**ЛАБОРАТОРНА РОБОТА № 4**

**ДОСЛІДЖЕННЯ МЕТОДІВ АНСАМБЛЕВОГО НАВЧАННЯ**

**ТА СТВОРЕННЯ РЕКОМЕНДАЦІЙНИХ СИСТЕМ**

<https://github.com/MaximVengel/AI>

**Завдання 1.** Створення класифікаторів на основі випадкових та гранично випадкових лісів.

LR\_4\_task\_1.py

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  
 # Завантаження вхідних даних  
 input\_file = 'data\_random\_forests.txt'  
 data = np.loadtxt(input\_file, delimiter=',')  
 X, Y = data[:, :-1], data[:, -1]  
 print(X)  
 # Розбиття вхідних даних на три класи  
 class\_0 = np.array(X[Y == 0])  
 class\_1 = np.array(X[Y == 1])  
 class\_2 = np.array(X[Y == 2])  
 # Візуалізація вхідних даних  
 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()  
# Розбивка даних на навчальний та тестовий набори  
X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.25, random\_state=5)  
# Класифікатор на основі ансамблевого навчання  
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')  
  
# Перевірка роботи класифікатора  
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:

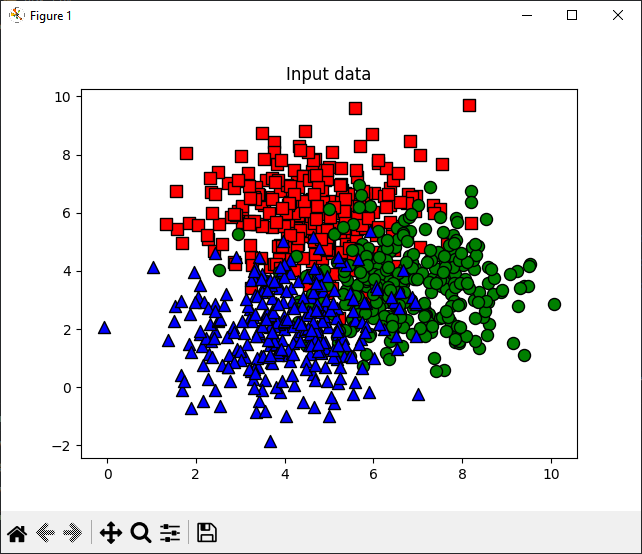


Рис. 1. Зображення розподілення даних.

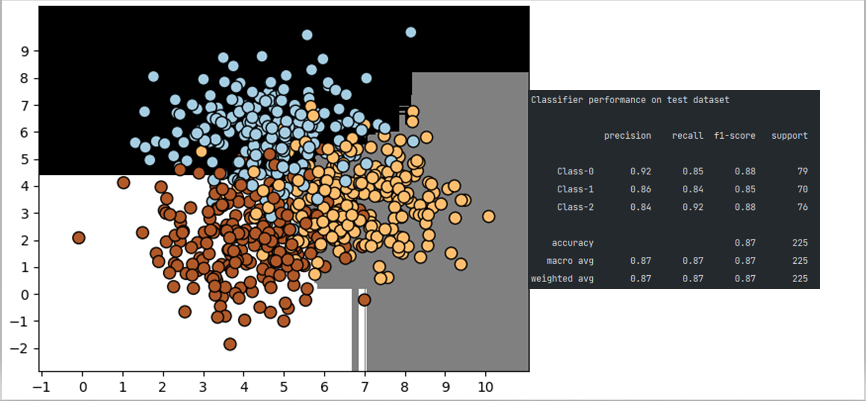


Рис. 2. Класифікація методом випадкових дерев + характеристики роботи методу випадкових дерев.

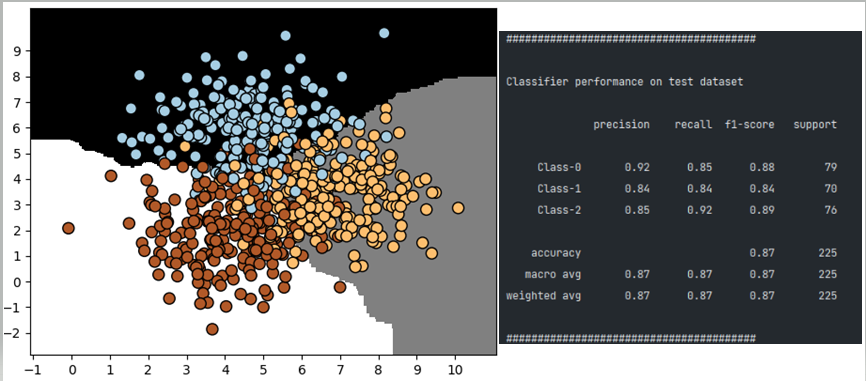


Рис. 3. Класифікація методом гранично випадкових дерев + характеристики роботи методу гранично випадкових дерев.

