Лабораторна робота №5

Тема: ДОСЛІДЖЕННЯ МЕТОДІВ АНСАМБЛЕВОГО НАВЧАННЯ

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

Хід роботи:

Завдання 5.1. Створення класифікаторів на основі випадкових та гранично випадкових лісів Використовувати файл вхідних даних: data_random_forests.txt, побудувати класифікатори на основі випадкових та гранично випадкових лісів.

Лістинг програми:

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

def plot_classifier(classifier, X, y):
    x_min, x_max = min(X[:, 0]) - 1.0, max(X[:, 0]) + 1.0
    y_min, y_max = min(X[:, 1]) - 1.0, max(X[:, 1]) + 1.0
    step_size = 0.01

    x_values, y_values = np.meshgrid(
        np.arange(x_min, x_max, step_size),
        np.arange(y_min, y_max, step_size)
    )

    mesh_output = classifier.predict(np.c_[x_values.ravel(), y_values.ravel()])
    mesh_output = mesh_output.reshape(x_values.shape)

    plt.pcolormesh(x_values, y_values, mesh_output, cmap=plt.cm.gray,
shading="auto")

    plt.scatter(X[:, 0], X[:, 1], c=y, s=80,
                edgecolors='black', linewidth=1, cmap=plt.cm.Paired)

    plt.xlim(x_values.min(), x_values.max())
    plt.ylim(y_values.min(), y_values.max())
def build_arg_parser():
    parser = argparse.ArgumentParser(description='Створення класифікаторів на
основі лісів')
```

```
    parser.add_argument(
        '--classifier-type',
        dest='classifier_type',
        required=True,
        choices=['rf', 'erf'],
        help='Тип класифікатора: "rf" = випадковий ліс, "erf" = гранично
випадковий ліс'
    )
    return parser

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

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

    class_0 = X[y == 0]
    class_1 = X[y == 1]
    class_2 = X[y == 2]

    plt.figure()
    plt.scatter(class_0[:, 0], class_0[:, 1], s=75, facecolors='white',
                edgecolors='black', linewidth=1, marker='s', label='Клас 0')
    plt.scatter(class_1[:, 0], class_1[:, 1], s=75, facecolors='white',
                edgecolors='black', linewidth=1, marker='o', label='Клас 1')
    plt.scatter(class_2[:, 0], class_2[:, 1], s=75, facecolors='white',
                edgecolors='black', linewidth=1, marker='^', label='Клас 2')
    plt.title('Вхідні дані')
    plt.legend()
    plt.show()

    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.25, random_state=5
    )

    params = {'n_estimators': 100, 'max_depth': 4, 'random_state': 0}

    if classifier_type == 'rf':
        classifier = RandomForestClassifier(**params)
        title = 'Випадковий ліс'
    else:
        classifier = ExtraTreesClassifier(**params)
        title = 'Гранично випадковий ліс'

    classifier.fit(X_train, y_train)

    plt.figure()
    plot_classifier(classifier, X, y)
    plt.title(title)
    plt.show()

    y_test_pred = classifier.predict(X_test)

    print(f"\nЗвіт про якість класифікації для {title}:\n")
    print(classification_report(y_test, y_test_pred))

    test_datapoints = np.array([[5, 5], [3, 6], [6, 4], [7, 2.5]])

    print("\nПараметри довірливості для нових точок даних:\n")
```

```
    for datapoint in test_datapoints:
        plt.figure()
        plot_classifier(classifier, X, y)
        plt.scatter(datapoint[0], datapoint[1], s=200,
                    facecolors='none', edgecolors='red',
                    linewidth=3, marker='x')
        plt.title(f'{title}: Тестова точка [{datapoint[0]}, {datapoint[1]}]')
        plt.show()

        probabilities = classifier.predict_proba([datapoint])[0]
        predicted_class = classifier.predict([datapoint])[0]

        print(f"Точка даних: {datapoint}")
        print(f" Ймовірність класу 0: {round(probabilities[0]*100, 2)}%")
        print(f" Ймовірність класу 1: {round(probabilities[1]*100, 2)}%")
        print(f" Ймовірність класу 2: {round(probabilities[2]*100, 2)}%")
        print(f" Передбачений клас: {int(predicted_class)}\n")
```

Результат роботи програми:

```
C:\Users\Admin\AppData\Local\Programs\Python\Python39\python.exe

Звіт про якість класифікації для Випадковий ліс:

              precision    recall  f1-score   support

         0.0       0.92      0.85      0.88        79
         1.0       0.86      0.84      0.85        70
         2.0       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
```

Рис. 5.1 Вивід у консоль

Параметри довірливості для нових точок даних:

```
Точка даних: [5. 5.]
 Ймовірність класу 0: 81.43%
 Ймовірність класу 1: 8.64%
 Ймовірність класу 2: 9.93%
 Передбачений клас: 0
```

