ЛАБОРАТОРНА РОБОТА №4

ДОСЛІДЖЕННЯ МЕТОДІВ АНСАМБЛЕВОГО НАВЧАННЯ ТА СТВОРЕННЯ РЕКОМЕНДАЦІЙНИХ СИСТЕМ

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

Хід роботи:

GitHub: https://github.com/ipz201svo/AI

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

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

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

if __name__ == "__main__":
    # Вилучення вхідних аргументів
    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]
    # 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="white",
    edgecolors="black",
    linewidth=1,
    marker="s",
)
plt.scatter(
    class_1[:, 0],
    class_1[:, 1],
    s=75,
    facecolors="white",
    edgecolors="black",
    linewidth=1,
    marker="o",
)
plt.scatter(
    class_2[:, 0],
    class_2[:, 1],
    s=75,
    facecolors="white",
    edgecolors="black",
    linewidth=1,
    marker="^",
)
plt.title("Input data")

# Розбивка даних на навчальний та тестовий набори
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_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")

y_test_pred = classifier.predict(X_test)
visualize_classifier(classifier, X_test, y_test, "Test dataset")

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

```
            print("#" * 40)
            print("\nClassifier performance on test dataset\n")
            print(classification_report(y_test, y_test_pred, target_names=class_names))
            print("#" * 40 + "\n")

            # Обчислення параметрів довірливості
            test_datapoints = np.array([[5, 5], [3, 6], [6, 4], [7, 2], [4, 4], [5, 2]])
            print("\nConfidence measure:")
            for datapoint in test_datapoints:
                probabilities = classifier.predict_proba([datapoint])[0]
                predicted_class = "Class-" + str(np.argmax(probabilities))
                print("\nDatapoint:", datapoint)
                print("Predicted class:", predicted_class)
            # Візуалізація точок даних
            visualize_classifier(
                    classifier, test_datapoints, [0] * len(test_datapoints), "Test
datapoints"
            )
            plt.show()
```

