ЛАБОРАТОРНА РОБОТА №4

ДОСЛІДЖЕННЯ МЕТОДІВ АНСАМБЛЕВОГО НАВЧАННЯ ТА СТВОРЕННЯ РЕКОМЕНДАЦІЙНИХ СИСТЕМ

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

ХІД РОБОТИ

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

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

# Парсер аргументів
def build_arg_parser():
    parser = argparse.ArgumentParser(description='Classify data using Ensemble
Learning techniques')
    parser.add_argument("--classifier-type", dest="classifier_type", re-
quired=True, choices=['rf', 'erf'],
                        help="Type of classifier to use; can be either 'rf' or
'erf'")
    return parser

if __name__ == '__main__':
    args = build_arg_parser().parse_args()
    classifier_type = args.classifier_type
    # Завантаження вхідних даних
    input_file = 'data_random_forests.txt'
    data = np.loadtxt(input_file, delimiter=',')
    X, Y = data[:, :-1], data[:, -1]
    print(X)
    # Розбиття вхідних даних на три класи
    class_0 = np.array(X[Y == 0])
    class_1 = np.array(X[Y == 1])
    class_2 = np.array(X[Y == 2])
    # Візуалізація вхідних даних
    plt.figure()
```

```
    plt.scatter(class_0[:, 0], class_0[:, 1], s=75, facecolors='red', edgecol-
ors='black', linewidth=1, marker='s')
    plt.scatter(class_1[:, 0], class_1[:, 1], s=75, facecolors='green', edgecol-
ors='black', linewidth=1, marker='o')
    plt.scatter(class_2[:, 0], class_2[:, 1], s=75, facecolors='blue', edgecol-
ors='black', linewidth=1, marker='^')

plt.title('Input data')
plt.show()
# Розбивка даних на навчальний та тестовий набори
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25, ran-
dom_state=5)
# Класифікатор на основі ансамблевого навчання
params = {'n_estimators': 100, 'max_depth': 4, 'random_state': 0}

if classifier_type == 'rf':
    classifier = RandomForestClassifier(**params)
else:
    classifier = ExtraTreesClassifier(**params)

classifier.fit(X_train, Y_train)
visualize_classifier(classifier, X_train, Y_train, 'Training dataset')

# Перевірка роботи класифікатора
class_names = ['Class-0', 'Class-1', 'Class-2']
print("\n" + "#" * 40)
print("\nClassifier performance on training dataset\n")
Y_train_pred = classifier.predict(X_train)
print(classification_report(Y_train, Y_train_pred, target_names=class_names))
print("#" * 40 + "\n")

print("#" * 40)
print("\nClassifier performance on test dataset\n")
Y_test_pred = classifier.predict(X_test)
print(classification_report(Y_test, Y_test_pred, target_names=class_names))
print("#" * 40 + "\n")
```

Результат виконання:

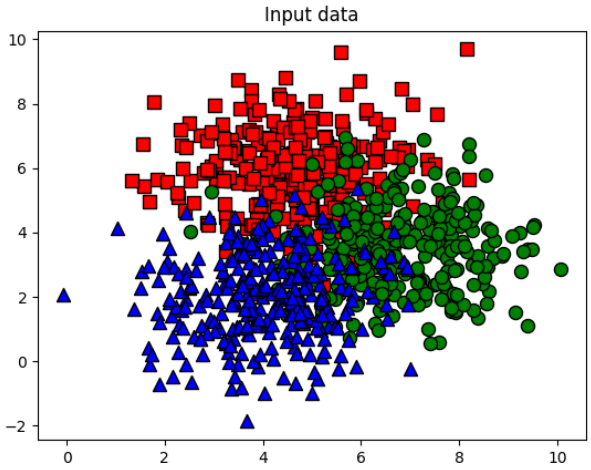

Рис. 1. Зображення розподілення даних.

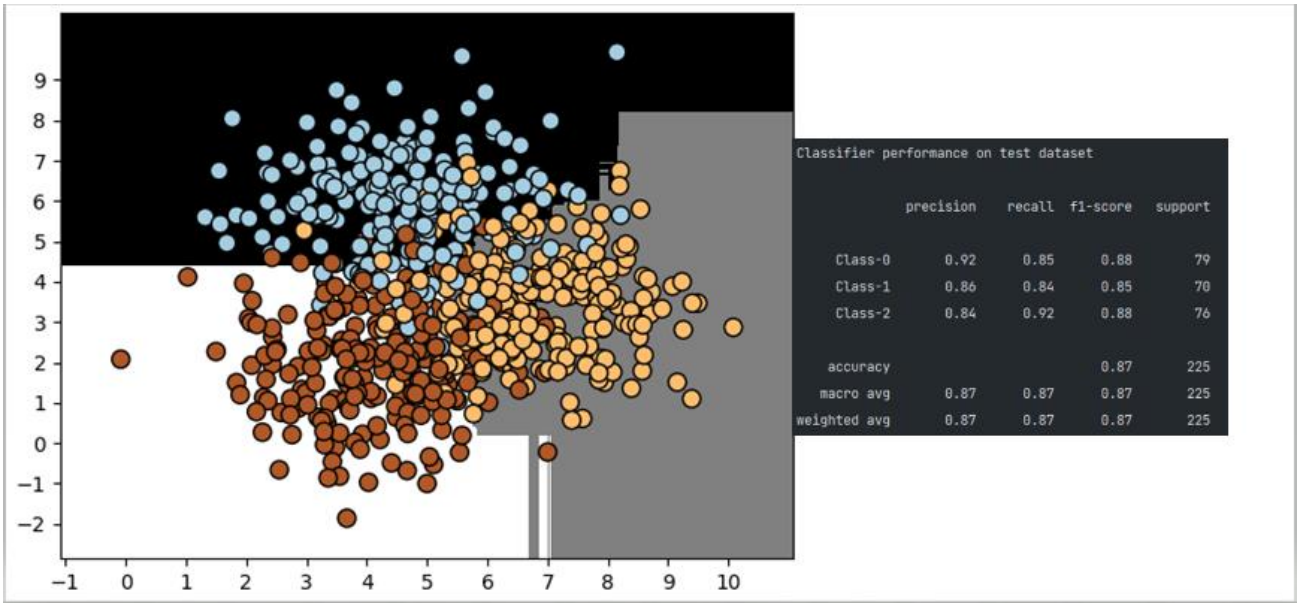

Рис. 2. Класифікація методом випадкових дерев + характеристики роботи методу випадкових дерев.

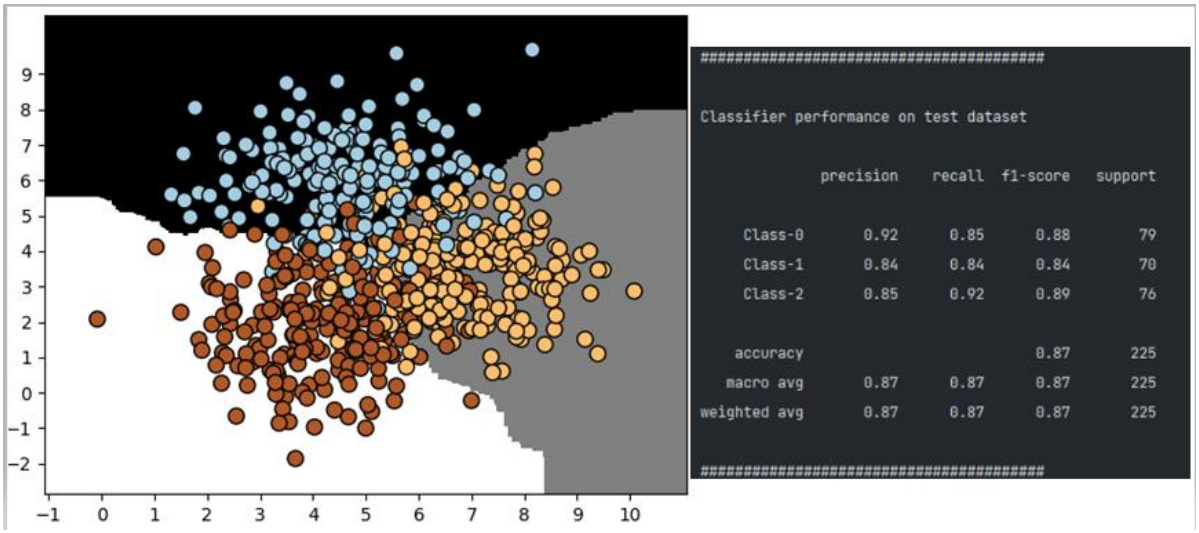

Рис. 3. Класифікація методом гранично випадкових дерев + характеристики роботи методу гранично випадкових дерев.

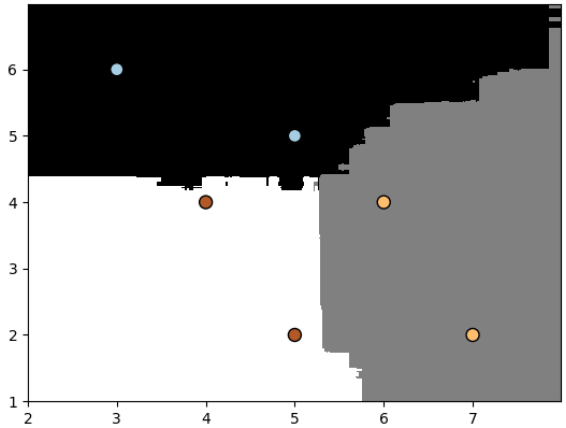

Рис. 4. Візуалізація можливих класів точок (rf)

