Лабораторна робота №5

ДОСЛІДЖЕННЯ МЕТОДІВ АНСАМБЛЕВОГО НАВЧАННЯ

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

Github - https://github.com/Oleksi89/ai-systems

Хід роботи

Завдання 2.1. 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.metrics import classification_report
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier
from sklearn.metrics import classification_report

from utilities import visualize_classifier


# Argument parser
def build_arg_parser():
    parser = argparse.ArgumentParser(description='Classify data using \
            Ensemble Learning techniques')
    parser.add_argument('--classifier-type', dest='classifier_type',
                        required=True, choices=['rf', 'erf'], help="Type of classifier \
                to use; can be either 'rf' or 'erf'")
    return parser


if __name__ == '__main__':
    # Parse the input arguments
    args = build_arg_parser().parse_args()
    classifier_type = args.classifier_type

    # Load input data
    input_file = '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()
```

Результати:

python random_forests.py --classifier-type rf

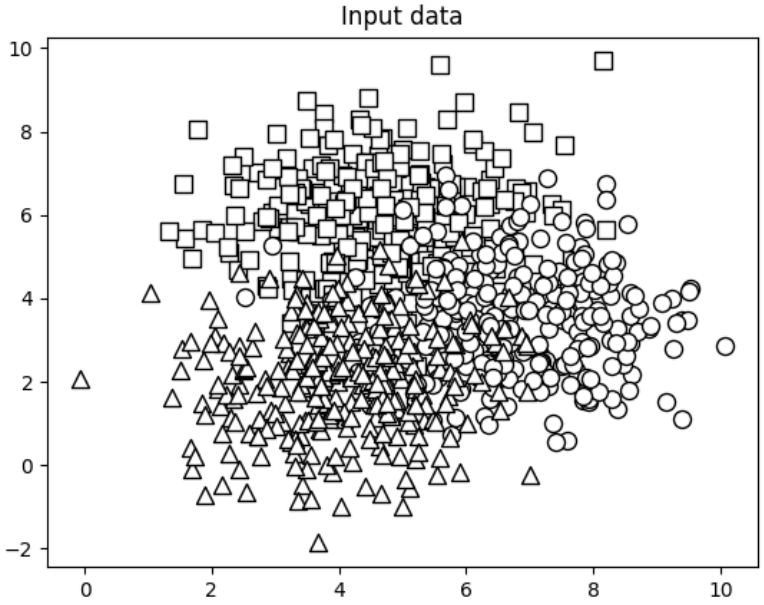

Рис.1 Візуалізація вхіжних даних

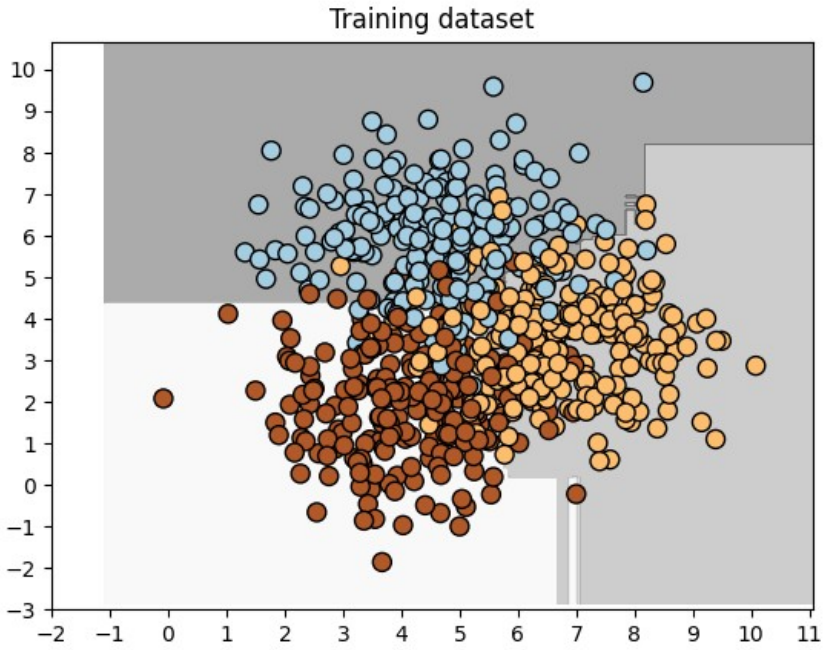

Рис.2 Візуалізація класифікатора на основі випадкового лісу для навчальних даних

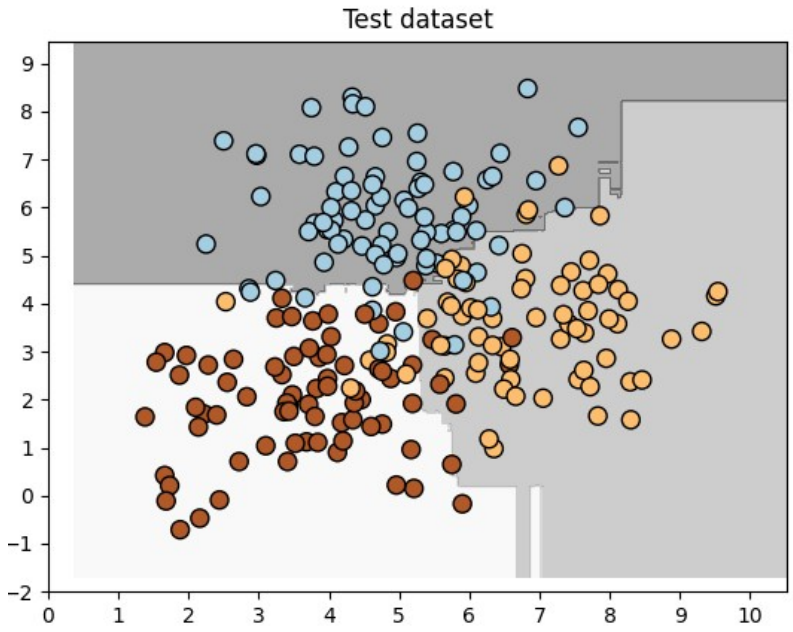

Рис.3 Візуалізація класифікатора на основі випадкового лісу для тестових даних

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

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

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

Рис.4 Вивід у консоль

