**ЛАБОРАТОРНА РОБОТА № 5**

**ДОСЛІДЖЕННЯ МЕТОДІВ АНСАМБЛЕВОГО НАВЧАННЯ.**

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

**Хід роботи**

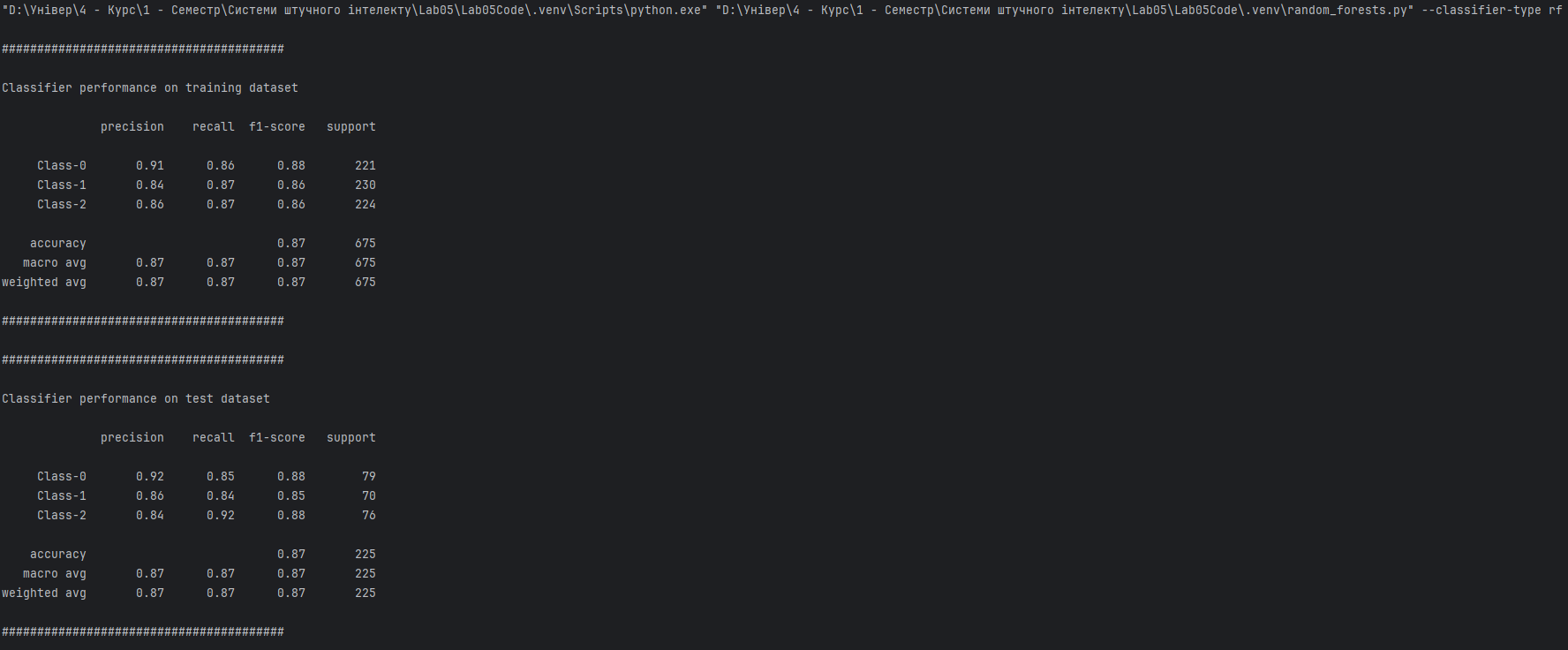
Завдання 2.1. Створення класифікаторів на основі випадкових та гранично випадкових лісів.

