**ЛАБОРАТОРНА РОБОТА № 4**

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

***Мета:*** 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  
  
  
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")

Результат виконання програми:

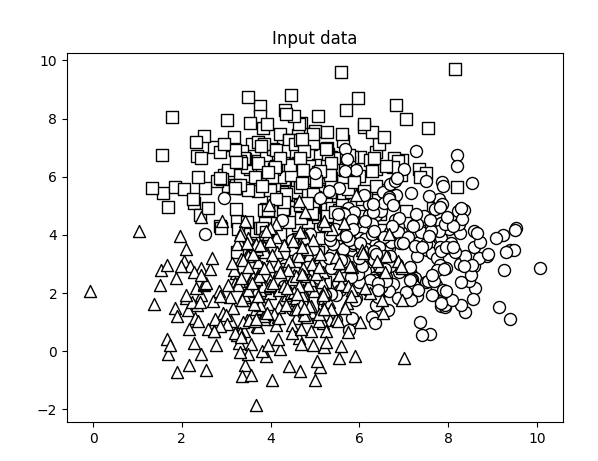


Рис. 4.1 Зображення розподілення даних

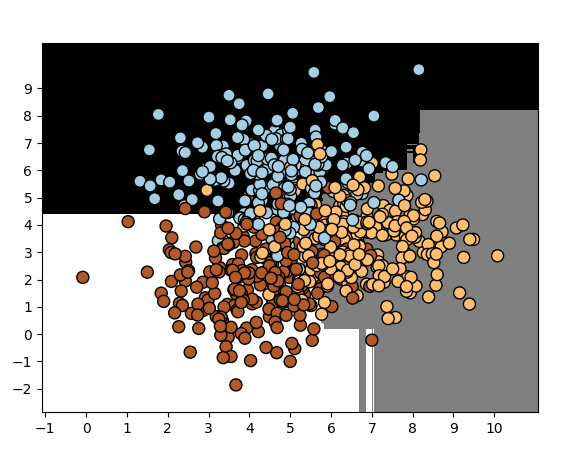


Рис. 4.2 Класифікація методом випадкових дерев

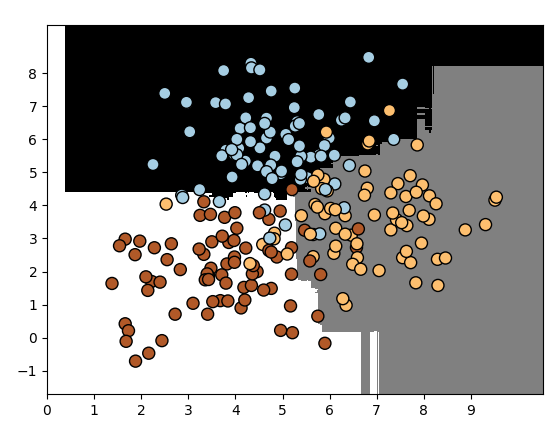


Рис. 4.3 Класифікація методом випадкових дерев

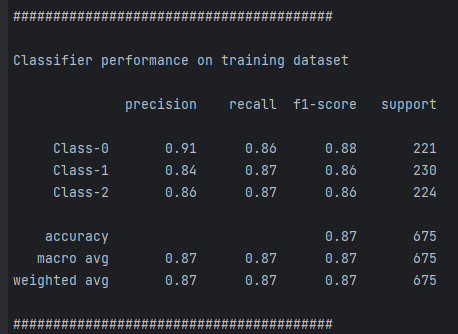


Рис. 4.4 Характеристики роботи методу випадкових дерев

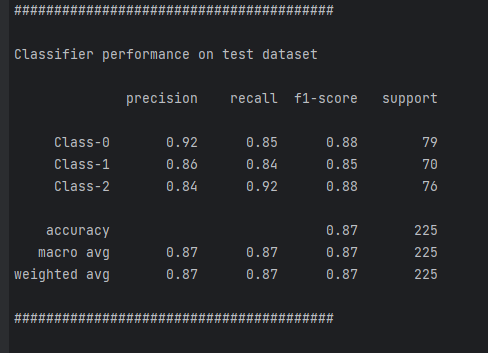


Рис. 4.5 Характеристики роботи методу гранично випадкових дерев

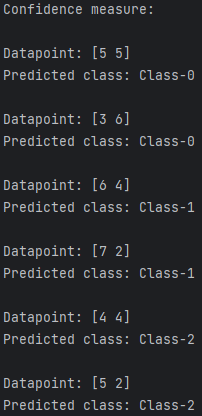


Рис. 4.6. Дані про можливі класи (rf)

