ЛАБОРАТОРНА РОБОТА № 6

СТВОРЕННЯ РЕКОМЕНДАЦІЙНИХ СИСТЕМ

Мета роботи: використовуючи спеціалізовані бібліотеки та мову програмування Python навчитися створювати рекомендаційні системи.

Хід роботи

Завдання 2.1. Створення навчального конвеєра (конвеєра машинного навчання)

```
from sklearn.datasets import make_classification
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.pipeline import Pipeline
from sklearn.ensemble import ExtraTreesClassifier

# Generate data
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 for i, x in enumerate(status) if x]
print("\nIndices of selected features:", ', '.join([str(x) for x in selected]))
```

Рис. 1.1 Код програми

					Державний університет "Житомирська політехніка"		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Дроботун Д. Я.			Звіт з практичної роботи	Літ.	Арк. 1 / Аркушів
Перевір.							
Керівник						Гр. ІПЗК-19-1	
Н. контр.							
Зав. каф.							

```
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 0 2 2 0 0 1 2 1 2 1 0 2 2 1
 1 2 2 2 0 1 0 2 1 2 2 1 0 1 2 2 2 2 0 2 2 0 2 2 0 1 0 2 2 1 1 1 2 0 1 0 2
 0 0 1 2 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 0 2 0 1 1
 0 2 1 1 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 0 1 2 0
 2 2]

Score: 0.88

Indices of selected features: 4, 7, 8, 12, 14, 17, 22
```

Рис. 1.2 Результат виконання програми

Висновок: є 3 різних API для оцінки якості прогнозів моделі:

Параметр оцінки: інструменти оцінки моделі, що використовують перехресну перевірку (наприклад, model_selection.cross_val_score та model_selection.GridSearchCV) покладаються на внутрішню стратегію оцінки.

Метричні функції: модуль sklearn.metrics реалізує функції, що оцінюють помилку прогнозування для певних цілей.

Оцінювач метод оцінки : оцінювачі є **score метод** , який забезпечує критерій оцінки за умовчанням, для завдання вони призначені для його вирішення.

