ЛАБОРАТОРНА РОБОТА № 5
ДОСЛІДЖЕННЯ МЕТОДІВ АНСАМБЛЕВОГО НАВЧАННЯ
Варіант 13
Хід роботи:

Завдання 1: Використовувати файл вхідних даних: data_random_forests.txt, побудувати класифікатори на основі випадкових та гранично випадкових лісів.

Код програми:

```
import argparse

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

from utilities import visualize_classifier


# Argument parser
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__":
    # Parse the input arguments
    args = build_arg_parser().parse_args()
    classifier_type = args.classifier_type

    # Load input data
    input_file = " Лабораторна робота 5/data_random_forests.txt"
    data = np.loadtxt(input_file, delimiter=",")
    X, y = data[:, :-1], data[:, -1]
```

```
    # Separate input data into three classes based on labels
    class_0 = np.array(X[y == 0])
    class_1 = np.array(X[y == 1])
    class_2 = np.array(X[y == 2])

    # Visualize input data
    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")

    # Split data into training and testing datasets
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.25, random_state=5
    )

    # Ensemble Learning classifier
    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")

    # Evaluate classifier performance
    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")

    # Compute confidence
    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)

    # Visualize the datapoints
    visualize_classifier(
        classifier, test_datapoints, [0] * len(test_datapoints), "Test datapoints"
    )

    plt.show()
```

Виконання:

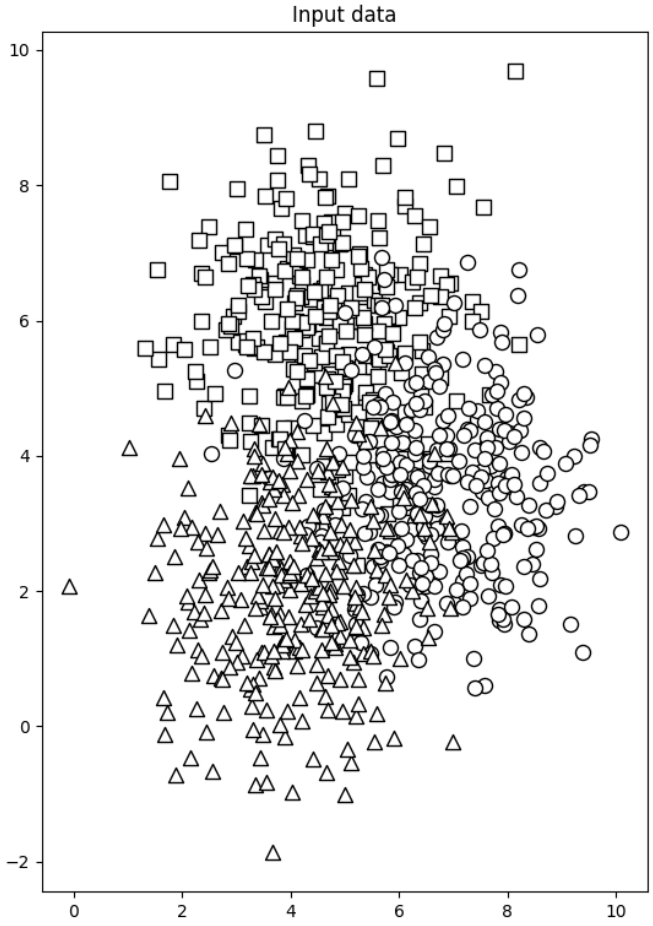

Рисунок 1.1 – Графік тестових даних

Виконуємо код з прапорцем *rf:*

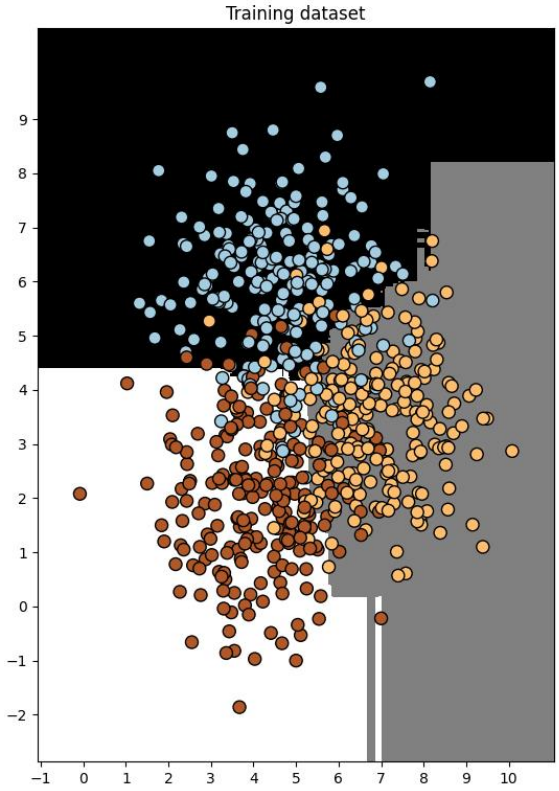

Рисунок 1.2 – Тренувальні дані

