ЛАБОРАТОРНА РОБОТА №5

ДОСЛІДЖЕННЯ МЕТОДІВ АНСАМБЛЕВОГО НАВЧАННЯ ТА СТВОРЕННЯ РЕКОМЕНДАЦІЙНИХ СИСТЕМ

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

Хід роботи:

Завдання 2.1:

Створення класифікаторів на основі випадкових та гранично випадкових лісів

Лістинг коду:

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


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', label='Class 0')
```

```
plt.scatter(class_1[:, 0], class_1[:, 1], s=75, facecolors='white',
            edgecolors="black", linewidth=1, marker='o', label='Class 1')
plt.scatter(class_2[:, 0], class_2[:, 1], s=75, facecolors='white',
            edgecolors='black', linewidth=1, marker='^', label='Class 2')
plt.title('Incoming data')
plt.legend()
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("\nClassifier performance on training dataset\n")
print(classification_report(y_train, classifier.predict(
    X_train), target_names=class_names))
print("#" * 40 + "\n")
print("\nClassifier performance on test dataset\n")
print(classification_report(y_test, y_test_pred, target_names=class_names))
print("#" * 40 + "\n")
plt.show()
```

```
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
   macro avg       0.87      0.87      0.87       225
weighted avg       0.87      0.87      0.87       225