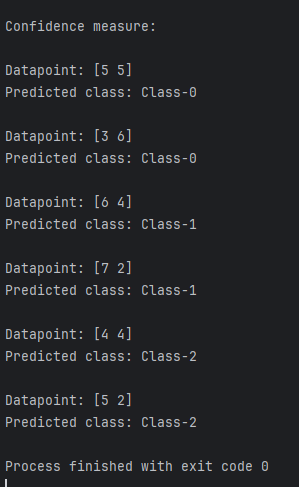
Лістинг програми:

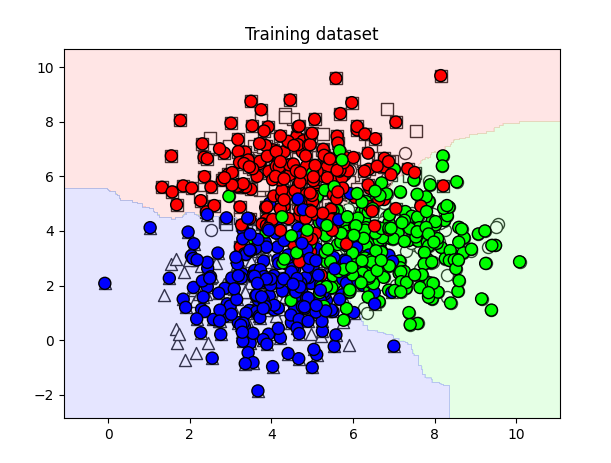
import argparse   
import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.metrics import classification\_report  
from sklearn.model\_selection import train\_test\_split  
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier  
from matplotlib.colors import ListedColormap  
  
# 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

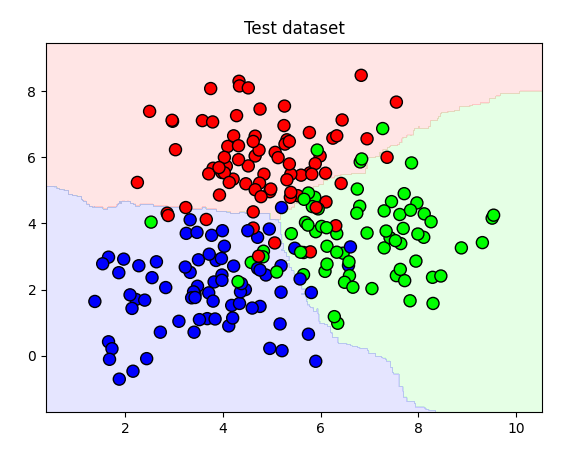
args = build\_arg\_parser().parse\_args()  
 classifier\_type = args.classifier\_type  
  
 # Load input data  
 input\_file = 'data\_random\_forests.txt'  
 data = np.loadtxt(input\_file, delimiter=',')  
 X, y = data[:, :-1], data[:, -1]  
  
 # Separate input data into three classes based on labels  
 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()

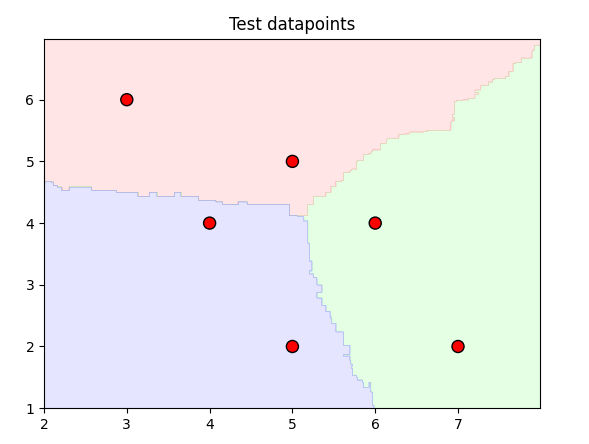
Виконання програми:  
--classifier-type rf



