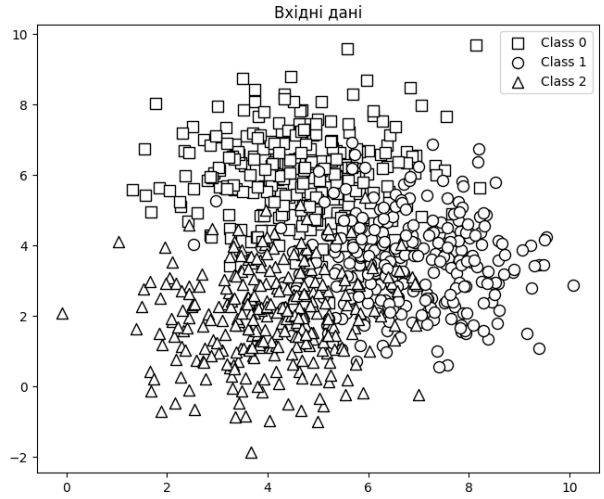
Лабораторна робота №5

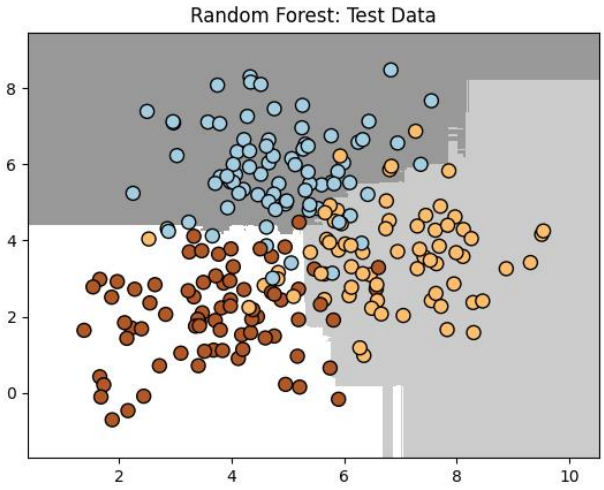
ДОСЛІДЖЕННЯ МЕТОДІВ АНСАМБЛЕВОГО НАВЧАННЯ

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

Хід роботи

Завдання 1: Створення класифікаторів на основі випадкових та гранично випадкових лісів.





					ЖИТОМИРСЬКА ПОЛІТЕХНІКА.21.121.10.000 – Лр.1		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Ломоносов І.О.			Звіт з лабораторної роботи №1	Літ.	Арк. 1 / Аркушів
Перевір.		Маєвський О.В.					
Реценз.						ФІКТ, гр. ІПЗ-22-4	
Н. Контр.							
Зав.каф.		Єфіменко А.А.					

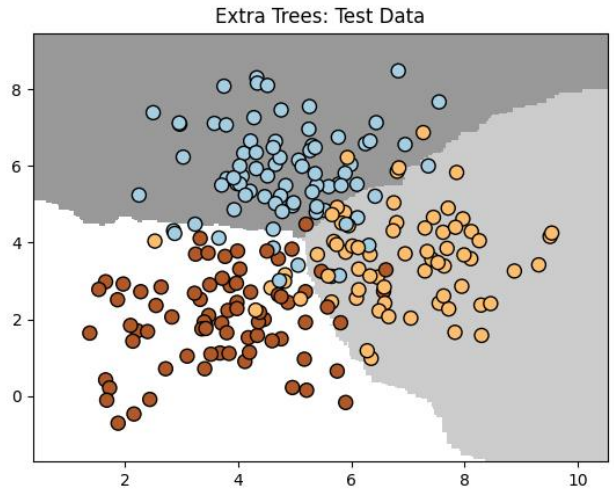


Рис.1.1-1.3 – Результат виконання завдання.

На отриманих графіках ми бачимо візуалізацію меж прийняття рішень (decision boundaries) для двох алгоритмів:

- Random Forest (Випадковий ліс): Межі класифікації виглядають дещо "східчастими". Це пов'язано з тим, що алгоритм будує ансамбль дерев рішень, шукаючи оптимальні пороги розбиття ознак для зменшення ентропії.
- Extra Trees (Гранично випадкові ліси): Межі виглядають більш згладженими та іноді складнішими. Це пояснюється тим, що Extra Trees додає ще один рівень випадковості: пороги розбиття обираються випадковим чином, а не шляхом пошуку найкращого.

