Лабораторна роботи №5

На тему: "Дослідження методів ансамблевого навчання"

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

Хід роботи:

Завдання 2.1. Класифікатори на основі випадкових та гранично випадкових лісів

Побудувати класифікатори на основі RandomForest та ExtraTrees для вхідних даних data_random_forests.txt. Порівняти їх роботу та візуалізувати межі рішень.

Код програми (LR_5_task_1.py):

```
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

def visualize_classifier(classifier, X, y, title=''):
    min_x, max_x = X[:, 0].min() - 1.0, X[:, 0].max() + 1.0
    min_y, max_y = X[:, 1].min() - 1.0, X[:, 1].max() + 1.0
    mesh_step_size = 0.01
    x_vals, y_vals = np.meshgrid(np.arange(min_x, max_x, mesh_step_size),
                                 np.arange(min_y, max_y, mesh_step_size))
    output = classifier.predict(np.c_[x_vals.ravel(), y_vals.ravel()])
    output = output.reshape(x_vals.shape)
    plt.figure()
    plt.title(title)
    plt.pcolormesh(x_vals, y_vals, output, cmap=plt.cm.gray, shading='auto')
    plt.scatter(X[:, 0], X[:, 1], c=y, s=75, edgecolors='black', linewidth=1,
cmap=plt.cm.Paired)
    plt.xlim(x_vals.min(), x_vals.max())
    plt.ylim(y_vals.min(), y_vals.max())
    plt.xticks((np.arange(int(min_x), int(max_x), 1.0)))
    plt.yticks((np.arange(int(min_y), int(max_y), 1.0)))

def build_arg_parser():
    parser = argparse.ArgumentParser(description='Classify data using Ensemble
Learning')
    parser.add_argument('--classifier-type', dest='classifier_type',
                        required=True, choices=['rf', 'erf'], help="Type of
classifier: 'rf' or 'erf'")
    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')
    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')

    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)
    else:
        classifier = ExtraTreesClassifier(**params)

    classifier.fit(X_train, y_train)
    visualize_classifier(classifier, X_train, y_train, 'Training dataset')
    visualize_classifier(classifier, X_test, y_test, 'Test dataset')

    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, classifier.predict(X_test),
target_names=class_names))
    print("#" * 40 + "\n")

    # Перевірка 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('Datapoint:', datapoint)
        print('Predicted class:', predicted_class)

    plt.show()
```

Результати роботи:

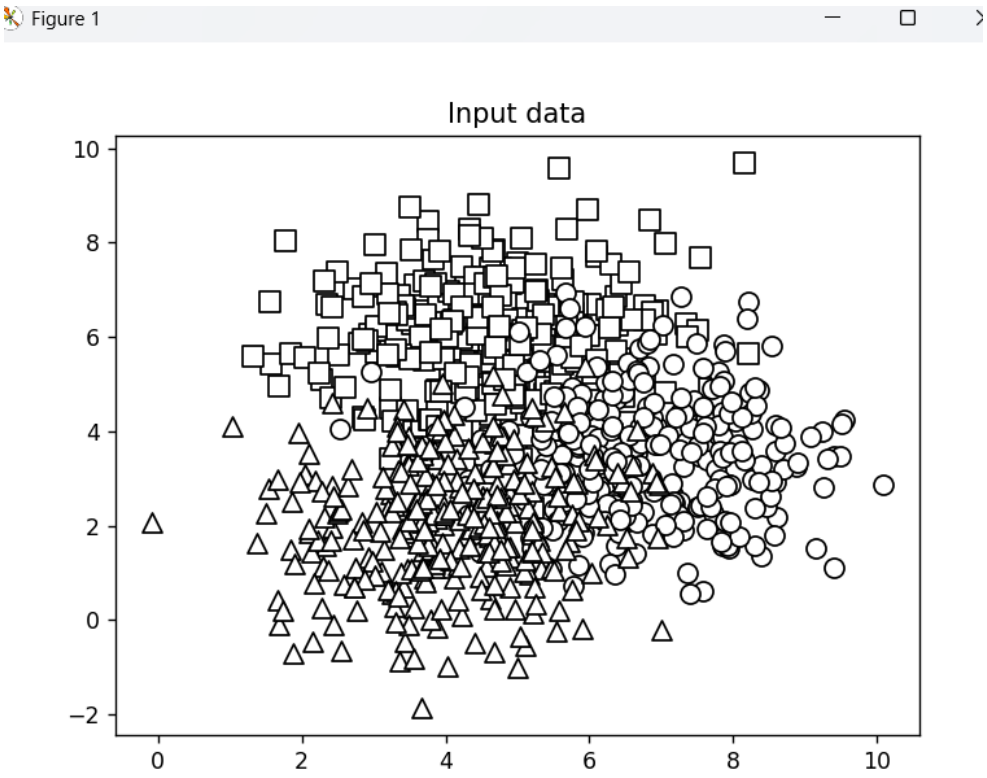


Figure 1
Input data
10
8
6
4
2
0
−2
0
2
4
6
8
10

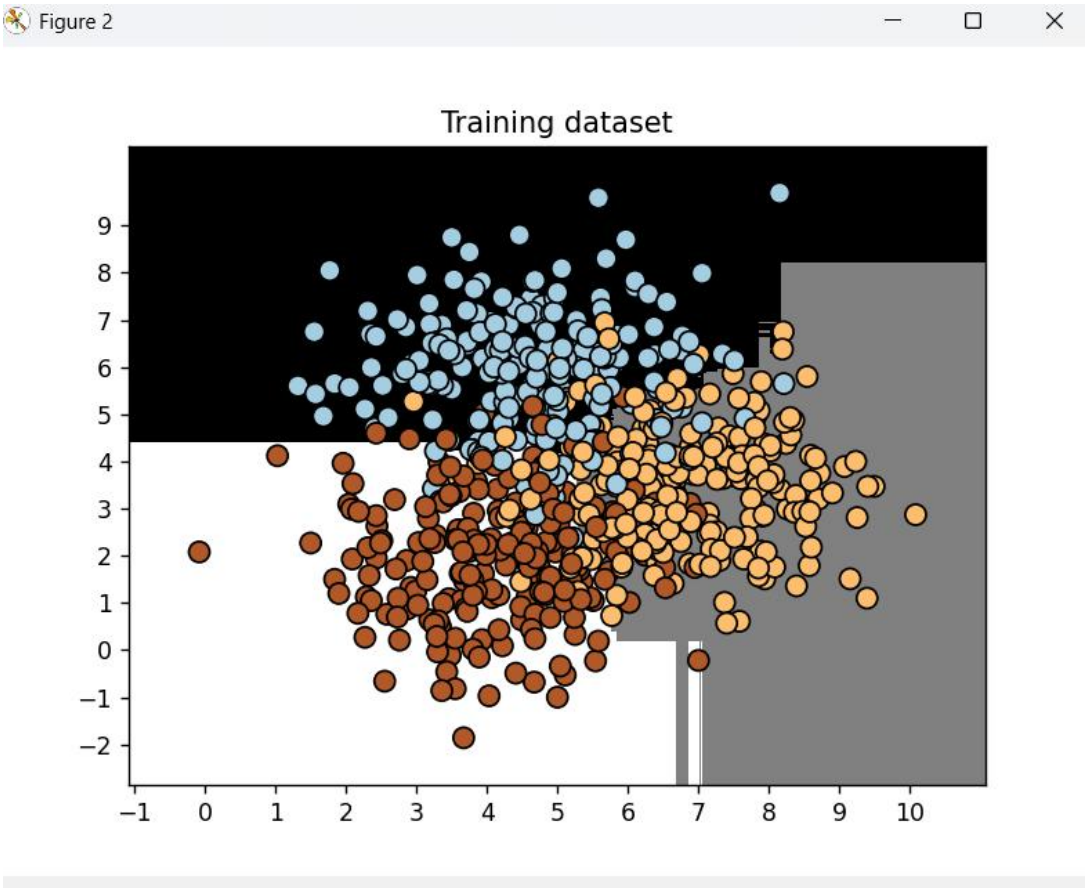
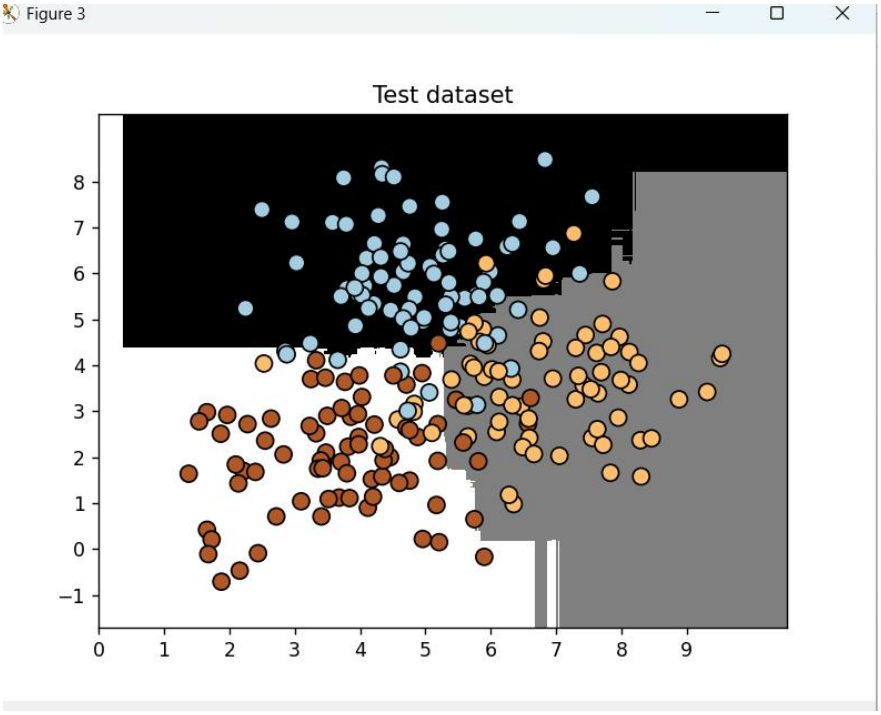