```
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
```

Рис.5 Вивід у консоль оцінок впевненості

python random_forests.py --classifier-type erf

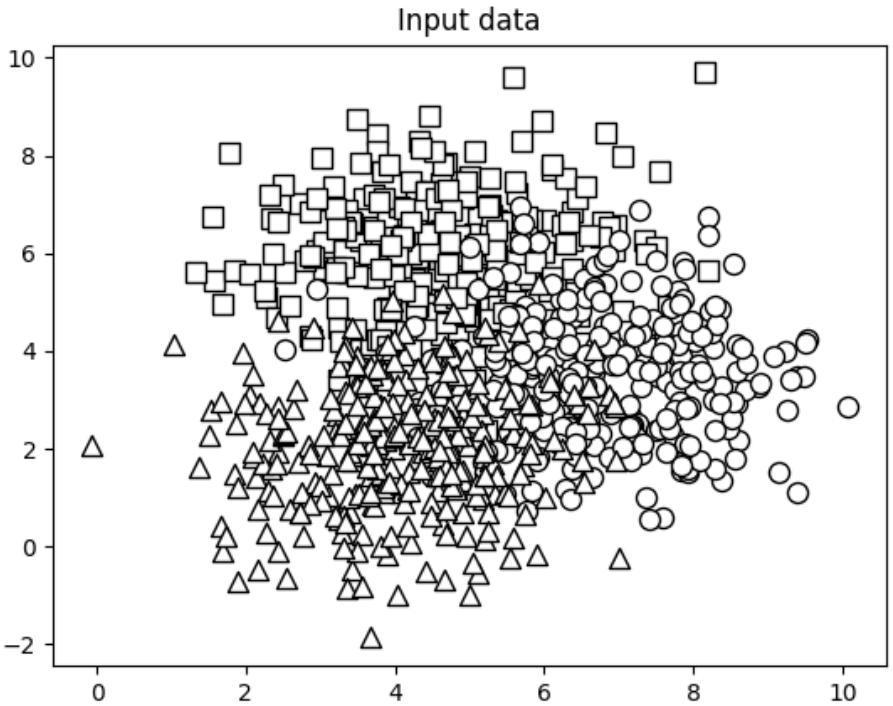

Рис.6 Візуалізація вхідних даних

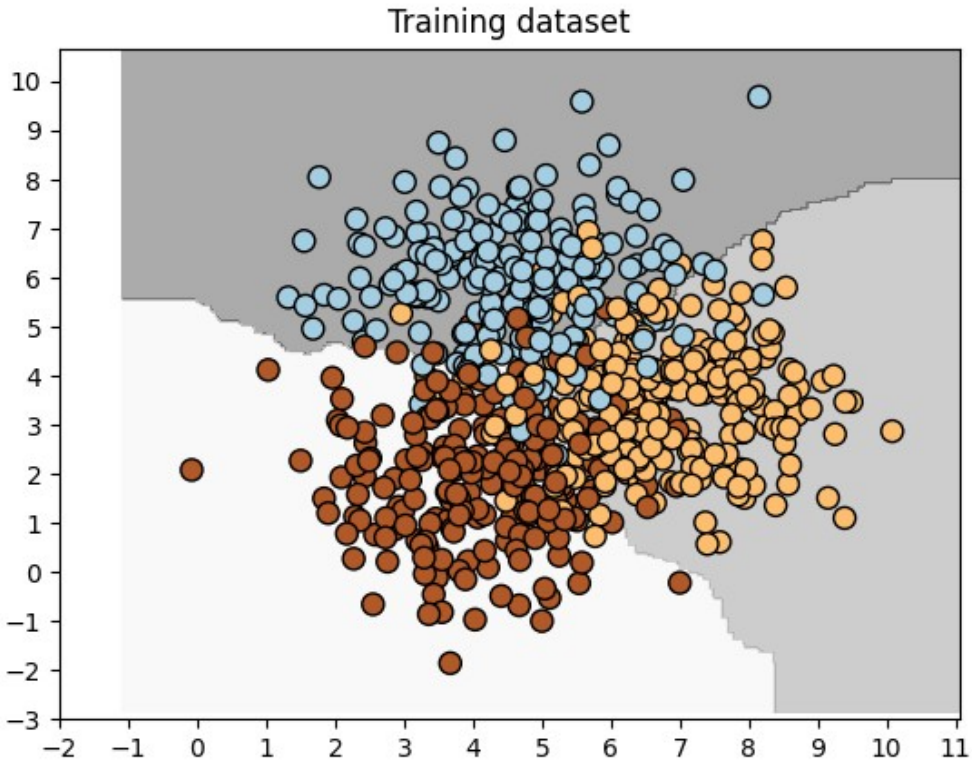

Рис.7 Візуалізація класифікатора на основі випадкового лісу для навчальних даних

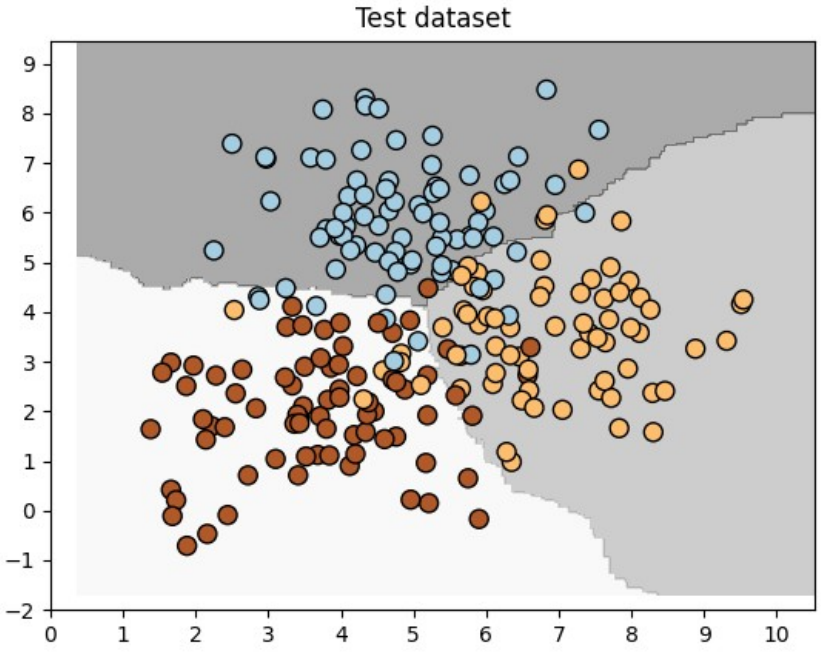

Рис.8 Візуалізація класифікатора на основі випадкового лісу для тестових даних

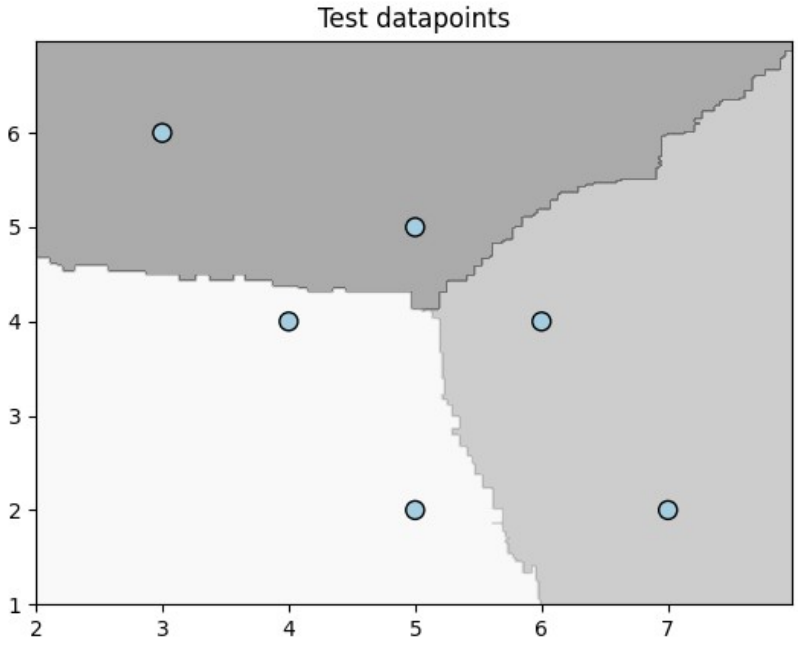

Рис.9 Візуалізація тестових точок даних