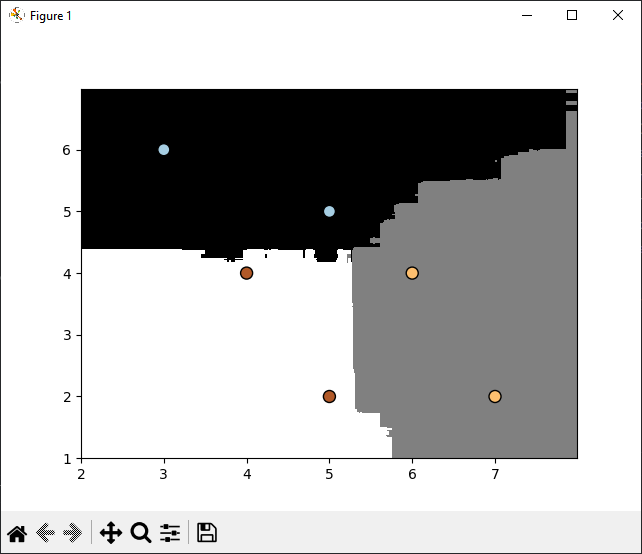


Рис. 4. Візуалізація можливих класів точок (rf)

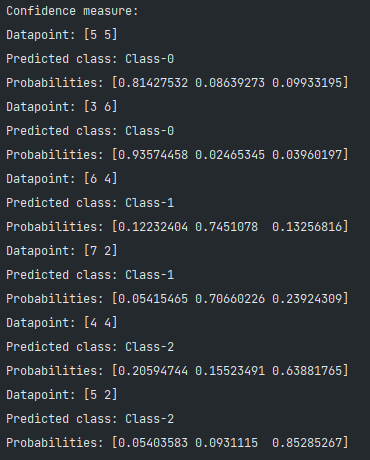


Рис. 5. Дані про можливі класи (rf)

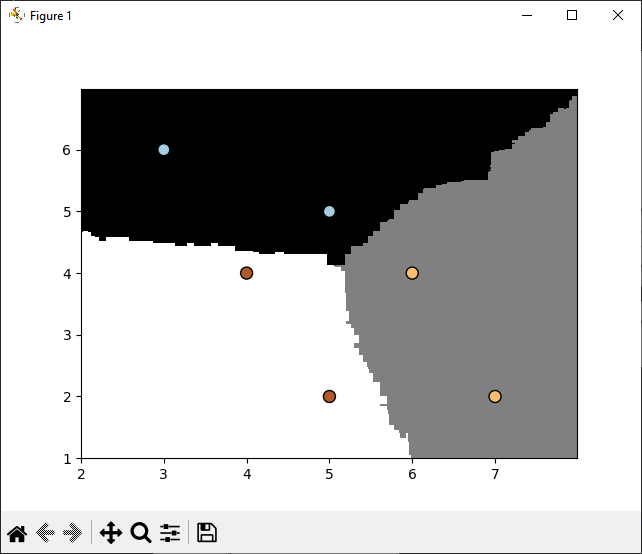


Рис. 6. Візуалізація можливих класів точок (erf)

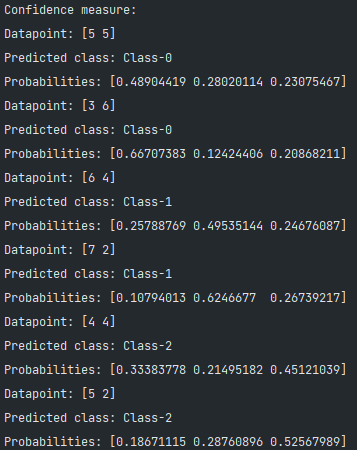


Рис. 7. Дані про можливі класи (erf)

Висновок по завданню: Використання випадкових дерев та граничних випадкових дерев дозволяє ефективно класифікувати дані. Як показала практика, з двох методів – останній (граничних випадкових дерев) показує більшу ефективність.

**Завдання 2.** Обробка дисбалансу класів.

LR\_4\_task\_2.py

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\_\_':  
 # Завантаження вхідних даних  
 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')  
 # Розбиття даних на навчальний та тестовий набори  
 X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.25, random\_state=5)  
 # Класифікатор на основі гранично випадкових лісів  
 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)  
 # Обчислення показників ефективності класифікатора  
 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:

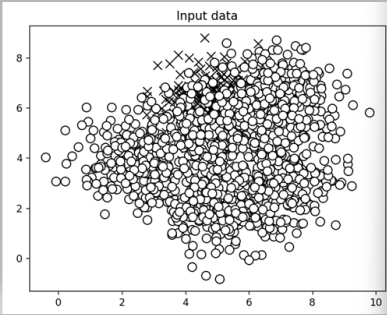


Рис. 8. Розподілення незбалансованих даних.

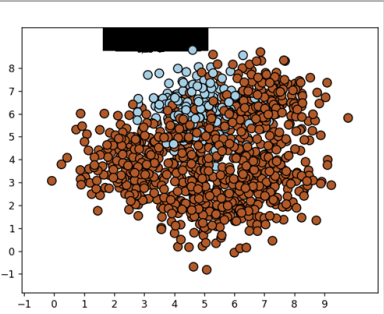


Рис. 9. Розподілення незбалансованих даних.

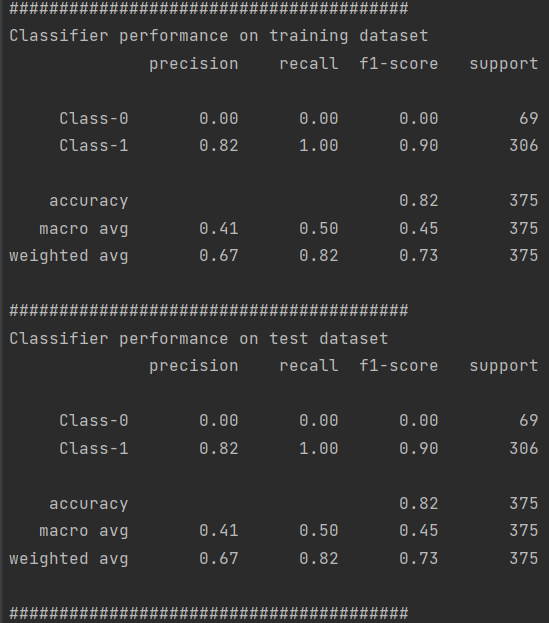


Рис. 10. Характеристика незбалансованої класифікації.