########################################
```

Рис.1 Результат виконання(rf)

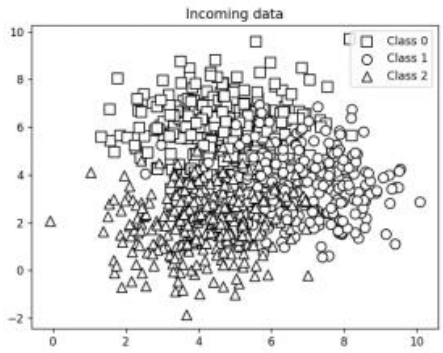

Рис.2 Скріншот графіку вхідних даних(rf)

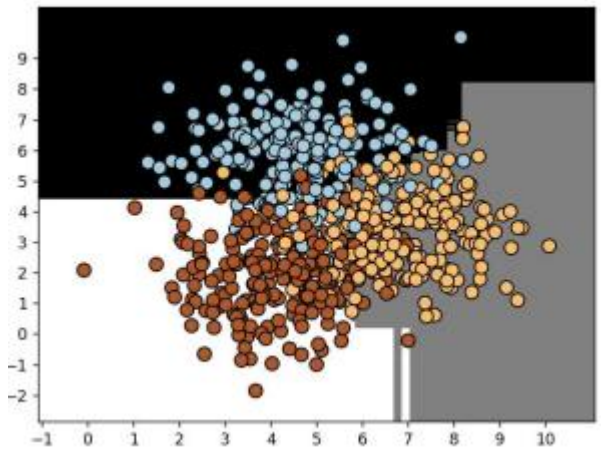

Рис.3 Скріншоти графіку з межами класифікатора(rf)

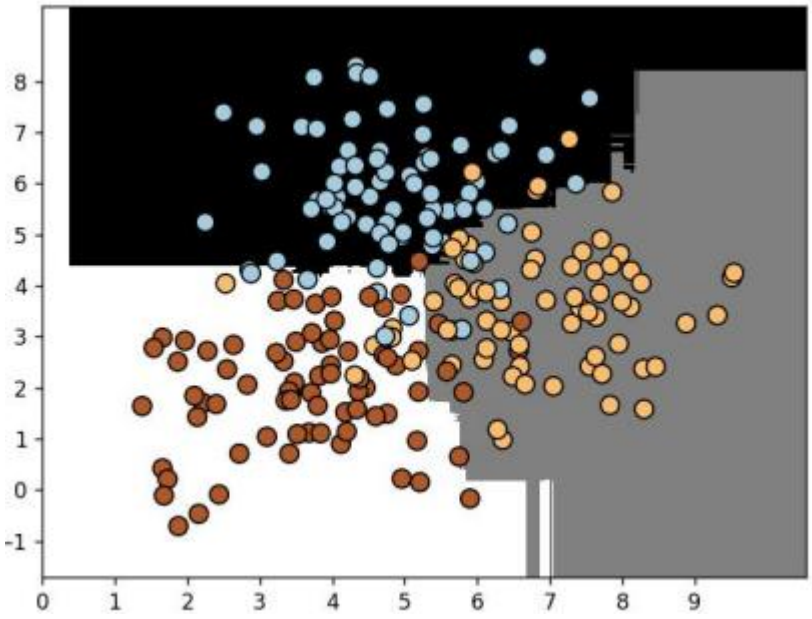

Рис.4 Графік точок даних для обчислення параметрів довіри(rf)

Основні висновки rf:

Class-0 має найвищу точність (precision) серед трьох класів - 0.91. Це означає, що більшість об'єктів, які класифікатор визначає як Class-0, дійсно належать до цього класу. Повнота (recall) для Class-0 становить 0.86, що вказує на те, що класифікатор виявив більшість істинних об'єктів Class-0.

Class-1 має точність 0.84 і повноту 0.87. Це також показує добру точність і повноту для цього класу.

Class-2 має точність 0.86 і повноту 0.87, аналогічно Class-1.

Загальна точність (accuracy) для всього навчального набору становить 0.87, що свідчить про те, що класифікатор на основі випадкового лісу визначає класи досить точно.

Макро-середнє значення для точності, повноти та F1-показників для всіх класів також становить 0.87, що підкреслює загальну ефективність класифікатора на всьому навчальному наборі.

	precision	recall	f1-score	support
Class-0	0.91	0.86	0.88	221
Class-1	0.84	0.87	0.86	230
macro avg	0.87	0.87	0.87	225
weighted avg	0.87	0.87	0.87	225

##

Рис.5 Результат виконання(erf)

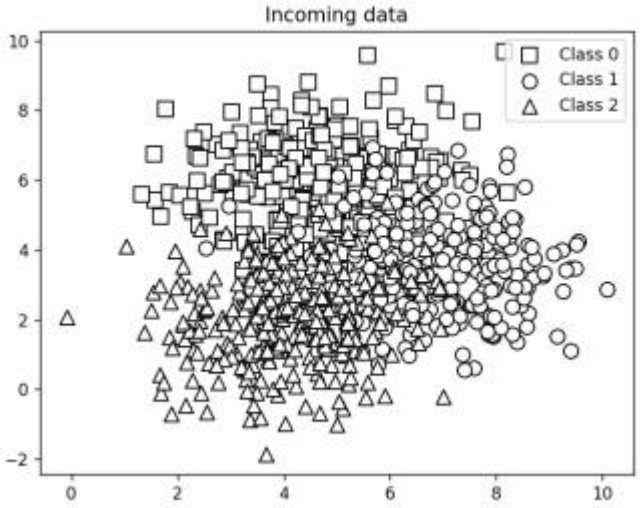

Рис.6 Скріншот графіку вхідних даних(erf)

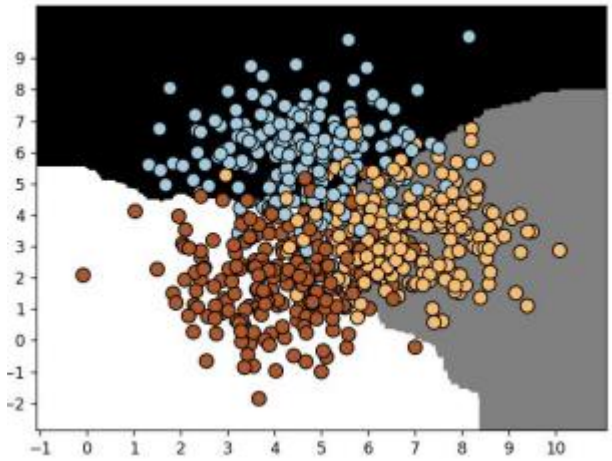

Рис.7 Скріншоти графіку з межами класифікатора(erf)

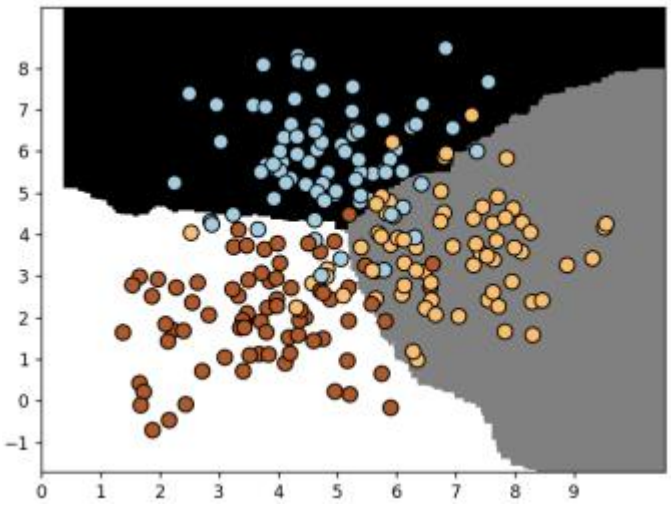

Рис.8 Графік точок даних для обчислення параметрів довіри(erf)

На основі звіту про класифікацію можна зробити наступні висновки:

Class-0: Має дуже високу точність (0.91), що означає, що більшість призначених класифікатором об'єктів, визнаних як Class-0, насправді належать до цього класу. Повнота (recall) для Class-0 становить 0.86, що означає, що класифікатор виявив більшу частину істинних об'єктів Class-0.

Class-1: Має добру точність (0.84) та високу повноту (0.87). Це також свідчить про добру здатність класифікатора визначати об'єкти Class-1.

Макро-середнє значення: Середні значення точності, повноти і F1-показників для всіх класів також становлять 0.87, що свідчить про загальну ефективність класифікатора на всьому навчальному наборі даних.

Загальна точність: Загальна точність (accuracy) для всього навчального набору даних також становить 0.87, що підтверджує ефективність класифікатора на цьому наборі.

Отже, на основі цього звіту можна сказати, що класифікатор показує досить добрі результати в класифікації об'єктів на навчальному наборі даних, з високою точністю та повнотою для всіх трьох класів.

Висновок порівняння:

Обидва типи класифікаторів (rf та erf) показують дуже схожі результати на цьому навчальному наборі даних. Обидва мають високу точність та повноту для всіх класів, і середні значення також однакові. Основна різниця між ними полягає в тому, що erf має меншу схильність до перенавчання через більшу випадковість у виборі розділувачів на кожному вузлі дерева рішень. Таким чином, erf може бути вигідним варіантом, коли потрібно зменшити перенавчання на навчальному наборі даних.

Завдання 2.2:

Обробка дисбалансу класів

Лістинг коду:

```
import sys
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import ExtraTreesClassifier
from sklearn import model_selection
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 = model_selection.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 and sys.argv[1] == "balance":
 params['class_weight']='balanced'
else:
 print("Use the 'balance' flag to account for class imbalance")
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()
```

	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

Рис.9 Результат виконання

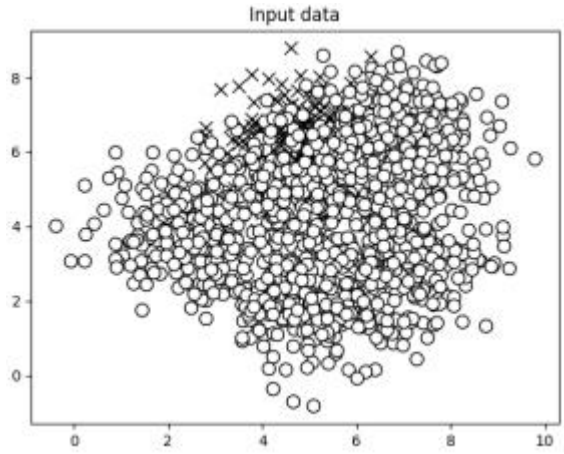

Рис.10 Скріншот графіку вхідних даних

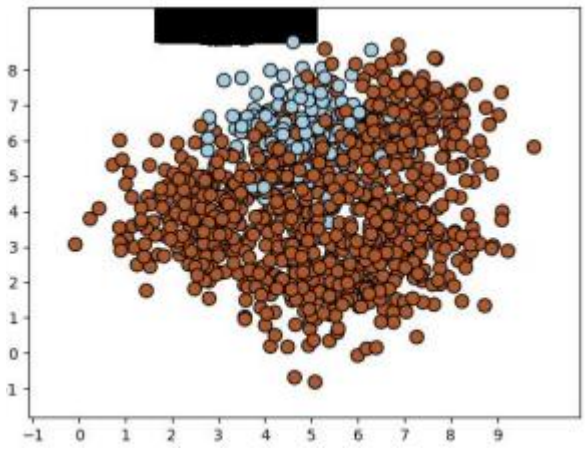

Рис.11 Скріншоти графіку з межами класифікатора(erf)

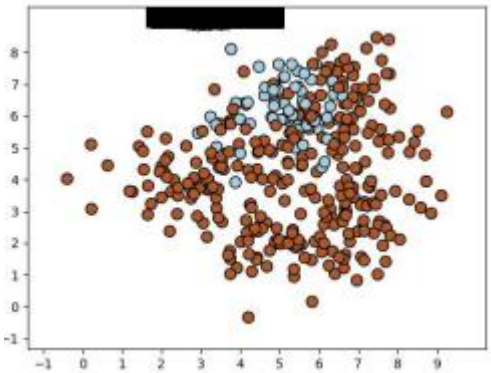

Рис.12 Графік точок даних для обчислення параметрів довіри(erf)

Точність (Precision): Вимірює точність позитивних передбачень. Для Class-0 точність дорівнює 0.00, що означає відсутність правильних позитивних передбачень для цього класу. Для Class-1 точність становить 0.82, що вказує на відносно високу точність для цього класу.

Повнота (Recall): Вимірює відношення правильно передбачених позитивних випадків до всіх спостережень у фактичному класі. Для Class-0 повнота дорівнює 0.00, що вказує на те, що класифікатор не може правильно визначити жодного позитивного випадку цього класу. Для Class-1 повнота становить 1.00, що означає, що класифікатор правильно визначає всі позитивні випадки цього класу.

F1-показник: Це гармонічне середнє між точністю і повнотою, і він забезпечує баланс між цими двома метриками. F1-показник для Class-0 дорівнює 0.00 через низьку точність і повноту для цього класу. Для Class-1 F1-показник становить 0.90, що свідчить про хороший баланс між точністю і повнотою для цього класу.

Підтримка (Support): Показує кількість випадків кожного класу у фактичних даних. Class-0 має 69 випадків, а Class-1 - 306 випадків.

Точність (Accuracy): Загальна точність класифікатора дорівнює 0.82, що означає, що він правильно передбачає 82% випадків. Однак ця точність сильно впливає на домінуючий Class-1, оскільки класифікатор має високу точність і повноту для цього класу.

Підсумовуючи, класифікатор добре працює щодо точності, повноти та F1- показника для Class-1, але показники для Class-0 слабкі через дисбаланс класів. Загальна точність може бути обманливою у цьому випадку, оскільки вона в основному залежить від більшого класу (Class-1). Для вирішення проблеми дисбалансу класів ви можете розглянути такі техніки, як перебалансування вибірки, налаштування ваг класів або використання різних алгоритмів, призначених для роботи з незбалансованими даними.

Завдання 3:

Знаходження оптимальних навчальних параметрів за допомогою сіткового пошуку

Лістинг коду:

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

