Лабораторна робота №4

Тема: ДОСЛІДЖЕННЯ МЕТОДІВ АНСАМБЛЕВОГО НАВЧАННЯ ТА СТВОРЕННЯ РЕКОМЕНДАЦІЙНИХ СИСТЕМ

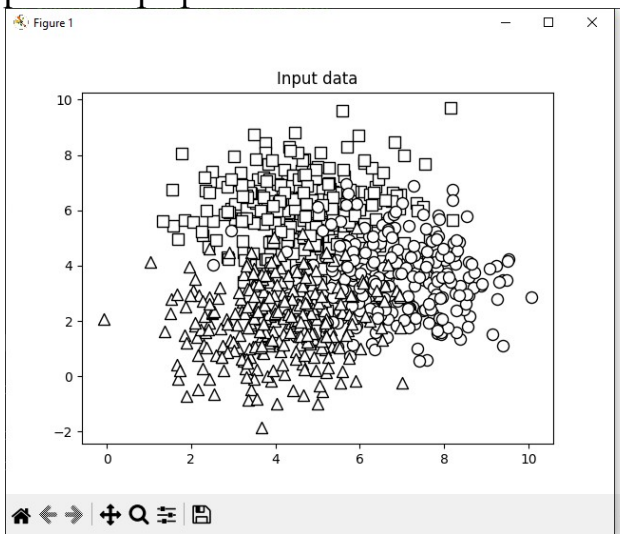
Мета: *використовуючи спеціалізовані бібліотеки та мову програмування Python дослідити методи ансамблів у машинному навчанні та створити рекомендаційні системи.*

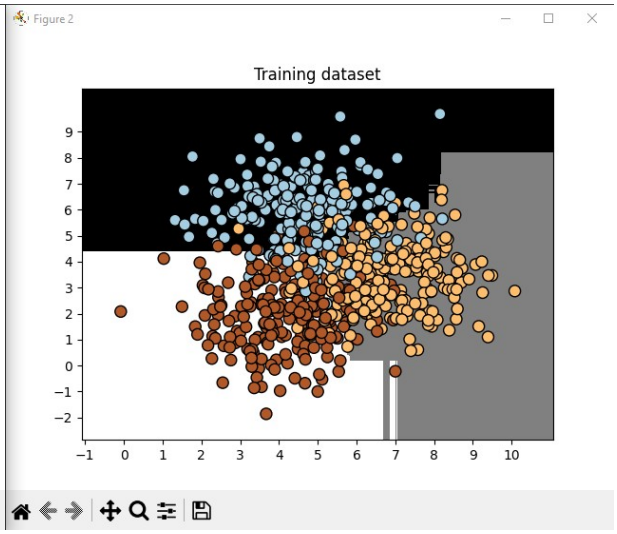
Хід роботи

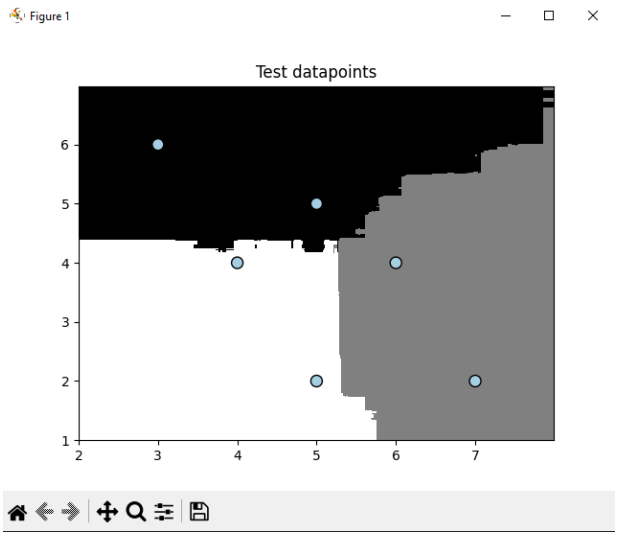
GitHub репозиторій:
https://github.com/BashmanivskiyMaxim/Artificial_intelligence_labs

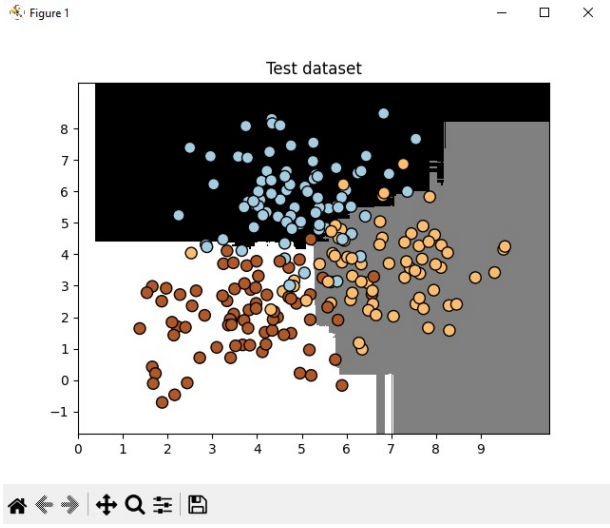
Завдання 2.1: Створення класифікаторів на основі випадкових та гранично випадкових лісів

Отримані графіки:







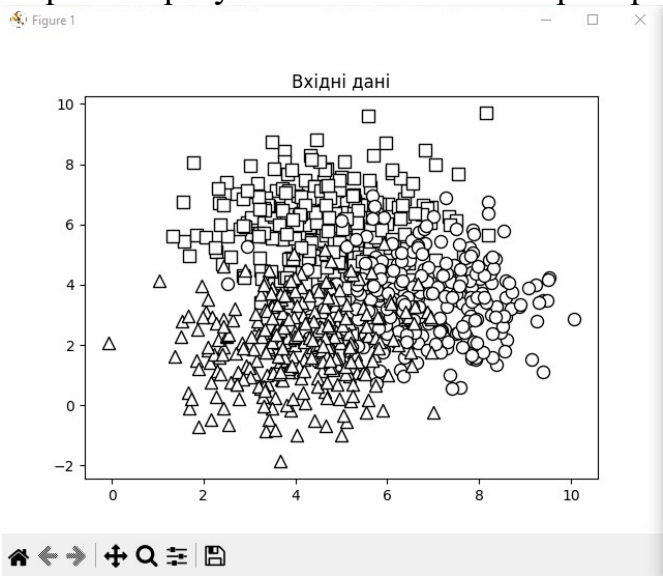


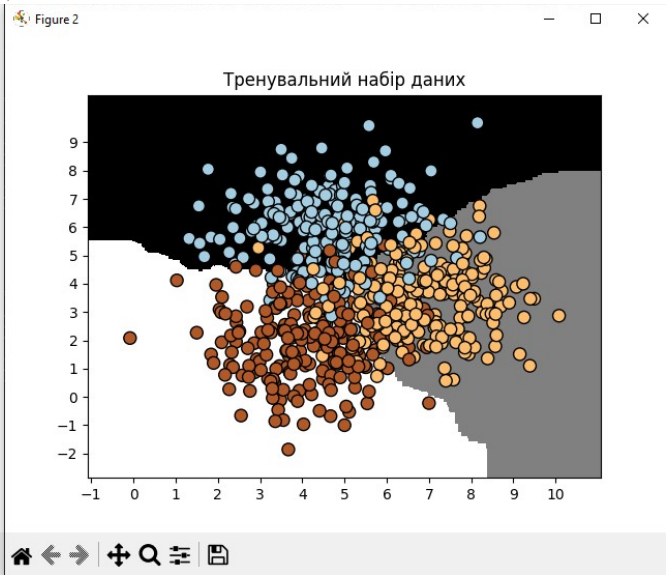
					ДУ «Житомирська політехніка».23.121.3.000 – Лр2		
Змн.	*Арк.*	*№ докум.*	*Підпис*	*Дата*			
Розроб.		*Башманівський М.*			Звіт з лабораторної роботи	*Літ.*	*Арк.* 1 / *Аркушів*
Перевір.		*Голенко М. Ю.*					
Керівник						*ФІКТ Гр. ІПЗ-20-3[1]*	
Н. контр.							
Зав. каф.							

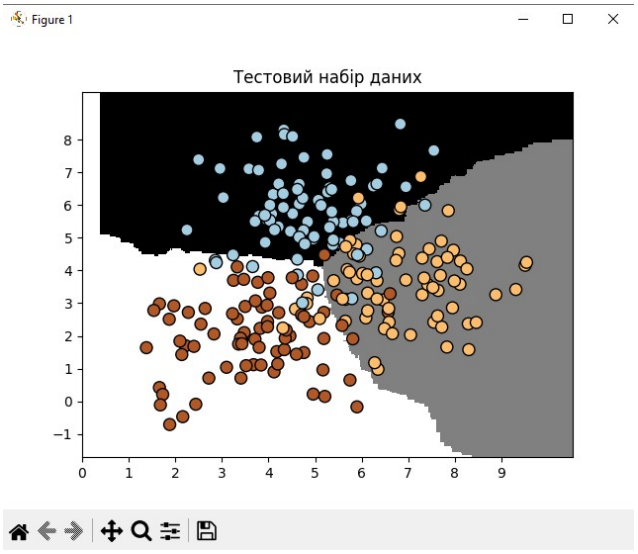
Результат виконання програми rf:

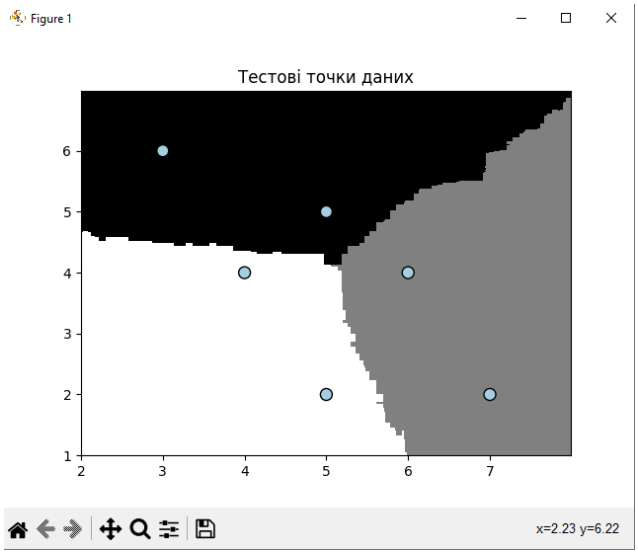
```
PS C:\AI_labs\lab4> python3 random_forests.py --classifier-type rf
########################################

Classifier performance on training dataset

              precision    recall  f1-score   support

     Class-0       0.91      0.86      0.88       221
     Class-1       0.84      0.87      0.86       230
     Class-2       0.86      0.87      0.86       224

    accuracy                           0.87       675
   macro avg       0.87      0.87      0.87       675
weighted avg       0.87      0.87      0.87       675

########################################

########################################

Classifier performance on test dataset

              precision    recall  f1-score   support

     Class-0       0.92      0.85      0.88        79
     Class-1       0.86      0.84      0.85        70
     Class-2       0.84      0.92      0.88        76

    accuracy                           0.87       225
   macro avg       0.87      0.87      0.87       225
weighted avg       0.87      0.87      0.87       225

########################################


Confidence measure:

Datapoint: [5 5]
Predicted class: Class-0

Datapoint: [3 6]
Predicted class: Class-0

Datapoint: [6 4]
Predicted class: Class-1

Datapoint: [7 2]
Predicted class: Class-1

Datapoint: [4 4]
Predicted class: Class-2
```

Отримані результати після зміни прапорця:









Результат виконання програми з прапорцем erf:

```
PS C:\AI_labs\lab4> python3 LR_4_task_1.py --classifier-type erf
########################################

Classifier performance on training dataset

              precision    recall  f1-score   support

     Class-0       0.89      0.83      0.86       221
     Class-1       0.82      0.84      0.83       230
     Class-2       0.83      0.86      0.85       224

    accuracy                           0.85       675
   macro avg       0.85      0.85      0.85       675
weighted avg       0.85      0.85      0.85       675

########################################

########################################

Classifier performance on test dataset

              precision    recall  f1-score   support

     Class-0       0.92      0.85      0.88        79
     Class-1       0.84      0.84      0.84        70
     Class-2       0.85      0.92      0.89        76

    accuracy                           0.87       225
   macro avg       0.87      0.87      0.87       225
weighted avg       0.87      0.87      0.87       225

########################################


Confidense measure:

Datapoint: [5 5]
Predicted class: Class-0

Datapoint: [3 6]
Predicted class: Class-0

Datapoint: [6 4]
Predicted class: Class-1

Datapoint: [7 2]
Predicted class: Class-1

Datapoint: [4 4]
Predicted class: Class-2

Datapoint: [5 2]
Predicted class: Class-2
```

Лістинг програми:

```
import argparse
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier
from sklearn.model_selection import train_test_split
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("Вхідні дані")

    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, "Тренувальний набір даних")

    y_test_pred = classifier.predict(X_test)
    visualize_classifier(classifier, X_test, y_test, "Тестовий набір даних")

    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("\nConfidense 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), "Тестові точки даних"
    )
    plt.show()
```

Висновок: RandomForestClassifier і ExtraTreesClassifier - це два різних класифікатори, які обидва належать до сімейства ансамблевих методів, спрямованих на підвищення точності класифікації та зменшення перенавчання. Обидва вони базуються на ідеї "випадкових лісів" (Random Forests), але мають деякі відмінності:

RandomForestClassifier: Під час навчання випадковим чином вибирає підмножину ознак для розгляду на кожному вузлі дерева рішень. Це робить RandomForest більш стійким до перенавчання і дозволяє враховувати більше можливих ознак при класифікації.