```
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

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

Рис.10 Вивід у консоль

```
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
```

Рис.11 Вивід у консоль оцінок впевненості

Порівняння алгоритмів Random Forest (rf) та Extremely Random Forests (erf) показало ідентичну ефективність на тестовій вибірці з точністю 0.87. Незначне відставання erf на тренувальних даних (0.85 проти 0.87 у rf), проте це не вплинуло на фінальну узагальнювальну здатність. Обидві моделі демонструють збалансовані показники F1-score для всіх класів та однаково коректно класифікують контрольні точки, що підтверджує надійність обох ансамблевих підходів для заданого датасету.

Завдання 2.2. 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

matplotlib.use('TkAgg')
import matplotlib.pyplot as plt
from sklearn.ensemble import ExtraTreesClassifier
```

```
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from utilities import visualize_classifier

# Вхідні дані
input_file = 'data_imbalance.txt'
data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1]

# Поділ вхідних даних на два класи на підставі міток
class_0 = np.array(X[y == 0])
class_1 = np.array(X[y == 1])

# Візуалізація вхідних даних
plt.figure()
plt.scatter(class_0[:, 0], class_0[:, 1], s=75, facecolors='black', edgecolors='black', linewidth=1, marker='x')
plt.scatter(class_1[:, 0], class_1[:, 1], s=75, facecolors='white', edgecolors='black', linewidth=1, marker='o')
plt.title('Вхідні дані')

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

# Класифікатор на основі гранично випадкових лісів
params = {'n_estimators': 100, 'max_depth': 4, 'random_state': 0}
if len(sys.argv) > 1:
    if sys.argv[1] == 'balance':
        params = {'n_estimators': 100, 'max_depth': 4, 'random_state': 0, 'class_weight': 'balanced'}
    else:
        raise TypeError("Invalid input argument; should be 'balance'")

# Візуалізація для тестового набору даних
classifier = ExtraTreesClassifier(**params)
classifier.fit(X_train, y_train)
visualize_classifier(classifier, X_train, y_train, 'Training dataset')

# Передбачення та візуалізація результату для тестового набору даних
y_test_pred = classifier.predict(X_test)
visualize_classifier(classifier, X_test, y_test, 'Test dataset')

# Обчислення показників ефективності класифікатора
class_names = ['Class-0', 'Class-1']
print("\n" + "#" * 40)
print("\nClassifier performance on training dataset\n")
print(classification_report(y_train, classifier.predict(X_train), target_names=class_names))
print("#" * 40 + "\n")

print("#" * 40)
print("\nClassifier performance on test dataset\n")
print(classification_report(y_test, y_test_pred, target_names=class_names))
print("#" * 40 + "\n")