python random_forests.py --classifier-type rf

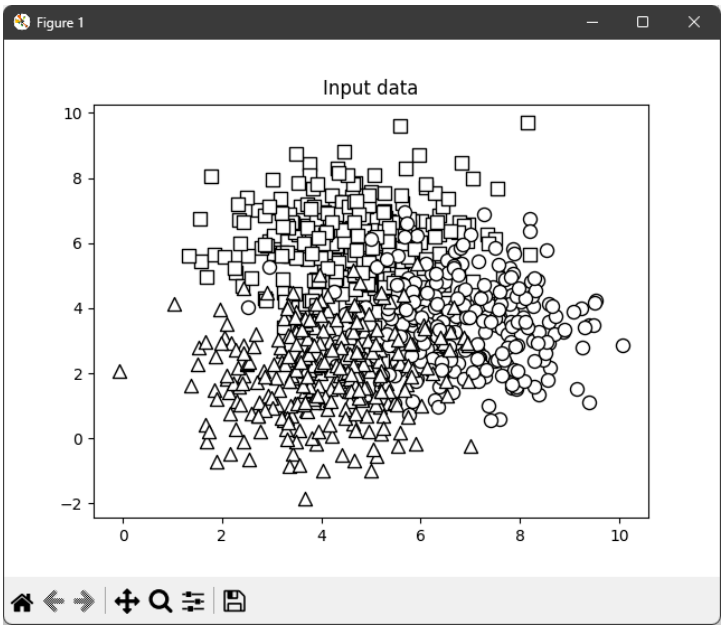

Рис. 1.1 – Результат виконання програми

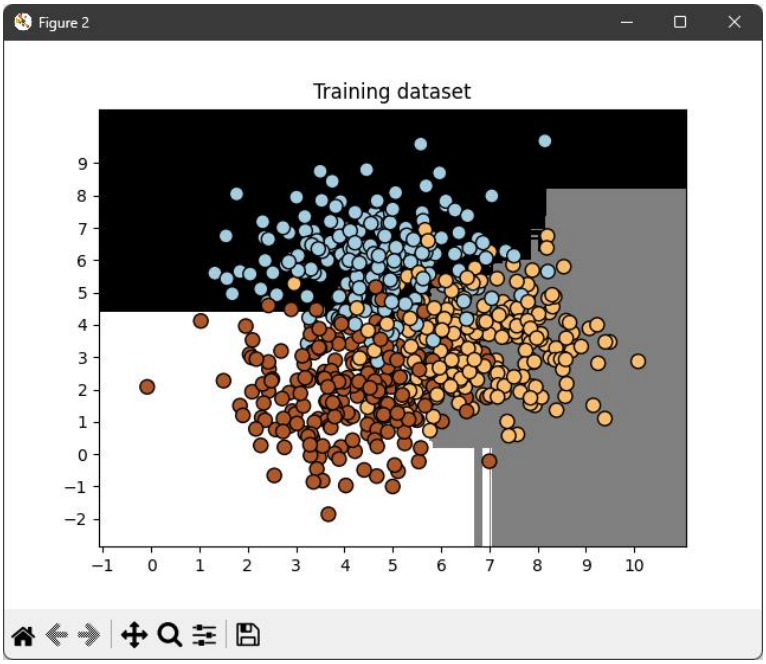

Рис. 1.2 – Результат виконання програми

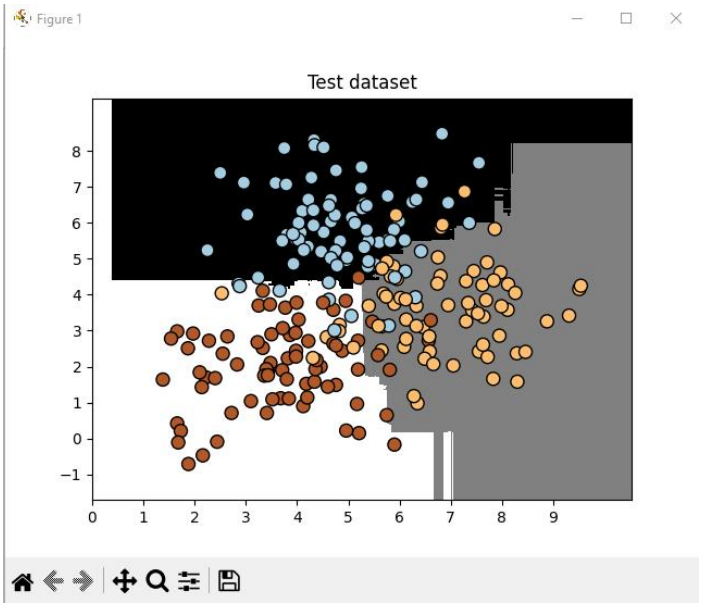

Рис. 1.3 – Результат виконання програми

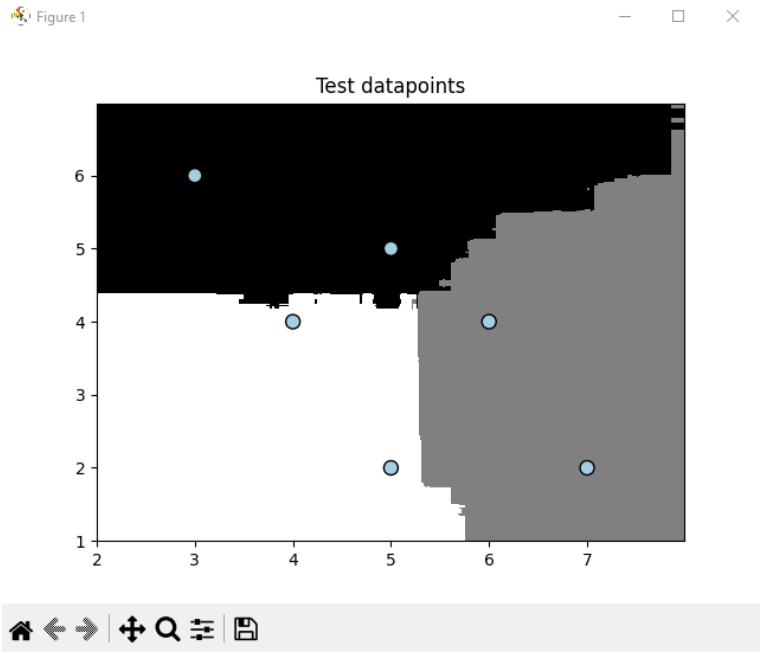

Рис. 1.4 – Результат виконання програми

python random_forests.py --classifier-type erf

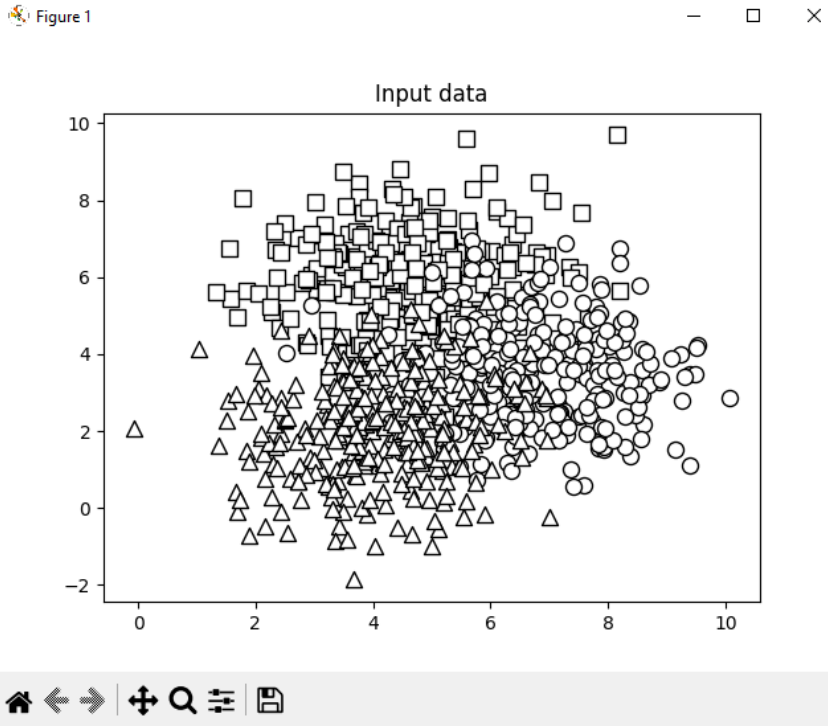

Рис. 1.5 – Результат виконання програми

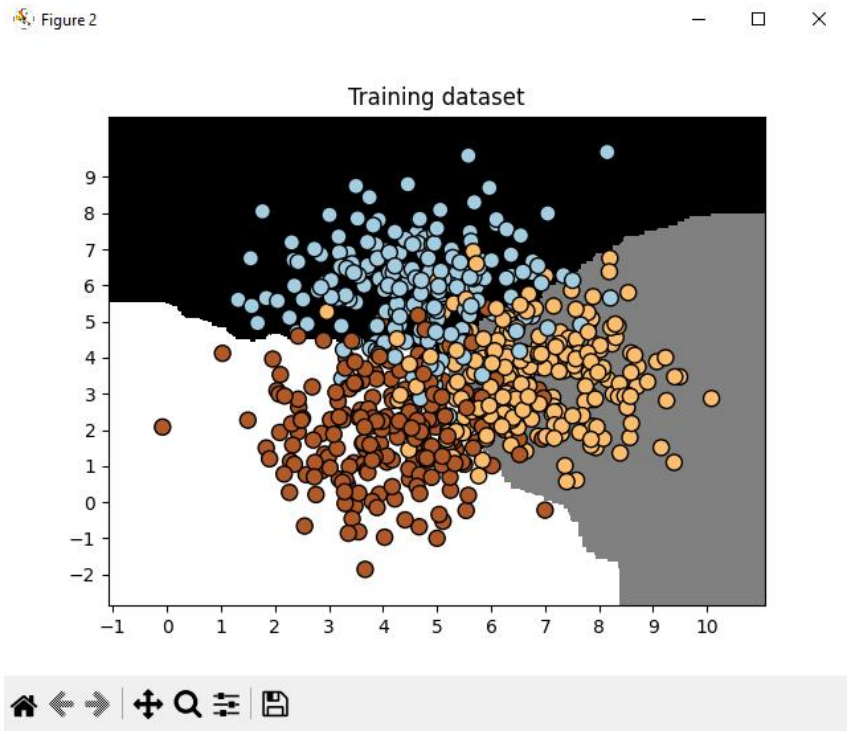

Рис. 1.6 – Результат виконання програми

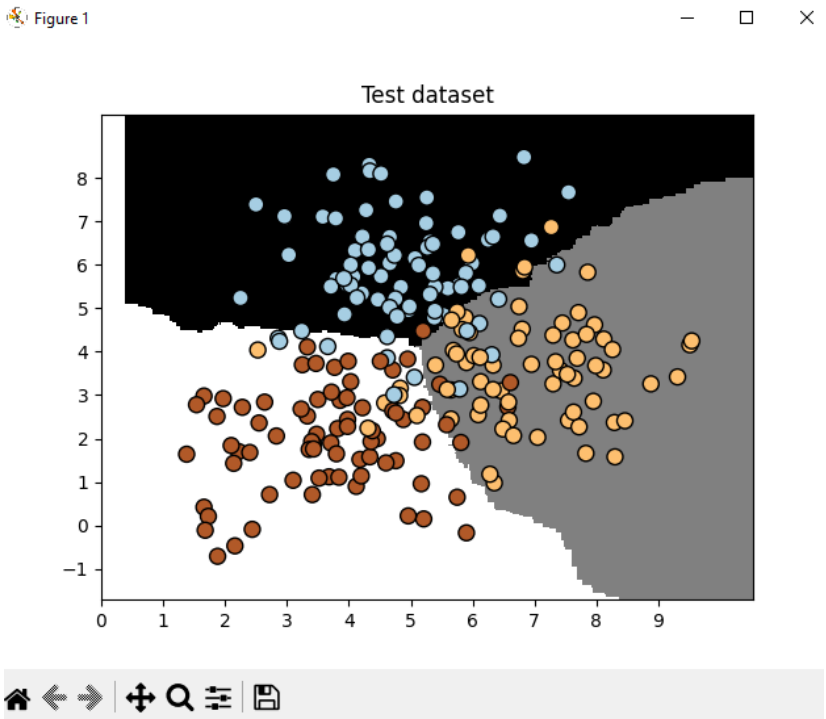

Рис. 1.7 – Результат виконання програми

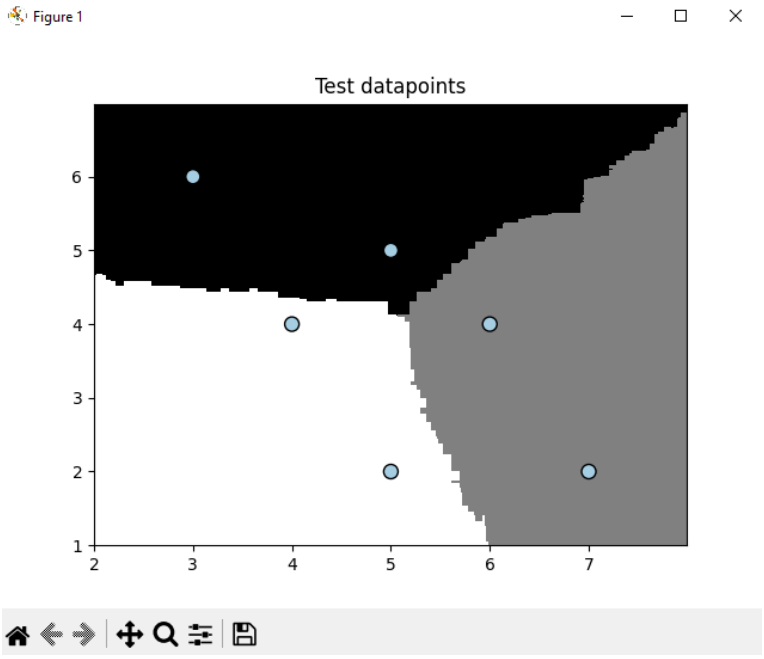

Рис. 1.8 – Результат виконання програми

Результат виконання після додавання масиву, що визначає тестові точки даних:

python random_forests.py --classifier-type rf

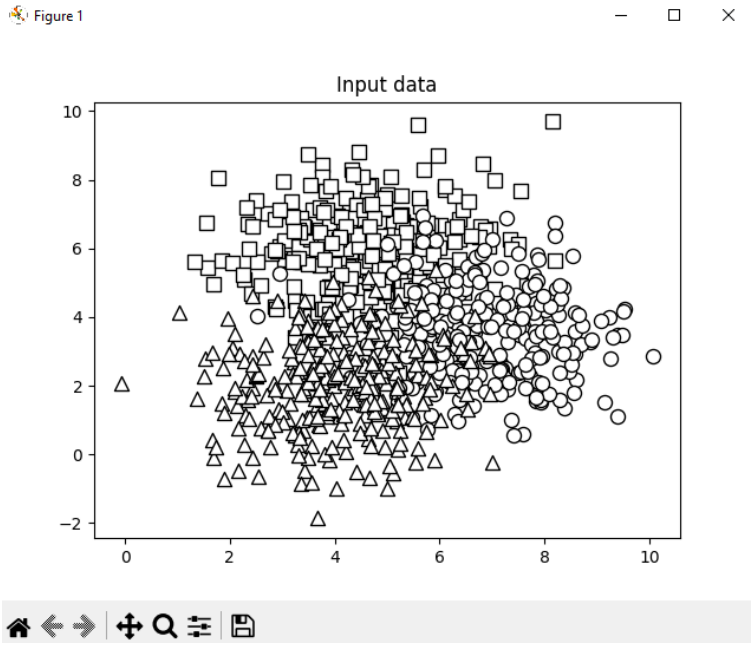

Рис. 1.9 – Результат виконання програми

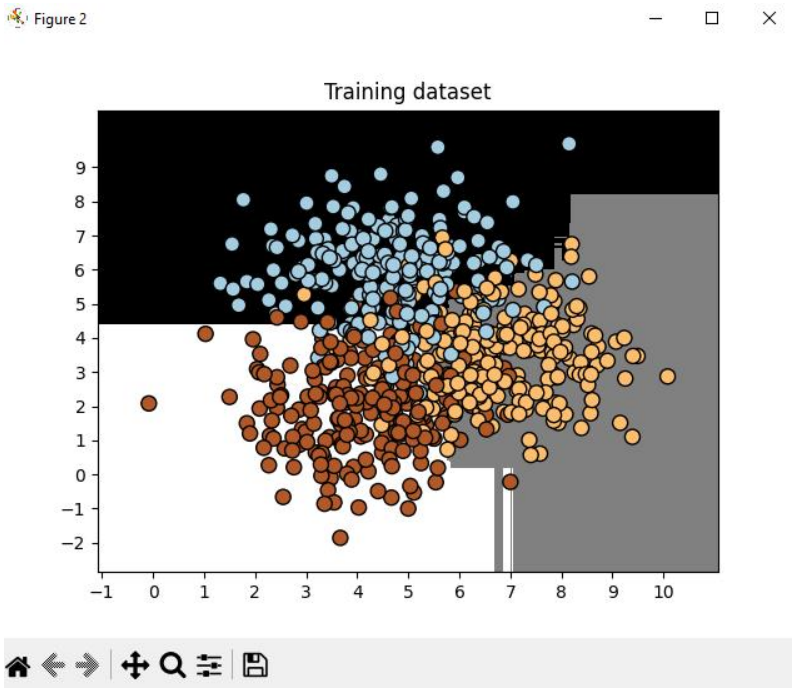

Рис. 1.10 – Результат виконання програми

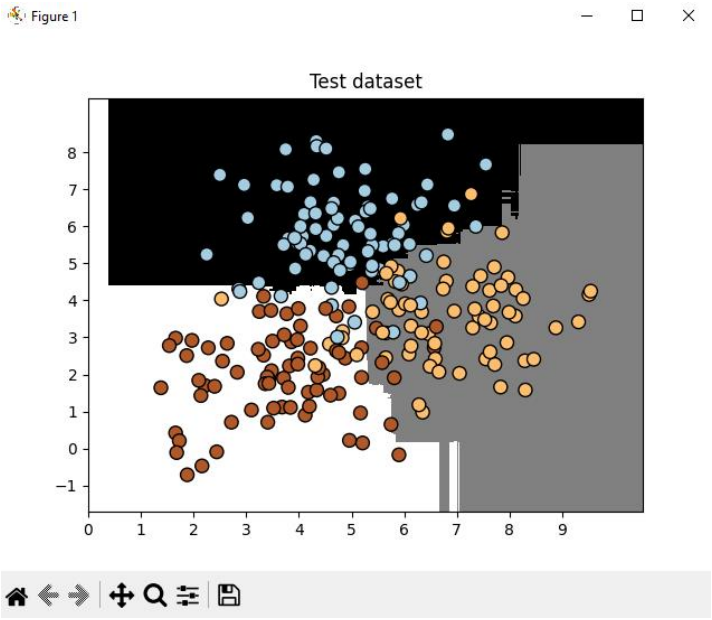

Рис. 1.11 – Результат виконання програми

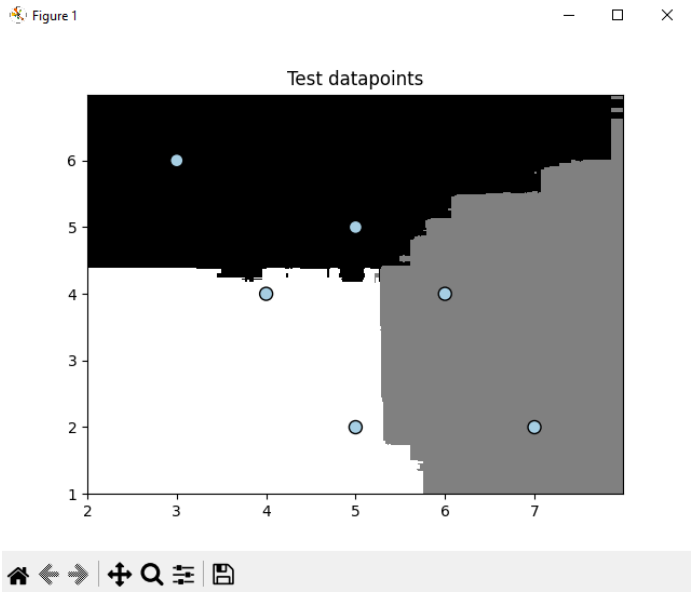

Рис. 1.12 – Результат виконання програми

```
Classifier performance on test dataset

              precision    recall  f1-score   support

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

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

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


Confidence measure:

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

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

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

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

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

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

Рис. 1.13 – Результат виконання програми

python random_forests.py --classifier-type erf

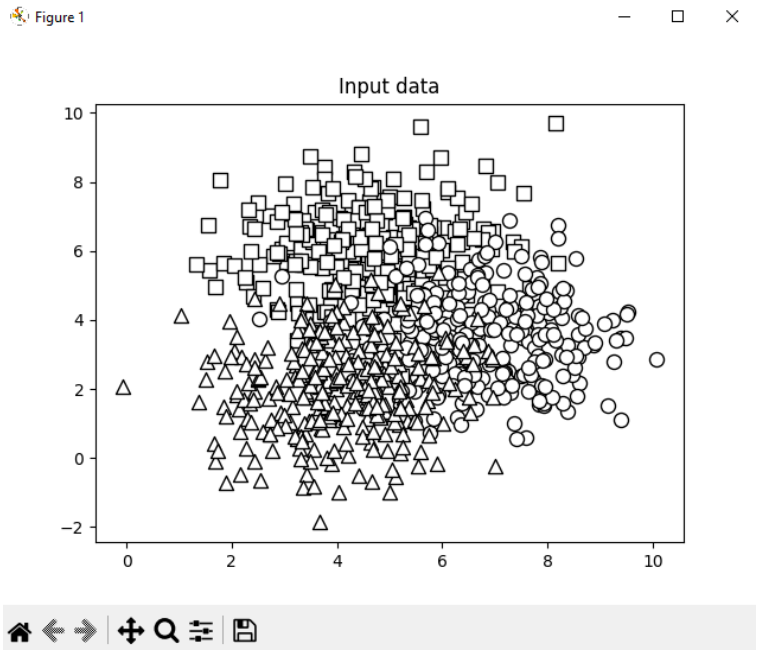

Рис. 1.14 – Результат виконання програми

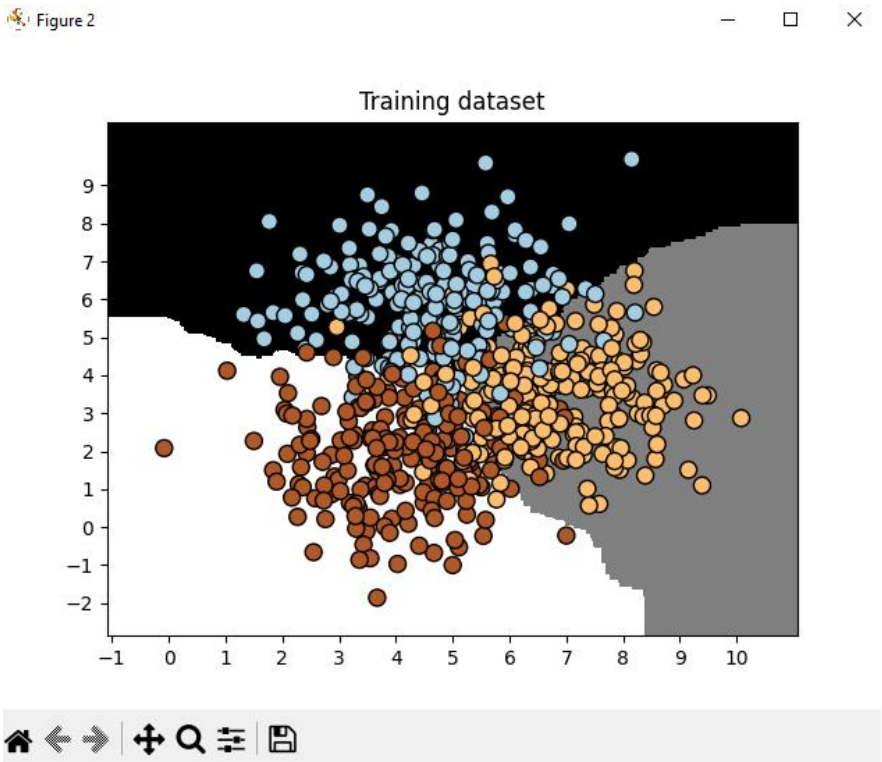

Рис. 1.15 – Результат виконання програми

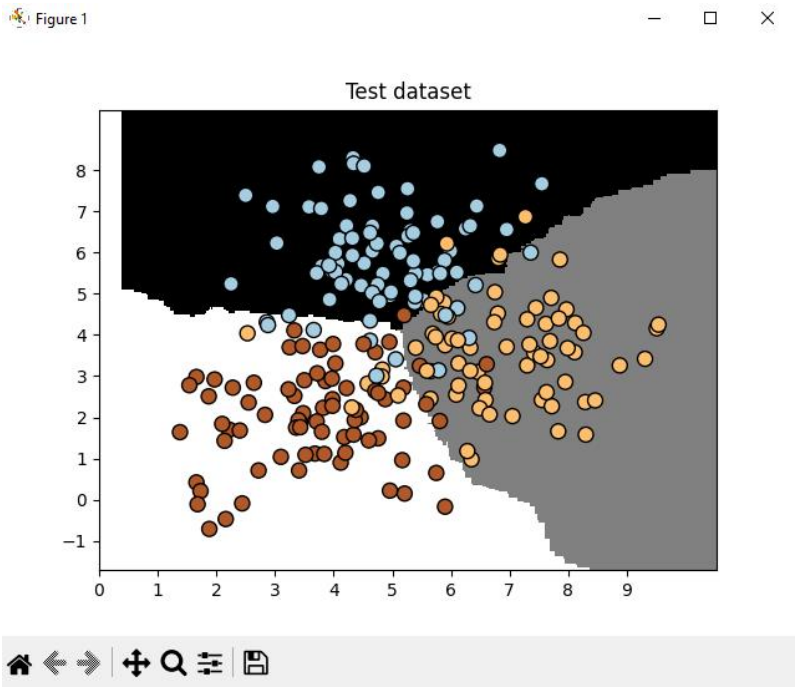

Рис. 1.16 – Результат виконання програми

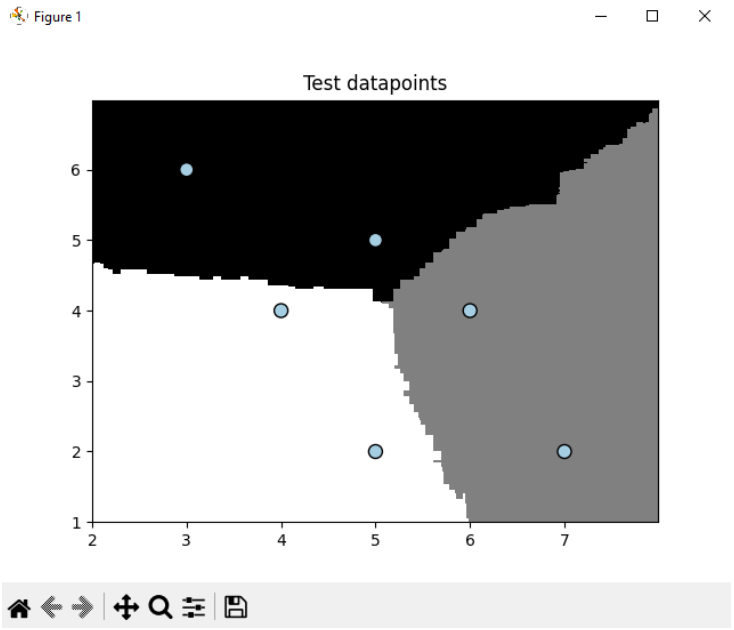

Рис. 1.17 – Результат виконання програми