```
Confidence measure:
Datapoint: [5 5]
Predicted class: Class-0
Probabilities: [0.81427532 0.08639273 0.09933195]
Datapoint: [3 6]
Predicted class: Class-0
Probabilities: [0.93574458 0.02465345 0.03960197]
Datapoint: [6 4]
Predicted class: Class-1
Probabilities: [0.12232404 0.7451078  0.13256816]
Datapoint: [7 2]
Predicted class: Class-1
Probabilities: [0.05415465 0.70660226 0.23924309]
Datapoint: [4 4]
Predicted class: Class-2
Probabilities: [0.20594744 0.15523491 0.63881765]
Datapoint: [5 2]
Predicted class: Class-2
Probabilities: [0.05403583 0.0931115  0.85285267]
```

Рис. 5. Дані про можливі класи (rf)

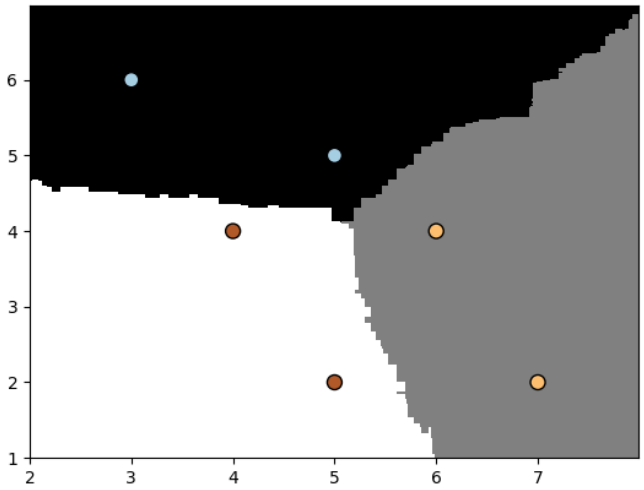

Рис. 6. Візуалізація можливих класів точок (erf)

```
Confidence measure:
Datapoint: [5 5]
Predicted class: Class-0
Probabilities: [0.48904419 0.28020114 0.23075467]
Datapoint: [3 6]
Predicted class: Class-0
Probabilities: [0.66707383 0.12424406 0.20868211]
Datapoint: [6 4]
Predicted class: Class-1
Probabilities: [0.25788769 0.49535144 0.24676087]
Datapoint: [7 2]
Predicted class: Class-1
Probabilities: [0.10794013 0.6246677  0.26739217]
Datapoint: [4 4]
Predicted class: Class-2
Probabilities: [0.33383778 0.21495182 0.45121039]
Datapoint: [5 2]
Predicted class: Class-2
Probabilities: [0.18671115 0.28760896 0.52567989]
```

Рис. 7. Дані про можливі класи (erf)

Висновок по завданню:

Використання випадкових дерев та граничних випадкових дерев дозволяє ефективно класифікувати дані. З практики випливає, що з двох методів, останній, тобто граничні випадкові дерева, виявляється більш ефективним.

Завдання 2. Обробка дисбалансу класів.

```
import sys
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.metrics import classification_report
from utilities import visualize_classifier

if __name__ == '__main__':
    # Завантаження вхідних даних
    input_file = 'data_imbalance.txt'
    data = np.loadtxt(input_file, delimiter=',')
    X, Y = data[:, :-1], data[:, -1]
    # Поділ вхідних даних на два класи на підставі міток
    class_0 = np.array(X[Y == 0])
    class_1 = np.array(X[Y == 1])
    # Візуалізація вхідних даних
    plt.figure()
    plt.scatter(class_0[:, 0], class_0[:, 1], s=75, facecolors='black', edgecol-
ors='black', linewidth=1, marker='x')
    plt.scatter(class_1[:, 0], class_1[:, 1], s=75, facecolors='white', edgecol-
ors='black', linewidth=1, marker='o')
    plt.title('Input data')
    # Розбиття даних на навчальний та тестовий набори
```

```
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25, ran-
dom_state=5)
    # Класифікатор на основі гранично випадкових лісів
    params = {'n_estimators': 100, 'max_depth': 4, 'random_state': 0}

    if len(sys.argv) > 1:
        if sys.argv[1] == 'balance':
            params['class_weight'] = 'balanced'
        else:
            raise TypeError("Invalid input argument; should be 'balance' or noth-
ing")

    classifier = ExtraTreesClassifier(**params)
    classifier.fit(X_train, Y_train)
    visualize_classifier(classifier, X_train, Y_train)

    Y_test_pred = classifier.predict(X_test)
    # Обчислення показників ефективності класифікатора
    class_names = ['Class-0', 'Class-1']
    print("\n" + "#"*40)
    print("Classifier performance on training dataset")
    print(classification_report(Y_test, Y_test_pred, target_names=class_names))
    print("#"*40)
    print("Classifier performance on test dataset")
    print(classification_report(Y_test, Y_test_pred, target_names=class_names))
    print("#"*40 + "\n")
    plt.show()
```

Результат виконання:

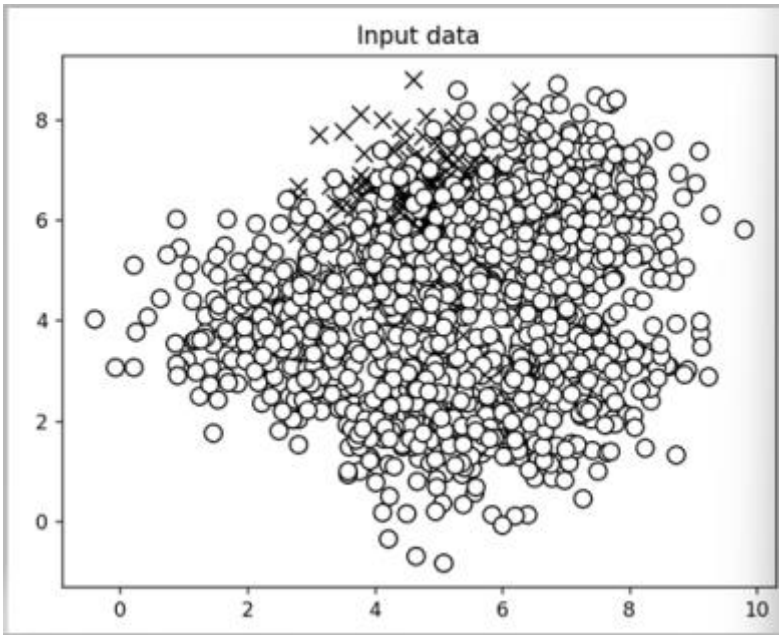

Рис. 8. Розподілення незбалансованих даних.

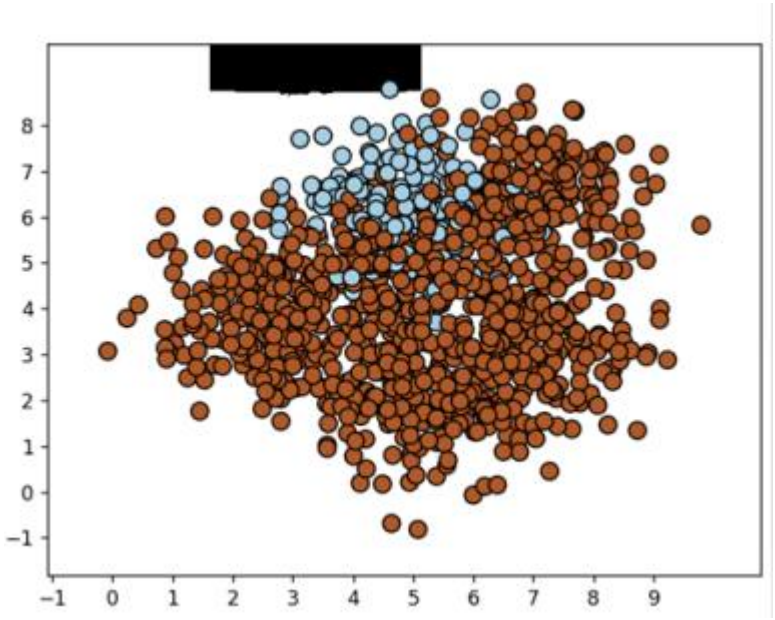

Рис. 9. Розподілення незбалансованих даних.

