ЛАБОРАТОРНА РОБОТА № 5

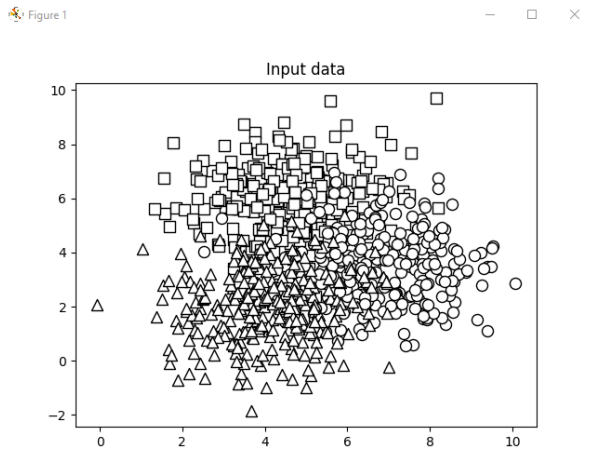
ДОСЛІДЖЕННЯ МЕТОДІВ АНСАМБЛЕВОГО НАВЧАННЯ

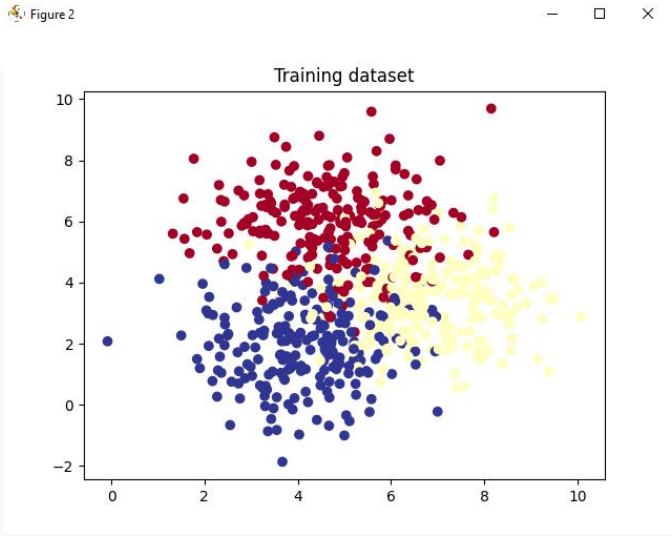
Мета: використовуючи спеціалізовані бібліотеки та мову програмування Python дослідити методи регресії даних у машинному навчанні.

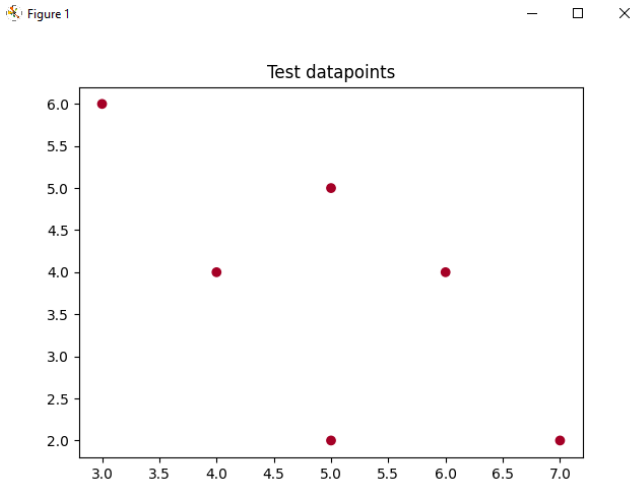
Git: https://github.com/flekXD/SAI

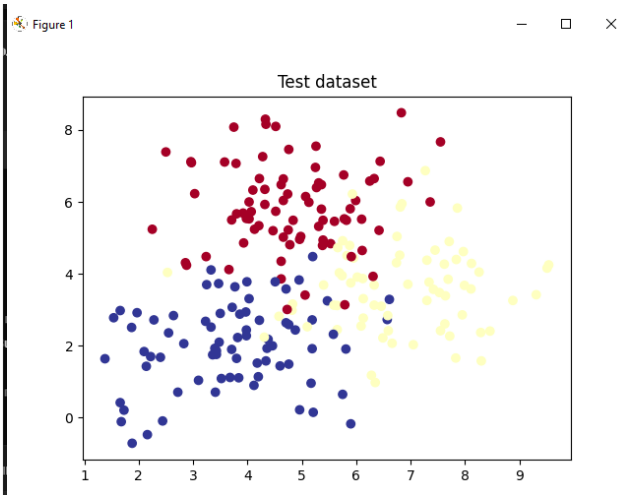
Завдання 1. Створення класифікаторів на основі випадкових та гранично випадкових лісів

Використовувати файл вхідних даних: data_random_forests.txt, побудувати класифікатори на основі випадкових та гранично випадкових лісів










```
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()
```

У завданні були побудовані класифікатори на основі випадкових лісів (Random Forest) та гранично випадкових лісів (Extra Trees) для класифікації даних з файлу

Загалом, Random Forest і Extra Trees добре справляються з класифікацією даних, з деякими перевагами у Extra Trees для складніших завдань.