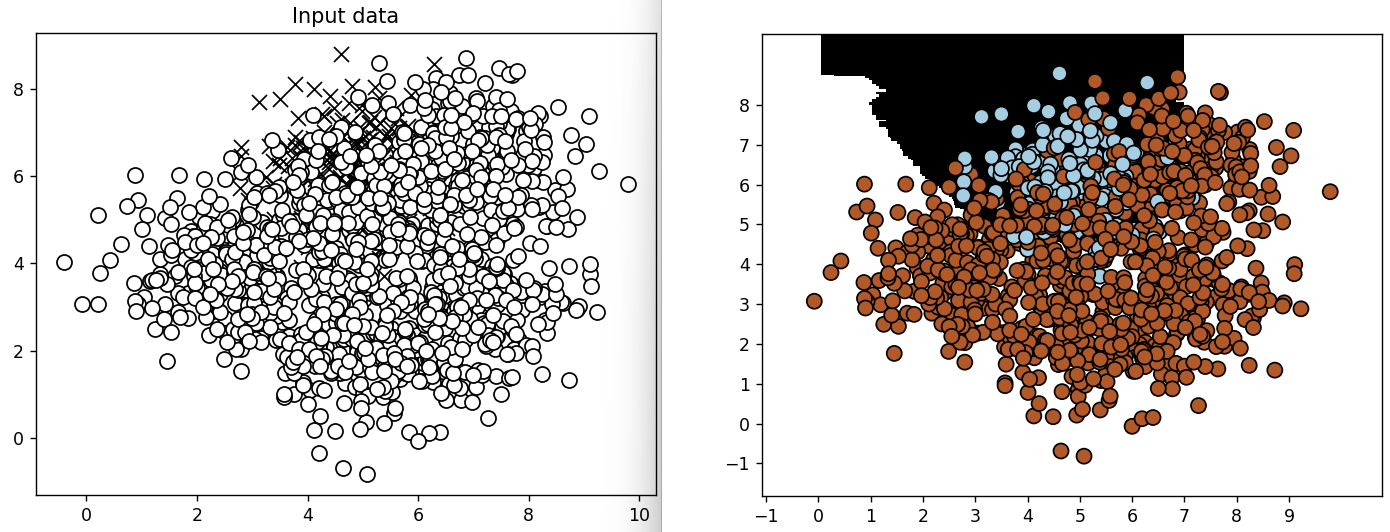


Рис. 11. Розподілення збалансованих даних.

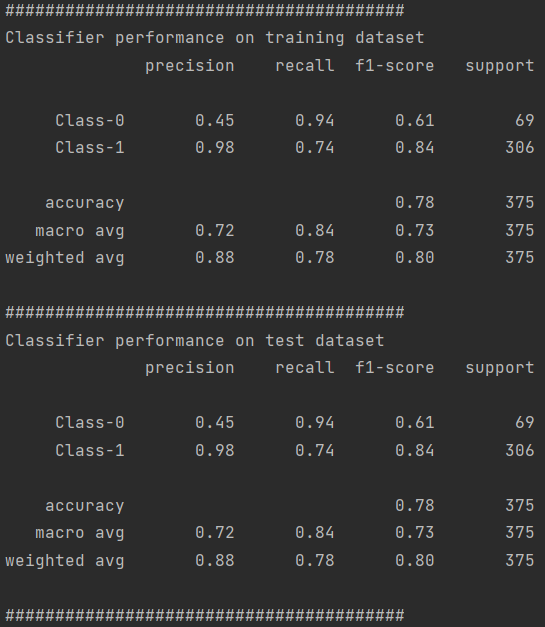


Рис. 12. Характеристики збалансованої класифікації.

Висновок по завданню:

Завдяки балансуванню даних було отримано коректно та ефективно класифіковано дані.

**Завдання 3.** Знаходження оптимальних навчальних параметрів за допомогою сіткового пошуку.

LR\_4\_task\_3.py

import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.model\_selection import cross\_val\_score, 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]  
# Розбиття даних на три класи на підставі міток  
class\_0 = np.array(X[Y == 0])  
class\_1 = np.array(X[Y == 1])  
class\_2 = np.array(X[Y == 2])  
# Розбиття даних на навчальний та тестовий набори  
X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.25, random\_state=5)  
# Визначення сітки значень параметрів  
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)

Результат виконання:

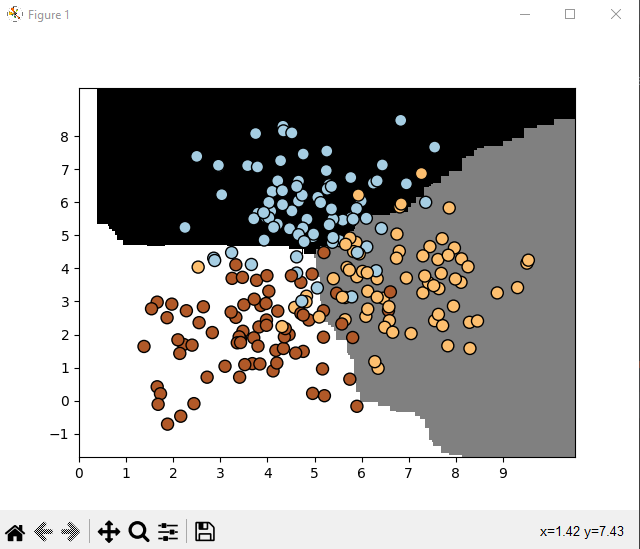


Рис. 13. Візуалізація класифікації даних зі сітковим пошуком.