Завдання 2.2. Пошук найближчих сусідів

```python
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. 1.3 Код програми

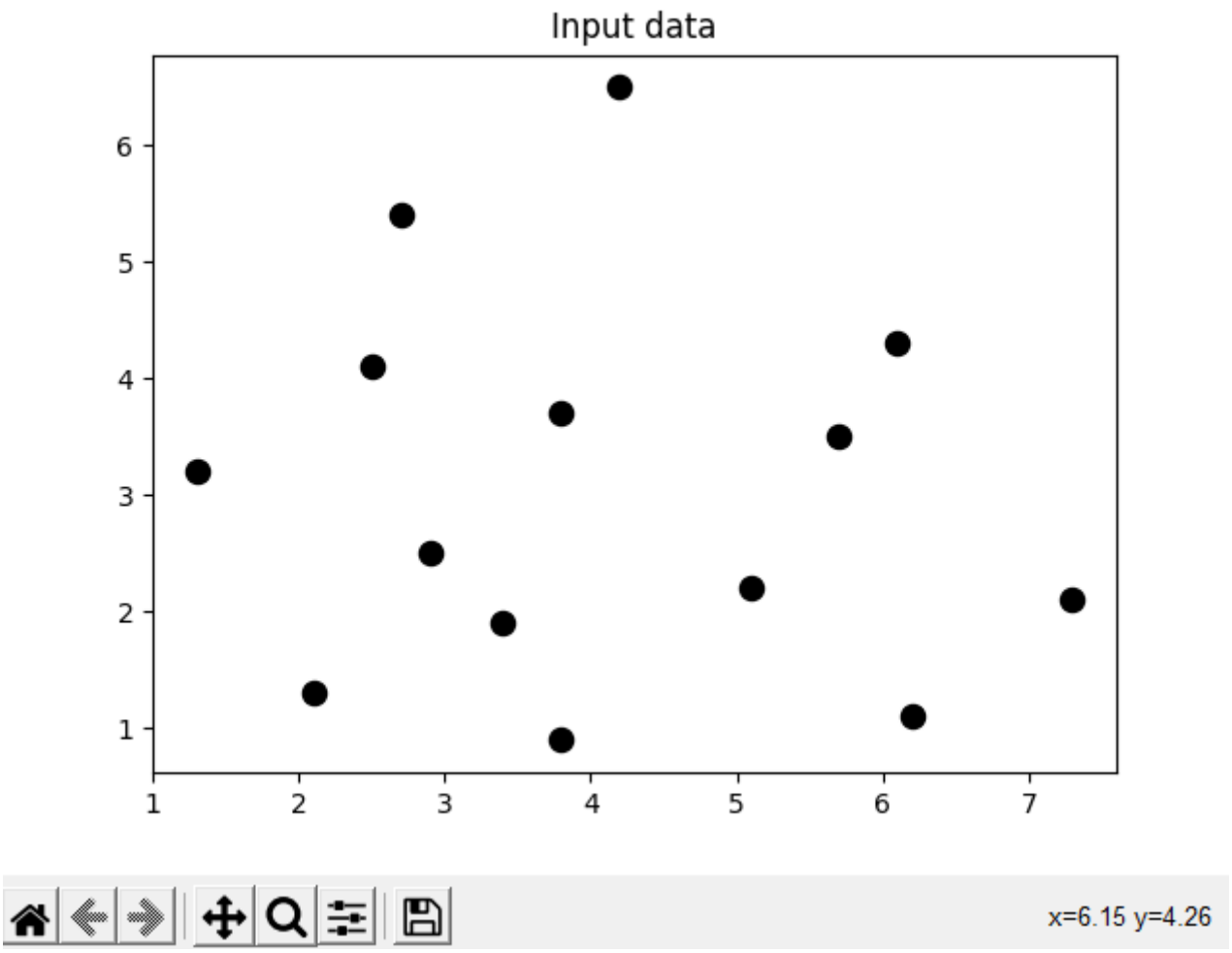

Рис. 1.4 Результат виконання програми

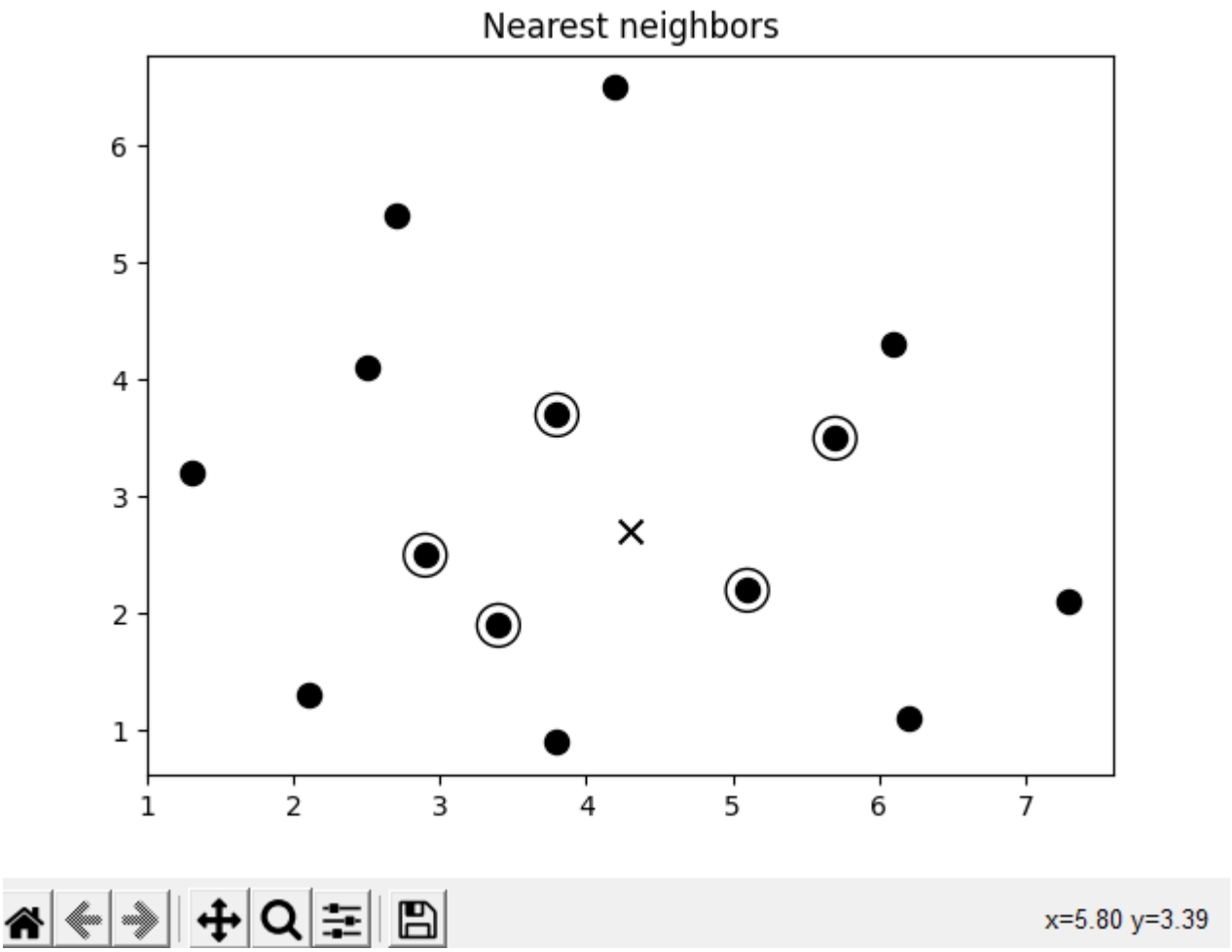

Рис. 1.5 Результат виконання програми