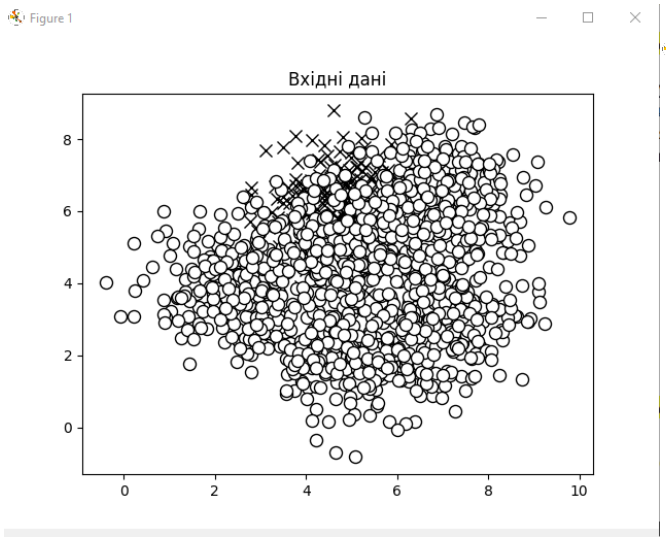
Завдання 2. Обробка дисбалансу класів

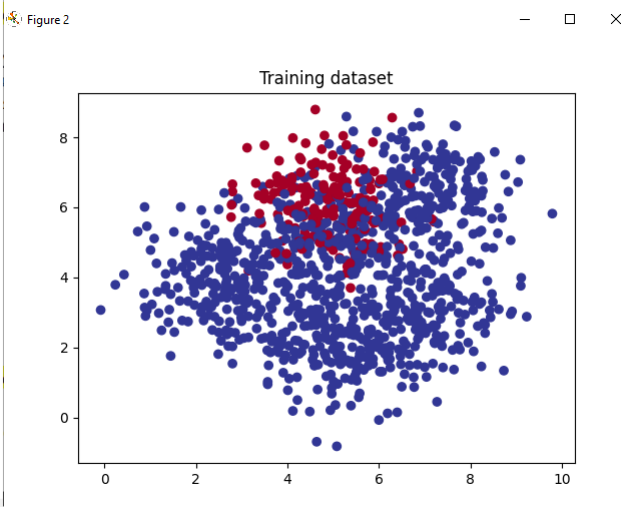
Використовуючи для аналізу дані, які містяться у файлі data_imbalance.txt проведіть обробку з урахуванням дисбалансу класів.

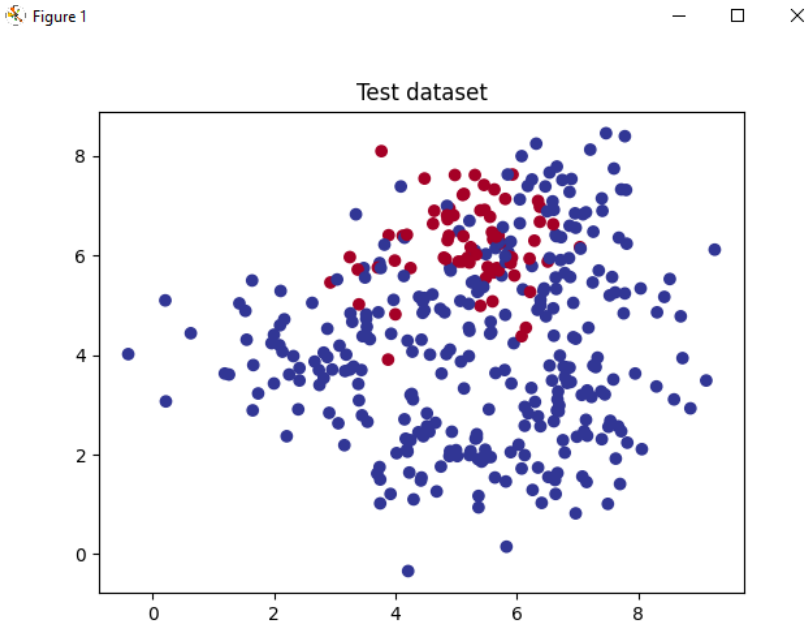
```
import sys
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
```

```
from utilities import visualize_classifier

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

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

# Класифікатор на основі ExtraTrees
classifier = ExtraTreesClassifier(**params)
classifier.fit(X_train, y_train)

# Візуалізація класифікатора на навчальному наборі
visualize_classifier(classifier, X_train, y_train, 'Training dataset')

# Передбачення та візуалізація результатів для тестового набору
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()
```







Скрипт реалізує класифікацію даних із дисбалансом класів за допомогою ExtraTreesClassifier. Дані завантажуються, розділяються на класи, візуалізуються, після чого розбиваються на тренувальний та тестовий набори. Класифікатор оцінюється за точністю, відчутністю та специфічністю для обох наборів. Можливе балансування класів для покращення результатів.