```
Classifier performance on test dataset

              precision    recall  f1-score   support

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

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

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


Confidence measure:

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

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

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

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

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

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

Рис. 1.18 – Результат виконання програми

Під час використання -erf було отримано більш точні виміри. Це обумовлено тим, що в процесі навчання гранично випадкові ліси мають більше можливостей для вибору оптимальних дерев рішень, тому, як правило, вони забезпечують отримання кращих границь. Але кінцеві результати виявилися майже однаковими при використанні обох прапорців.

Завдання 2: Обробка дисбалансу класів.

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

# 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",
    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")

# Розбиття даних на навчальний та тестовий набори
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_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 = {
```

```
            "n_estimators": 100,
            "max_depth": 4,
            "random_state": 0,
            "class_weight": "balanced",
        }
    else:
        raise TypeError("Invalid input argument; should be 'balance'")

classifier = ExtraTreesClassifier(**params)
classifier.fit(X_train, y_train)
visualize_classifier(classifier, X_train, y_train, "Training dataset")

y_test_pred = classifier.predict(X_test)
visualize_classifier(classifier, X_test, y_test, "Test dataset")

# Обчислення показників ефективності класифікатора
class_names = ["Class-0", "Class-1"]
print("\n" + "#" * 40)
print("\nClassifier performance on training dataset\n")
print(
    classification_report(
        y_train, classifier.predict(X_train), target_names=class_names
    )
)
print("#" * 40 + "\n")

print("#" * 40)
print("\nClassifier performance on test dataset\n")
print(classification_report(y_test, y_test_pred, target_names=class_names))
print("#" * 40 + "\n")