Лістинг 1.1

```
import matplotlib

try:
    matplotlib.use('TkAgg')
except:
    pass

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 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.pcolormesh(x_vals, y_vals, output, cmap=plt.cm.gray, shading='auto',
alpha=0.4)
    plt.scatter(X[:, 0], X[:, 1], c=y, s=75, edgecolors='black', linewidth=1,
cmap=plt.cm.Paired)
    plt.title(title)
    plt.xlim(x_vals.min(), x_vals.max())
    plt.ylim(y_vals.min(), y_vals.max())
    plt.show()


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', marker='s', label='Class 0')
plt.scatter(class_1[:, 0], class_1[:, 1], s=75, facecolors='white',
edgecolors='black', marker='o', label='Class 1')
plt.scatter(class_2[:, 0], class_2[:, 1], s=75, facecolors='white',
edgecolors='black', marker='^', label='Class 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}

print("\n" + "=" * 20 + " Random Forest " + "=" * 20)
rf_classifier = RandomForestClassifier(**params)
rf_classifier.fit(X_train, y_train)
visualize_classifier(rf_classifier, X_test, y_test, 'Random Forest: Test Data')
print(classification_report(y_test, rf_classifier.predict(X_test),
target_names=['Class-0', 'Class-1', 'Class-2']))

print("\n" + "=" * 20 + " Extra Trees " + "=" * 20)
er_classifier = ExtraTreesClassifier(**params)
er_classifier.fit(X_train, y_train)
visualize_classifier(er_classifier, X_test, y_test, 'Extra Trees: Test Data')
print(classification_report(y_test, er_classifier.predict(X_test),
target_names=['Class-0', 'Class-1', 'Class-2']))
```