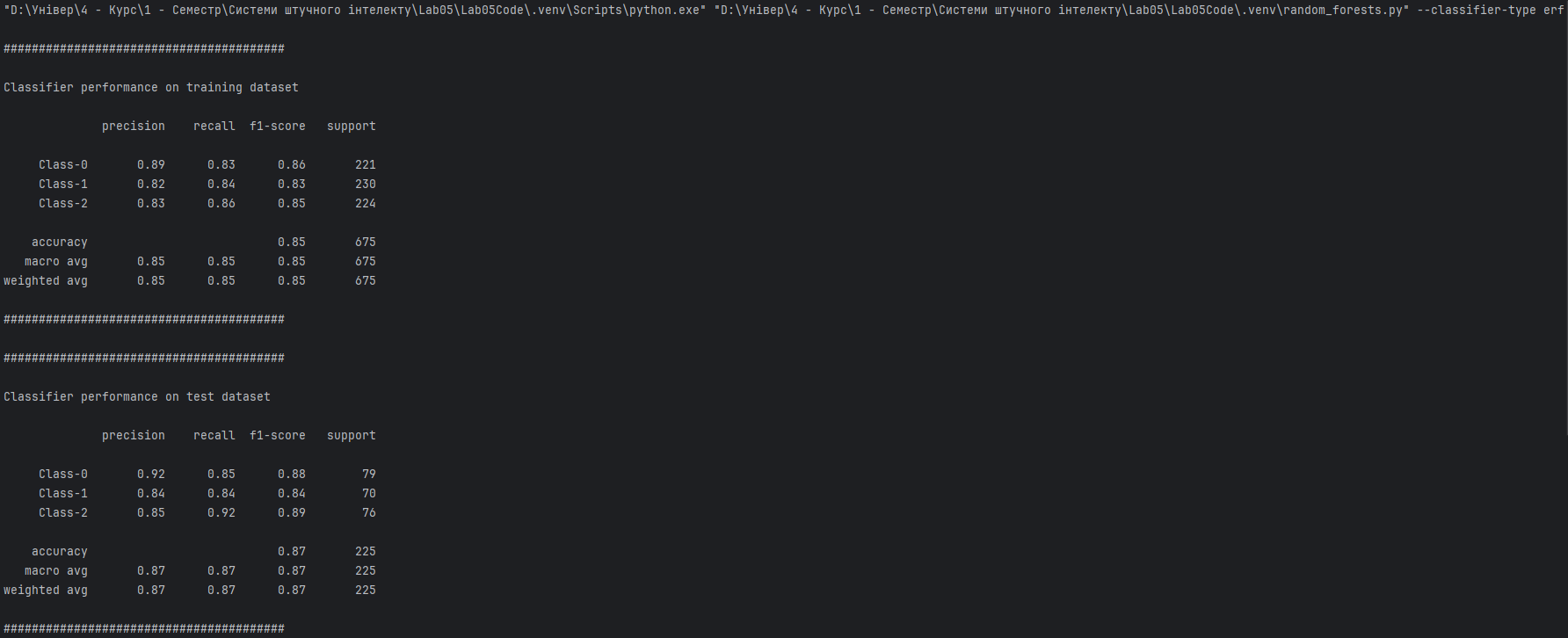


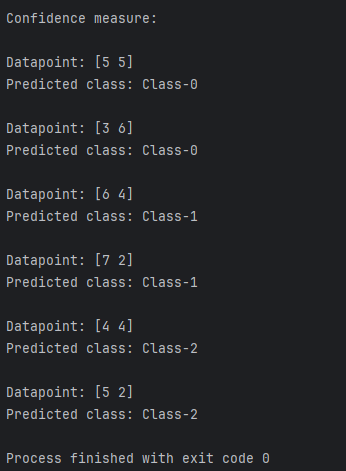


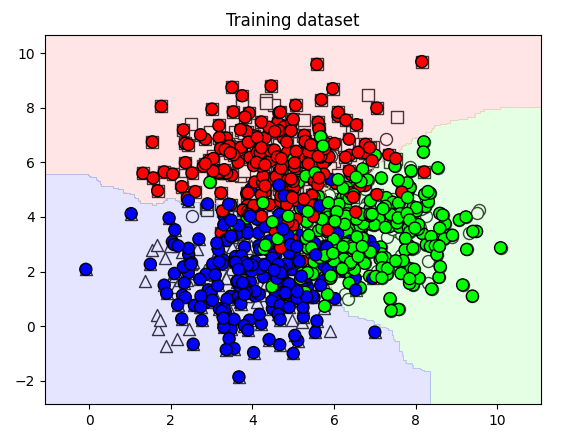


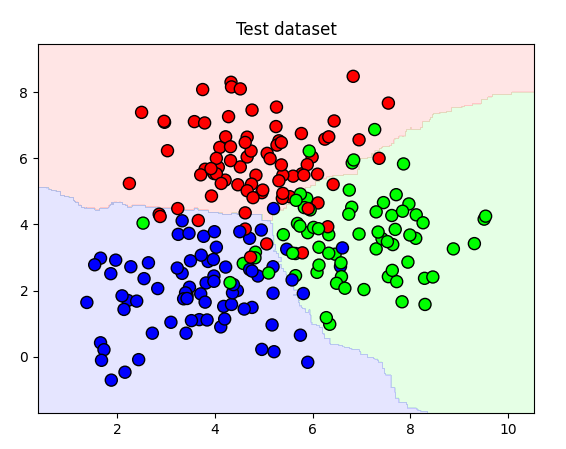