plt.show()
```

Результати:

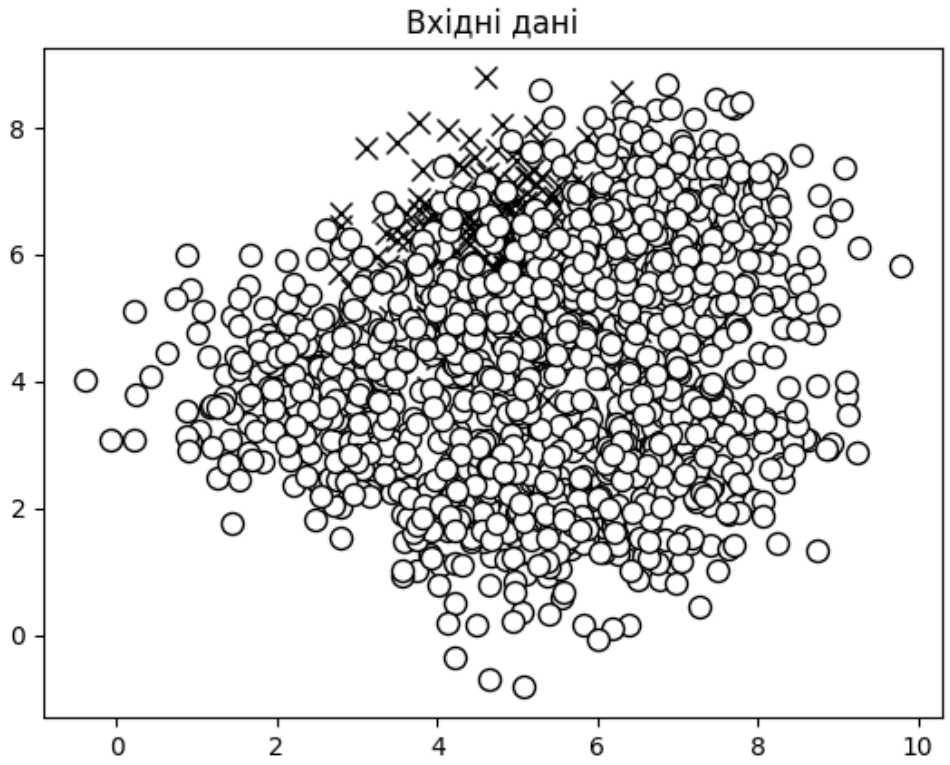

Рис.12 Візуалізація вхіжних даних

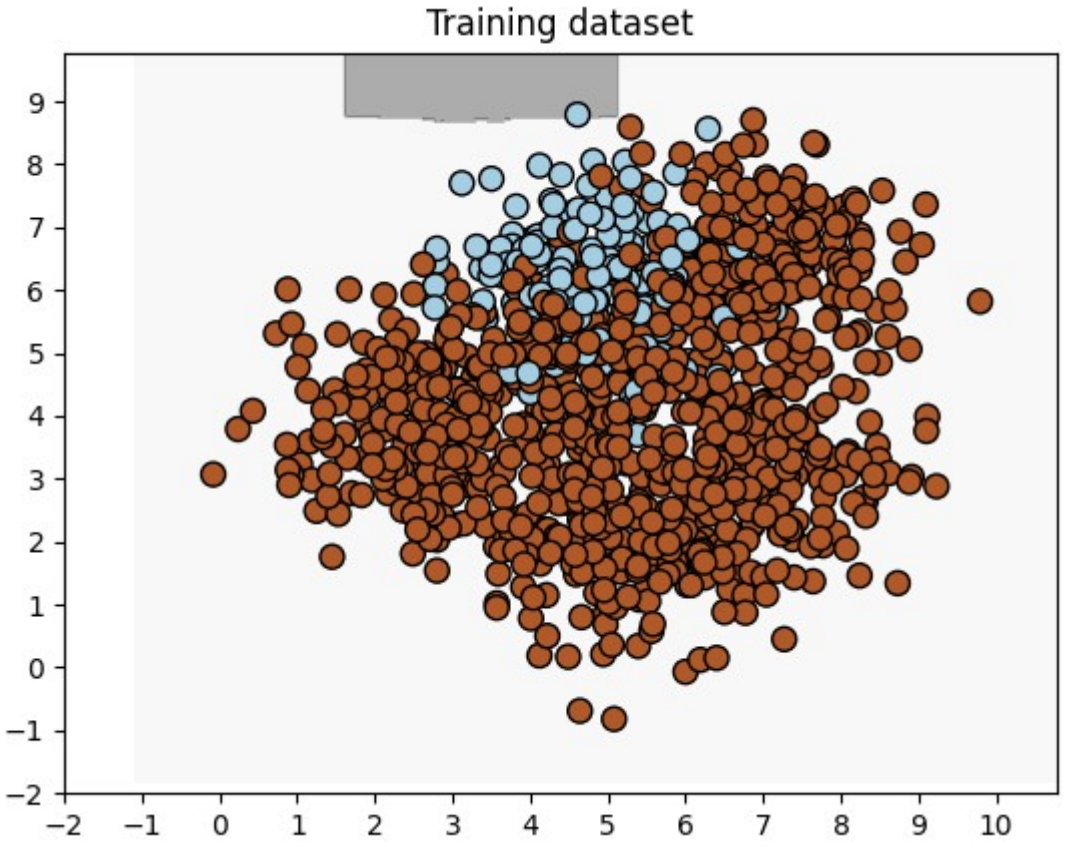

Рис.13 Візуалізація класифікатора навчальних даних

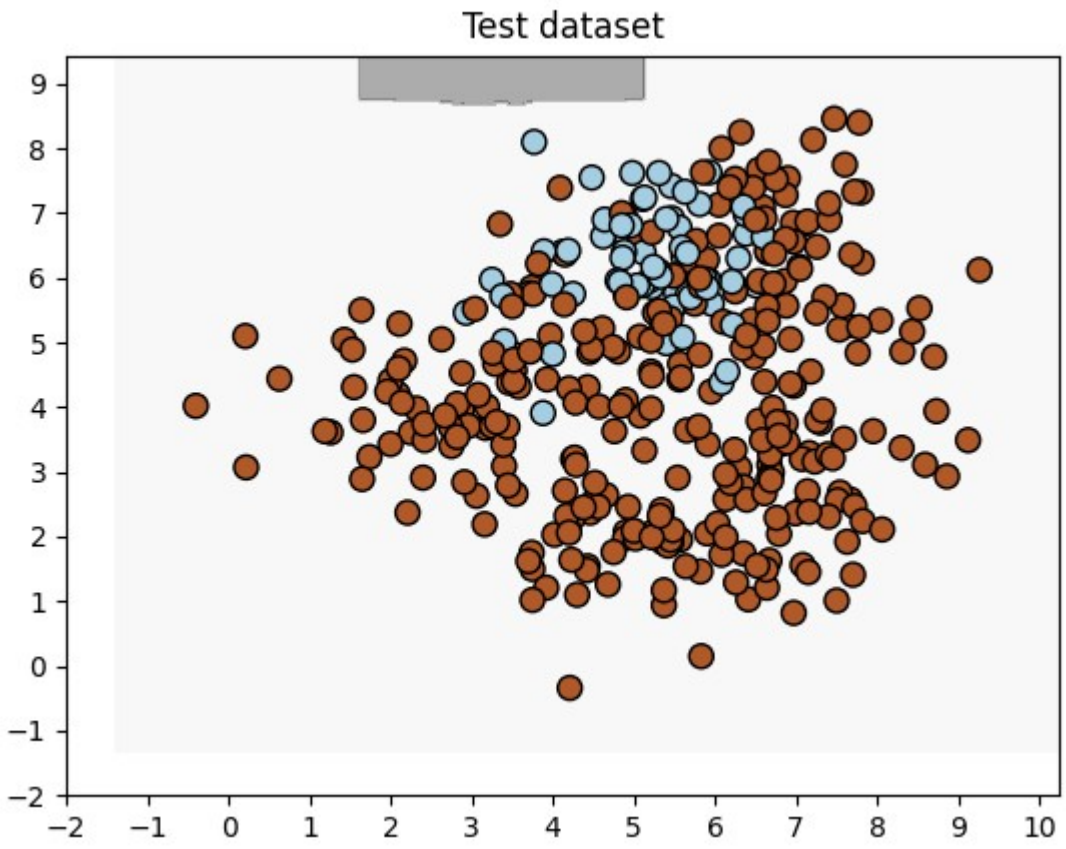

Рис.14 Візуалізація класифікатора тестових даних

```
            precision    recall  f1-score   support

    Class-0       1.00      0.01      0.01       181
    Class-1       0.84      1.00      0.91       944

   accuracy                           0.84      1125
  macro avg       0.92      0.50      0.46      1125
weighted avg      0.87      0.84      0.77      1125

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

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

Classifier performance on test dataset

            precision    recall  f1-score   support

    Class-0       0.00      0.00      0.00        69
    Class-1       0.82      1.00      0.90       306

   accuracy                           0.82       375
  macro avg       0.41      0.50      0.45       375
weighted avg      0.67      0.82      0.73       375