```
Точка даних: [3. 6.]
 Ймовірність класу 0: 93.57%
 Ймовірність класу 1: 2.47%
 Ймовірність класу 2: 3.96%
 Передбачений клас: 0
```

```
Точка даних: [6. 4.]
 Ймовірність класу 0: 12.23%
 Ймовірність класу 1: 74.51%
 Ймовірність класу 2: 13.26%
 Передбачений клас: 1
```

```
Точка даних: [7.  2.5]
 Ймовірність класу 0: 6.23%
 Ймовірність класу 1: 73.55%
 Ймовірність класу 2: 20.21%
 Передбачений клас: 1
```

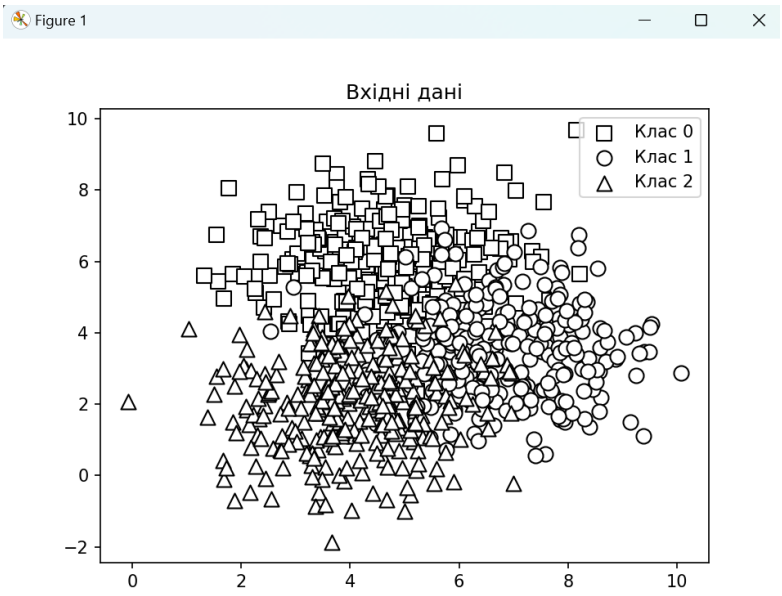
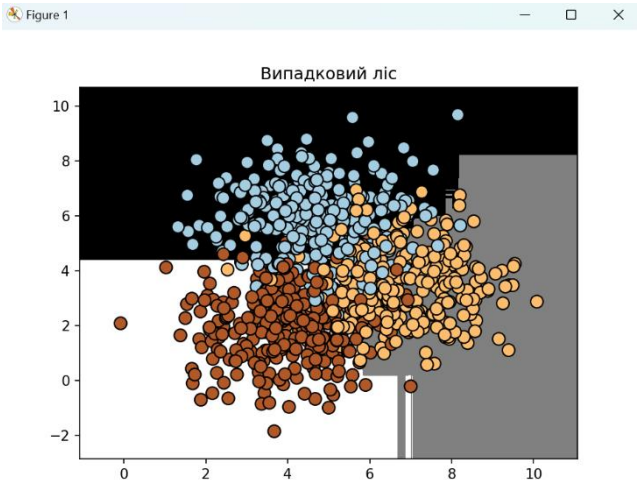
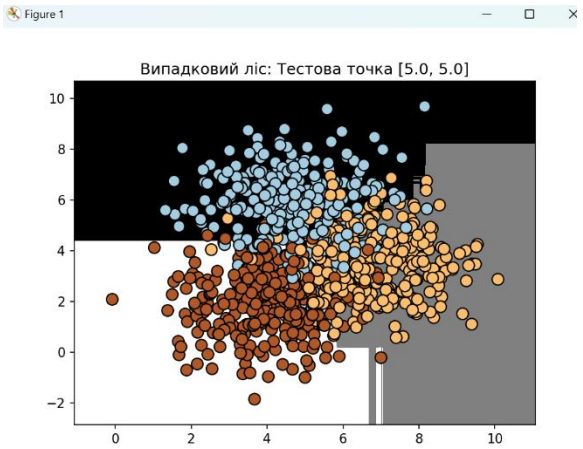
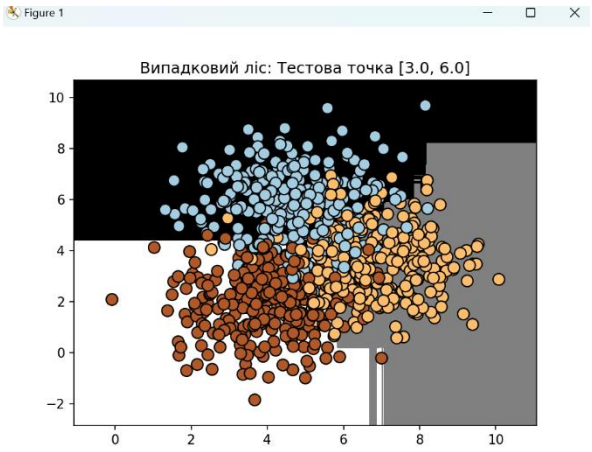
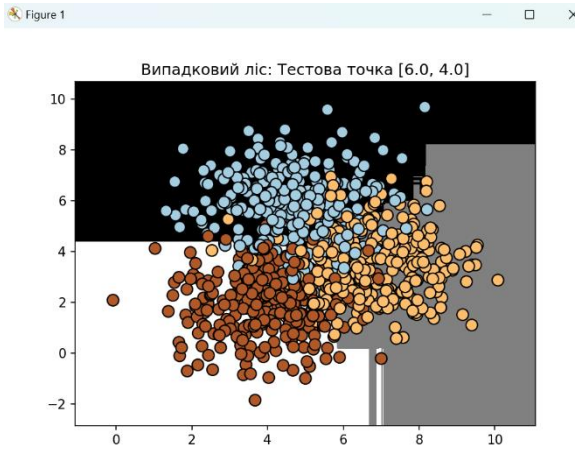


Рис. 5.2. Графік вхідних даних rf









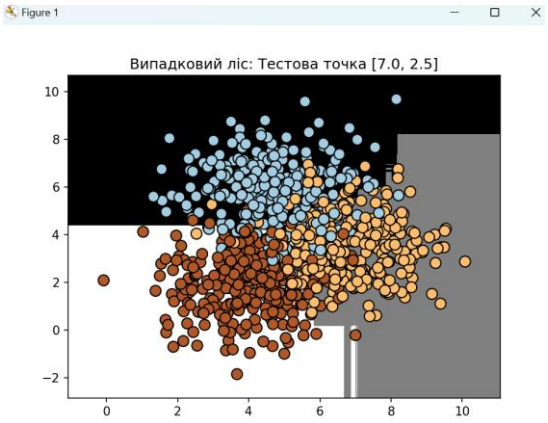


Рис. 5.7. Графіки випадкового лісу для точки rf

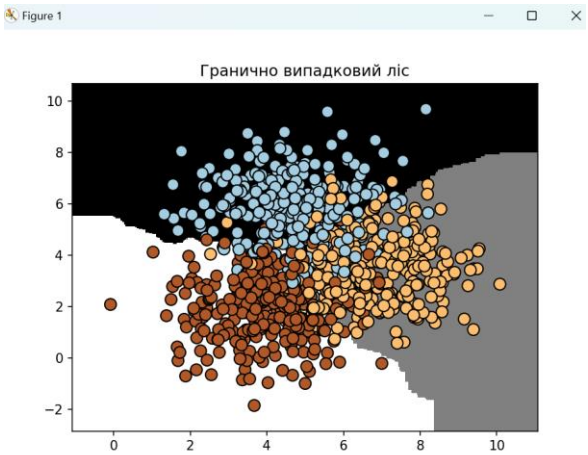
Результат роботи програми:

```
C:\Users\Admin\AppData\Local\Programs\Python\Python39\python.

Звіт про якість класифікації для Гранично випадковий ліс:

              precision    recall  f1-score   support

         0.0       0.92      0.85      0.88        79
         1.0       0.84      0.84      0.84        70
         2.0       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
```

Рис. 5.8. Вивід у консоль

```
Точка даних: [5. 5.]
 Ймовірність класу 0: 48.9%
 Ймовірність класу 1: 28.02%
 Ймовірність класу 2: 23.08%
 Передбачений клас: 0
```

```
Точка даних: [3. 6.]
 Ймовірність класу 0: 66.71%
 Ймовірність класу 1: 12.42%
 Ймовірність класу 2: 20.87%
 Передбачений клас: 0
```



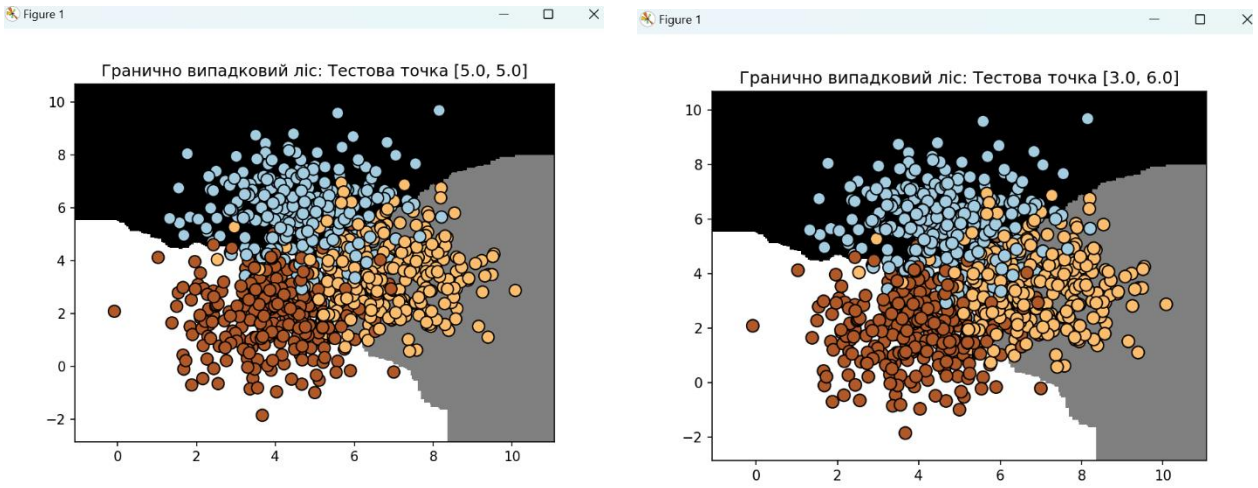


Рис. 5.9. Графіки гранично випадкового лісу для точки erf

Висновки по завданню: обидва методи, випадковий ліс (rf) та гранично випадковий ліс (erf), показали високу точність класифікації на тестових даних. Загальна точність для обох моделей склала 0.87, тобто 87% правильних передбачень на тестовому наборі.

Середнє значення та зважене середнє значення для точності, повноти та f1-міри також приблизно однакові (0.87), що свідчить про збалансовану роботу моделей для всіх трьох класів.

Завдання 5.2. Обробка дисбалансу класів Використовуючи для аналізу дані, які містяться у файлі data_imbalance.txt проведіть обробку з урахуванням дисбалансу класів.

Лістинг 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

def plot_classifier(classifier, X, y, title):
    x_min, x_max = min(X[:, 0]) - 1.0, max(X[:, 0]) + 1.0
    y_min, y_max = min(X[:, 1]) - 1.0, max(X[:, 1]) + 1.0
    step_size = 0.01
    x_values, y_values = np.meshgrid(np.arange(x_min, x_max, step_size),
np.arange(y_min, y_max, step_size))

    mesh_output = classifier.predict(np.c_[x_values.ravel(), y_values.ravel()])
    mesh_output = mesh_output.reshape(x_values.shape)
```

```
    plt.figure()
    plt.pcolormesh(x_values, y_values, mesh_output, cmap=plt.cm.gray)
    plt.scatter(X[:, 0], X[:, 1], c=y, s=80, edgecolors='black', linewidth=1,
cmap=plt.cm.Paired)
    plt.xlim(x_values.min(), x_values.max())
    plt.ylim(y_values.min(), y_values.max())
    plt.title(title)
    plt.show()


if __name__ == '__main__':
    input_file = 'data_imbalance.txt'
    data = np.loadtxt(input_file, delimiter=',')
    X, y = data[:, :-1], data[:, -1]

    class_0 = np.array(X[y == 0])
    class_1 = np.array(X[y == 1])

    plt.figure()
    plt.scatter(class_0[:, 0], class_0[:, 1], s=75, facecolors='black',
                edgecolors='black', linewidth=1, marker='x', label='Клас 0
(Більшість)')
    plt.scatter(class_1[:, 0], class_1[:, 1], s=75, facecolors='white',
                edgecolors='black', linewidth=1, marker='o', label='Клас 1
(Меншість)')
    plt.title('Вхідні дані з дисбалансом класів')
    plt.legend()
    plt.show()

    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.25, random_state=5)

    params = {'n_estimators': 100, 'max_depth': 4, 'random_state': 0}
    if len(sys.argv) > 1 and sys.argv[1] == 'balance':
        params['class_weight'] = 'balanced'
        title = 'Класифікатор з урахуванням дисбалансу'
        print("Виконується класифікація з параметром class_weight='balanced'.")
    else:
        title = 'Класифікатор без урахування дисбалансу'
        print("Виконується класифікація без урахування дисбалансу класів.")

    classifier = ExtraTreesClassifier(**params)
    classifier.fit(X_train, y_train)
    plot_classifier(classifier, X_test, y_test, f'{title} (тестові дані)')

    y_test_pred = classifier.predict(X_test)
    print(f"\nЗвіт про якість класифікації для '{title}':\n")
    print(classification_report(y_test, y_test_pred))
```

Результат роботи програми:

```
                precision    recall  f1-score   support

         0.0       0.00      0.00      0.00        69
         1.0       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
```