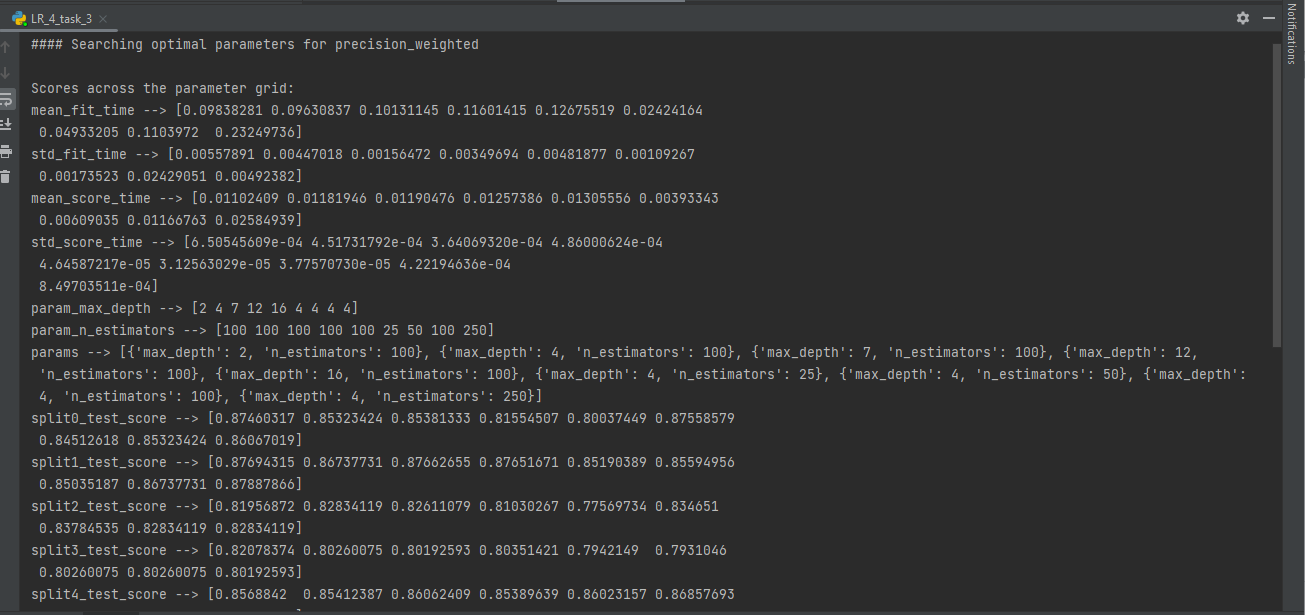


Рис. 14. Отримання даних процесу класифікації.

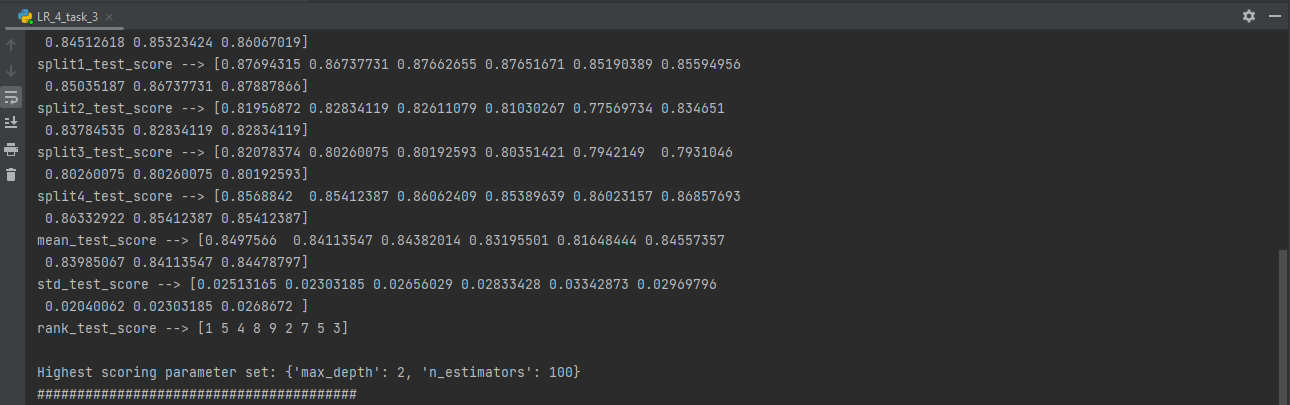


Рис. 15. Отримання даних процесу класифікації.

