ЛАБОРАТОРНА РОБОТА № 5

ДОСЛІДЖЕННЯ МЕТОДІВ АНСАМБЛЕВОГО НАВЧАННЯ.

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

Хід роботи

Завдання 2.1. Створення класифікаторів на основі випадкових та гранично випадкових лісів.

Лістинг 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 matplotlib.colors import ListedColormap

# 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


def visualize_classifier(classifier, X, y, title):
    # Визначення меж сітки
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01),
                         np.arange(y_min, y_max, 0.01))

    # Прогноз для кожної точки сітки
    Z = classifier.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    # Побудова графіка
    plt.contourf(xx, yy, Z, alpha=0.3, cmap=ListedColormap(['#FFAAAA', '#AAFFAA',
'#AAAAFF']))
    plt.scatter(X[:, 0], X[:, 1], c=y, s=75, edgecolor='k',
cmap=ListedColormap(['#FF0000', '#00FF00', '#0000FF']))
    plt.title(title)
    plt.show()


if __name__=='__main__':
    # Parse the input arguments
```

					ДУ «Житомирська політехніка».24.122.09.000 – Лр5		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Марчук Н.А.			Звіт з лабораторної роботи	Літ.	Арк. 1 / Аркушів
Перевір.		Маєвський О.В.					
Керівник						ФІКТ Гр. КН-21-1[1]	
Н. контр.							
Зав. каф.							

```
    args = build_arg_parser().parse_args()
    classifier_type = args.classifier_type

    # Load input data
    input_file = '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)
```

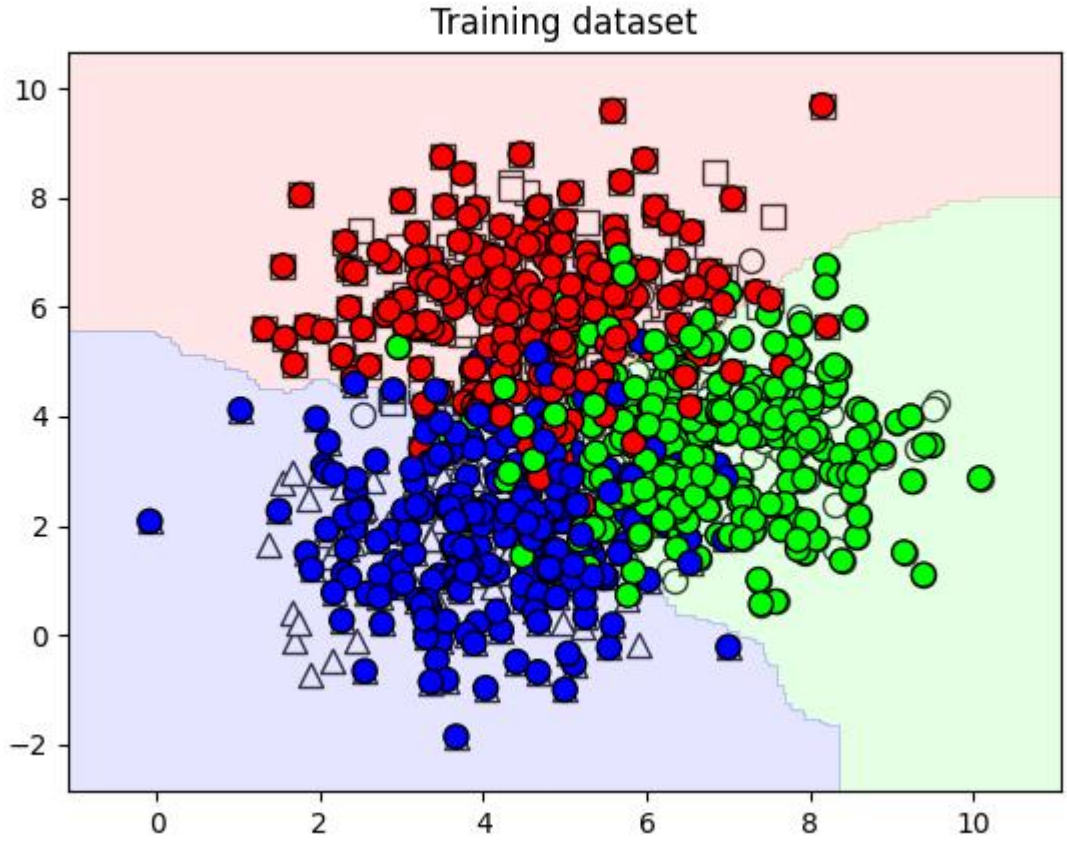
```
    # Visualize the datapoints
    visualize_classifier(classifier, test_datapoints, [0]*len(test_datapoints),
            'Test datapoints')