```
#####################################
Classifier performance on training dataset
              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

#####################################
Classifier performance on test dataset
              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. 10. Характеристика незбалансованої класифікації.

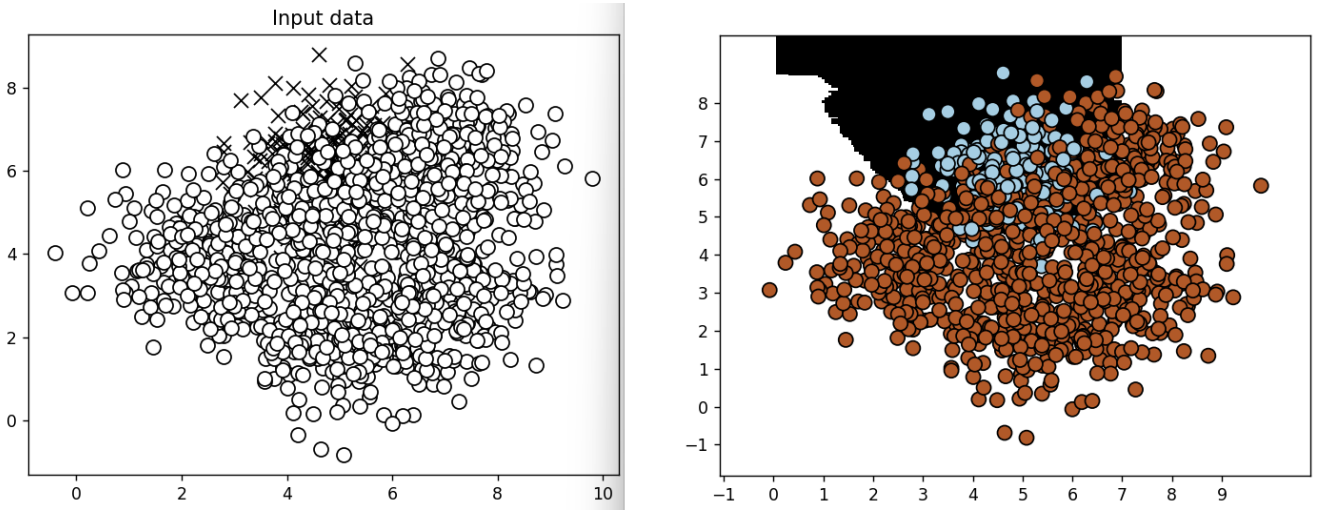

Рис. 11. Розподілення збалансованих даних.

```
########################################
Classifier performance on training 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

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

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

Рис. 12. Характеристики збалансованої класифікації.

Висновок по завданню:

Аналізуючи показники якості класифікації, помічаємо, що Recall для Class-0 є високим, але для Class-1 нижчим. Це може виникнути через нерівномірний розподіл даних, зокрема, якщо кількість прикладів у різних класах є незбалансованою. Accuracy на тестовій вибірці становить приблизно 78%. Однак в умовах дисбалансу класів цей показник може бути неточним, оскільки класифікатор може віддавати перевагу передбаченню більш численного класу.

Показники якості також вказують на те, що класифікатору важко класифікувати Class-1, що може бути пов'язано з дисбалансом класів у даних. Навчання на незбалансованих даних може призводити до того, що модель частіше робить правильні передбачення для більш численного класу.

Загалом, результати класифікації можуть бути покращені шляхом вирішення проблеми дисбалансу класів.

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

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

input_file = 'data_random_forests.txt'
data = np.loadtxt(input_file, delimiter=',')
X, Y = data[:, :-1], data[:, -1]
# Розбиття даних на три класи на підставі міток
class_0 = np.array(X[Y == 0])
class_1 = np.array(X[Y == 1])
class_2 = np.array(X[Y == 2])
# Розбиття даних на навчальний та тестовий набори
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25, ran-
dom_state=5)
# Визначення сітки значень параметрів
parameter_grid = [{'n_estimators': [100], 'max_depth': [2, 4, 7, 12, 16]},
                  {'max_depth': [4], 'n_estimators': [25, 50, 100, 250]}]

metrics = ['precision_weighted', 'recall_weighted']

for metric in metrics:
    print("#### Searching optimal parameters for", metric)
    classifier = GridSearchCV(ExtraTreesClassifier(random_state=0), parame-
ter_grid, cv=5, scoring=metric)
    classifier.fit(X_train, Y_train)
    print("\nScores across the parameter grid:")

    for params, avg_score in classifier.cv_results_.items():
        print(params, '-->', avg_score)
    print("\nHighest scoring parameter set:", classifier.best_params_)

    Y_test_pred = classifier.predict(X_test)
    class_names = ['Class-0', 'Class-1', 'Class-2']
    print("#"*40)
    print("Classifier performance on training dataset")
```

```
    print(classification_report(Y_test, Y_test_pred, target_names=class_names))
    print("#"*40 + "\n")

    visualize_classifier(classifier, X_test, Y_test)
```

Результат виконання:

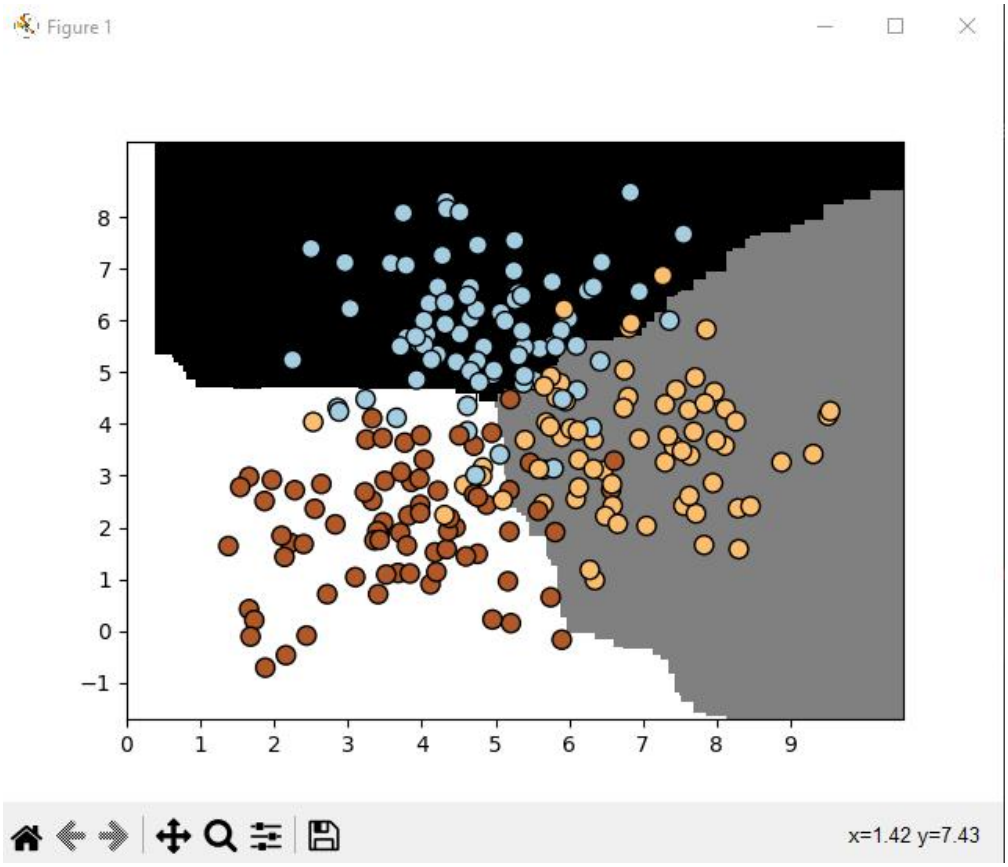

Рис. 13. Візуалізація класифікації даних зі сітковим пошуком.

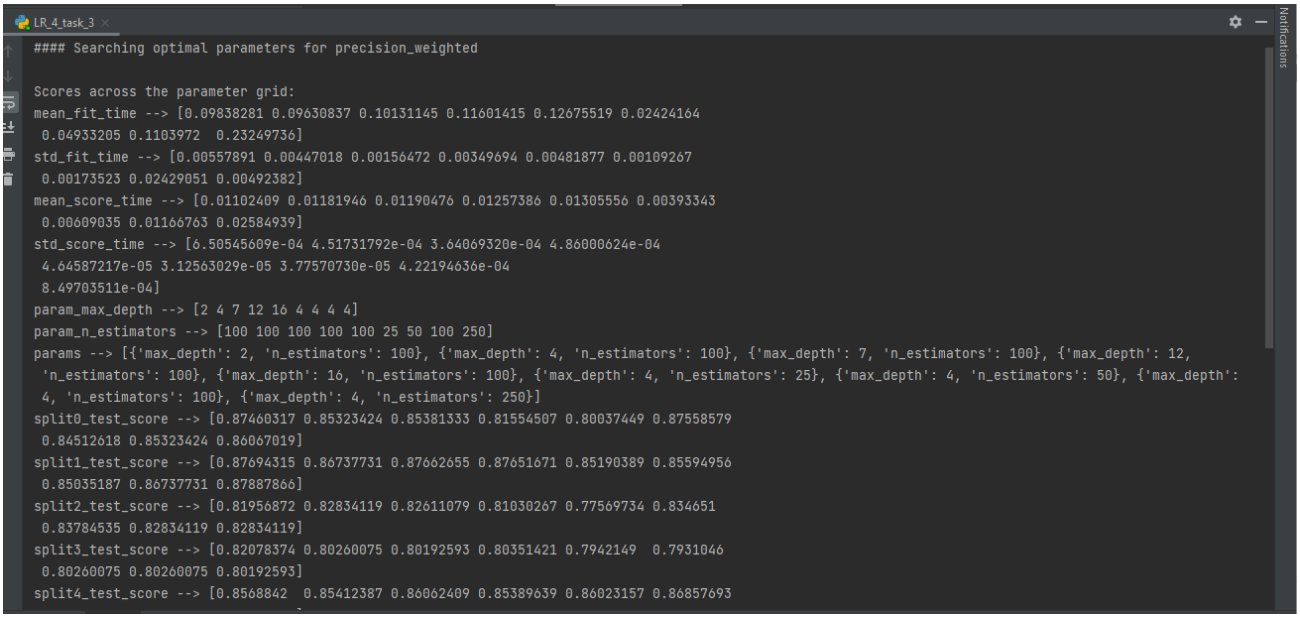

Рис. 14. Отримання даних процесу класифікації.

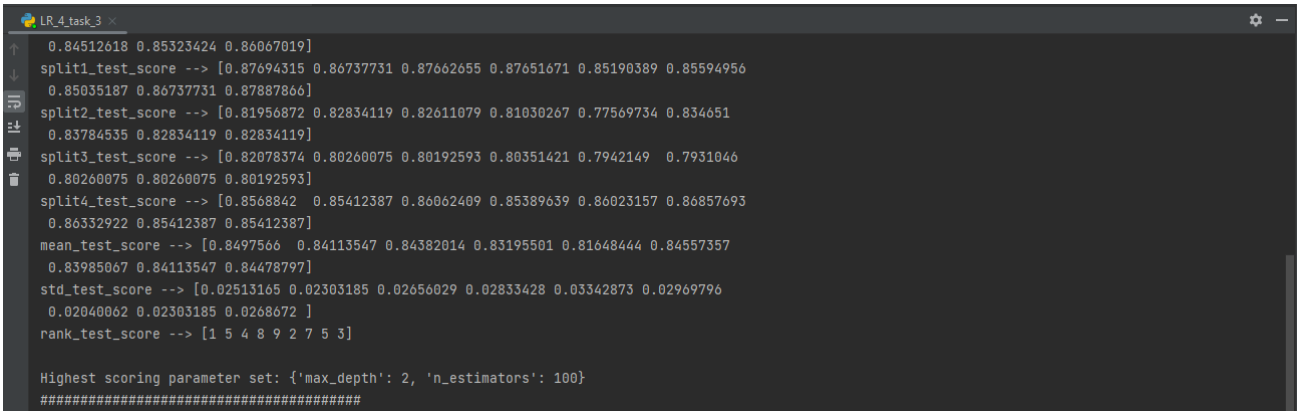

Рис. 15. Отримання даних процесу класифікації.

```
########################################
Classifier performance on training dataset
              precision    recall  f1-score   support

     Class-0       0.94      0.81      0.87        79
     Class-1       0.81      0.86      0.83        70
     Class-2       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. 16. Характеристика класифікації зі сітковим пошуком.

Налаштування параметрів для precision та recall:

- Для визначення найкращих параметрів використовувалася сітка параметрів. Кращий результат досягнуто при значеннях max_depth=2 і n_estimators=100, що відображено в найкращому значенні mean_test_score.

- Звіт про класифікацію для тестового набору показав високі показники precision, recall і F1-score для кожного класу, а також високу accuracy.

- Налаштування параметрів для recall збігаються з найкращими параметрами для precision. Виявлено, що параметри, які забезпечують підвищену precision, також позитивно впливають на recall.

Отже, модель з параметрами max_depth=2 і n_estimators=100 досягає гармонійного балансу між precision і recall на даному наборі даних, демонструючи добрі здібності до узагальнення.

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

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

input_file = 'traffic_data.txt'
data = []
with open(input_file, 'r') as f:
    for line in f.readlines():
        items = line[:-1].split(',')
        data.append(items)

data = np.array(data)

label_encoder = []
X_encoded = np.empty(data.shape)
for i, item in enumerate(data[0]):
    if item.isdigit():
        X_encoded[:, i] = data[:, i]
    else:
        label_encoder.append(preprocessing.LabelEncoder())
        X_encoded[:, i] = label_encoder[-1].fit_transform(data[:, i])

X = X_encoded[:, :-1].astype(int)
Y = X_encoded[:, -1].astype(int)

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25, ran-
dom_state=5)
params = {'n_estimators': 100, 'max_depth': 4, 'random_state': 0}
regressor = ExtraTreesRegressor(**params)
regressor.fit(X_train, Y_train)

Y_pred = regressor.predict(X_test)
print("Mean absolute error =", round(mean_absolute_error(Y_test, Y_pred), 2))

test_datapoint = ['Saturday', '10:20', 'Atlanta', 'no']
test_datapoint_encoded = [-1] * len(test_datapoint)
count = 0

for i, item in enumerate(test_datapoint):
    if item.isdigit():
        test_datapoint_encoded[i] = int(test_datapoint[i])
    else:
        test_datapoint_encoded[i] = int(label_encoder[count].transform([test_data-
point[i]]))
        count = count + 1

test_datapoint_encoded = np.array(test_datapoint_encoded)
```

```
print("Predicted traffic:", int(regressor.predict([test_datapoint_encoded])[0]))
```

Результат виконання:

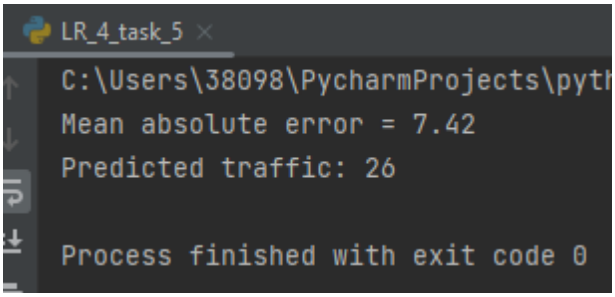

Рис. 17. Результат регресії на основі гранично випадкових лісів.