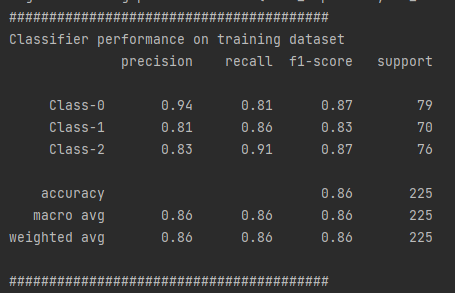


Рис. 16. Характеристика класифікації зі сітковим пошуком.

**Завдання 4.** Обчислення відносної важливості ознак.

Завдання неможливо виконати, адже відсутні дані.

**Завдання 5.** Прогнозування інтенсивності дорожнього руху за допомогою класифікатора на основі гранично випадкових лісів.

LR\_4\_task\_5.py

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:

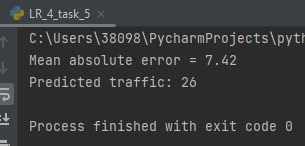
****

Рис. 17. Результат регресії на основі гранично випадкових лісів.

Висновок до завдання:

Отримано значення 26, яке є дуже близьким до фактичного значення.

**Завдання 6.** Створення навчального конвеєра (конвеєра машинного навчання)

LR\_4\_task\_6.py

from sklearn.datasets import \_samples\_generator  
from sklearn.feature\_selection import SelectKBest, f\_regression  
from sklearn.pipeline import Pipeline  
from sklearn.ensemble import ExtraTreesClassifier  
# Генерування даних  
X, Y = \_samples\_generator.make\_classification(n\_samples=150, n\_features=25, n\_classes=3,  
 n\_informative=6, n\_redundant=0, random\_state=7)  
# Вибір k найважливіших ознак  
k\_best\_selector = SelectKBest(f\_regression, k=10)  
# Ініціалізація класифікатора на основі гранично випадкового лісу  
classifier = ExtraTreesClassifier(n\_estimators=60, max\_depth=4)  
# Створення конвеєра  
processor\_pipeline = Pipeline([('selector', k\_best\_selector), ('erf', classifier)])  
# Встановлення параметрів  
processor\_pipeline.set\_params(selector\_\_k=7, erf\_\_n\_estimators=30)  
# Навчання конвеєра  
processor\_pipeline.fit(X, Y)  
# Прогнозування результатів для вхідних даних  
print("Predicted output:", processor\_pipeline.predict(X))  
# Виведення оцінки  
print("Score:", processor\_pipeline.score(X, Y))  
# Виведення ознак, відібраних селектором конвеєра  
status = processor\_pipeline.named\_steps['selector'].get\_support()  
# Вилучення та виведення індексів обраних ознак  
selected = [i for i, x in enumerate(status) if x]  
print("Selected features:", selected)

Результат виконання:

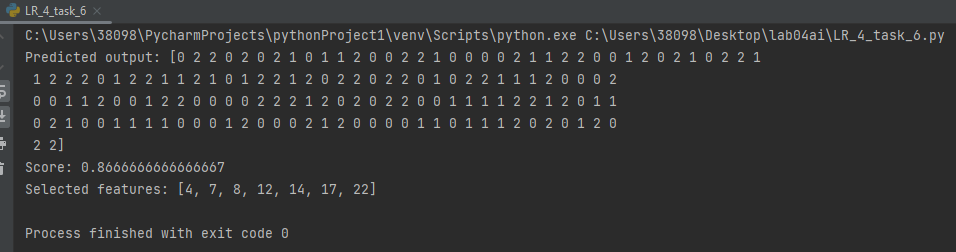


Рис. 18. Отримані результати навчального конвеєра.

Висновок до завдання:

Обрали найбільш важливі ознаки з вхідних даних.

З**авдання 7.** Пошук найближчих сусідів.

LR\_4\_task\_7.py

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]])  
  
# Number of nearest neighbors  
k = 5  
  
# Test datapoint  
test\_datapoint = [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([test\_datapoint])  
  
# 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 test datapoint  
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()

Результат виконання:

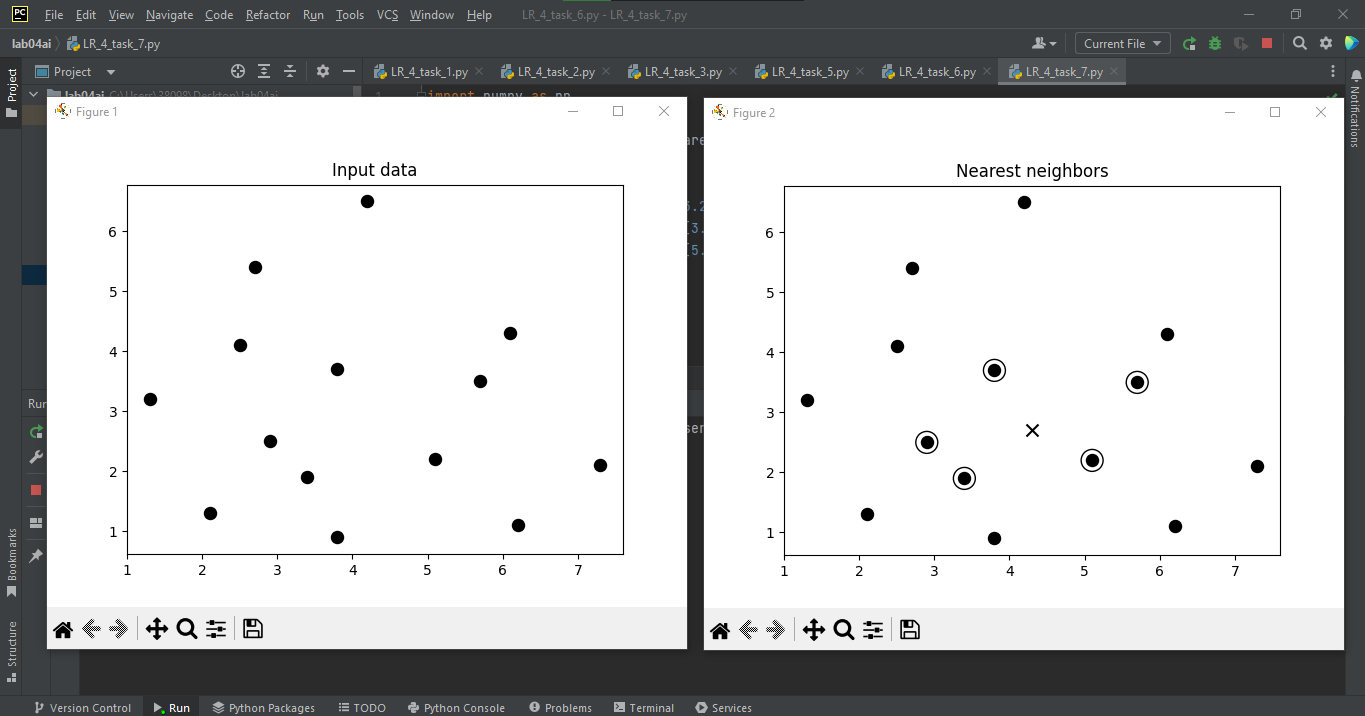


Рис. 19. Пошук найближчих сусідів.

На графіку зліва – вхідні дані.

Найближчі сусіди зображені на графіку зправа, координати в терміналі:

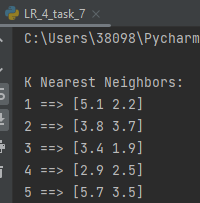


Рис. 20. Дані про найближчих сусідів.

**Завдання 8.** Створити класифікатор методом k найближчих сусідів.

LR\_4\_task\_8.py

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]  
  
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:



Рис. 21. Класифікація методом К-найближчих сусідів та найближчі сусіди введеної точки.

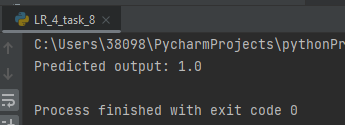


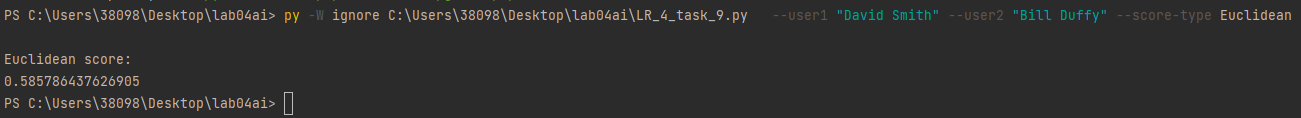
Рис. 22. Обрахований клас точки.

**Завдання 9.** Обчислення оцінок подібності.

LR\_4\_task\_9.py

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  
  
  
# 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  
  
 num\_ratings = len(common\_movies)  
  
 # If there are no common movies between user1 and user2, then the score is 0  
 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 = '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:



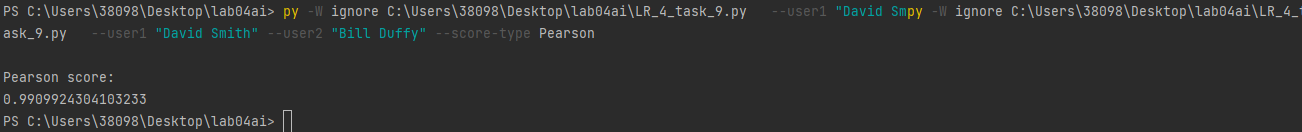


Рис. 23 – 24. Обрахунок оцінок для David Smith та Bill Duffy.

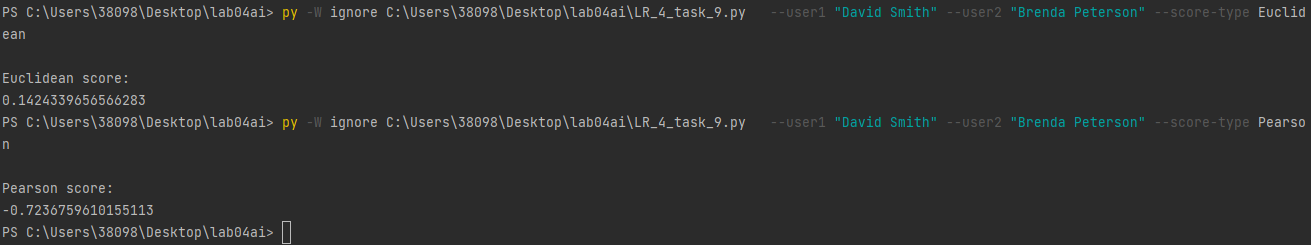


Рис. 25. Обрахунок оцінок для David Smith та Brenda Peterson.