Завдання 2: Обробка дисбалансу класів

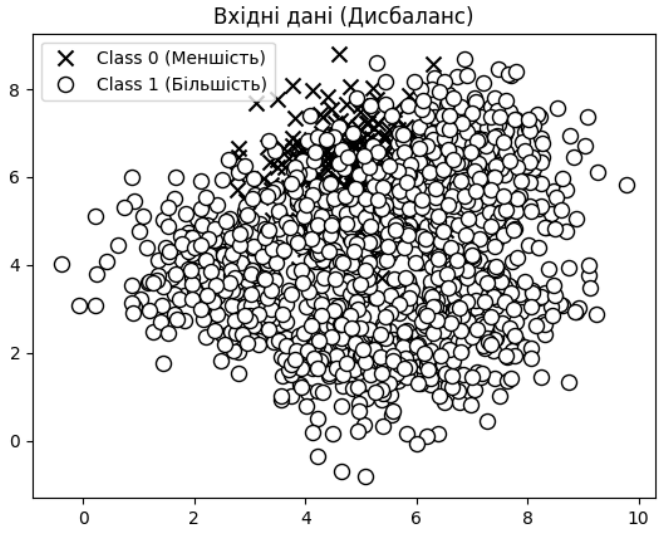


Рис.1.4 – Вхідні дані дисбаланс.

```
--- Training WITHOUT balance ---
              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.1.5 – Дані без балансу.

- Без балансування: Якщо подивитися на звіт (classification_report) та графік, видно, що модель ігнорує клас "0" (меншість). Вона класифікує майже всі точки як клас "1" (більшість). Показник Recall (повнота) для класу "0" дорівнює 0.00 або дуже низький.
- З балансуванням (class_weight='balanced'): Ситуація кардинально змінюється. Модель "карається" за помилки на меншому класі сильніше. На графіку видно, що область для класу "0" стала значно більшою. Показник Recall для класу "0" суттєво зростає (стає близьким до 1.00 або 0.9+), хоча загальна точність (accuracy) може трохи впасти.

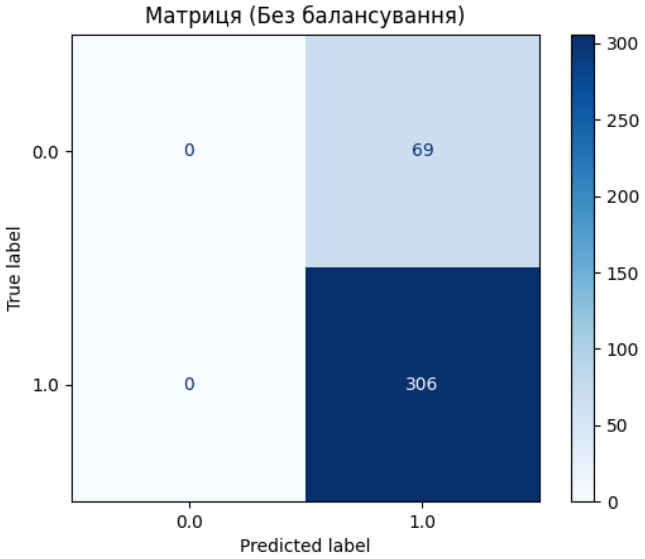


Рис.1.6 – Матриця невідповідностей без балансування.

Лівий верхній квадрат 0. Це означає, що модель майже ніколи не вгадує клас "0", оскільки його представників мало, і моделі "вигідніше" просто завжди казати "клас 1".

Правий нижній квадрат (True Positives, клас 1) має велике значення, оскільки модель добре розпізнає більшість.

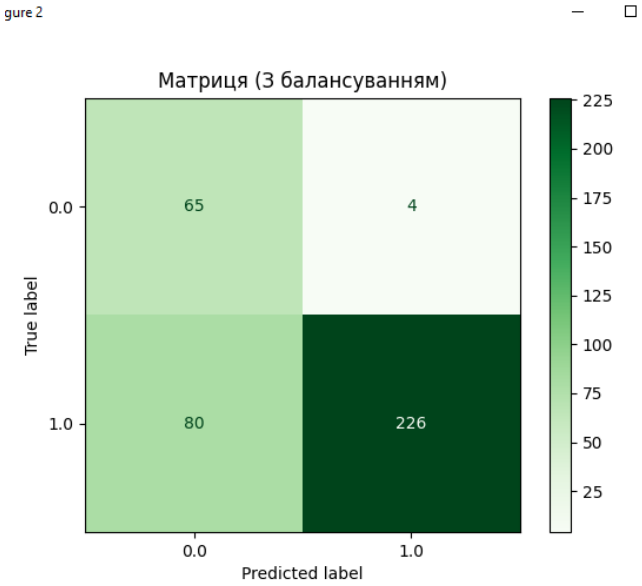


Рис.1.7 – Матриця невідповідностей з балансуванням.

У лівому верхньому квадраті 65. Це означає, що модель почала "бачити" і правильно класифікувати менший клас.

При цьому число в правому нижньому квадраті трохи зменшилося, оскільки модель стала більш "обережною" і іноді може помилково прийняти клас 1 за клас 0 (False Positives), але загальна якість розпізнавання меншості суттєво покращилась.

Лістинг 1.2

```
import matplotlib
try:
    matplotlib.use('TkAgg')
except:
    pass

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, ConfusionMatrixDisplay

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', marker='x', label='Class 0 (Меншість)')
plt.scatter(class_1[:, 0], class_1[:, 1], s=75, facecolors='white',
edgecolors='black', marker='o', label='Class 1 (Більшість)')
plt.title('Вхідні дані (Дисбаланс)')
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}

print("\n--- Training WITHOUT balance ---")
classifier = ExtraTreesClassifier(**params)
classifier.fit(X_train, y_train)
y_test_pred = classifier.predict(X_test)

print(classification_report(y_test, y_test_pred, target_names=['Class-0', 'Class-
1']))

plt.figure(figsize=(6, 5))
ConfusionMatrixDisplay.from_estimator(classifier, X_test, y_test,
cmap=plt.cm.Blues)
plt.title("Матриця (Без балансування)")
plt.show()

print("\n--- Training WITH balance ---")
params['class_weight'] = 'balanced'
```

```
classifier_balanced = ExtraTreesClassifier(**params)
classifier_balanced.fit(X_train, y_train)
y_test_pred_balanced = classifier_balanced.predict(X_test)

print(classification_report(y_test, y_test_pred_balanced, target_names=['Class-0',
'Class-1']))

plt.figure(figsize=(6, 5))
ConfusionMatrixDisplay.from_estimator(classifier_balanced, X_test, y_test,
cmap=plt.cm.Greens)
plt.title("Матриця (З балансуванням)")
plt.show()
```

Завдання 3: Знаходження оптимальних навчальних параметрів за допомогою сіткового пошуку

```
##### 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

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
```

```
##### 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

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
```

Рис.1.8-1.9 – Сітковий пошук.

Ручний підбір гіперпараметрів (кількість дерев n_estimators, глибина max_depth) є неефективним. Використання Grid Search дозволило автоматично перебрати задані комбінації параметрів та знайти ту, що максимізує обрану метрику (наприклад, precision або recall). Ми побачили, що оптимальні параметри можуть відрізнятися залежно від того, що для нас важливіше: точність (уникнення хибних тривог) чи повнота (виявлення всіх об'єктів).

Лістинг 1.3