Висновок до завдання:

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

Завдання 6. Створення навчального конвеєра (конвеєра машинного навчання)

```
from sklearn.datasets import _samples_generator
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.pipeline import Pipeline
from sklearn.ensemble import ExtraTreesClassifier
# Генерування даних
X, Y = _samples_generator.make_classification(n_samples=150, n_features=25,
n_classes=3,
                                              n_informative=6, n_redundant=0,
random_state=7)
# Вибір k найважливіших ознак
k_best_selector = SelectKBest(f_regression, k=10)
# Ініціалізація класифікатора на основі гранично випадкового лісу
classifier = ExtraTreesClassifier(n_estimators=60, max_depth=4)
# Створення конвеєра
processor_pipeline = Pipeline([('selector', k_best_selector), ('erf',
classifier)])
# Встановлення параметрів
processor_pipeline.set_params(selector__k=7, erf__n_estimators=30)
# Навчання конвеєра
processor_pipeline.fit(X, Y)
# Прогнозування результатів для вхідних даних
print("Predicted output:", processor_pipeline.predict(X))
# Виведення оцінки
print("Score:", processor_pipeline.score(X, Y))
# Виведення ознак, відібраних селектором конвеєра
status = processor_pipeline.named_steps['selector'].get_support()
# Вилучення та виведення індексів обраних ознак
selected = [i for i, x in enumerate(status) if x]
print("Selected features:", selected)
```

Результат виконання:

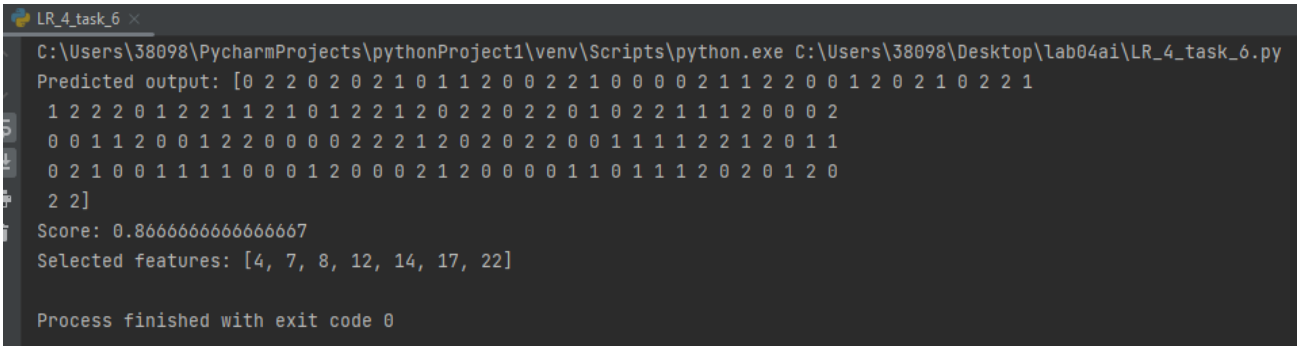

Рис. 18. Отримані результати навчального конвеєра.

Висновок до завдання:

Ваш пояснювальний текст чудово розкриває важливі аспекти результатів моделі. Давайте організуємо цю інформацію в компактніший формат:

- Список Predicted Output (Попередні прогнози): 150 передбачених значень, кожне вказує на клас, до якого ймовірно належить відповідний приклад.

-Score (Оцінка): Точність моделі на навчальних даних склала приблизно 88.67%. Це відсоток правильно класифікованих зразків і вказує на ефективність моделі.

-Вибрані ознаки: Використано метод SelectKBest для відбору найбільш важливих ознак з вхідних даних. Останній рядок містить індекси вибраних ознак, які вказують на відповідні елементи набору даних.

Цей компактний опис надає зрозумілу та інформативну інформацію про результати моделі та процес відбору ознак.

Завдання 7. Пошук найближчих сусідів.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import NearestNeighbors

# Input data
X = np.array([[2.1, 1.3], [1.3, 3.2], [2.9, 2.5], [2.7, 5.4], [3.8, 0.9],
        [7.3, 2.1], [4.2, 6.5], [3.8, 3.7], [2.5, 4.1], [3.4, 1.9],
        [5.7, 3.5], [6.1, 4.3], [5.1, 2.2], [6.2, 1.1]])

# Number of nearest neighbors
k = 5

# Test datapoint
test_datapoint = [4.3, 2.7]

# Plot input data
plt.figure()
plt.title('Input data')
```

```
plt.scatter(X[:,0], X[:,1], marker='o', s=75, color='black')

# Build K Nearest Neighbors model
knn_model = NearestNeighbors(n_neighbors=k, algorithm='ball_tree').fit(X)
distances, indices = knn_model.kneighbors([test_datapoint])

# Print the 'k' nearest neighbors
print("\nK Nearest Neighbors:")
for rank, index in enumerate(indices[0][:k], start=1):
    print(str(rank) + " ==>", X[index])

# Visualize the nearest neighbors along with the test datapoint
plt.figure()
plt.title('Nearest neighbors')
plt.scatter(X[:, 0], X[:, 1], marker='o', s=75, color='k')
plt.scatter(X[indices][0][:][:, 0], X[indices][0][:][:, 1],
        marker='o', s=250, color='k', facecolors='none')
plt.scatter(test_datapoint[0], test_datapoint[1],
        marker='x', s=75, color='k')

