**ЛАБОРАТОРНА РОБОТА №4**

**ДОСЛІДЖЕННЯ МЕТОДІВ АНСАМБЛЕВОГО НАВЧАННЯ ТА СТВОРЕННЯ РЕКОМЕНДАЦІЙНИХ СИСТЕМ**

**Мета роботи**: використовуючи спеціалізовані бібліотеки та мову програмування Python дослідити методи ансамблів у машинному навчанні та створити рекомендаційні системи.

Хід роботи:

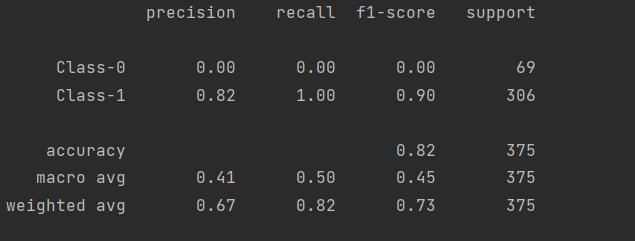
Завдання 2.1. Створення класифікаторів на основі випадкових та гранично випадкових лісів

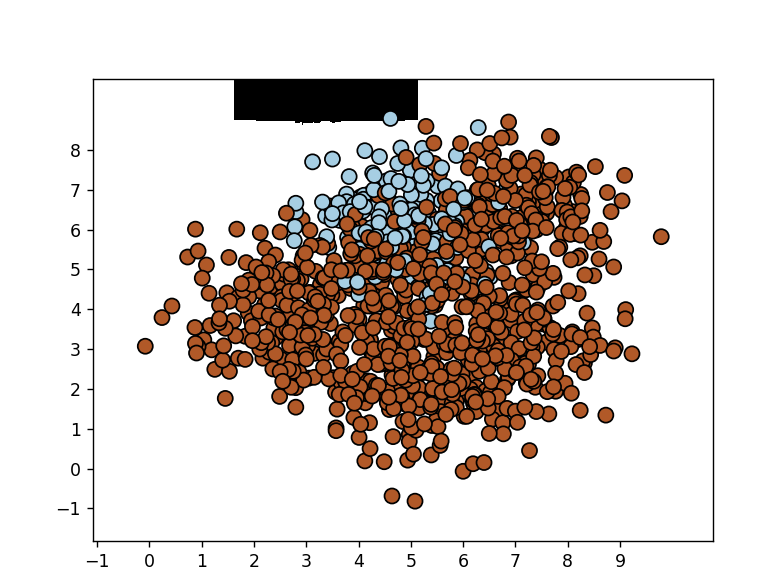
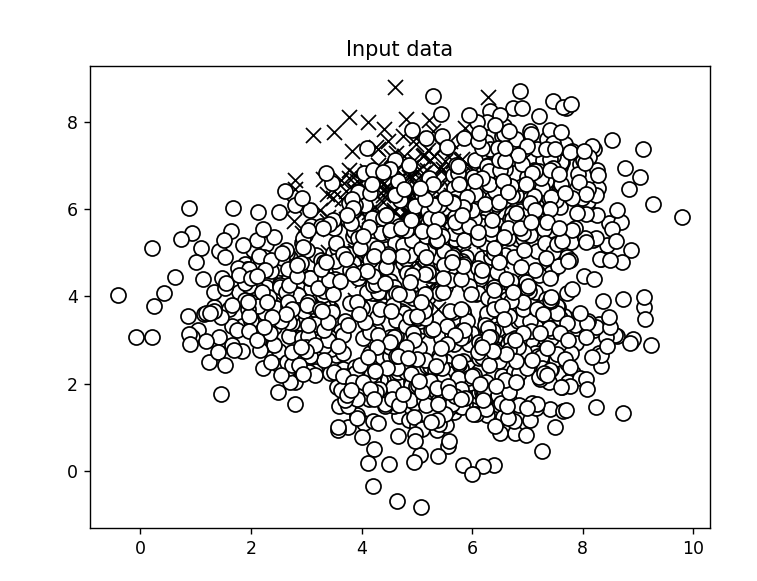
Лістинг програми:

import argparse  
import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.metrics import classification\_report  
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier  
from utilities import visualize\_classifier  
from sklearn.model\_selection import cross\_val\_score, train\_test\_split  
# Парсер аргументів  
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\_\_':  
 args = build\_arg\_parser().parse\_args()  
 classifier\_type = args.classifier\_type  
 input\_file = 'data\_random\_forests.txt'  
 data = np.loadtxt(input\_file, delimiter=',')  
 X, Y = data[:, :-1], data[:, -1]  
 print(X)  
 class\_0 = np.array(X[Y == 0])  
 class\_1 = np.array(X[Y == 1])  
 class\_2 = np.array(X[Y == 2])  
 plt.figure()  
 plt.scatter(class\_0[:, 0], class\_0[:, 1], s=75, facecolors='red', edgecolors='black', linewidth=1, marker='s')  
 plt.scatter(class\_1[:, 0], class\_1[:, 1], s=75, facecolors='green', edgecolors='black', linewidth=1, marker='o')  
 plt.scatter(class\_2[:, 0], class\_2[:, 1], s=75, facecolors='blue', edgecolors='black', linewidth=1, marker='^')  
plt.title('Input data')  
plt.show()  
X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.25, random\_state=5)  
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')  
class\_names = ['Class-0', 'Class-1', 'Class-2']  
print("\n" + "#" \* 40)  
print("\nClassifier performance on training dataset\n")  
Y\_train\_pred = classifier.predict(X\_train)  
print(classification\_report(Y\_train, Y\_train\_pred, target\_names=class\_names))  
print("#" \* 40 + "\n")  
print("#" \* 40)  
print("\nClassifier performance on test dataset\n")  
Y\_test\_pred = classifier.predict(X\_test)  
print(classification\_report(Y\_test, Y\_test\_pred, target\_names=class\_names))  
print("#" \* 40 + "\n")

Завдання 2.2. Обробка дисбалансу класів

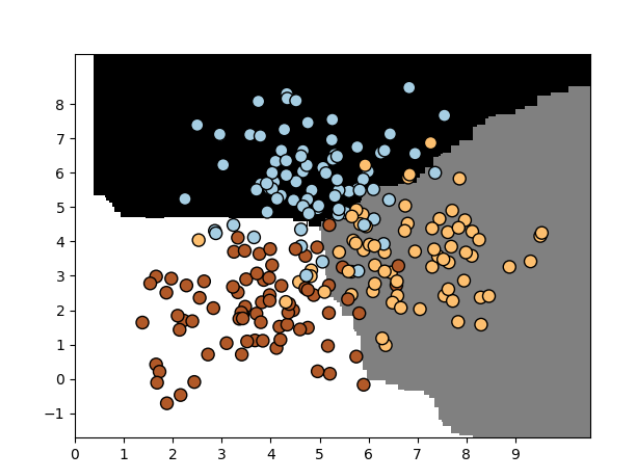
import sys  
import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.model\_selection import train\_test\_split  
from sklearn.ensemble import ExtraTreesClassifier  
from sklearn.metrics import classification\_report  
from utilities import visualize\_classifier  
if \_\_name\_\_ == '\_\_main\_\_':  
 input\_file = 'data\_imbalance.txt'  
 data = np.loadtxt(input\_file, delimiter=',')  
 X, Y = data[:, :-1], data[:, -1]  
 # Поділ вхідних даних на два класи на підставі міток  
 class\_0 = np.array(X[Y == 0])  
 class\_1 = np.array(X[Y == 1])  
 # Візуалізація вхідних даних  
 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')  
 X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.25, random\_state=5)  
 params = {'n\_estimators': 100, 'max\_depth': 4, 'random\_state': 0}  
  
 if len(sys.argv) > 1:  
 if sys.argv[1] == 'balance':  
 params['class\_weight'] = 'balanced'  
 else:  
 raise TypeError("Invalid input argument; should be 'balance' or nothing")  
 classifier = ExtraTreesClassifier(\*\*params)  
 classifier.fit(X\_train, Y\_train)  
 visualize\_classifier(classifier, X\_train, Y\_train)  
 Y\_test\_pred = classifier.predict(X\_test)  
 class\_names = ['Class-0', 'Class-1']  
 print("\n" + "#"\*40)  
 print("Classifier performance on training dataset")  
 print(classification\_report(Y\_test, Y\_test\_pred, target\_names=class\_names))  
 print("#"\*40)  
 print("Classifier performance on test dataset")  
 print(classification\_report(Y\_test, Y\_test\_pred, target\_names=class\_names))  
 print("#"\*40 + "\n")  
 plt.show()

