**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.

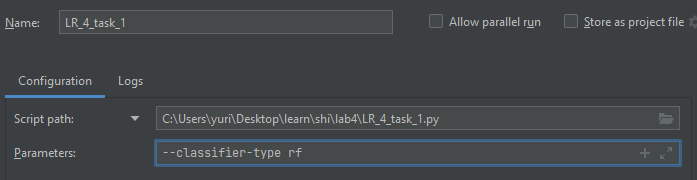
**Завдання №1:** Створення класифікаторів на основі випадкових та гранично випадкових лісів.

Код скрипту **LR\_4\_task\_1.py**:

import argparse  
import numpy as np  
import matplotlib.pyplot as plt  
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\_\_':  
 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]  
 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='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')

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)  
 y\_test\_pred = classifier.predict(X\_test)  
 visualize\_classifier(classifier, X\_test, y\_test)  
 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)  
 visualize\_classifier(classifier, test\_datapoints, [0] \* len(test\_datapoints))  
 plt.show()

Передача аргументів в середовищі PyCharm:



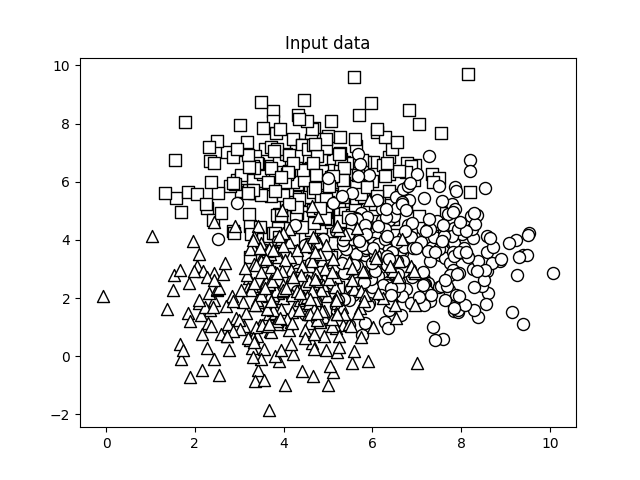


Рис. 1. Результат виконання скрипту LR\_4\_task\_1

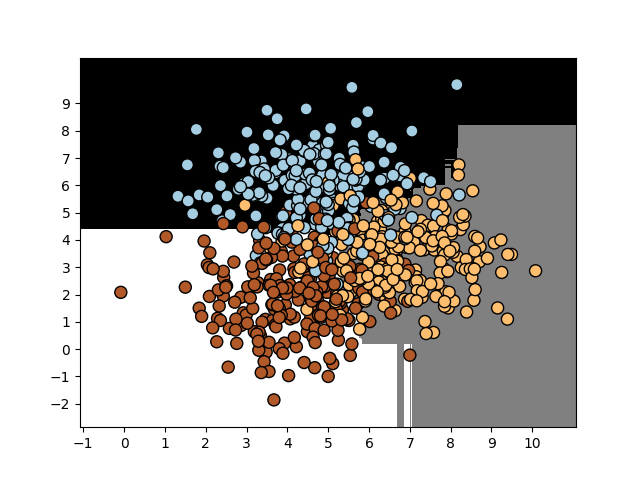


Рис. 2. Результат виконання скрипту LR\_4\_task\_1

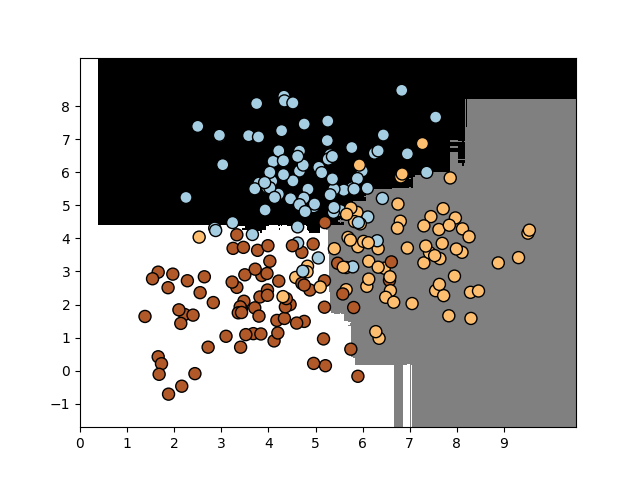


Рис. 3. Результат виконання скрипту LR\_4\_task\_1

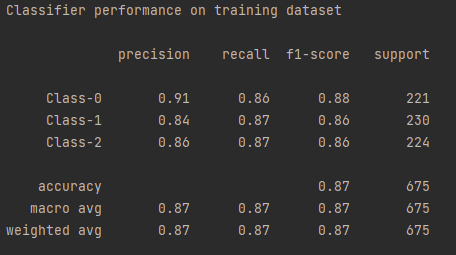


Рис. 4. Результат виконання скрипту LR\_4\_task\_1

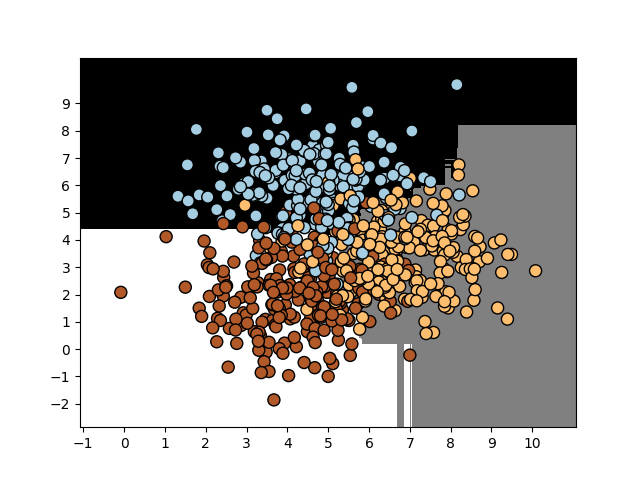


Рис. 5. Результат виконання скрипту LR\_4\_task\_1

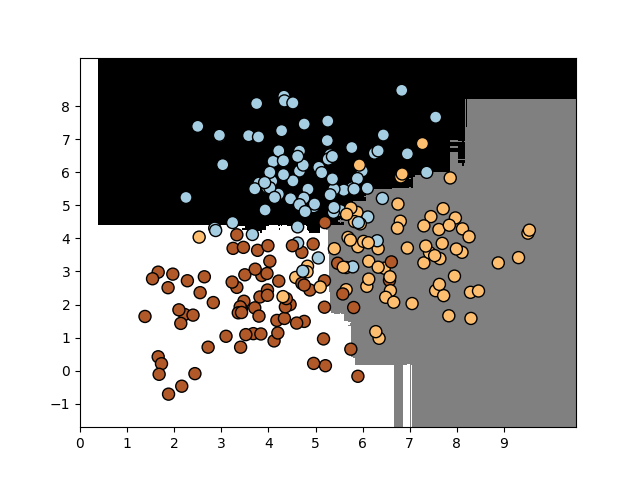


Рис. 6. Результат виконання скрипту LR\_4\_task\_1

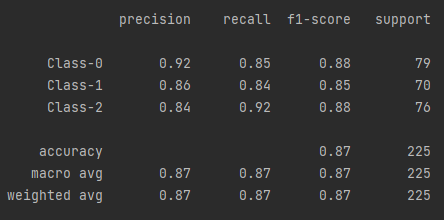


Рис. 7. Результат виконання скрипту LR\_4\_task\_1

