**Лабораторна робота 3**

**ДОСЛІДЖЕННЯ МЕТОДІВ РЕГРЕСІЇ ТА НЕКОНТРОЬОВАНОГО НАВЧАННЯ**

**Завдання 2.1.** Створення класифікаторів на основі випадкових та гранично випадкових лісів.

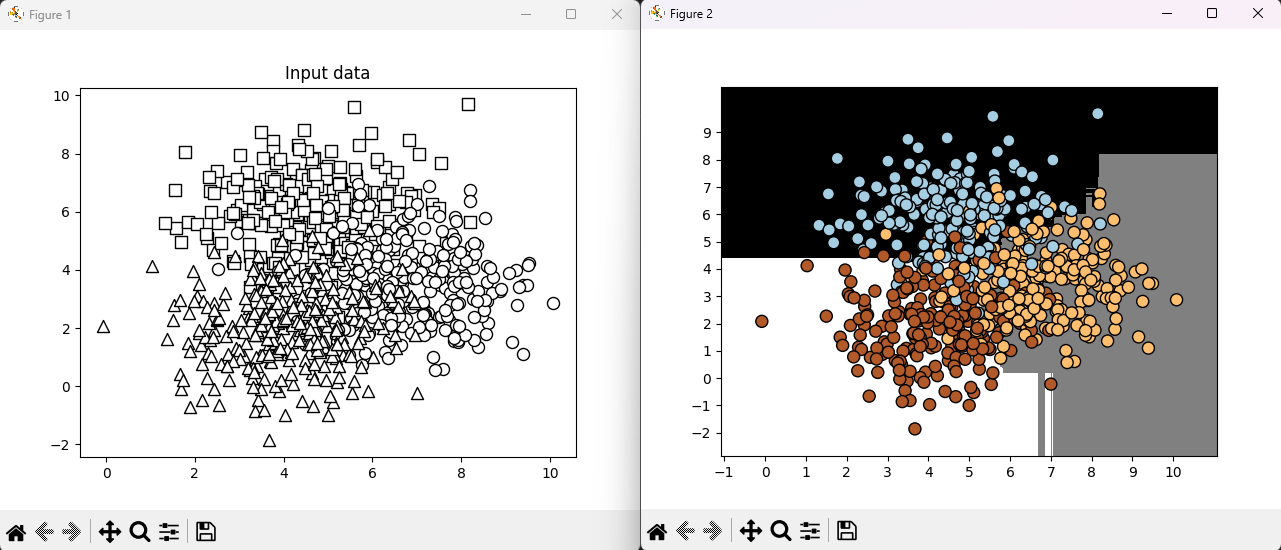
Лістинг програми:

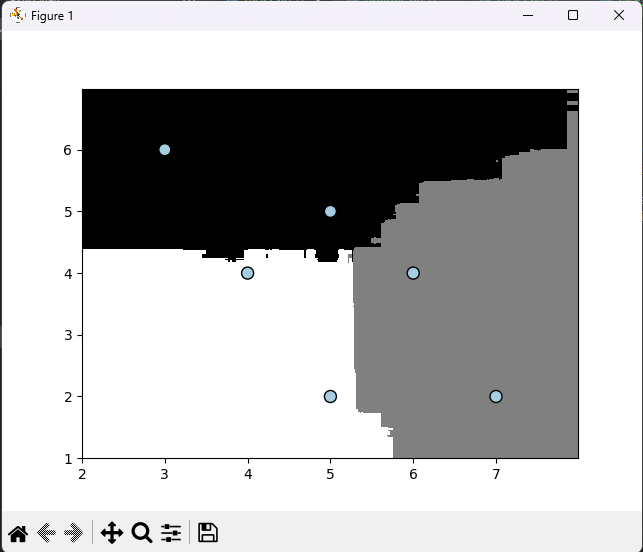
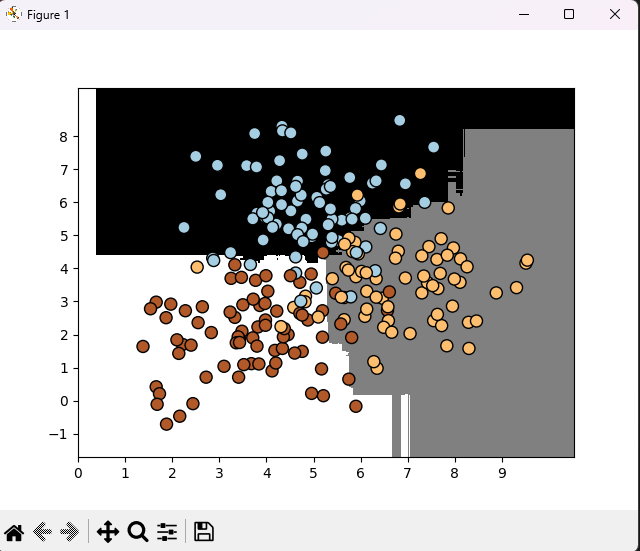
import argparse  
import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.metrics import classification\_report  
from sklearn.model\_selection import train\_test\_split  
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier  
from sklearn.metrics import classification\_report  
from utilities import visualize\_classifier  
  
  
def build\_arg\_parser():  
 parser = argparse.ArgumentParser(description='Classify data using Ensemble Learning techniques')  
 parser.add\_argument('--classifier-type', dest='classifier\_type', required=True, choices=['rf', 'erf'],  
 help="Type of classifier to use; can be either 'rf' or 'erf'")  
 return parser  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # Parse the input arguments  
 args = build\_arg\_parser().parse\_args()  
 classifier\_type = args.classifier\_type  
  
 # Load input data  
 input\_file = 'data\_random\_forests.txt'  
 data = np.loadtxt(input\_file, delimiter=',')  
 X, y = data[:, :-1], data[:, -1]  
  
 # Separate input data into three classes based on labels  
 class\_0 = np.array(X[y == 0])  
 class\_1 = np.array(X[y == 1])  
 class\_2 = np.array(X[y == 2])  
  
 # Visualize input data  
 plt.figure()  
 plt.scatter(class\_0[:, 0], class\_0[:, 1], s=75, facecolors='white', edgecolors='black', linewidth=1, marker='s')  
 plt.scatter(class\_1[:, 0], class\_1[:, 1], s=75, facecolors='white', edgecolors='black', linewidth=1, marker='o')  
 plt.scatter(class\_2[:, 0], class\_2[:, 1], s=75, facecolors='white', edgecolors='black', linewidth=1, marker='^')  
 plt.title('Input data')  
  
 # Split data into training and testing datasets  
 X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state=5)

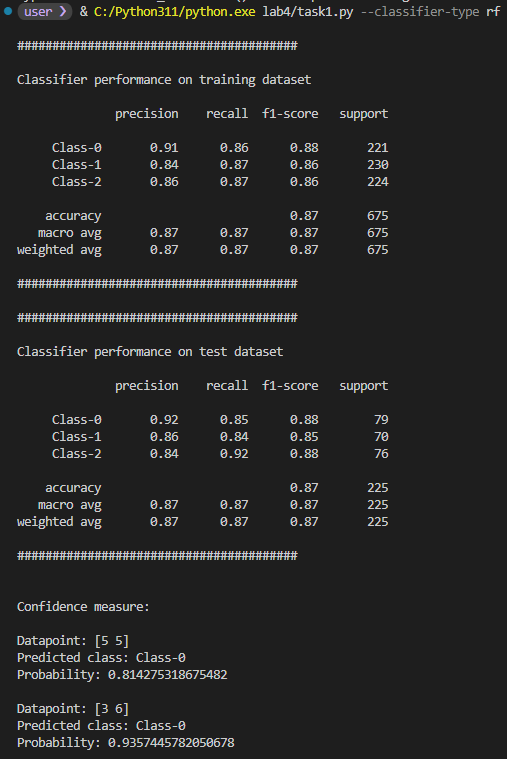
# Ensemble Learning classifier  
 params = {'n\_estimators': 100, 'max\_depth': 4, 'random\_state': 0}  
 if classifier\_type == 'rf':  
 classifier = RandomForestClassifier(\*\*params)  
 else:  
 classifier = ExtraTreesClassifier(\*\*params)  
  
 classifier.fit(X\_train, y\_train)  
 visualize\_classifier(classifier, X\_train, y\_train, 'Training dataset')  
 y\_test\_pred = classifier.predict(X\_test)  
 visualize\_classifier(classifier, X\_test, y\_test, 'Test dataset')  
  