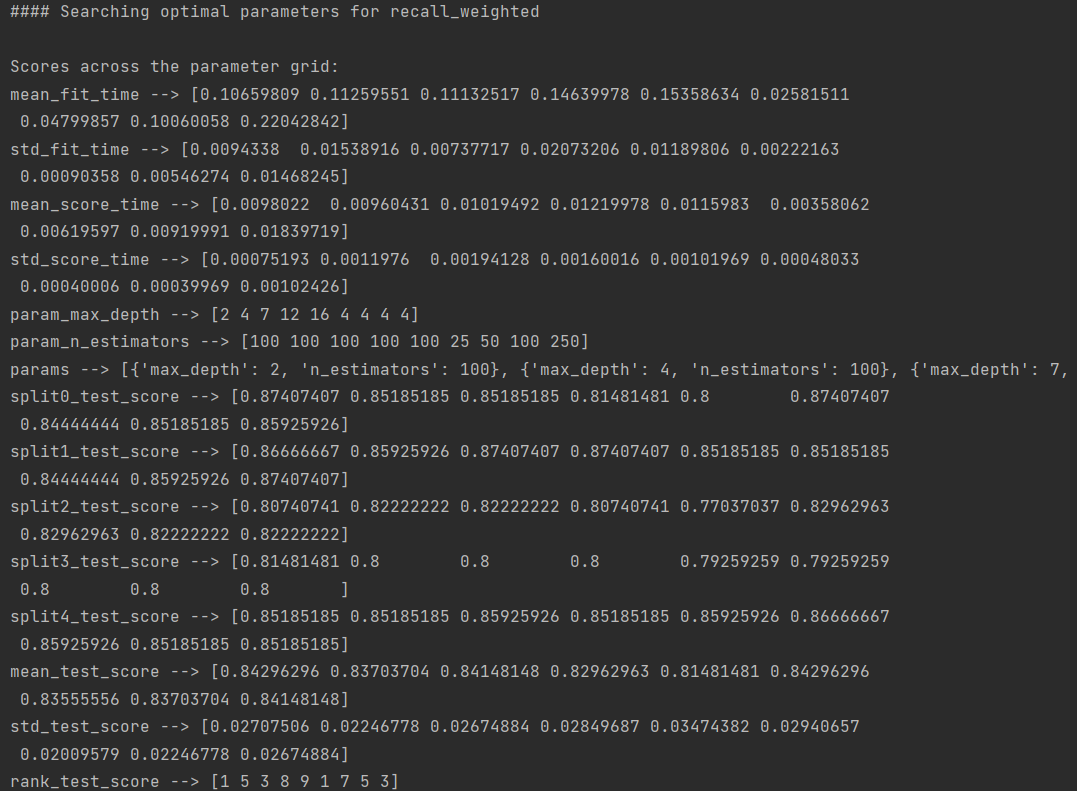
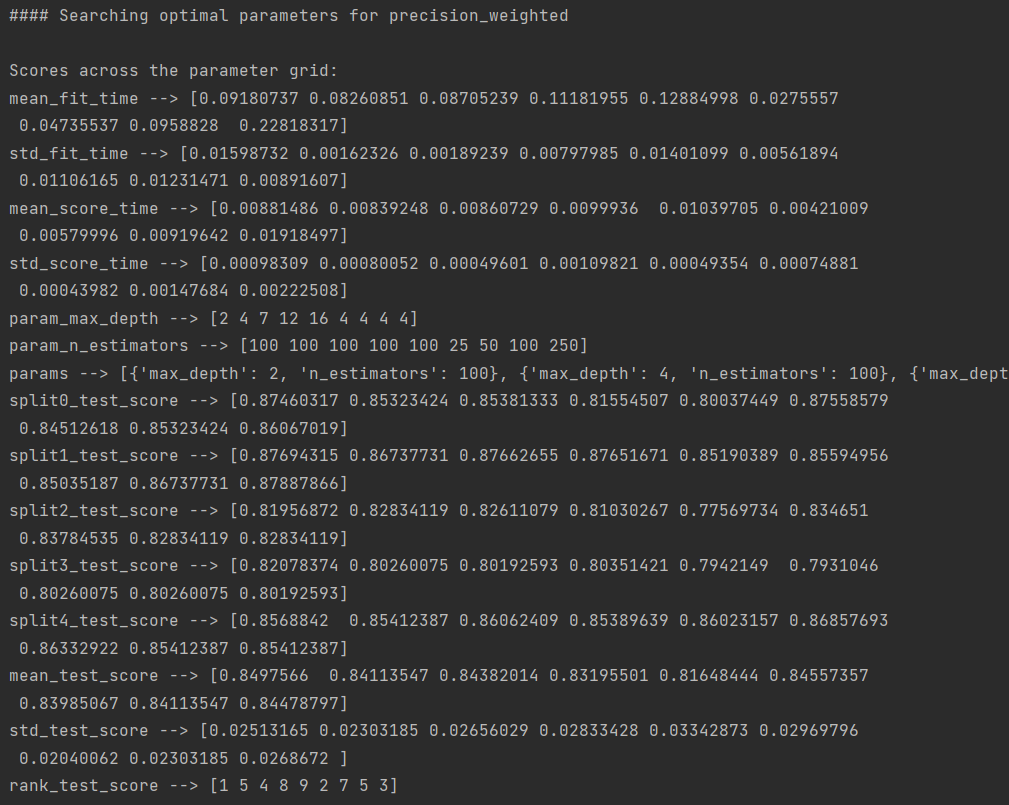