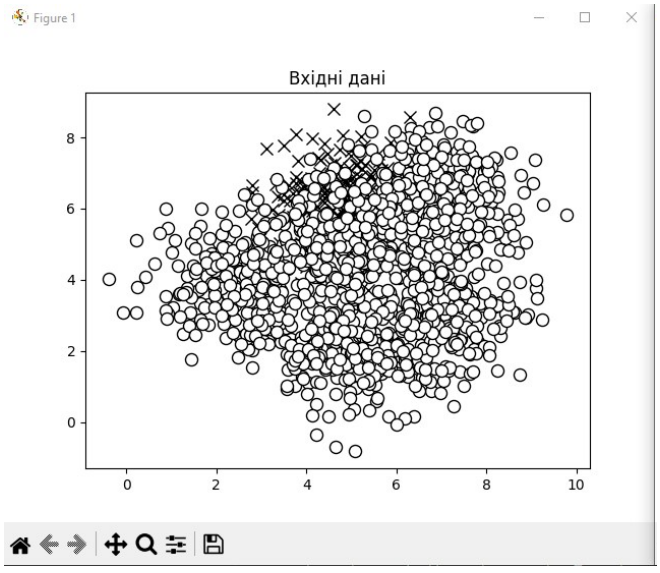
ExtraTreesClassifier: Extra Trees є ще більш випадковими, оскільки вони випадково вибирають ознаки та розподіл даних для кожного вузла дерева. Це призводить до ще більшої випадковості та робить їх менш вразливими до перенавчання.

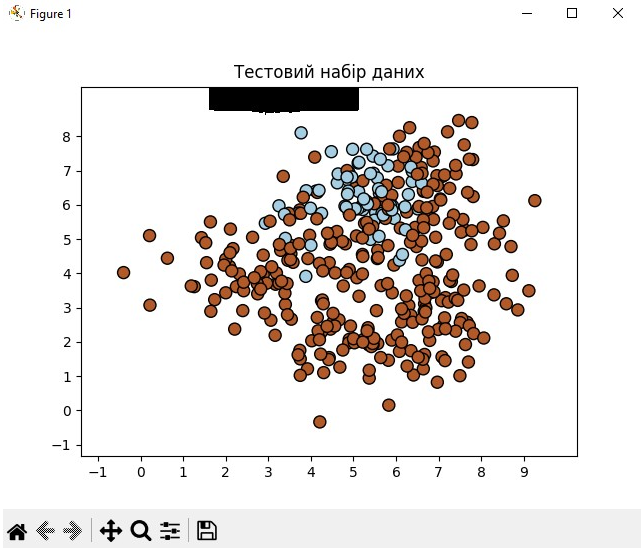
RandomForestClassifier: Вибирає найкраще розбиття на основі критерію, такого як індекс Джині або ентропія, відповідно до вибору ознак.

ExtraTreesClassifier: Вибирає розбиття випадковим чином для кожної ознаки та обчислює, яке з них є найкращим.

Завдання 2.2: Обробка дисбалансу класів

Отримані графіки:

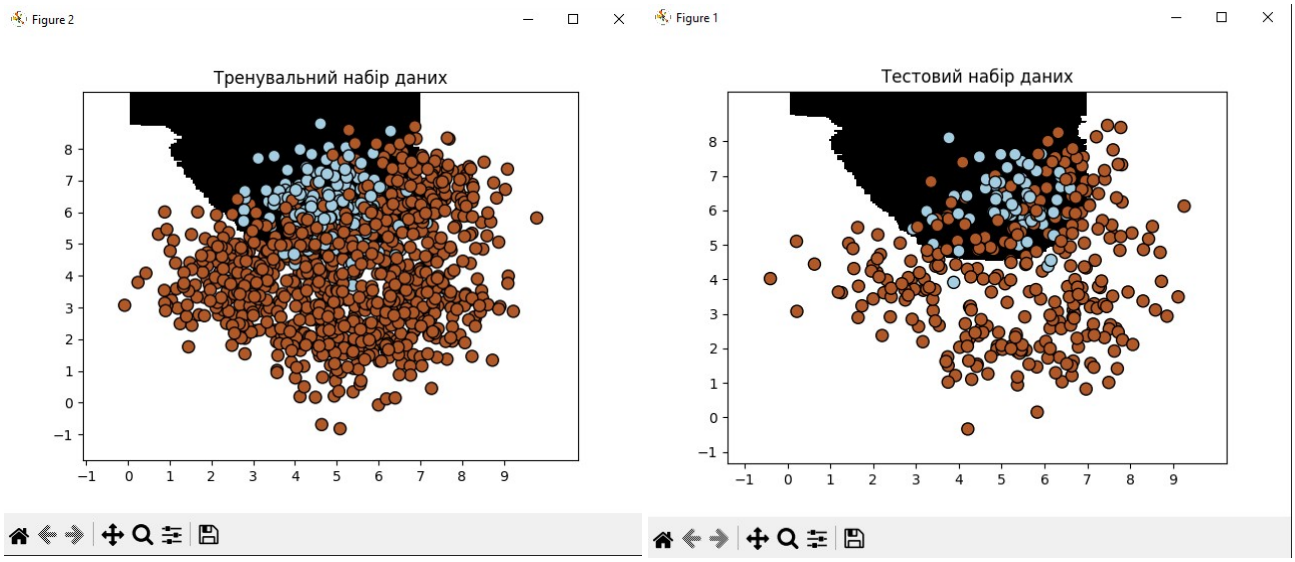




Результат виконання програми:

```
PS C:\AI_labs> & E:/Users/madma/AppData/Local/Microsoft/W
c:\AI_labs\lab4\LR_4_task_2.py:17: UserWarning: You passe
lor.  This behavior may change in the future.
  plt.scatter(

########################################

Classifier performance on training dataset

              precision    recall  f1-score   support

     Class-0       1.00      0.01      0.01       181
     Class-1       0.84      1.00      0.91       944

    accuracy                           0.84      1125
   macro avg       0.92      0.50      0.46      1125
weighted avg       0.87      0.84      0.77      1125

########################################

########################################

Classifier performance on test dataset

E:\Users\madma\AppData\Local\Packages\PythonSoftwareFounda
definedMetricWarning: Precision and F-score are ill-defin
  _warn_prf(average, modifier, msg_start, len(result))
E:\Users\madma\AppData\Local\Packages\PythonSoftwareFounda
definedMetricWarning: Precision and F-score are ill-defin
  _warn_prf(average, modifier, msg_start, len(result))
E:\Users\madma\AppData\Local\Packages\PythonSoftwareFounda
definedMetricWarning: Precision and F-score are ill-defin
  _warn_prf(average, modifier, msg_start, len(result))
              precision    recall  f1-score   support

     Class-0       0.00      0.00      0.00        69
     Class-1       0.82      1.00      0.90       306

    accuracy                           0.82       375
   macro avg       0.41      0.50      0.45       375
weighted avg       0.67      0.82      0.73       375

########################################

PS C:\AI_labs>
```

Графік з врахуванням дисбалансу класів даних класифікатора:



Результат виконання програми:

```
PS C:\AI_labs\lab4> python3 LR_4_task_2.py balance
C:\AI_labs\lab4\LR_4_task_2.py:17: UserWarning: You pas
lor.  This behavior may change in the future.
  plt.scatter(

########################################

Classifier performance on training dataset

              precision    recall  f1-score   support

     Class-0       0.44      0.93      0.60       181
     Class-1       0.98      0.77      0.86       944

    accuracy                           0.80      1125
   macro avg       0.71      0.85      0.73      1125
weighted avg       0.89      0.80      0.82      1125

########################################

########################################

Classifier performance on test dataset

              precision    recall  f1-score   support

     Class-0       0.45      0.94      0.61        69
     Class-1       0.98      0.74      0.84       306

    accuracy                           0.78       375
   macro avg       0.72      0.84      0.73       375
weighted avg       0.88      0.78      0.80       375

########################################

PS C:\AI_labs\lab4>
```

Висновок: Цей код має можливість враховувати дисбаланс класів за допомогою параметра `class_weight` для класифікатора ExtraTreesClassifier. В частині параметрів класифікатора можна встановити `class_weight` на значення "balanced", яке автоматично розраховує ваги класів на основі їх розподілу в тренувальному наборі даних. Це корисний підхід для обробки дисбалансу класів, коли один клас має значно більше прикладів, ніж інший.

Завдання 2.3: Знаходження оптимальних навчальних параметрів за допомогою сіткового пошуку

Результат виконання програми:

```
PS C:\AI_labs\lab4> & E:/Users/madma/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/AI_labs/lab4/LR_4_task_3.py

#### Searching for optimal parameters for precision_weighted

Grid scores for the parameter grid:
{'max_depth': 2, 'n_estimators': 100} --> 1
{'max_depth': 4, 'n_estimators': 100} --> 5
{'max_depth': 7, 'n_estimators': 100} --> 4
{'max_depth': 12, 'n_estimators': 100} --> 8
{'max_depth': 17, 'n_estimators': 100} --> 9
{'max_depth': 4, 'n_estimators': 25} --> 2
{'max_depth': 4, 'n_estimators': 50} --> 7
{'max_depth': 4, 'n_estimators': 100} --> 5
{'max_depth': 4, 'n_estimators': 250} --> 3

Best parameters: {'max_depth': 2, 'n_estimators': 100}

#### Searching for optimal parameters for recall_weighted

Grid scores for the parameter grid:
{'max_depth': 2, 'n_estimators': 100} --> 1
{'max_depth': 4, 'n_estimators': 100} --> 5
{'max_depth': 7, 'n_estimators': 100} --> 3
{'max_depth': 12, 'n_estimators': 100} --> 8
{'max_depth': 17, 'n_estimators': 100} --> 9
{'max_depth': 4, 'n_estimators': 25} --> 1
{'max_depth': 4, 'n_estimators': 50} --> 7
{'max_depth': 4, 'n_estimators': 100} --> 5
{'max_depth': 4, 'n_estimators': 250} --> 3

Best parameters: {'max_depth': 2, 'n_estimators': 100}

Performance report:

              precision    recall  f1-score   support

         0.0       0.94      0.81      0.87        79
         1.0       0.81      0.86      0.83        70
         2.0       0.83      0.91      0.87        76

    accuracy                           0.86       225
   macro avg       0.86      0.86      0.86       225
weighted avg       0.86      0.86      0.86       225
```

Результат виконання програми з іншими значеннями параметрів:

```
PS C:\AI_labs\lab4> & E:/Users/madma/AppData/Local/Microsoft/Wi

#### Searching for optimal parameters for precision_weighted

Grid scores for the parameter grid:
{'max_depth': 2, 'n_estimators': 100} --> 1
{'max_depth': 4, 'n_estimators': 100} --> 5
{'max_depth': 7, 'n_estimators': 100} --> 4
{'max_depth': 12, 'n_estimators': 100} --> 8
{'max_depth': 18, 'n_estimators': 100} --> 9
{'max_depth': 4, 'n_estimators': 25} --> 2
{'max_depth': 4, 'n_estimators': 50} --> 7
{'max_depth': 4, 'n_estimators': 100} --> 5
{'max_depth': 4, 'n_estimators': 250} --> 3

Best parameters: {'max_depth': 2, 'n_estimators': 100}

#### Searching for optimal parameters for recall_weighted

Grid scores for the parameter grid:
{'max_depth': 2, 'n_estimators': 100} --> 1
{'max_depth': 7, 'n_estimators': 100} --> 3
{'max_depth': 12, 'n_estimators': 100} --> 8
{'max_depth': 18, 'n_estimators': 100} --> 9
{'max_depth': 4, 'n_estimators': 25} --> 1
{'max_depth': 4, 'n_estimators': 50} --> 7
{'max_depth': 4, 'n_estimators': 100} --> 5
{'max_depth': 4, 'n_estimators': 250} --> 3

Best parameters: {'max_depth': 2, 'n_estimators': 100}

Performance report:

              precision    recall  f1-score   support

         0.0       0.94      0.81      0.87        79
         1.0       0.81      0.86      0.83        70
         2.0       0.83      0.91      0.87        76

    accuracy                           0.86       225
   macro avg       0.86      0.86      0.86       225
weighted avg       0.86      0.86      0.86       225
```