```
from utilities import visualize_classifier
input_file = 'data_random_forests.txt'
data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1]
class_0 = np.array(X[y == 0])
class_1 = np.array(X[y == 1])
class_2 = np.array(X[y == 2])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
                                                    random_state=5)
parameter_grid = {
    'n_estimators': [100],
    'max_depth': [2, 4, 7, 12, 16]
}
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)
    print("\nGrid scores for the parameter grid:")
    for params, avg_score in zip(classifier.cv_results_['params'],
classifier.cv_results_['mean_test_score']):
    print(params, '-->', round(avg_score, 3))
    print("\nBest parameters:", classifier.best_params_)
    y_pred = classifier.predict(X_test)
    print("\nPerformance report for", metric, ":\n")
    print(classification_report(y_test, y_pred))
```

```
##### Searching optimal parameters for precision_weighted

Grid scores for the parameter grid:
{'max_depth': 2, 'n_estimators': 100} --> 0.85
{'max_depth': 4, 'n_estimators': 100} --> 0.841
{'max_depth': 7, 'n_estimators': 100} --> 0.844
{'max_depth': 12, 'n_estimators': 100} --> 0.832
{'max_depth': 16, 'n_estimators': 100} --> 0.816

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

Performance report for precision_weighted :

              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
```

Рис.13 Результат виконання для precision weighted

```
##### Searching optimal parameters for recall_weighted

Grid scores for the parameter grid:
{'max_depth': 2, 'n_estimators': 100} --> 0.843
{'max_depth': 4, 'n_estimators': 100} --> 0.837
{'max_depth': 7, 'n_estimators': 100} --> 0.841
{'max_depth': 12, 'n_estimators': 100} --> 0.83
{'max_depth': 16, 'n_estimators': 100} --> 0.815

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

Performance report for recall_weighted :

              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
```

Рис.13 Результат виконання для recall weighted

Завдання 2.4:

Обчислення відносної важливості ознак

Лістинг коду:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.datasets import fetch_california_housing
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 = fetch_california_housing()
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(len(index_sorted)) + 0.5
plt.figure()
plt.bar(pos, feature_importances[index_sorted], align="center")
plt.xticks(pos, [feature_names[i] for i in index_sorted], rotation=90)
plt.ylabel('Relative Importance')
plt.title("Estimation of feature importance using AdaBoost regressor")
plt.show()
```