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

Рис.15 Вивід у консоль(без врахування дисбалансу)

Врахування дисбалансу класів:

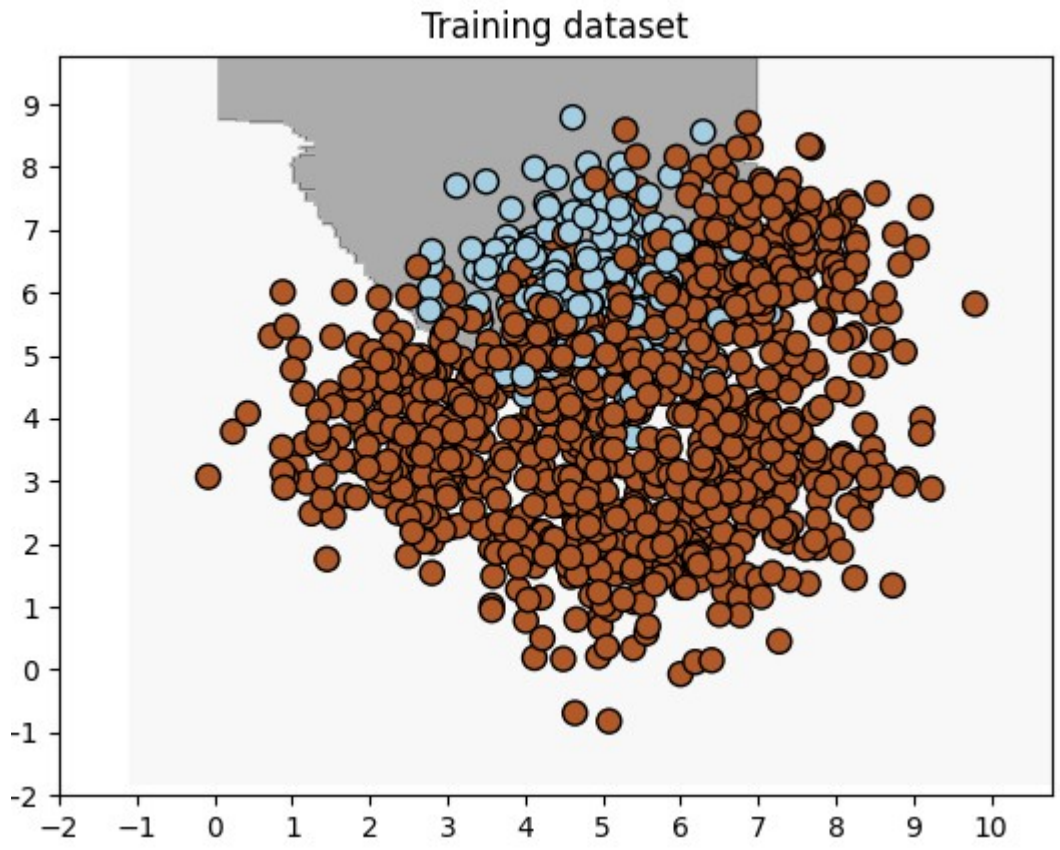

Рис.16 Візуалізація класифікатора навчальних даних

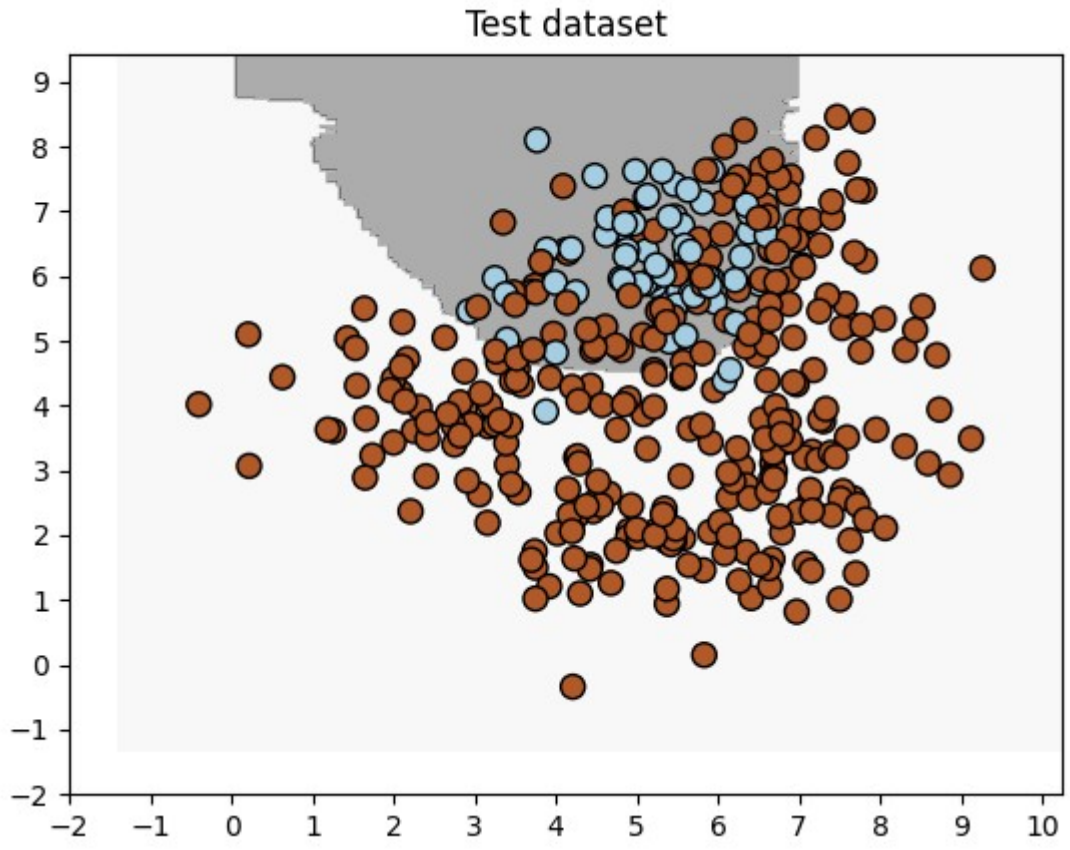

Рис.17 Візуалізація класифікатора тестових даних

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

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

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

Рис.18 Вивід у консоль(з врахуванням дисбалансу)

Дослідження показало, що на незбалансованих даних звичайний класифікатор працює некоректно: він ігнорує малочисельний клас (Class-0), просто відносячи всі об“єкти до більш популярного (Class-1). Висока точність (Accuracy 0.82) у цьому випадку оманлива, оскільки Recall для першого класу дорівнює нулю.

Використання параметра class_weight=balanced виправило ситуацію шляхом автоматичного підвищення значущості рідкісного класу. Це призвело до незначного зниження загальної точності (до 0.78), проте дозволило моделі реально розпізнавати об’єкти малочисельного класу, піднявши його Recall з 0 до 0.94. Це доводить, що при сильному дисбалансі даних не можна орієнтуватися

лише на загальну точність, а необхідно балансувати ваги, щоб модель не ігнорувала класи з меншою кількістю прикладів.

Завдання 2.3. 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
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.metrics import classification_report

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

# Розбиття даних на три класи на підставі міток
class_0 = np.array(X[y == 0])
class_1 = np.array(X[y == 1])
class_2 = np.array(X[y == 2])

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

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

# Визначимо метричні характеристики
metrics = ['precision_weighted', 'recall_weighted']

# Сітковий пошук
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)

    # Виведемо оцінку для кожної комбінації параметрів
    print("\nGrid scores for the parameter grid:")
    for i in range(len(classifier.cv_results_['params'])):
        print(
            classifier.cv_results_['params'][i],
            '-->',
            round(classifier.cv_results_['mean_test_score'][i], 3)
```

```
    )

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

Результати:

```
##### Searching optimal parameters for precision_weighted

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

Performance report:

              precision    recall  f1-score   support

         0.0       0.94      0.81      0.87        79
         1.0       0.81      0.86      0.83        70
         2.0       0.83      0.91      0.87        76