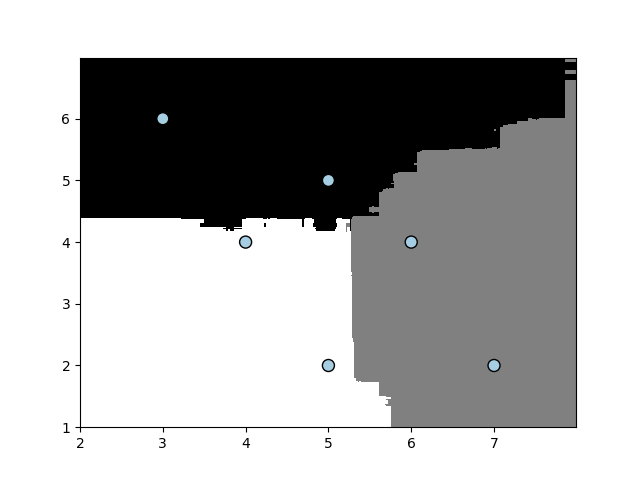


Рис. 8. Результат виконання скрипту LR\_4\_task\_1

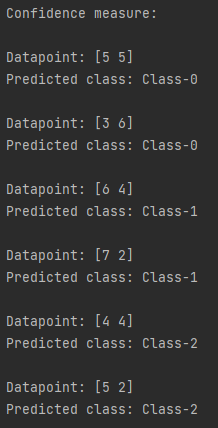


Рис. 9. Результат виконання скрипту LR\_4\_task\_1

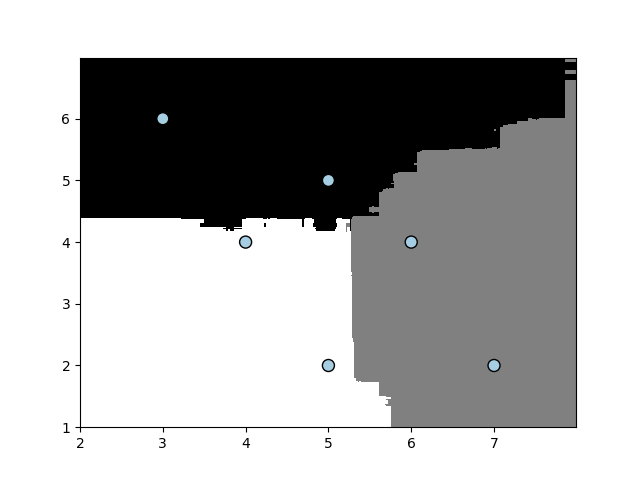


Рис. 10. Результат виконання скрипту LR\_4\_task\_1

Зображення, що містить текст

Автоматично згенерований опис

Рис. 11. Результат виконання скрипту LR\_4\_task\_1

Було встановлено, що при аргументів **-erf** отримуються більш валідні піки. Це обумовлено тим, що в процесі навчання гранично випадкові ліси мають більше можливостей для вибору оптимальних дерев рішень, тому, як правило, вони забезпечують отримання кращих границь. Але кінцеві результати виявилися майже однаковими при використанні обох прапорців.

**Завдання №2: Обробка дисбалансу класів.**

Код скрипту **LR\_4\_task\_2.py**:

import sys  
import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.ensemble import ExtraTreesClassifier  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import classification\_report  
from utilities import visualize\_classifier  
  
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 = {'n\_estimators': 100, 'max\_depth': 4, 'random\_state': 0, 'class\_weight': 'balanced'}  
 else:  
 raise TypeError("Invalid input argument; should be 'balance'")  
classifier = ExtraTreesClassifier(\*\*params)  
classifier.fit(X\_train, y\_train)  
visualize\_classifier(classifier, X\_train, y\_train)  
y\_test\_pred = classifier.predict(X\_test)  
visualize\_classifier(classifier, X\_test, y\_test)  
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()