    plt.show()
```

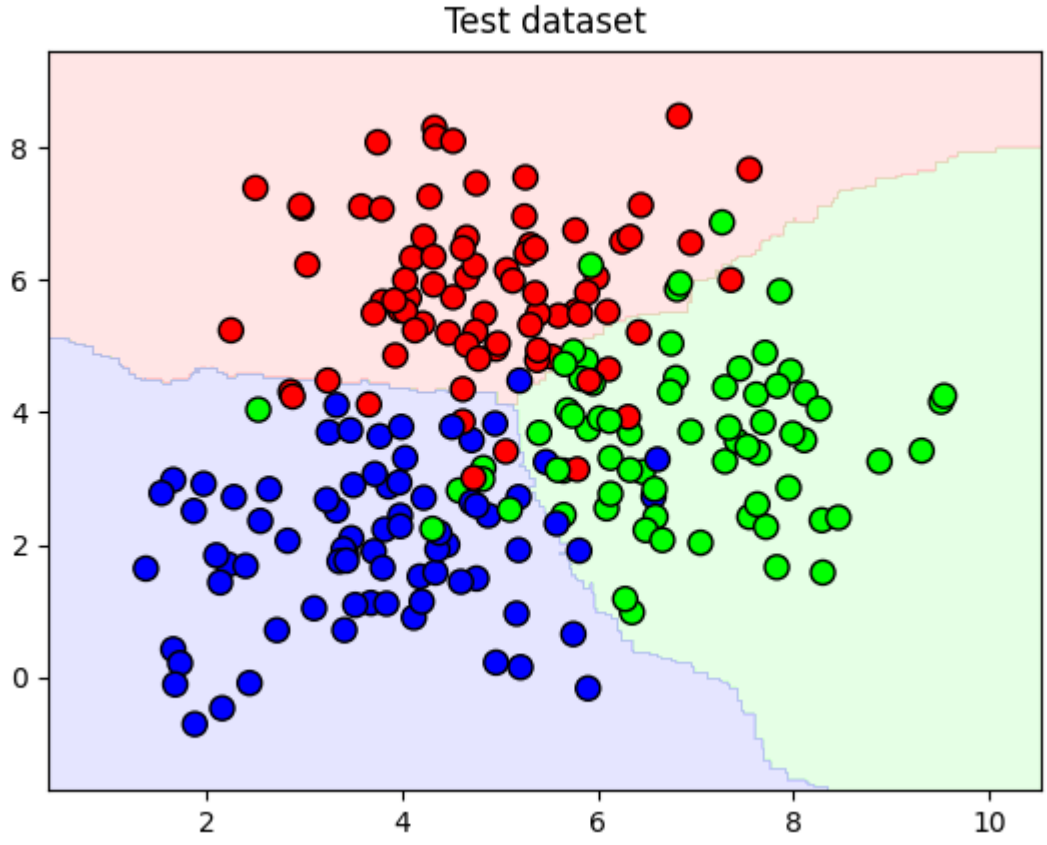
Виконання програми:

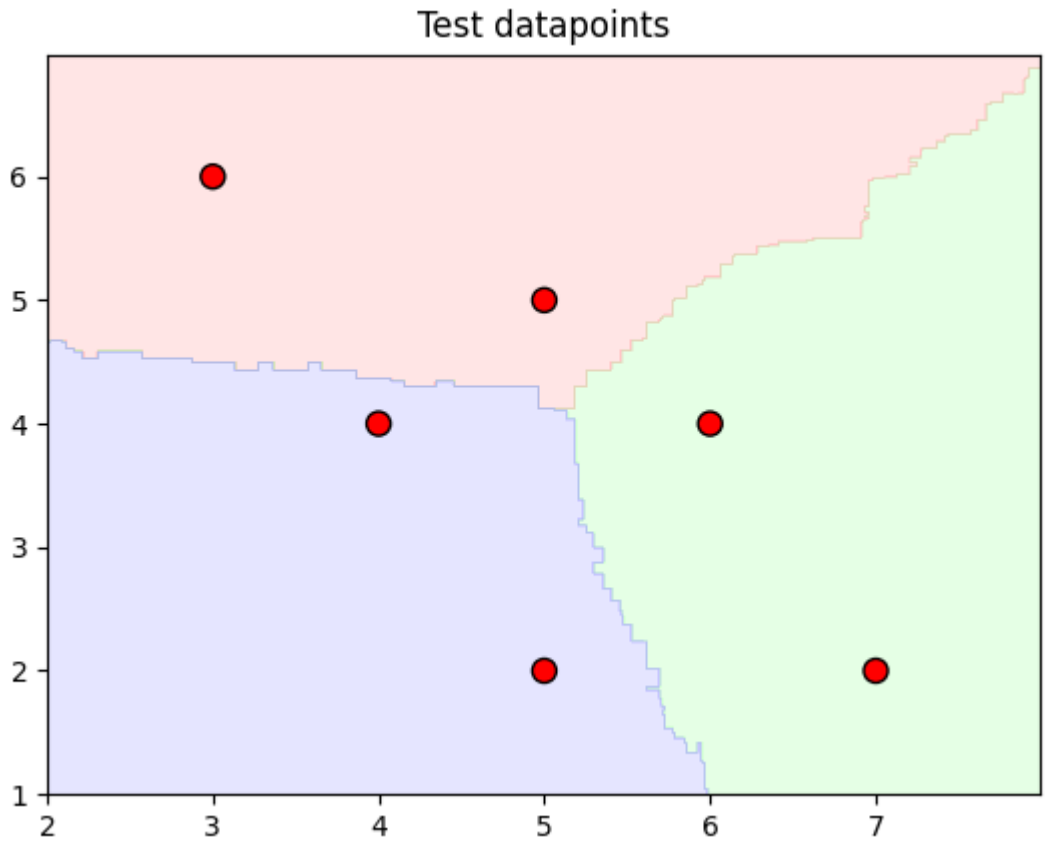
--classifier-type rf

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

Classifier performance on training dataset

              precision    recall  f1-score   support

     Class-0       0.91      0.86      0.88       221
     Class-1       0.84      0.87      0.86       230
     Class-2       0.86      0.87      0.86       224

    accuracy                           0.87       675
   macro avg       0.87      0.87      0.87       675
weighted avg       0.87      0.87      0.87       675

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

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

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

Process finished with exit code 0
```