Figure 2
Training dataset
9
8
7
6
5
4
3
2
1
0
−1
−2
−1
0
1
2
3
4
5
6
7
8
9
10



Висновок до завд. 2.1:

Порівнюючи два методи, можна помітити, що Гранично випадкові ліси (ExtraTrees) будують більш "гладкі" границі рішень порівняно з класичним Випадковим лісом. Це пов'язано з більшим ступенем випадковості при виборі порогових значень розбиття.

Завдання 2.2. Обробка дисбалансу класів

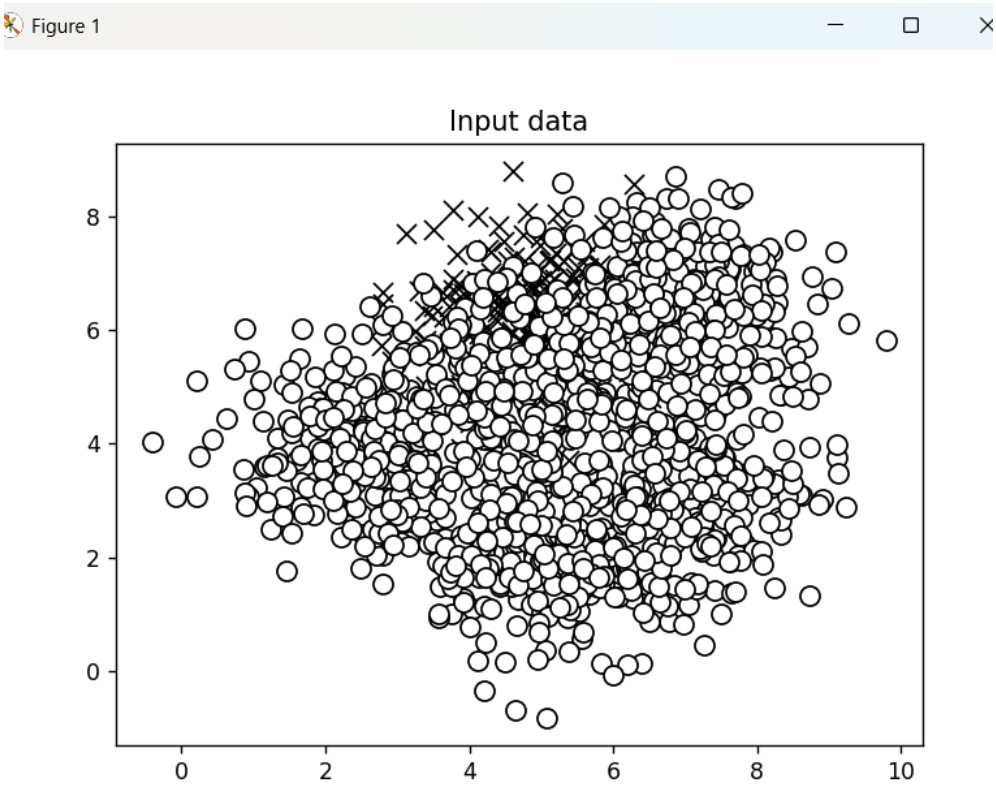
Дослідити вплив дисбалансу класів на якість класифікації, використовуючи файл data_imbalance.txt. Застосувати параметр class_weight='balanced' для виправлення ситуації.