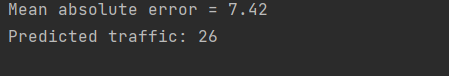
Завдання 2.3. Знаходження оптимальних навчальних параметрів за допомогою сіткового пошуку

import numpy as np  
from sklearn.model\_selection import train\_test\_split, GridSearchCV  
from sklearn.ensemble import ExtraTreesClassifier  
from sklearn.metrics import classification\_report  
from utilities import visualize\_classifier  
input\_file = 'data\_random\_forests.txt'  
data = np.loadtxt(input\_file, delimiter=',')  
X, Y = data[:, :-1], data[:, -1]  
class\_0 = np.array(X[Y == 0])  
class\_1 = np.array(X[Y == 1])  
class\_2 = np.array(X[Y == 2])  
X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.25, random\_state=5)  
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("#### Searching optimal parameters for", metric)  
 classifier = GridSearchCV(ExtraTreesClassifier(random\_state=0), parameter\_grid, cv=5, scoring=metric)  
 classifier.fit(X\_train, Y\_train)  
 print("\nScores across the parameter grid:")  
 for params, avg\_score in classifier.cv\_results\_.items():  
 print(params, '-->', avg\_score)  
 print("\nHighest scoring parameter set:", classifier.best\_params\_)  
 Y\_test\_pred = classifier.predict(X\_test)  
 class\_names = ['Class-0', 'Class-1', 'Class-2']  
 print("#"\*40)  
 print("Classifier performance on training dataset")  
 print(classification\_report(Y\_test, Y\_test\_pred, target\_names=class\_names))  
 print("#"\*40 + "\n")  
 visualize\_classifier(classifier, X\_test, Y\_test)





