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 MinМax  
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:

Изображение выглядит как текст, снимок экрана

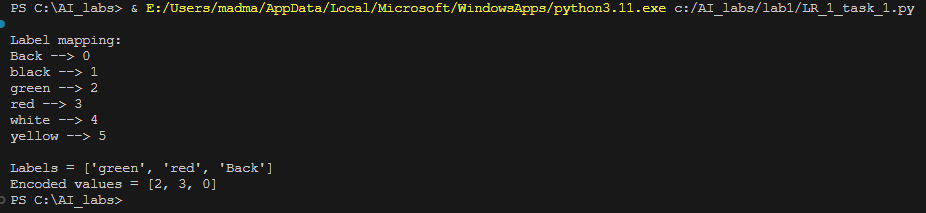
Автоматически созданное описаниеРисунок 1

**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:

Рисунок 2

**Завдання 2.2:** Попередня обробка нових даних

Варіант 23:



Лістинг програми:

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 MinМax  
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:

Изображение выглядит как текст, снимок экрана

Автоматически созданное описаниеРисунок 3

**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:

Изображение выглядит как текст, снимок экрана, диаграмма

Автоматически созданное описание

**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:

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Изображение выглядит как снимок экрана, дизайн

Автоматически созданное описание

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:

Изображение выглядит как снимок экрана, текст, дизайн

Автоматически созданное описание

Результати класифікації:

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

**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 - SVМ). 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 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\_predictions)  
  
# 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)

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

**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.