```
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]
```

Рис. 1.6 Результат виконання програми

Висновок: здійснюється пошук найближчих сусідів заданої точки даних. Nearest neighbours суть якого полягає у знаходженні тих точок заданого набору, які розташовані на найближчих відстанях від зазначеної. Такий підхід часто застосовується для створення систем, що класифікують точку даних на підставі її близькості до різних класів.

Завдання 2.3. Створити класифікатор методом k найближчих сусідів

```
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. 1.7 Код програми

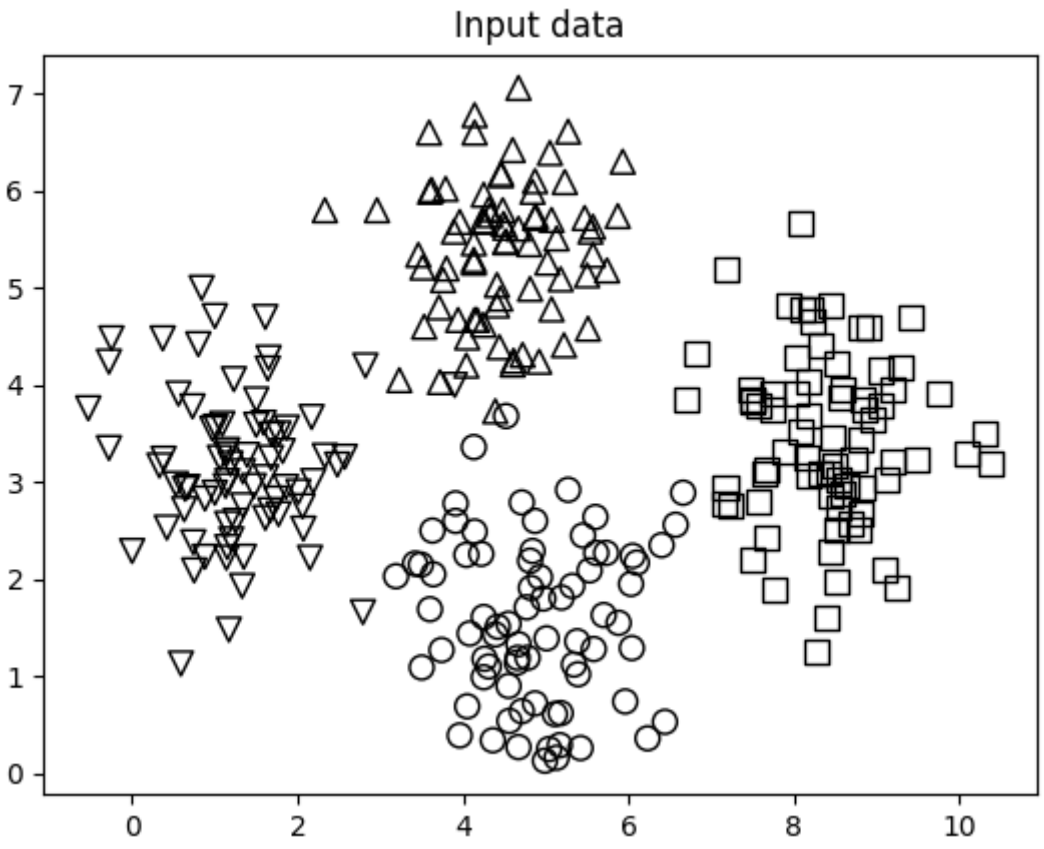

Рис. 1.8 Результат виконання програми