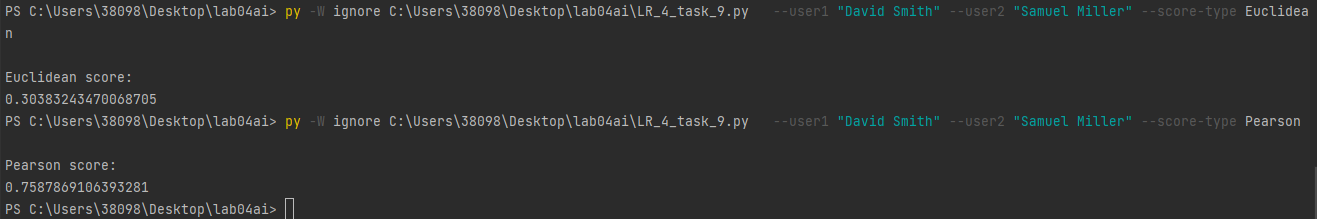


Рис. 26. Обрахунок оцінок для David Smith та Samuel Miller.

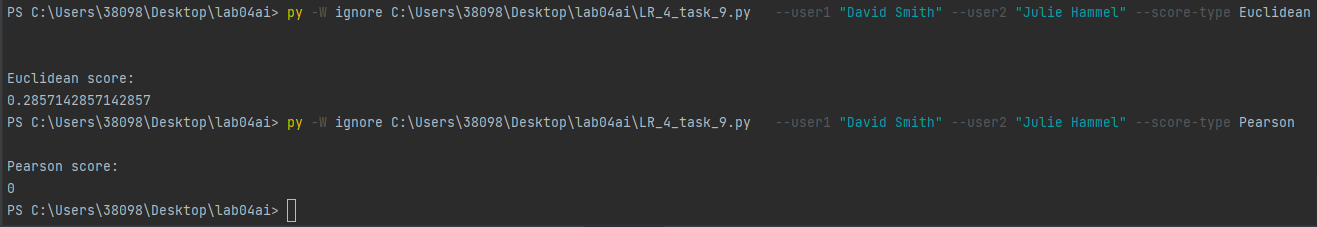


Рис. 27. Обрахунок оцінок для David Smith та Julie Hammel.

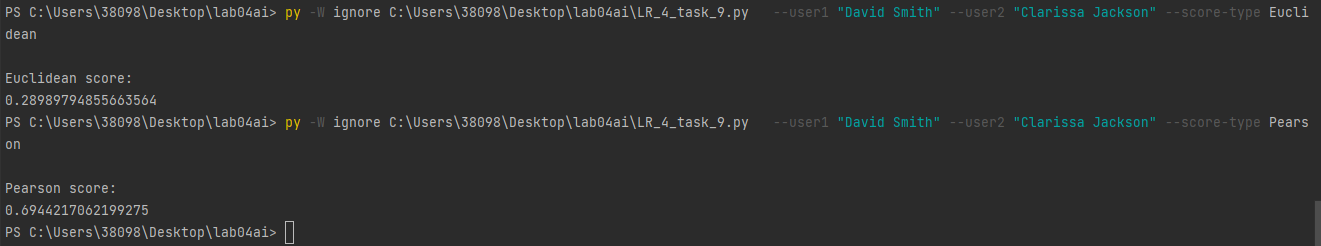


Рис. 28. Обрахунок оцінок для David Smith та Clarissa Jackson.

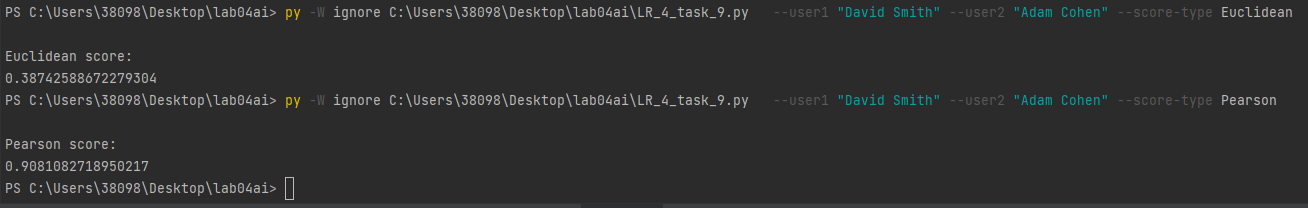


Рис. 29. Обрахунок оцінок для David Smith та Adam Cohen.

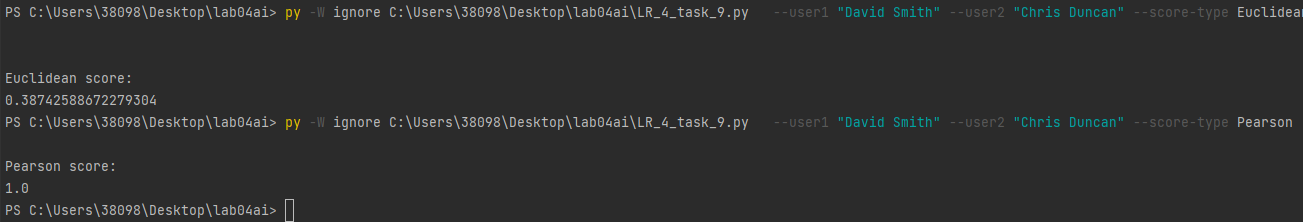


Рис. 30. Обрахунок оцінок для David Smith та Chris Duncan.