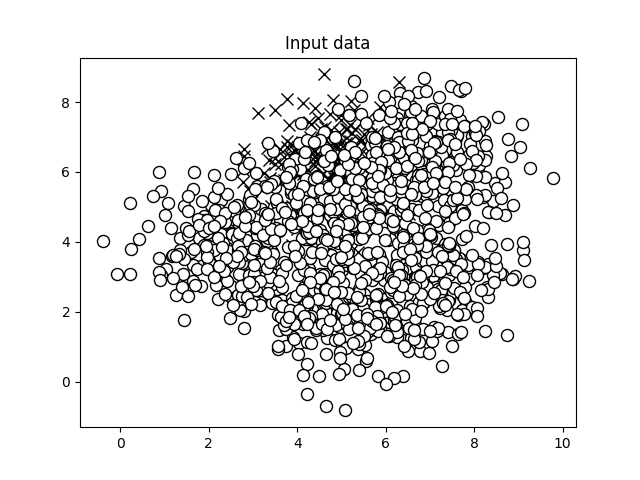


Рис. 12. Результат виконання скрипту LR\_4\_task\_2

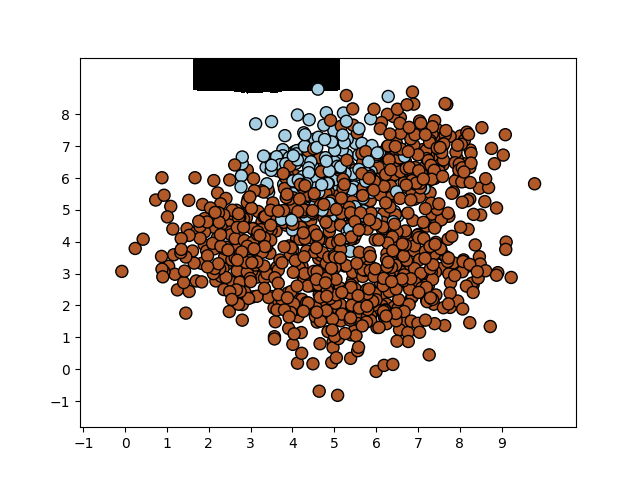


Рис. 13. Результат виконання скрипту LR\_4\_task\_2

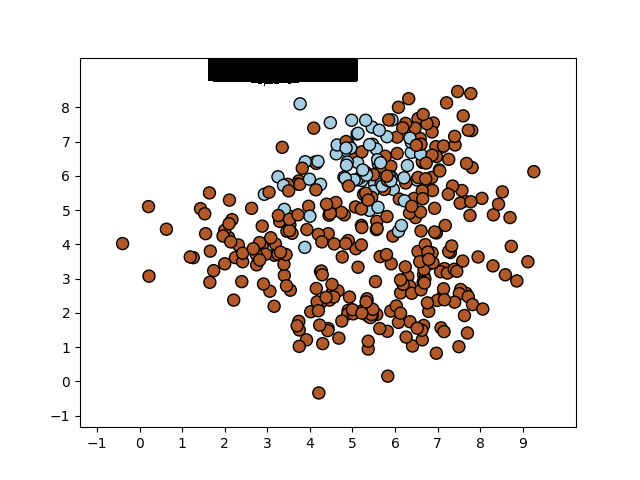


Рис. 14. Результат виконання скрипту LR\_4\_task\_2

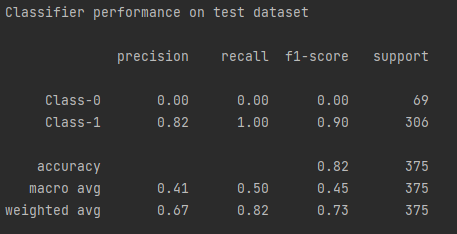


Рис. 15. Результат виконання скрипту LR\_4\_task\_2

**Завдання №3: Знаходження оптимальних навчальних параметрів за допомогою сіткового пошуку.**

**Код скрипту LR\_4\_task\_3.py:**

import numpy as np  
from sklearn.model\_selection import GridSearchCV  
from sklearn.ensemble import ExtraTreesClassifier  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import classification\_report  
import pandas as pd  
  
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("\n##### Searching optimal parameters for", metric)  
 classifier = GridSearchCV(  
 ExtraTreesClassifier(random\_state=0),  
 parameter\_grid, cv=5, scoring=metric)  
 classifier.fit(X\_train, y\_train)  
 df = pd.DataFrame(classifier.cv\_results\_)  
 df\_columns\_to\_print = [column for column in df.columns if 'param' in column or 'score' in column]  
 print(df[df\_columns\_to\_print])  
 print("\nBest parameters:", classifier.best\_params\_)  
 y\_pred = classifier.predict(X\_test)  
 print("\nPerformance report:\n")  
 print(classification\_report(y\_test, y\_pred))

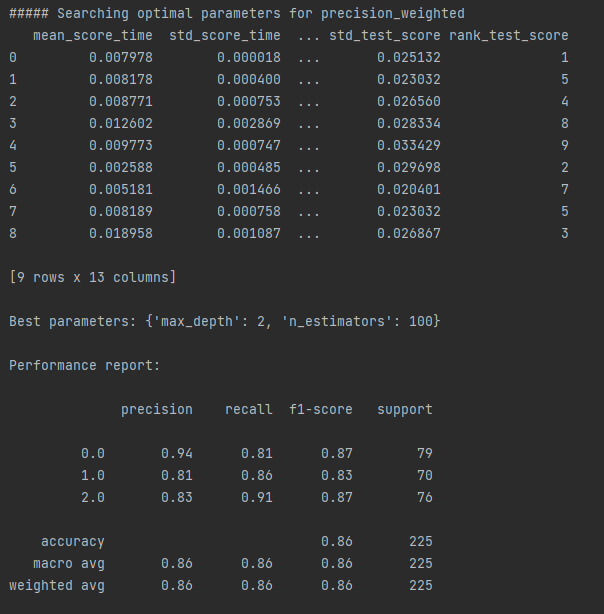


Рис. 16. Результат виконання скрипту LR\_4\_task\_3

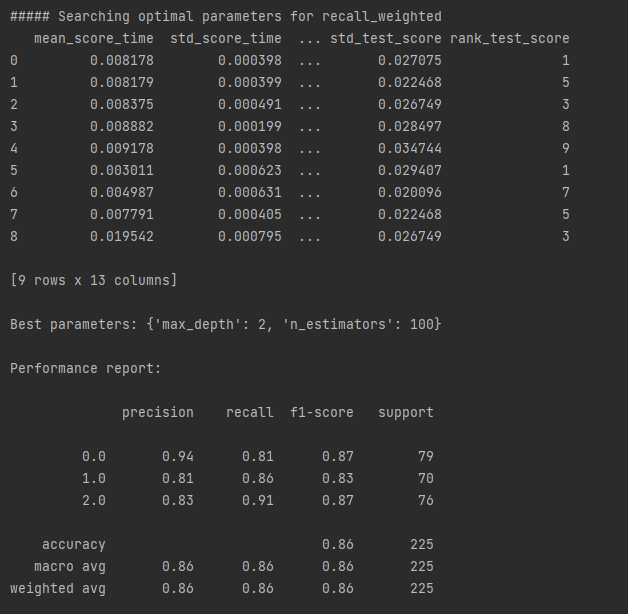


Рис. 17. Результат виконання скрипту LR\_4\_task\_3

**Завдання №4: Обчислення відносної важливості ознак.**

**Код скрипта LR\_4\_task\_4.py:**

import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.tree import DecisionTreeRegressor  
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(DecisionTreeRegressor(max\_depth=4),  
 n\_estimators=400, random\_state=7)  
regressor.fit(X\_train, y\_train)  
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))  
feature\_importances = regressor.feature\_importances\_  
feature\_names = housing\_data.feature\_names  
feature\_importances = 100.0 \* (feature\_importances / max(feature\_importances))  
index\_sorted = np.flipud(np.argsort(feature\_importances))  
pos = np.arange(index\_sorted.shape[0]) + 0.5  
plt.figure()  
plt.bar(pos, feature\_importances[index\_sorted], align='center')  
plt.xticks(pos, feature\_names[index\_sorted])  
plt.ylabel('Relative Importance')  
plt.title('Feature importance using AdaBoost regressor')  
plt.show()

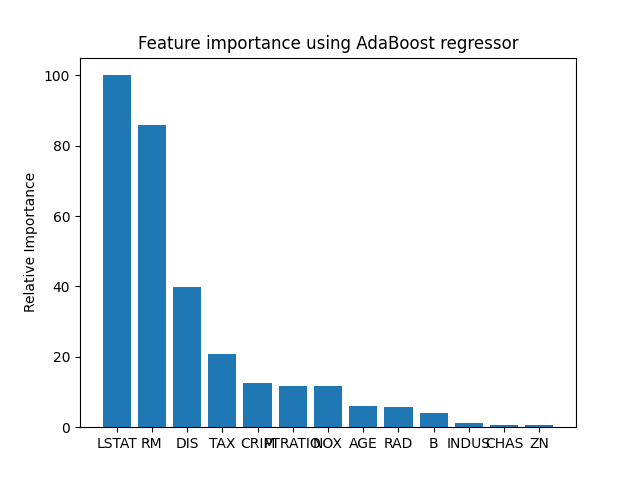


Рис. 18. Результат виконання скрипту LR\_4\_task\_4

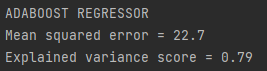


Рис. 19. Результат виконання скрипту LR\_4\_task\_4

**Завдання №5: Прогнозування інтенсивності дорожнього руху за допомогою класифікатора на основі гранично випадкових лісів.**