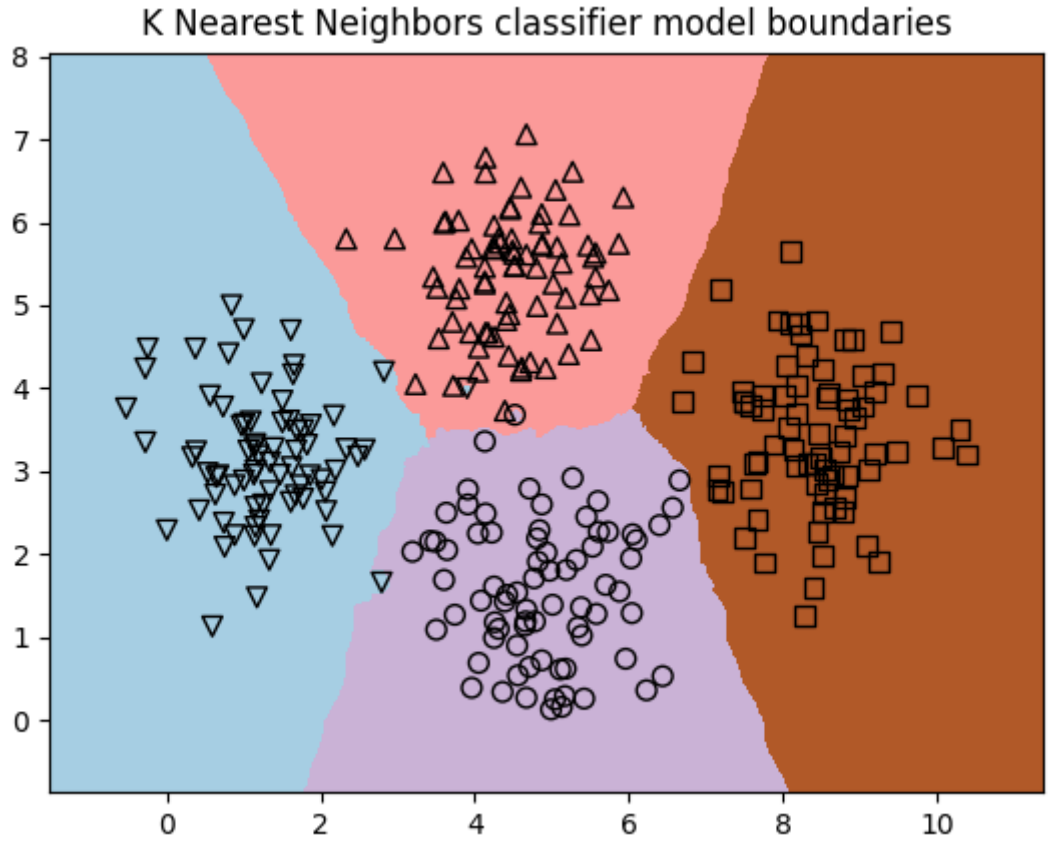

Рис. 1.9 Результат виконання програми

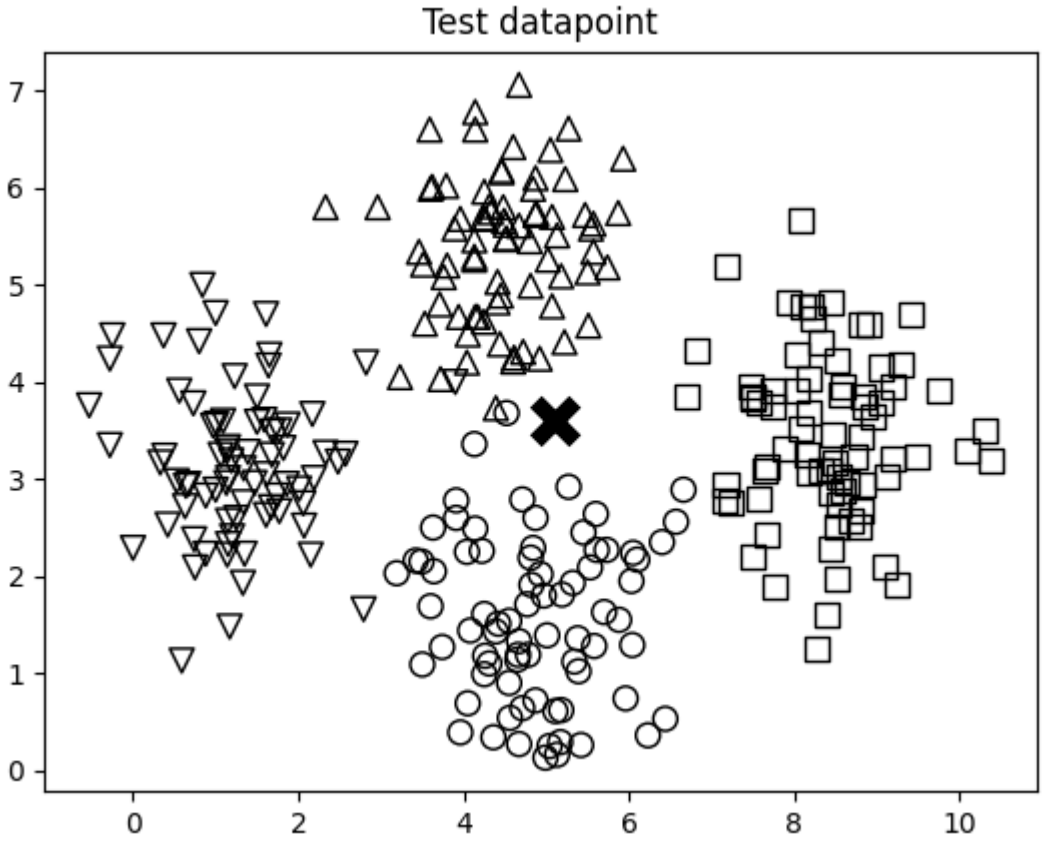

Рис. 2.1 Результат виконання програми

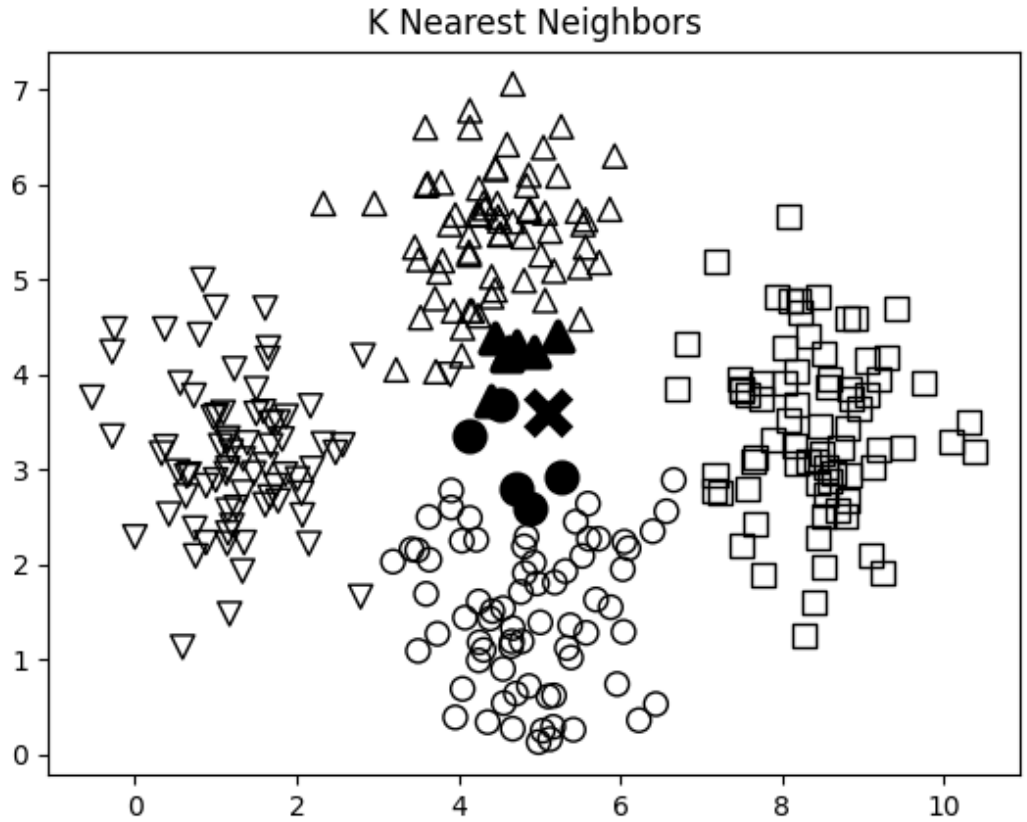

Рис. 2.2 Результат виконання програми

```
Predicted output: 1

Process finished with exit code 0
```