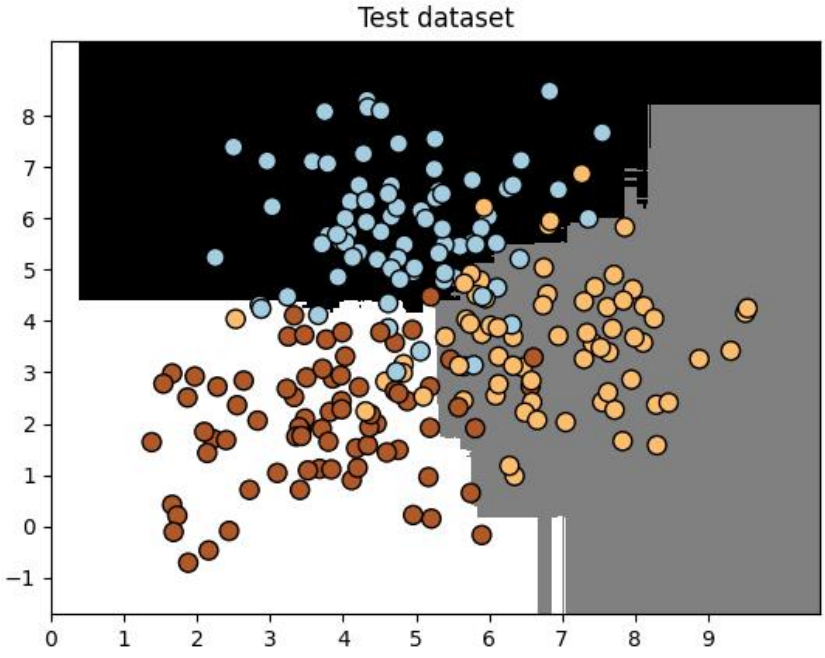

Рисунок 1.3 – Тестові дані

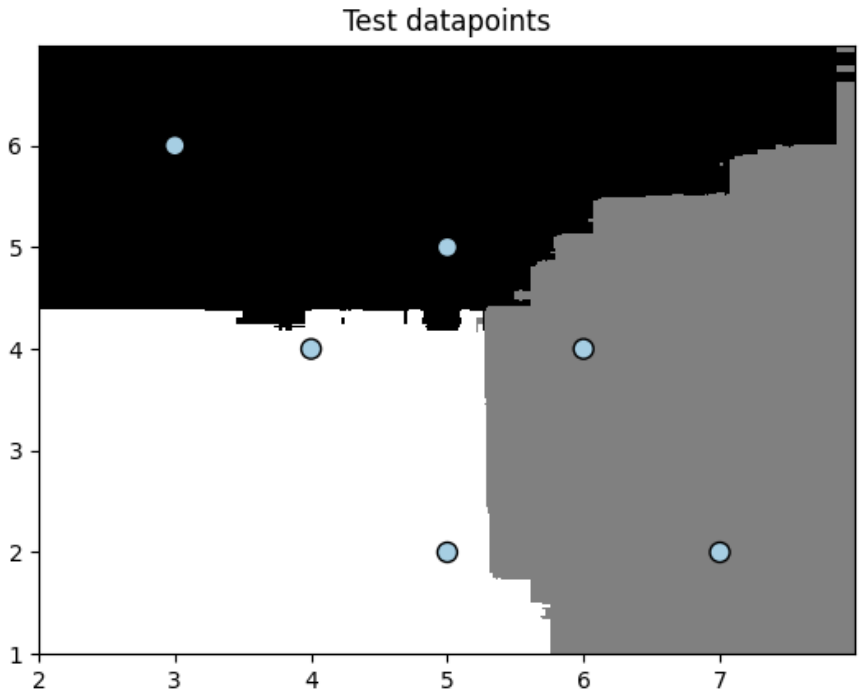

Рисунок 1.4 – Тестові точки даних

```
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
```