Training dataset
10
8
6
4
2
0
−2
0
2
4
6
8
10

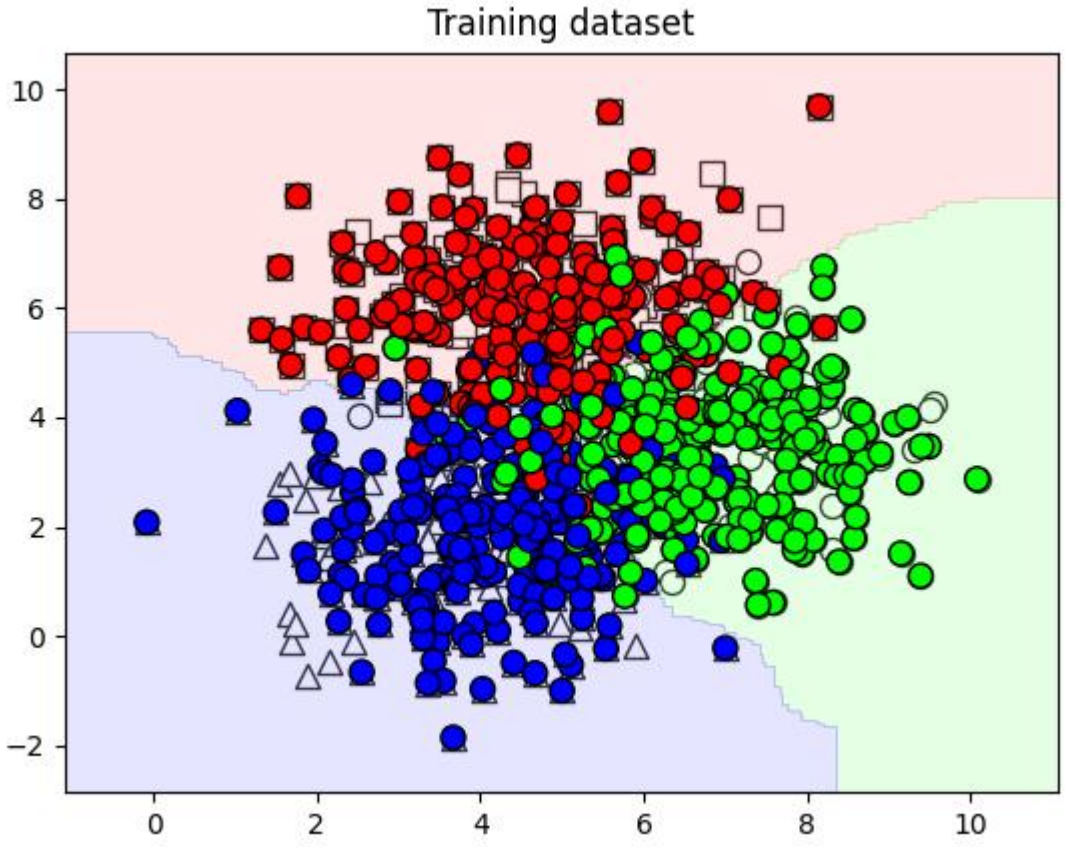

Test dataset
8
6
4
2
0
2
4
6
8
10

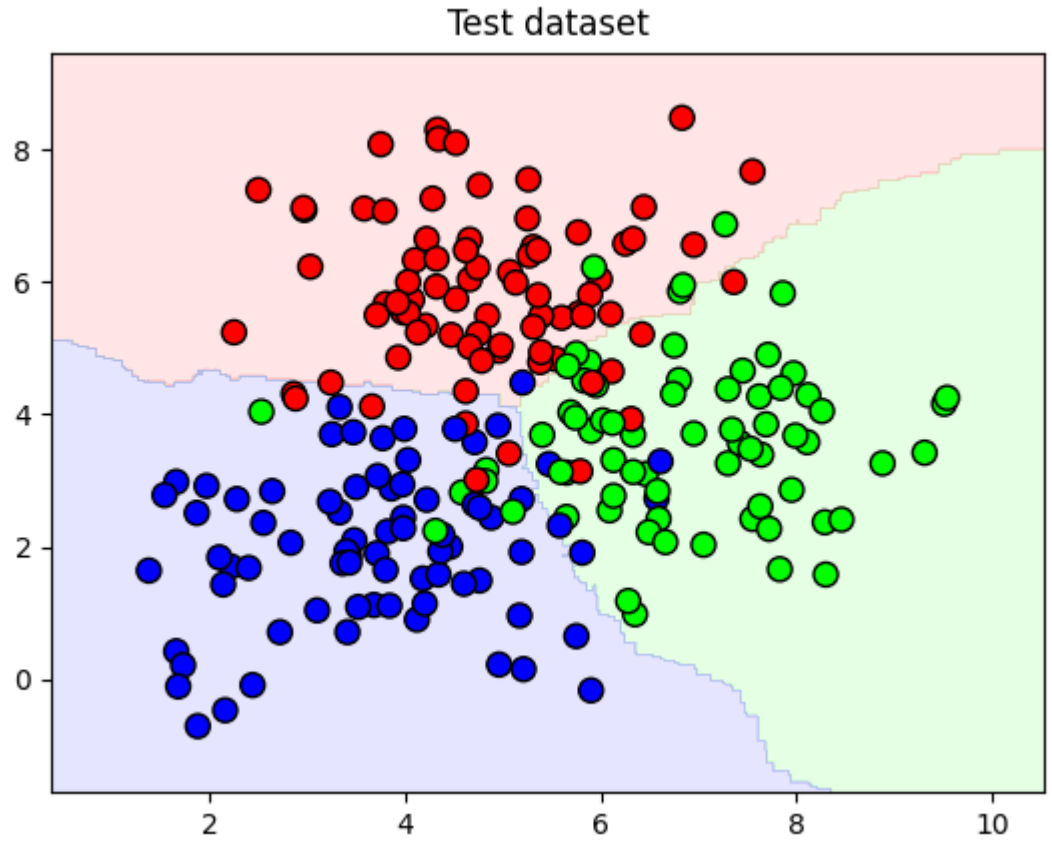


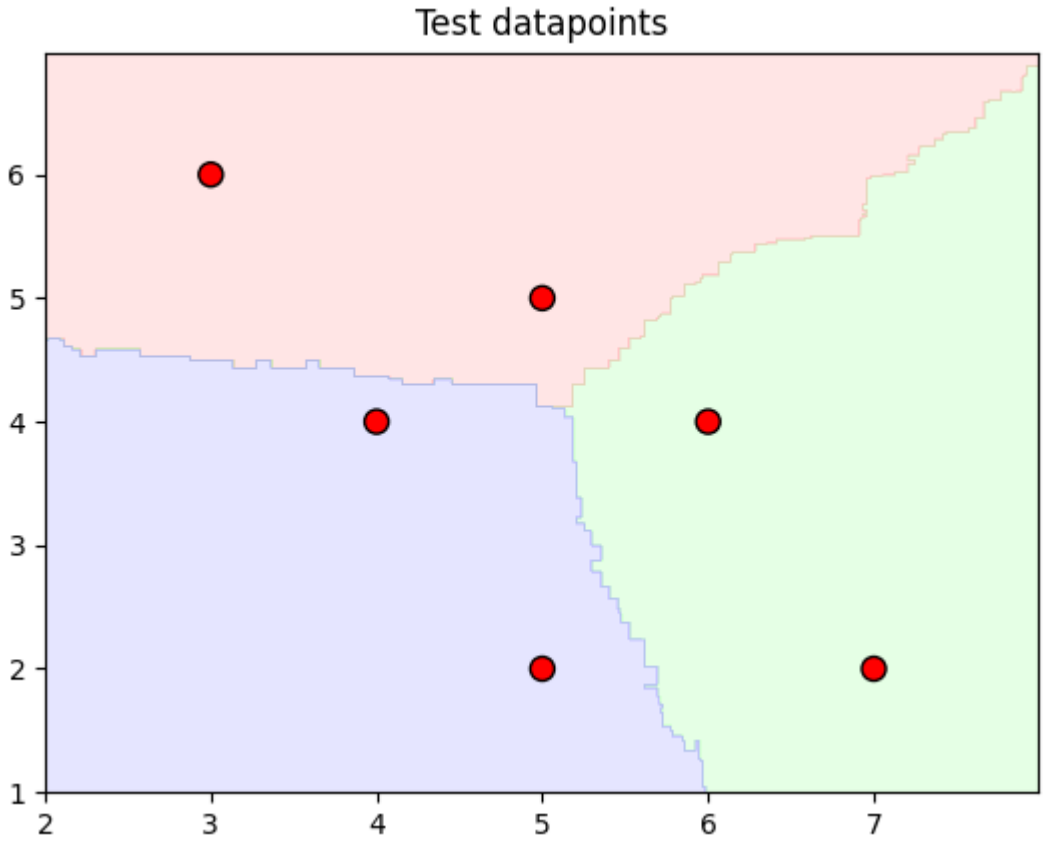
--classifier-type erf

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

Classifier performance on training dataset

              precision    recall  f1-score   support

     Class-0       0.89      0.83      0.86       221
     Class-1       0.82      0.84      0.83       230
     Class-2       0.83      0.86      0.85       224

    accuracy                           0.85       675
   macro avg       0.85      0.85      0.85       675
weighted avg       0.85      0.85      0.85       675