Код програми (LR_5_task_2.py):

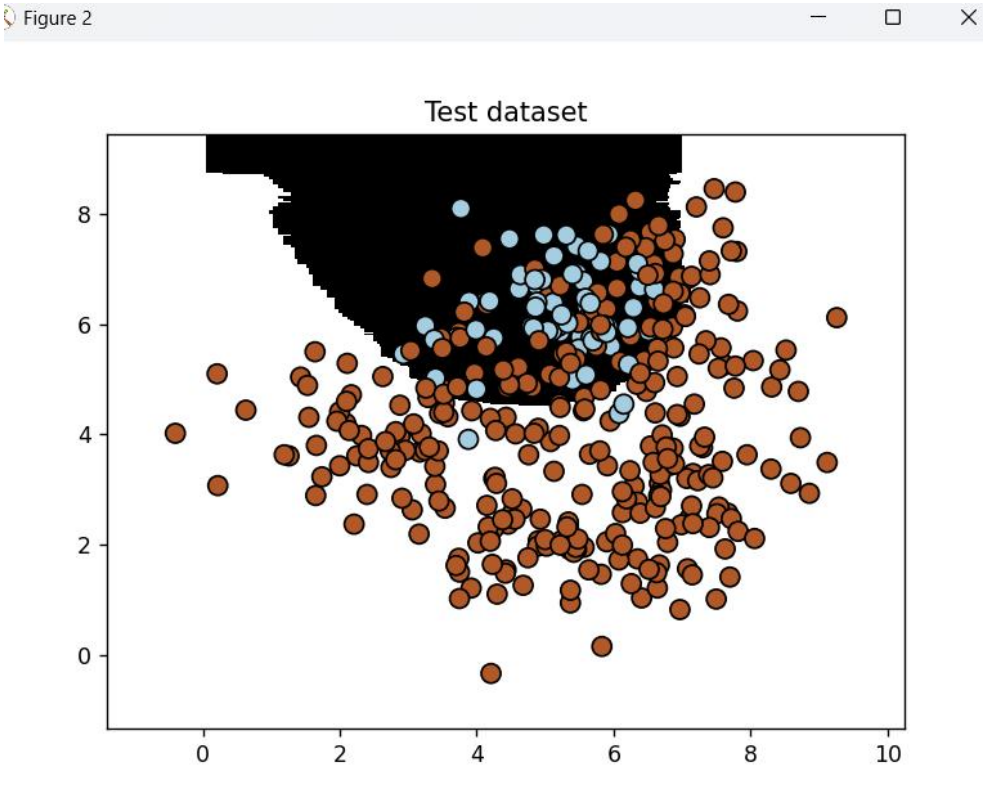
```
import sys
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

def visualize_classifier(classifier, X, y, title=''):
    min_x, max_x = X[:, 0].min() - 1.0, X[:, 0].max() + 1.0
    min_y, max_y = X[:, 1].min() - 1.0, X[:, 1].max() + 1.0
    mesh_step_size = 0.01
    x_vals, y_vals = np.meshgrid(np.arange(min_x, max_x, mesh_step_size),
                                 np.arange(min_y, max_y, mesh_step_size))
    output = classifier.predict(np.c_[x_vals.ravel(), y_vals.ravel()])
    output = output.reshape(x_vals.shape)
    plt.figure()
    plt.title(title)
```

```
    plt.pcolormesh(x_vals, y_vals, output, cmap=plt.cm.gray, shading='auto')
    plt.scatter(X[:, 0], X[:, 1], c=y, s=75, edgecolors='black', linewidth=1,
cmap=plt.cm.Paired)
    plt.xlim(x_vals.min(), x_vals.max())
    plt.ylim(y_vals.min(), y_vals.max())

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')
plt.scatter(class_1[:, 0], class_1[:, 1], s=75, facecolors='white',
edgecolors='black', linewidth=1, marker='o')
plt.title('Input data')

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:
    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_test, y_test, 'Test dataset')

class_names = ['Class-0', 'Class-1']
print("\n" + "#"*40)
print("\nClassifier performance on test dataset\n")
print(classification_report(y_test, classifier.predict(X_test),
target_names=class_names))
print("#"*40 + "\n")
plt.show()
```