**Код скрипту LR\_4\_task\_5.py:**

import numpy as np  
from sklearn.metrics import mean\_absolute\_error  
from sklearn import preprocessing  
from sklearn.model\_selection import train\_test\_split  
from sklearn.ensemble import ExtraTreesRegressor  
  
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]))



Рис. 20. Результат виконання скрипту LR\_4\_task\_5

**Завдання №6: Створення навчального конвеєра (конвеєра машинного навчання).**

Код скрипту **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  
  
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=9)  
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)  
output = processor\_pipeline.predict(X)  
print("\nPredicted output:\n", output)  
print("\nScore:", processor\_pipeline.score(X, y))  
status = processor\_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]))

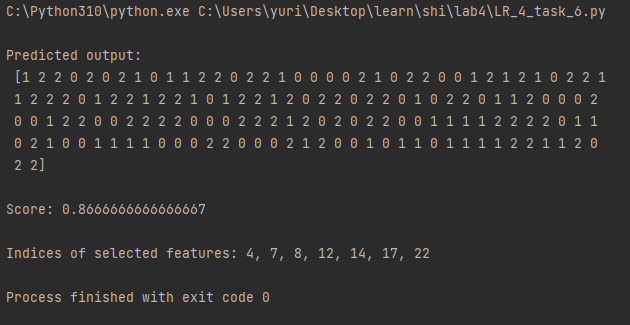


Рис. 21. Результат виконання скрипту LR\_4\_task\_6

**Завдання №7: Пошук найближчих сусідів.**

**Код скрипту LR\_4\_task\_7.py:**

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()

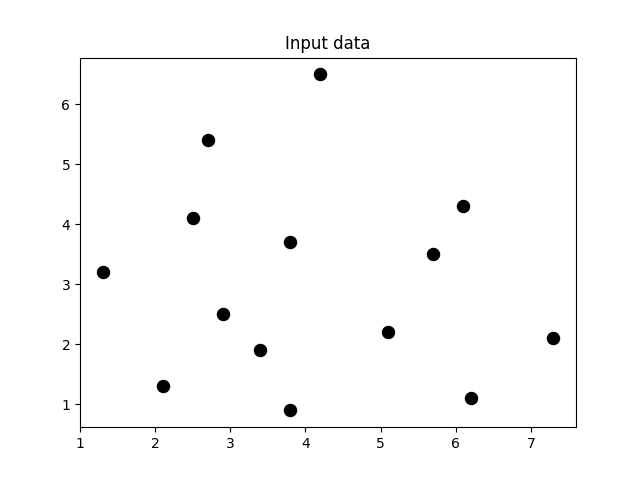
****

Рис. 22. Результат виконання скрипту LR\_4\_task\_7

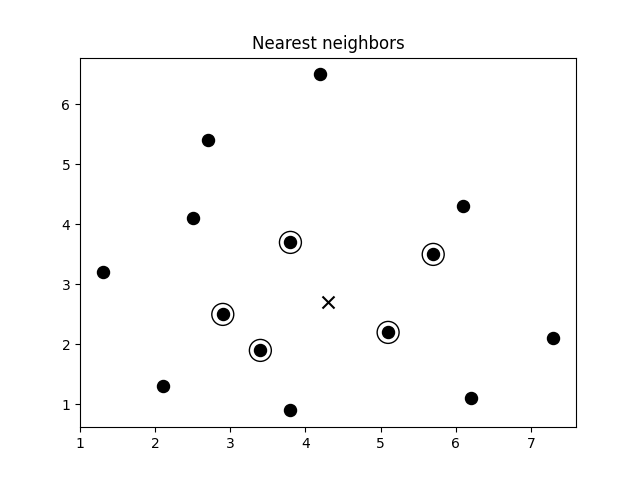
****

Рис. 23. Результат виконання скрипту LR\_4\_task\_7

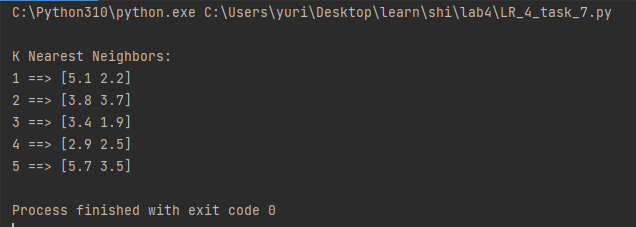


Рис. 24. Результат виконання скрипту LR\_4\_task\_7

На першому рисунку вхідні дані

На другому рисунку вхідні дані, тестова точка та її 5 найближчих сусідів (обведені)

На третьому рисунку п’ять найближчих сусідів

**Завдання №8: Створити класифікатор методом k найближчих сусідів.**

**Код скрипту LR\_4\_task\_8.py:**

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(np.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(np.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()