 # Evaluate classifier performance  
 class\_names = ['Class-0', 'Class-1', 'Class-2']  
 print("\n" + "#" \* 40)  
 print("\nClassifier performance on training dataset\n")  
 print(classification\_report(y\_train, classifier.predict(X\_train), target\_names=class\_names))  
 print("#" \* 40 + "\n")  
 print("#" \* 40)  
 print("\nClassifier performance on test dataset\n")  
 print(classification\_report(y\_test, y\_test\_pred, target\_names=class\_names))  
 print("#" \* 40 + "\n")  
  
 test\_datapoints = np.array([[5, 5], [3, 6], [6, 4], [7, 2], [4, 4], [5, 2]])  
 print("\nConfidence measure:")  
 for datapoint in test\_datapoints:  
 probabilities = classifier.predict\_proba([datapoint])[0]  
 predicted\_class = 'Class-' + str(np.argmax(probabilities))  
 print('\nDatapoint:', datapoint)  
 print('Predicted class:', predicted\_class)  
 print('Probability:', np.max(probabilities))  
  
 # Visualize the datapoints  
 visualize\_classifier(classifier, test\_datapoints, [0] \* len(test\_datapoints), 'Test datapoints')  
 plt.show()

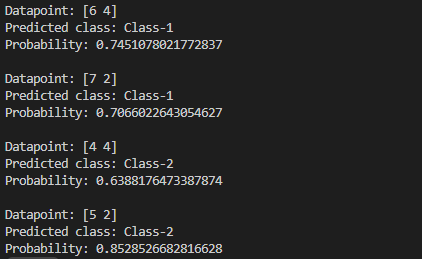
Результат виконання:

rf:

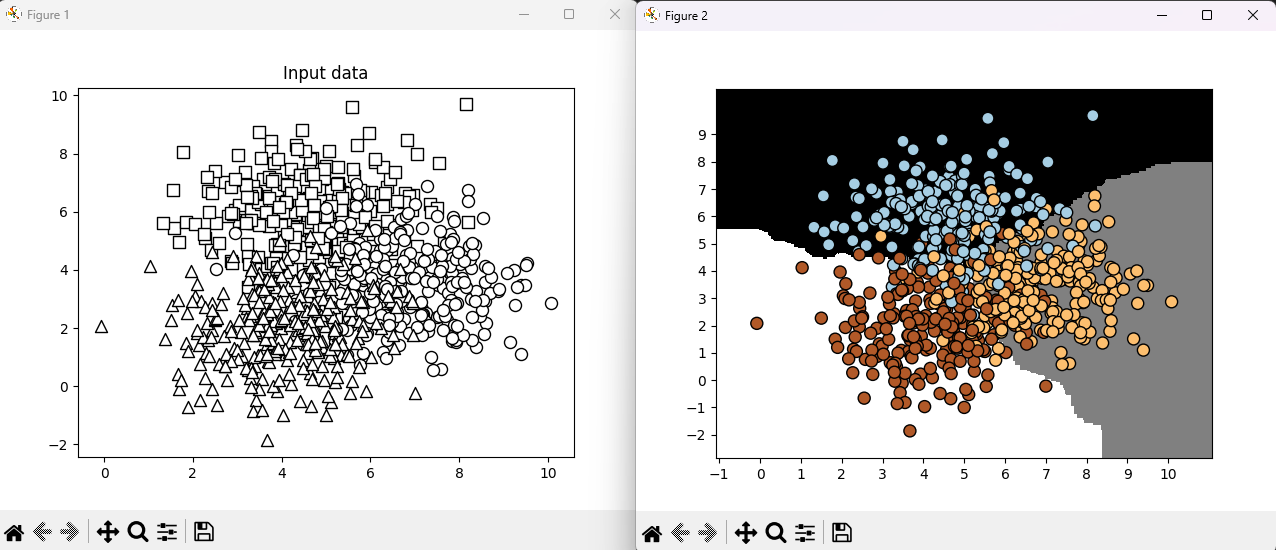


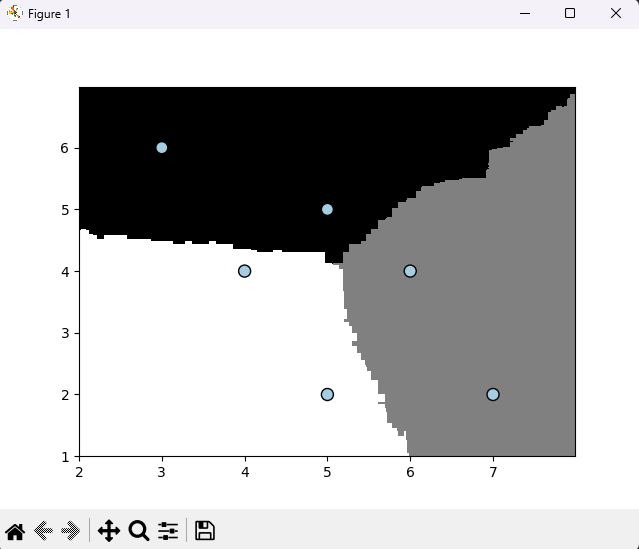
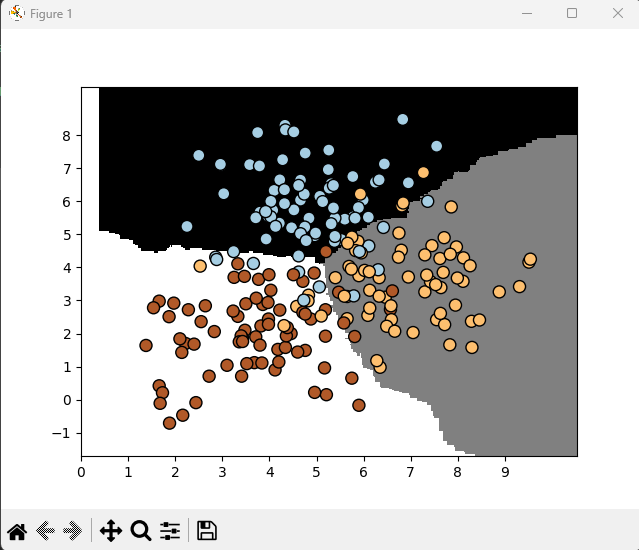