Лістинг програми:

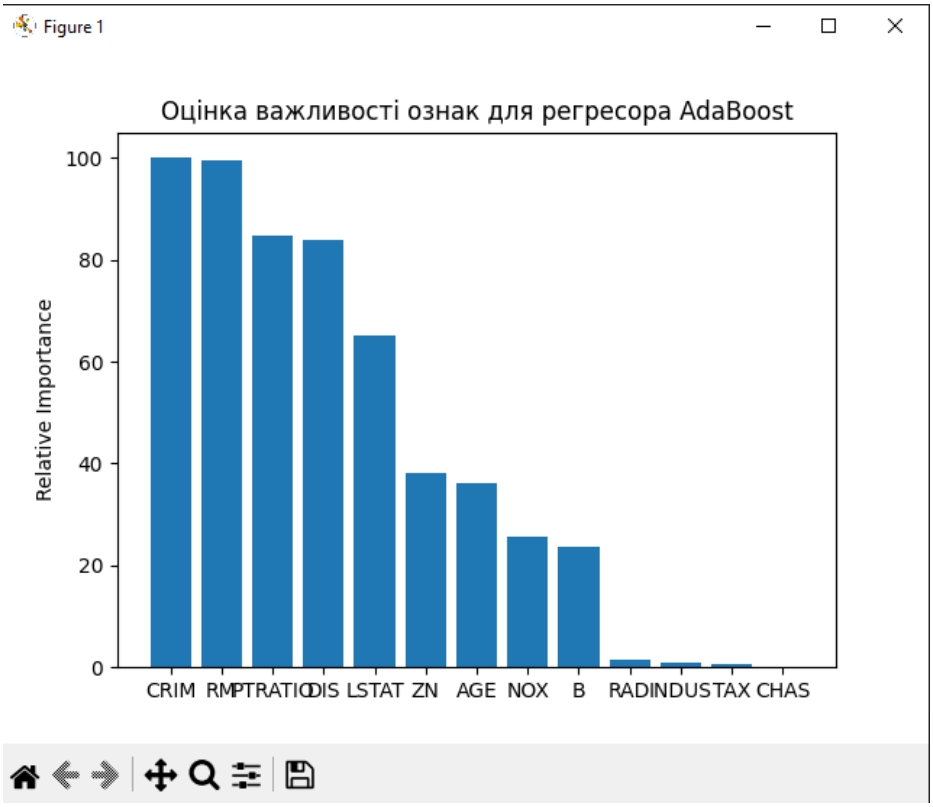
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.model_selection import train_test_split


# Завантаження даних з вхідного файлу
input_file = "data_random_forests.txt"
data = np.loadtxt(input_file, delimiter=",")
X, y = data[:, :-1], data[:, -1]
# Розділення даних на класи
class_0 = np.array(X[y == 0])
class_1 = np.array(X[y == 1])
class_2 = np.array(X[y == 2])
# Розділення даних на навчальний і тестовий набори
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=5
)
# Визначення сітки параметрів для пошуку оптимальних параметрів
parameter_grid = [
    {"n_estimators": [100], "max_depth": [2, 4, 7, 12, 18]},
    {"max_depth": [4], "n_estimators": [25, 50, 100, 250]},
]
# Метрики, для яких будуть шукатися оптимальні параметри
```

```
metrics = ["precision_weighted", "recall_weighted"]
for metric in metrics:
    print("\n#### Searching for optimal parameters for", metric)
    # Створення класифікатора і пошук оптимальних параметрів
    classifier = GridSearchCV(
        ExtraTreesClassifier(random_state=0), parameter_grid, cv=5, scoring=metric
    )
    classifier.fit(X_train, y_train)
    # Виведення результатів пошуку оптимальних параметрів
    print("\nGrid scores for the parameter grid:")
    for i in range(0, len(classifier.cv_results_["params"])):
        print(
            classifier.cv_results_["params"][i],
            "-->",
            classifier.cv_results_["rank_test_score"][i],
        )
    print("\nBest parameters:", classifier.best_params_)
# Передбачення класів на тестовому наборі та виведення звіту про продуктивність
y_pred = classifier.predict(X_test)
print("\nPerformance report:\n")
print(classification_report(y_test, y_pred))
```

Висновок: Цей код демонструє ефективний підхід до знаходження оптимальних параметрів моделі, який допомагає покращити її продуктивність для конкретних метрик якості, таких як точність (precision) та відгук (recall). Сітковий пошук дозволяє автоматизувати цей процес і знайти найкращі параметри на основі вказаних метрик.

Завдання 2.4: Обчислення відносної важливості ознак

Отримана діаграма:



Аналіз діаграми: Діаграма показує відносну важливість кожної ознаки для регресора AdaBoost, побудованого на наборі даних Boston Housing. Найбільш важливою ознакою є LSTAT (відсоток населення з низьким соціально-економічним статусом). Це означає, що ціна будинку в основному залежить від доходу населення в районі. Другою за важливістю ознакою є RM (кількість кімнат у будинку). Це означає, що більші будинки, як правило, дорожчі. Інші важливі ознаки включають CRIM (рівень злочинності), NOX (концентрація оксиду азоту), AGE (середній вік будівель) і DIS (відстань до центру міста). У цілому діаграма показує, що регресор AdaBoost вважає, що ціна будинку в основному залежить від соціально-економічного статусу населення, розміру будинку і рівня злочинності в районі.

Лістинг програми:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn import datasets, preprocessing
from sklearn.metrics import mean_squared_error, explained_variance_score
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle

# Завантаження даних з оригінального джерела
data_url = "http://lib.stat.cmu.edu/datasets/boston"
raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=None)
data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
target = raw_df.values[1::2, 2]

# Кодування міток
label_encoder = preprocessing.LabelEncoder()
y = label_encoder.fit_transform(target)

# Перемішування даних та розділення на навчальний і тестовий набори
X, y = shuffle(data, y, random_state=7)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=7)

# Створення та навчання моделі AdaBoostRegressor
regressor = AdaBoostClassifier(
    DecisionTreeClassifier(max_depth=4), n_estimators=400, random_state=7
)
regressor.fit(X_train, y_train)

# Передбачення та оцінка моделі
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 error =", round(evs, 2))

# Оцінка важливості ознак
feature_importances = regressor.feature_importances_
feature_names = [
    "CRIM",
    "ZN",
    "INDUS",
    "CHAS",
    "NOX",
    "RM",
```

```
    "AGE",
    "DIS",
    "RAD",
    "TAX",
    "PTRATIO",
    "B",
    "LSTAT",
]
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[i] for i in index_sorted])
plt.ylabel("Relative Importance")
plt.title("Оцінка важливості ознак для регресора AdaBoost")
plt.show()
```