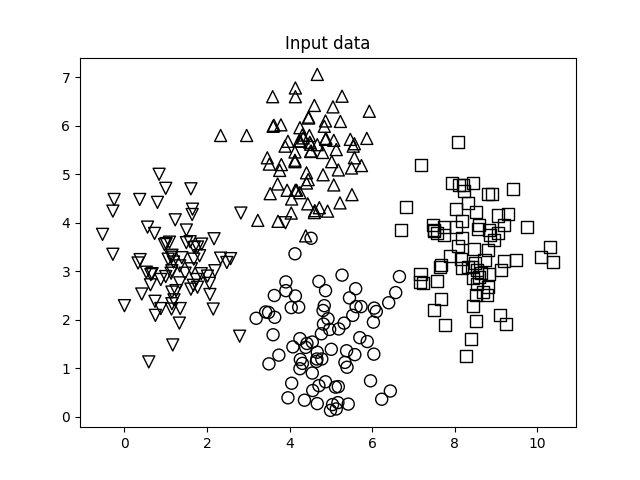
****

Рис. 24. Результат виконання скрипту LR\_4\_task\_8

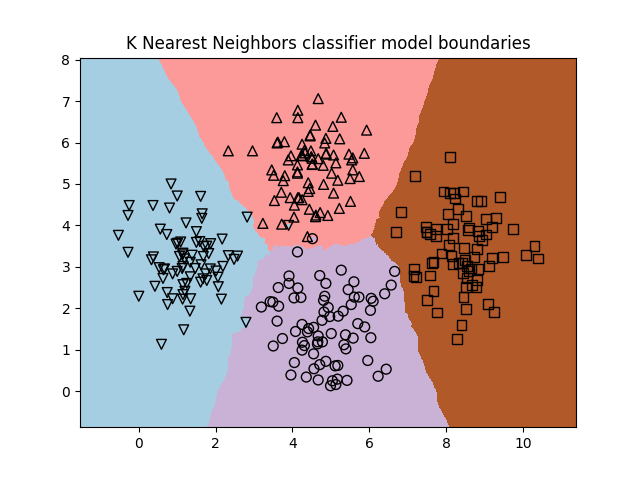


Рис. 25. Результат виконання скрипту LR\_4\_task\_8

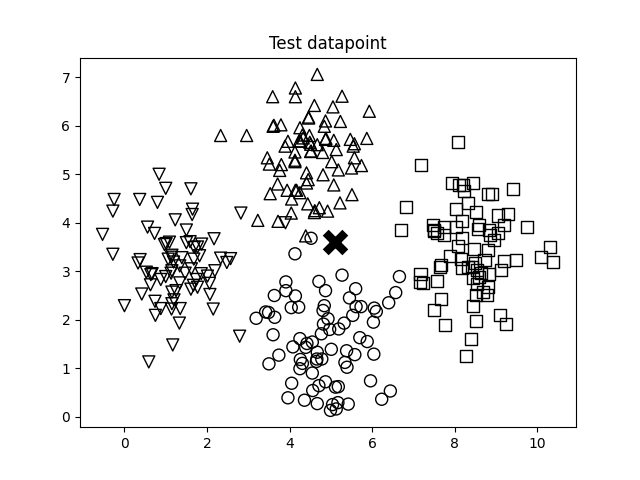
****

Рис. 26. Результат виконання скрипту LR\_4\_task\_8

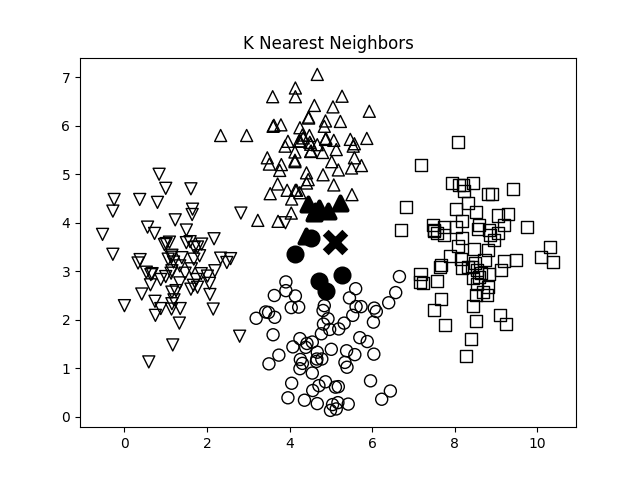
****

Рис. 27. Результат виконання скрипту LR\_4\_task\_8

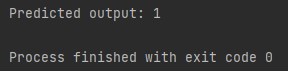


Рис. 28. Результат виконання скрипту LR\_4\_task\_8

**Завдання №9: Обчислення оцінок подібності.**

**Код скрипту 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  
  
  
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))

**Запускаємо скрипт з різними параметрами**

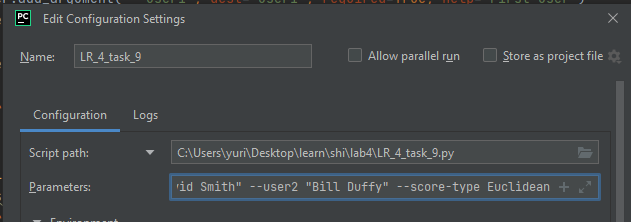


Рис. 29. Вікно параметрів

--user1 "David Smith" --user2 "Bill Duffy" --score-type Euclidean



--user1 "David Smith" --user2 "Bill Duffy" --score-type Pearson



--user1 "David Smith" --user2 "Samuel Miller" --score-type Euclidean



--user1 "David Smith" --user2 "Samuel Miller" --score-type Pearson



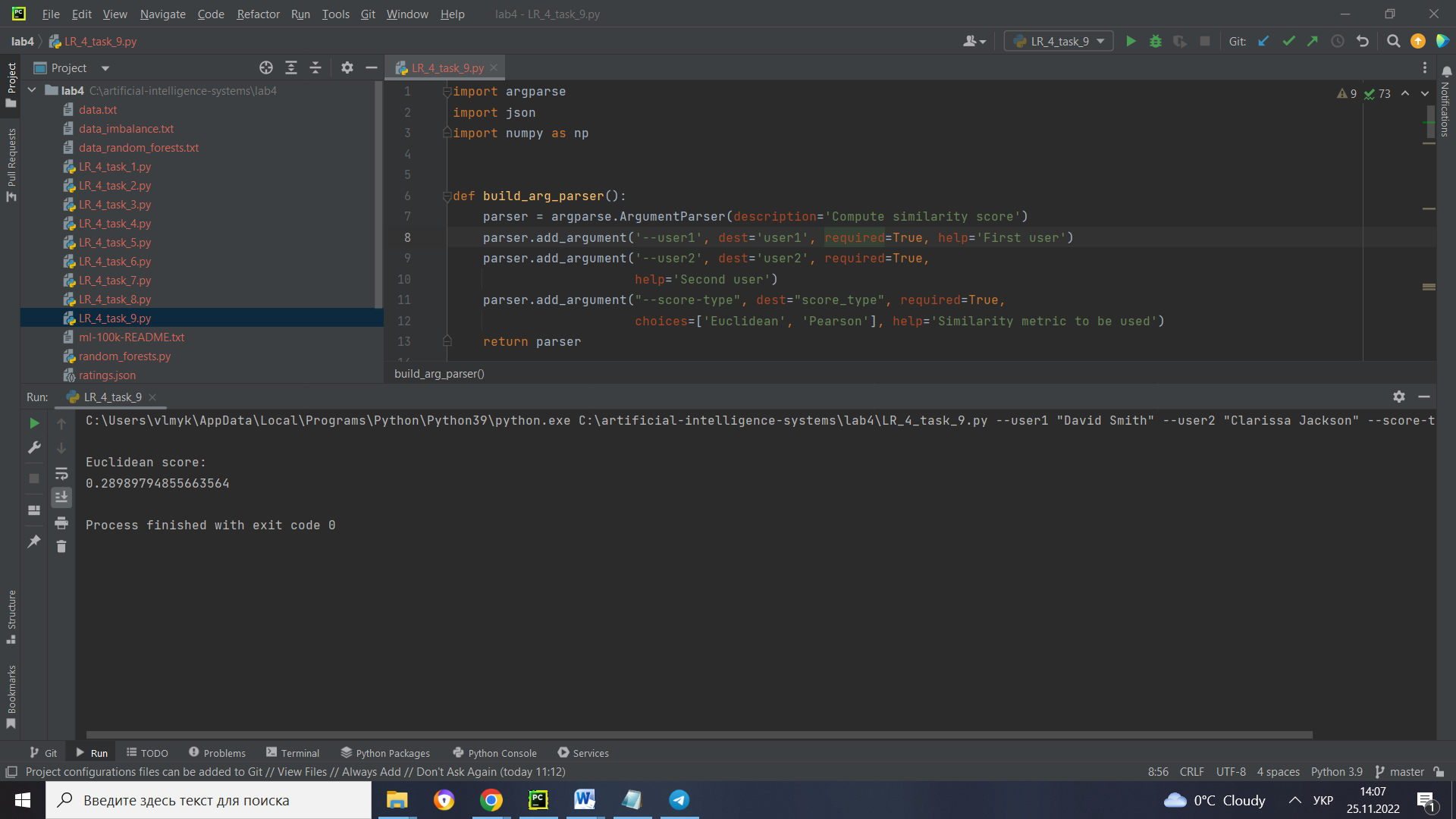
--user1 "David Smith" --user2 "Julie Hammel" --score-type Euclidean