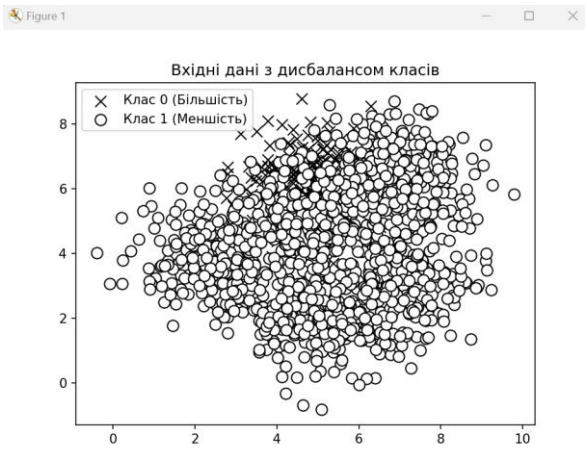


Рис. 5.10. Графік вхідних даних з дисбалансом класів

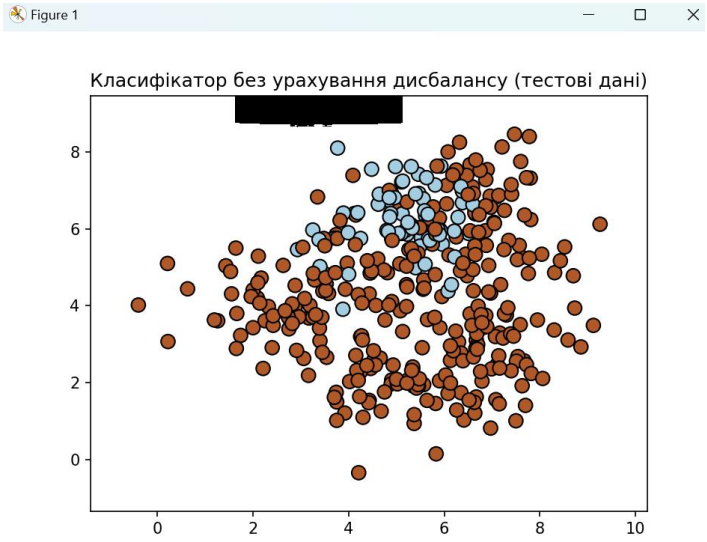


Рис. 5.11. Графік класифікатора без дисбалансу

Висновок по завданню: результати показали, що без урахування дисбалансу модель добре класифікує більший клас, але майже ігнорує менший, що призводить до низьких показників точності**,** повноти та f1-міри для класу меншості. Загальна точність склала 0.82, але середнє та зважене середнє значення метрик свідчать про нерівномірну ефективність для різних класів. Це підкреслює необхідність застосування методів балансування, наприклад, class_weight = 'balanced' для отримання стабільної та збалансованої класифікації для всіх класів.

Завдання 5.3. Знаходження оптимальних навчальних параметрів за допомогою сіткового пошуку. Використовуючи дані, що містяться у файлі знайти оптимальних навчальних параметрів за допомогою сіткового пошуку.

Лістинг програми:

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

if __name__ == '__main__':
    input_file = 'data_random_forests.txt'
    data = np.loadtxt(input_file, delimiter=',')
    X, y = data[:, :-1], data[:, -1]

    class_0 = np.array(X[y == 0])
    class_1 = np.array(X[y == 1])
    class_2 = np.array(X[y == 2])

    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.25, random_state=5
    )

    parameter_grid = [
        {'n_estimators': [25, 50, 100, 250],
         'max_depth': [2, 4, 7, 12, 16]},
    ]

    metrics = ['precision_weighted', 'recall_weighted']

    for metric in metrics:
        print("\n#################################################################")
        print(f"Пошук оптимальних параметрів для метрики: '{metric}'")
        print("#################################################################")

        classifier = GridSearchCV(
            ExtraTreesClassifier(random_state=0),
            parameter_grid,
            cv=5,
            scoring=metric
        )

        classifier.fit(X_train, y_train)

        print("\nОцінки для кожної комбінації параметрів:")
        for mean_score, params in zip(classifier.cv_results_['mean_test_score'],
                                      classifier.cv_results_['params']):
            print(f" Оцінка: {mean_score:.4f} для параметрів: {params}")

        print(f"\nНайкращі параметри для метрики '{metric}':")
        print(classifier.best_params_)
        print(f"\nНайкраща оцінка на перехресній перевірці ({metric}):
{classifier.best_score_:.4f}")