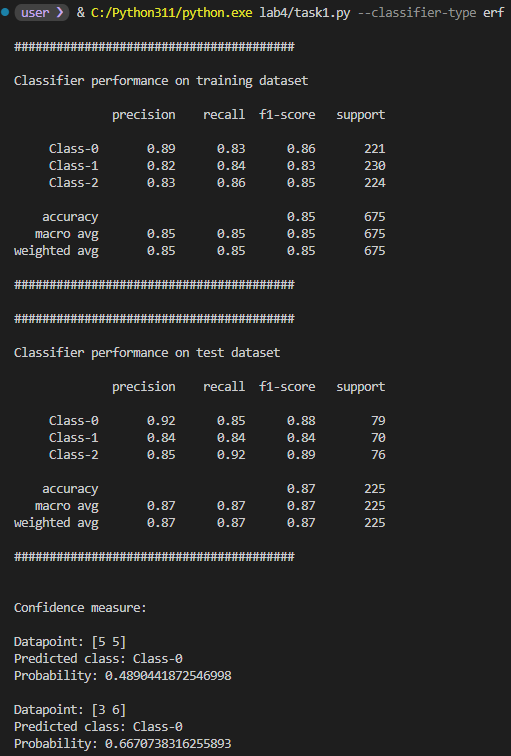


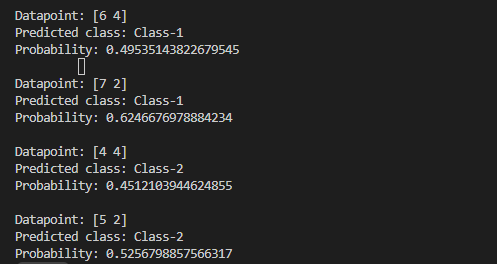


erf:







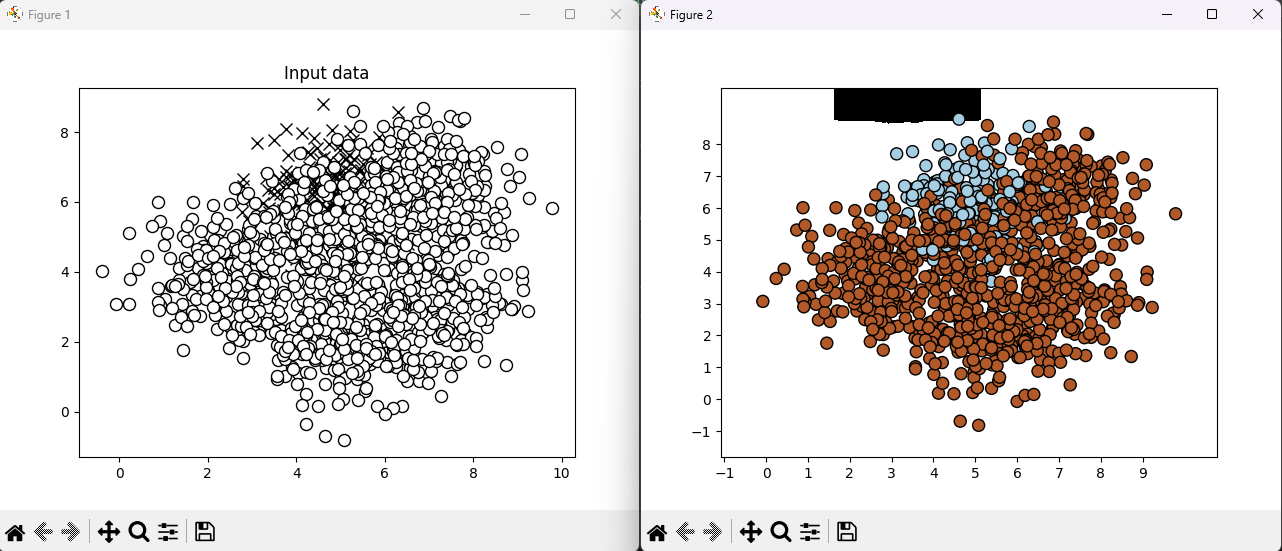


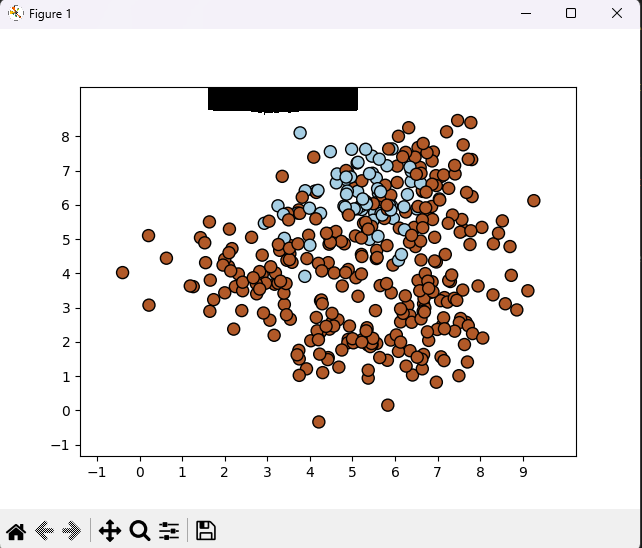
**Завдання 2.2.** Обробка дисбалансу класів.

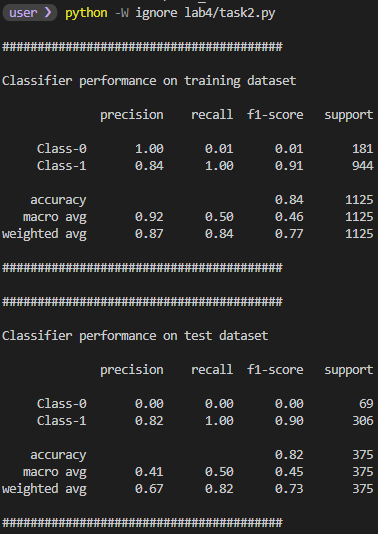
Лістинг програми:

import sys  
import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import classification\_report  
from sklearn.ensemble import ExtraTreesClassifier  
from utilities import visualize\_classifier  
  
# Input file containing data  
input\_file = 'data\_imbalance.txt'  
data = np.loadtxt(input\_file, delimiter=',')  
X, y = data[:, :-1], data[:, -1]  
  
# Separate input data into three classes based on labels  
class\_0 = np.array(X[y == 0])  
class\_1 = np.array(X[y == 1])  
  
# Visualize input data  
plt.figure()  
plt.scatter(class\_0[:, 0], class\_0[:, 1], s=75, facecolors='black', edgecolors='black', linewidth=1, marker='x')  
plt.scatter(class\_1[:, 0], class\_1[:, 1], s=75, facecolors='white', edgecolors='black', linewidth=1, marker='o')  
plt.title('Input data')  
  
# Split the data into training and testing datasets  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state=5)  
  
# Extremely Random Forests classifier  
params = {'n\_estimators': 100, 'max\_depth': 4, 'random\_state': 0}  
if len(sys.argv) > 1:  
 if sys.argv[1] == 'balance':  
 params = {'n\_estimators': 100, 'max\_depth': 4, 'random\_state': 0, 'class\_weight': 'balanced'}  
 else:  
 raise TypeError("Invalid input argument; should be 'balance'")  
  
# Create the classifier  
classifier = ExtraTreesClassifier(\*\*params)  
classifier.fit(X\_train, y\_train)  
  
# Visualize the performance of the classifier  
visualize\_classifier(classifier, X\_train, y\_train, 'Training dataset')  
  
y\_test\_pred = classifier.predict(X\_test)  
visualize\_classifier(classifier, X\_test, y\_test, 'Test dataset')  
  
# Evaluate classifier performance  
class\_names = ['Class-0', 'Class-1']  
print("\n" + "#" \* 40)  
print("\nClassifier performance on training dataset\n")  
print(classification\_report(y\_train, classifier.predict(X\_train), target\_names=class\_names))  
print("#" \* 40 + "\n")  
print("#" \* 40)  
print("\nClassifier performance on test dataset\n")  
print(classification\_report(y\_test, y\_test\_pred, target\_names=class\_names))  
print("#" \* 40 + "\n")  
plt.show()