**Завдання 10.** Пошук користувачів зі схожими уподобаннями методом колаборативної фільтрації.

LR\_4\_task\_10.py

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  
  
# Finds users in the dataset that are similar to the input user  
def find\_similar\_users(dataset, user, num\_users):  
 if user not in dataset:  
 raise TypeError('Cannot find ' + user + ' in the dataset')  
  
 # Compute Pearson score between input user  
 # and all the users in the dataset  
 scores = np.array([[x, pearson\_score(dataset, user,  
 x)] for x in dataset if x != user])  
  
 # Sort the scores in decreasing order  
 scores\_sorted = np.argsort(scores[:, 1])[::-1]  
  
 # Extract the top 'num\_users' scores  
 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('\nUsers similar to ' + user + ':\n')  
 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:

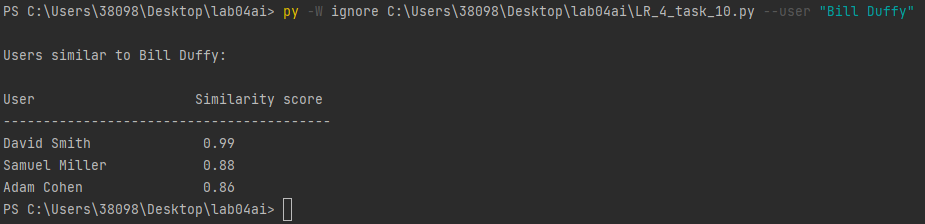


Рис. 31. Знаходження користувачів схожих на Bill Duffy.

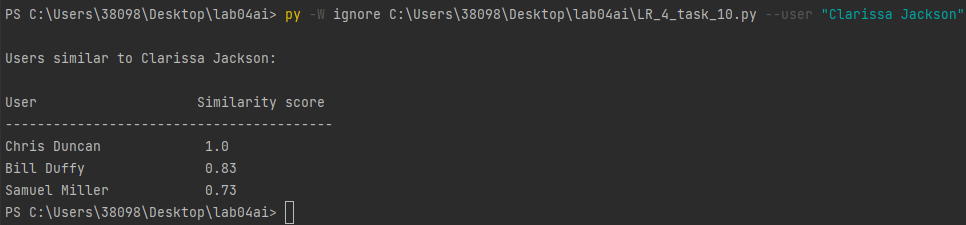


Рис. 32. Знаходження користувачів схожих на Clarissa Jackson.

**Завдання 11.** Створення рекомендаційної системи фільмів.

LR\_4\_task\_11.py

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 the movie recommendations for the given user')  
 parser.add\_argument('--user', dest='user', required=True,  
 help='Input user')  
 return parser  
  
  
# Get movie recommendations for the input user  
def get\_recommendations(dataset, input\_user):  
 if input\_user not in dataset:  
 raise TypeError('Cannot find ' + input\_user + ' in the dataset')  
  
 overall\_scores = {}  
 similarity\_scores = {}  
  
 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 = [x for x in dataset[user] if x not in \  
 dataset[input\_user] or dataset[input\_user][x] == 0]  
  
 for item in filtered\_list:  
 overall\_scores.update({item: dataset[user][item] \* similarity\_score})  
 similarity\_scores.update({item: similarity\_score})  
  
 if len(overall\_scores) == 0:  
 return ['No recommendations possible']  
  
 # Generate movie ranks by normalization  
 movie\_scores = np.array([[score / similarity\_scores[item], item]  
 for item, score in overall\_scores.items()])  
  
 # Sort in decreasing order  
 movie\_scores = movie\_scores[np.argsort(movie\_scores[:, 0])[::-1]]  
  
 # Extract the movie recommendations  
 movie\_recommendations = [movie for \_, movie in movie\_scores]  
  
 return movie\_recommendations  
  
  
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("\nMovie recommendations for " + user + ":")  
 movies = get\_recommendations(data, user)  
 for i, movie in enumerate(movies):  
 print(str(i + 1) + '. ' + movie)

Результат виконання:

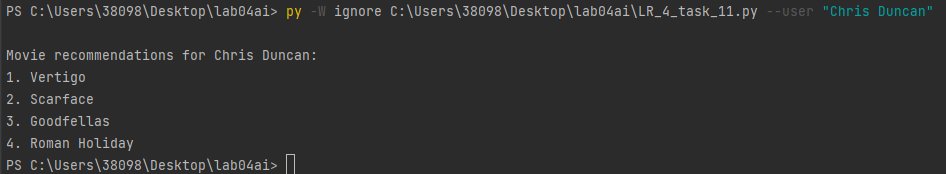


Рис. 33. Рекомендації для Chris Duncan.

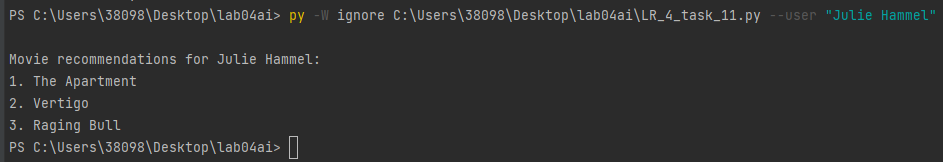


Рис. 34. Рекомендації для Julie Hammel.

**Висновок:** під час виконання завдань лабораторної роботи з використанням спеціалізованих бібліотек та мови програмування Python було досліджено методи ансамблів у машинному навчанні та створити рекомендаційні системи. Було використано можливості наступних бібліотек: matplotlib, argparse, sclearn.metrics, sklearn.model\_selection, sklearn.feature\_selection, sklearn.neighbors.json, sklearn.preprocessing, sklearn.ensemble.