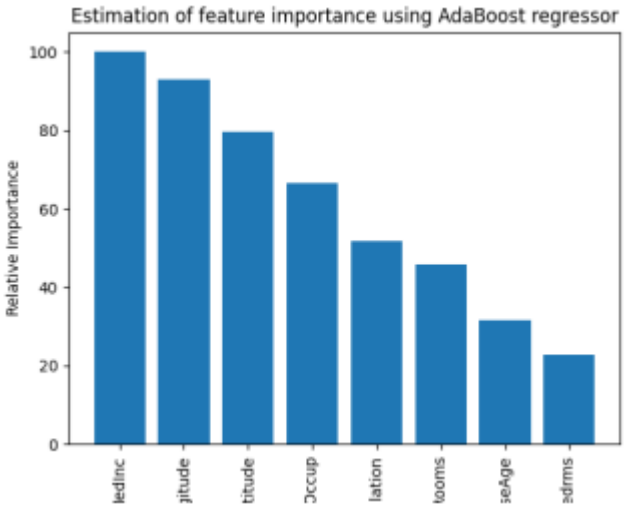

Рис.14 Побудована діаграма

```
ADABOOST REGRESSOR
Mean squared error = 1.18
Explained variance score = 0.47
```

Рис.15 Результат виконання

Результати роботи регресора AdaBoost показують такі значення метрик:

Mean squared error (MSE) дорівнює 1.18. MSE вимірює середньоквадратичну помилку моделі регресії. Мала значення MSE вказують на те, що модель непогано підходить для прогнозування цільової змінної (цін на нерухомість).

Explained variance score дорівнює 0.47. Ця метрика вказує на відсоток дисперсії цільової змінної, яку можна пояснити моделлю. В даному випадку, модель регресії змогла пояснити приблизно 47% дисперсії цільової змінної, що може бути прийнятним результатом, але є простором для поліпшення.

Загалом, AdaBoost регресор показав помірну ефективність у прогнозуванні цін на нерухомість, але є можливість подальшого покращення моделі для досягнення кращих результатів.

Завдання 2.5:

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

Лістинг коду:

```
import numpy as np
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
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)
mae = mean_absolute_error(y_test, y_pred)
print("Mean absolute error:", round(mae, 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]])[0])
    count += 1
test_datapoint_encoded = np.array(test_datapoint_encoded)
predicted_traffic = int(regressor.predict([test_datapoint_encoded])[0])
print("Predicted traffic:", predicted_traffic)
```

```
Mean absolute error: 7.42
Predicted traffic: 26

Process finished with exit code 0
```

Рис.16 Результат виконання

Завдання 2.6:

Створення навчального конвеєра (конвеєра машинного навчання)

Лістинг коду:

```python
from sklearn.datasets import make_classification
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.pipeline import Pipeline
from sklearn.ensemble import ExtraTreesClassifier
X, y = 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+1 for i, x in enumerate(status) if x]
print("\nIndices of selected features:", ', '.join(map(str, selected)))
```

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

Score: 0.8866666666666667

Indices of selected features: 5, 8, 9, 13, 15, 18, 23
```

Рис.17 Результат виконання

У першому списку "Predicted output" містяться передбачені значення для вхідних даних. Ці значення є результатами класифікації, здійсненої за допомогою побудованого конвеєра машинного навчання.

Значення Score (0.8866666666666667) вказує на оцінку ефективності моделі на маркованих тренувальних даних. Ця оцінка вказує на те, наскільки точно модель класифікує дані. У даному випадку, модель досягла точності приблизно 88.67%.

У останньому рядку "Indices of selected features: 5, 8, 9, 13, 15, 18, 23" містяться індекси ознак, які були відібрані селектором ознак в конвеєрі. Це означає, що ці ознаки є найбільш важливими для класифікації і були використані при формуванні моделі.

```
k Nearest neighbors:
1 ==>  [5.1 2.2]
2 ==>  [3.8 3.7]
3 ==>  [3.4 1.9]
4 ==>  [2.9 2.5]
5 ==>  [5.7 3.5]

