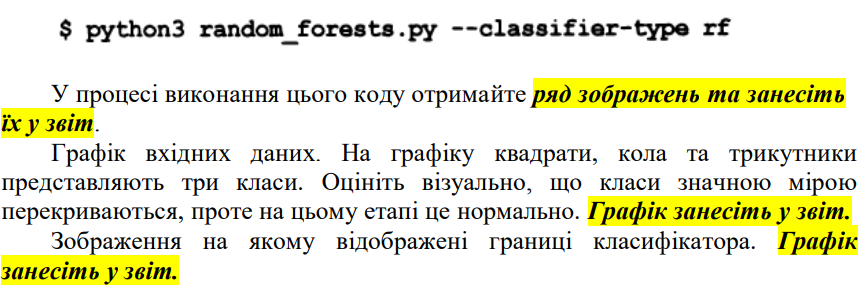
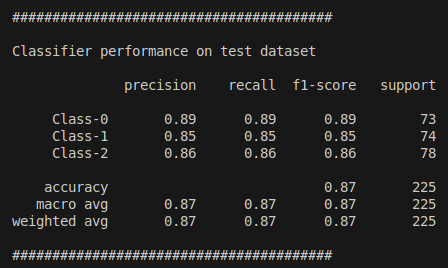
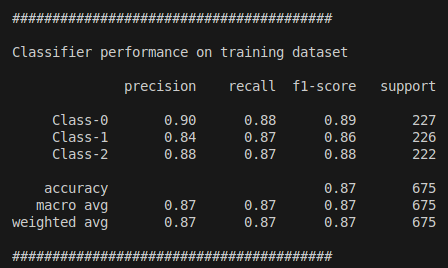
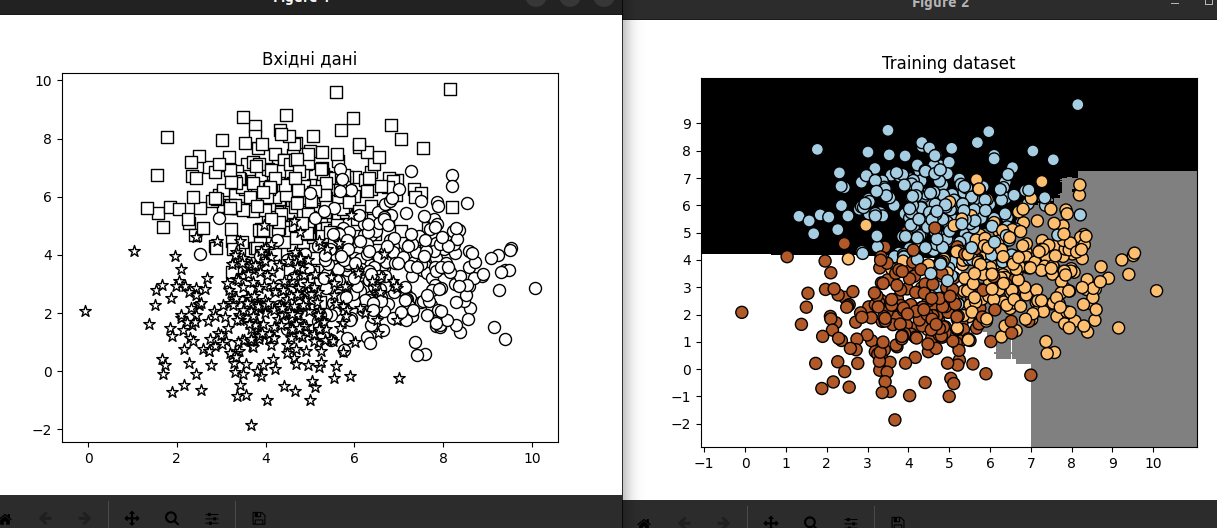
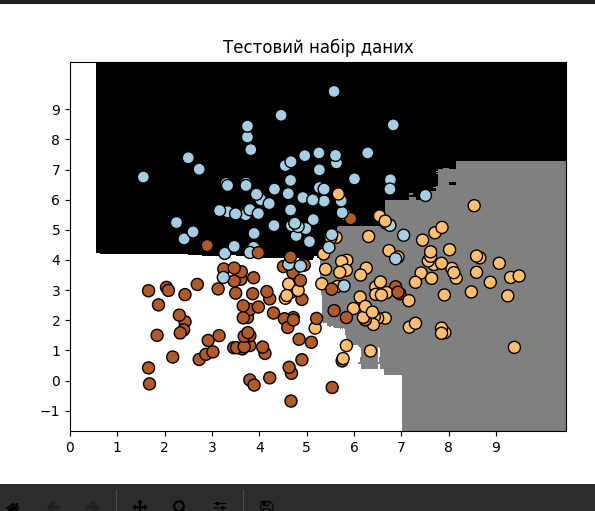