plt.show()
```

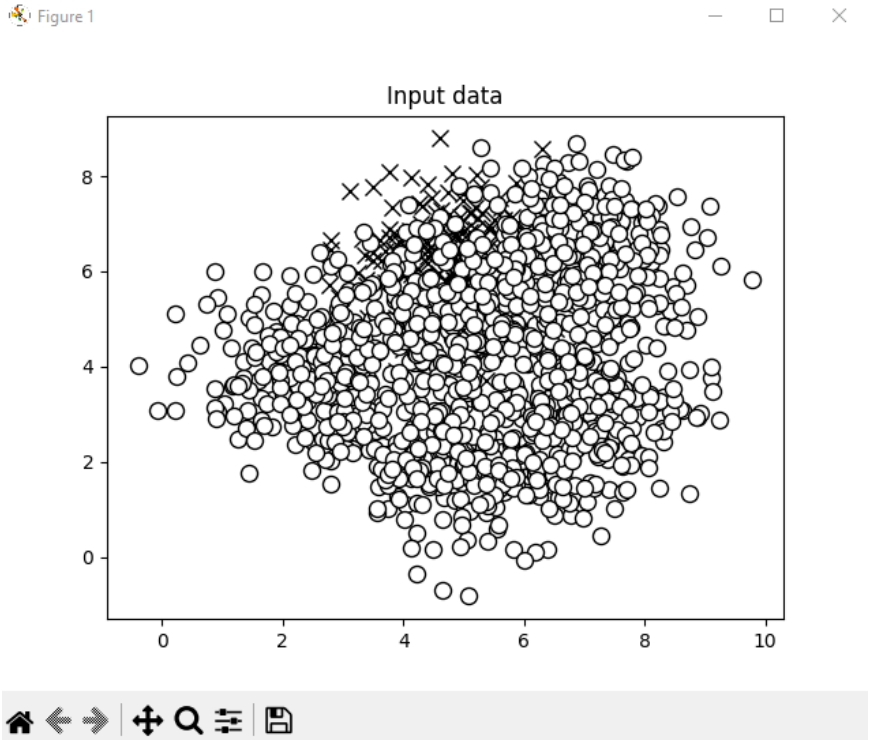

Рис. 2.1 – Результат виконання програми

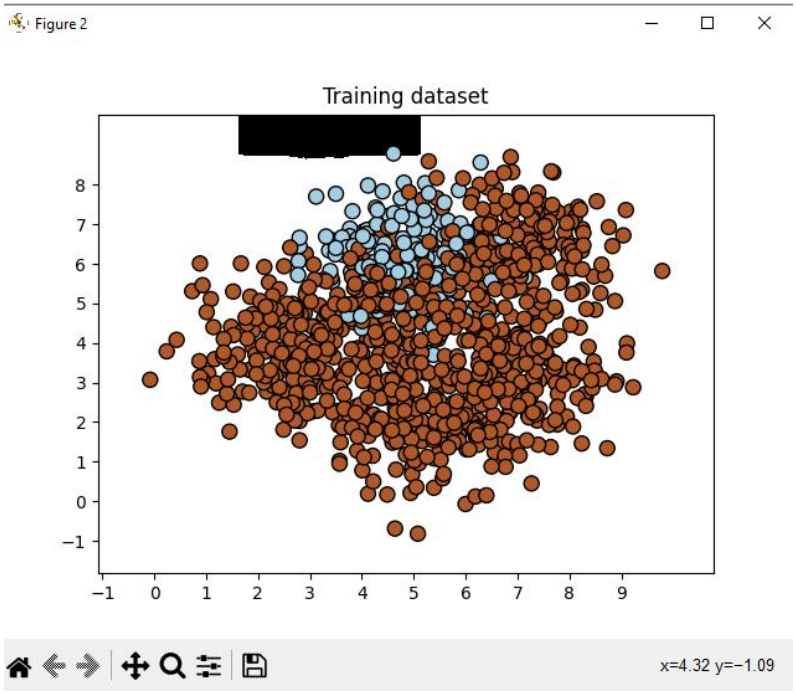

Рис. 2.2 – Результат виконання програми

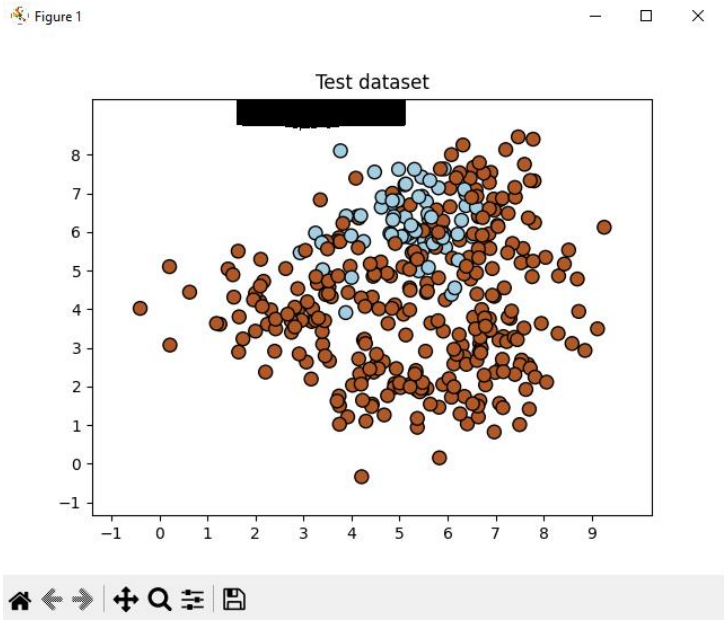

Рис. 2.3 – Результат виконання програми

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

Classifier performance on training dataset

              precision    recall  f1-score   support

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

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

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

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

Classifier performance on test dataset

              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. 2.4 – Результат виконання програми

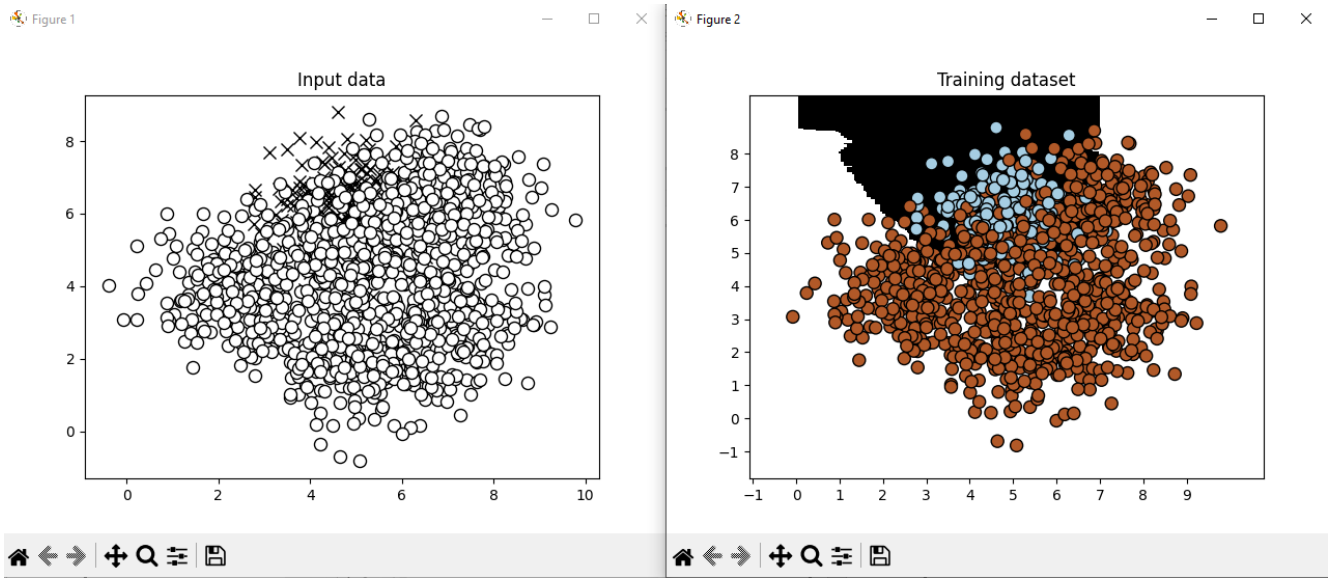

Рис. 2.5 – Результат виконання програми

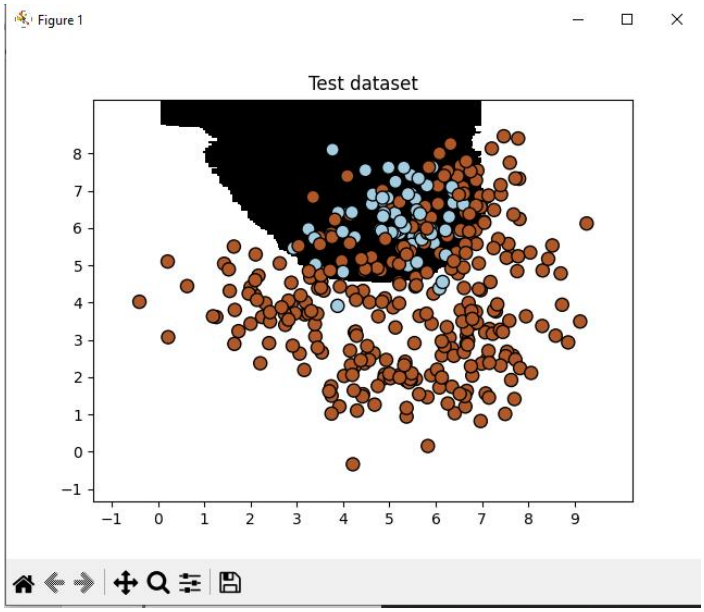

Рис. 2.6 – Результат виконання програми

```
Classifier performance on training dataset

              precision    recall  f1-score   support

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

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

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

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

Classifier performance on test dataset

              precision    recall  f1-score   support

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

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

Рис. 2.7 – Результат виконання програми

Обробка дисбалансу класів важлива при аналізі даних. Результати класифікації з врахуванням дисбалансу класів були кращими.

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

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

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, random_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("\n##### Searching optimal parameters for", metric)
            classifier = GridSearchCV(
                        ExtraTreesClassifier(random_state=0),   parameter_grid,   cv=5,
scoring=metric
            )
            classifier.fit(X_train, y_train)
            df = pd.DataFrame(classifier.cv_results_)
            df_columns_to_print = [
                 column for column in df.columns if "param" in column or "score" in
column
            ]
            print(df[df_columns_to_print])
            print("\nBest parameters:", classifier.best_params_)
            y_pred = classifier.predict(X_test)
            print("\nPerformance report:\n")
            print(classification_report(y_test, y_pred))
```

```
##### Searching optimal parameters for precision_weighted
   mean_score_time  std_score_time  ... std_test_score rank_test_score
0         0.006228        0.000979  ...       0.025132              1
1         0.006082        0.001520  ...       0.023032              5
2         0.009050        0.002521  ...       0.026560              4
3         0.006882        0.000458  ...       0.028334              8
4         0.007861        0.000775  ...       0.033429              9
5         0.002607        0.000487  ...       0.029698              2
6         0.003388        0.000806  ...       0.020401              7
7         0.006037        0.001131  ...       0.023032              5
8         0.010659        0.001099  ...       0.026867              3

[9 rows x 13 columns]

