**Лабораторна робота №5**

Ресурси Keras. TensorFlow. Навчання лінійної регресії

**Виконав:** ІПЗ-21-3 Осипчук Антон Олексійович

**Github:** https://github.com/AntonOsypchuk1/ai\_lab/tree/main/lab5

**Завдання 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 sklearn.metrics import classification\_report

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 of 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/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('Input data')

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 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")

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)

print('Probability:', np.max(probabilities))

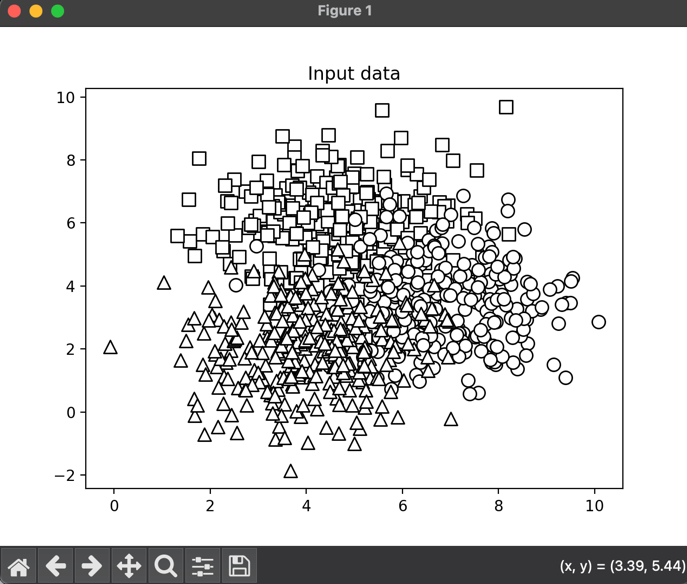
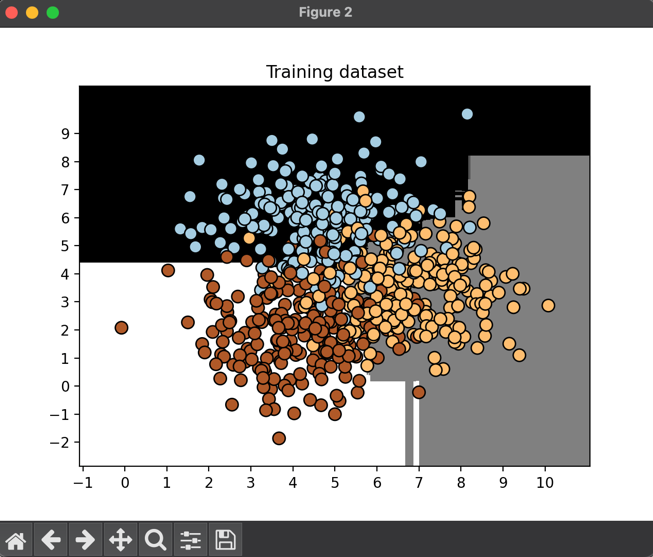
visualize\_classifier(classifier, test\_datapoints,

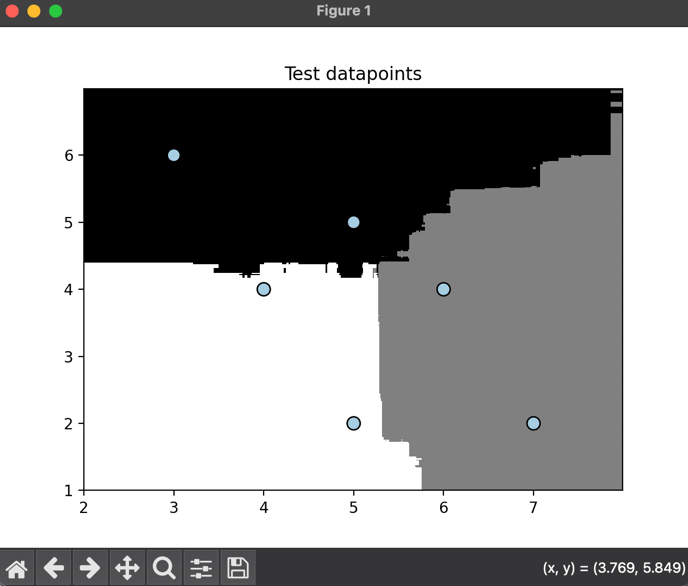
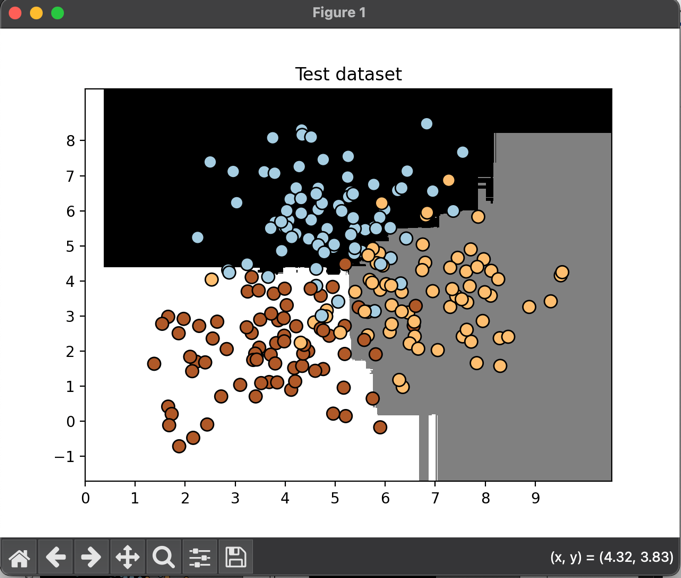
[0]\*len(test\_datapoints), 'Test datapoints')