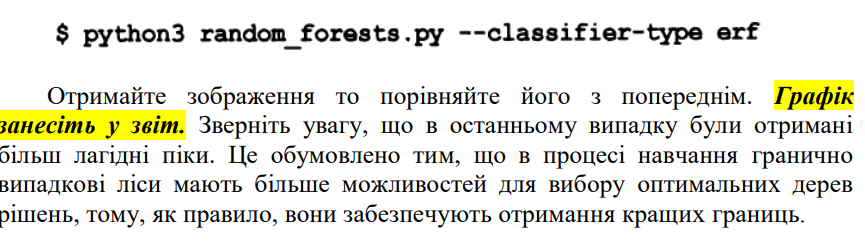
Завдання 1

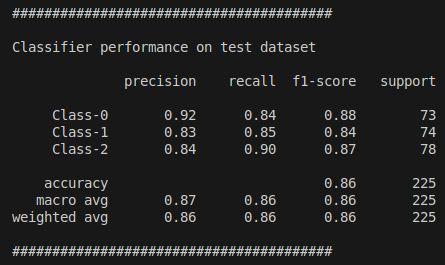
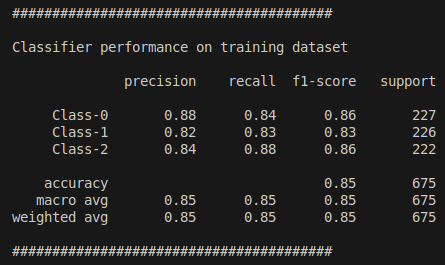