        y_pred = classifier.predict(X_test)
        print("\nЗвіт класифікації для тестового набору даних:")
        print(classification_report(y_test, y_pred))
```

Результат роботи програми:

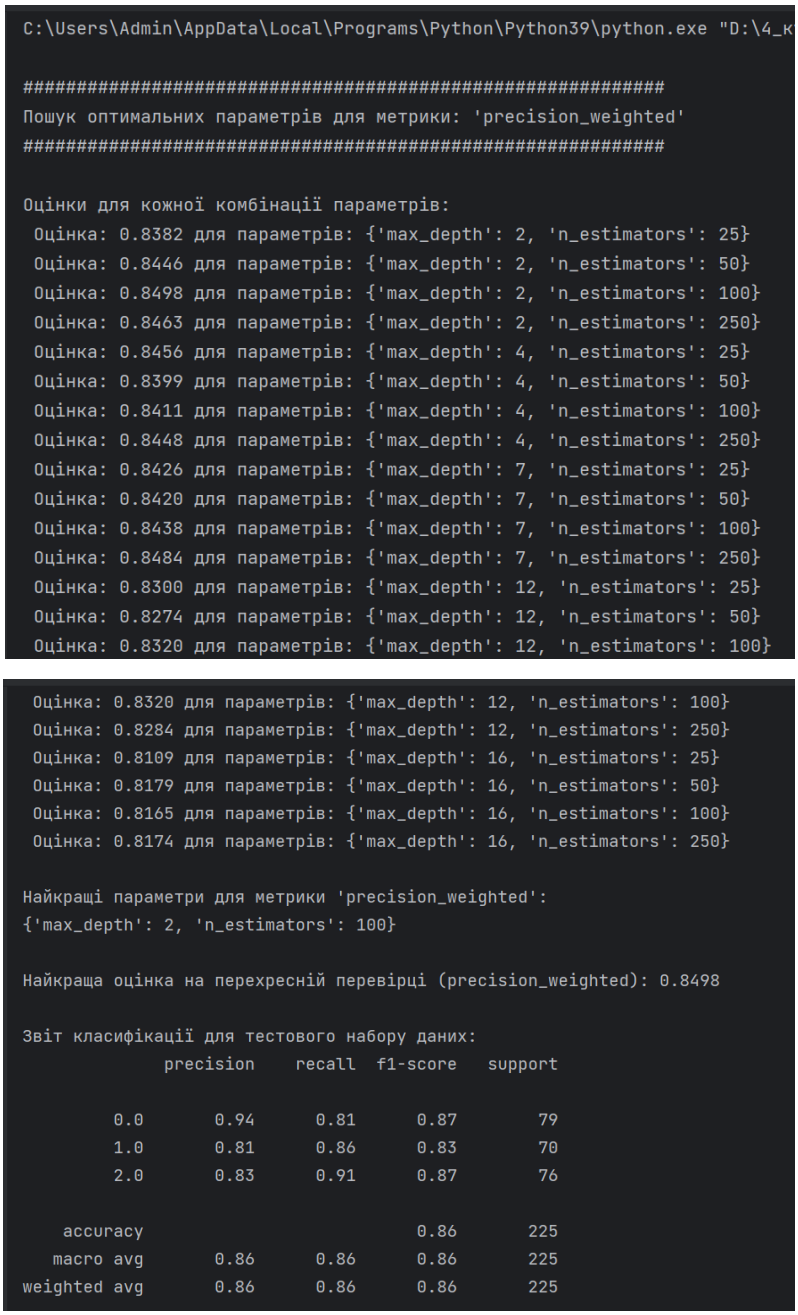


Рис. 5.12. Пошук оптимальних параметрів для метрики: 'precision_weighted'

```
############################################################
Пошук оптимальних параметрів для метрики: 'recall_weighted'
############################################################

Оцінки для кожної комбінації параметрів:
 Оцінка: 0.8326 для параметрів: {'max_depth': 2, 'n_estimators': 25}
 Оцінка: 0.8370 для параметрів: {'max_depth': 2, 'n_estimators': 50}
 Оцінка: 0.8430 для параметрів: {'max_depth': 2, 'n_estimators': 100}
 Оцінка: 0.8415 для параметрів: {'max_depth': 2, 'n_estimators': 250}
 Оцінка: 0.8430 для параметрів: {'max_depth': 4, 'n_estimators': 25}
 Оцінка: 0.8356 для параметрів: {'max_depth': 4, 'n_estimators': 50}
 Оцінка: 0.8370 для параметрів: {'max_depth': 4, 'n_estimators': 100}
 Оцінка: 0.8415 для параметрів: {'max_depth': 4, 'n_estimators': 250}
 Оцінка: 0.8400 для параметрів: {'max_depth': 7, 'n_estimators': 25}
 Оцінка: 0.8385 для параметрів: {'max_depth': 7, 'n_estimators': 50}
 Оцінка: 0.8415 для параметрів: {'max_depth': 7, 'n_estimators': 100}
 Оцінка: 0.8459 для параметрів: {'max_depth': 7, 'n_estimators': 250}
 Оцінка: 0.8281 для параметрів: {'max_depth': 12, 'n_estimators': 25}
 Оцінка: 0.8252 для параметрів: {'max_depth': 12, 'n_estimators': 50}
 Оцінка: 0.8296 для параметрів: {'max_depth': 12, 'n_estimators': 100}
 Оцінка: 0.8267 для параметрів: {'max_depth': 12, 'n_estimators': 250}
 Оцінка: 0.8089 для параметрів: {'max_depth': 16, 'n_estimators': 25}
 Оцінка: 0.8089 для параметрів: {'max_depth': 16, 'n_estimators': 25}
 Оцінка: 0.8163 для параметрів: {'max_depth': 16, 'n_estimators': 50}
 Оцінка: 0.8148 для параметрів: {'max_depth': 16, 'n_estimators': 100}
 Оцінка: 0.8148 для параметрів: {'max_depth': 16, 'n_estimators': 250}