plt.show()
```

Результат виконання:

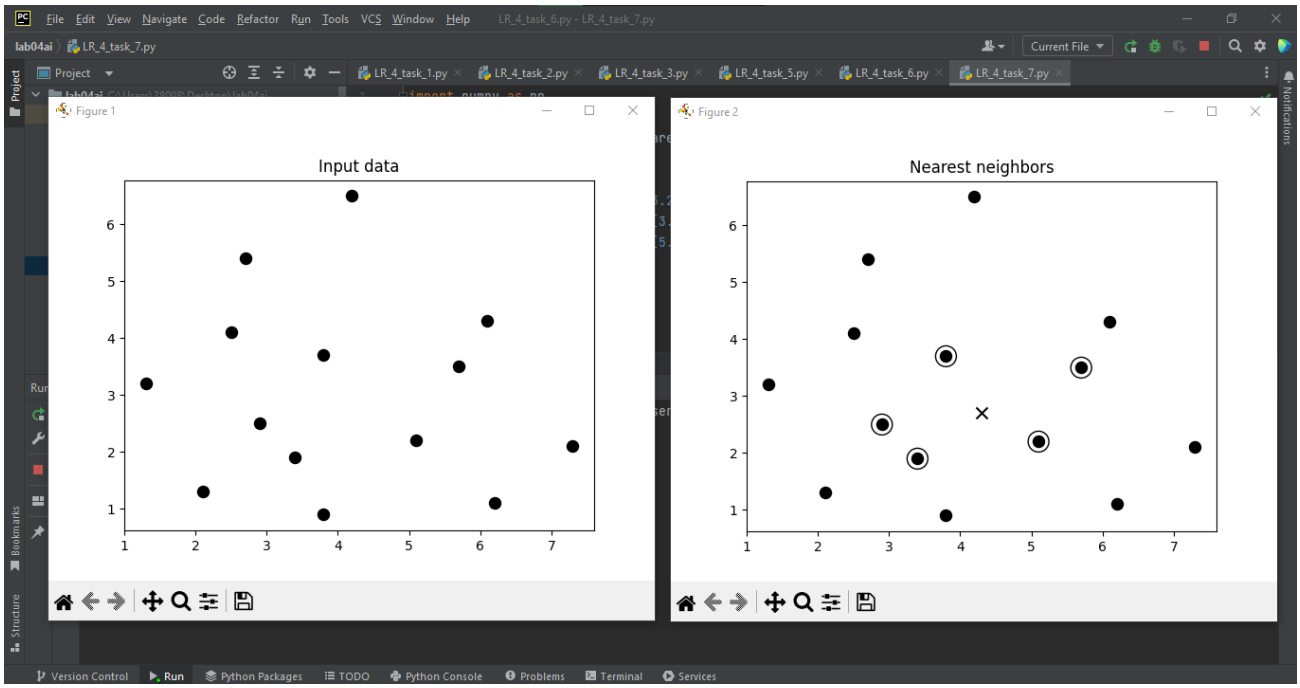

Рис. 19. Пошук найближчих сусідів.

На графіку зліва – вхідні дані.

Найближчі сусіди зображені на графіку справа, координати в терміналі:

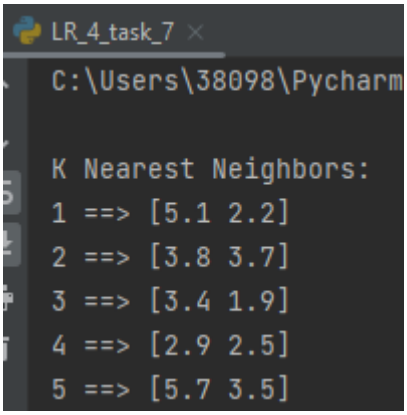

Рис. 20. Дані про найближчих сусідів.

Завдання 8. Створити класифікатор методом k найближчих сусідів.

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from sklearn import neighbors, datasets

input_file = 'data.txt'
data = np.loadtxt(input_file, delimiter=',')
X, Y = data[:, :-1], data[:, -1]

num_neighbors = 12
step_size = 0.01
classifier = neighbors.KNeighborsClassifier(num_neighbors, weights='distance')
classifier.fit(X, Y)

X_min, X_max = X[:, 0].min() - 1, X[:, 0].max() + 1
Y_min, Y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
X_values, Y_values = np.meshgrid(np.arange(X_min, X_max, step_size),
np.arange(Y_min, Y_max, step_size))

output_mesh = classifier.predict(np.c_[X_values.ravel(), Y_values.ravel()])
output_mesh = output_mesh.reshape(X_values.shape)

plt.figure()
plt.pcolormesh(X_values, Y_values, output_mesh, cmap=cm.Paired)
plt.scatter(X[:, 0], X[:, 1], c=Y, s=80, edgecolors='black', linewidth=1,
cmap=cm.Paired)
plt.xlim(X_values.min(), X_values.max())
plt.ylim(Y_values.min(), Y_values.max())
plt.title('K Nearest Neighbors classifier on input data')

test_datapoint = [5.1, 3.6]
plt.scatter(test_datapoint[0], test_datapoint[1], marker='o', s=100, linewidths=3,
color='black')

_, indices = classifier.kneighbors([test_datapoint])
indices = np.asarray(indices).flatten()
plt.scatter(X[indices][:, 0], X[indices][:, 1], marker='*', s=80, linewidths=1,
color='black', facecolors='none')
plt.show()

print("Predicted output:", classifier.predict([test_datapoint])[0])
```

Результат виконання:

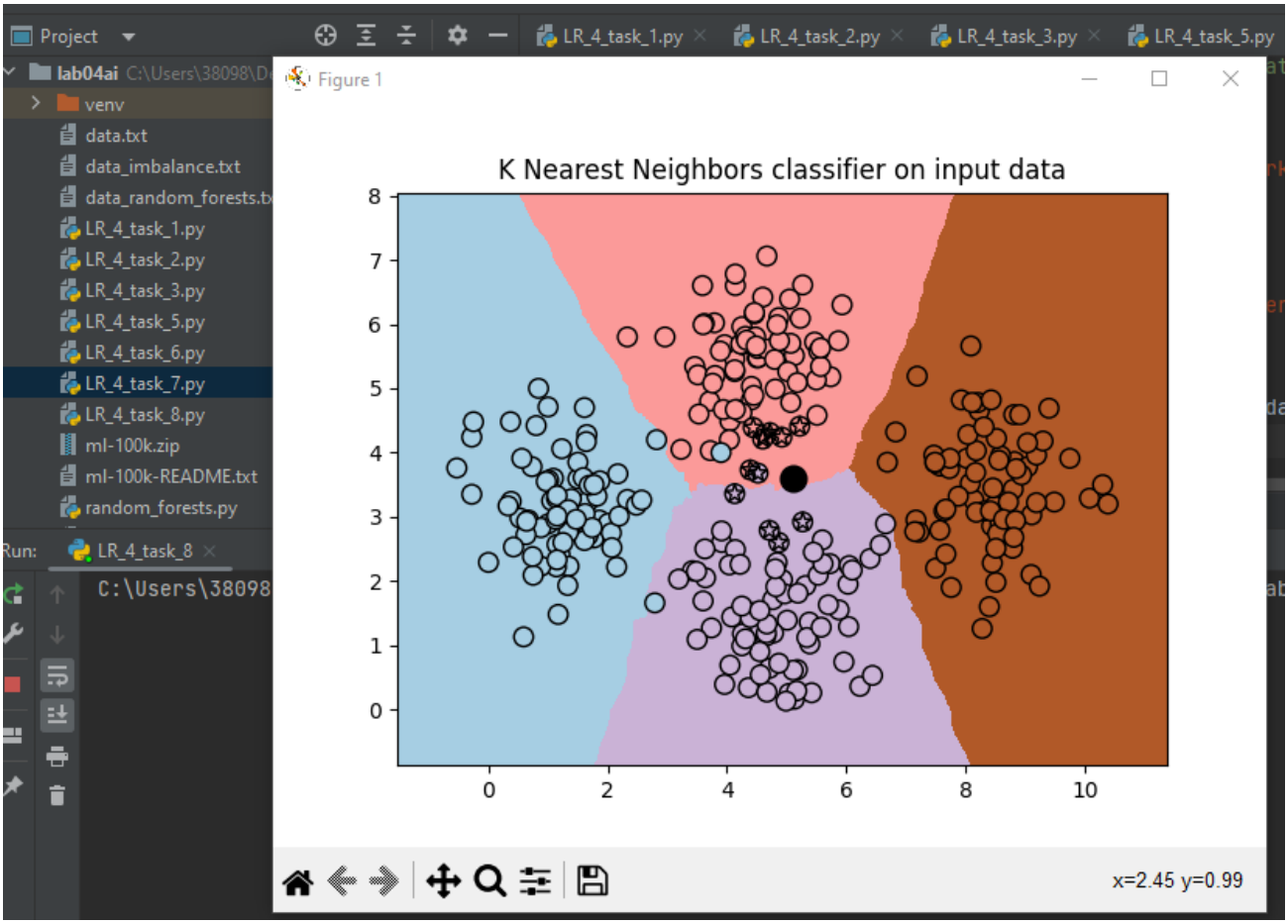

Рис. 21. Класифікація методом К-найближчих сусідів та найближчі сусіди введеної точки.

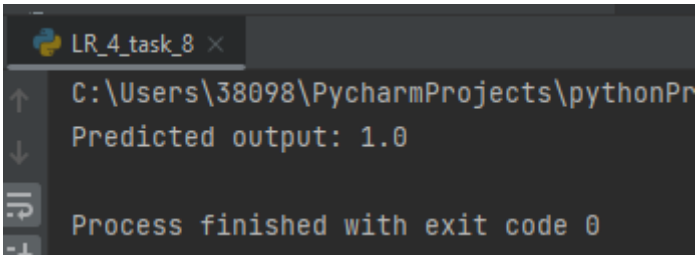

Рис. 22. Обрахований клас точки.