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

Performance report:

              precision    recall  f1-score   support

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

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

Рис. 3.1 – Результат виконання програми

```
##### Searching optimal parameters for recall_weighted
   mean_score_time  std_score_time  ... std_test_score rank_test_score
0         0.006192        0.001932  ...       0.027075              1
1         0.005864        0.001481  ...       0.022468              5
2         0.008252        0.002657  ...       0.026749              3
3         0.006297        0.000600  ...       0.028497              8
4         0.006400        0.000800  ...       0.034744              9
5         0.001805        0.000380  ...       0.029407              1
6         0.003805        0.002143  ...       0.020096              7
7         0.005692        0.001323  ...       0.022468              5
8         0.014734        0.002682  ...       0.026749              3

[9 rows x 13 columns]

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

Performance report:

              precision    recall  f1-score   support

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

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

Рис. 3.2 – Результат виконання програми

Ціль сіткового пошуку - знайти набір параметрів, які максимізують вибрану метрику класифікації на тестовому наборі даних. Комбінації значень па-

раметрів відрізняються між собою, оскільки precision і recall - різні метричні характеристики, що вимагають використання різних комбінацій параметрів.

Завдання 4: Обчислення відносної важливості ознак.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.datasets import load_boston
from sklearn.metrics import mean_squared_error, explained_variance_score
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle

# Завантаження даних із цінами на нерухомість
housing_data = load_boston()
# Перемішування даних
X, y = shuffle(housing_data.data, housing_data.target, random_state=7)
# Розбиття даних на навчальний та тестовий набори
X_train,   X_test,   y_train,   y_test   =   train_test_split(X,   y,   test_size=0.2,
random_state=7)
# Модель на основі регресора AdaBoost
regressor = AdaBoostRegressor(
    DecisionTreeRegressor(max_depth=4), n_estimators=400, random_state=7
)
regressor.fit(X_train, y_train)

# Обчислення показників ефективності регресора AdaBoost
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))
# Вилучення важливості ознак
feature_importances = regressor.feature_importances_
feature_names = housing_data.feature_names
# Нормалізація значень важливості ознак
feature_importances = 100.0 * (feature_importances / max(feature_importances))
# Сортування та перестановка значень
index_sorted = np.flipud(np.argsort(feature_importances))
# Розміщення міток уздовж осі X
pos = np.arange(index_sorted.shape[0]) + 0.5
# Побудова стовпчастої діаграми
plt.figure()
plt.bar(pos, feature_importances[index_sorted], align="center")
plt.xticks(pos, feature_names[index_sorted])
plt.ylabel("Relative Importance")
plt.title("Feature importance using AdaBoost regressor")
plt.show()
```

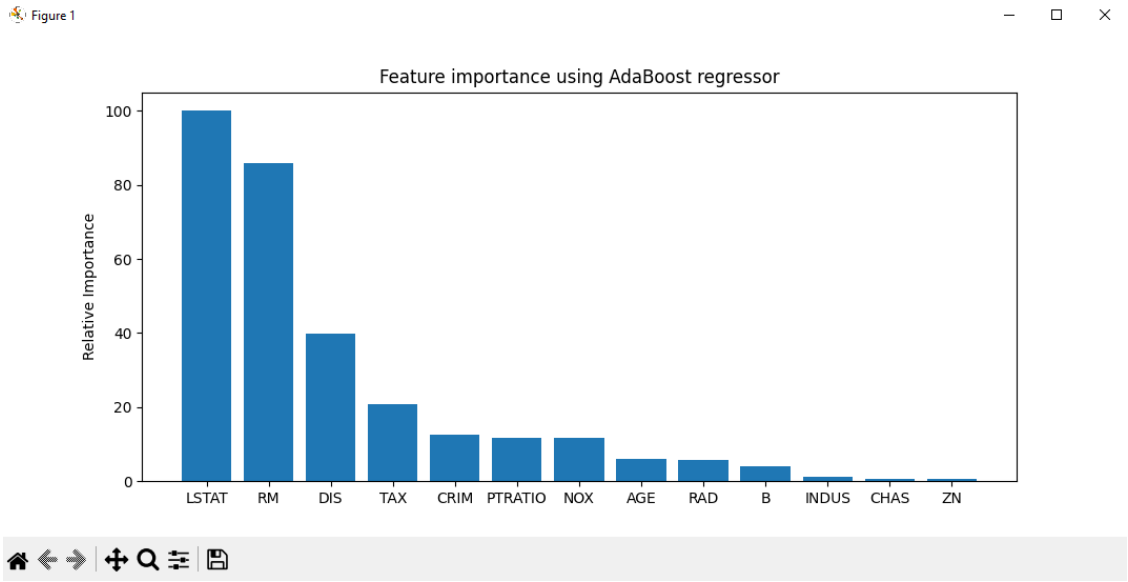

Рис. 4.1 – Результат виконання програми

```
ADABOOST REGRESSOR
Mean squared error = 22.7
Explained variance score = 0.79
```

Рис. 4.2 – Результат виконання програми

Ознаки RM, LSTAT і NOX є ключовими факторами при прогнозуванні. CRIM і AGE мають найменший вплив і можуть бути менш важливими при прогнозуванні.

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

```
import numpy as np
from sklearn.metrics import mean_absolute_error
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.ensemble import ExtraTreesRegressor

# Завантаження вхідних даних
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)

    # Перетворення рядкових даних на числові
    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)

    # Розбиття даних на навчальний та тестовий набори
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.25, random_state=5
    )

    # Регресор на основі гранично випадкових лісів
    params = {"n_estimators": 100, "max_depth": 4, "random_state": 0}
    regressor = ExtraTreesRegressor(**params)
    regressor.fit(X_train, y_train)

    # Обчислення характеристик ефективності регресора на тестових даних
    y_pred = regressor.predict(X_test)
    print("Mean absolute error:", round(mean_absolute_error(y_test, y_pred), 2))

    # Тестування кодування на одиночному прикладі
    test_datapoint = ["Saturday", "10:20", "Atlanta", "no"]
    test_datapoint_encoded = [-1] * len(test_datapoint)
    count = 0
    for i, item in enumerate(test_datapoint):
        if item.isdigit():
            test_datapoint_encoded[i] = int(test_datapoint[i])
        else:
            test_datapoint_encoded[i] = int(
```

```
                label_encoder[count].transform([test_datapoint[i]])
            )
            count = count + 1


    test_datapoint_encoded = np.array(test_datapoint_encoded)


    # Прогнозування результату для тестової точки даних
    print("Predicted traffic:", int(regressor.predict([test_datapoint_encoded])[0]))
```

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

Рис. 5.1 – Результат виконання програми

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

# Генерація 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=9)

# 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
output = processor_pipeline.predict(X)
print("\nPredicted output:\n", output)
# Виведення оцінки
print("\nScore:", 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("\nIndices of selected features:", ", ".join([str(x) for x in selected]))
```

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

Score: 0.8866666666666667

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

Рис. 6.1 – Результат виконання програми

- В першому абзаці представлено спрогнозовані вихідні мітки за допомогою конвеєра.
- Значення Score відображає ефективність конвеєра.
- В останньому абзаці представлено індекси вибраних ознак.

Завдання 7: Пошук найближчих сусідів.

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

# Вхідні дані
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],
        ]
    )

    # Кількість найближчих сусідів
    k = 5
    # Тестова точка даних
    test_datapoint = [4.3, 2.7]

    # Відображення вхідних даних на графіку
    plt.figure()
    plt.title("Input data")
    plt.scatter(X[:, 0], X[:, 1], marker="o", s=75, color="black")

    # Побудова моделі на основі методу k найближчих сусідів
    knn_model = NearestNeighbors(n_neighbors=k, algorithm="ball_tree").fit(X)
    distances, indices = knn_model.kneighbors([test_datapoint])

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

    # Візуалізація найближчих сусідів разом із тестовою точкою даних
    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()
```