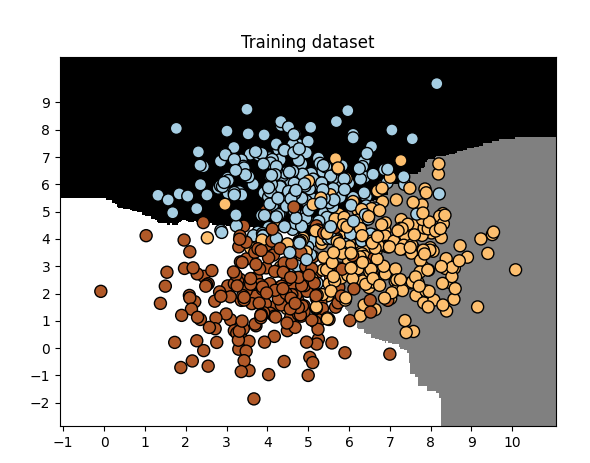


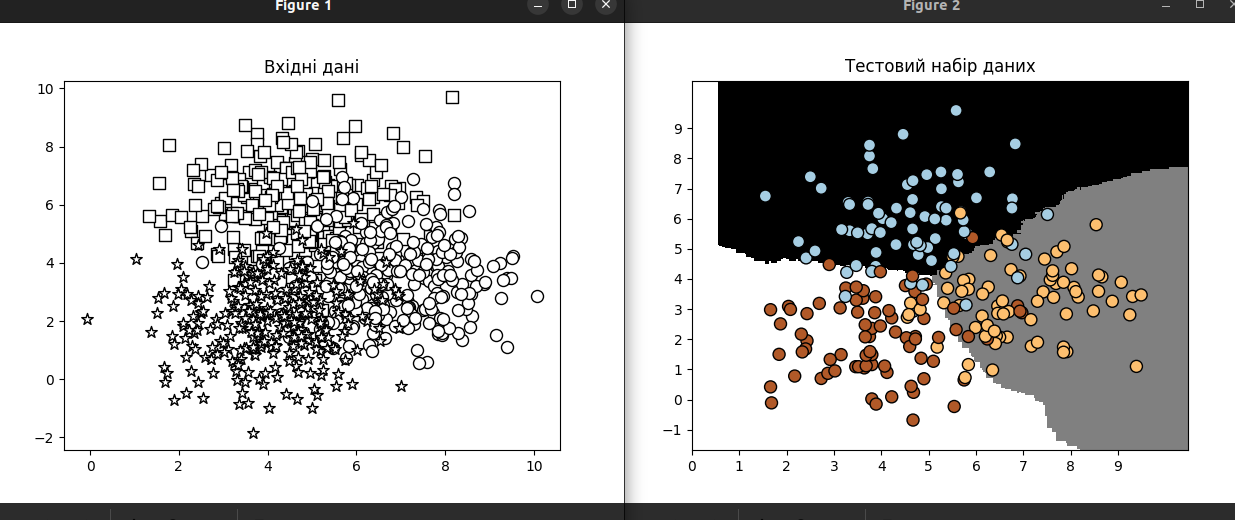


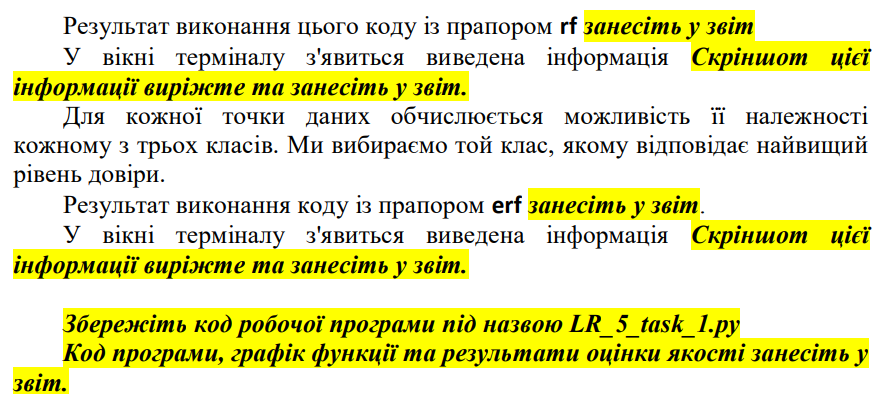




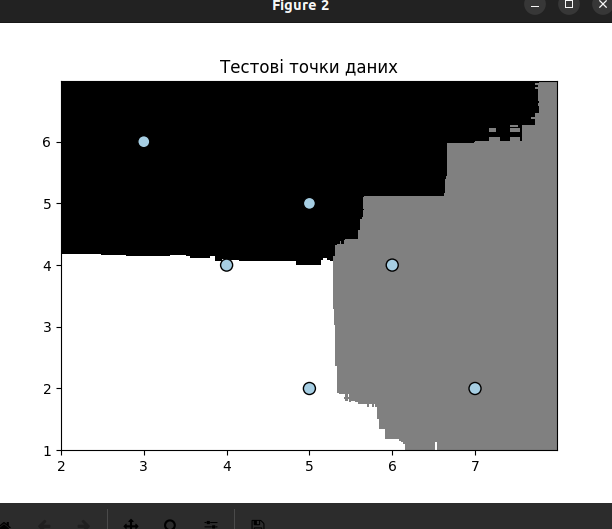


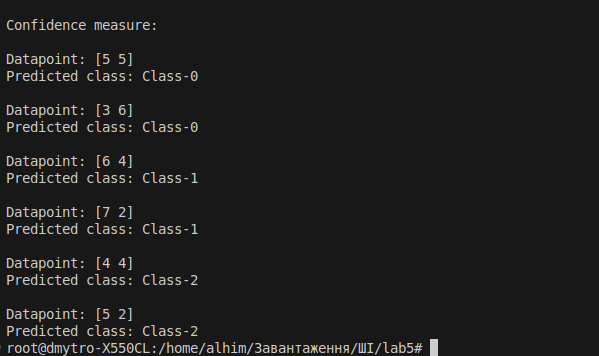




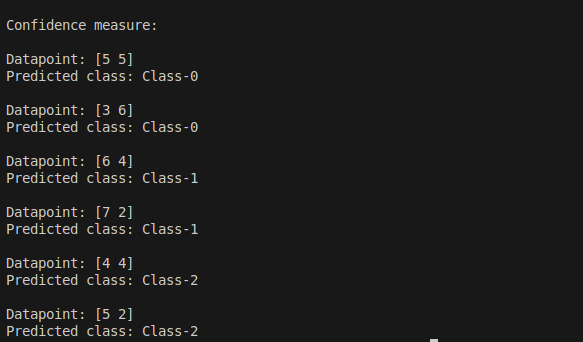


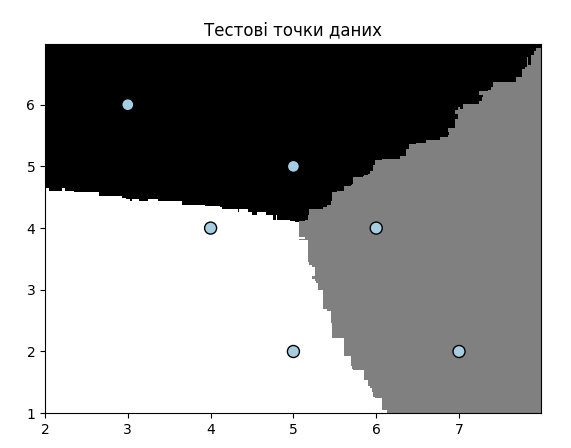
RF:





ERF:





Лістинг:

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 # Updated import

from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier

from utilities import visualize\_classifier

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 classifier to use; can be either 'rf' or 'erf'")

return parser

if \_\_name\_\_ == '\_\_main\_\_':

args = build\_arg\_parser().parse\_args()

classifier\_type = args.classifier\_type

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',

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('Вхідні дані')

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state=2)

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, 'Тестовий набір даних')

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")

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) )

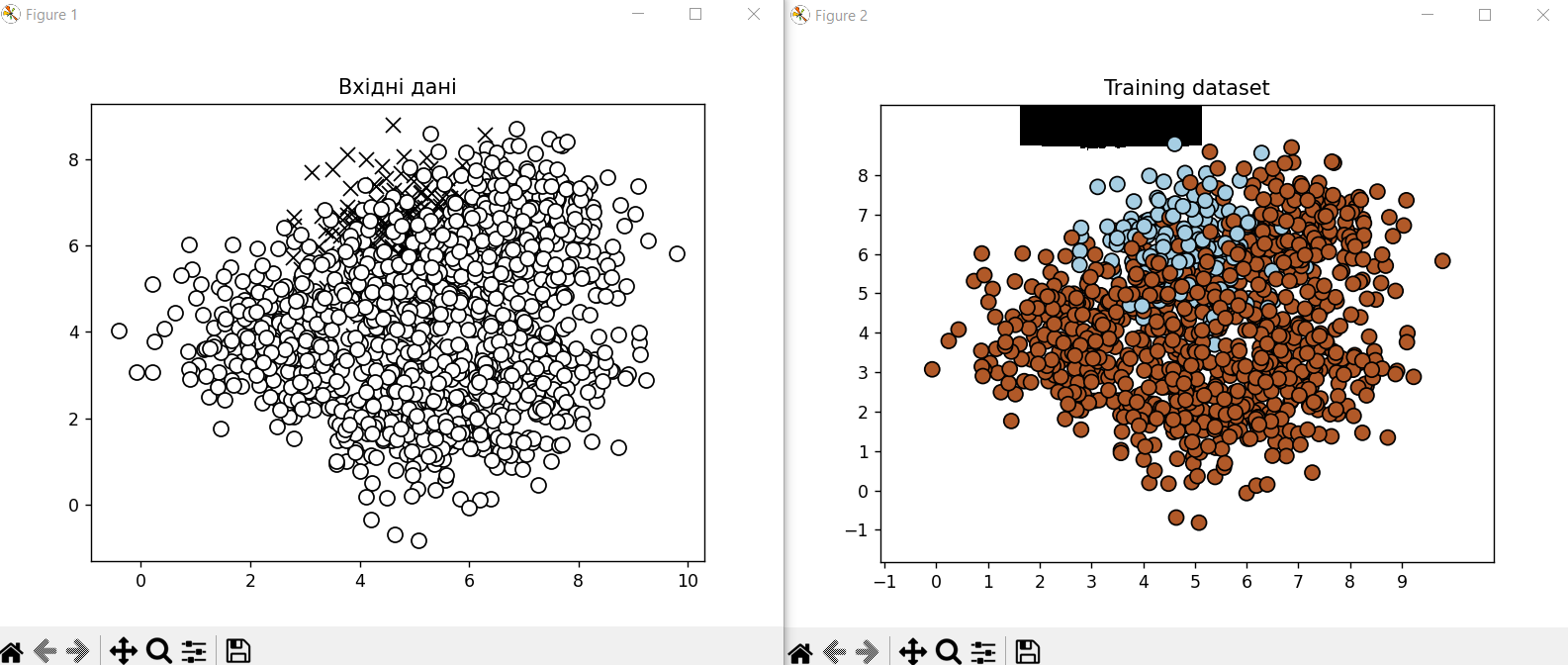
print ('\nDatapoint:', datapoint)