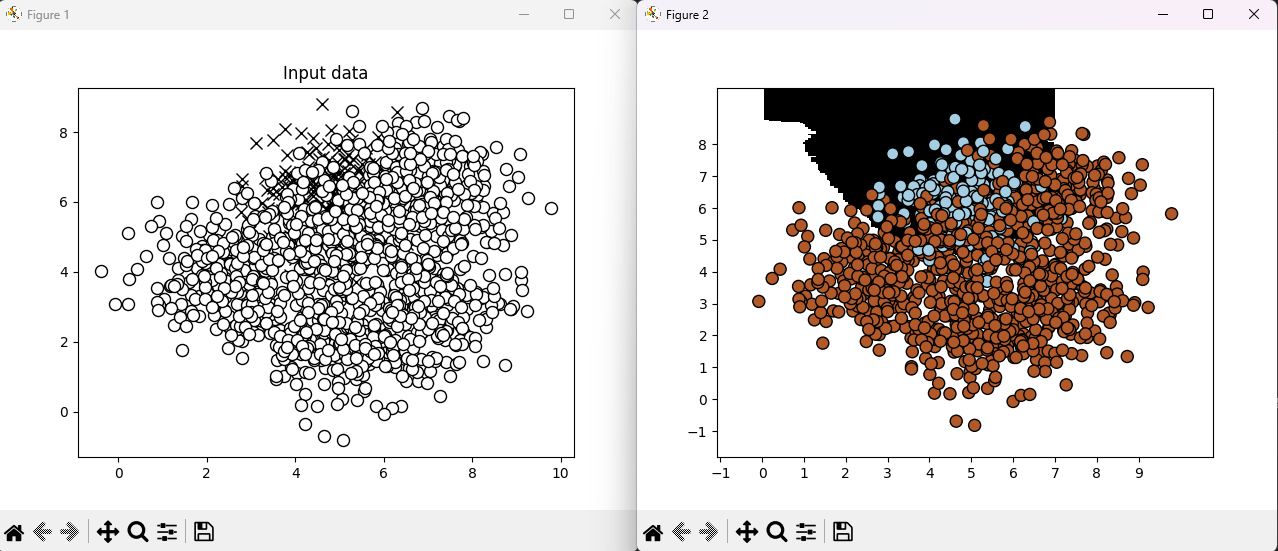
Результат виконання:

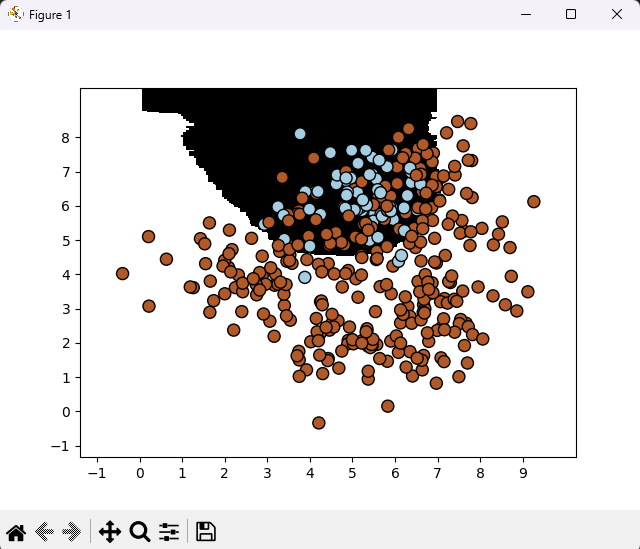


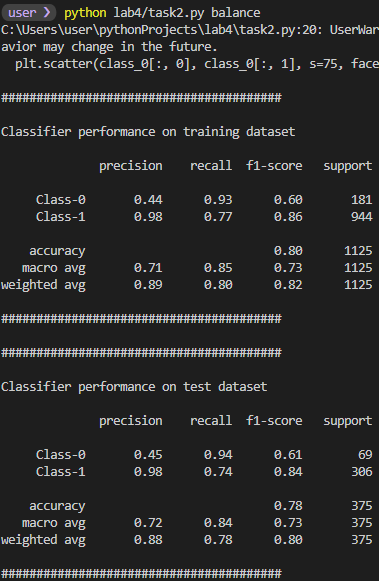




balance:







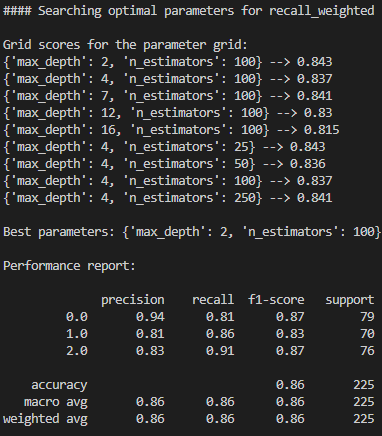
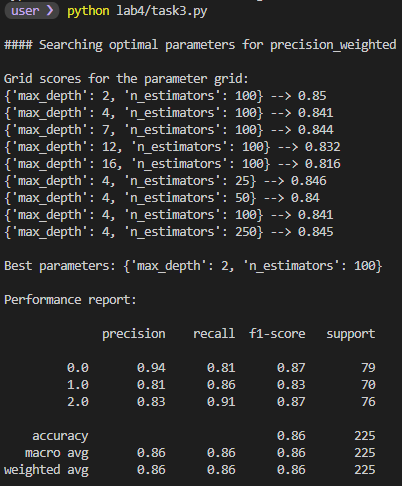
При встановленні аргументу ExtraTreesClassifier class\_weight значення balanced, алгоритм буде розраховувати ваги классів на основій розподілу їх кількості, що може компенсувати велику різницю в кількості представників класів.

**Завдання 2.3.** Знаходження оптимальних навчальних параметрів за допомогою сіткового пошуку.

Лістинг програми:

import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.model\_selection import train\_test\_split, GridSearchCV  
from sklearn.metrics import classification\_report  
from sklearn.ensemble import ExtraTreesClassifier  
  
# Load input data  
input\_file = 'data\_random\_forests.txt'  
data = np.loadtxt(input\_file, delimiter=',')  
X, y = data[:, :-1], data[:, -1]  
  
# Separate input data into three classes based on labels  
class\_0 = np.array(X[y == 0])  
class\_1 = np.array(X[y == 1])  
class\_2 = np.array(X[y == 2])  
  
# Split data into training and testing datasets  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state=5)  
  
# Define the parameter grid  
parameter\_grid = [{'n\_estimators': [100], 'max\_depth': [2, 4, 7, 12, 16]},  
 {'max\_depth': [4], 'n\_estimators': [25, 50, 100, 250]}]  
metrics = ['precision\_weighted', 'recall\_weighted']  
  
for metric in metrics:  
 print("\n#### Searching optimal parameters for", metric)  
  
 classifier = GridSearchCV(ExtraTreesClassifier(random\_state=0), parameter\_grid, cv=5, scoring=metric)  
 classifier.fit(X\_train, y\_train)  
  
 print("\nGrid scores for the parameter grid:")  
 for i in classifier.cv\_results\_['params']:  
 print(i, '-->', round(classifier.cv\_results\_['mean\_test\_score'][classifier.cv\_results\_['params'].index(i)], 3))  
  
 print("\nBest parameters:", classifier.best\_params\_)  
  
 y\_pred = classifier.predict(X\_test)  
 print("\nPerformance report:\n")  
 print(classification\_report(y\_test, y\_pred))

Результат виконання:



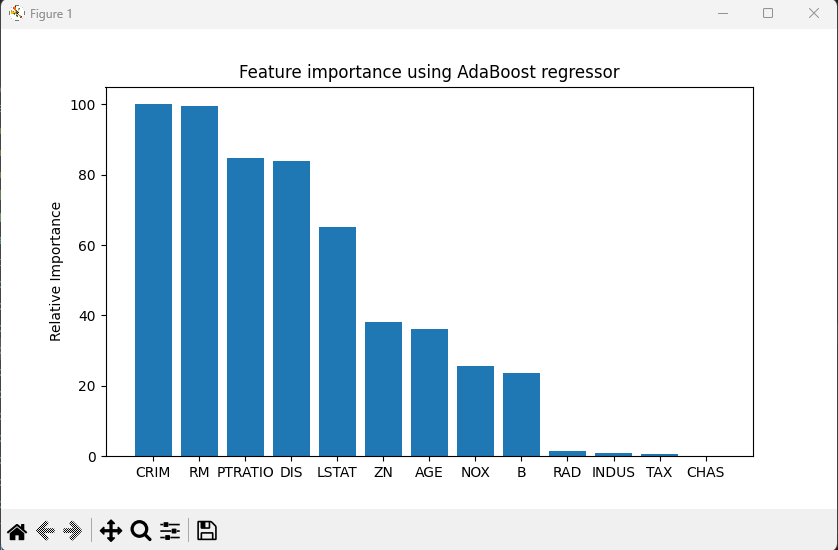
GridSearchCV використовується для автоматичного пошуку оптимальних параметрів моделі. Він робить це шляхом виконання систематичного пошуку через сітку параметрів моделі, використовуючи крос-валідацію.

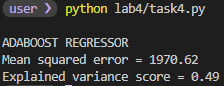
**Завдання 2.4.** Обчислення відносної важливості ознак.

Лістинг програми:

import numpy as np  
import matplotlib.pyplot as plt  
import pandas as pd  
from sklearn import preprocessing  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.ensemble import AdaBoostClassifier  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import mean\_squared\_error, explained\_variance\_score  
from sklearn.utils import shuffle  
  
# Load housing data from UCI repository  
data\_url = "http://lib.stat.cmu.edu/datasets/boston"  
raw\_df = pd.read\_csv(data\_url, sep="\s+", skiprows=22, header=None)  
housing\_data = np.hstack([raw\_df.values[::2, :], raw\_df.values[1::2, :2]])  
target = raw\_df.values[1::2, 2]  
  
# Encode labels  
label\_encoder = preprocessing.LabelEncoder()  
y = label\_encoder.fit\_transform(target)  
  
# Shuffle the data  
X, y = shuffle(housing\_data, y, random\_state=7)  
  
# Split the data into training and testing datasets  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=7)  
  
# AdaBoost Regressor model  
regressor = AdaBoostClassifier(DecisionTreeClassifier(max\_depth=4), n\_estimators=400, random\_state=7)  
regressor.fit(X\_train, y\_train)  
  
# Compute performance metrics  
y\_pred = regressor.predict(X\_test)  
mse = mean\_squared\_error(y\_test, y\_pred)  
evs = explained\_variance\_score(y\_test, y\_pred)  
print("\nADABOOST REGRESSOR")  
print("Mean squared error =", round(mse, 2))  
print("Explained variance score =", round(evs, 2))  
  
# Extract feature importances  
feature\_importances = regressor.feature\_importances\_  
feature\_names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT']  
  
# Normalize the importance values  
feature\_importances = 100.0 \* (feature\_importances / max(feature\_importances))  
  
# Sort the values and flip them  
index\_sorted = np.flipud(np.argsort(feature\_importances))  
  
# Arrange the X ticks  
pos = np.arange(index\_sorted.shape[0]) + 0.5  
  
# Plot the bar graph  
plt.figure()  
plt.bar(pos, feature\_importances[index\_sorted], align='center')  
plt.xticks(pos, [feature\_names[i] for i in index\_sorted])  
plt.ylabel('Relative Importance')  
plt.title('Feature importance using AdaBoost regressor')  
plt.show()

Результат виконання:





* CRIM - рівень злочинності на душу населення по містах
* ZN - частка земель житлової забудови, виділених на ділянки площею понад 25 000 кв.
* INDUS - частка акрів нероздрібного бізнесу на місто.
* CHAS – фіктивна змінна річки Чарльз (1, якщо тракт обмежує річку; 0 інакше)
* NOX - концентрація оксидів азоту (частини на 10 мільйонів)
* RM - середня кількість кімнат на житло
* AGE - частка квартир, які займають власники, побудованих до 1940 року
* DIS - зважені відстані до п'яти бостонських центрів зайнятості
* RAD - індекс доступності радіальних магістралей
* TAX - ставка податку на повну вартість майна за 10 000 доларів США
* PTRATIO - співвідношення учень/вчитель за містом
* B - 1000(Bk - 0,63)^2, де Bk - частка темношкірих у містах
* LSTAT - % нижчий статус населення
* MEDV – середня вартість будинків, зайнятих власниками, у 1000 доларів США

Діаграма показує величину впливу на ціну житла перелічених ознак. Відповідно до графіка, найбільший вплив мають ознаки CRIM, RM, PTRATIO та DIS. А ознаками RAD, INDUS, TAX та CHAS можна знехтувати, оскільки вони мають дуже малий вплив.

**Завдання 2.5.** Прогнозування інтенсивності дорожнього руху за допомогою класифікатора на основі гранично випадкових лісів.

Лістинг програми:

import numpy as np  
import matplotlib.pyplot as plt  
from sklearn import preprocessing  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import classification\_report, mean\_absolute\_error  
from sklearn.ensemble import ExtraTreesRegressor  
  
# Load input data  
input\_file = 'traffic\_data.txt'  
data = []  
with open(input\_file, 'r') as f:  
 for line in f.readlines():  
 items = line[:-1].split(',')  
 data.append(items)  
  
data = np.array(data)  
  
# Convert string data to numerical data  
label\_encoder = []  
X\_encoded = np.empty(data.shape)  
for i, item in enumerate(data[0]):  
 if item.isdigit():  
 X\_encoded[:, i] = data[:, i]  
 else:  
 label\_encoder.append(preprocessing.LabelEncoder())  
 X\_encoded[:, i] = label\_encoder[-1].fit\_transform(data[:, i])  
  
X = X\_encoded[:, :-1].astype(int)  
y = X\_encoded[:, -1].astype(int)  
  
# Split data into training and testing datasets  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state=5)  
  
# Extremely Random Forests regressor  
params = {'n\_estimators': 100, 'max\_depth': 4, 'random\_state': 0}  
regressor = ExtraTreesRegressor(\*\*params)  
regressor.fit(X\_train, y\_train)  
  
# Compute the regressor performance on test data  
y\_pred = regressor.predict(X\_test)  
print("Mean absolute error:", round(mean\_absolute\_error(y\_test, y\_pred), 2))  
  
# Testing encoding on single data instance  
test\_datapoint = ['Saturday', '10:20', 'Atlanta', 'no']  
test\_datapoint\_encoded = [-1] \* len(test\_datapoint)  
count = 0  
for i, item in enumerate(test\_datapoint):  
 if item.isdigit():  
 test\_datapoint\_encoded[i] = int(test\_datapoint[i])  
 else:  
 test\_datapoint\_encoded[i] = int(label\_encoder[count].transform([test\_datapoint[i]])[0])  
 count += 1  
  
test\_datapoint\_encoded = np.array(test\_datapoint\_encoded)  
  
# Predict and print output for a particular datapoint  
print("Predicted traffic:", int(regressor.predict([test\_datapoint\_encoded])[0]))

Результат виконання:

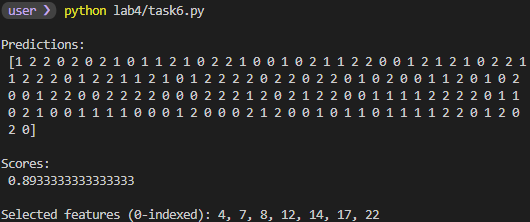


**Завдання 2.6.** Створення навчального конвеєра (конвеєра машинного навчання).

Лістинг програми:

from sklearn.datasets import \_samples\_generator  
from sklearn.feature\_selection import SelectKBest, f\_regression  
from sklearn.pipeline import Pipeline  
from sklearn.ensemble import ExtraTreesClassifier  
  
# Generate sample data  
X, y = \_samples\_generator.make\_classification(n\_samples=150, n\_features=25, n\_classes=3, n\_informative=6, n\_redundant=0, random\_state=7)  
  
# Select features using the chi-squared test  
selector\_k\_best = SelectKBest(f\_regression, k=9)  
  
# Build the machine learning pipeline  
classifier = ExtraTreesClassifier(n\_estimators=60, max\_depth=4)  
  
pipeline\_classifier = Pipeline([('selector', selector\_k\_best), ('erf', classifier)])  
  
# Establish the parameters  
pipeline\_classifier.set\_params(selector\_\_k=7, erf\_\_n\_estimators=30)  
  
# Training the classifier  
pipeline\_classifier.fit(X, y)  
  
# Predict the output  
prediction = pipeline\_classifier.predict(X)  
print("\nPredictions:\n", prediction)  
  
# Print scores  
print("\nScores:\n", pipeline\_classifier.score(X, y))  
  
# Print the selected features chosen by the selector  
features\_status = pipeline\_classifier.named\_steps['selector'].get\_support()  
  
# Get selected feature indices  
selected\_features = [i for i, x in enumerate(features\_status) if x]  
print("\nSelected features (0-indexed):", ', '.join([str(x) for x in selected\_features]))

Результат виконання:

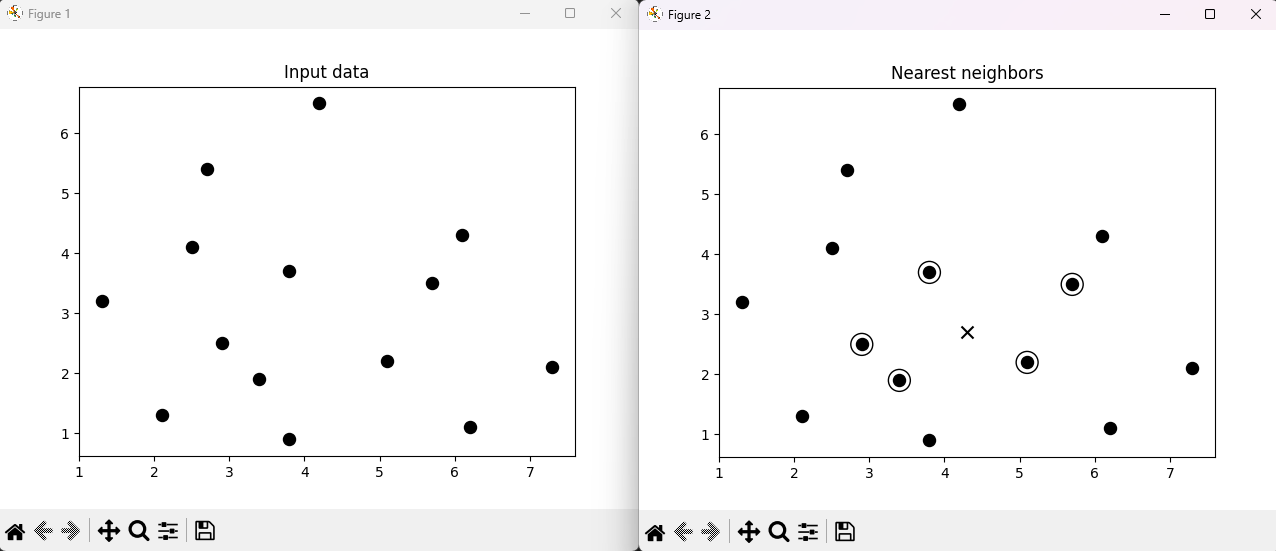


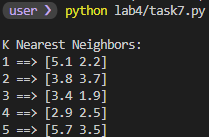
**Завдання 2.7.** Пошук найближчих сусідів.

Лістинг програми:

import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.neighbors import NearestNeighbors  
  
# Input data  
X = np.array(  
 [[2.1, 1.3], [1.3, 3.2], [2.9, 2.5], [2.7, 5.4], [3.8, 0.9], [7.3, 2.1], [4.2, 6.5], [3.8, 3.7], [2.5, 4.1],  
 [3.4, 1.9], [5.7, 3.5], [6.1, 4.3], [5.1, 2.2], [6.2, 1.1]])  
k = 5  
  
# Query point  
query\_point = np.array([[4.3, 2.7]])  
  
# Plot input data  
plt.figure()  
plt.title('Input data')  
plt.scatter(X[:, 0], X[:, 1], marker='o', s=75, color='black')  
  
# Build K Nearest Neighbors model  
knn\_model = NearestNeighbors(n\_neighbors=k, algorithm='ball\_tree').fit(X)  
distances, indices = knn\_model.kneighbors(query\_point)  
  
# Print the 'k' nearest neighbors  
print("\nK Nearest Neighbors:")  
for rank, index in enumerate(indices[0][:k], start=1):  
 print(str(rank) + " ==>", X[index])  
  
# Visualize the nearest neighbors along with the query point  
plt.figure()  
plt.title('Nearest neighbors')  
plt.scatter(X[:, 0], X[:, 1], marker='o', s=75, color='k')  
plt.scatter(X[indices][0][:][:, 0], X[indices][0][:][:, 1], marker='o', s=250, color='k', facecolors='none')  
plt.scatter(query\_point[:, 0], query\_point[:, 1], marker='x', s=75, color='k')  
plt.show()

Результат виконання:





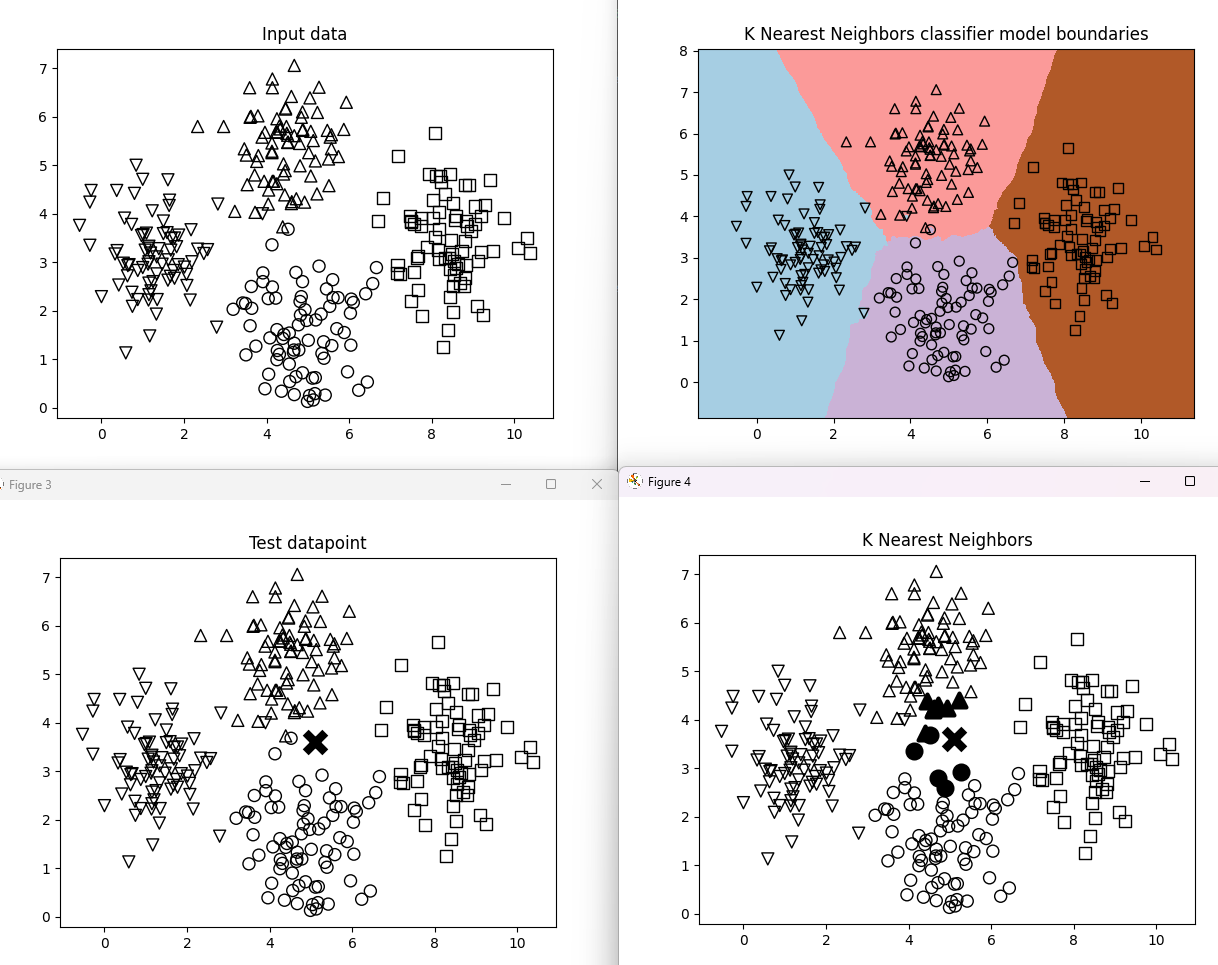
На графіку 1 відображено вхідні дані. На графіку 2 відображено 5 найближчих сусідів обраної тестової точки. В терміналі відображено “координати” найближчих сусідів точки.

**Завдання 2.8.** Створити класифікатор методом k найближчих сусідів.

Лістинг програми:

import numpy as np  
import matplotlib.pyplot as plt  
import matplotlib.cm as cm  
from sklearn import neighbors  
  
# Load input data  
input\_file = 'data.txt'  
data = np.loadtxt(input\_file, delimiter=',')  
X, y = data[:, :-1], data[:, -1].astype(int)  
  
# Visualize input data  
plt.figure()  
plt.title('Input data')  
marker\_shapes = 'v^os'  
mapper = [marker\_shapes[i] for i in y]  
for i in range(X.shape[0]):  
 plt.scatter(X[i, 0], X[i, 1], marker=mapper[i], s=75, edgecolors='black', facecolors='none')  
  
# Number of nearest neighbors to consider  
num\_neighbors = 12  
step\_size = 0.01  
  
# Create a K Nearest Neighbors classifier model  
classifier = neighbors.KNeighborsClassifier(num\_neighbors, weights='distance')  
  
# Train the K Nearest Neighbors model  
classifier.fit(X, y)  
  
# Create the mesh to plot the boundaries  
x\_min, x\_max = min(X[:, 0]) - 1, max(X[:, 0]) + 1  
y\_min, y\_max = min(X[:, 1]) - 1, max(X[:, 1]) + 1  
x\_values, y\_values = np.meshgrid(np.arange(x\_min, x\_max, step\_size), np.arange(y\_min, y\_max, step\_size))  
  
# Evaluate the classifier on all the points on the grid  
output = classifier.predict(np.c\_[x\_values.ravel(), y\_values.ravel()])  
  
# Visualize the predicted output  
output = output.reshape(x\_values.shape)  
plt.figure()  
plt.pcolormesh(x\_values, y\_values, output, cmap=cm.Paired)  
  
# Overlay the training points on the map  
for i in range(X.shape[0]):  
 plt.scatter(X[i, 0], X[i, 1], marker=mapper[i], s=50, edgecolors='black', facecolors='none')  
  
plt.xlim(x\_values.min(), x\_values.max())  
plt.ylim(y\_values.min(), y\_values.max())  
plt.title('K Nearest Neighbors classifier model boundaries')  
  
# Test input datapoint  
test\_datapoint = [5.1, 3.6]  
plt.figure()  
plt.title('Test datapoint')  
for i in range(X.shape[0]):  
 plt.scatter(X[i, 0], X[i, 1], marker=mapper[i], s=75, edgecolors='black', facecolors='none')  
  
plt.scatter(test\_datapoint[0], test\_datapoint[1], marker='x', linewidth=6, s=200, facecolors='black')  
  
# Extract the K nearest neighbors  
\_, indices = classifier.kneighbors([test\_datapoint])  
indices = indices.astype(int)[0]  
  
# Plot k nearest neighbors  
plt.figure()  
plt.title('K Nearest Neighbors')  
for i in indices:  
 plt.scatter(X[i, 0], X[i, 1], marker=mapper[y[i]], linewidth=3, s=100, facecolors='black')  
  
plt.scatter(test\_datapoint[0], test\_datapoint[1], marker='x', linewidth=6, s=200, facecolors='black')  
  
for i in range(X.shape[0]):  
 plt.scatter(X[i, 0], X[i, 1], marker=mapper[i], s=75, edgecolors='black', facecolors='none')  
  
print("Predicted output:", classifier.predict([test\_datapoint])[0])  
plt.show()

Результат виконання:





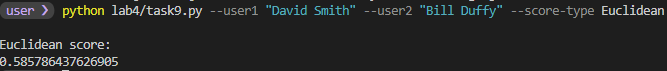
На графіку 1 (рахунок зліва зверху) зображено вхідні дані. На графіку 2 зображено межі класифікатора даних. На графіку 3 зображено обрану тестову точку. На графіку 4 зображено 12 найближчих сусідів тестової точки.

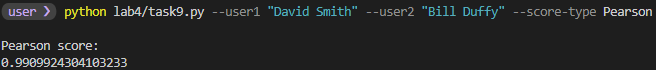
**Завдання 2.9.** Обчислення оцінок подібності.

Лістинг програми:

import argparse  
import numpy as np  
import json  
  
  
def build\_arg\_parser():  
 parser = argparse.ArgumentParser(description='Compute similarity score')  
 parser.add\_argument('--user1', dest='user1', required=True, help='First user')  
 parser.add\_argument('--user2', dest='user2', required=True, help='Second user')  
 parser.add\_argument('--score-type', dest='score\_type', required=True, choices=['Euclidean', 'Pearson'],  
 help='Similarity metric to be used')  
 return parser  
  
  
# Compute the Euclidean distance score between user1 and user2  
def euclidean\_score(dataset, user1, user2):  
 if user1 not in dataset:  
 raise TypeError('Cannot find ' + user1 + ' in the dataset')  
  
 if user2 not in dataset:  
 raise TypeError('Cannot find ' + user2 + ' in the dataset')  
  
 # Movies rated by both user1 and user2  
 common\_movies = {}  
  
 for item in dataset[user1]:  
 if item in dataset[user2]:  
 common\_movies[item] = 1  
  
 # If there are no common movies between the users, then the score is 0   
 if len(common\_movies) == 0:  
 return 0  
  
 squared\_diff = []  
  
 for item in dataset[user1]:  
 if item in dataset[user2]:  
 squared\_diff.append(np.square(dataset[user1][item] - dataset[user2][item]))  
  
 return 1 / (1 + np.sqrt(np.sum(squared\_diff)))  
  