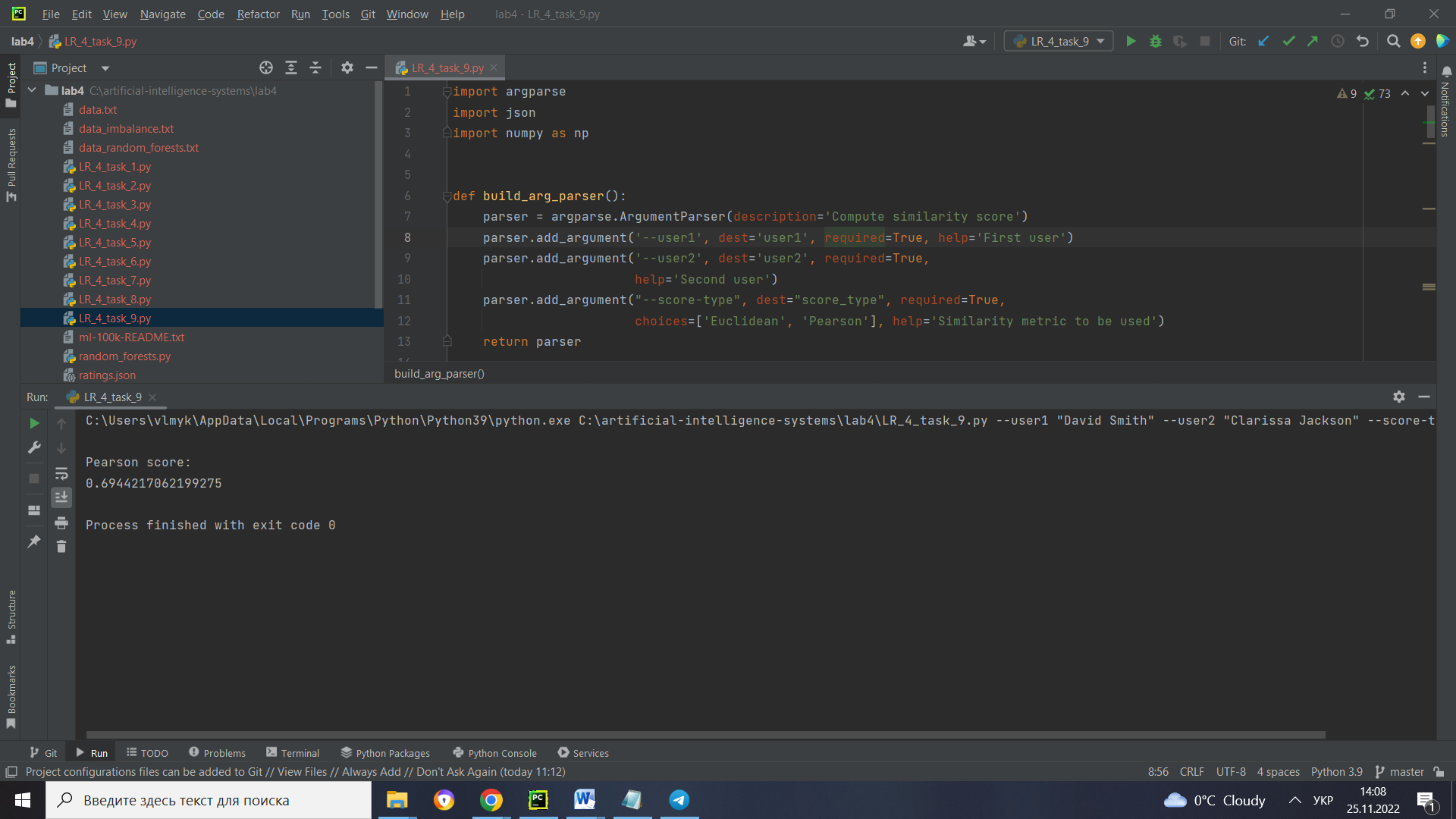
--user1 "David Smith" --user2 "Julie Hammel" --score-type Pearson