Рис. 2.3 Результат виконання програми

Завдання 2.4. Обчислення оцінок подібності

```
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))
```

Рис. 2.4 Код програми

```
PS E:\ЖДУ\Python\Work_Drobotun_6> python LR_6_task_4.py --user1 "David Smith" --user2 "Bill Duffy" --score-type Euclidean

Euclidean score:
0.585786437626905
```

Рис. 2.5 Результат виконання програми

```
PS E:\ЖДУ\Python\Work_Drobotun_6> python LR_6_task_4.py --user1 "David Smith" --user2 "Bill Duffy" --score-type Pearson

Pearson score:
0.9909924304103233
```

Рис. 2.6 Результат виконання програми

```
PS E:\ЖДУ\Python\Work_Drobotun_6> python LR_6_task_4.py --user1 "David Smith" --user2 "Brenda Peterson" --score-type Pearson

Pearson score:
-0.7236759610155113
```

Рис. 2.7 Результат виконання програми

```
PS E:\ЖДУ\Python\Work_Drobotun_6> python LR_6_task_4.py --user1 "David Smith" --user2 "Samuel Miller" --score-type Pearson

Pearson score:
0.7587869106393281
```

Рис. 2.8 Результат виконання програми

```
PS E:\ЖДУ\Python\Work_Drobotun_6> python LR_6_task_4.py --user1 "David Smith" --user2 "Clarissa Jackson" --score-type Pearson

Pearson score:
0.6944217062199275
```

Рис. 2.9 Результат виконання програми

```
PS E:\ЖДУ\Python\Work_Drobotun_6> python LR_6_task_4.py --user1 "David Smith" --user2 "Adam Cohen" --score-type Pearson

Pearson score:
0.9081082718950217
```

Рис. 3.1 Результат виконання програми

```
PS E:\ЖДУ\Python\Work_Drobotun_6> python LR_6_task_4.py --user1 "David Smith" --user2 "Chris Duncan" --score-type Pearson

Pearson score:
1.0
```

Рис. 3.2 Результат виконання програми

Завдання 2.5. Пошук користувачів зі схожими уподобаннями методом колаборативної фільтрації

```
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])

    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)
    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 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. 3.3 Код програми

```
PS E:\ЖДУ\Python\Work_Drobotun_6> python LR_6_task_5.py --user "Bill Duffy"

Movie recommendations for Bill Duffy:
1. Raging Bull
2. Roman Holiday
```

Рис. 3.4 Результат виконання програми

```
PS E:\ЖДУ\Python\Work_Drobotun_6> python LR_6_task_5.py --user "Clarissa Jackson"

Movie recommendations for Clarissa Jackson:
1. No recommendations possible
```

Рис. 3.6 Результат виконання програми

Завдання 2.6.Створення рекомендаційної системи фільмів

```
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])

    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)
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 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
```

```python
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. 3.7 Код програми

```
PS E:\ЖДУ\Python\Work_Drobotun_6> python LR_6_task_6.py --user "Chris Duncan"

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

Рис. 3.8 Результат виконання програми

```
PS E:\ЖДУ\Python\Work_Drobotun_6> python LR_6_task_6.py --user "Julie Hammel"

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

Рис. 3.9 Результат виконання програми

Висновок: я використовав спеціалізовані бібліотеки та мову програмування Python дослідивши методи регресії даних у машинному навчанні.

Термін колаборативна фільтрація (collaborative filtering) відноситься до процесу ідентифікації шаблонів поведінки об'єктів набору даних з метою прийняття рішень щодо нового об'єкта. У контексті рекомендаційних систем метод колаборативної фільтрації використовують для прогнозування уподобань нового користувача на підставі наявної інформації про уподобання інших користувачів з аналогічними смаками.