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

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

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

Process finished with exit code 0
```







Завдання 2.2. Обробка дисбалансу класів

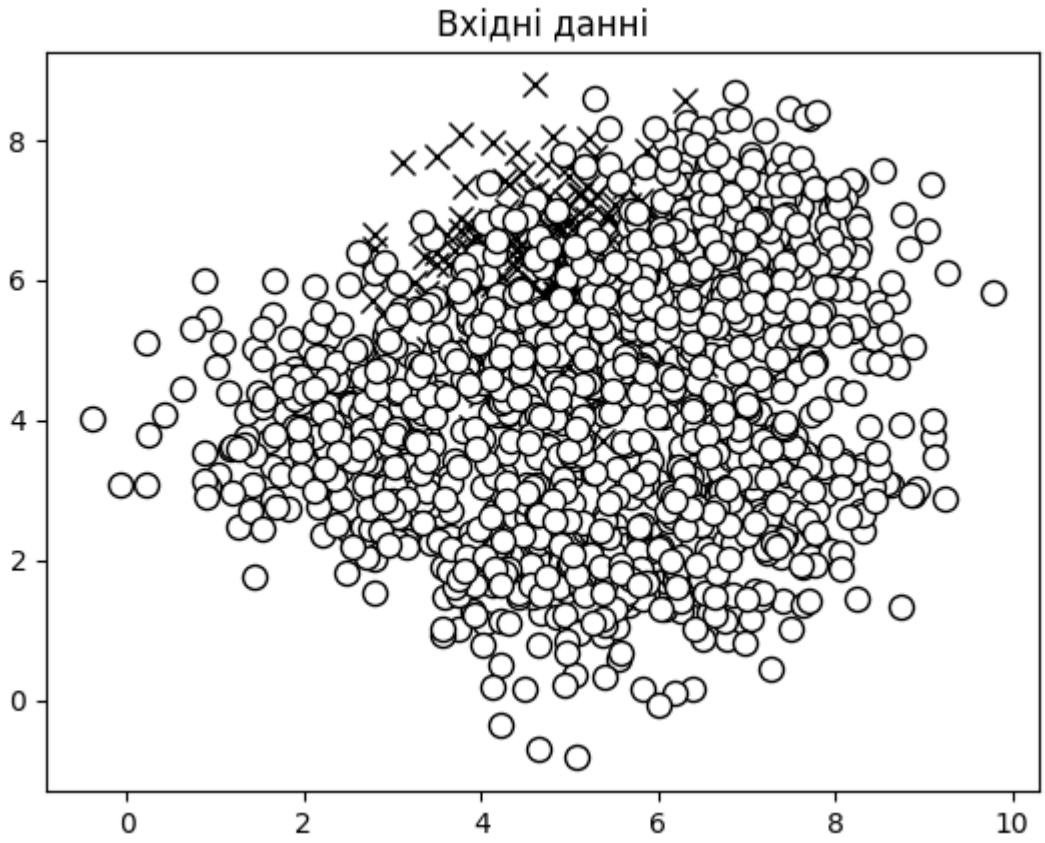
Лістинг програми:

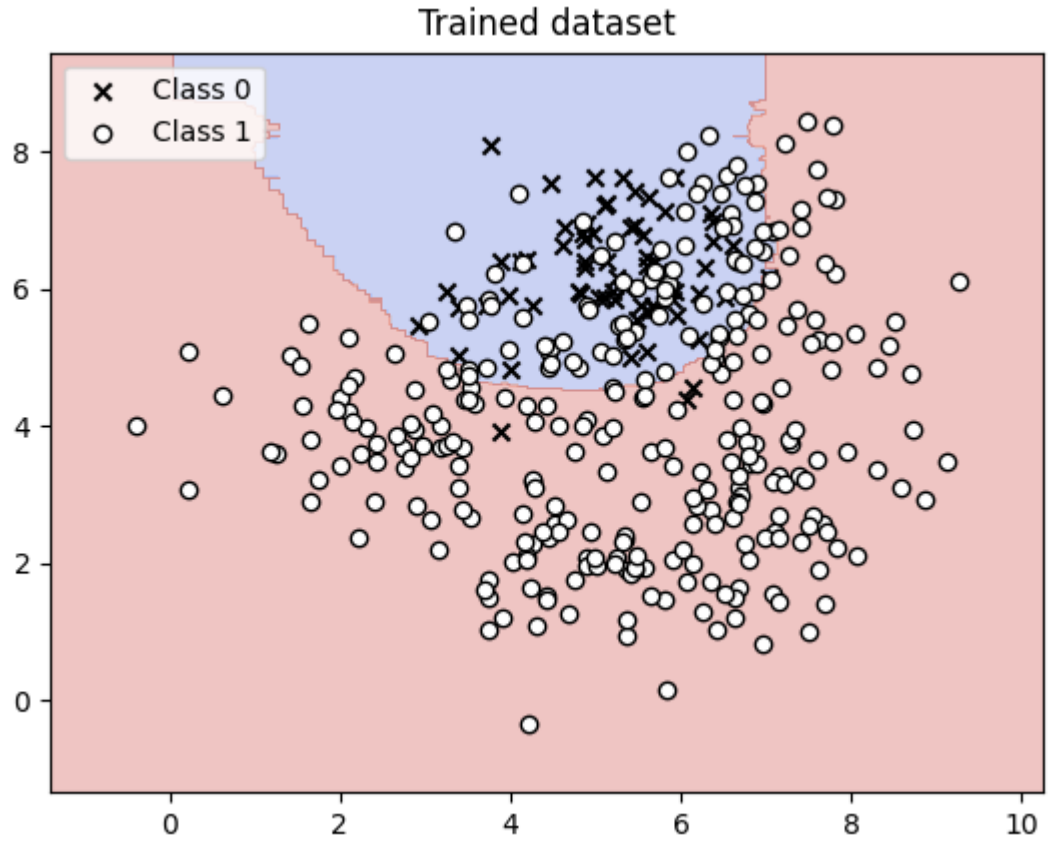
```
import sys
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.metrics import classification_report

def visualize_classifier(classifier, X, y, title='Classifier boundaries'):
    # Визначення меж для сітки
    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  # Крок сітки для відображення області рішень