Рисунок 1.5 – Міра достовірності

Виконуємо код з прапорцем *erf:*

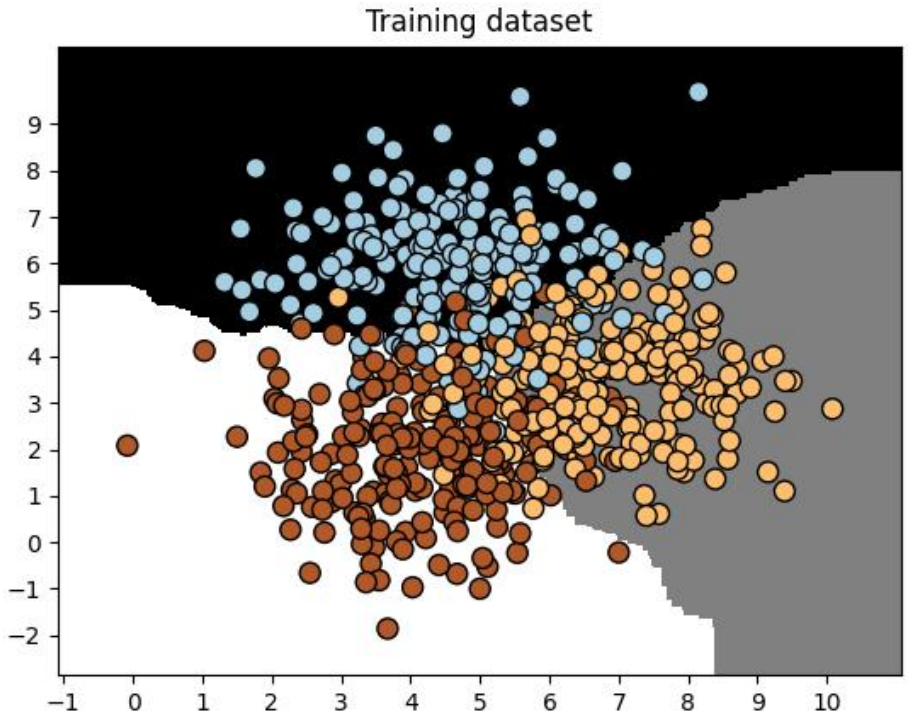

Рисунок 1.6 – Тренувальні дані

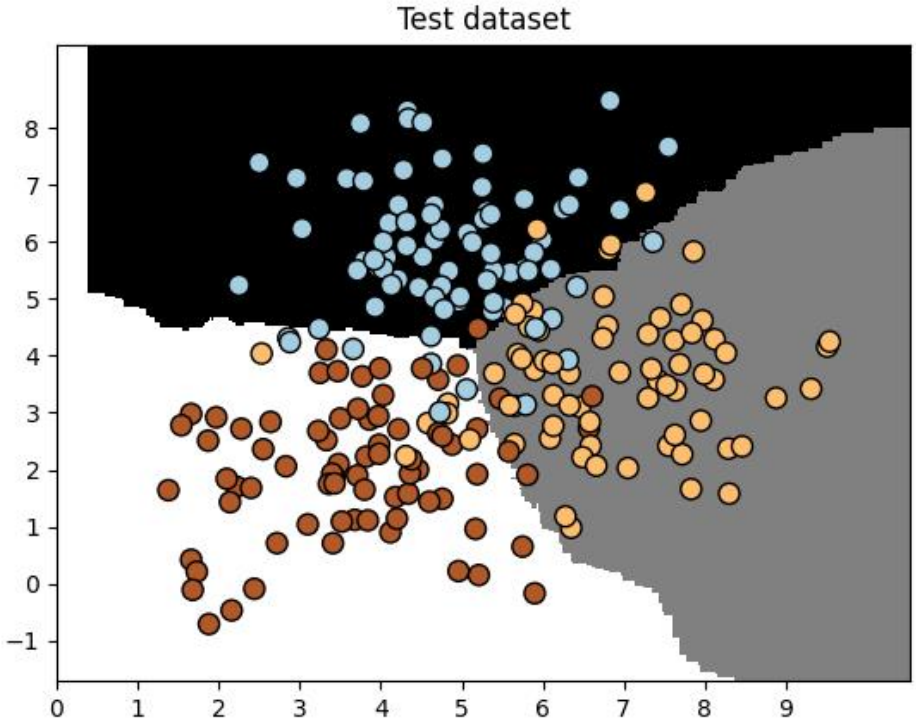

Рисунок 1.7 – Тестові дані

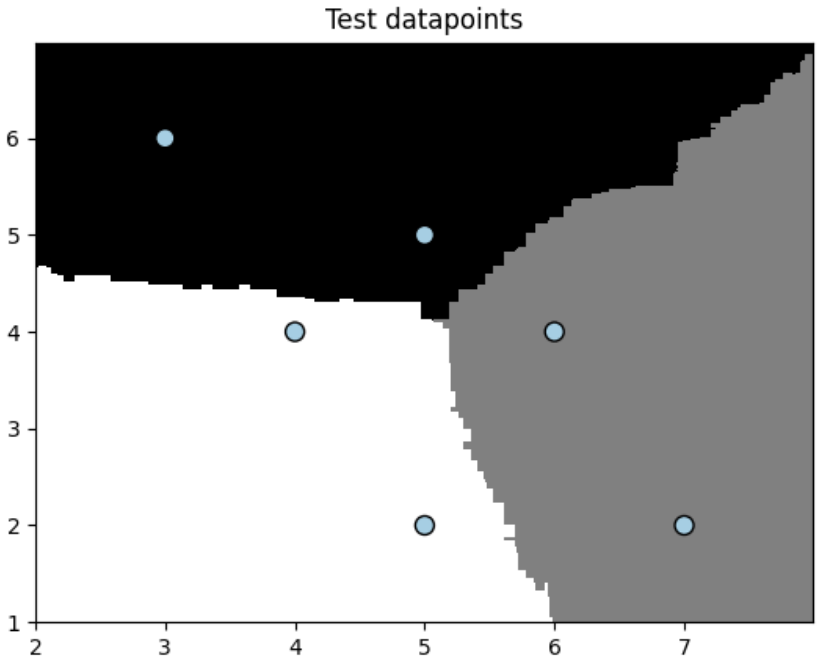

Рисунок 1.8 – Тестові точки даних

```
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
```

Рисунок 1.9 – Міра достовірності

Висновок: Моделі здебільшого мають схожі високі результати, проте rf показує себе незначно краще на навчальному наборі.

Завдання 2: Використовуючи для аналізу дані, які містяться у файлі data_imbalance.txt проведіть обробку з урахуванням дисбалансу класів.

Код програми:

```
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 = "Лабораторна робота 5/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_0[:, 0],
    class_0[:, 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()
```

Виконання:

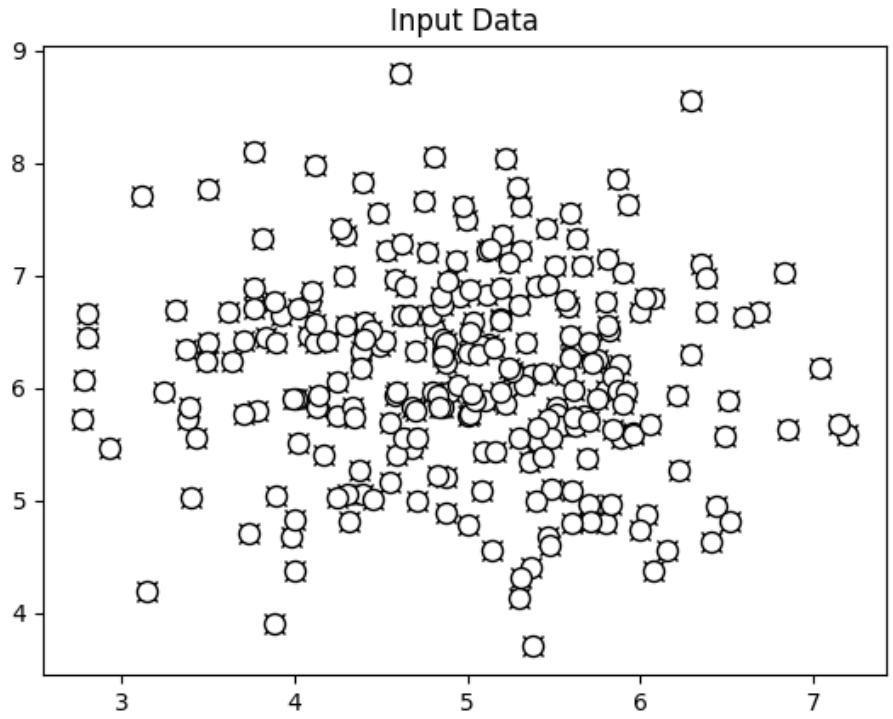

Рисунок 2.1 – Вхідні дані

Без урахування дисбалансу класів:

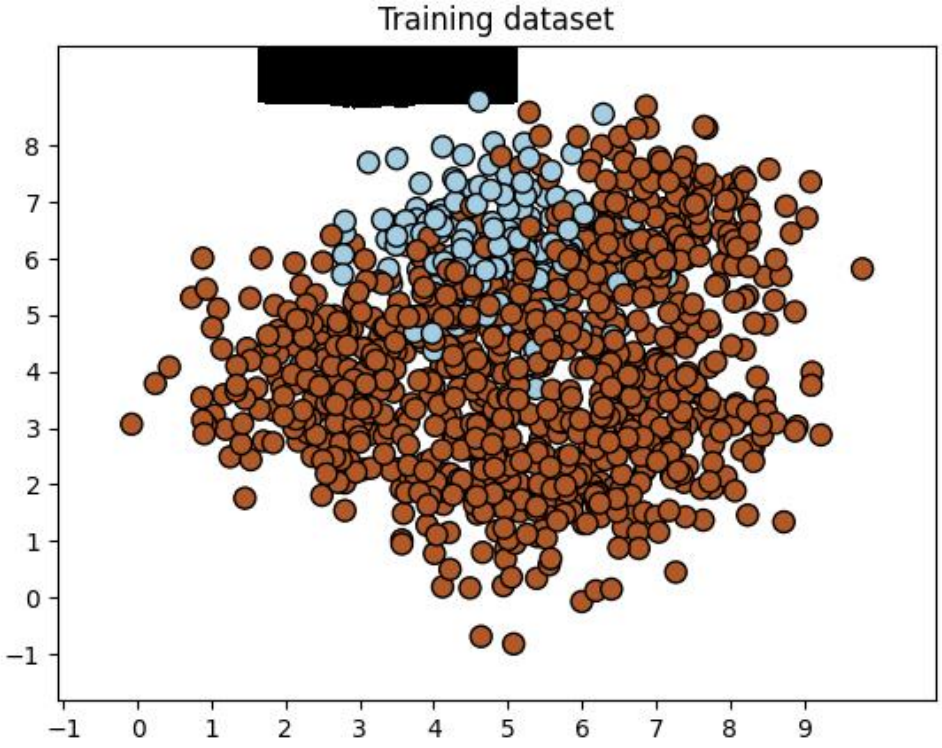

Рисунок 2.2 – Тренувальні дані

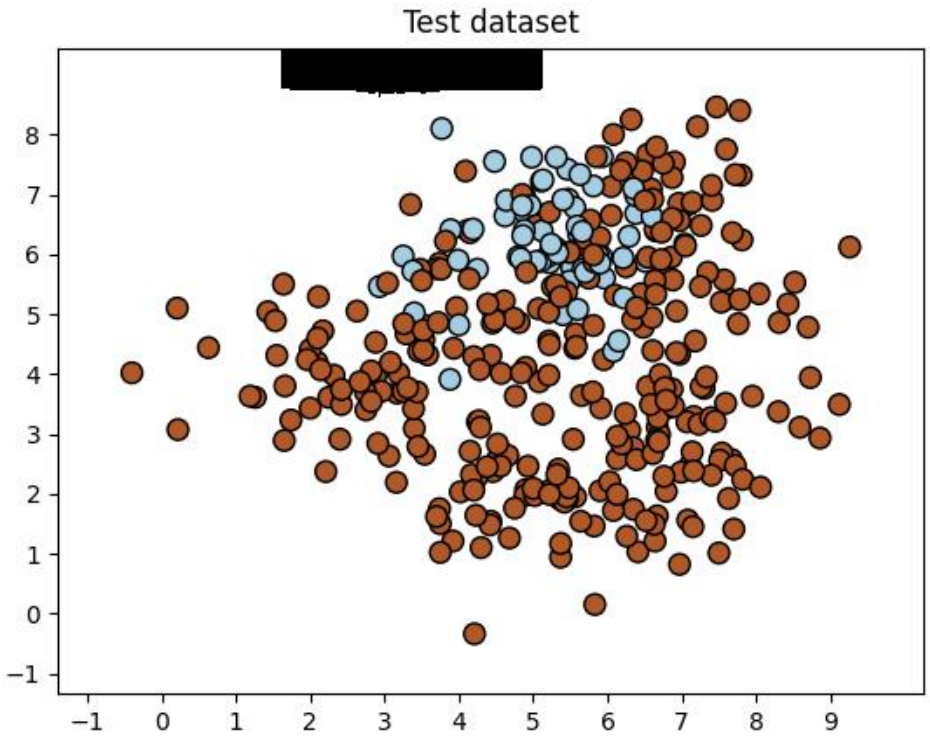

Рисунок 2.3 – Тестові дані

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

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

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

Рисунок 2.4 – Ефективність

З урахуванням дисбалансу класів:

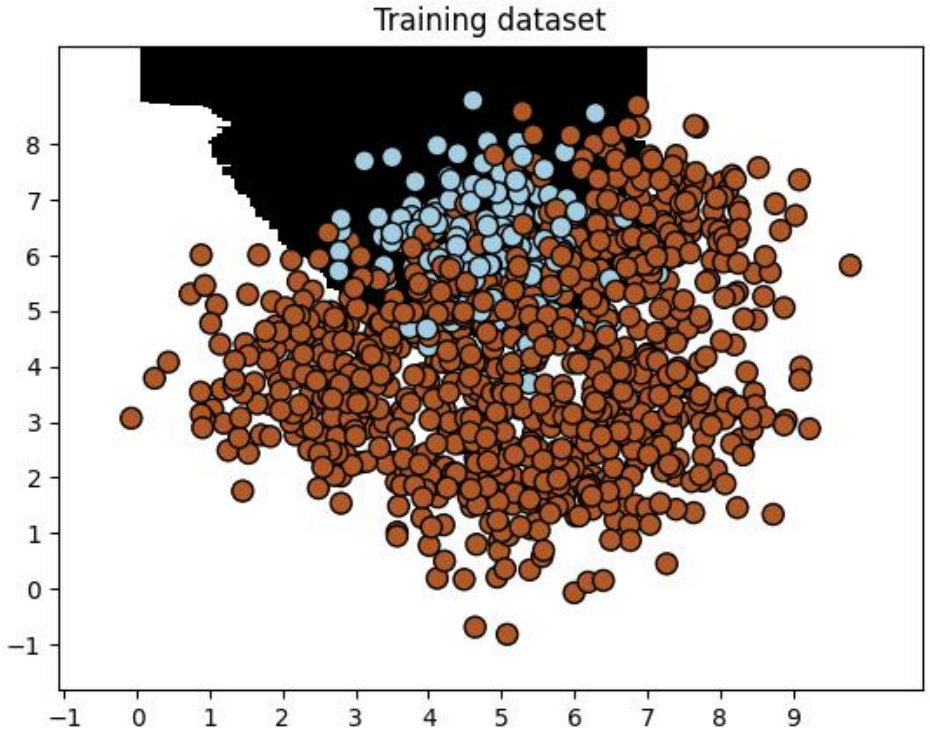

Рисунок 2.5 – Тренувальні дані

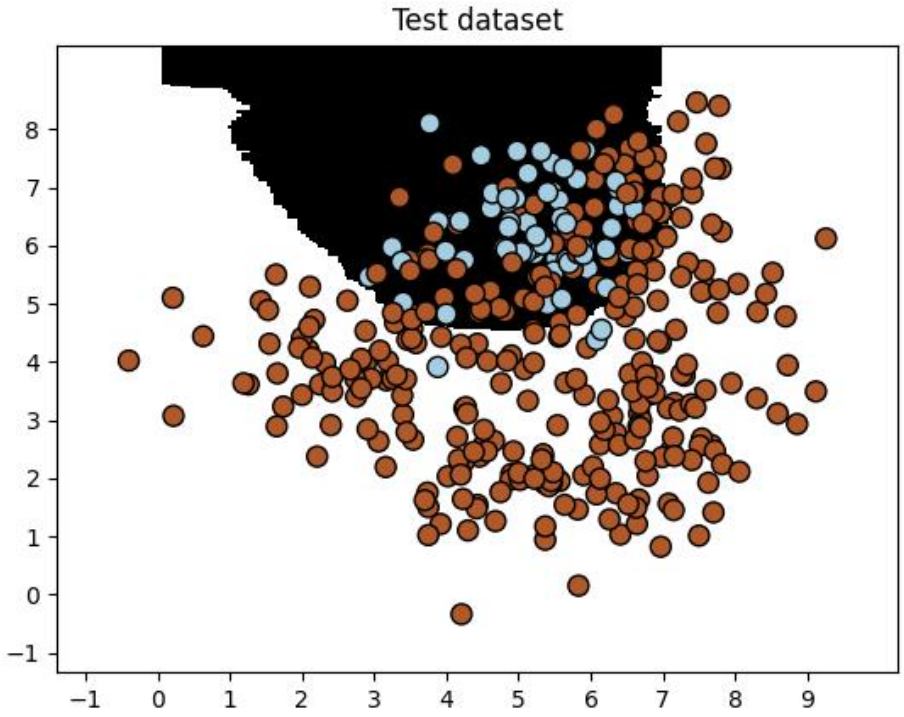

Рисунок 2.6 – Тестові дані

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

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

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

Рисунок 2.7 – Ефективність

Висновок: Урахування дизбалансу класів покращило показники для менш числених класів (Class-0), що демонструє збалансовану модель.

Завдання 3: Використовуючи дані, що містяться у файлі знайти оптимальних навчальних параметрів за допомогою сіткового пошуку.

Код програми:

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


input_file = "Лабораторна робота 5/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
params_grid = [
    {"n_estimators": [100], "max_depth": [2, 4, 7, 12, 16]},
    {"n_estimators": [25, 50, 100, 250], "max_depth": [4]},
]
metrics = ["precision_weighted", "recall_weighted"]

for m in metrics:
    print(f"\n##### Searching optimal parameters for {m}")
    classifier = GridSearchCV(
        ExtraTreesClassifier(random_state=0), params_grid, cv=5, scoring=m
    )
    classifier.fit(X_train, y_train)

    print("\nGrid scores for the parameter grid:")
    for i, params in enumerate(classifier.cv_results_["params"]):
        avg_score = classifier.cv_results_["mean_test_score"][i]
        print(params, "-->", round(avg_score, 3))

    print("\nBest parameters:", classifier.best_params_)
    y_pred = classifier.predict(X_test)
    print("\nPerformance report:\n")
    print(classification_report(y_test, y_pred))
```

Виконання:

```
Grid scores for the parameter grid:
{'max_depth': 2, 'n_estimators': 100} --> 0.85
{'max_depth': 4, 'n_estimators': 100} --> 0.841
{'max_depth': 7, 'n_estimators': 100} --> 0.844
{'max_depth': 12, 'n_estimators': 100} --> 0.832
{'max_depth': 16, 'n_estimators': 100} --> 0.816
{'max_depth': 4, 'n_estimators': 25} --> 0.846
{'max_depth': 4, 'n_estimators': 50} --> 0.84
{'max_depth': 4, 'n_estimators': 100} --> 0.841
{'max_depth': 4, 'n_estimators': 250} --> 0.845

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


##### Searching optimal parameters for recall_weighted

Grid scores for the parameter grid:
{'max_depth': 2, 'n_estimators': 100} --> 0.843
{'max_depth': 4, 'n_estimators': 100} --> 0.837
{'max_depth': 7, 'n_estimators': 100} --> 0.841
{'max_depth': 12, 'n_estimators': 100} --> 0.83
{'max_depth': 16, 'n_estimators': 100} --> 0.815
{'max_depth': 4, 'n_estimators': 25} --> 0.843
{'max_depth': 4, 'n_estimators': 50} --> 0.836
{'max_depth': 4, 'n_estimators': 100} --> 0.837
{'max_depth': 4, 'n_estimators': 250} --> 0.841

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
```

Рисунок 3.1 – Оптимальні параметри

Висновок: Найкращу модель забезпечують параметри max_depth=2, n_estimators=100.

Завдання 4: Обчислення відносної важливості ознак

Код програми:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn import datasets
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 = (
    datasets.fetch_california_housing()
)  # Boston dataset вилучений у нових версіях

# 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, ran-
dom_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 = np.array(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("Adaboost")
plt.show()
```

Виконання:

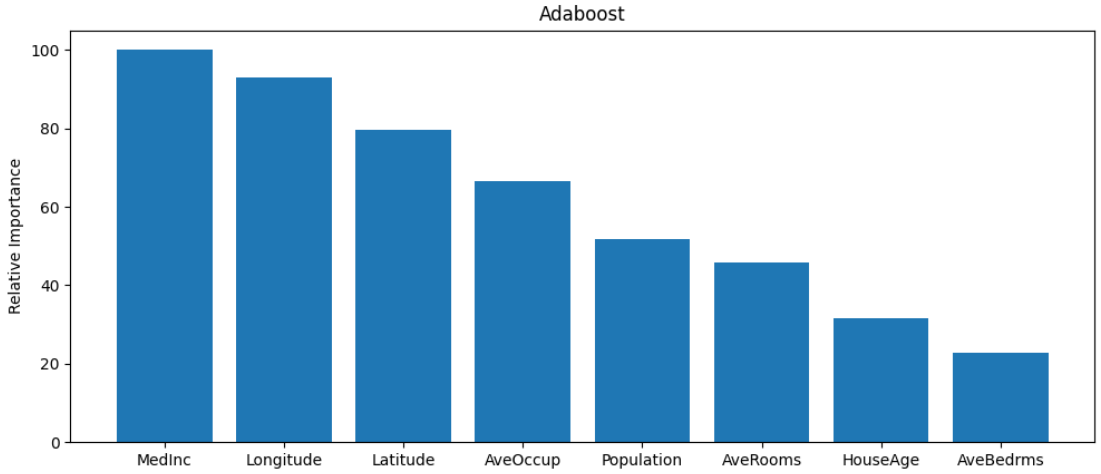

Рисунок 4.1 – Графік

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

Рисунок 4.2 – Виконання програми

Висновок: Ознаки MedInc, Longitude, і Latitude є критичними. На відміну від HouseAge та AveBedrms.

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

Код програми:

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

input_file = "Лабораторна робота 5/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]])[0]
        )
        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]))
```

Виконання:

```
Predicted traffic: 26
```

Рисунок 5.1 – Виконання програми

Посилання на GitHub: https://github.com/ipz215kvv/artificial-inteligence-systems