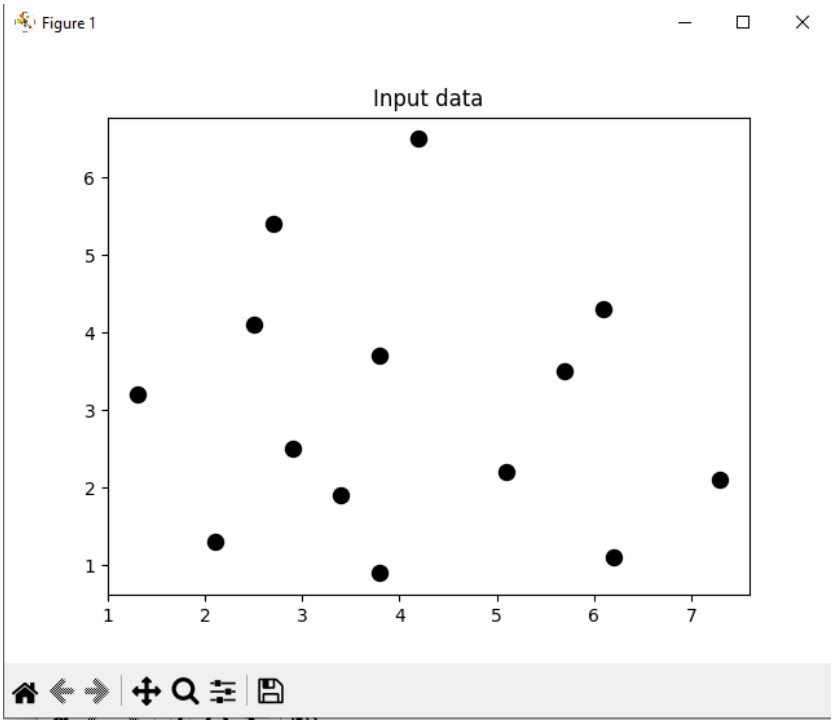

Рис. 7.1 – Результат виконання програми

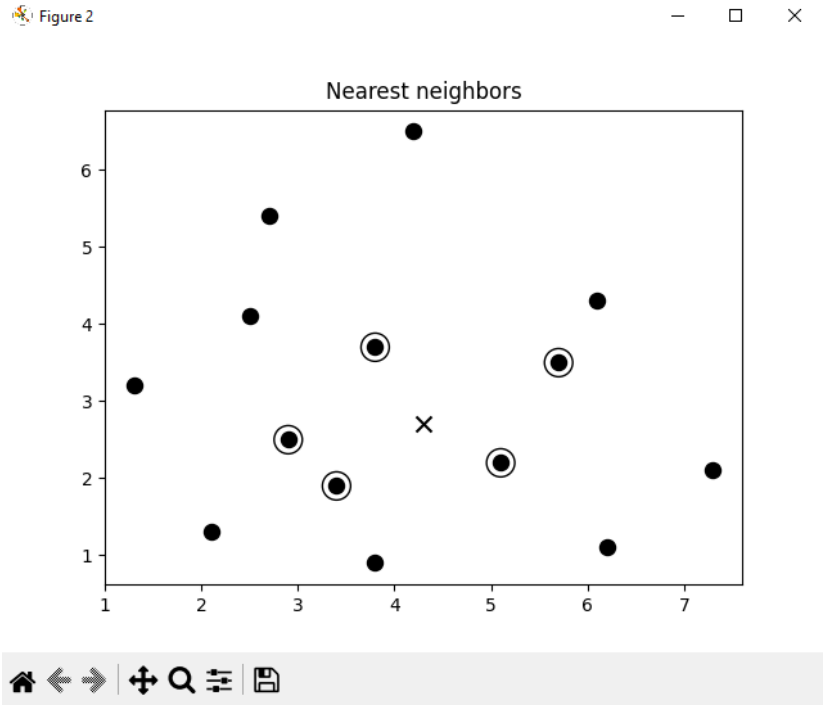

Рис. 7.2 – Результат виконання програми

```
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. 7.3 – Результат виконання програми

- Перший графік – це вхідні дані
- Другий графік – це вхідні дані, тестова точка та її 5 найближчих сусідів (обведені)
- Вікно терміналу – 5 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

# Завантаження вхідних даних
input_file = "data.txt"
data = np.loadtxt(input_file, delimiter=",")
X, y = data[:, :-1], data[:, -1].astype(np.int)

# Відображення вхідних даних на графіку
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"
    )

# Кількість найближчих сусідів
num_neighbors = 12
# Розмір кроку сітки візуалізації
```

```
    step_size = 0.01
    # Створення класифікатора на основі методу k найближчих сусідів
    classifier = neighbors.KNeighborsClassifier(num_neighbors, weights="distance")

    # Навчання моделі на основі методу k найближчих сусідів
    classifier.fit(X, y)

    # Створення сітки для відображення меж на графіку
    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)
    )
    # Виконання класифікатора на всіх точках сітки
    output = classifier.predict(np.c_[x_values.ravel(), y_values.ravel()])

    # Візуалізація передбачуваного результату
    output = output.reshape(x_values.shape)
    plt.figure()
    plt.pcolormesh(x_values, y_values, output, cmap=cm.Paired)

    # Накладання навчальних точок на карту
    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")

    # Тестування вхідної точки даних
    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",
        )

        # Вилучення K найближчих сусідів
        _, indices = classifier.kneighbors([test_datapoint])
        indices = indices.astype(np.int)[0]

        # Відображення K найближчих сусідів на графіку
        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()
```