    # Визначення сітки точок для області рішень
    x_vals, y_vals = np.meshgrid(np.arange(min_x, max_x, mesh_step_size),
                                 np.arange(min_y, max_y, mesh_step_size))

    # Прогнозування для кожної точки на сітці
    output = classifier.predict(np.c_[x_vals.ravel(), y_vals.ravel()])
    output = output.reshape(x_vals.shape)

    # Створення графіку
    plt.figure()
    plt.title(title)

    # Відображення областей рішень
    plt.contourf(x_vals, y_vals, output, cmap=plt.cm.coolwarm, alpha=0.3)

    # Відображення вхідних точок
    plt.scatter(X[y == 0][:, 0], X[y == 0][:, 1], c='black', marker='x',
label='Class 0')
    plt.scatter(X[y == 1][:, 0], X[y == 1][:, 1], c='white', edgecolors='black',
marker='o', label='Class 1')

    # Додавання легенди та показ графіку
    plt.legend()
    plt.show()

# Зчитування даних
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])

# Візуалізація вхідних даних
plt.figure()
plt.scatter(class_0[:, 0], class_0[:, 1], s=75, facecolors='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('Вхідні данні')

# Розділення даних на тренувальні та тестові
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
random_state=5)

# Налаштування параметрів для класифікатора
params = {'n_estimators': 100, 'max_depth': 4, 'random_state': 0}
```

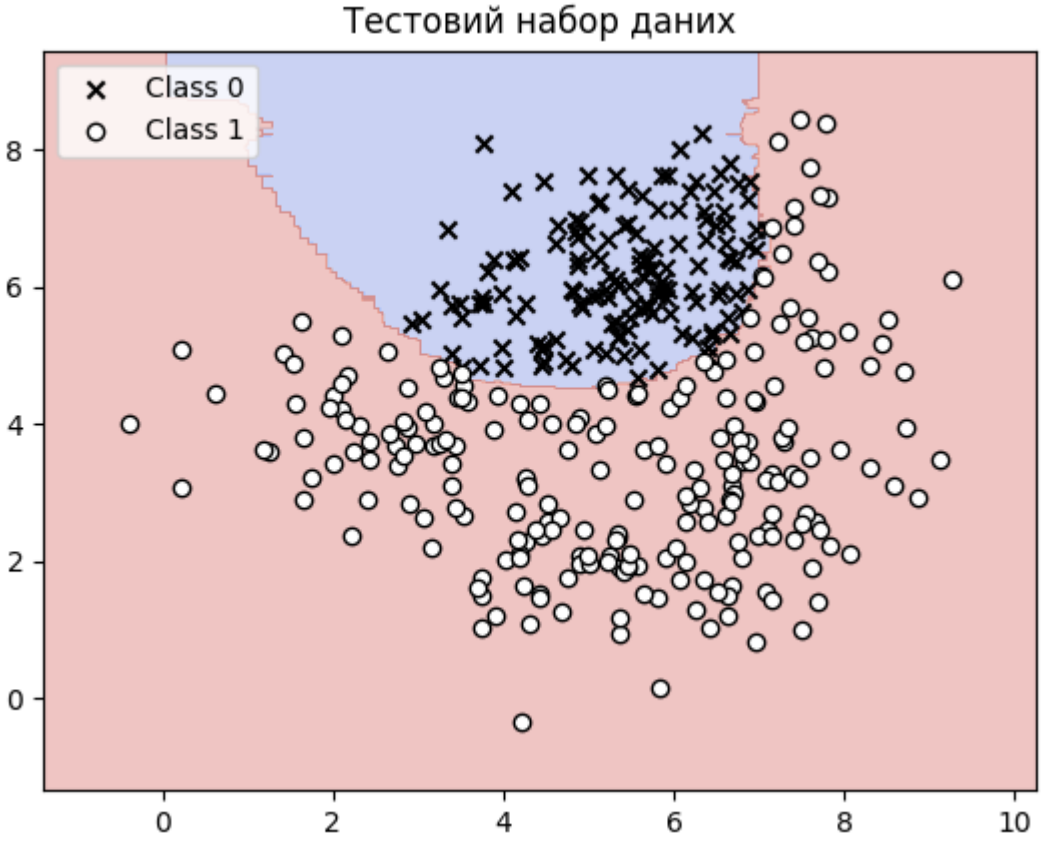
```python
if len(sys.argv) > 1:
    if sys.argv[1] == 'balance':
        params['class_weight'] = 'balanced'
    else:
        raise TypeError("Invalid input argument; should be 'balance'")

# Ініціалізація та тренування класифікатора
classifier = ExtraTreesClassifier(**params)
classifier.fit(X_train, y_train)
visualize_classifier(classifier, X_test, y_test, 'Trained dataset')

# Прогнозування та візуалізація результатів
y_test_pred = classifier.predict(X_test)
visualize_classifier(classifier, X_test, y_test_pred, 'Тестовий набор даних')

