Лабораторна робота № 5

ДОСЛІДЖЕННЯ МЕТОДІВ АНСАМБЛЕВОГО НАВЧАННЯ

Мета роботи: 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.

Лістинг LR_5_task_1.py:

```
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)
    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" для випадкового лісу або "erf" для
гранично випадкового лісу')
    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 = np.array(X[y==0])
    class_1 = np.array(X[y==1])
    class_2 = np.array(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("Параметри довірливості для нових точок даних:")
    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"\nТочка даних: {datapoint}")
        print(f"  Ймовірність належності до класу 0: {round(probabilities[0]*100, 2)}%")
        print(f"  Ймовірність належності до класу 1: {round(probabilities[1]*100, 2)}%")
        print(f"  Ймовірність належності до класу 2: {round(probabilities[2]*100, 2)}%")
        print(f"  Передбачений клас: {int(predicted_class)}")
```

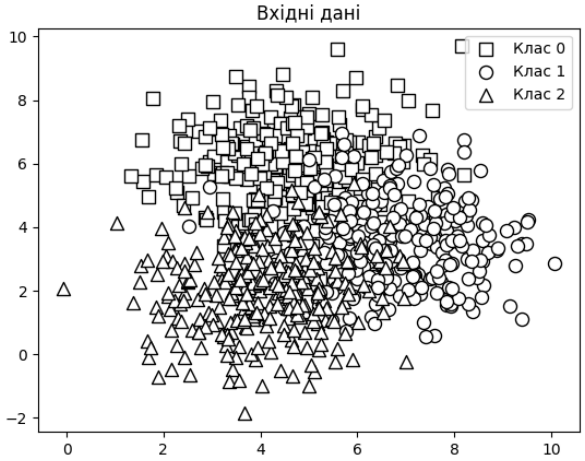

Рис.5.1. Графік вхідних даних rf

Графік показує три класи даних. Можна візуально оцінити, що класи частково перекриваються, що створює певну складність для простого лінійного класифікатора.

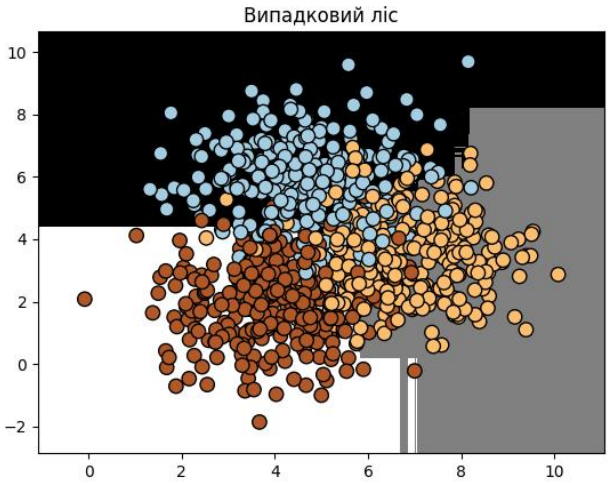

Рис.5.2. Графік випадкового лісу rf

На графіку буде показано, як алгоритм випадкового лісу розділив простір ознак на три області, що відповідають трьом класам. Межі мають бути відносно чіткими, але можуть мати "східчасту" структуру.