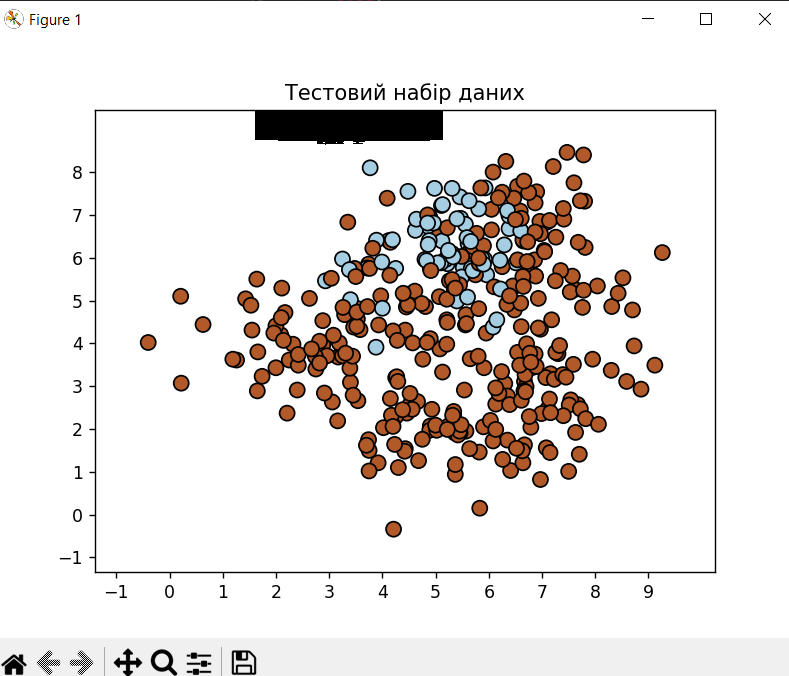
print ('Predicted class:', predicted\_class)

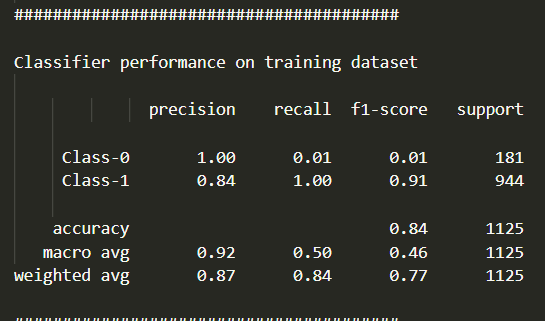
visualize\_classifier(classifier, test\_datapoints, [0]\*len(test\_datapoints), 'Тестові точки даних')

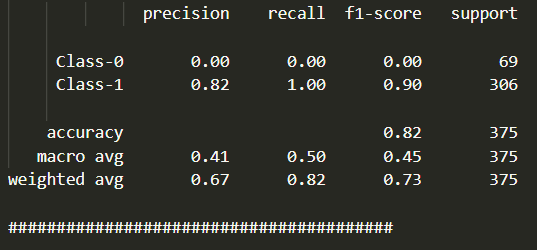
plt.show()