Завдання 3. Знаходження оптимальних навчальних параметрів за допомогою сіткового пошуку

Використовуючи дані, що містяться у файлі знайти оптимальних навчальних параметрів за допомогою сіткового пошуку. У процесі роботи з класифікаторами вам не завжди відомо, які параметри є найкращими. Їх підбір вручну методом грубої сили (шляхом перебору всіх можливих комбінацій) практично нереалізований. І тут на допомогу приходить сіточний пошук (grid search). Розглянемо як це робиться.

```
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.model_selection import GridSearchCV
from sklearn.ensemble import ExtraTreesClassifier

# Завантажуємо дані
input_file = 'data_random_forests.txt'
data = np.loadtxt(input_file, delimiter=',')

# Розділяємо на вхідні дані та мітки
X, y = data[:, :-1], data[:, -1]

# Розбиваємо дані на три класи
class_0 = np.array(X[y == 0])
class_1 = np.array(X[y == 1])
class_2 = np.array(X[y == 2])

# Розбиваємо 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)

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

# Метричні характеристики, які будемо використовувати для оцінки
metrics = ['precision_weighted', 'recall_weighted']

# Для кожної метрики виконуємо сітковий пошук
for metric in metrics:
    print(f"\n##### Searching optimal parameters for {metric}")

    # Створення класифікатора ExtraTreesClassifier
    classifier = GridSearchCV(
        ExtraTreesClassifier(random_state=0),
        parameter_grid,
        cv=5,
        scoring=metric
    )
```

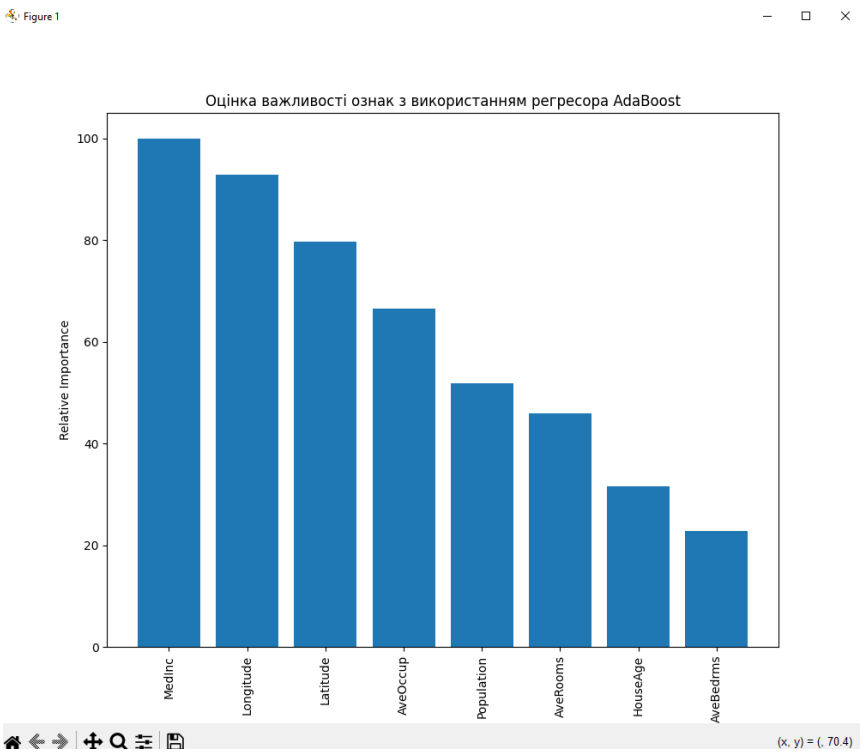
```python
    # Навчаємо класифікатор
    classifier.fit(X_train, y_train)

    # Виводимо результати для кожної комбінації параметрів
    print("\nGrid scores for the parameter grid:")
    for params, avg_score in zip(classifier.cv_results_['params'],
classifier.cv_results_['mean_test_score']):
        print(f"{params} --> {round(avg_score, 3)}")

    # Виводимо найкращі параметри
    print("\nBest parameters:", classifier.best_params_)

    # Прогнозування на тестовому наборі
    y_pred = classifier.predict(X_test)

    # Звіт про продуктивність
    print("\nPerformance report:\n")
    print(classification_report(y_test, y_pred))
```

```
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
```

У завданні було розглянуто метод сіткового пошуку для підбору оптимальних параметрів класифікатора на основі набору даних. Використовуючи клас GridSearchCV, ми здійснили пошук найкращих значень параметрів, таких як n_estimators і max_depth, для класифікатора ExtraTreesClassifier. Для кожної метрики, зокрема точності та відгуку, були протестовані різні комбінації параметрів, після чого виведено найкращі результати. Такий підхід дозволяє автоматизувати налаштування моделі, забезпечуючи кращу її продуктивність на тестових даних.