Результат виконання програми:

```
PS C:\AI_labs> & E:/Users/madma/Ap

ADABOOST REGRESSOR
Mean squared error = 1970.62
Explained variance error = 0.49
PS C:\AI_labs>
```

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

Результат виконання програми:

```
PS C:\AI_labs\lab4> & E:/Users
Mean absolute error = 5.57
Predicted traffic: 24
PS C:\AI_labs\lab4>
```

Лістинг програми:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, mean_absolute_error
from sklearn import preprocessing
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.model_selection import train_test_split

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": 200, "max_depth": 15, "random_state": 0}
regressor = ExtraTreesClassifier(**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:
        encoder = label_encoder[count]
        test_datapoint_encoded[i] = int(encoder.transform([test_datapoint[i]])[0])
        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)

Результат виконання програми:

```
PS C:\AI_labs> & E:/Users/madma/AppData/Local/Microsoft/WindowsApps/python3.1

Predicted output:
 [0 2 2 0 2 0 2 1 0 1 1 2 0 0 2 2 1 0 0 1 0 2 1 1 2 2 0 0 1 2 1 2 1 0 2 2 1
 1 2 2 2 0 1 2 2 1 2 2 1 0 1 2 2 1 2 0 2 2 0 2 2 0 1 0 2 2 0 1 1 2 0 1 0 2
 0 0 1 1 2 0 0 1 2 2 2 0 0 0 2 2 2 2 2 0 2 1 2 2 0 0 1 1 1 1 2 2 2 2 0 1 1
 0 2 1 0 0 1 1 1 1 0 0 0 1 2 1 0 0 2 1 2 0 0 1 0 1 1 0 1 1 1 1 0 2 1 1 2 0
 2 2]

Score: 0.8733333333333333

Indices of selected features: 4, 7, 8, 12, 14, 17, 22
PS C:\AI_labs>
```

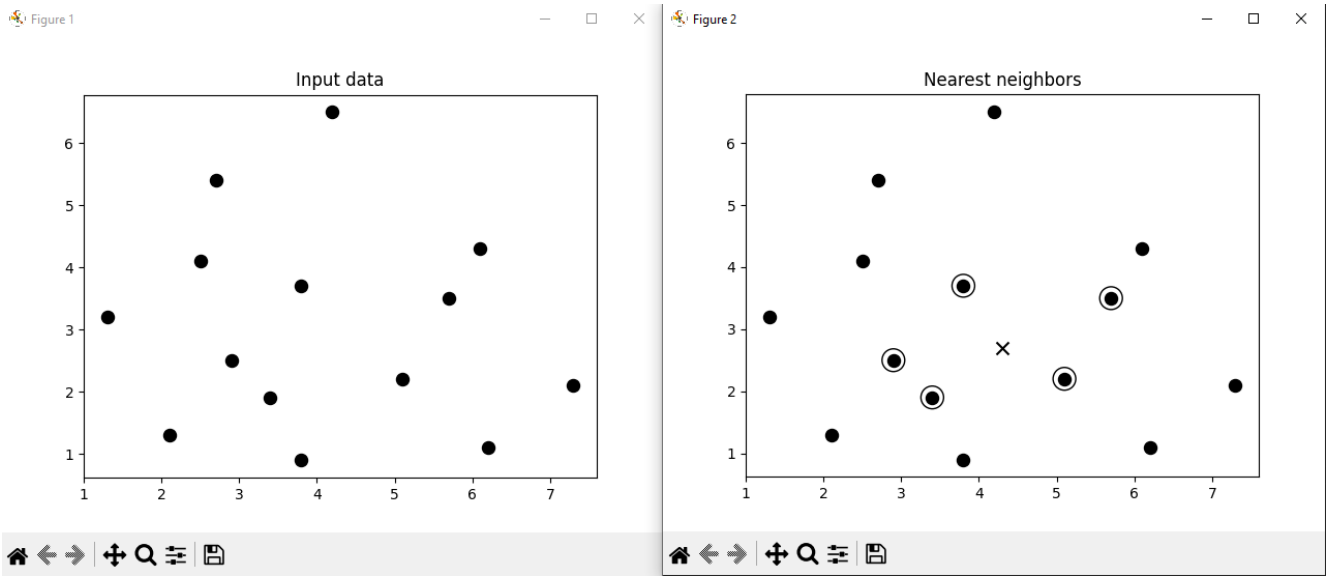
Висновок:

1. У першому списку з назвою "Predicted output" міститься прогнозовані класи для кожного прикладу в навчальному наборі даних X. Кожен елемент цього списку є передбаченою міткою класу для відповідного прикладу в X. Наприклад, перший елемент у списку вказує на передбачений клас для першого прикладу у X, другий елемент - для другого прикладу, і так далі.
2. Значення "Score" (0.8733333333333333) є оцінкою точності моделі на навчальних даних. В даному випадку точність визначає, наскільки добре модель класифікує дані в навчальному наборі. Це число вказує на відсоток правильно класифікованих прикладів. У цьому випадку, приблизно 87.33% прикладів було правильно класифіковано моделлю.

3. Останній рядок виводу вказує на індекси обраних ознак. Ці індекси відповідають ознакам в початковому наборі даних `X`. В даному випадку, вибір ознак за допомогою `SelectKBest` відібрав 7 ознак з початкового набору ознак і вивів їхні індекси, які вказують на те, які саме ознаки були вибрані для використання в моделі. Індекси 4, 7, 8, 12, 14, 17 і 22 відповідають обраним ознакам.