**Завдання 2.2. Обробка дисбалансу класів**



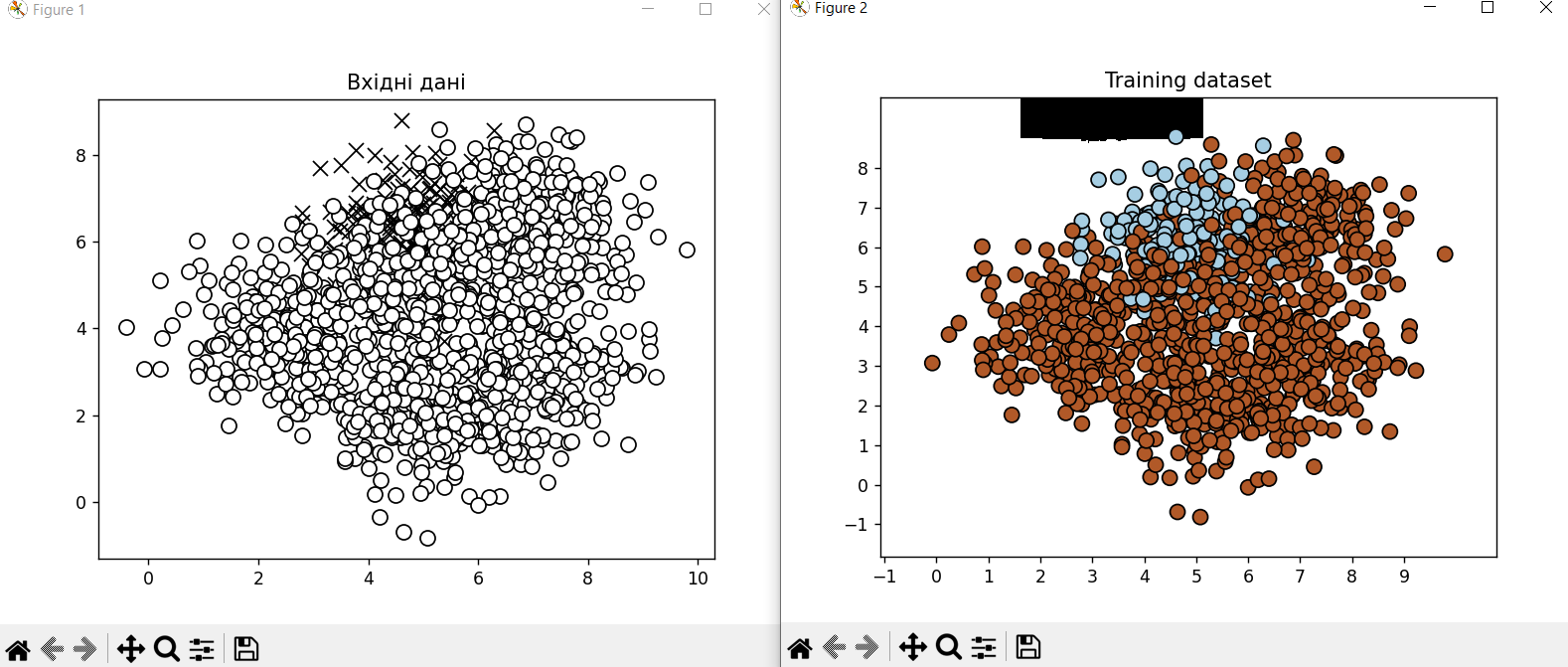


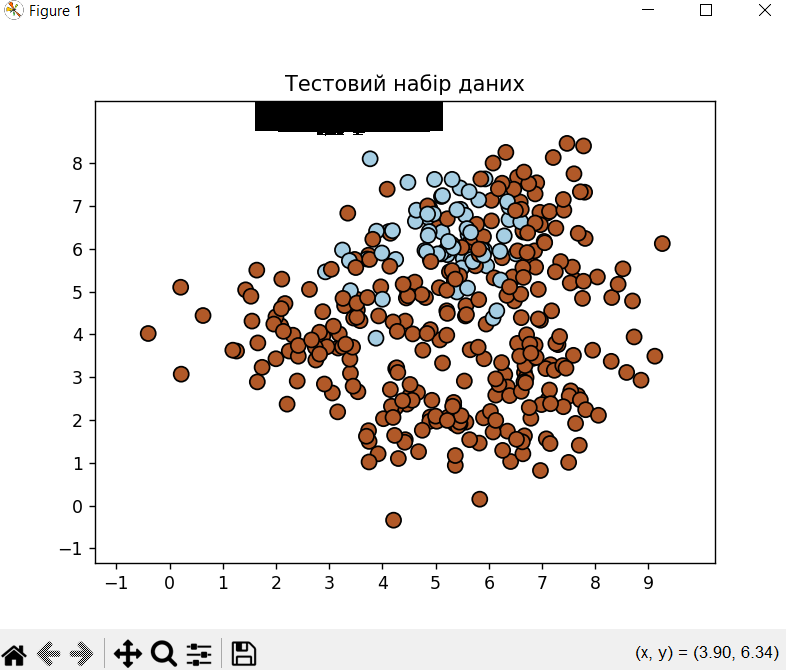


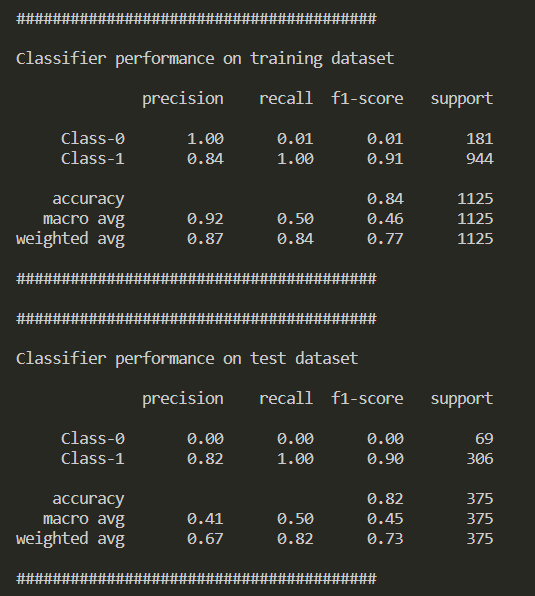


Тепер запустимо програму з команди:

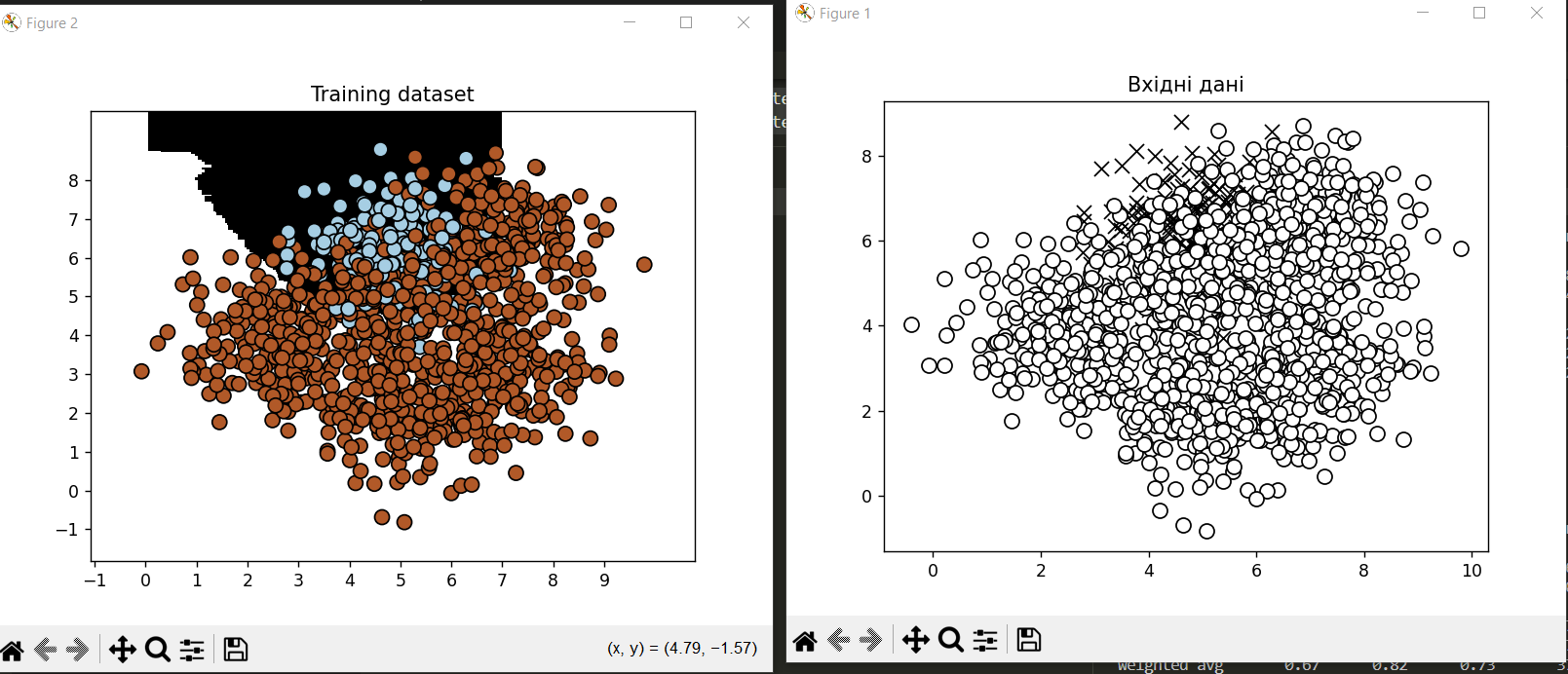


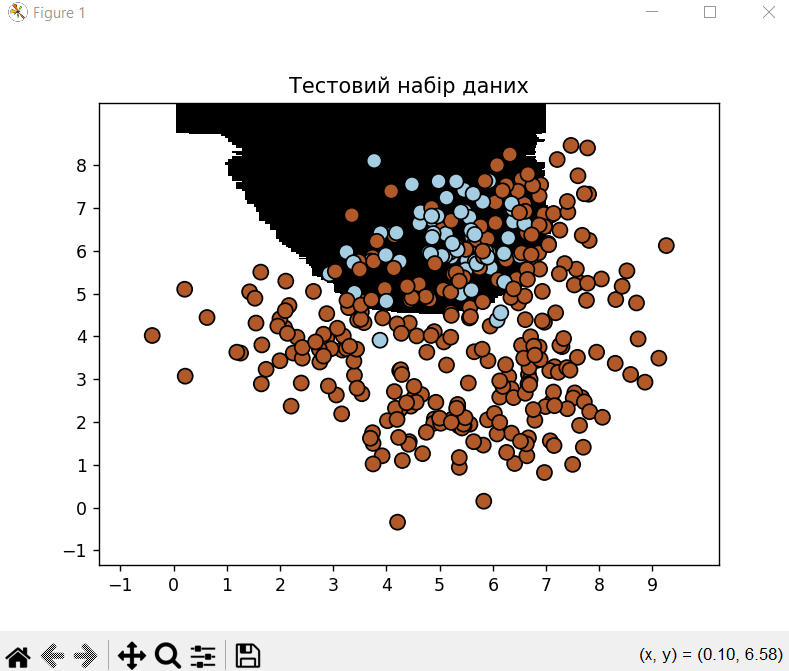


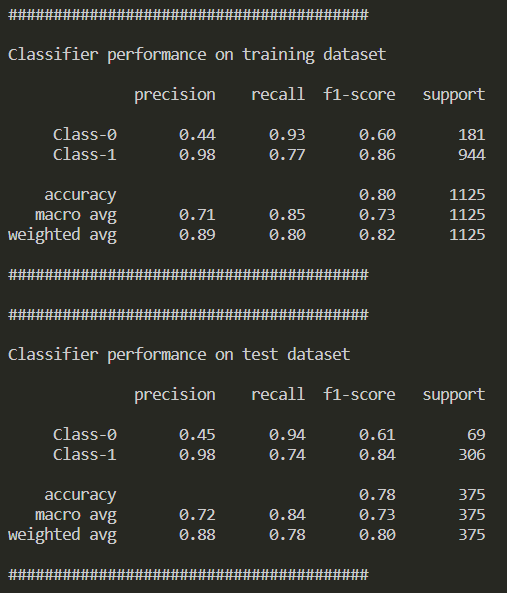




Тепер з використанням 







Лістинг:

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

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("Вхідні дані")

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 ValueError("Invalid input argument; should be 'balance'")

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, 'Тестовий набір 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)

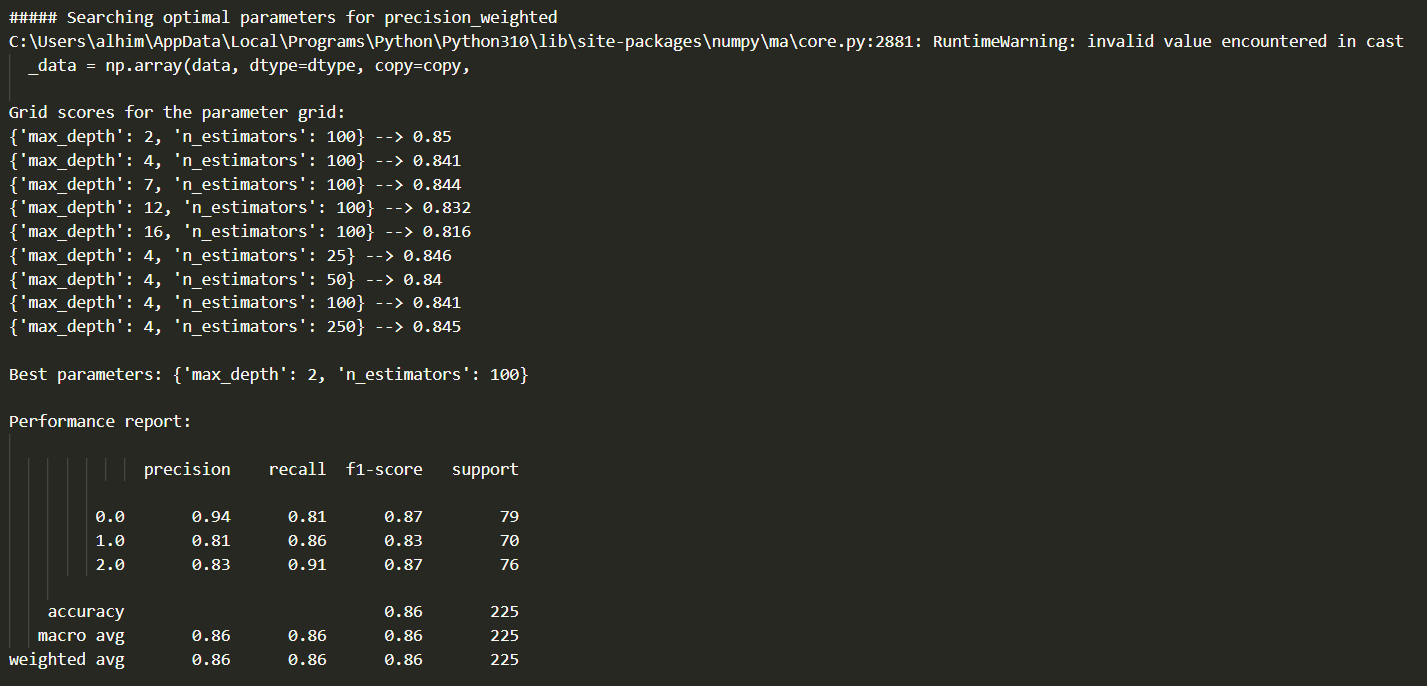
print("\nClassifier performance on test dataset\n")

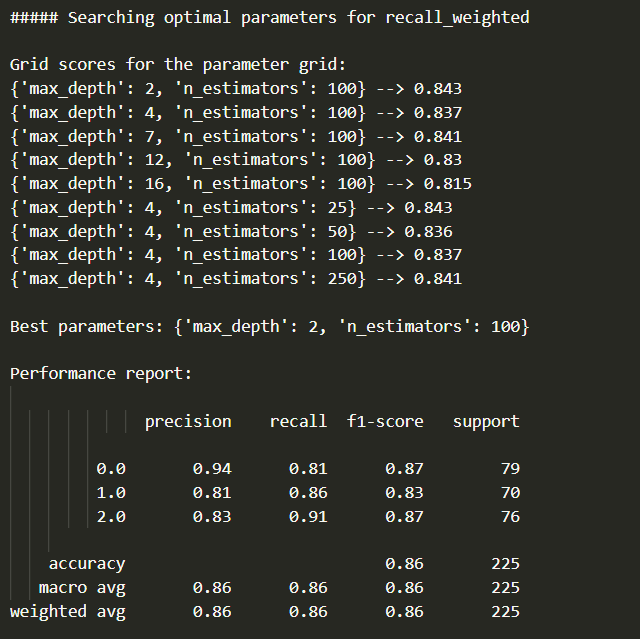
print(classification\_report(y\_test, y\_test\_pred, *target\_names*=class\_names))

print("#" \* 40 + "\n")

plt.show()

**Завдання 2.3. Знаходження оптимальних навчальних параметрів за допомогою сіткового пошуку**





Лістинг:

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 utilities import visualize\_classifier

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])

