**ЛАБОРАТОРНА РОБОТА № 5**

ДОСЛІДЖЕННЯ МЕТОДІВ АНСАБЛЕВОГО НАВЧАННЯ

***Мета:*** використовуючи спеціалізовані бібліотеки та мову програмування Python дослідити різні ансаблів у машинному навчанні.

Варіант 12

**Хід роботи:**

**Завдання 1**

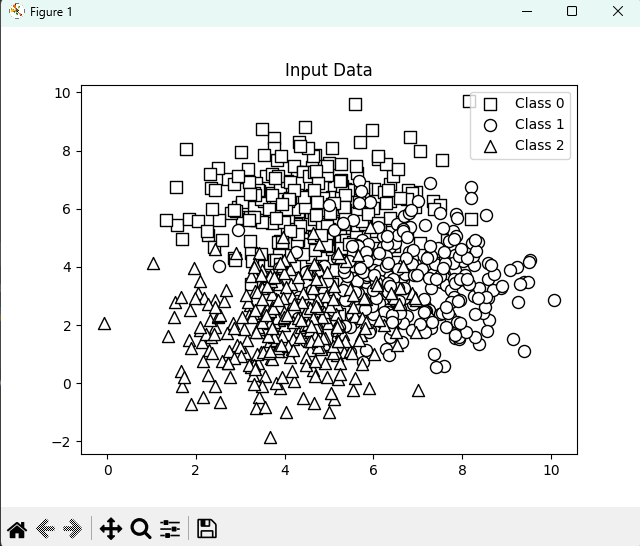
Створення класифікаторів на основі випадкових тагранично випадкових лісів

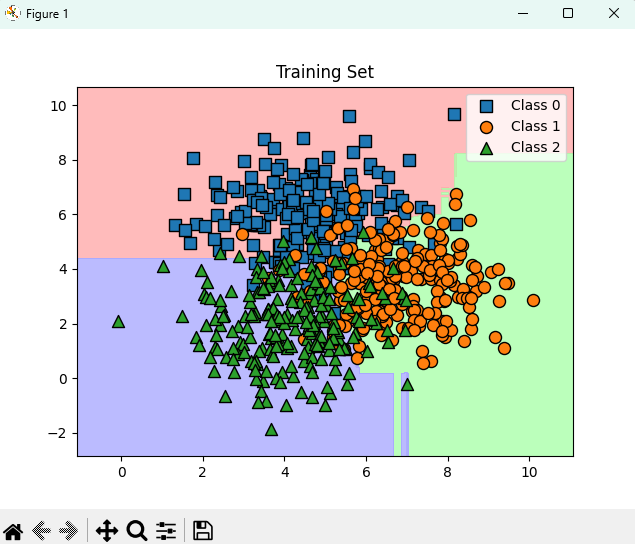
**Програмний код:**

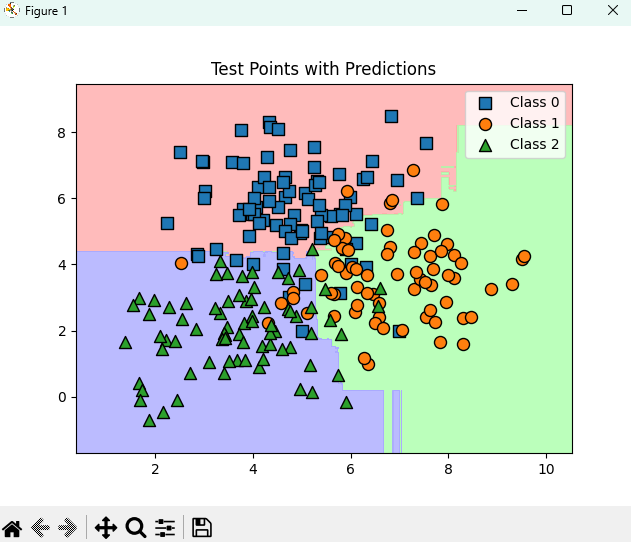
import argparse  
import numpy as np  
from sklearn.metrics import classification\_report  
from sklearn.model\_selection import train\_test\_split  
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier  
from matplotlib.colors import ListedColormap  
import matplotlib  
import matplotlib.pyplot as plt  
  
matplotlib.use('TkAgg')  
  
  
def build\_arg\_parser():  
 parser = argparse.ArgumentParser(  
 description='Classify data using Ensemble Learning techniques'  
 )  
 parser.add\_argument(  
 '--classifier-type',  
 dest='classifier\_type',  
 required=False,  
 default='rf',  
 choices=['rf', 'erf'],  
 help="Type of classifier to use; can be either 'rf' or 'erf'"  
 )  
 return parser  
  
  
def load\_data(input\_file):  
 data = np.loadtxt(input\_file, delimiter=',')  
 X, y = data[:, :-1], data[:, -1]  
 return X, y