Під час виконання завдання я використовував метод k найближчих сусідів для класифікації даних. Для наглядності результатів були побудовані графіки.

На графіку вхідних даних наведено представлення вхідних даних за допомогою різних маркерів, які відображають класи даних.

На графіку "Кордони моделі класифікатора на основі k найближчих сусідів" відображені області, розділені межами класів. Ці межі були передбачені класифікатором на основі навчальних даних, і кольорами позначено передбачувані області для кожного класу.

На третьому графіку представлені всі навчальні точки, а також виділена одна точка (test datapoint) чорним хрестиком. Для цієї виокремленої точки будемо шукати найближчих сусідів.

На останньому графіку надано зображення k найближчих сусідів для test datapoint. Сусідні точки відмічені різними маркерами відповідно до їх класів і підкреслені кольорами для кращої видимості на графіку.

Завдання 9. 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


# Compute the Euclidean distance score between user1 and user2
def euclidean_score(dataset, user1, user2):
    if user1 not in dataset:
        raise TypeError('Cannot find ' + user1 + ' in the dataset')

    if user2 not in dataset:
        raise TypeError('Cannot find ' + user2 + ' in the dataset')

    # Movies rated by both user1 and user2
    common_movies = {}

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

    # If there are no common movies between the users,
```

```
    # then the score is 0
    if len(common_movies) == 0:
        return 0

    squared_diff = []

    for item in dataset[user1]:
        if item in dataset[user2]:
            squared_diff.append(np.square(dataset[user1][item] - da-
taset[user2][item]))

    return 1 / (1 + np.sqrt(np.sum(squared_diff)))


# Compute the Pearson correlation score between user1 and user2
def pearson_score(dataset, user1, user2):
    if user1 not in dataset:
        raise TypeError('Cannot find ' + user1 + ' in the dataset')

    if user2 not in dataset:
        raise TypeError('Cannot find ' + user2 + ' in the dataset')

    # Movies rated by both user1 and user2
    common_movies = {}

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

    num_ratings = len(common_movies)

    # If there are no common movies between user1 and user2, then the score is 0
    if num_ratings == 0:
        return 0

    # Calculate the sum of ratings of all the common movies
    user1_sum = np.sum([dataset[user1][item] for item in common_movies])
    user2_sum = np.sum([dataset[user2][item] for item in common_movies])

    # Calculate the sum of squares of ratings of all the common movies
    user1_squared_sum = np.sum([np.square(dataset[user1][item]) for item in com-
mon_movies])
    user2_squared_sum = np.sum([np.square(dataset[user2][item]) for item in com-
mon_movies])

    # Calculate the sum of products of the ratings of the common movies
    sum_of_products = np.sum([dataset[user1][item] * dataset[user2][item] for item
in common_movies])

    # Calculate the Pearson correlation score
    Sxy = sum_of_products - (user1_sum * user2_sum / num_ratings)
    Sxx = user1_squared_sum - np.square(user1_sum) / num_ratings
    Syy = user2_squared_sum - np.square(user2_sum) / num_ratings

    if Sxx * Syy == 0:
        return 0

    return Sxy / np.sqrt(Sxx * Syy)


if __name__ == '__main__':
    args = build_arg_parser().parse_args()
    user1 = args.user1
```

```
    user2 = args.user2
    score_type = args.score_type

    ratings_file = 'ratings.json'

    with open(ratings_file, 'r') as f:
        data = json.loads(f.read())

    if score_type == 'Euclidean':
        print("\nEuclidean score:")
        print(euclidean_score(data, user1, user2))
    else:
        print("\nPearson score:")
        print(pearson_score(data, user1, user2))
```

Результат виконання:

```
PS C:\Users\38098\Desktop\lab04ai> py -W ignore C:\Users\38098\Desktop\lab04ai\LR_4_task_9.py   --user1 "David Smith" --user2 "Bill Duffy" --score-type Euclidean

Euclidean score:
0.585786437626905
PS C:\Users\38098\Desktop\lab04ai>
```

```
PS C:\Users\38098\Desktop\lab04ai> py -W ignore C:\Users\38098\Desktop\lab04ai\LR_4_task_9.py   --user1 "David Smpy -W ignore C:\Users\38098\Desktop\lab04ai\LR_4_t
ask_9.py   --user1 "David Smith" --user2 "Bill Duffy" --score-type Pearson

Pearson score:
0.9909924304103233
PS C:\Users\38098\Desktop\lab04ai>
```

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

```
PS C:\Users\38098\Desktop\lab04ai> py -W ignore C:\Users\38098\Desktop\lab04ai\LR_4_task_9.py   --user1 "David Smith" --user2 "Brenda Peterson" --score-type Euclid
ean

Euclidean score:
0.1424339656566283
PS C:\Users\38098\Desktop\lab04ai> py -W ignore C:\Users\38098\Desktop\lab04ai\LR_4_task_9.py   --user1 "David Smith" --user2 "Brenda Peterson" --score-type Pearso
n

Pearson score:
-0.7236759610155113
PS C:\Users\38098\Desktop\lab04ai>
```

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

```
PS C:\Users\38098\Desktop\lab04ai> py -W ignore C:\Users\38098\Desktop\lab04ai\LR_4_task_9.py   --user1 "David Smith" --user2 "Samuel Miller" --score-type Euclidea
n

Euclidean score:
0.30383243470068705
PS C:\Users\38098\Desktop\lab04ai> py -W ignore C:\Users\38098\Desktop\lab04ai\LR_4_task_9.py   --user1 "David Smith" --user2 "Samuel Miller" --score-type Pearson

Pearson score:
0.7587869106393281
PS C:\Users\38098\Desktop\lab04ai>
```

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

```
PS C:\Users\38098\Desktop\lab04ai> py -W ignore C:\Users\38098\Desktop\lab04ai\LR_4_task_9.py   --user1 "David Smith" --user2 "Julie Hammel" --score-type Euclidean


Euclidean score:
0.2857142857142857
PS C:\Users\38098\Desktop\lab04ai> py -W ignore C:\Users\38098\Desktop\lab04ai\LR_4_task_9.py   --user1 "David Smith" --user2 "Julie Hammel" --score-type Pearson