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:")

    for params, avg\_score in zip(classifier.cv\_results\_['params'], classifier.cv\_results\_['mean\_test\_score']):

        print(params, '-->', round(avg\_score, 3))

    print("\nBest parameters:", classifier.best\_params\_)

    y\_pred = classifier.predict(X\_test)

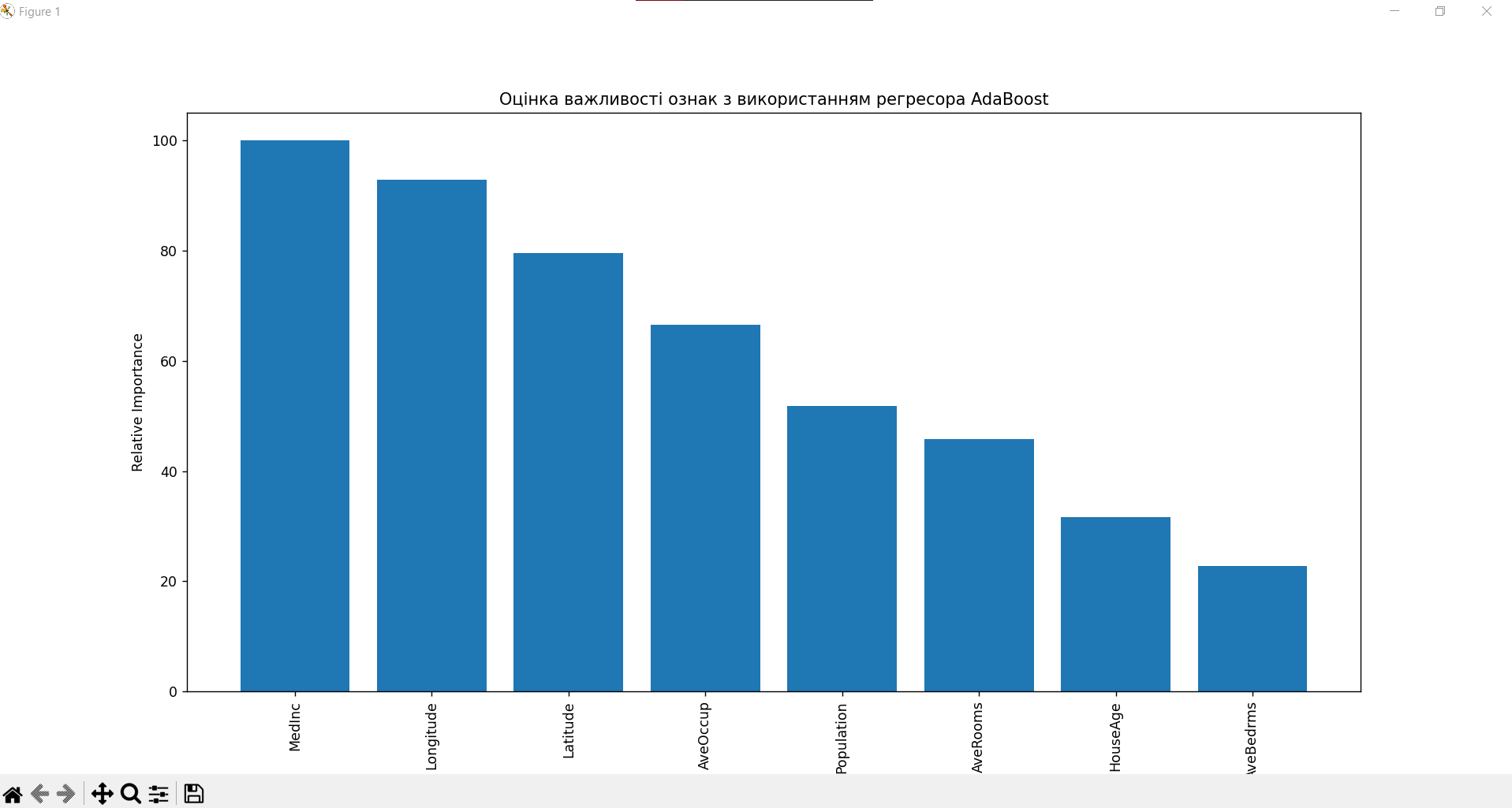
    print("\nPerformance report: \n")

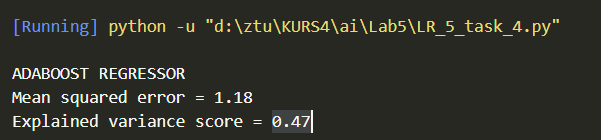
    print(classification\_report(y\_test, y\_pred))

**Завдання 2.4. Обчислення відносної важливості ознак**

Код з завдання було змінено з даної причини:







Лістинг:

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

# Load the California housing dataset

housing\_data = 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, np.array(feature\_names)[index\_sorted], *rotation*=90)

plt.ylabel("Relative Importance")

plt.title("Оцінка важливості ознак з використанням регресора AdaBoost")

plt.show()

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



Лістинг:

import numpy as np

import matplotlib.pyplot as plt

from sklearn.metrics import classification\_report, 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 = []

with open(input\_file, 'r') as f:

    for line in f.readlines():

        items = line[:-1].split(',')

        data.append(items)

data = np.array(data)

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)

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 = [-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 = count + 1

test\_datapoint\_encoded = np.array(test\_datapoint\_encoded)

print("Predicted traffic:", int(regressor.predict([test\_datapoint\_encoded])[0]))

GIT: https://github.com/Alhim616/AI\_Labs\_Yanushevych