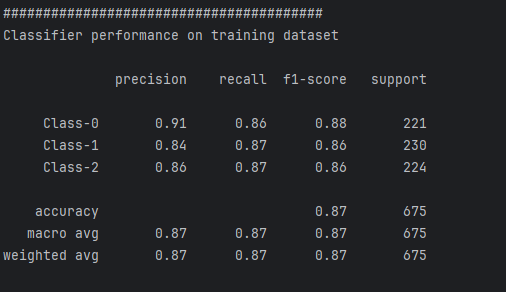
def plot\_input\_data(X, y):  
 classes = np.unique(y)  
 markers = ['s', 'o', '^']  
 plt.figure()  
 for idx, cls in enumerate(classes):  
 class\_data = X[y == cls]  
 plt.scatter(  
 class\_data[:, 0], class\_data[:, 1],  
 s=75, facecolors='white', edgecolors='black',  
 linewidth=1, marker=markers[idx], label=f'Class {int(cls)}'  
 )  
 plt.title('Input Data')  
 plt.legend()  
 plt.show()  
  
  
def get\_classifier(classifier\_type, params):  
 if classifier\_type == 'rf':  
 return RandomForestClassifier(\*\*params)  
 elif classifier\_type == 'erf':  
 return ExtraTreesClassifier(\*\*params)  
  
  
def plot\_decision\_boundaries(classifier, X, y, title):  
 h = 0.01  
 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, h),  
 np.arange(y\_min, y\_max, h))  
 Z = classifier.predict(np.c\_[xx.ravel(), yy.ravel()])  
 Z = Z.reshape(xx.shape)  
  
 plt.figure()  
 plt.contourf(xx, yy, Z, alpha=0.8, cmap=ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF']))  
 markers = ['s', 'o', '^']  
 for idx, cls in enumerate(np.unique(y)):  
 class\_data = X[y == cls]  
 plt.scatter(  
 class\_data[:, 0], class\_data[:, 1],  
 s=75, edgecolor='black', marker=markers[idx], label=f'Class {int(cls)}'  
 )  
 plt.title(title)  
 plt.legend()  
 plt.show()  
  
  
def main():  
 args = build\_arg\_parser().parse\_args()  
 classifier\_type = args.classifier\_type  
 params = {'n\_estimators': 100, 'max\_depth': 4, 'random\_state': 0}  
  
 input\_file = 'data\_random\_forests.txt'  
 X, y = load\_data(input\_file)  
 plot\_input\_data(X, y)  
 X\_train, X\_test, y\_train, y\_test = train\_test\_split(  
 X, y, test\_size=0.25, random\_state=5  
 )  
 classifier = get\_classifier(classifier\_type, params)  
 classifier.fit(X\_train, y\_train)  
  
 plot\_decision\_boundaries(classifier, X\_train, y\_train, "Training Set")  
 plot\_decision\_boundaries(classifier, X\_test, y\_test, "Test Set")  
  
 class\_names = [f'Class-{int(cls)}' for cls in np.unique(y)]  
 print("\n" + "#" \* 40)  
 print("Classifier performance on training dataset\n")  
 print(classification\_report(y\_train, classifier.predict(X\_train), target\_names=class\_names))  
 print("#" \* 40 + "\n")  
 print("Classifier performance on test dataset\n")  
 print(classification\_report(y\_test, classifier.predict(X\_test), target\_names=class\_names))  
 print("#" \* 40 + "\n")  
  
 test\_datapoints = np.array([[5, 5], [3, 6], [6, 4], [7, 2], [4, 4], [5, 2]])  
 print("Confidence measure:")  
 for datapoint in test\_datapoints:  
 probabilities = classifier.predict\_proba([datapoint])[0]  
 predicted\_class = f"Class-{np.argmax(probabilities)}"  
 print('\nDatapoint:', datapoint)  
 print('Predicted class:', predicted\_class)  
  
 plot\_decision\_boundaries(classifier, np.vstack((X\_test, test\_datapoints)),  
 np.hstack((y\_test, [0] \* len(test\_datapoints))), "Test Points with Predictions")  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