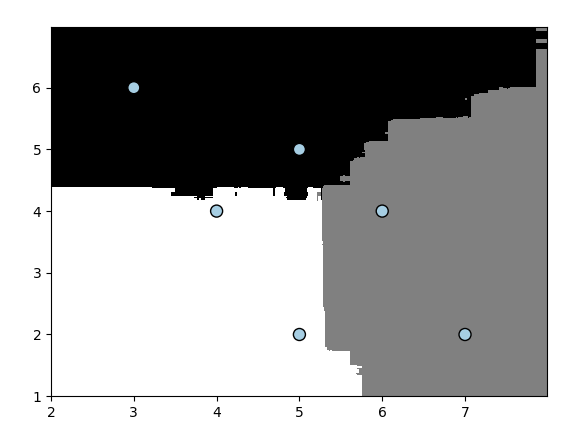


Рис. 4.7 Візуалізація можливих класів точок (rf).

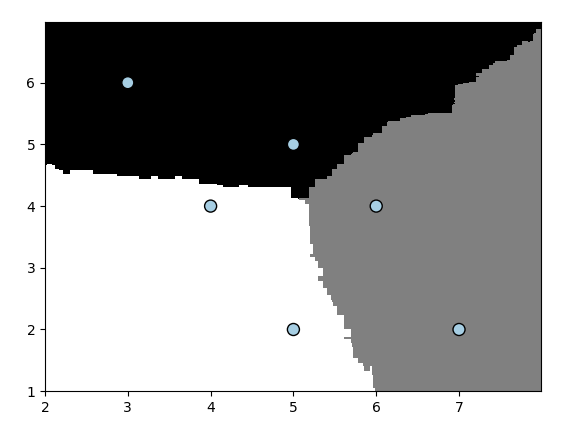


Рис 4.8 Візуалізація можливих класів точок (erf)

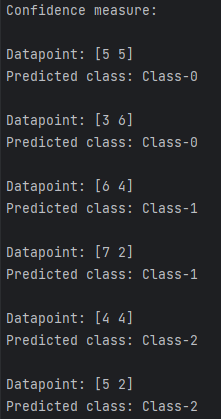


Рис 4.9 Дані про можливі класи (erf)

**Завдання 2.2.** Обробка дисбалансу класів

Лістинг програми:

import sys  
import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.model\_selection import cross\_val\_score, 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]  
  
 class\_0 = np.array(X[Y == 0])  
 class\_1 = np.array(X[Y == 1])  
  
 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()

Результат виконання програми:

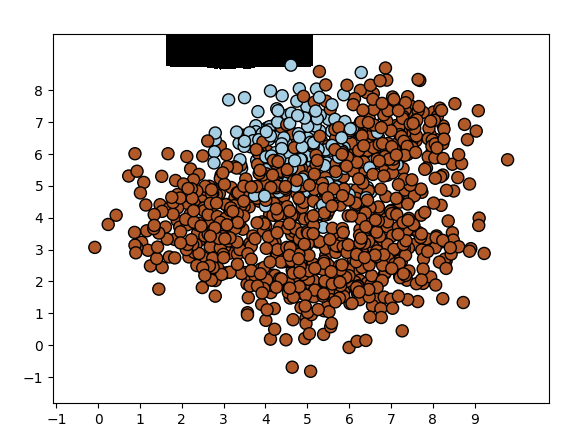


Рис 4.10 Розподілення незбалансованих даних

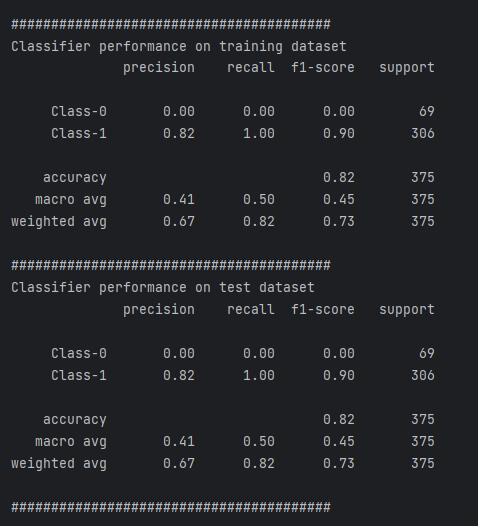


Рис 4.11 Характеристика незбалансованого класифікатора

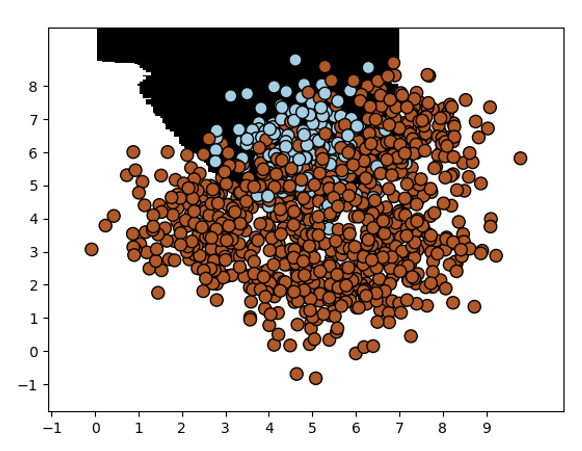


Рис 4.11 Збалансована класифікація

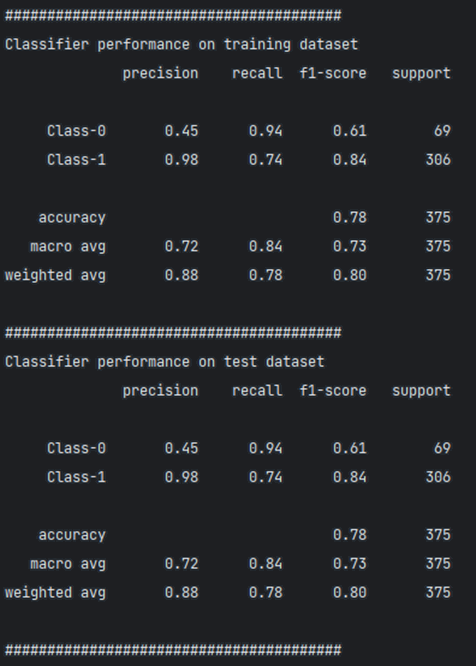


Рис 4.12 Характеристики збалансованої класифікації

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

Лістинг програми:

Результат виконання програми:

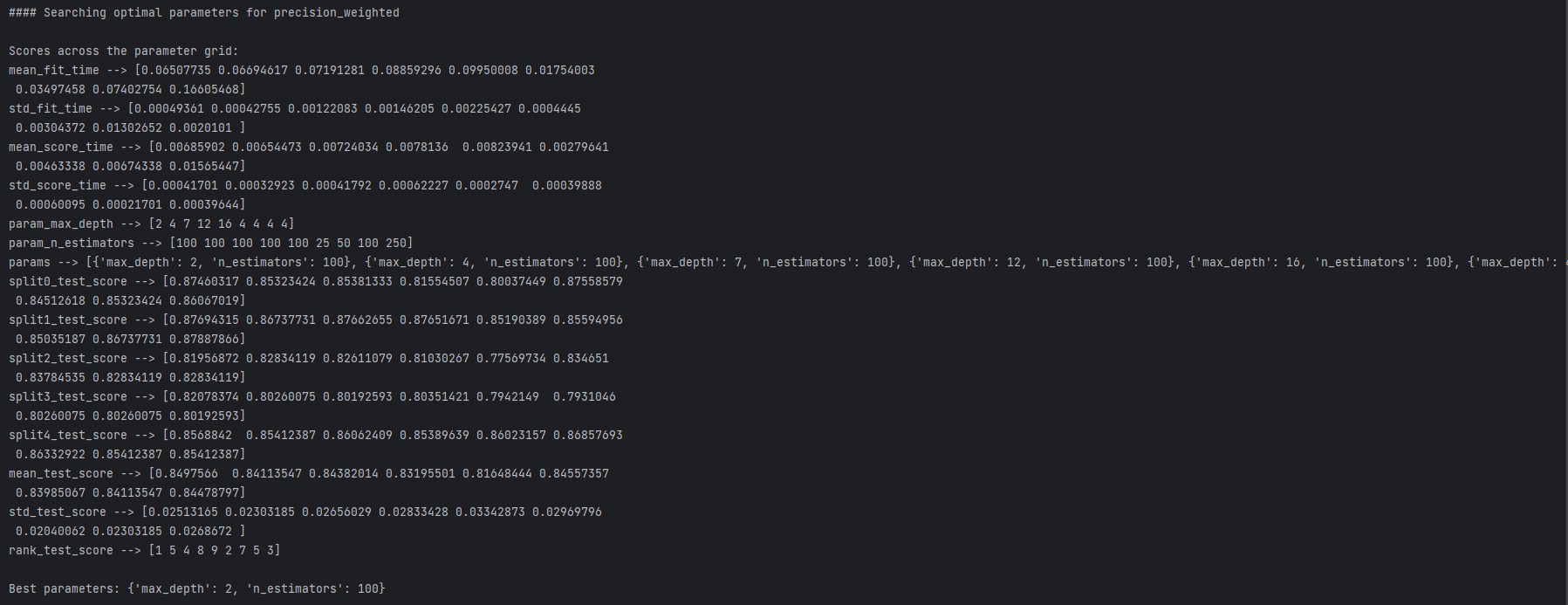


Рис 4.13 Результат виконання завдання

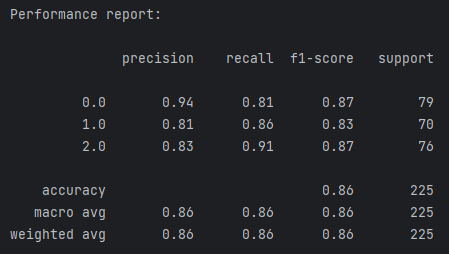


Рис 4.14 Характеристика класифікації зі сітковим пошуком

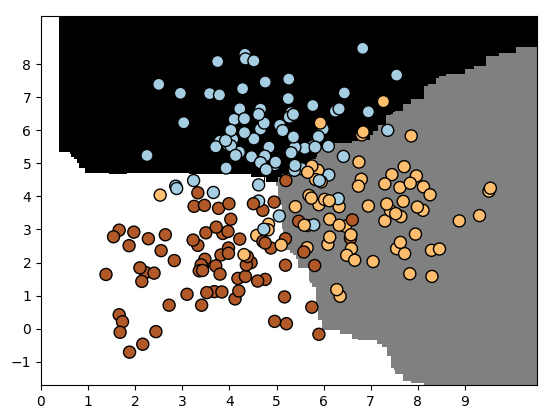


Рис 4.15 Класифікація даних зі сітковим пошуком

**Завдання 2.4.** Обчислення відносної важливості ознак

Лістинг програми:

import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.ensemble import AdaBoostRegressor  
from sklearn import datasets  
from sklearn.metrics import mean\_squared\_error, explained\_variance\_score  
from sklearn.model\_selection import train\_test\_split  
from sklearn.utils import shuffle  
  
housing\_data = datasets.load\_boston()  
X, Y = shuffle(housing\_data.data, housing\_data.target, random\_state=7)  
X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.2, random\_state=7)  
  
regressor = AdaBoostRegressor(DecisionTreeClassifier(max\_depth=4), n\_estimators=400, random\_state=7)  
regressor.fit(X\_train, Y\_train)  
  
Y\_train\_pred = regressor.predict(X\_train)  
mse = mean\_squared\_error(Y\_train, Y\_train\_pred)  
evs = explained\_variance\_score(Y\_train, Y\_train\_pred)  
print("ADABOOST REGRESSOR")  
print("Mean squared error =", round(mse, 2))  
print("Explained variance score =", round(evs, 2))  
  
feature\_importance = regressor.feature\_importances\_  
feature\_names = housing\_data.feature\_names  
  
feature\_importance = 100.0 \* (feature\_importance / max(feature\_importance))  
  
index\_sorted = np.flipud(np.argsort(feature\_importance))  
pos = np.arange(index\_sorted.shape[0]) + .5  
  
plt.figure()  
plt.bar(pos, feature\_importance[index\_sorted], align='center')  
plt.xticks(pos, feature\_names[index\_sorted])  
plt.ylabel('Relative Importance')  
plt.title('Variable Importance')  
plt.show()

Результат виконання завдання:

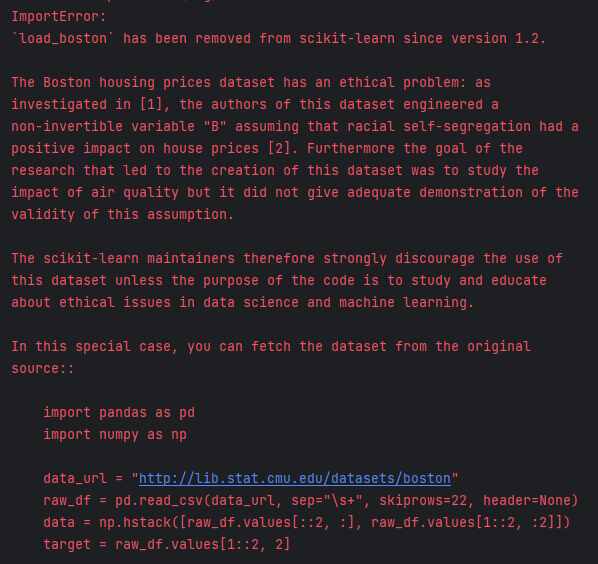


Рис 4.16 Спроба виконання завдання

При виконання завдання, було помічено що дані є застарілими та немає до їх доступу.

**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.metrics import classification\_report, mean\_absolute\_error  
from sklearn.model\_selection import train\_test\_split, cross\_val\_score  
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]))

Результат виконання завдання:

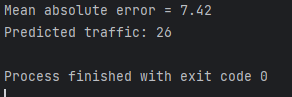


Рис 4.17 Результат регресії на основі гранично випадкових лісів

**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  
  
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)  
  
pipeline = Pipeline([('selector', k\_best\_selector), ('erf', classifier)])  
  
pipeline.set\_params(selector\_\_k=7, erf\_\_n\_estimators=30)  
  
pipeline.fit(X, Y)  
  
output = pipeline.predict(X)  
print("Predicted output:", output)  
  
print("Score:", pipeline.score(X, Y))  
  
status = pipeline.named\_steps['selector'].get\_support()  
selected = [i for i, x in enumerate(status) if x]  
print("\nIndices of selected features:", ', '.join([str(x) for x in selected]))

Результат виконання завдання:

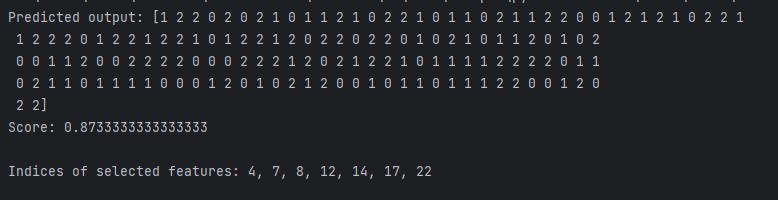


Рис 4.18 Отримані результати навчального конвеєра

**Завдання 2.7.** Пошук найближчих сусідів

Лістинг програми:

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 = np.array([[4.3, 2.7]])  
  
plt.figure()  
plt.title("Input data")  
plt.scatter(X[:, 0], X[:, 1], marker='o', s=75, color='k')  
  
knn\_model = NearestNeighbors(n\_neighbors=k, algorithm='ball\_tree').fit(X)  
distances, indices = knn\_model.kneighbors(test\_datapoint)  
  
print("K Nearest Neighbors:")  
for rank, index in enumerate(indices[0][:k], start=1):  
 print(str(rank) + ":", X[index])  
  
plt.figure()  
plt.title("K Nearest Neighbors")  
plt.scatter(X[:, 0], X[:, 1], marker='o', s=75, color='k')  
plt.scatter(test\_datapoint[:, 0], test\_datapoint[:, 1], marker='o', s=75, color='red')  
plt.scatter(X[indices][0][:][:, 0], X[indices][0][:][:, 1], marker='o', s=250, color='k', facecolors='none')  
plt.show()

Результат виконання завдання:

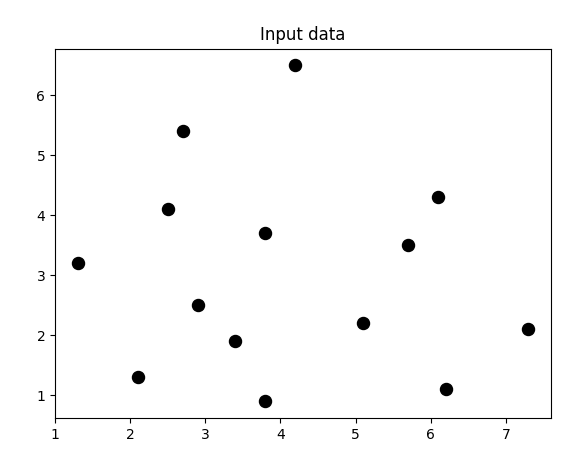


Рис 4.19 Вхідні данні

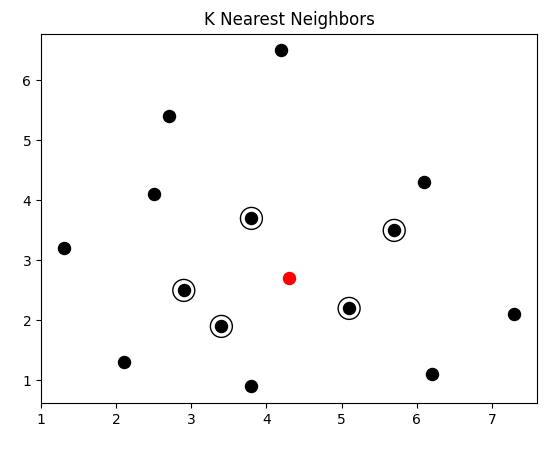


Рис 4.20 Пошук найближчих сусідів

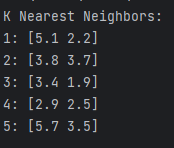


Рис 4.21 Дані про найближчих сусідів

**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, datasets  
  
input\_file = 'data.txt'  
data = np.loadtxt(input\_file, delimiter=',')  
X, Y = data[:, :-1], data[:, -1]  
  
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\_shapes[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\_mesh = classifier.predict(np.c\_[X\_values.ravel(), Y\_values.ravel()])  
output\_mesh = output\_mesh.reshape(X\_values.shape)  
  
plt.figure()  
plt.pcolormesh(X\_values, Y\_values, output\_mesh, cmap=cm.Paired)  
plt.scatter(X[:, 0], X[:, 1], c=Y, s=80, edgecolors='black', linewidth=1, cmap=cm.Paired)  
plt.xlim(X\_values.min(), X\_values.max())  
plt.ylim(Y\_values.min(), Y\_values.max())  
plt.title('K Nearest Neighbors classifier on input data')  
  
test\_datapoint = [5.1, 3.6]  
plt.scatter(test\_datapoint[0], test\_datapoint[1], marker='o', s=100, linewidths=3, color='black')  
  
\_, indices = classifier.kneighbors([test\_datapoint])  
indices = np.asarray(indices).flatten()  
plt.scatter(X[indices][:, 0], X[indices][:, 1], marker='\*', s=80, linewidths=1, color='black', facecolors='none')  
plt.show()  
  
print("Predicted output:", classifier.predict([test\_datapoint])[0])

Результат виконання завдання:

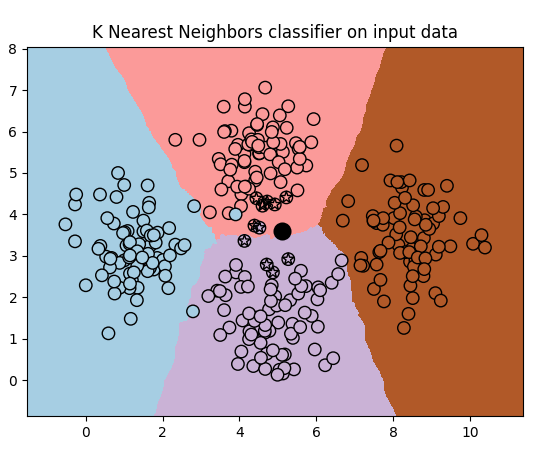


Рис 4.22 Класифікація методом К-найближчих сусідів та найближчі сусіди введеної точки



Рис 4.23 Результат виконання програми

**Завдання 2.9.** Обчислення оцінок подібності

Лістинг програми:

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 us-er")  
 parser.add\_argument('--score-type', dest='score\_type', required=True,  
 choices=['Euclidean', 'Pearson'], help="Similarity score to be computed")  
 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])  
  
 product\_sum = np.sum([dataset[user1][item] \* dataset[user2][item] for item in common\_movies])  
  
 Sxy = product\_sum - (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  
  
 with open('ratings.json', '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 завдання:

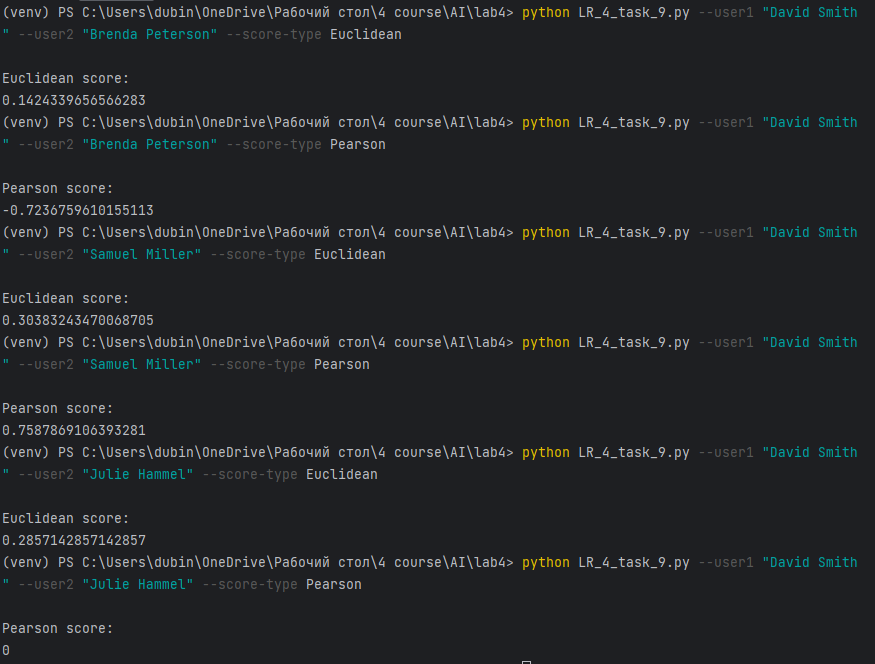


Рис 4.24 Обрахунок оцінок

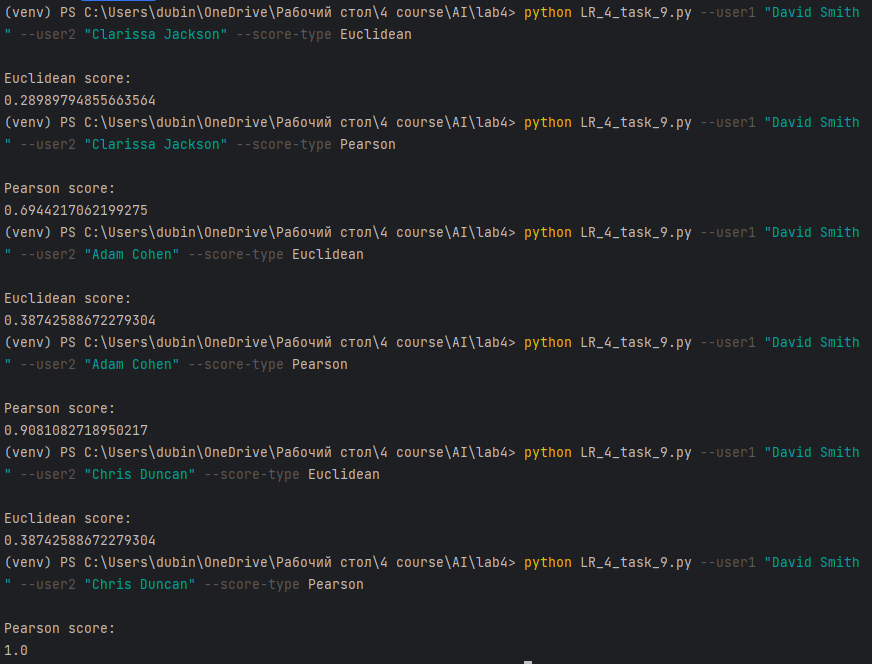


Рис 4.25 Обрахунок оцінок

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

Лістинг програми:

import argparse  
import json  
import numpy as np  
from LR\_4\_task\_9 import pearson\_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):  
 if user not in dataset:  
 raise TypeError('Cannot find ' + user + ' in the dataset')  
  
 scores = np.array([[x, pearson\_score(dataset, user, x)] for x in dataset if x != user])  
 scores\_sorted = np.argsort(scores[:, 1])[::-1]  
 top\_users = scores\_sorted[:num\_users]  
 return scores[top\_users]  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 args = build\_arg\_parser().parse\_args()  
 user = args.user  
  
 ratings\_file = 'ratings.json'  
 with open(ratings\_file, 'r') as f:  
 data = json.loads(f.read())  
  
 print("Users similar to " + user + ":")  
 similar\_users = find\_similar\_users(data, user, 3)  
 print('User\t\t\tSimilarity score')  
 print('-'\*41)  
 for item in similar\_users:  
 print(item[0], '\t\t', round(float(item[1]), 2))

Результат виконання завдання:

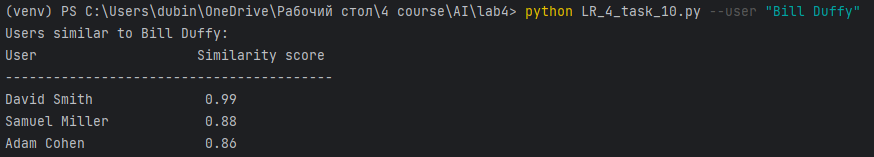


Рис 4.26 Знаходження найбільших оцінок

**Завдання 2.11.** Створення рекомендаційної системи фільмів

Лістинг програми:

import argparse  
import json  
import numpy as np  
from LR\_4\_task\_9 import pearson\_score  
from LR\_4\_task\_10 import find\_similar\_users  
  
  
def build\_arg\_parser():  
 parser = argparse.ArgumentParser(description='Find movies recommended for the input user')  
 parser.add\_argument('--user', dest='user', required=True, help='Input user')  
 return parser  
  
  
def get\_recommendations(dataset, input\_user):  
 if input\_user not in dataset:  
 raise TypeError('Cannot find ' + input\_user + ' in the dataset')  
  
 total\_scores = {}  
 similarity\_sums = {}  
 for user in [x for x in dataset if x != input\_user]:  
 similarity\_score = pearson\_score(dataset, input\_user, user)  
  
 if similarity\_score <= 0:  
 continue  
  
 filtered\_list = [movie for movie in dataset[user]  
 if movie not in dataset[input\_user] or dataset[input\_user][movie] == 0]  
  
 for movie in filtered\_list:  
 total\_scores.update({movie: dataset[user][movie] \* similarity\_score})  
 similarity\_sums.update({movie: similarity\_score})  
  
 if len(total\_scores) == 0:  
 return ['No recommendations possible']  
  
 movie\_ranks = np.array([[total/similarity\_sums[item], item] for item, total in total\_scores.items()])  
 movie\_ranks = movie\_ranks[np.argsort(movie\_ranks[:, 0])[::-1]]  
 recommended\_movies = [movie for \_, movie in movie\_ranks]  
  
 return recommended\_movies[:10]  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 args = build\_arg\_parser().parse\_args()  
 user = args.user  
  
 ratings\_file = 'ratings.json'  
 with open(ratings\_file, 'r') as f:  
 data = json.loads(f.read())  
  
 print("Movies recommended for " + user + ":")  
 movies = get\_recommendations(data, user)  
 for i, movie in enumerate(movies):  
 print(str(i+1) + '. ' + movie)

Результат виконання завдання:

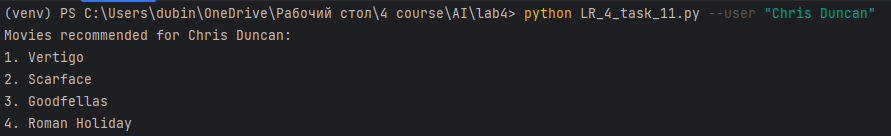


Рис 4.27 Знаходження найбільших оцінок

Посилання на GitHub: <https://github.com/BogdanStelmah/Basics-of-AI_labs>

**Висновок:** На даній лабораторній роботі мивикористовуючи спеціалізовані бібліотеки та мову програмування Python дослідили методи ансамблів у машинному навчанні та створили рекомендаційні системи.