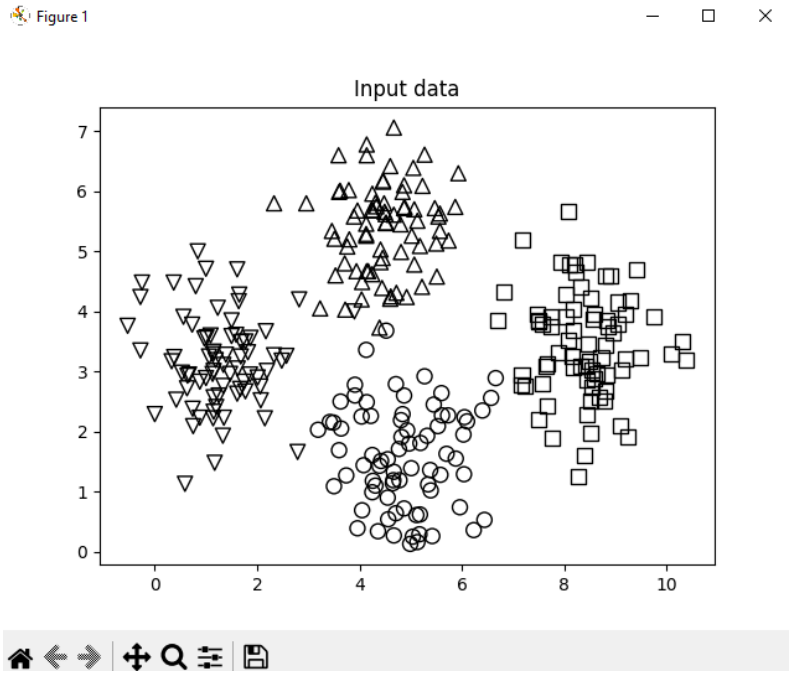

Рис. 8.1 – Результат виконання програми

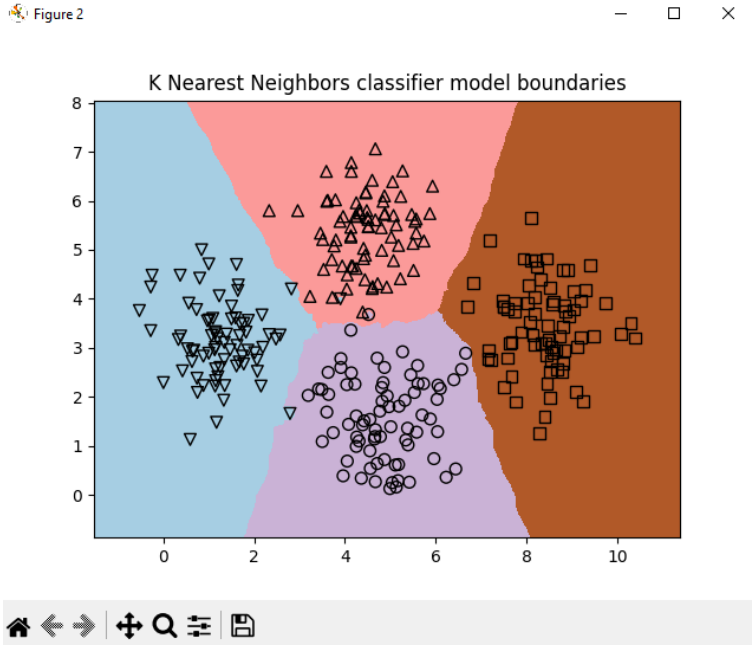

Рис. 8.2 – Результат виконання програми

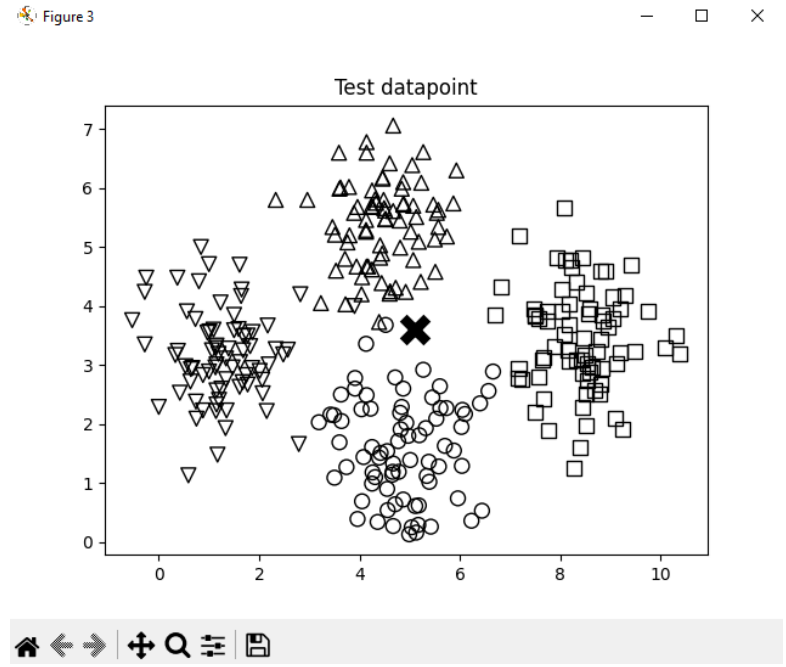

Рис. 8.3 – Результат виконання програми

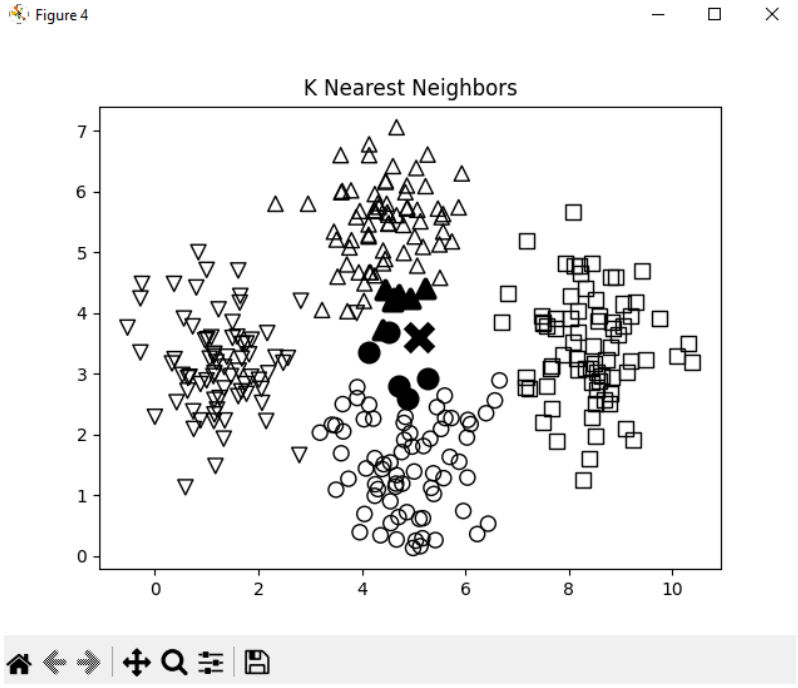

Рис. 8.4 – Результат виконання програми

- Перший скрін – це вхідні дані
- Другий - це межі класифікатора
- Третій - це тестова точка до вхідного набору даних
- Четвертий – це 12 найближчих сусідів
- Тестова точка – це 1 клас

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]   -
dataset[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 common_movies]
        )
        user2_squared_sum = np.sum(
            [np.square(dataset[user2][item]) for item in common_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))
```

python LR_4_task_9.py --user1 "David Smith" --user2 "Bill Duffy" --score-type Euclidean

```
PS C:\Users\38098\Desktop\4c\python\lab4> python LR_4_task_9.py --user1 "David Smith" --user2 "Bill Duffy" --score-type Euclidean

Euclidean score:
0.585786437626905
```

Рис. 9.1 – Результат виконання програми

python LR_4_task_9.py --user1 "David Smith" --user2 "Bill Duffy" --score-type Pearson

```
PS C:\Users\38098\Desktop\4c\python\lab4> python LR_4_task_9.py --user1 "David Smith" --user2 "Bill Duffy" --score-type Pearson

Pearson score:
0.9909924304103233
```

Рис. 9.2 – Результат виконання програми

```
Pearson score:
0.9909924304103233
PS C:\Users\38098\Desktop\4c\python\lab4> python LR_4_task_9.py --user1 "David Smith" --user2 "Brenda Peterson" --score-type Euclidean

Euclidean score:
0.1424339656566283
PS C:\Users\38098\Desktop\4c\python\lab4> python LR_4_task_9.py --user1 "David Smith" --user2 "Samuel Miller" --score-type Euclidean

Euclidean score:
0.30383243470068705
PS C:\Users\38098\Desktop\4c\python\lab4> python LR_4_task_9.py --user1 "David Smith" --user2 "Julie Hammel" --score-type Euclidean

Euclidean score:
0.2857142857142857
PS C:\Users\38098\Desktop\4c\python\lab4> python LR_4_task_9.py --user1 "David Smith" --user2 "Clarissa Jackson" --score-type Euclidean

Euclidean score:
0.28989794855663564
PS C:\Users\38098\Desktop\4c\python\lab4> python LR_4_task_9.py --user1 "David Smith" --user2 "Adam Cohen" --score-type Euclidean

Euclidean score:
0.38742588672279304
PS C:\Users\38098\Desktop\4c\python\lab4> python LR_4_task_9.py --user1 "David Smith" --user2 "Chris Duncan" --score-type Euclidean

Euclidean score:
0.38742588672279304
```

Рис. 9.3 – Результат виконання програми

```
PS C:\Users\38098\Desktop\4c\python\lab4> python LR_4_task_9.py --user1 "David Smith" --user2 "Brenda Peterson" --score-type Pearson

Pearson score:
-0.7236759610155113
PS C:\Users\38098\Desktop\4c\python\lab4> python LR_4_task_9.py --user1 "David Smith" --user2 "Samuel Miller" --score-type Pearson

Pearson score:
0.7587869106393281
PS C:\Users\38098\Desktop\4c\python\lab4> python LR_4_task_9.py --user1 "David Smith" --user2 "Julie Hammel" --score-type Pearson

Pearson score:
0
PS C:\Users\38098\Desktop\4c\python\lab4> python LR_4_task_9.py --user1 "David Smith" --user2 "Clarissa Jackson" --score-type Pearson

Pearson score:
0.6944217062199275
PS C:\Users\38098\Desktop\4c\python\lab4> python LR_4_task_9.py --user1 "David Smith" --user2 "Adam Cohen" --score-type Pearson

Pearson score:
0.9081082718950217
PS C:\Users\38098\Desktop\4c\python\lab4> python LR_4_task_9.py --user1 "David Smith" --user2 "Clarissa Jackson" --score-type Pearson

Pearson score:
0.6944217062199275
```

Рис. 9.4 – Результат виконання програми

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

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

```
import argparse
import json
import numpy as np
from LR_4_task_9 import pearson_score

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

# Знаходження користувачів у наборі даних, схожих на введеного користувача
def find_similar_users(dataset, user, num_users):
    if user not in dataset:
        raise TypeError("Cannot find " + user + " in the dataset")
    # Обчислення оцінки подібності за Пірсоном між
    # вказаним користувачем та всіма іншими
```

```
        # користувачами в наборі даних
        scores = np.array(
            [[x, pearson_score(dataset, user, x)] for x in dataset if x != user]
        )
        # Сортування оцінок за спаданням
        scores_sorted = np.argsort(scores[:, 1])[::-1]
        # Вилучення оцінок перших 'num_users' користувачів
        top_users = scores_sorted[:num_users]
        return scores[top_users]

    if __name__ == "__main__":
        args = build_arg_parser().parse_args()
        user = args.user
        ratings_file = "ratings.json"
        with open(ratings_file, "r") as f:
            data = json.loads(f.read())
        print("\nUsers similar to " + user + ":\n")
        similar_users = find_similar_users(data, user, 3)
        print("User\t\t\tSimilarity score")
        print("-" * 41)
        for item in similar_users:
            print(item[0], "\t\t", round(float(item[1]), 2))
```

python LR_4_task_10.py --user "Bill Duffy"

```
PS C:\Users\38098\Desktop\4c\python\lab4> python 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
```

Рис. 10.1 – Результат виконання програми

```
PS C:\Users\38098\Desktop\4c\python\lab4> python 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
```

Рис. 10.2 – Результат виконання програми

Користувач Clarissa Jackson має одинакові вподобання з користувачем Chris Duncan. Користувач David Smith має майже однакові вподобання з Bill Duffy.

Завдання 11: Створення рекомендаційної системи фільмів.

```
import argparse
import json
import numpy as np
from LR_4_task_9 import pearson_score
from LR_4_task_10 import find_similar_users

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

# Отримання рекомендації щодо фільмів для вказаного користувача
def get_recommendations(dataset, input_user):
    if input_user not in dataset:
        raise TypeError("Cannot find " + input_user + " in the dataset")

    overall_scores = {}
    similarity_scores = {}

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

```
            if similarity_score <= 0:
                continue
            filtered_list = [
                x
                for x in dataset[user]
                if x not in dataset[input_user] or dataset[input_user][x] == 0
            ]
            for item in filtered_list:
                overall_scores.update({item: dataset[user][item] * similarity_score})
                similarity_scores.update({item: similarity_score})

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

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

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

    # Вилучення рекомендацій фільмів
    movie_recommendations = [movie for _, movie in movie_scores]
    return movie_recommendations

if __name__ == "__main__":
    args = build_arg_parser().parse_args()
    user = args.user
    ratings_file = "ratings.json"
    with open(ratings_file, "r") as f:
        data = json.loads(f.read())
    print("\nMovie recommendations for " + user + ":")
    movies = get_recommendations(data, user)
    for i, movie in enumerate(movies):
        print(str(i + 1) + ". " + movie)
```

python LR_4_task_11.py --user "Chris Duncan"

```
PS C:\Users\38098\Desktop\4c\python\lab4> python LR_4_task_11.py --user "Chris Duncan"

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

Рис. 11.1 – Результат виконання програми

```
PS C:\Users\38098\Desktop\4c\python\lab4> python LR_4_task_11.py --user "Julie Hammel"

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

Рис. 11.2 – Результат виконання програми

Для Julie Hammel і Chris Duncan були знайдені рекомендації фільмів на основі схожості з іншими користувачами, використовуючи коефіцієнт кореляції Пірсона.

Висновок: в ході виконання лабораторної роботи було використано спеціалізовані бібліотеки та мову програмування Python для дослідження методів ансамблів у машинному навчанні та створення рекомендаційних системи.