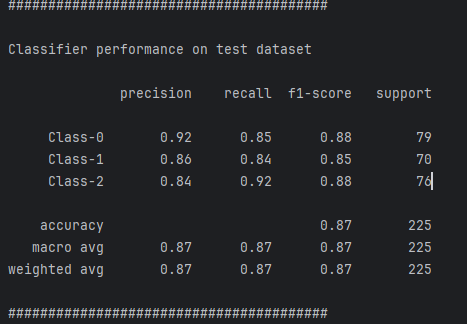
**Результат виконання:**

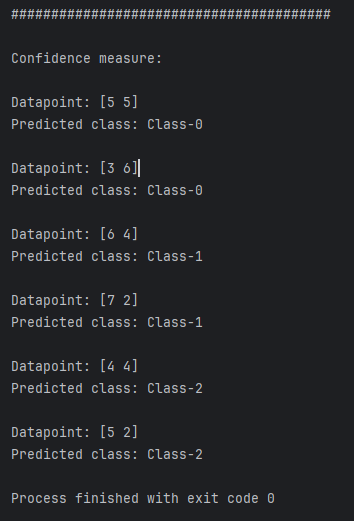




****

****

****

****

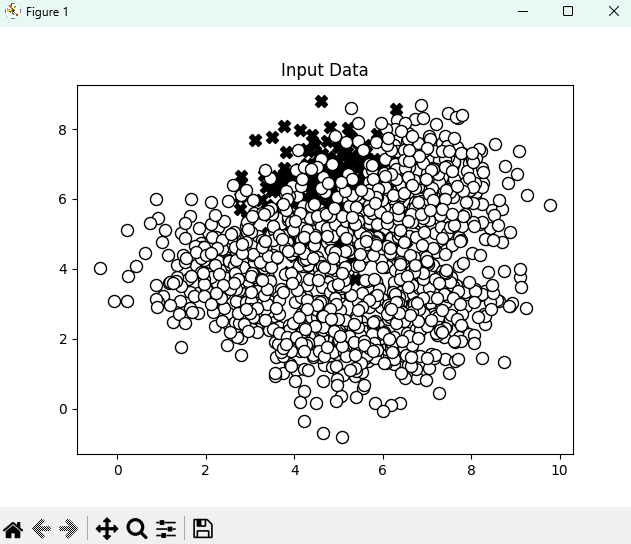
**Висновок:** Класифікатори сконструйовані на основі випадкового і гранично випадкового лісу схожі між собою, але класифікатор на основі випадкового лісу виконує свою роботу дещо краще за інший варіант.

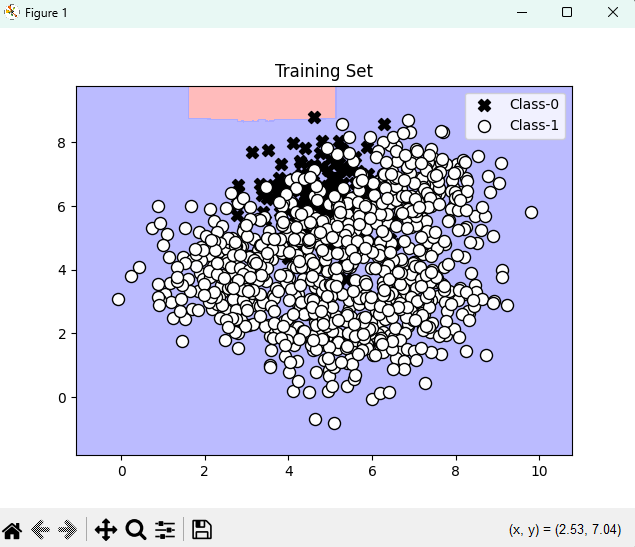
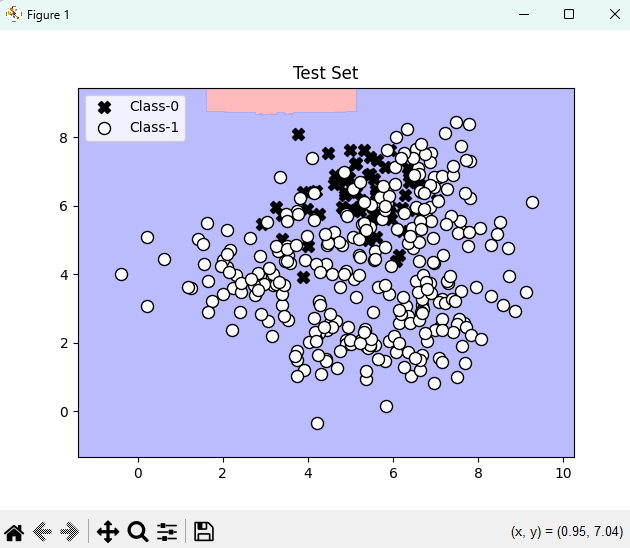
**Завдання 2** Обробка дисбалансу класів