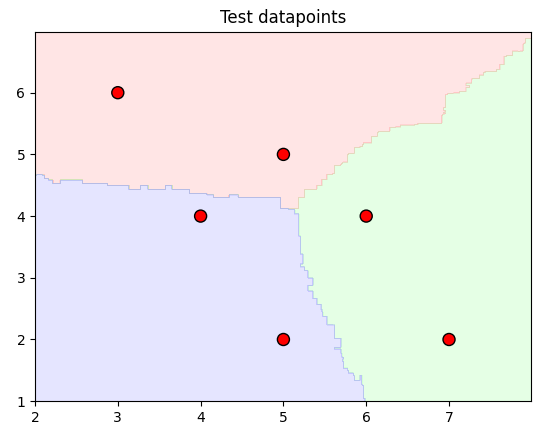
--classifier-type erf







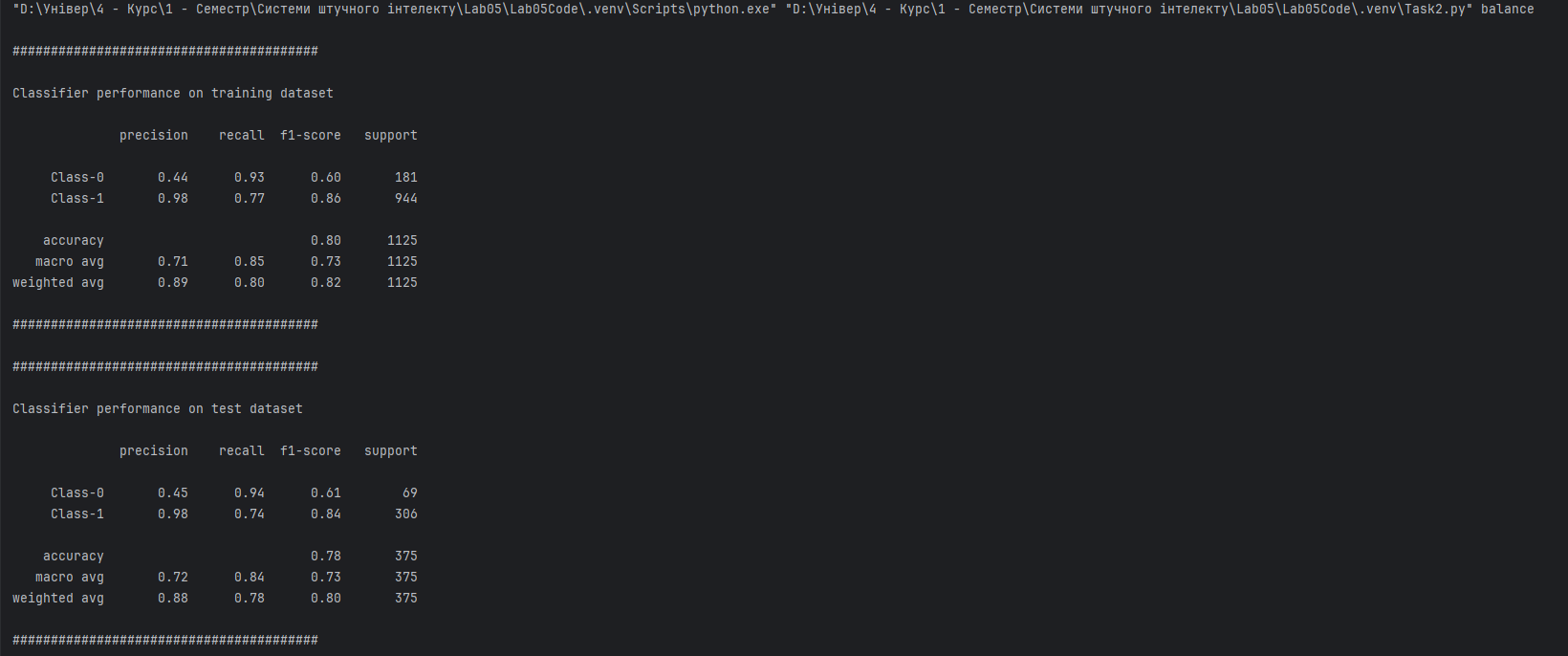


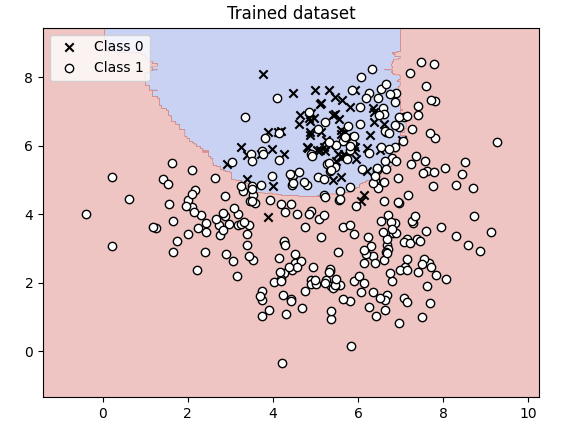
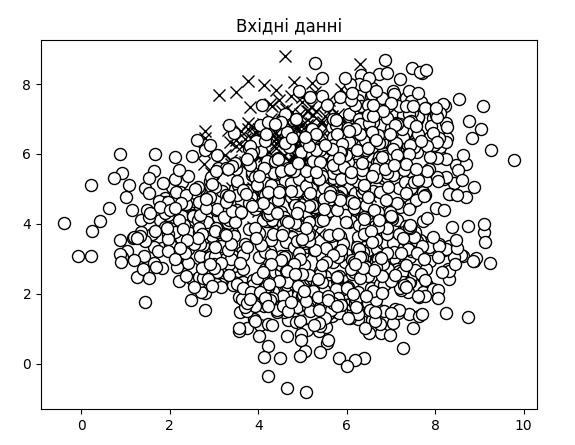


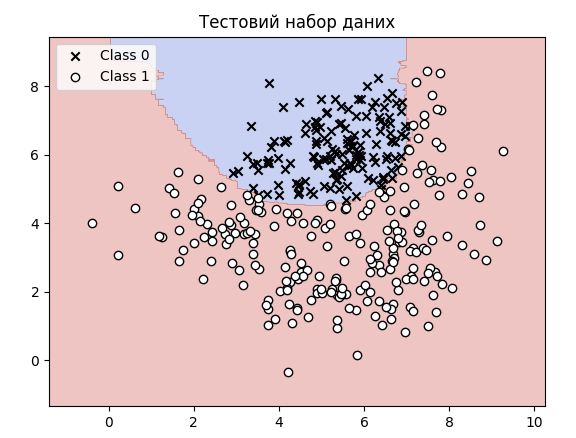
Завдання 2.2. Обробка дисбалансу класів

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}  
  
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: 