Завдання 2.4. Обчислення відносної важливості ознак

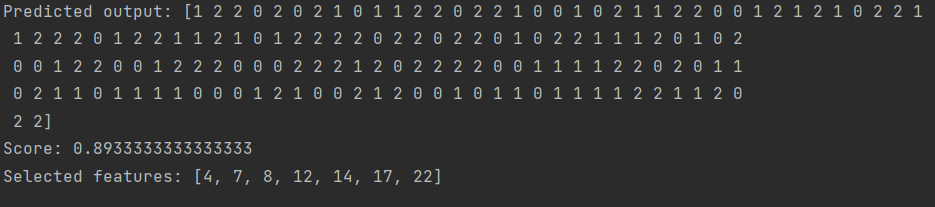


Завдання 2.5. Прогнозування інтенсивності дорожнього руху за допомогою класифікатора на основі гранично випадкових лісів

import numpy as np  
from sklearn.metrics import mean\_absolute\_error  
from sklearn.model\_selection import train\_test\_split  
from sklearn.ensemble import ExtraTreesRegressor  
from sklearn import preprocessing  
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)  
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)  
X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.25, random\_state=5)  
params = {'n\_estimators': 100, 'max\_depth': 4, 'random\_state': 0}  
regressor = ExtraTreesRegressor(\*\*params)  
regressor.fit(X\_train, Y\_train)  
Y\_pred = regressor.predict(X\_test)  
print("Mean absolute error =", round(mean\_absolute\_error(Y\_test, Y\_pred), 2))  
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]]))  
 count = count + 1  
test\_datapoint\_encoded = np.array(test\_datapoint\_encoded)  
print("Predicted traffic:", int(regressor.predict([test\_datapoint\_encoded])[0]))

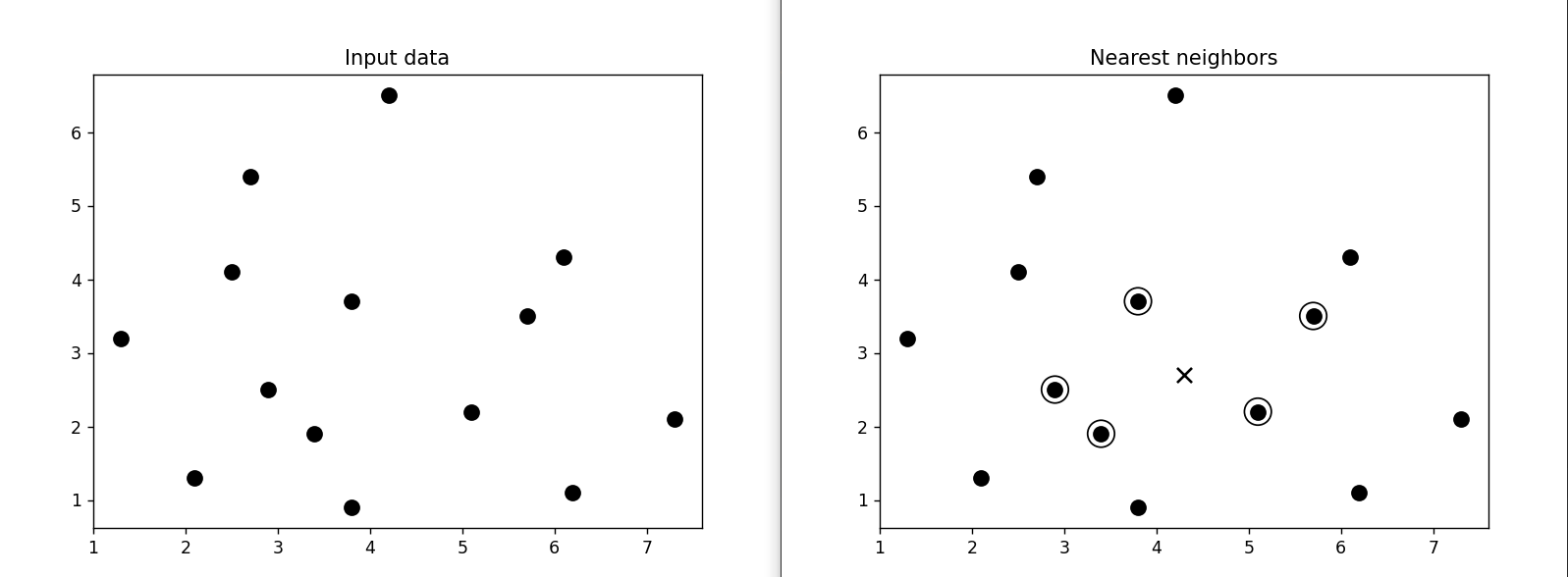
Завдання 2.6. Створення навчального конвеєра (конвеєра машинного навчання)