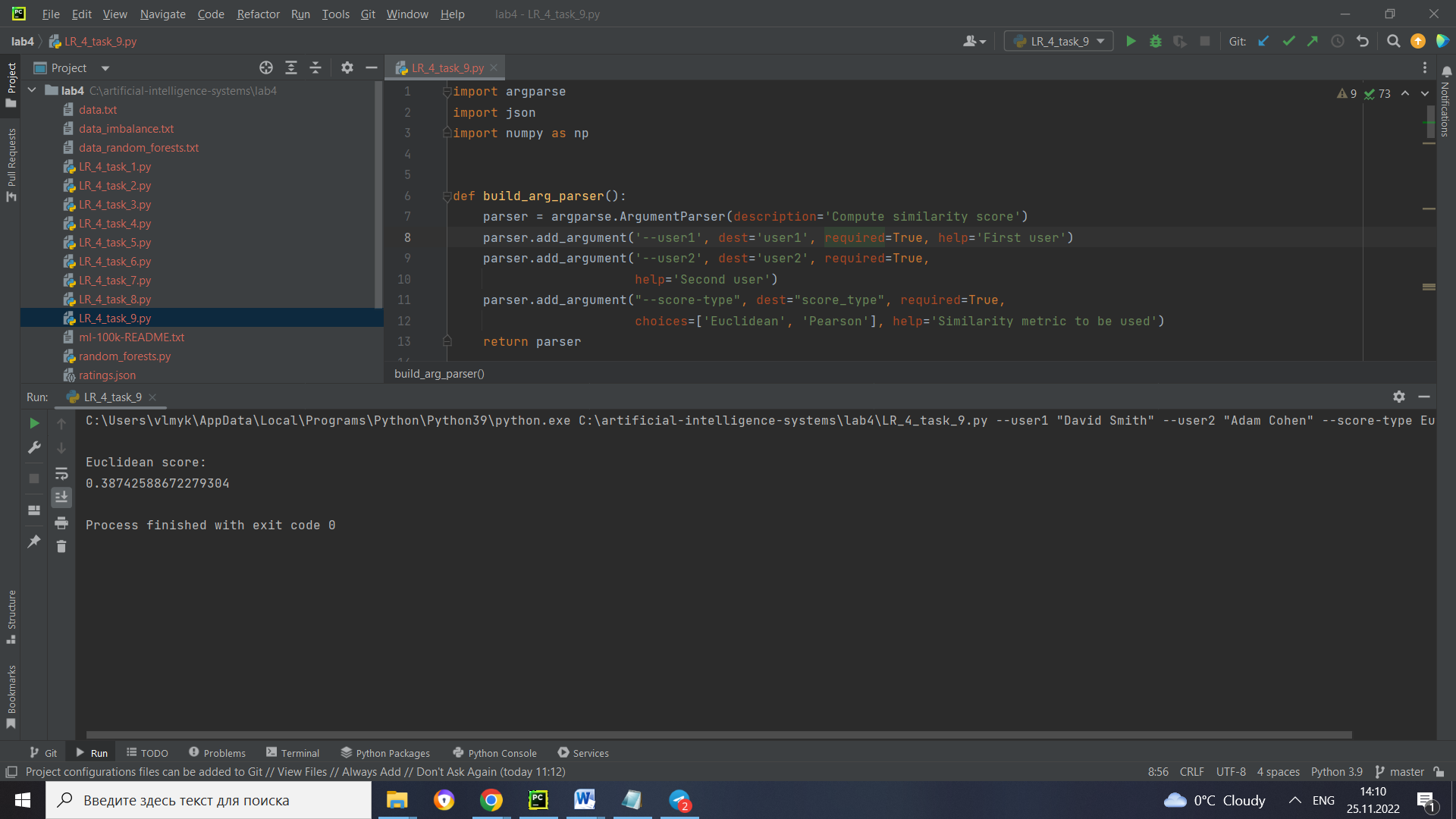
--user1 "David Smith" --user2 "Clarissa Jackson" --score-type Euclidean



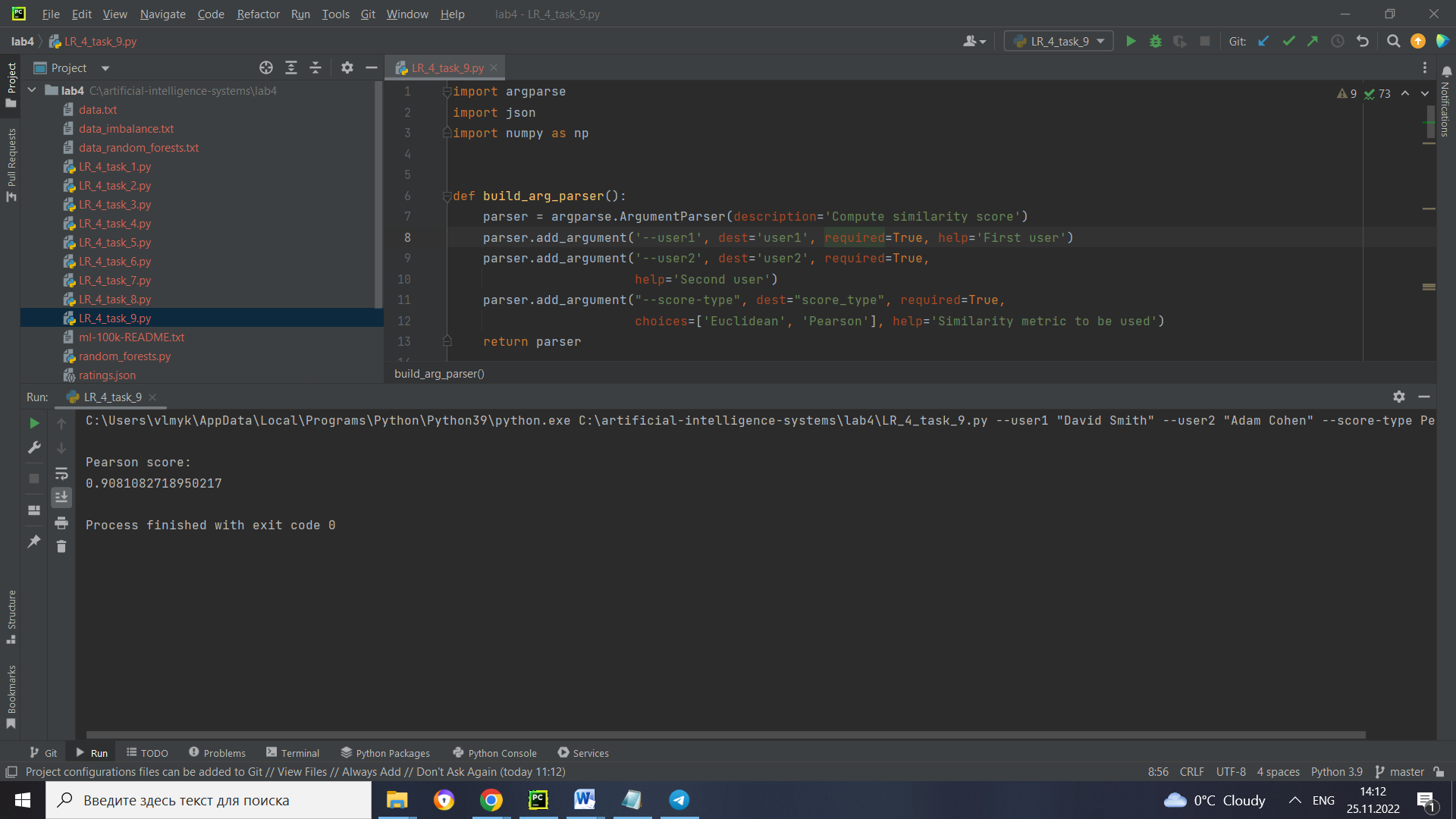
--user1 "David Smith" --user2 "Clarissa Jackson" --score-type Pearson