Найкращі параметри для метрики 'recall_weighted':
{'max_depth': 7, 'n_estimators': 250}

Найкраща оцінка на перехресній перевірці (recall_weighted): 0.8459

Звіт класифікації для тестового набору даних:
              precision    recall  f1-score   support

         0.0       0.91      0.85      0.88        79
         1.0       0.84      0.83      0.83        70
         2.0       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
```

Рис. 5.13. Пошук оптимальних параметрів для метрики: 'recall_weighted'

Висновок по завданню: було здійснено пошук оптимальних параметрів для ансамблевого класифікатора «Гранично випадковий ліс» за двома метриками: точність (precision_weighted) та повнота (recall_weighted).

Для метрики точність найкращими параметрами виявились max_depth = 2 та n_estimators = 100, при цьому точність класифікації на тестовій вибірці склала 0.86, а метрики для всіх класів були збалансованими.

Для метрики повнота оптимальні параметри — max_depth = 7 та n_estimators = 250, при цьому тестова точність зросла до 0.87, а macro avg і weighted avg для precision, recall та f1-score також склали 0.87, що свідчить про збалансовану класифікацію для всіх класів.

Завдання 5.4. Обчислення відносної важливості ознак.

Лістинг програми:

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

if __name__ == '__main__':
    data_url = "http://lib.stat.cmu.edu/datasets/boston"
    raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=None)

    data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
    target = raw_df.values[1::2, 2]

    feature_names = np.array([
        'CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE',
        'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT'
    ])

    X, y = shuffle(data, target, random_state=7)

    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=7
    )

    regressor = AdaBoostRegressor(
        DecisionTreeRegressor(max_depth=4),
        n_estimators=400,
        random_state=7
    )

    regressor.fit(X_train, y_train)

    y_pred = regressor.predict(X_test)

    mse = mean_squared_error(y_test, y_pred)
    evs = explained_variance_score(y_test, y_pred)

    print("\nADABOOST REGRESSOR")
    print(f"Mean squared error = {round(mse, 2)}")
    print(f"Explained variance score = {round(evs, 2)}")

    feature_importances = regressor.feature_importances_
    feature_importances = 100.0 * (feature_importances /
feature_importances.max())
    index_sorted = np.flipud(np.argsort(feature_importances))
    pos = np.arange(index_sorted.shape[0]) + 0.5
```

```
plt.figure(figsize=(12, 7))
plt.bar(pos, feature_importances[index_sorted], align='center')
plt.xticks(pos, feature_names[index_sorted], rotation=45, ha='right')
plt.ylabel('Відносна важливість (%)')
plt.title('Важливість ознак для прогнозування цін на нерухомість')
plt.tight_layout()
plt.show()
```

Результат роботи програми:

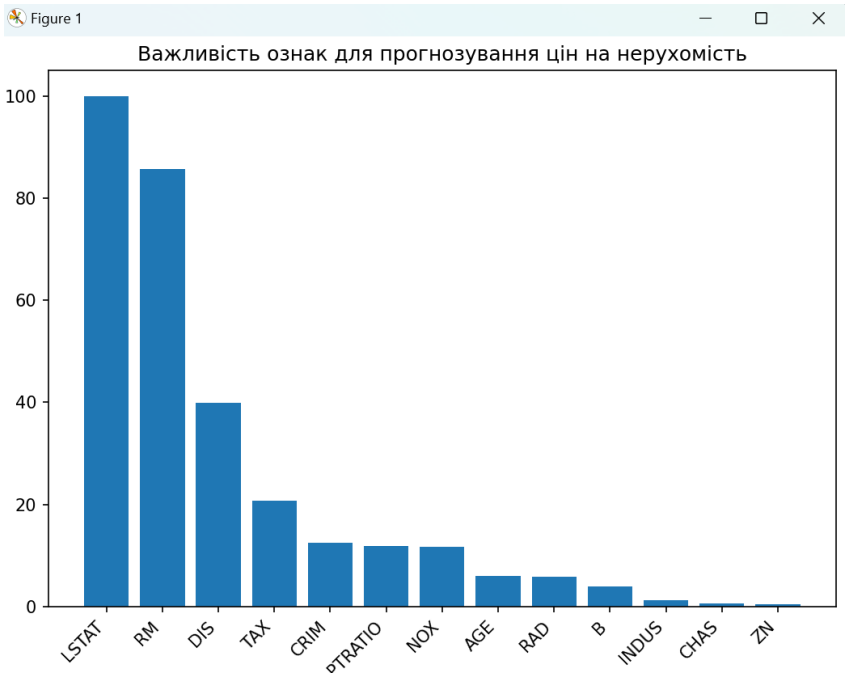


Рис. 5.14 Діаграма важливості ознак

```
C:\Users\Admin\AppData\Local\Progr

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

Рис. 5.15. Вивід у консоль

Висновок по завданню: аналізуючи дану діаграму можна скказати, що найважливіші ознаки:

- LSTAT (відсоток населення з низьким статусом) - найбільший внесок у модель, 100% (найвища відносна важливість).
- RM (середня кількість кімнат у будинку) - друга за важливістю ознака, близько 86% відносної важливості.

Середньо важливі ознаки:

- DIS (відстань до робочих місць у Бостоні), TAX (податок на нерухомість), CRIM (рівень злочинності), PTRATIO (співвідношення учнів до вчителів у школах) та NOX (концентрація оксидів азоту) мають помірний вплив. Їх важливість у діапазоні 10 - 40%.

Мало важливі ознаки:

AGE, RAD, B, INDUS, CHAS, ZN - дуже малий внесок у прогнозування ціни (практично не впливають на модель).

Завдання 5.5. Прогнозування інтенсивності дорожнього руху за допомогою класифікатора на основі гранично випадкових лісів.

Лістинг програми:

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

def load_data(file_path):
    data = []
    with open(file_path, 'r') as f:
        for line in f:
            items = line.strip().split(',')
            data.append(items)
    return np.array(data)


def encode_features(data):
    label_encoders = []
    X_encoded = np.empty(data.shape, dtype=int)

    for i in range(data.shape[1]):
        if np.all([item.isdigit() for item in data[:, i]]):
            X_encoded[:, i] = data[:, i].astype(int)
        else:
            encoder = preprocessing.LabelEncoder()
            X_encoded[:, i] = encoder.fit_transform(data[:, i])
            label_encoders.append(encoder)

    return X_encoded, label_encoders


if __name__ == '__main__':
    input_file = 'traffic_data.txt'
    data = load_data(input_file)

    X_encoded, label_encoders = encode_features(data)

    X = X_encoded[:, :-1]
    y = X_encoded[:, -1]

    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.25, random_state=5
    )
```

```
    regressor = ExtraTreesRegressor(n_estimators=100, max_depth=4,
random_state=0)
    regressor.fit(X_train, y_train)

    y_pred = regressor.predict(X_test)
    print(f"Mean absolute error: {mean_absolute_error(y_test, y_pred):.2f}")

    test_datapoint = ['Saturday', '10:20', 'Atlanta', 'no']
    test_datapoint_encoded = []

    print("\nOriginal test data point:", test_datapoint)

    count = 0
    for i, item in enumerate(test_datapoint):
        if item.isdigit():
            test_datapoint_encoded.append(int(item))
        else:
            encoded_value = label_encoders[count].transform([item])[0]
            test_datapoint_encoded.append(int(encoded_value))
            count += 1

    print("Encoded test data point:", test_datapoint_encoded)

    predicted_traffic = regressor.predict([test_datapoint_encoded])
    print(f"\nPredicted traffic for the test data point:
{int(predicted_traffic[0])}")
```

Результат роботи програми:

```
C:\Users\Admin\AppData\Local\Programs\Python\Python39\python.exe "D
Mean absolute error: 12.75

Original test data point: ['Saturday', '10:20', 'Atlanta', 'no']
Encoded test data point: [2, 124, 1, 0]

Predicted traffic for the test data point: 24

Process finished with exit code 0
```

Рис. 5.16. Вивід у консоль

Висновок по завданню: завантажено та підготовлено дані про трафік. Категоріальні ознаки (день тижня, місто, наявність події) було закодовано за допомогою LabelEncoder, числові ознаки залишено без змін.

Дані було розділено на тренувальний та тестовий набори у співвідношенні 75% на 25%.

Модель ExtraTreesRegressor (100 дерев, максимальна глибина 4) навчалася на тренувальному наборі.

Для оцінки точності передбачення використано середню абсолютну помилку (MAE), яка склала 12.75, що означає, що в середньому передбачене значення

трафіку відхиляється від реального на ~13 одиниць.

Тестовий приклад ['Saturday', '10:20', 'Atlanta', 'no'] було закодовано як [2, 124, 1, 0]. Модель передбачила 24 одиниці трафіку для цієї точки.

Модель демонструє прийнятну точність для приблизного прогнозування трафіку, однак помилка ~13 може бути значною, залежно від масштабу даних. Для підвищення точності можна збільшити глибину дерев, кількість дерев, або використати додаткові ознаки.

Висновок: під час виконання лабораторної роботи використовуючи спеціалізовані бібліотеки та мову програмування Python було досліджено методи ансамблів у машинному навчанні.

Посилання на git: https://github.com/KokhanTetiana/AI-Systems