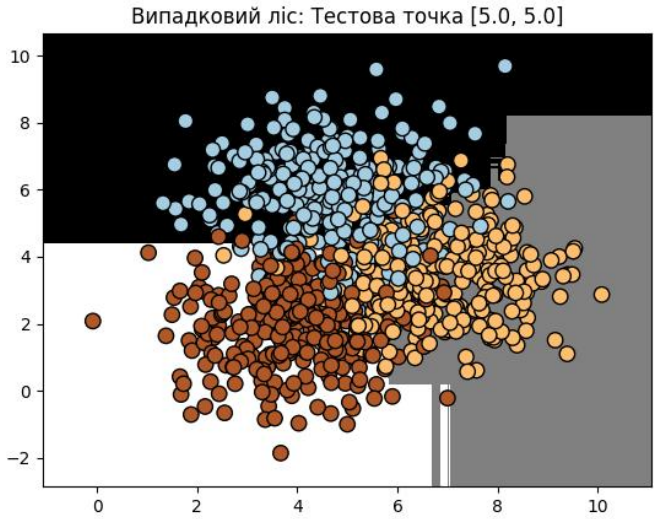

Рис.5.3. Графік випадкового лісу для точки rf

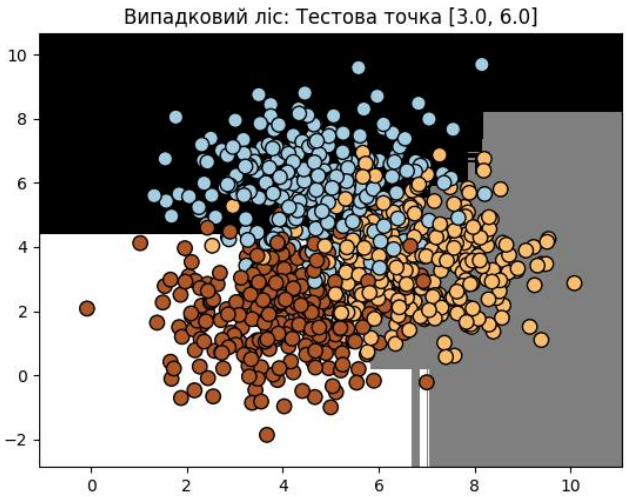

Рис.5.4. Графік випадкового лісу для точки rf

Вивід в консоль RF:

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

Параметри довірливості для нових точок даних:

…

Точка даних: [5. 5.]

Ймовірність належності до класу 0: 81.43%

Ймовірність належності до класу 1: 8.64%

Ймовірність належності до класу 2: 9.93%

Передбачений клас: 0

Точка даних: [3. 6.]