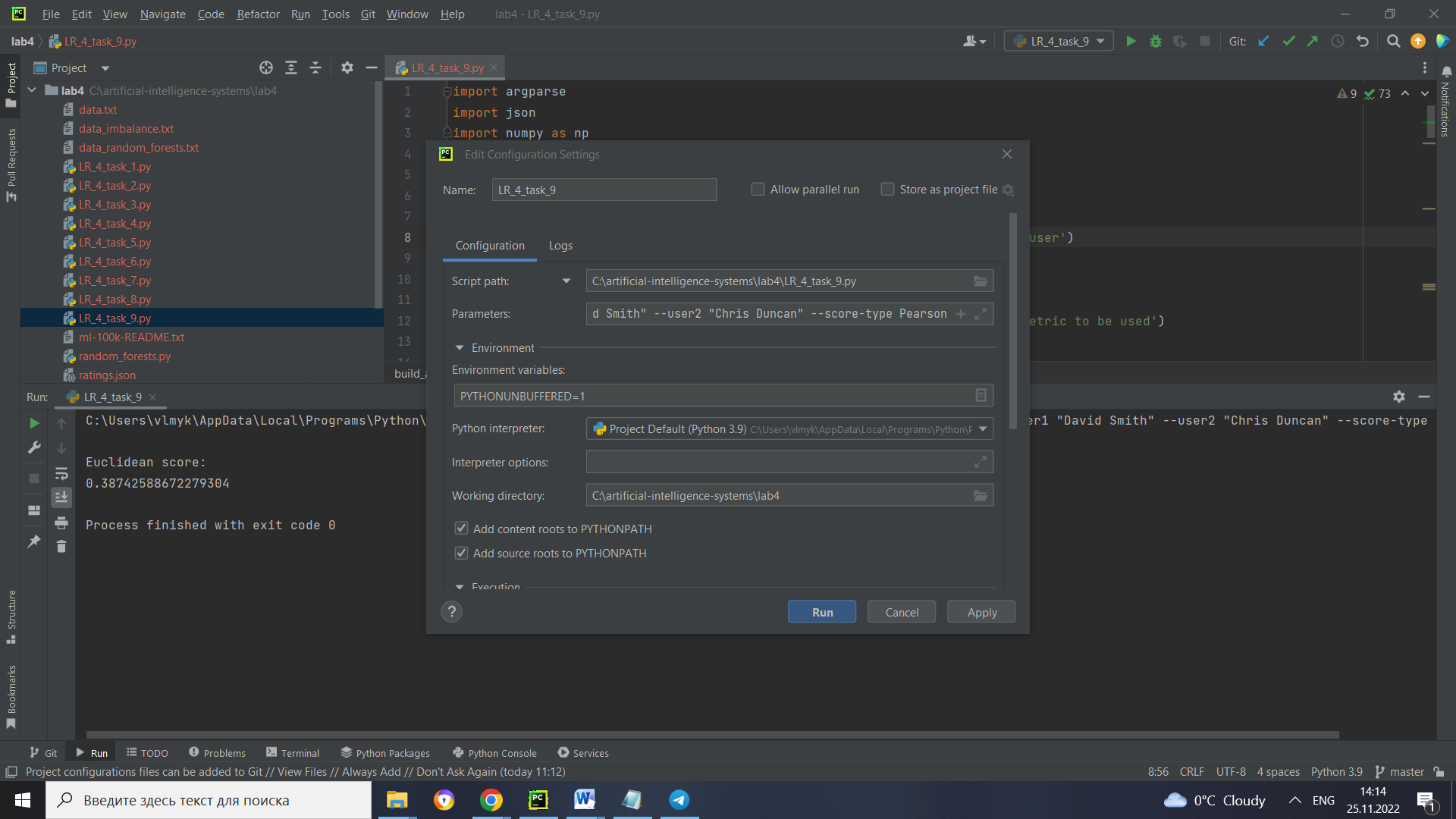
--user1 "David Smith" --user2 "Adam Cohen" --score-type Euclidean



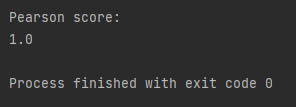
--user1 "David Smith" --user2 "Adam Cohen" --score-type Pearson



--user1 "David Smith" --user2 "Chris Duncan" --score-type Euclidean



--user1 "David Smith" --user2 "Chris Duncan" --score-type Pearson

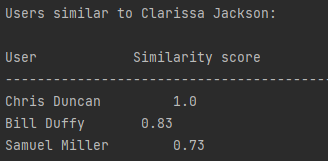


Оцінка подібності за Пірсоном демонструє кращі результати в порівнянні з евклідовою оцінкою подібності.

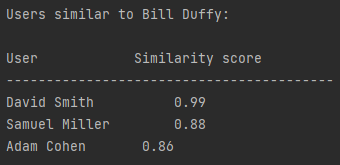
**Завдання №10: Пошук користувачів зі схожими уподобаннями методом колаборативної фільтрації.**

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 in-put 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('\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))

--user "Clarissa Jackson"



--user "Bill Duffy"



Юзер “Clarissa Jackson” має одинакові вподобання з користувачем “Chris Duncan”, а користувач “Bill Duffy” – майже однакові з “David Smith”.

**Завдання №11: Створення рекомендаційної системи фільмів.**

Код скрипту LR\_4\_task\_11.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 the movie recommendations for the given 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')  
 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']  
 movie\_scores = np.array([[score / similarity\_scores[item], item]  
 for item, score in overall\_scores.items()])  
 movie\_scores = movie\_scores[np.argsort(movie\_scores[:, 0])[::-1]]  
 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)

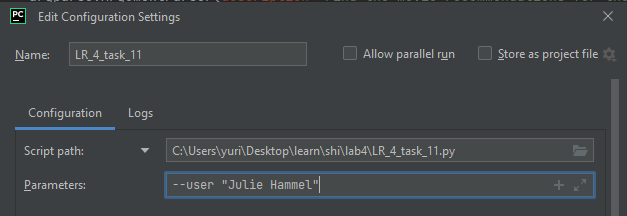


Рис. 30. Параметри

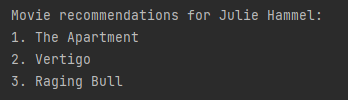


Рис. 31. Код скрипту LR\_4\_task\_11.py

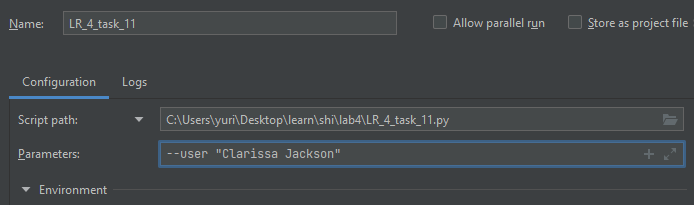


Рис. 31. Параметри



Рис. 31. Код скрипту LR\_4\_task\_11.py

**Висновок**: У ході виконання лабораторної роботи було використано спеціалізовані бібліотеки мови програмування Python, досліджено методи ансамблів у машинному навчанні та створено рекомендаційні системи.

GitHub відкритий на пошту ***valeriifilippovzt@gmail.com***.