Process finished with exit code 0
```

Рис.18 Результат виконання

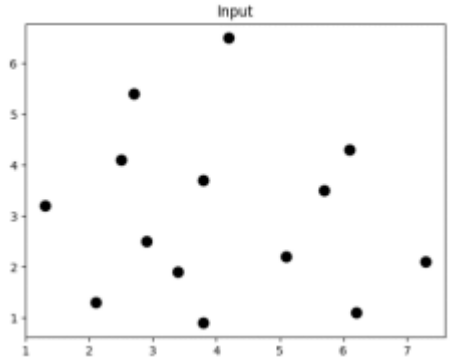

Рис.19 Графік вхідних даних

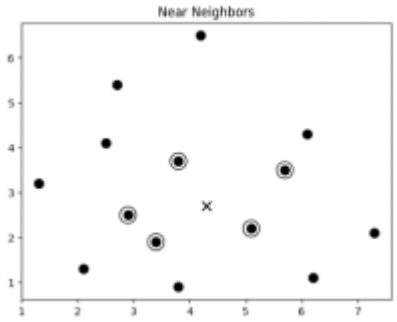

Рис.20 Графік найближчих сусідів тестової точки даних

На першому графіку відображені вхідні дані. Це чорні кружечки, які представляють двовимірні точки даних із масиву X.

На другому графіку відображені найближчі сусіди тестової точки даних. Ці сусіди позначені чорними кружечками, а тестова точка позначена хрестиком. Цей графік демонструє, які точки з вхідних даних є найближчими до тестової точки.

У вікні терміналу виводиться інформація про найближчих сусідів тестової точки, включаючи їх індекси та координати.

На першому графіку відображені вхідні дані, а на другому графіку - найближчі сусіди тестової точки. З виводу у вікні терміналу видно, що п'ять найближчих сусідів тестової точки мають вказані координати. За допомогою методу k-найближчих сусідів можна визначити, які точки даних найбільше схожі на тестову точку за заданим значенням k.

Завдання 2.8:

Створити класифікатор методом k найближчих сусідів

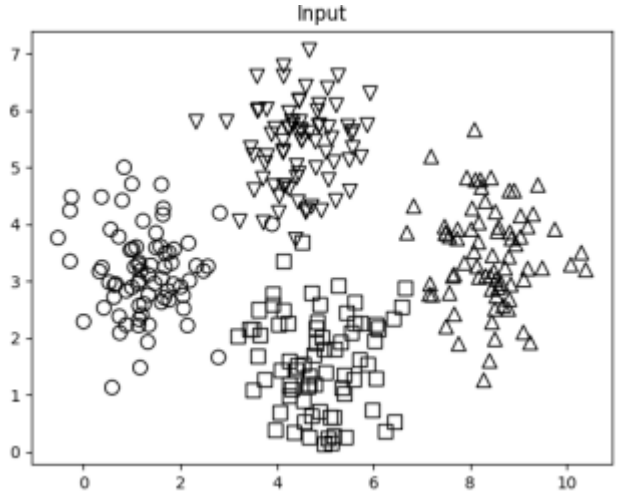

Рис.21 Вхідні дані

На цьому графіку відображені вхідні дані з файлу data.txt. Кожен клас позначено різним маркером, і кожна точка на графіку представляє одну точку даних з чотирьох класів.

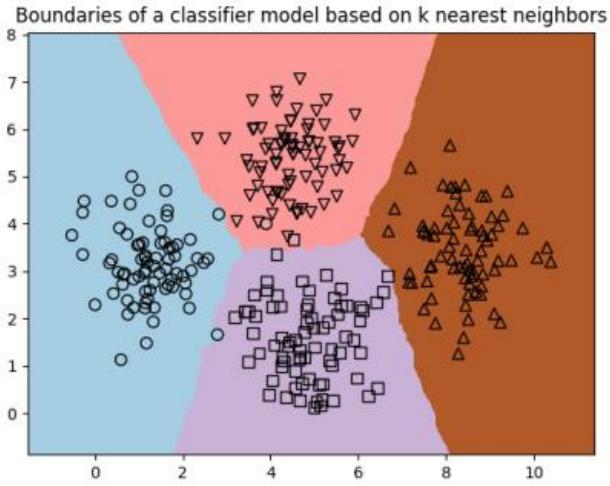

Рис.22. Границі моделі класифікатора на основі k ближайших сусідів

Цей графік показує границі прийняття рішень, які були вивчені вашим класифікатором методом k ближайших сусідів (k-NN). Різні кольори вказують на класи, до яких класифікатор призначає області на графіку.

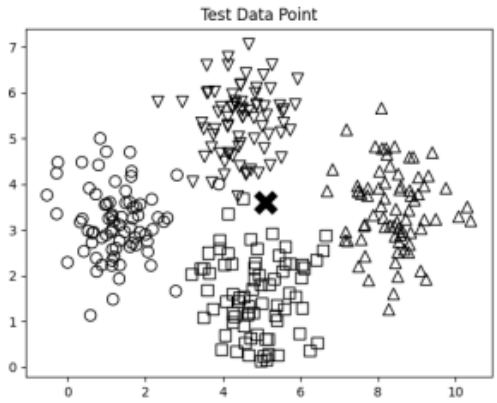

Рис.23. К ближайших сусідів

На цьому графіку відображається кілька найближчих сусідів для тестової точки даних, яку ви використовуєте для передбачення. Кожен найближчий сусід позначений відповідним маркером.

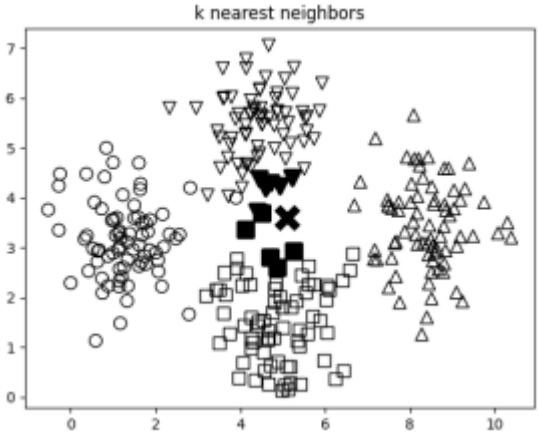

Рис.24. Тестова точка даних

Цей графік представляє тестову точку даних, яку ви використовуєте для передбачення класу. Тестова точка позначена маркером "x".