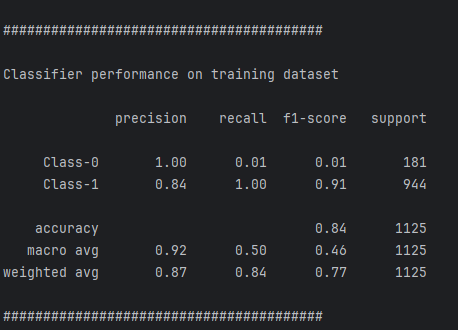
**Програмний код:**

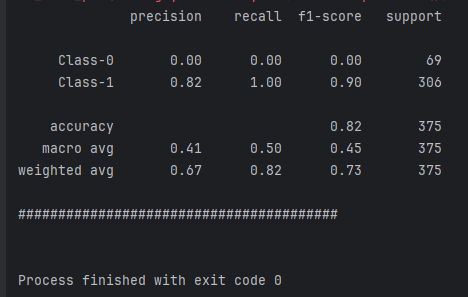
import sys  
import numpy as np  
from sklearn.ensemble import ExtraTreesClassifier  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import classification\_report  
from matplotlib.colors import ListedColormap  
import matplotlib  
import matplotlib.pyplot as plt  
  
matplotlib.use('TkAgg')  
  
  
def visualize\_classifier(classifier, X, y, title):  
 h = 0.01  
 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, h),  
 np.arange(y\_min, y\_max, h))  
 Z = classifier.predict(np.c\_[xx.ravel(), yy.ravel()])  
 Z = Z.reshape(xx.shape)  
  
 plt.figure()  
 plt.contourf(xx, yy, Z, alpha=0.8, cmap=ListedColormap(['#FFAAAA', '#AAAAFF']))  
 markers = ['X', 'o']  
 colors = ['black', 'white']  
 for idx, cls in enumerate(np.unique(y)):  
 plt.scatter(  
 X[y == cls, 0], X[y == cls, 1],  
 c=colors[idx], edgecolor='black', marker=markers[idx], s=75,  
 label=f'Class-{int(cls)}'  
 )  
 plt.title(title)  
 plt.legend()  
 plt.show()  
  
  
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')  
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}  
if len(sys.argv) > 1:  
 if sys.argv[1] == 'balance':  
 params['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\_train, y\_train, title="Training Set")  
visualize\_classifier(classifier, X\_test, y\_test, title="Test Set")  
  
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, classifier.predict(X\_test), target\_names=class\_names))  
print("#" \* 40 + "\n")

**Результат виконання:**

****

**** ****

****

****

**Висновок:** З параметром «'class\_weight': 'balanced'» класифікатор працює більш збалансовано і параметри класифікації не скачуть від одиниці до нуля.

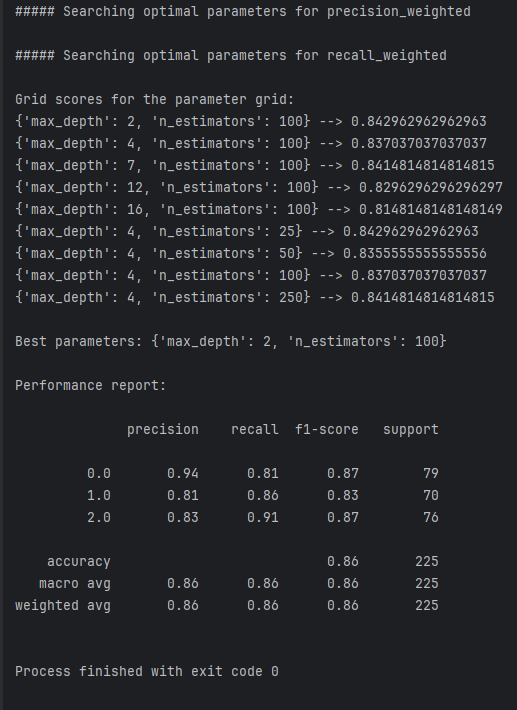
**Завдання 3**

Знаходження оптимальних навчальних параметрів за допомогою сіткового пошуку

**Програмний код:**

import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.metrics import classification\_report  
from sklearn.model\_selection import GridSearchCV  
from sklearn.ensemble import ExtraTreesClassifier  
from sklearn.model\_selection import train\_test\_split  
  