Завдання 2.7: Пошук найближчих сусідів

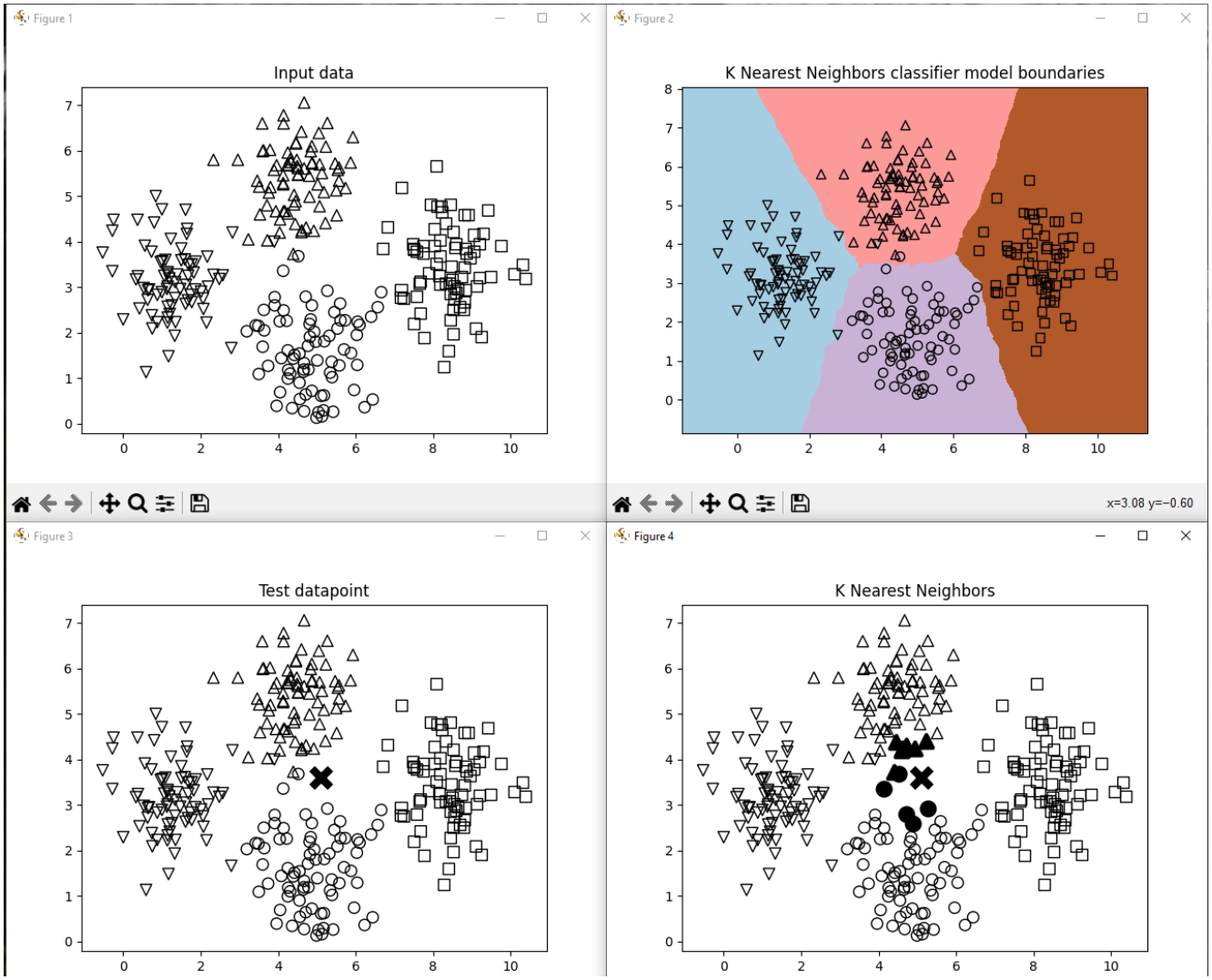
Отримані графіки:



Висновок: На першому графіку відображено вхідні дані для методу k найближчих сусідів. Дані представлені в двовимірному просторі, де кожна точка представляє один об'єкт. На другому графіку відображено найближчі сусіди тестової точки даних, а також сама тестова точка даних. Тестова точка даних позначена хрестиком, а найближчі сусіди позначені великими чорними точками. У консолі відображається виведення функції print(). Ця функція виводить на консоль список з п'яти найближчих сусідів тестової точки даних. Кожен сусід представляється як кортеж з двох чисел, які представляють координати точки в двовимірному просторі.

Завдання 2.8: Створити класифікатор методом k найближчих сусідів

Отримані графіки:



Опис графіків:

- **Figure 1:** Вхідні дані, представлені як точкова діаграма. Кожен маркер представляє один навчальний приклад, а його колір відповідає класу, до якого він належить.
- **Figure 2:** Межі класифікатора k найближчих сусідів. Межі показані як кольорова область, причому різні кольори представляють різні класи.
- **Figure 3:** Тестова точка даних, представлена як хрест.
- **Figure 4:** Найближчі 12 сусідів тестової точки даних. Найближчі сусіди показані як маркери, причому їх форма відповідає класу, до якого вони належать.

Висновок: Цей код демонструє використання методу k найближчих сусідів (K-Nearest Neighbors, KNN) для класифікації точок даних в двовимірному просторі.

Завдання 2.9: Обчислення оцінок подібності

Результати виконання програми:

```
PS C:\AI_labs\lab4> python3 LR_4_task_9.py --user1 "David Smith" --user2 "Bill Duffy" --score-type Euclidean

Euclidean score:
0.585786437626905
PS C:\AI_labs\lab4>
```

```
PS C:\AI_labs\lab4> python3 LR_4_task_9.py --user1 "David Smith" --user2 "Bill Duffy" --score-type Pearson

Pearson score:
0.9909924304103233
PS C:\AI_labs\lab4>
```

```
PS C:\AI_labs\lab4> python3 LR_4_task_9.py --user1 "David Smith" --user2 "Brenda Peterson" --score-type Pearson

Pearson score:
-0.7236759610155113
PS C:\AI_labs\lab4> python3 LR_4_task_9.py --user1 "David Smith" --user2 "Brenda Peterson" --score-type Euclidean

Euclidean score:
0.1424339656566283
PS C:\AI_labs\lab4>
```

```
PS C:\AI_labs\lab4> python3 LR_4_task_9.py --user1 "David Smith" --user2 "Samuel Miller" --score-type Euclidean

Euclidean score:
0.30383243470068705
PS C:\AI_labs\lab4> python3 LR_4_task_9.py --user1 "David Smith" --user2 "Samuel Miller" --score-type Pearson

Pearson score:
0.7587869106393281
PS C:\AI_labs\lab4>
```

```
PS C:\AI_labs\lab4> python3 LR_4_task_9.py --user1 "David Smith" --user2 "Julie Hammel" --score-type Pearson

Pearson score:
0
PS C:\AI_labs\lab4> python3 LR_4_task_9.py --user1 "David Smith" --user2 "Julie Hammel" --score-type Euclidean

Euclidean score:
0.2857142857142857
PS C:\AI_labs\lab4>
```

```
PS C:\AI_labs\lab4> python3 LR_4_task_9.py --user1 "David Smith" --user2 "Clarissa Jackson" --score-type Euclidean

Euclidean score:
0.28989794855663564
PS C:\AI_labs\lab4> python3 LR_4_task_9.py --user1 "David Smith" --user2 "Clarissa Jackson" --score-type Pearson

Pearson score:
0.6944217062199275
PS C:\AI_labs\lab4>
```

```
PS C:\AI_labs\lab4> python3 LR_4_task_9.py --user1 "David Smith" --user2 "Adam Cohen" --score-type Pearson

Pearson score:
0.9081082718950217
PS C:\AI_labs\lab4> python3 LR_4_task_9.py --user1 "David Smith" --user2 "Adam Cohen" --score-type Euclidean

Euclidean score:
0.38742588672279304
PS C:\AI_labs\lab4>
```

```
PS C:\AI_labs\lab4> python3 LR_4_task_9.py --user1 "David Smith" --user2 "Chris Duncan" --score-type Euclidean

Euclidean score:
0.38742588672279304
PS C:\AI_labs\lab4> python3 LR_4_task_9.py --user1 "David Smith" --user2 "Chris Duncan" --score-type Pearson

Pearson score:
1.0
```