from sklearn.datasets import \_samples\_generator  
from sklearn.feature\_selection import SelectKBest, f\_regression  
from sklearn.pipeline import Pipeline  
from sklearn.ensemble import ExtraTreesClassifier  
X, Y = \_samples\_generator.make\_classification(n\_samples=150, n\_features=25, n\_classes=3, n\_informative=6, n\_redundant=0, random\_state=7)  
k\_best\_selector = SelectKBest(f\_regression, k=10)  
classifier = ExtraTreesClassifier(n\_estimators=60, max\_depth=4)  
processor\_pipeline = Pipeline([('selector', k\_best\_selector), ('erf', classifier)])  
processor\_pipeline.set\_params(selector\_\_k=7, erf\_\_n\_estimators=30)  
processor\_pipeline.fit(X, Y)  
print("Predicted output:", processor\_pipeline.predict(X))  
print("Score:", processor\_pipeline.score(X, Y))  
status = processor\_pipeline.named\_steps['selector'].get\_support()  
selected = [i for i, x in enumerate(status) if x]  
print("Selected features:", selected)



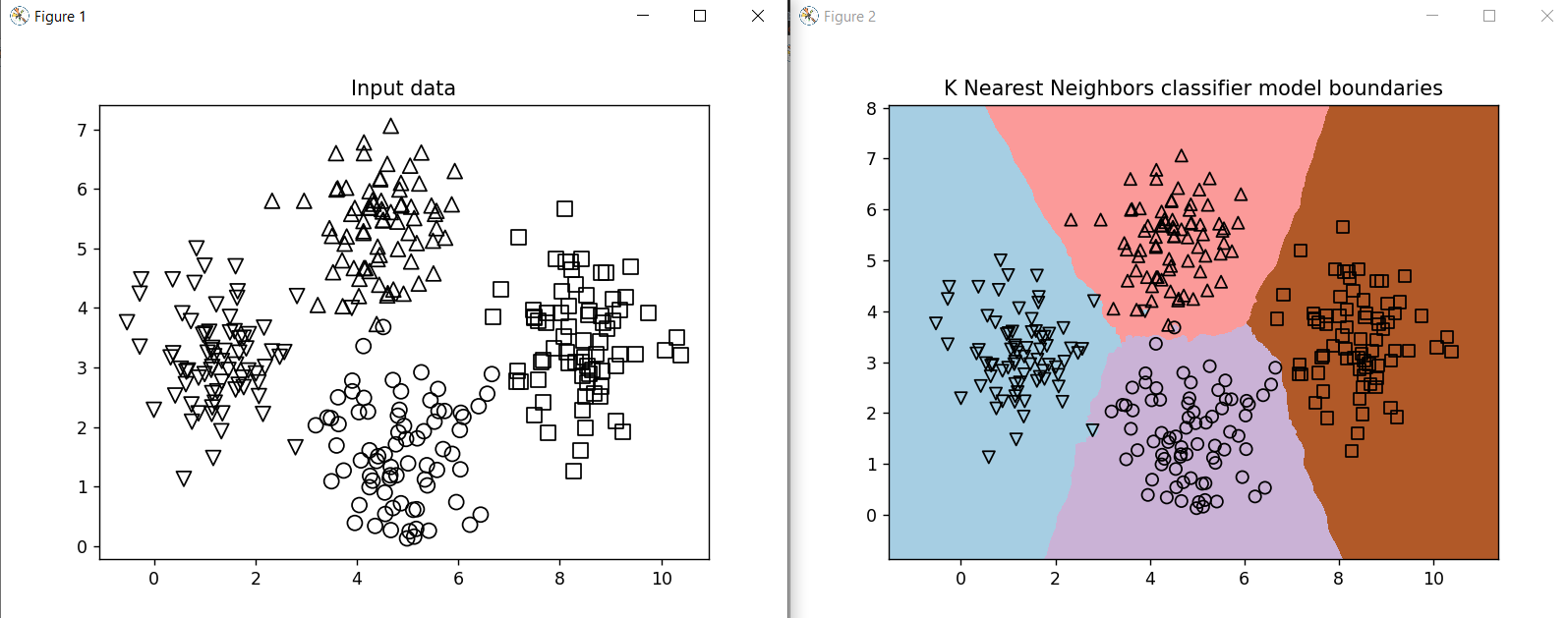
Завдання 2.7. Пошук найближчих сусідів

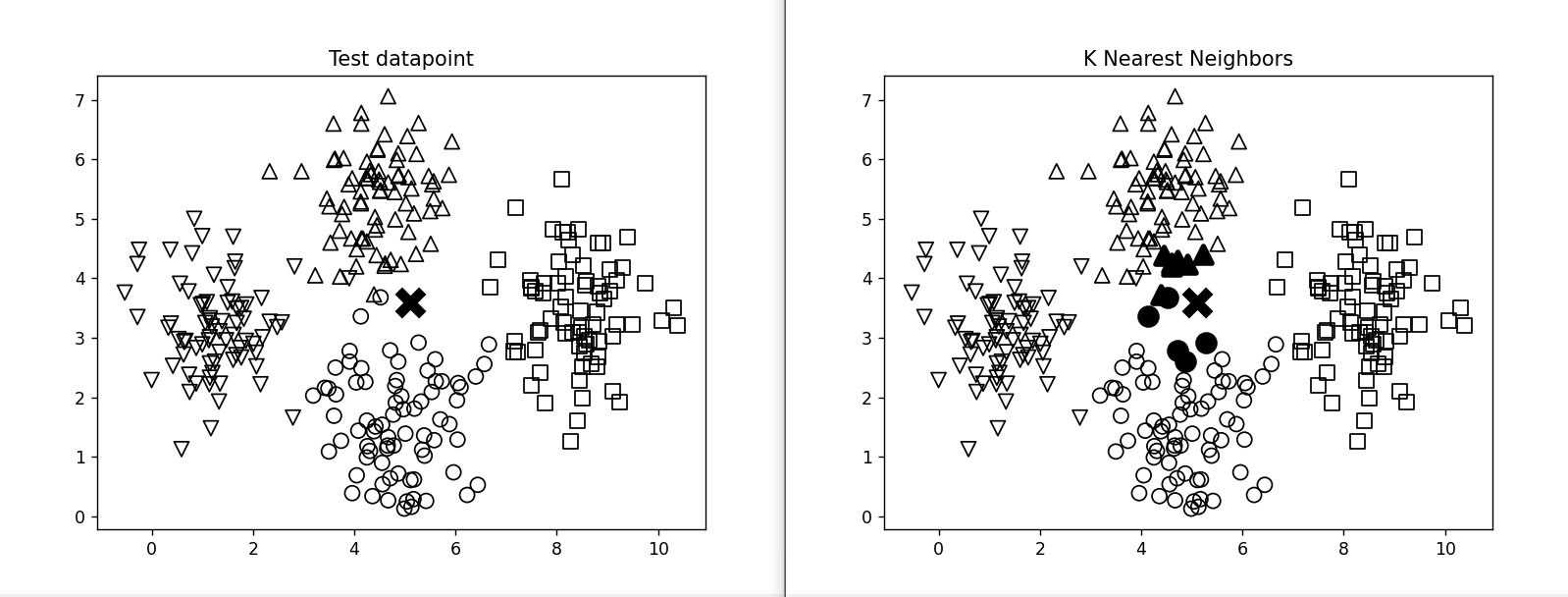
import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.neighbors import NearestNeighbors  
X = np.array([[2.1, 1.3], [1.3, 3.2], [2.9, 2.5], [2.7, 5.4], [3.8, 0.9],  
 [7.3, 2.1], [4.2, 6.5], [3.8, 3.7], [2.5, 4.1], [3.4, 1.9],  
 [5.7, 3.5], [6.1, 4.3], [5.1, 2.2], [6.2, 1.1]])  
k = 5  
test\_datapoint = [4.3, 2.7]  
plt.figure()  
plt.title('Input data')  
plt.scatter(X[:,0], X[:,1], marker='o', s=75, color='black')  
knn\_model = NearestNeighbors(n\_neighbors=k, algorithm='ball\_tree').fit(X)  
distances, indices = knn\_model.kneighbors([test\_datapoint])  
print("\nK Nearest Neighbors:")  
for rank, index in enumerate(indices[0][:k], start=1):  
 print(str(rank) + " ==>", X[index])  
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(test\_datapoint[0], test\_datapoint[1],  
 marker='x', s=75, color='k')  
plt.show()