  
# Compute the Pearson correlation score between user1 and user2  
def pearson\_score(dataset, user1, user2):  
 if user1 not in dataset:  
 raise TypeError('Cannot find ' + user1 + ' in the dataset')  
  
 if user2 not in dataset:  
 raise TypeError('Cannot find ' + user2 + ' in the dataset')  
  
 # Movies rated by both user1 and user2  
 common\_movies = {}  
  
 for item in dataset[user1]:  
 if item in dataset[user2]:  
 common\_movies[item] = 1  
  
 # If there are no common movies between user1 and user2, then the score is 0   
 num\_ratings = len(common\_movies)  
  
 if num\_ratings == 0:  
 return 0  
  
 # Calculate the sum of ratings of all the common movies  
 user1\_sum = np.sum([dataset[user1][item] for item in common\_movies])  
 user2\_sum = np.sum([dataset[user2][item] for item in common\_movies])  
  
 # Calculate the sum of squares of ratings of all the common movies  
 user1\_squared\_sum = np.sum([np.square(dataset[user1][item]) for item in common\_movies])  
 user2\_squared\_sum = np.sum([np.square(dataset[user2][item]) for item in common\_movies])  
  
 # Calculate the sum of products of the ratings of the common movies  
 sum\_of\_products = np.sum([dataset[user1][item] \* dataset[user2][item] for item in common\_movies])  
  
 # Calculate the Pearson correlation score  
 Sxy = sum\_of\_products - (user1\_sum \* user2\_sum / num\_ratings)  
 Sxx = user1\_squared\_sum - np.square(user1\_sum) / num\_ratings  
 Syy = user2\_squared\_sum - np.square(user2\_sum) / num\_ratings  
  
 if Sxx \* Syy == 0:  
 return 0  
  
 return Sxy / np.sqrt(Sxx \* Syy)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 args = build\_arg\_parser().parse\_args()  
 user1 = args.user1  
 user2 = args.user2  
 score\_type = args.score\_type  
  
 ratings\_file = 'lab4/data/ratings.json'  
  
 with open(ratings\_file, 'r') as f:  
 data = json.loads(f.read())  
  
 if score\_type == 'Euclidean':  
 print("\nEuclidean score:")  
 print(euclidean\_score(data, user1, user2))  
 else:  
 print("\nPearson score:")  
 print(pearson\_score(data, user1, user2))

Результат виконання:







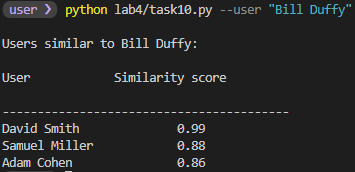
Алгоритм визначає оцінку схожості користувачів на основі їх оцінок фільмів. Оцінка розраховується алгоритмом Евклідової відстані та алгоритмом кореляційного коефіцієнта Пірсона.

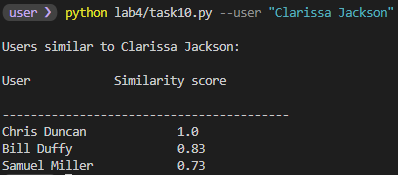
**Завдання 2.10.** Пошук користувачів зі схожими уподобаннями методом колаборативної фільтрації.

Лістинг програми:

import argparse  
import numpy as np  
import json  
from lr\_4\_task\_9 import pearson\_score, euclidean\_score  
  
def build\_arg\_parser():  
 parser = argparse.ArgumentParser(description='Find users who are similar to the input user')  
 parser.add\_argument('--user', dest='user', required=True, help='Input user')  
 return parser  
  
def find\_similar\_users(dataset, user, num\_users, metric='pearson'):  
 if user not in dataset:  
 raise TypeError('Cannot find ' + user + ' in the dataset')  
   
 # Compute Pearson score between user and all the users in the dataset  
 if metric == 'pearson':  
 scores = np.array([[x, pearson\_score(dataset, user, x)] for x in dataset if user != x])  
 else:  
 scores = np.array([[x, euclidean\_score(dataset, user, x)] for x in dataset if user != x])  
   
 # Sort the scores in decreasing order  
 scores\_sorted = np.argsort(scores[:, 1])[::-1]  
   
 # Extract top 'k' indices  
 top\_k = scores\_sorted[:num\_users]  
   
 return scores[top\_k]  
  
if \_\_name\_\_=='\_\_main\_\_':  
 args = build\_arg\_parser().parse\_args()  
 user = args.user  
   
 raitings\_file = 'lab4/data/ratings.json'  
 with open(raitings\_file, 'r') as f:  
 data = json.loads(f.read())  
   
 print("\nUsers similar to " + user + ":\n")  
 similar\_users = find\_similar\_users(data, user, 3)  
 print("User\t\tSimilarity score\n")  
 print("-"\*41)  
 for item in similar\_users:  
 print(item[0], '\t\t', round(float(item[1]), 2))

Результат виконання:





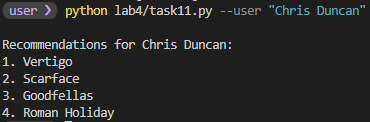
Алгорим знаходить найбільш схожих в уподобаннях користувачів спираючись на оцінку їх схожості, що розраховується тими ж алгоритмами що і в попередньому завданні.

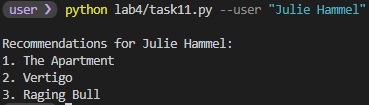
**Завдання 2.11.** Створення рекомендаційної системи фільмів.

Лістинг програми:

import argparse  
import numpy as np  
import json  
from lr\_4\_task\_9 import pearson\_score  
  
  
def build\_arg\_parser():  
 parser = argparse.ArgumentParser(description='Recommendation system')  
 parser.add\_argument('--user', dest='user', required=True, help='Input user')  
 return parser  
  
  
# Generate recommendations for the input user  
def generate\_recommendations(dataset, user):  
 if user not in dataset:  
 raise TypeError('Cannot find ' + user + ' in the dataset')  
  
 total\_scores = {}  
 similarity\_sums = {}  
  
 for u in [x for x in dataset if x != user]:  
 similarity\_score = pearson\_score(dataset, user, u)  
  
 if similarity\_score <= 0:  
 continue  
  
 filetered\_list = [x for x in dataset[u] if x not in dataset[user] or dataset[user][x] == 0]  
  
 for item in filetered\_list:  
 total\_scores.update({item: dataset[u][item] \* similarity\_score})  
 similarity\_sums.update({item: similarity\_score})  
  
 if len(total\_scores) == 0:  
 return ['No recommendations possible']  
  
 # Create the normalized list  
 movie\_ranks = np.array([[total / similarity\_sums[item], item] for item, total in total\_scores.items()])  
  
 # Sort in decreasing order based on the first column  
 movie\_ranks = movie\_ranks[np.argsort(movie\_ranks[:, 0])[::-1]]  
  
 # Extract the recommended movies  
 recommendations = [movie for \_, movie in movie\_ranks]  
  
 return recommendations  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 args = build\_arg\_parser().parse\_args()  
 user = args.user  
  
 raitings\_file = 'lab4/data/ratings.json'  
 with open(raitings\_file, 'r') as f:  
 data = json.loads(f.read())  
  
 print("\nRecommendations for " + user + ":")  
 movies = generate\_recommendations(data, user)  
 for i, movie in enumerate(movies):  
 print(str(i + 1) + '. ' + movie)

Результат виконання:





Алгоритм пропонує користувачу фільми, які користувачі схожі на нього оцінили високо, проте які він ще не дивився. Рекомендації виконуються на тій же оцінці та алгоритмах.