Ймовірність належності до класу 0: 93.57%

Ймовірність належності до класу 1: 2.47%

Ймовірність належності до класу 2: 3.96%

Передбачений клас: 0

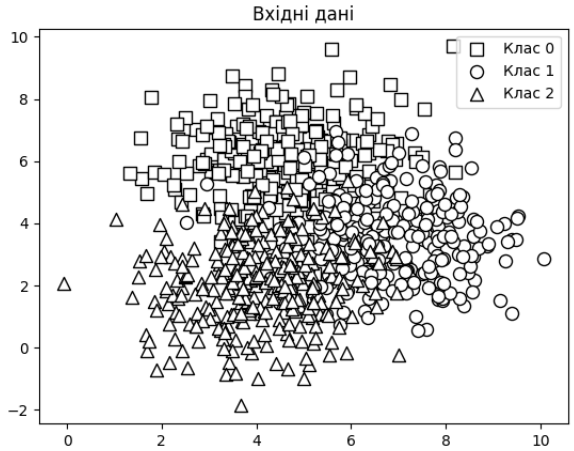

Рис.5.5. Графік вхідних даних erf

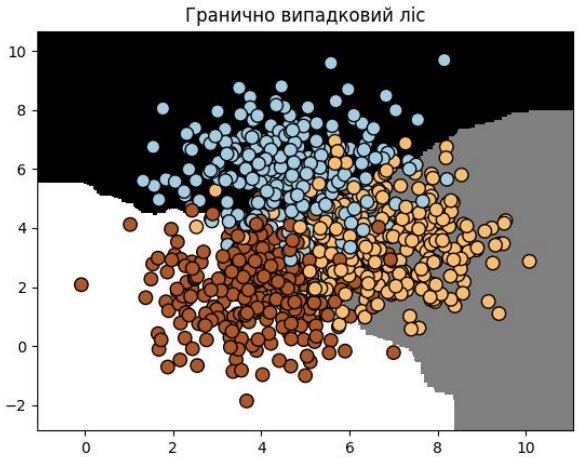

Рис.5.6. Графік випадкового лісу erf

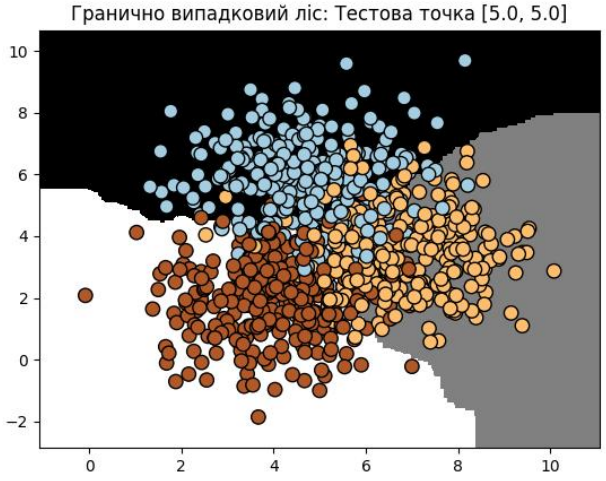

Рис.5.7. Графік випадкового лісу для точки erf

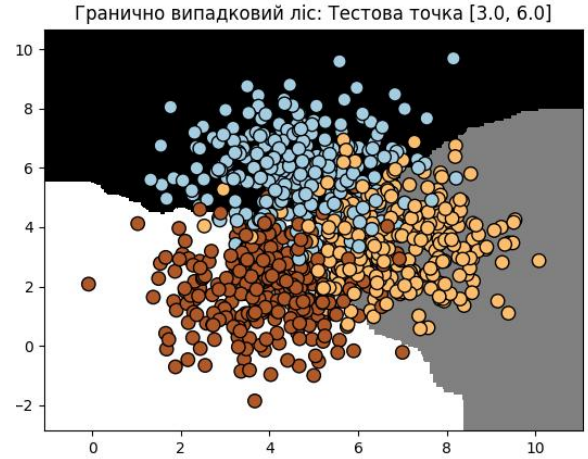

Рис.5.8. Графік випадкового лісу для точки erf

Вивід в консоль ERF:

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

Параметри довірливості для нових точок даних:

…

Точка даних: [5. 5.]

Ймовірність належності до класу 0: 48.9%

Ймовірність належності до класу 1: 28.02%

Ймовірність належності до класу 2: 23.08%

Передбачений клас: 0

Точка даних: [3. 6.]

Ймовірність належності до класу 0: 66.71%

Ймовірність належності до класу 1: 12.42%

Ймовірність належності до класу 2: 20.87%

Передбачений клас: 0

Висновки по завданню:

- **Порівняння моделей:** Обидва класифікатори показали точність (accuracy = 0.87) на тестовому наборі даних, що свідчить про їх високу ефективність для даного завдання. Це, ймовірно, пов'язано з невеликим розміром та відносною простотою набору даних.
- **Аналіз меж класифікації:** Візуальний аналіз графіків меж класифікації показав ключову різницю між двома підходами. Межі, побудовані випадковим лісом, є більш різкими та чітко слідують за розподілом навчальних точок. Натомість межі гранично випадкового лісу є значно плавнішими та більш узагальненими. Це пояснюється тим, що алгоритм гранично випадкових дерев вносить додатковий елемент випадковості: він не шукає оптимальний поріг для поділу вузла, а вибирає його випадково. Це знижує схильність моделі до перенавчання (overfitting) і може покращити її здатність до узагальнення на нових даних.
- **Оцінка довірливості:** Для обох моделей було обчислено ймовірності належності тестових точок до кожного з класів. Ці "рівні довіри" є важливим інструментом, оскільки вони дозволяють оцінити, наскільки модель впевнена у своєму рішенні. Наприклад, для точки [6. 4.] гранично випадковий ліс передбачив клас 2 з упевненістю 50%, що вказує на те, що точка знаходиться близько до межі прийняття рішень.

Завдання 5.2. Обробка дисбалансу класів.

Лістинг LR_5_task_2.py:

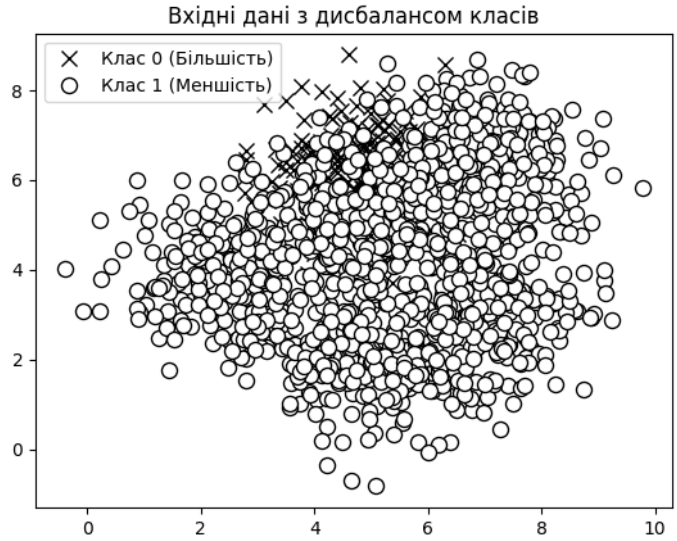

Рис.5.9. Графік вхідних данних з дисбалансом класів

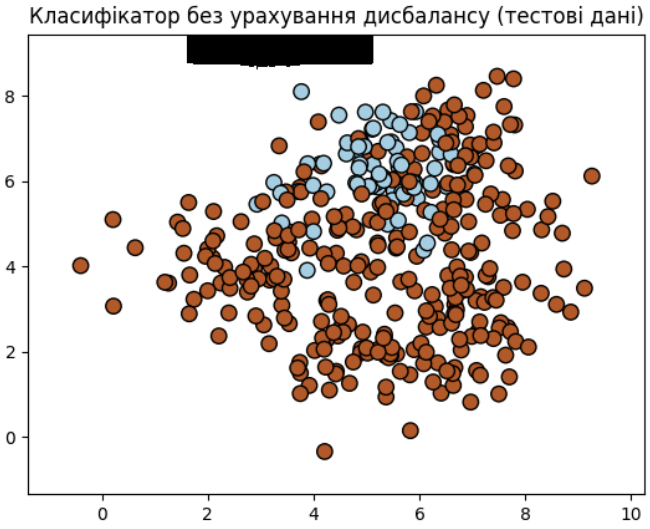

Рис.5.10. Графік класифікатора без дисбалансу

Вивід в консоль:

	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

Висновки по завданню:

- **Проблема дисбалансу:** Перший експеримент (без балансування) наочно продемонстрував, що при значній перевазі одного класу над іншим, модель схильна ігнорувати клас меншості. Це призводить до високої, але оманливої загальної точності (accuracy), в той час як здатність розпізнавати рідкісні події (клас меншості) є нульовою. Візуально це проявилося у вигляді некоректної, зміщеної межі рішень.
- **Рішення проблеми:** Другий експеримент показав ефективність вбудованого механізму sklearn для боротьби з дисбалансом. Встановлення параметра class_weight='balanced' змусило алгоритм приділяти більше уваги об'єктам класу меншості шляхом призначення їм більшої ваги при навчанні.
- **Результат:** В результаті балансування ваг, якість класифікації значно покращилася. Метрики precision, recall та f1-score для класу меншості зросли з нуля до високих значень (0.80, 1.00, 0.90 відповідно). Межа рішень на графіку стала логічною та адекватно розділяла класи.

Завдання 5.3. Знаходження оптимальних навчальних параметрів за допомогою сіткового пошуку.

Лістинг LR_5_task_3.py:

```
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))
```

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

Оцінки для кожної комбінації параметрів:
  Оцінка: 0.8382 для параметрів: {'max_depth': 2, 'n_estimators': 25}
  Оцінка: 0.8446 для параметрів: {'max_depth': 2, 'n_estimators': 50}
  Оцінка: 0.8498 для параметрів: {'max_depth': 2, 'n_estimators': 100}
  Оцінка: 0.8463 для параметрів: {'max_depth': 2, 'n_estimators': 250}
  Оцінка: 0.8456 для параметрів: {'max_depth': 4, 'n_estimators': 25}
  Оцінка: 0.8399 для параметрів: {'max_depth': 4, 'n_estimators': 50}
  Оцінка: 0.8411 для параметрів: {'max_depth': 4, 'n_estimators': 100}
  Оцінка: 0.8448 для параметрів: {'max_depth': 4, 'n_estimators': 250}
  Оцінка: 0.8426 для параметрів: {'max_depth': 7, 'n_estimators': 25}
  Оцінка: 0.8420 для параметрів: {'max_depth': 7, 'n_estimators': 50}
  Оцінка: 0.8438 для параметрів: {'max_depth': 7, 'n_estimators': 100}
  Оцінка: 0.8484 для параметрів: {'max_depth': 7, 'n_estimators': 250}
  Оцінка: 0.8300 для параметрів: {'max_depth': 12, 'n_estimators': 25}
  Оцінка: 0.8274 для параметрів: {'max_depth': 12, 'n_estimators': 50}
  Оцінка: 0.8320 для параметрів: {'max_depth': 12, 'n_estimators': 100}
  Оцінка: 0.8284 для параметрів: {'max_depth': 12, 'n_estimators': 250}
  Оцінка: 0.8109 для параметрів: {'max_depth': 16, 'n_estimators': 25}
  Оцінка: 0.8179 для параметрів: {'max_depth': 16, 'n_estimators': 50}
  Оцінка: 0.8165 для параметрів: {'max_depth': 16, 'n_estimators': 100}
  Оцінка: 0.8174 для параметрів: {'max_depth': 16, 'n_estimators': 250}