    accuracy                           0.86       225
   macro avg       0.86      0.86      0.86       225
weighted avg       0.86      0.86      0.86       225


##### Searching optimal parameters for recall_weighted

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

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
```

Рис.19

Було використано автоматичний перебір, щоб знайти найкращі налаштування для моделі. Ми шукали такі значення, які б давали найкращий результат окремо

для точності (Precision) і окремо для повноти (Recall). Зазвичай ці налаштування відрізняються, але тут вийшло так, що одні й ті самі параметри (глибина дерева 2) працюють найкраще в обох випадках. Це дозволило отримати стабільний результат у 86% правильних відповідей без необхідності обирати, що важливіше - точність чи повнота.

Завдання 2.4. Обчислення відносної важливості ознак з використанням регресора AdaBoost.

Лістинг програми:

```
import numpy as np
import matplotlib

matplotlib.use('TkAgg')
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_california_housing
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.metrics import mean_squared_error, explained_variance_score
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split

# Завантаження даних із цінами на нерухомість
# datasets.load_boston() вилучили з бібліотеки
housing_data = fetch_california_housing()

# Перемішування даних
X, y = shuffle(housing_data.data, housing_data.target, random_state=7)

# Розбиття даних на навчальний та тестовий набори
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=7)

# Модель на основі регресора AdaBoost
regressor = AdaBoostRegressor(DecisionTreeRegressor(max_depth=4),
                              n_estimators=400, random_state=7)
regressor.fit(X_train, y_train)

# Обчислення показників ефективності регресора AdaBoost
y_pred = regressor.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
evs = explained_variance_score(y_test, y_pred)

print("\nADABOOST REGRESSOR")
print("Mean squared errpr =", round(mse, 2))
print("Explained variance score =", round(evs, 2))

# Вилучення важливості ознак
feature_importances = regressor.feature_importances_
```

```
feature_names = np.array(housing_data.feature_names)

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

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

# Розміщення міток уздовж осі X
pos = np.arange(index_sorted.shape[0]) + 0.5

# Побудова стовпчастої діаграми
plt.figure()
plt.bar(pos, feature_importances[index_sorted], align='center')
plt.xticks(pos, feature_names[index_sorted])
plt.ylabel('Relative Importance')
plt.title('Оцінка важливості ознак з використанням регресора AdaBoost')
plt.show()
```

Результати:

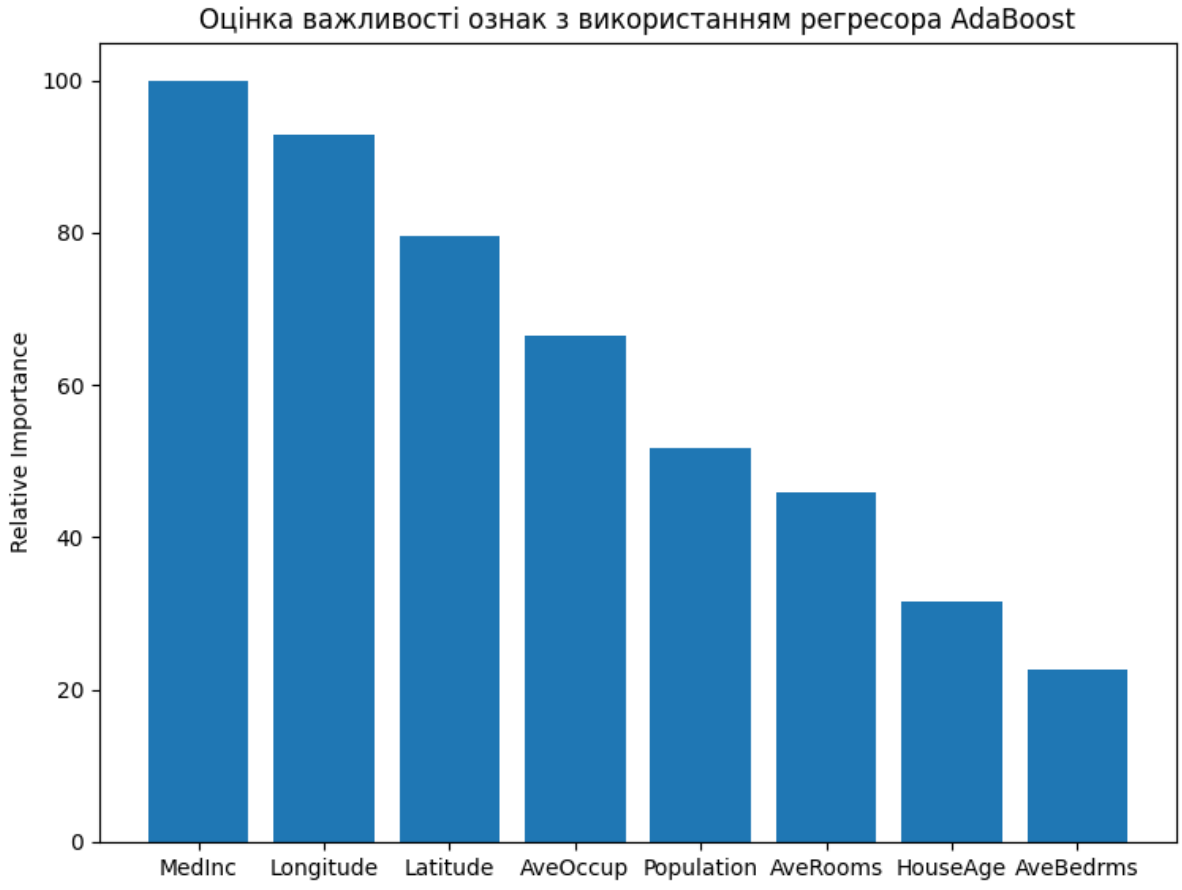

Рис.20 Графік

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

Рис.21 Вивід у консоль

вДіаграма чітко показує, що найбільший вплив на результат має MedInc (середній дохід). Також важливими є координати (Longitude та Latitude), тобто розташування. Вплив інших параметрів набагато менший, а останні два стовпчики (HouseAge та AveBedrms) мають найнижчі показники важливості.

Отже, головними факторами формування ціни є рівень доходу населення та місцезнаходження будинку. Параметри AveBedrms (спальні) та HouseAge (вік) виявилися найменш корисними для прогнозу. Ними можна знехтувати, щоб спростити модель, оскільки вони майже не впливають на кінцевий результат.

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

Проведіть прогнозування інтенсивності дорожнього руху за допомогою класифікатора на основі гранично випадкових лісів. Оцініть її якість.

Лістинг програми:

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

# Вхідні дані
input_file = 'traffic_data.txt'
data = []
with open(input_file, 'r') as f:
    for line in f.readlines():
        items = line[:-1].split(',')
        data.append(items)

data = np.array(data)

# Перетворення рядкових даних на числові
label_encoder = []
X_encoded = np.empty(data.shape)
for i, item in enumerate(data[0]):
    if item.isdigit():
        X_encoded[:, i] = data[:, i]
```

```
    else:
        label_encoder.append(preprocessing.LabelEncoder())
        X_encoded[:, i] = label_encoder[-1].fit_transform(data[:, i])

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

# Розіб'ємо дані на навчальний та тестовий набори
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=5)

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

#  Обчислення характеристик ефективності регресора на тестових даних
y_pred = regressor.predict(X_test)
print("Mean absolute error: ", round(mean_absolute_error(y_test, y_pred), 2))

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

# Прогнозування результату для тестової точки даних
print("Predicted trafic: ", int(regressor.predict([test_datapoint_encoded])[0]))
```

Результати:

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

Process finished with exit code 0
```

Рис.22 Вивід у консоль

Висновок: у цій лабораторній роботі я дослідив методи ансамблів у машинному навчанні використовуючи спеціалізовані бібліотеки та мову програмування Python.