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 програми:

```
"D:\Університ\4 - Курс\1 - Семестр\Системи штучного інтелекту\Lab05\Lab05Code\.venv\Scripts\python.exe" "D:\Університ\4 - Курс\1 - Семестр\Системи штучного інтелекту\Lab05\Lab05Code\.venv\Task2.py" balance

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

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

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


Вхідні данні
0
2
4
6
8
0
2
4
6
8
10


Trained dataset
Class 0
Class 1
0
2
4
6
8
0
2
4
6
8
10



Завдання 2.3. Знаходження оптимальних навчальних параметрів за допомогою сіткового пошуку

Лістинг програми:

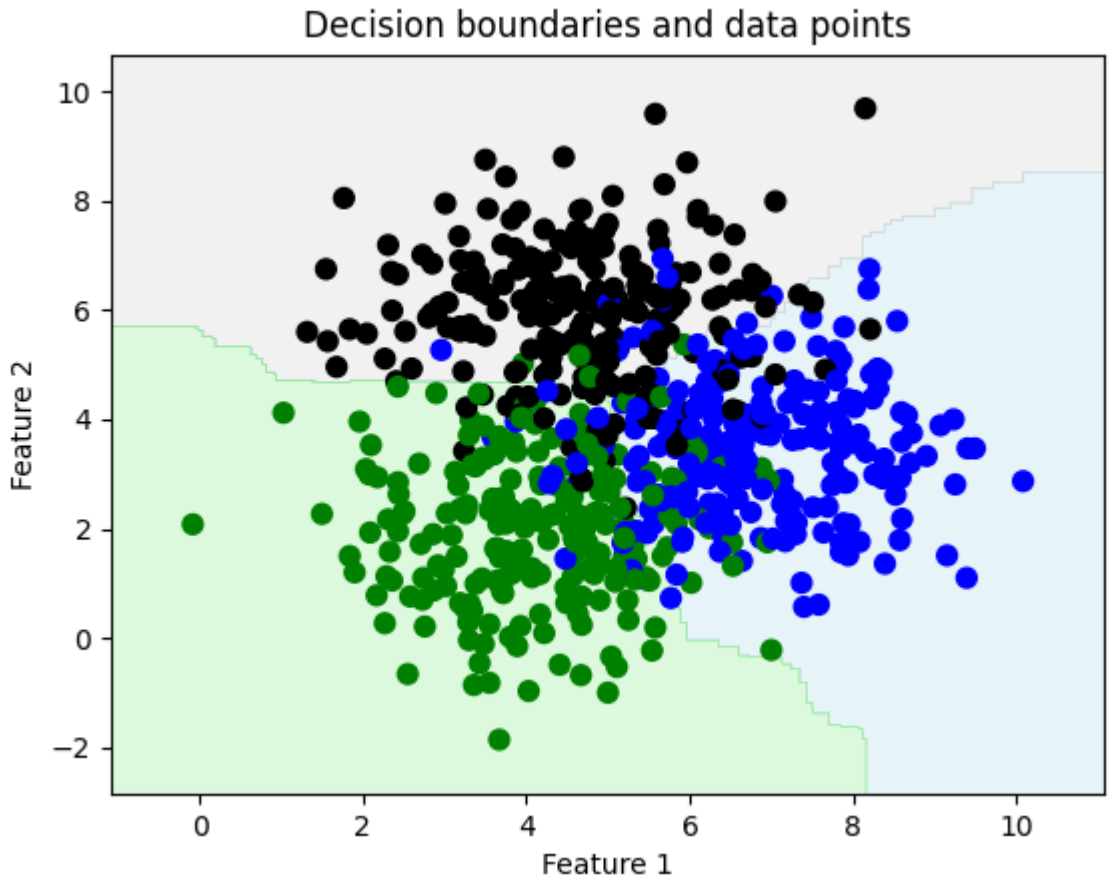
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import ExtraTreesClassifier
from matplotlib.colors import ListedColormap

def visualize_classifier(classifier, X, y):
    # Задаємо мінімум і максимум для графіка
    x_min, x_max = X[:, 0].min() - 1.0, X[:, 0].max() + 1.0
    y_min, y_max = X[:, 1].min() - 1.0, X[:, 1].max() + 1.0

    # Кроки сітки
    step_size = 0.01

    # Створюємо сітку точок
    xx, yy = np.meshgrid(np.arange(x_min, x_max, step_size),
                         np.arange(y_min, y_max, step_size))

    # Передбачаємо класи для кожної точки сітки
    Z = classifier.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    # Створюємо кольорову карту для відображення класифікаційних областей
    cmap_background = ListedColormap(['lightgray', 'lightblue', 'lightgreen'])
    cmap_points = ListedColormap(['black', 'blue', 'green'])
```

```
    # Відображаємо області класифікації
    plt.contourf(xx, yy, Z, cmap=cmap_background, alpha=0.3)

    # Відображаємо точки навчального набору
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap_points, s=50)

    # Настройки графіка
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.title('Decision boundaries and data points')
    plt.show()

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

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

# Налаштування параметрів для GridSearchCV
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']

# Пошук оптимальних параметрів
for metric in metrics:
    print("\n#### Пошук оптимальних параметрів для", metric)
    classifier = GridSearchCV(ExtraTreesClassifier(random_state=0),
parameter_grid, cv=5, scoring=metric)
    classifier.fit(X_train, y_train)

    # Виведення результатів
    print("\nРезультати оцінки параметрів:")
    for params, avg_score in zip(classifier.cv_results_['params'],
classifier.cv_results_['mean_test_score']):
        print(params, '-->', round(avg_score, 3))

    print("\nНайкращі параметри:", classifier.best_params_)

# Оцінка моделі
y_pred = classifier.predict(X_test)
print("\nЗвіт про продуктивність:\n")
print(classification_report(y_test, y_pred))

# Візуалізація класифікатора для тренувальних даних
visualize_classifier(classifier.best_estimator_, X_train, y_train)
```