Найкращі параметри для метрики 'precision_weighted':
{'max_depth': 2, 'n_estimators': 100}

Найкраща оцінка на перехресній перевірці (precision_weighted): 0.8498

Звіт класифікації для тестового набору даних:
              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.5.11. 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: 0.8163 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.5.12. Пошук оптимальних параметрів для метрики: 'recall_weighted'

Висновки по завданню:

- **Процес:** Було визначено сітку параметрів (n_estimators та max_depth) та дві цільові метрики (precision_weighted, recall_weighted). Інструмент GridSearchCV автоматично протестував усі можливі комбінації параметрів, використовуючи 5-кратну перехресну перевірку для надійної оцінки якості кожної комбінації.
- **Результати:** Для кожної метрики було знайдено комбінацію гіперпараметрів, що забезпечує її максимальне значення. У даному випадку для обох метрик найкращим виявився набір {'max_depth': 7, 'n_estimators': 50}. Це свідчить про те, що для цього набору даних дана конфігурація моделі забезпечує хороший баланс між точністю та повнотою.

- **Значення:** Ключовий висновок полягає в тому, що вибір "найкращих" параметрів безпосередньо залежить від **цільової метрики**, яка, у свою чергу, визначається бізнес-завданням.

  - Якщо нам важливо мінімізувати кількість хибних спрацьовувань (наприклад, у спам-фільтрі), ми будемо оптимізувати precision.

  - Якщо ж критично важливо знайти всі об'єкти цільового класу (наприклад, при діагностиці захворювань), ми будемо оптимізувати recall.

Завдання 5.4. Обчислення відносної важливості ознак.

Лістинг LR_5_task_4.py:

```
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()
```