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 user")
    parser.add_argument(
        "--score-type",
        dest="score_type",
        required=True,
        choices=["Euclidean", "Pearson"],
        help="Similarity metric to be used",
    )
    return parser


# Обчислення оцінки евклідова відстані між користувачами user1 та 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")

    # Фільми, оцінені обома користувачами, user1 та user2
    common_movies = {}

    for item in dataset[user1]:
        if item in dataset[user2]:
            common_movies[item] = 1

    # За відсутності фільмів, оцінених обома користувачами, оцінка приймається рівною 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)))


# Обчислення кореляційної оцінки Пірсона між користувачем1 і користувачем2
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")

    # Фільми, оцінені обома користувачами, user1 та user2
    common_movies = {}

    for item in dataset[user1]:
        if item in dataset[user2]:
            common_movies[item] = 1

    num_ratings = len(common_movies)

    # За відсутності фільмів, оцінених обома користувачами, оцінка приймається рівною 0
    if num_ratings == 0:
        return 0

    # Обчислення суми рейтингових оцінок усіх фільмів, оцінених обома користувачами
    user1_sum = np.sum([dataset[user1][item] for item in common_movies])
    user2_sum = np.sum([dataset[user2][item] for item in common_movies])

    # Обчислення Суми квадратів рейтингових оцінок всіх фільмів, оцінених обома кори-стувачами
    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]
    )

    # Обчислення суми творів рейтингових оцінок всіх фільмів, оцінених обома користува-чами
    sum_of_products = np.sum(
        [dataset[user1][item] * dataset[user2][item] for item in common_movies]
    )

    # Обчислення коефіцієнта кореляції Пірсона
    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))
```

Висновок: Цей код є прикладом програми для обчислення подібності між двома користувачами, на їхніх рейтингових оцінках фільмів. Основна ідея полягає в тому, щоб визначити, наскільки схожі користувачі за їхніми оцінками фільмів, використовуючи дві різні метрики подібності: Евклідова відстань і кореляційний коефіцієнт Пірсона.

Завдання 2.10: Пошук користувачів зі схожими уподобаннями методом колаборативної фільтрації

Результат виконання програми:

```
PS C:\AI_labs\lab4> python3 LR_4_task_10.py  --user "Bill Duffy"

Users similar to Bill Duffy:

User                     Similarity score
-----------------------------------------
David Smith              0.99
Samuel Miller            0.88
Adam Cohen               0.86
PS C:\AI_labs\lab4>
```

```
PS C:\AI_labs\lab4> python3 LR_4_task_10.py  --user "Clarissa Jackson"

Users similar to Clarissa Jackson:

User                     Similarity score
-----------------------------------------
Chris Duncan             1.0
Bill Duffy               0.83
Samuel Miller            0.73
PS C:\AI_labs\lab4>
```

Лістинг програми:

```
import argparse
import json
import numpy as np

from LR_4_task_9 import pearson_score
```

```python
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


# Знаходження користувачів у наборі даних, схожих на введеного користувача
def find_similar_users(dataset, user, num_users):
    if user not in dataset:
        raise TypeError("Cannot find " + user + " in the dataset")

    # Обчислення оцінки подібності за Пірсоном між
    # вказаним користувачем та всіма іншими
    # користувачами в наборі даних
    scores = np.array(
        [[x, pearson_score(dataset, user, x)] for x in dataset if x != user]
    )

    # Сортування оцінок за спаданням
    scores_sorted = np.argsort(scores[:, 1])[::-1]

    # Вилучення оцінок перших 'num_users' користувачів
    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))
```

Висновок: Цей код реалізує метод колаборативної фільтрації для пошуку користувачів із схожими уподобаннями щодо фільмів. Він використовує кореляційний коефіцієнт Пірсона для обчислення подібності між користувачами на основі їхніх рейтингових оцінок. Цей метод може бути використаний для рекомендацій фільмів користувачам, які мають схожі уподобання, базуючись на рейтингах інших користувачів.

Завдання 2.11: Створення рекомендаційної системи фільмів

Результат виконання програми:

```
PS C:\AI_labs\lab4> python3 LR_4_task_11.py  --user "Chris Duncan"

Movie recommendations for Chris Duncan:
1. Vertigo
2. Scarface
3. Goodfellas
4. Roman Holiday
PS C:\AI_labs\lab4> python3 LR_4_task_11.py  --user "Julie Hammel"

Movie recommendations for Julie Hammel:
1. The Apartment
2. Vertigo
3. Raging Bull
PS C:\AI_labs\lab4>
```

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 the movie recommendations for the given user"
    )
    parser.add_argument("--user", dest="user", required=True, help="Input user")
    return parser


# Отримання рекомендації щодо фільмів для вказаного користувача
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"]

    # Генерація рейтингів фільмів за допомогою їх нормалізації
    movie_scores = np.array(
        [
            [score / similarity_scores[item], item]
            for item, score in overall_scores.items()
        ]
    )

    # Сортування за спаданням
    movie_scores = movie_scores[np.argsort(movie_scores[:, 0])[::-1]]

    # Вилучення рекомендацій фільмів
    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)
```

Висновок: Цей код розширює попередню систему, де були знайдені користувачі із схожими уподобаннями, і додає можливість знаходити рекомендації щодо фільмів для конкретного користувача. Основна ідея полягає в тому, щоб пропонувати користувачу фільми, які інші схожі користувачі високо оцінили, але яких він ще не дивився.