Виконання програми:

```
Результати оцінки параметрів:
{'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

Найкращі параметри: {'max_depth': 2, 'n_estimators': 100}

#### Пошук оптимальних параметрів для recall_weighted

Результати оцінки параметрів:
{'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

Найкращі параметри: {'max_depth': 2, 'n_estimators': 100}
```

```
Звіт про продуктивність:

              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


Process finished with exit code 0
```



Завдання 2.4. Обчислення відносної важливості ознак

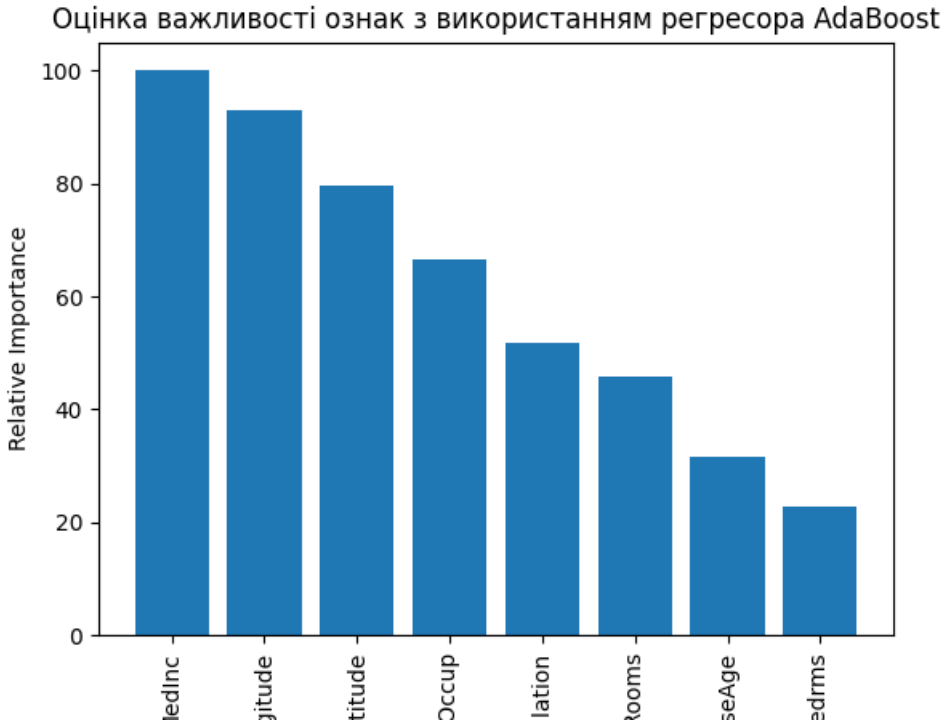
Лістинг програми:

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

# Завантаження набору даних з цінами на нерухомість (Каліфорнія)
housing_data = fetch_california_housing()

# Перемішування даних для підвищення об'єктивності аналізу
X, y = shuffle(housing_data.data, housing_data.target, random_state=7)

# Розділення 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)

# Визначення та навчання регресора AdaBoost
regressor = AdaBoostRegressor(DecisionTreeRegressor(max_depth=4),
n_estimators=400, random_state=7)
regressor.fit(X_train, y_train)

# Оцінка ефективності регресора
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)  # Конвертуємо у масив NumPy

# Нормалізація значень відносної ваги ознак
feature_importances = 100.0 * (feature_importances / max(feature_importances))

# Сортування та перестановка значень для відображення
index_sorted = np.flipud(np.argsort(feature_importances))

# Розміщення міток вздовж 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], rotation=90)
plt.ylabel('Relative Importance')
plt.title('Оцінка важливості ознак з використанням регресора AdaBoost')
plt.show()
```

Виконання програми:

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

Process finished with exit code 0
```



Завдання 2.5. Прогнозування інтенсивності дорожнього руху за допомогою класифікатора на основі гранично випадкових лісів

Лістинг програми:

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

input_file = 'traffic_data.txt'
data = []

# Завантаження та зчитування даних з файлу
with open(input_file, 'r') as f:
    for line in f.readlines():
        items = line.strip().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:
        le = preprocessing.LabelEncoder()
        X_encoded[:, i] = le.fit_transform(data[:, i])
        label_encoder.append(le)

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)

# Визначення та навчання регресора на основі гранично випадкових лісів
params = {'n_estimators': 100, 'max_depth': 4, 'random_state': 0}
regressor = ExtraTreesRegressor(**params)
regressor.fit(X_train, y_train)

# Прогнозування та обчислення показників ефективності
y_pred = regressor.predict(X_test)
print("Mean absolute error:", round(mean_absolute_error(y_test, y_pred), 2))

# Тестування кодування та прогнозування 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([item])[0])
        count += 1
test_datapoint_encoded = np.array(test_datapoint_encoded)
print("Predicted traffic:", int(regressor.predict([test_datapoint_encoded])[0]))
```

Виконання програми:

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

Process finished with exit code 0
```

Посилання на ГітХаб: https://github.com/Kn211mna/AI-YT

Висновок: в ході виконання лабораторної роботи опрацював спеціалізовані бібліотеки та мову програмування Python дослідити методи ансамблів у машинному навчанні.