Pearson score:
0
PS C:\Users\38098\Desktop\lab04ai>
```

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

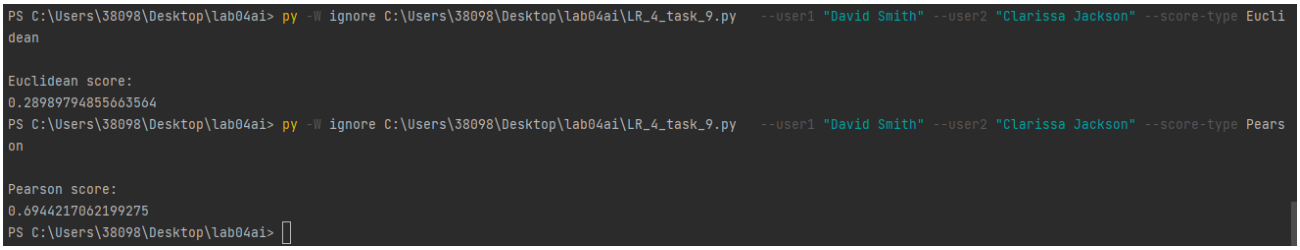

Рис. 28. Обрахунок оцінок для David Smith та Clarissa Jackson.

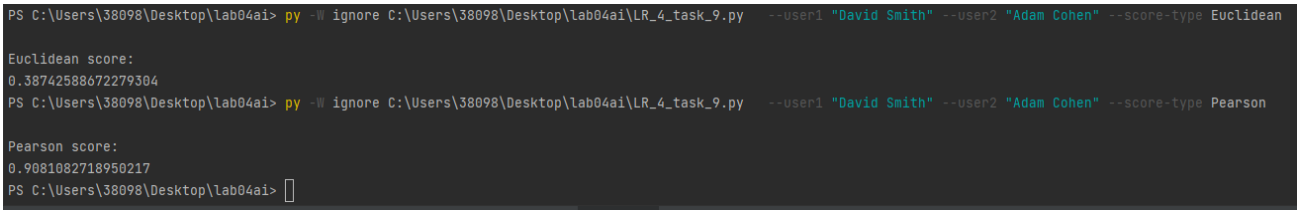

Рис. 29. Обрахунок оцінок для David Smith та Adam Cohen.

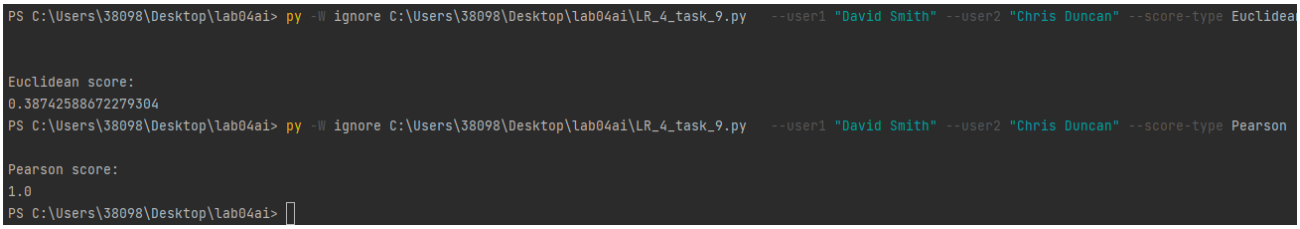

Рис. 30. Обрахунок оцінок для David Smith та Chris Duncan.

Результати вказують на те, що метрика Pearson зазвичай видає вищі оцінки схожості порівняно з метрикою Euclidean. Проте слід зауважити, що оцінки схожості можуть варіюватися залежно від вибору конкретних пар користувачів та їх оцінок фільмів. Важливо вибирати метрику відповідно до конкретного завдання та особливостей даних.

Також варто відзначити, що для пари користувачів "David Smith" та "Chris Duncan" метрика Pearson показує максимальну схожість, рівну 1.0. Це свідчить про те, що вони оцінюють фільми абсолютно однаково, що може бути використано для рекомендацій користувачам або аналізу їх схожості.

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

```python
import argparse
import json
import numpy as np

from LR_4_task_9 import pearson_score

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

# Finds users in the dataset that are similar to the input user
def find_similar_users(dataset, user, num_users):
    if user not in dataset:
        raise TypeError('Cannot find ' + user + ' in the dataset')

    # Compute Pearson score between input user
    # and all the users in the dataset
    scores = np.array([[x, pearson_score(dataset, user,
            x)] for x in dataset if x != user])

    # Sort the scores in decreasing order
    scores_sorted = np.argsort(scores[:, 1])[::-1]

    # Extract the top 'num_users' scores
    top_users = scores_sorted[:num_users]

    return scores[top_users]

if __name__=='__main__':
    args = build_arg_parser().parse_args()
    user = args.user

    ratings_file = 'ratings.json'

    with open(ratings_file, 'r') as f:
        data = json.loads(f.read())

    print('\nUsers similar to ' + user + ':\n')
    similar_users = find_similar_users(data, user, 3)
    print('User\t\t\tSimilarity score')
    print('-'*41)
    for item in similar_users:
        print(item[0], '\t\t', round(float(item[1]), 2))
```

Результат виконання:

```
PS C:\Users\38098\Desktop\lab04ai> py -W ignore C:\Users\38098\Desktop\lab04ai\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:\Users\38098\Desktop\lab04ai>
```

Рис. 31. Знаходження користувачів схожих на Bill Duffy.

```
PS C:\Users\38098\Desktop\lab04ai> py -W ignore C:\Users\38098\Desktop\lab04ai\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:\Users\38098\Desktop\lab04ai>
```

Рис. 32. Знаходження користувачів схожих на Clarissa Jackson.

Шукання користувачів, схожих на певного користувача за допомогою кореляції Пірсона є потужним методом для побудови рекомендаційних систем на основі схожості між користувачами. Зокрема, ми використовуємо кореляцію Пірсона для визначення ступеня схожості між оцінками фільмів, які надали різні користувачі.

-Найбільше на Bill Duffy схожий користувач David Smith. Показник подібності склав 0.99. Це означає, що вони мають дуже схожі смаки. На другому та третьому місці знаходяться Samuel Miller (0.88) та Adam Cohen (0.86). Усі ці користувачі мають схожі смаки у фільмах.

-На Clarissa Jackson найбільше схожий в аспекті вибору фільмів Chris Duncan (показник подібності 1.0). Їх смаки практично ідентичні. На другому та третьому місці Bill Duffy (0.83) та Samuel Miller (0.73).

Ці дані корисно використовувати для рекомендації фільмів користувачам. Адже Bill Duffy найбільше сподобаються фільми, що сподобалися David Smith, а Clarissa Jackson з великою вірогідністю вподобає фільми, які сподобалися Chris Duncan.

Завдання 11. Створення рекомендаційної системи фільмів.

```python
import argparse
import json
import numpy as np

from LR_4_task_9 import pearson_score
from LR_4_task_10 import find_similar_users
```

```
def build_arg_parser():
    parser = argparse.ArgumentParser(description='Find the movie recommendations
for the given user')
    parser.add_argument('--user', dest='user', required=True,
                        help='Input user')
    return parser


# Get movie recommendations for the input user
def get_recommendations(dataset, input_user):
    if input_user not in dataset:
        raise TypeError('Cannot find ' + input_user + ' in the dataset')

    overall_scores = {}
    similarity_scores = {}

    for user in [x for x in dataset if x != input_user]:
        similarity_score = pearson_score(dataset, input_user, user)

        if similarity_score <= 0:
            continue

        filtered_list = [x for x in dataset[user] if x not in \
                         dataset[input_user] or dataset[input_user][x] == 0]

        for item in filtered_list:
            overall_scores.update({item: dataset[user][item] * similarity_score})
            similarity_scores.update({item: similarity_score})

    if len(overall_scores) == 0:
        return ['No recommendations possible']

    # Generate movie ranks by normalization
    movie_scores = np.array([[score / similarity_scores[item], item]
                             for item, score in overall_scores.items()])

    # Sort in decreasing order
    movie_scores = movie_scores[np.argsort(movie_scores[:, 0])[::-1]]

    # Extract the movie recommendations
    movie_recommendations = [movie for _, movie in movie_scores]

    return movie_recommendations


if __name__ == '__main__':
    args = build_arg_parser().parse_args()
    user = args.user

    ratings_file = 'ratings.json'

    with open(ratings_file, 'r') as f:
        data = json.loads(f.read())

    print("\nMovie recommendations for " + user + ":")
    movies = get_recommendations(data, user)
    for i, movie in enumerate(movies):
        print(str(i + 1) + '. ' + movie)
```

Результат виконання:

```
PS C:\Users\38098\Desktop\lab04ai> py -W ignore C:\Users\38098\Desktop\lab04ai\LR_4_task_11.py --user "Chris Duncan"

Movie recommendations for Chris Duncan:
1. Vertigo
2. Scarface
3. Goodfellas
4. Roman Holiday
PS C:\Users\38098\Desktop\lab04ai>
```

Рис. 33. Рекомендації для Chris Duncan.

```
PS C:\Users\38098\Desktop\lab04ai> py -W ignore C:\Users\38098\Desktop\lab04ai\LR_4_task_11.py --user "Julie Hammel"

Movie recommendations for Julie Hammel:
1. The Apartment
2. Vertigo
3. Raging Bull
PS C:\Users\38098\Desktop\lab04ai>
```

Рис. 34. Рекомендації для Julie Hammel.

Даний код рекомендує фільми користувачеві на основі аналізу подібності його оцінок з оцінками інших користувачів у наборі даних:

Для Chris Duncan рекомендовано фільми Vertigo, Scarface, Goodfells та Ro- man Holiday.

Для Julie Hammel — The Apartment, Vertigo та Raging Bull.

Ці фільми з великою імовірністю відповідають смакам введених користувачів.

GitHub: https://github.com/unravee1/AI_labs