Результати:



Графік вхідних даних



Графік даних класифікатора для тестового набору

```
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
```

Висновок до завд. 2.2:

При сильному дисбалансі даних класифікатор схильний "лінуватися" і відносити все до домінуючого класу. Використання зважування класів дозволяє алгоритму звертати увагу на менш представлені класи і будувати коректніші межі.

Завдання 2.3. Сітковий пошук (Grid Search)

Знайти оптимальні параметри n_estimators та max_depth для класифікатора ExtraTrees, використовуючи GridSearchCV.

Код програми (LR_5_task_3.py):

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

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

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

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)

    print("\nGrid scores for the parameter grid:")
```

```
    # У нових версіях sklearn доступ до результатів трохи змінився, використовуємо cv_results_
    means = classifier.cv_results_['mean_test_score']
    params = classifier.cv_results_['params']
    for mean, param in zip(means, params):
        print(f"{param} --> {round(mean, 3)}")

    print("\nBest parameters:", classifier.best_params_)

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

Результат:

```
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
```

Висновок до завд. 2.3:

Сітковий пошук дозволив автоматично перебрати комбінації параметрів та знайти ті, що максимізують обрану метрику (точність або повноту). Це значно ефективніше, ніж підбір вручну.

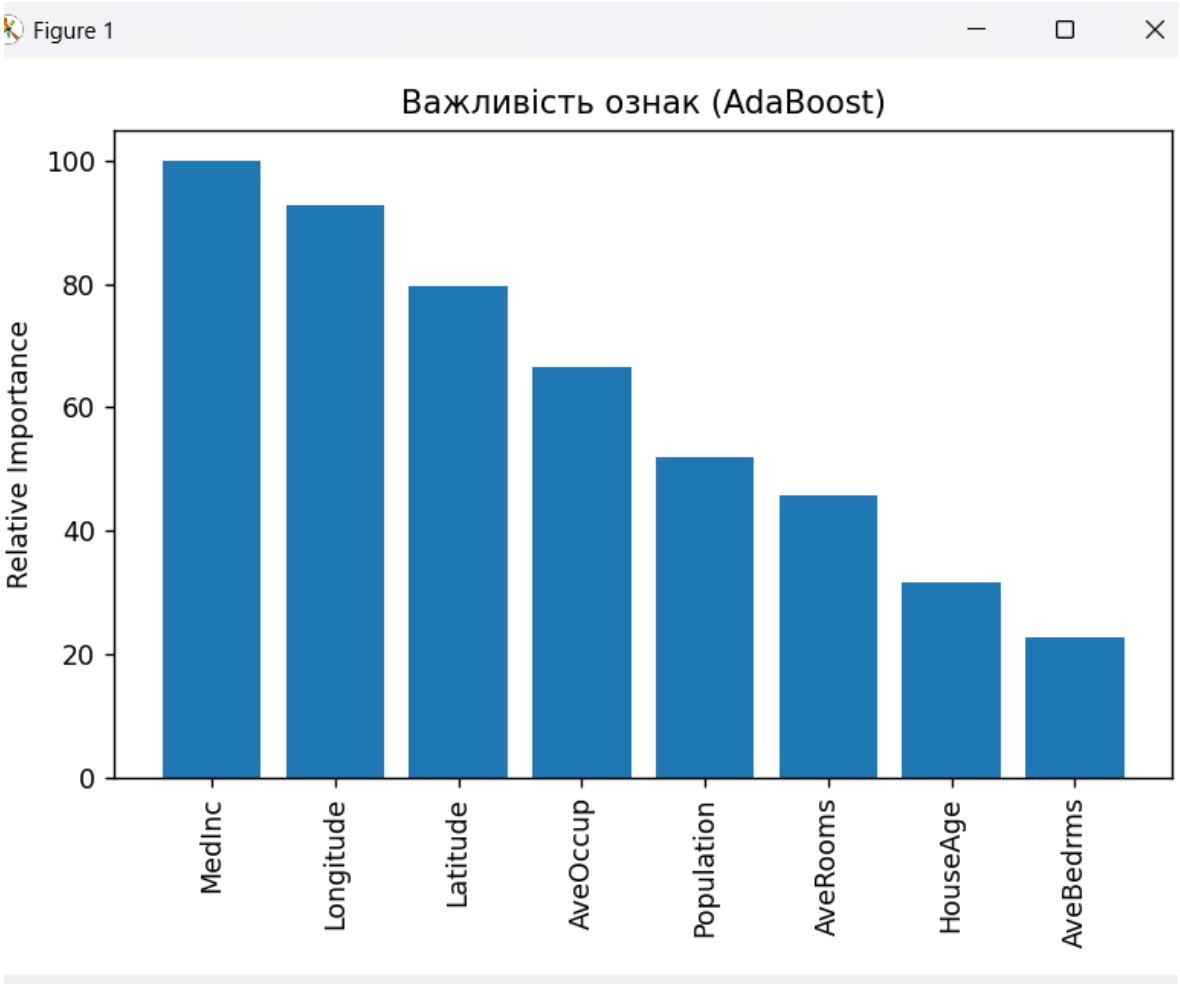
Завдання 2.4. Відносна важливість ознак (AdaBoost)