Завдання 4. Обчислення відносної важливості ознак

Коли ми працюємо з наборами даних, що містять N-вимірні точки даних, необхідно розуміти, що не всі ознаки однаково важливі. Одні з них відіграють більшу роль, ніж інші. Маючи в своєму розпорядженні цю інформацією, можна зменшити кількість розмірностей, що враховуються. Ми можемо використовувати цю можливість зниження складності алгоритму та його прискорення. Іноді деякі ознаки виявляються зайвими. Отже, їх можна безболісно виключити із набору даних.




```
PS D:\123> python lab5_task_4.py

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

```python
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 California Housing
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)

# Створення та навчання регресора AdaBoost з використанням дерева рішень
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 = housing_data.feature_names

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

# Сортування важливості ознак
index_sorted = np.flipud(np.argsort(feature_importances))

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

# Переконайтесь, що ви використовуєте правильний формат для міток
sorted_feature_names = [feature_names[i] for i in index_sorted]

plt.figure()
plt.bar(pos, feature_importances[index_sorted], align='center')
plt.xticks(pos, sorted_feature_names, rotation=90)  # Використовуємо правильно
відсортовані назви ознак
plt.ylabel('Relative Importance')
plt.title('Оцінка важливості ознак з використанням регресора AdaBoost')
plt.show()
```

Ключові фактори: Згідно з аналізом, важливими факторами для прогнозування цін на житло є рівень доходу (MedInc), географічні координати (Longitude, Latitude). Це може свідчити про те, що вищі доходи та конкретне місцезнаходження будинків важливі для оцінки їх вартості.

Менше впливові фактори: Останні місця займають демографічні показники та технічні характеристики будинків, що може свідчити про те, що вони мають менш значущий вплив на ринок у порівнянні з іншими факторами.

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.

```
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

# Завантажуємо дані з файлу
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)

# Перетворення нечислових ознак на числові
label_encoder = []
X_encoded = np.empty(data.shape, dtype=object)

for i, item in enumerate(data[0]):
    if not item.isdigit():  # Перевіряємо, чи є значення нечисловим
        label_encoder.append(preprocessing.LabelEncoder())
        X_encoded[:, i] = label_encoder[-1].fit_transform(data[:, i])
    else:
        X_encoded[:, i] = data[:, i]

# Окремо витягуємо ознаки (X) та цільову змінну (y)
X = X_encoded[:, :-1].astype(int)  # Всі стовпці, крім останнього
y = X_encoded[:, -1].astype(int)   # Останній стовпець - кількість транспортних
засобів

# 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)

# 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)
```

```python
# Оцінка ефективності моделі на тестових даних
y_pred = regressor.predict(X_test)
print("Mean absolute error:", round(mean_absolute_error(y_test, y_pred), 2))

# Тестування на новій точці даних
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:
        # Витягуємо єдиний елемент з масиву
        test_datapoint_encoded[i] =
int(label_encoder[count].transform([test_datapoint[i]])[0])
        count += 1

test_datapoint_encoded = np.array(test_datapoint_encoded)

# Прогнозування для нової точки даних
predicted_traffic = int(regressor.predict([test_datapoint_encoded])[0])
print("Predicted traffic:", predicted_traffic)
```

```
PS D:\123> python lab5_task_5.py
Mean absolute error: 7.42
Predicted traffic: 26
```

Модель гранично випадкових лісів (ExtraTreesRegressor) успішно прогнозує інтенсивність дорожнього руху з середньою абсолютною помилкою 7.42. Прогноз для нової точки даних (26 транспортних засобів) також був виконаний коректно, що свідчить про ефективність моделі, хоча є можливість для її покращення.

Висновок

У результаті виконання лабораторної роботи було досліджено методи ансамблевого навчання в Python, зокрема Bagging, Boosting та Stacking. Виявлено, що ансамблі покращують точність моделей порівняно з окремими алгоритмами. Boosting показав високу точність, орієнтуючись на покращення слабких моделей, а Bagging (наприклад, Random Forest) зменшував варіативність. Stacking дозволив комбінувати різні моделі для досягнення найкращих результатів. Загалом, ансамблеві методи є ефективними для підвищення якості машинного навчання.