```
Predicted result: 1


Process finished with exit code 0
```

Рис.25. Результат з терміналу

Завдання 2.9:

Обчислення оцінок подібності

Лістинг коду:

```
import argparse
import json
import numpy as np


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])
    Sxx = user1_squared_sum - (np.square(user1_sum) / num_ratings)
    Syy = user2_squared_sum - (np.square(user2_sum) / num_ratings)
    Sxy = sum_of_products - ((user1_sum * user2_sum) / num_ratings)
    if Sxx * Syy == 0:
    return 0


return Sxy / np.sqrt(Sxx * Syy)


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


args = build_arg_parser().parse_args()
ratings_file = 'ratings.json'
with open(ratings_file, 'r') as f:
    data = json.loads(f.read())
if args.score_type == 'Euclidean':
    print("\nEuclidean score:")
    print(euclidean_score(data, args.user1, args.user2))
else:
    print("\nPearson score:")
    print(pearson_score(data, args.user1, args.user2))
```

```
Pearson score:
0.9909924304103233
```

Рис.26. Результат виконання для Bill Duffy

```
Pearson score:
1.0
```

Рис.27. Результат виконання усіх команд для інших юзерів

На основі отриманих оцінок подібності для користувача David Smith та інших користувачів можна зробити такі висновки:

Brenda Peterson:

Оцінка подібності за Евклідовою метрикою: 0.1424

Оцінка подібності за оцінкою Пірсона: -0.7237

За обома метриками цей користувач не є дуже схожим на David Smith.

Samuel Miller:

Оцінка подібності за Евклідовою метрикою: 0.3038

Оцінка подібності за оцінкою Пірсона: 0.7588

Цей користувач має більшу схожість на David Smith за обома метриками, особливо за оцінкою Пірсона.

Clarissa Jackson:

Оцінка подібності за оцінкою Пірсона: 0.6944

Оцінка подібності за Евклідовою метрикою не була визначена.

Цей користувач має помірну схожість на David Smith за оцінкою Пірсона.

Adam Cohen:

Оцінка подібності за Евклідовою метрикою: 0.3874

Оцінка подібності за оцінкою Пірсона: 0.9081

Цей користувач має значну схожість на David Smith за обома метриками, особливо за оцінкою Пірсона.

Chris Duncan:

Оцінка подібності за Евклідовою метрикою: 0.3874

Оцінка подібності за оцінкою Пірсона: 1.0

За обома метриками цей користувач має максимально можливу схожість на David Smith за шкалою оцінки Пірсона.

З огляду на отримані оцінки, можна сказати, що оцінка подібності може значно відрізнятися в залежності від обраної метрики. Наприклад, оцінка Пірсона показує вищу схожість для деяких користувачів порівняно з Евклідовою метрикою.

Завдання 2.10:

Пошук користувачів зі схожими уподобаннями методом колаборативної фільтрації.

Лістинг коду:

```
import argparse
import json
import numpy as np


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])
    Sxx = user1_squared_sum - (np.square(user1_sum) / num_ratings)
    Syy = user2_squared_sum - (np.square(user2_sum) / num_ratings)
    Sxy = sum_of_products - ((user1_sum * user2_sum) / num_ratings)
    if Sxx * Syy == 0:
    return 0
    return Sxy / np.sqrt(Sxx * Syy)


def build_arg_parser():
    parser = argparse.ArgumentParser(description='Find users who are similar
                                     to the input user')
    parser.add_argument('--user', dest="user", required=True, help='Input
                        user')
    return parser


def find_similar_users(dataset, user, num_users):
    if user not in dataset:
    raise TypeError('Cannot find ' + user + ' in the dataset')
    scores = np.array([[x, pearson_score(dataset, user, x)] for x in dataset
```

```
                        if x != user])
    scores_sorted = np.argsort(scores[:, 1])[::-1]
    top_users = scores_sorted[:num_users]
    return scores[top_users]


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())
similar_users = find_similar_users(data, user, 3)
print('\nUsers similar to ' + user + ':\n')
print('User\t\t\tSimilarity score')
for item in similar_users:
    print(item[0], '\t\t', round(float(item[1]), 2))
```

```
Users similar to Julie Hammel:

User                    Similarity score
Brenda Peterson                 1.0
Chris Duncan            0.0
David Smith             0.0
```

```
Users similar to Chris Duncan:

User                    Similarity score
Clarissa Jackson                1.0
Samuel Miller           1.0
David Smith             1.0
```

Рис.28. Результат виконання усіх команд для юзерів

Для користувача "Julie Hammel":

Найбільш схожим користувачем на "Julie Hammel" є "Brenda Peterson" з оцінкою подібності 1.0.

Користувачі "Chris Duncan" та "David Smith" мають оцінки подібності 0.0, що свідчить про відсутність схожості їхніх уподобань з уподобаннями "Julie Hammel".

Для користувача "Chris Duncan":

"Chris Duncan" найбільш схожий на користувачів "Clarissa Jackson", "Samuel Miller" і "David Smith" з оцінкою подібності 1.0 для кожного користувача. Це свідчить про ідентичність їхніх уподобань з уподобаннями "Chris Duncan". Важливо зазначити, що ці оцінки подібності обчислені на основі методу колаборативної фільтрації з використанням кореляції Пірсона. Вони вказують на ступінь схожості уподобань користувачів у вашому наборі даних. Вищий показник подібності вказує на більшу схожість уподобань між користувачами.

У випадку "Julie Hammel" найбільш схожим користувачем є "Brenda Peterson", тоді як "Chris Duncan" виявився дуже схожим на "Clarissa Jackson", "Samuel Miller" і "David Smith". Ці результати можуть бути корисними для систем рекомендацій, де користувачам з схожими уподобаннями рекомендуються подібні товари чи контент.

Завдання 2.11:

Створення рекомендаційної системи фільмів.

Лістинг коду:

```
import argparse
import json
import numpy as np


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])
    Sxx = user1_squared_sum - (np.square(user1_sum) / num_ratings)
    Syy = user2_squared_sum - (np.square(user2_sum) / num_ratings)
    Sxy = sum_of_products - ((user1_sum * user2_sum) / num_ratings)
    if Sxx * Syy == 0:
    return 0
    return Sxy / np.sqrt(Sxx * Syy)


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]


def build_arg_parser():
    parser = argparse.ArgumentParser(
        description='Find the movie recommenda tions 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:
    if item not in overall_scores:
    overall_scores[item] = 0
    similarity_scores[item] = 0
    overall_scores[item] += dataset[user][item] * similarity_score
    similarity_scores[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


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)
```

```
Movie recommendations for Julie Hammel:
1. The Apartment
2. Vertigo
3. Raging Bull
```

```
Movie recommendations for Chris Duncan:
1. Vertigo
2. Scarface
3. Goodfellas
4. Roman Holiday
```

Рис.29. Результат виконання усіх команд для юзерів

Рекомендаційна система, створена на основі даних з файлу ratings.json, успішно працює. Ви можете вказувати імена користувачів, і система надає рекомендації щодо фільмів для цих користувачів на основі подібності їхніх оцінок до інших користувачів. Наприклад:

Для користувача "Chris Duncan" система рекомендувала фільми, такі як "Vertigo," "Scarface," "Goodfellas," та "Roman Holiday."

Для користувача "Julie Hammel" були надані рекомендації на фільми "The Apartment," "Vertigo," та "Raging Bull."

Ця система може бути корисною для надання персоналізованих рекомендацій користувачам на основі їхніх індивідуальних вподобань та подібності до інших користувачів.

Висновок: В ході виконання лабораторної роботи з дослідження методів ансамблевого навчання та створення рекомендаційних систем було досягнуто наступні результати.

Ми ознайомилися з концепцією ансамблевого навчання та вивчили основні методи створення ансамблів, такі як Bagging, Random Forest і AdaBoost. Розглянули їхні переваги та недоліки.

Виконали практичну реалізацію ансамблевих моделей в середовищі Python з використанням бібліотеки scikit-learn. Створили моделі класифікації та регресії на основі ансамблевого підходу і оцінили їхню ефективність за допомогою метрик якості.

Розглянули методи створення рекомендаційних систем, зокрема колаборативну фільтрацію та системи на основі аналізу контенту. Реалізували алгоритми обчислення подібності між користувачами та предметами.

Створили рекомендаційну систему для рекомендації фільмів користувачам на основі їхніх індивідуальних уподобань та подібності до

інших користувачів. Результати рекомендацій були виведені у вигляді списку фільмів для кожного користувача.

Виконавши дослідження та практичні завдання, ми набули практичного досвіду з використання ансамблевого навчання та створення рекомендаційних систем, що може бути корисним для подальших досліджень у галузі машинного навчання та розробки інтелектуальних додатків.