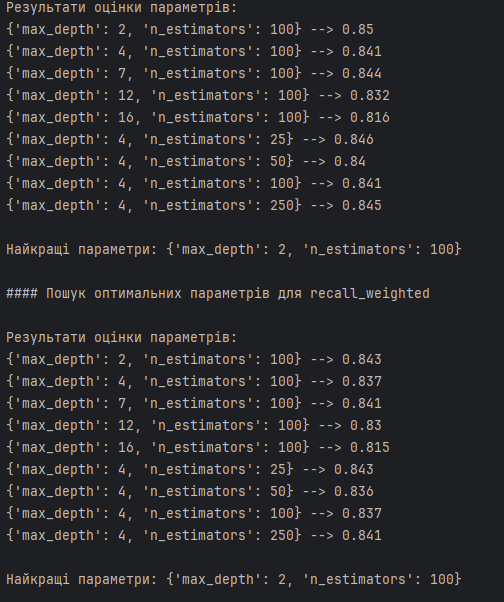


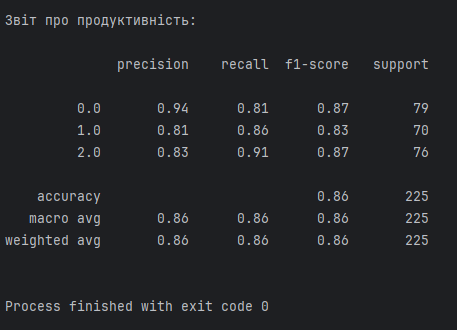
Завдання 2.3. Знаходження оптимальних навчальних параметрів за допомогою сіткового пошуку

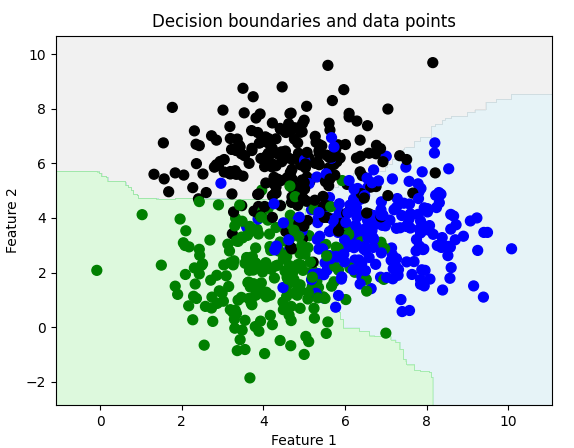
Лістинг програми:

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:





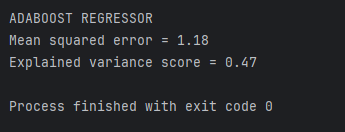


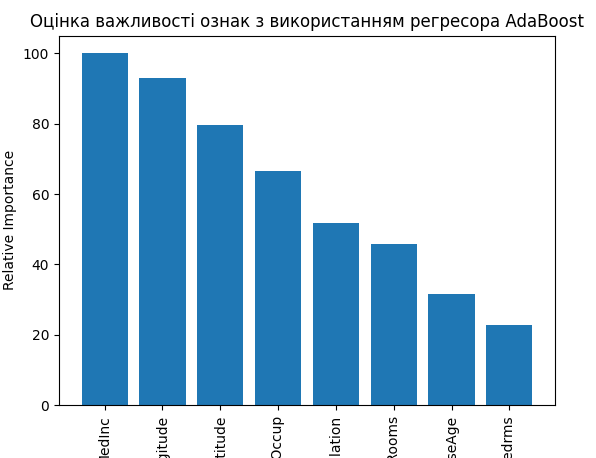
Завдання 2.4. Обчислення відносної важливості ознак

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 Х  
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:



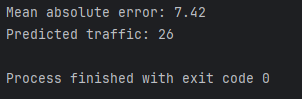


Завдання 2.5. Прогнозування інтенсивності дорожнього руху за допомогою класифікатора на основі гранично випадкових лісів

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:



Посилання на ГітХаб: <https://github.com/KaidanovychBohdan/SystemOfAI>

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