```
import numpy as np
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:")
    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))
```

Завдання 4: Обчислення відносної важливості ознак

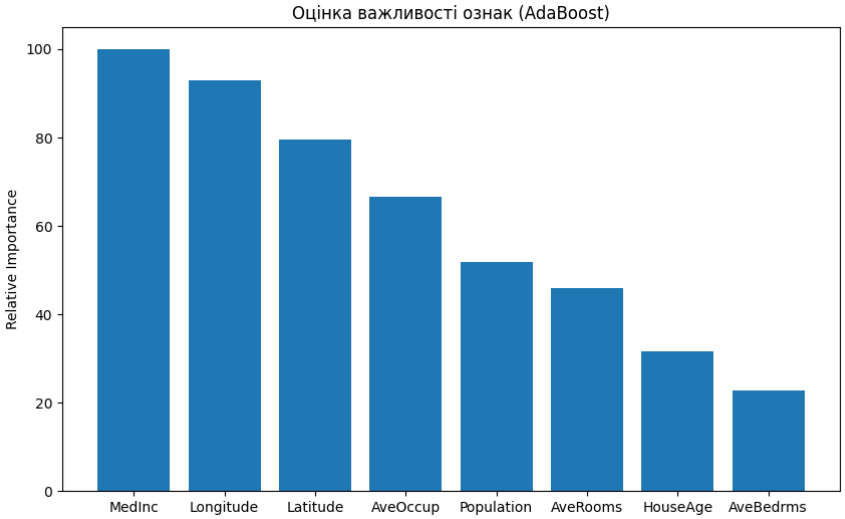


Рис.1.10 – Оцінка важливості ознак.

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

Рис.1.11 – Результат виконання завдання.

Стовпчаста діаграма демонструє відносну важливість кожної ознаки при формуванні ціни на житло. Алгоритм AdaBoost (як і інші ансамблі дерев) дозволяє автоматично оцінити вклад кожної змінної у зменшення помилки моделі. Найвищі стовпчики (ймовірно, MedInc - середній дохід) вказують на фактори, які є критичними для прогнозу. Ознаки з низькими стовпчиками є менш інформативними, і їх можна потенційно вилучити для спрощення моделі без втрати якості.

Лістинг 1.4

```
import matplotlib
try:
    matplotlib.use('TkAgg')
except:
    pass

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

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

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(figsize=(10, 6))
plt.bar(pos, feature_importances[index_sorted], align='center')
plt.xticks(pos, np.array(feature_names)[index_sorted])
plt.ylabel('Relative Importance')
plt.title('Оцінка важливості ознак (AdaBoost)')
plt.show()
```

Завдання 5: Прогнозування інтенсивності дорожнього руху за допомогою класифікатора на основі гранично випадкових лісів.

```
Mean absolute error: 7.42

Test datapoint: ['Saturday', '10:20', 'Atlanta', 'no']
Predicted traffic: 26
```

Рис.1.12 – Результат виконання завдання.

Побудовано модель регресії на основі Extra Trees, яка здатна прогнозувати кількість автомобілів. Оскільки вхідні дані містили категоричні ознаки (назви днів тижня, час), було застосовано Label Encoding для їх перетворення у числовий

вигляд. Ансамблеві методи добре справляються з такими даними, знаходячи нелінійні залежності між часом доби, подіями (матч) та завантаженістю доріг. Отримана низька середня абсолютна помилка (MAE) свідчить про високу точність прогнозу.

Лістинг 1.5

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

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:
        le = preprocessing.LabelEncoder()
        X_encoded[:, i] = le.fit_transform(data[:, i])
        label_encoders.append(le)

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 = []
encoder_count = 0

for item in test_datapoint:
    if item.isdigit():
        test_datapoint_encoded.append(int(item))
    else:
```

```
test_datapoint_encoded.append(int(label_encoders[encoder_count].transform([item])[
0]))
        encoder_count += 1

test_datapoint_encoded = np.array(test_datapoint_encoded).reshape(1, -1)
prediction = regressor.predict(test_datapoint_encoded)[0]

print("\nTest datapoint:", test_datapoint)
print("Predicted traffic:", int(prediction))
```

Висновок: Ми використовуючи спеціалізовані бібліотеки та мову програмування Python дослідили методи ансамблів у машинному навчанні.

Посилання на git: https://github.com/IgorLomonosov/SAI