision:", svm_precision)
```

```
print("Recall:", svm_recall)
print("F1 Score:", svm_f1)

print("\nРезультати для моделі наївного байєсівського класифікатора:")
print("Accuracy:", nb_accuracy)
print("Precision:", nb_precision)
print("Recall:", nb_recall)
print("F1 Score:", nb_f1)
```

```
C:\Users\toxa1\PycharmProjects\lab01\venv\Scripts\python.exe C:\Users\toxa1\PycharmProjects\lab01\LR_1_task_5.py
Результати для моделі SVM:
Accuracy: 0.9875
Precision: 0.9885416666666667
Recall: 0.9875
F1 Score: 0.9876263902932255

Результати для моделі наївного байєсівського класифікатора:
Accuracy: 0.9875
Precision: 0.9885416666666667
Recall: 0.9875
F1 Score: 0.9876263902932255

Process finished with exit code 0
```

Висновок: обидві моделі (SVM і наївний байєсівський класифікатор) показують ідентичні результати з точністю до чотирьох знаків після коми для всіх показників якості (Accuracy, Precision, Recall, і F1 Score), в цьому конкретному випадку можна сказати, що обидві моделі працюють однаково добре на цьому наборі даних.