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]}]  
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:")  
for i in range(len(classifier.cv\_results\_['params'])):  
 print(classifier.cv\_results\_['params'][i], '-->', classifier.cv\_results\_['mean\_test\_score'][i])  
print("\nBest parameters:", classifier.best\_params\_)  
y\_pred = classifier.predict(X\_test)  
print("\nPerformance report:\n")  
print(classification\_report(y\_test, y\_pred))

**Результат виконання:**

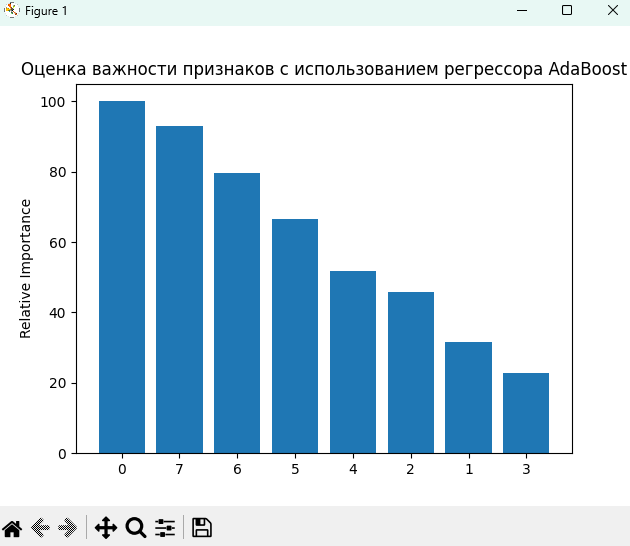
****

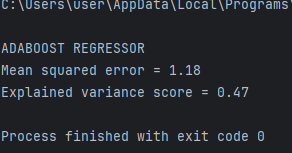
**Завдання 4** Обчислення відносної важливості ознак

**Програмний код:**

import numpy as np  
import matplotlib  
import matplotlib.pyplot as plt  
  
matplotlib.use('TkAgg')  
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.utils import shuffle  
from sklearn.model\_selection import train\_test\_split  
from sklearn.datasets import fetch\_california\_housing  
import pandas as pd  
import numpy as np  
  
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()  
plt.bar(pos, feature\_importances[index\_sorted], align='center')  
plt.xticks(pos, index\_sorted)  
plt.ylabel('Relative Importance')  
plt.title('Оценка важности признаков с использованием регрессора AdaBoost')  
plt.show()

**Результат виконання:**

****

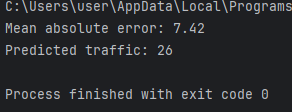
****

**Завдання 5** Прогнозування інтенсивності дорожнього руху задопомогою класифікатора на основі гранично випадкових лісів

**Програмний код:**

import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.metrics import classification\_report, 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, dtype=object)  
  
for i in range(data.shape[1]):  
 if data[:, i][0].isdigit():  
 X\_encoded[:, i] = data[:, i]  
 else:  
 encoder = preprocessing.LabelEncoder()  
 X\_encoded[:, i] = encoder.fit\_transform(data[:, i])  
 label\_encoders.append(encoder)  
  
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 = []  
  
for i, item in enumerate(test\_datapoint):  
 if item.isdigit():  
 test\_datapoint\_encoded.append(int(item))  
 else:  
 encoder = label\_encoders[i]  
 test\_datapoint\_encoded.append(encoder.transform([item])[0])  
  
test\_datapoint\_encoded = np.array(test\_datapoint\_encoded)  
  
print("Predicted traffic:", int(regressor.predict([test\_datapoint\_encoded])[0]))

**Результат виконання:**

****

***Висновки:*** в ході виконання лабораторної роботи було використано спеціалізовані бібліотеки та мову програмування Python та дослідив методи ансамблів у машинному навчанні.