Завдання 2.8. Створити класифікатор методом *k* найближчих сусідів

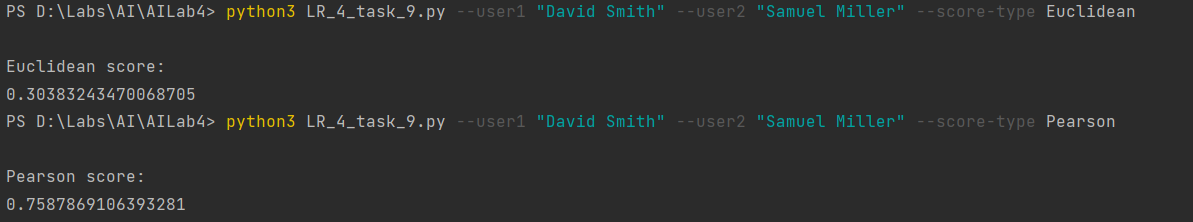
import numpy as np  
import matplotlib.pyplot as plt  
import matplotlib.cm as cm  
from sklearn import neighbors  
input\_file = 'data.txt'  
data = np.loadtxt(input\_file, delimiter=',')  
X, y = data[:, :-1], data[:, -1].astype(int)  
plt.figure()  
plt.title('Input data')  
marker\_shapes = 'v^os'  
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')  
num\_neighbors = 12  
step\_size = 0.01  
classifier = neighbors.KNeighborsClassifier(num\_neighbors, weights='distance')  
classifier.fit(X, y)  
x\_min, x\_max = X[:, 0].min() - 1, X[:, 0].max() + 1  
y\_min, y\_max = X[:, 1].min() - 1, X[:, 1].max() + 1  
x\_values, y\_values = np.meshgrid(np.arange(x\_min, x\_max, step\_size),  
 np.arange(y\_min, y\_max, step\_size))  
  
output = classifier.predict(np.c\_[x\_values.ravel(), y\_values.ravel()])  
output = output.reshape(x\_values.shape)  
plt.figure()  
plt.pcolormesh(x\_values, y\_values, output, cmap=cm.Paired)  
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\_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')  
\_, indices = classifier.kneighbors([test\_datapoint])  
indices = indices.astype(int)[0]  
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()





Завдання 2.9. Обчислення оцінок подібності

import argparse  
import json  
import numpy as np  
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  
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')  
 common\_movies = {}  
 for item in dataset[user1]:  
 if item in dataset[user2]:  
 common\_movies[item] = 1  
 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)))  
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')  
 common\_movies = {}  
 for item in dataset[user1]:  
 if item in dataset[user2]:  
 common\_movies[item] = 1  
 num\_ratings = len(common\_movies)  
 if num\_ratings == 0:  
 return 0  
 user1\_sum = np.sum([dataset[user1][item] for item in common\_movies])  
 user2\_sum = np.sum([dataset[user2][item] for item in 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])  
 sum\_of\_products = np.sum([dataset[user1][item] \* dataset[user2][item] for item in common\_movies])  
 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 = '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))



***Висновки:*** в ході виконання лабораторної роботи було досліджено та отримано знання, уміння та навики, щодо особливостей використання спеціалізованих бібліотеки та мову програмування Python дослідити методи ансамблів у машинному навчанні та створити рекомендаційні системи.