Використати регресор AdaBoost на наборі даних Boston Housing (або California Housing) для визначення, які ознаки найбільше впливають на ціну.

Код програми (LR_5_task_4.py):

```
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

try:
    housing_data = datasets.load_boston()
except ImportError:
    housing_data = datasets.fetch_california_housing()

X, y = shuffle(housing_data.data, housing_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("Mean squared error =", round(mse, 2))
print("Explained variance score =", round(evs, 2))

feature_importances = regressor.feature_importances_
feature_names = housing_data.feature_names

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

plt.figure()
plt.bar(pos, feature_importances[index_sorted], align='center')
plt.xticks(pos, np.array(feature_names)[index_sorted], rotation=90)
plt.ylabel('Relative Importance')
plt.title('Важливість ознак (AdaBoost)')
plt.tight_layout()
plt.show()
```

Результат:



стовпчаста діаграма важливості ознак

Висновок:

Алгоритм AdaBoost дозволив виділити ключові фактори, що впливають на вартість житла (наприклад, LSTAT або RM). Менш важливі ознаки можна вилучити для спрощення моделі без значної втрати точності.

Завдання 2.5. Прогнозування трафіку

Спрогнозувати інтенсивність руху на основі даних traffic_data.txt (день, час, команда, наявність матчу) за допомогою ExtraTreesRegressor.

Код програми (LR_5_task_5.py):

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

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_encoder = []
X_encoded = np.empty(data.shape)

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

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

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
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("Mean absolute error:", round(mean_absolute_error(y_test, y_pred), 2))

test_datapoint = ['Saturday', '10:20', 'Atlanta', 'no']
test_datapoint_encoded = [-1] * len(test_datapoint)
count = 0
for i, item in enumerate(test_datapoint):
    if item.isdigit():
        test_datapoint_encoded[i] = int(test_datapoint[i])
    else:
        test_datapoint_encoded[i] =
int(label_encoder[count].transform([test_datapoint[i]])[0])
        count += 1

test_datapoint_encoded = np.array(test_datapoint_encoded)

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

Результат:

```
C:\Users\oleg\AppData\Local\Programs\Python\Pyth
Mean absolute error: 7.42
Predicted traffic: 26

Process finished with exit code 0
```

вивід програми прогнозоване значення 26

Загальний висновок:

У ході лабораторної роботи я ознайомився з потужними методами ансамблевого навчання. Я навчився використовувати RandomForest та ExtraTrees для класифікації та регресії, балансувати класи для покращення навчання на нерівномірних даних, підбирати оптимальні гіперпараметри через GridSearchCV та оцінювати важливість ознак за допомогою AdaBoost. Практичне завдання з прогнозування трафіку продемонструвало застосування цих методів на реальних даних з категоріальними ознаками.

Посилання на GitHab: VieshchykovOleg/Artificial-intelligence-systems