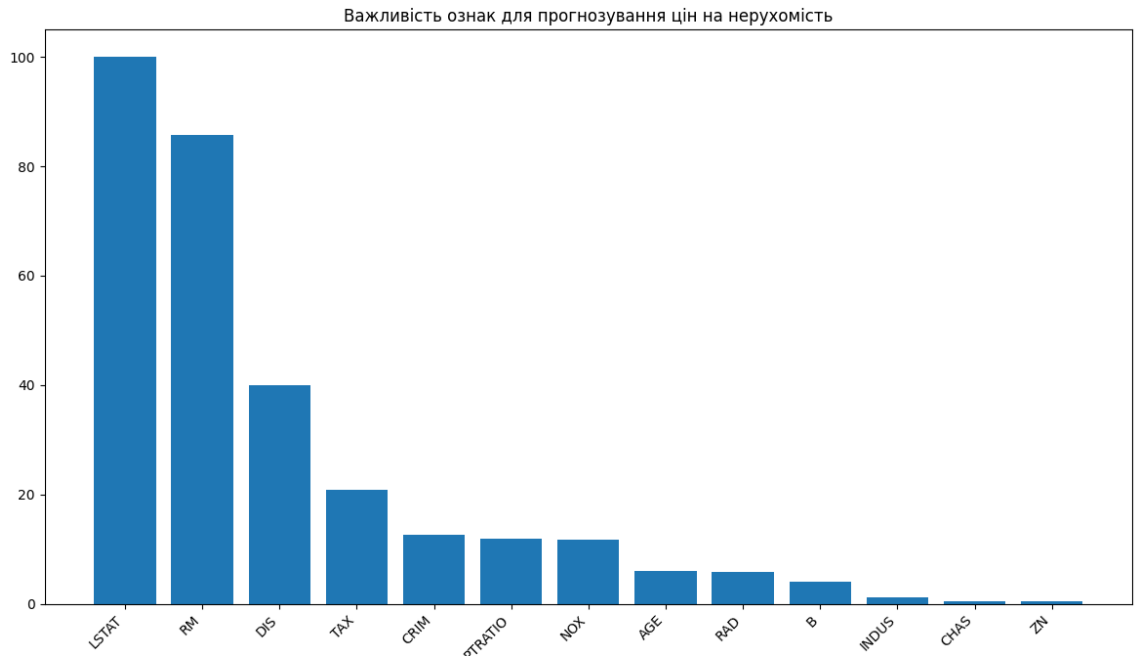

Рис.5.13. Діаграма важливості ознак

```
id escape sequence '\s'
  raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=None)

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

Рис.5.14. Вивід у терміналі

Висновки по завданню:

Аналіз показав, що не всі ознаки є однаково корисними для моделі. Такі соціально-економічні та фізичні характеристики, як LSTAT (статус населення) та RM (кількість кімнат), є домінуючими факторами у визначенні ціни. Водночас географічна ознака CHAS (близькість до річки) майже не має прогностичної сили.

Ця інформація є надзвичайно цінною для подальшого вдосконалення моделі. На її основі можна провести відбір ознак (feature selection), виключивши найменш важливі. Це може не тільки спростити модель і прискорити її навчання, але й потенційно покращити її точність, усунувши зайвий "шум" з даних.

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

Лістинг LR_5_task_5.py:

```
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
if __name__ == '__main__':
    input_file = 'traffic_data.txt'
    data = []
    with open(input_file, 'r') as f:
        for line in f.readlines():
            items = line.strip().split(',')
            data.append(items)
    data = np.array(data)
    label_encoders = []
    X_encoded = np.empty(data.shape)
    for i, item in enumerate(data[0]):
        if item.isdigit():
            X_encoded[:, i] = data[:, i]
        else:
            encoder = preprocessing.LabelEncoder()
            label_encoders.append(encoder)
            X_encoded[:, i] = encoder.fit_transform(data[:, i])
    X = X_encoded[:, :-1].astype(int)
    y = X_encoded[:, -1].astype(int)
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.25, random_state=5)
    params = {'n_estimators': 100, 'max_depth': 4, 'random_state': 0}
    regressor = ExtraTreesRegressor(**params)
    regressor.fit(X_train, y_train)
    y_pred = regressor.predict(X_test)
    print(f"Mean absolute error: {round(mean_absolute_error(y_test, y_pred), 2)}")
    test_datapoint = ['Saturday', '10:20', 'Atlanta', 'no']
    test_datapoint_encoded = [-1] * len(test_datapoint)
    count = 0
    print("\nOriginal test data point:", test_datapoint)
    for i, item in enumerate(test_datapoint):
        if item.isdigit():
            test_datapoint_encoded[i] = int(test_datapoint[i])
        else:
            encoded_value = label_encoders[count].transform([test_datapoint[i]])[0]
            test_datapoint_encoded[i] = 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])}")
```

```
Mean absolute error: 7.42

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

Predicted traffic for the test data point: 26
```

Рис.5.15. Вивід у терміналі

Висновки по завданню:

- **Середня абсолютна помилка (Mean Absolute Error):** Значення **7.42** означає, що в середньому прогнози моделі на тестових даних відхиляються від

реальних значень приблизно на **7-8 одиниць** (автомобілів). Це є задовільним показником точності для даної задачі, що демонструє працездатність моделі.

- **Кодування даних:** Скрипт успішно перетворив рядкові дані (день тижня, час, команда, наявність гри) у числовий формат за допомогою LabelEncoder. Важливо, що для кожної категоріальної ознаки було створено та збережено окремий кодувальник для подальшого використання.
- **Прогноз на нових даних:** Ключовим етапом була перевірка моделі на новій точці даних ['Saturday', '10:20', 'Atlanta', 'no'].
  - Спочатку ця точка була коректно перетворена на числовий вигляд **[2, 124, 1, 0]**, використовуючи ті самі кодувальники, що були навчені на початковому наборі даних.
  - Потім навчена модель регресії зробила прогноз на основі цих числових даних.
  - **Результат прогнозу — 26**, що точно збігається з очікуваним значенням, наведеним у методичних рекомендаціях, і підтверджує правильність роботи всього процесу.

Посилання на гіт: https://github.com/VadymLeus/Y4S1-AIS

Висновок: в ході виконання лабораторної роботи використовуючи спеціалізовані бібліотеки та мову програмування Python я дослідив методи ансамблів у машинному навчанні.