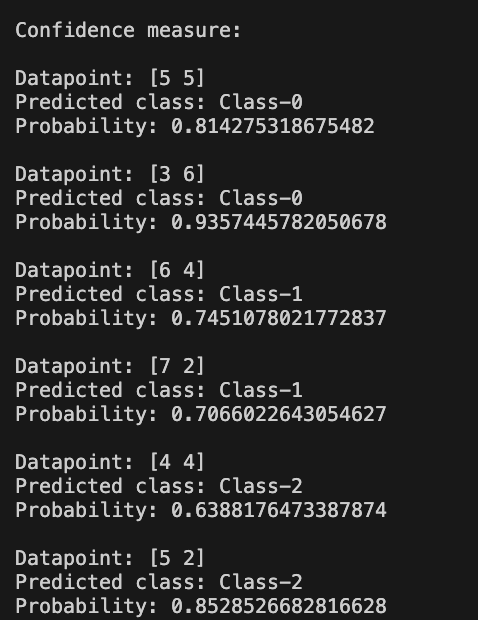
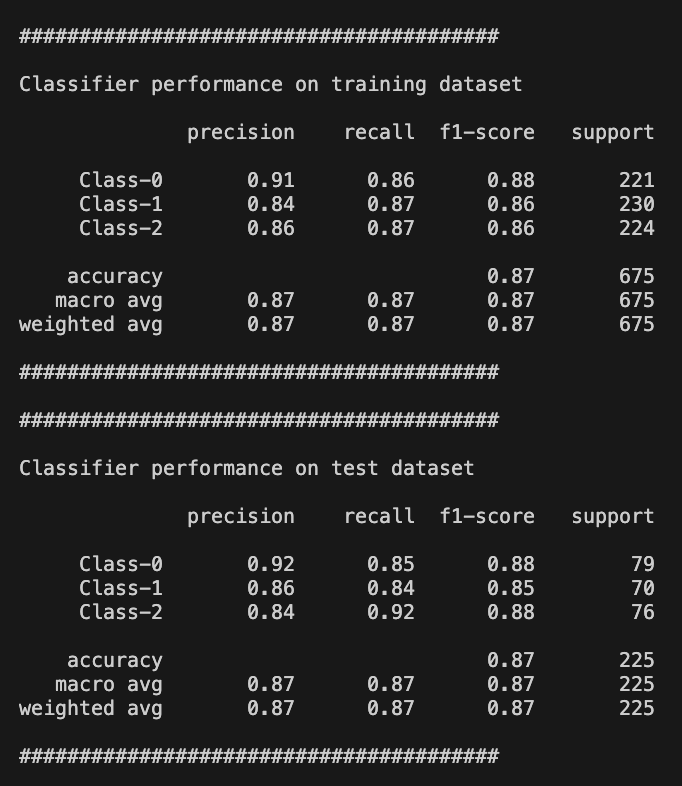
plt.show()

Результат виконання:

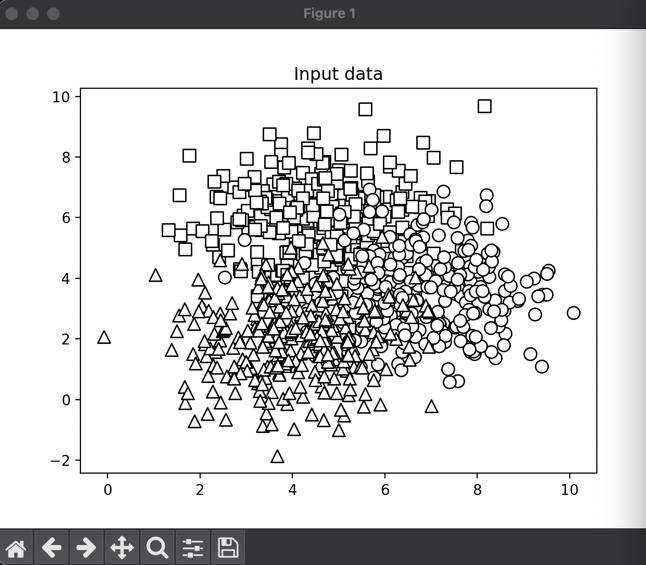
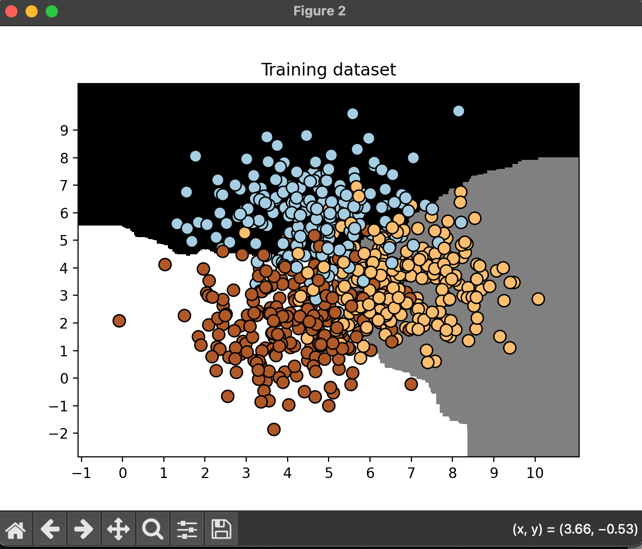
--classifier-type rf

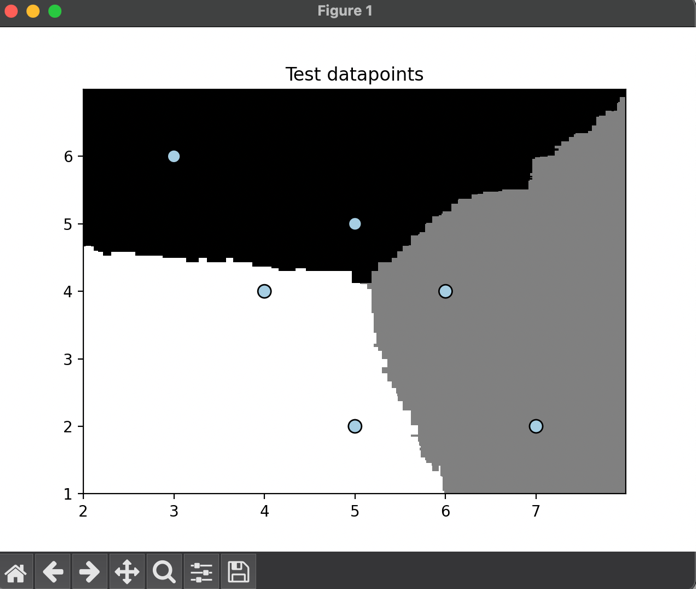
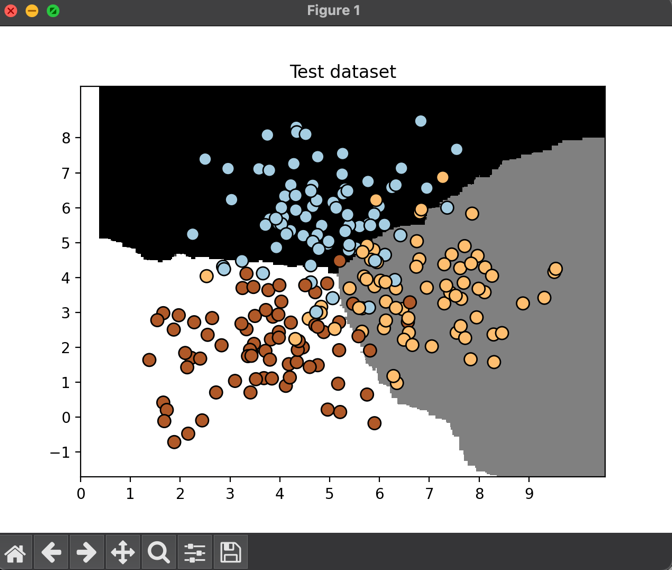
 

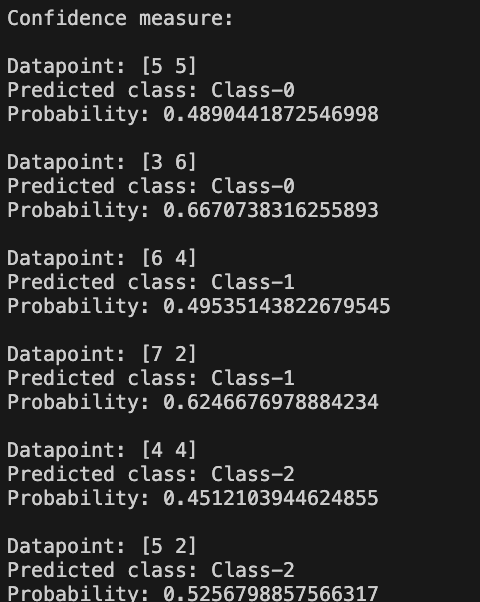
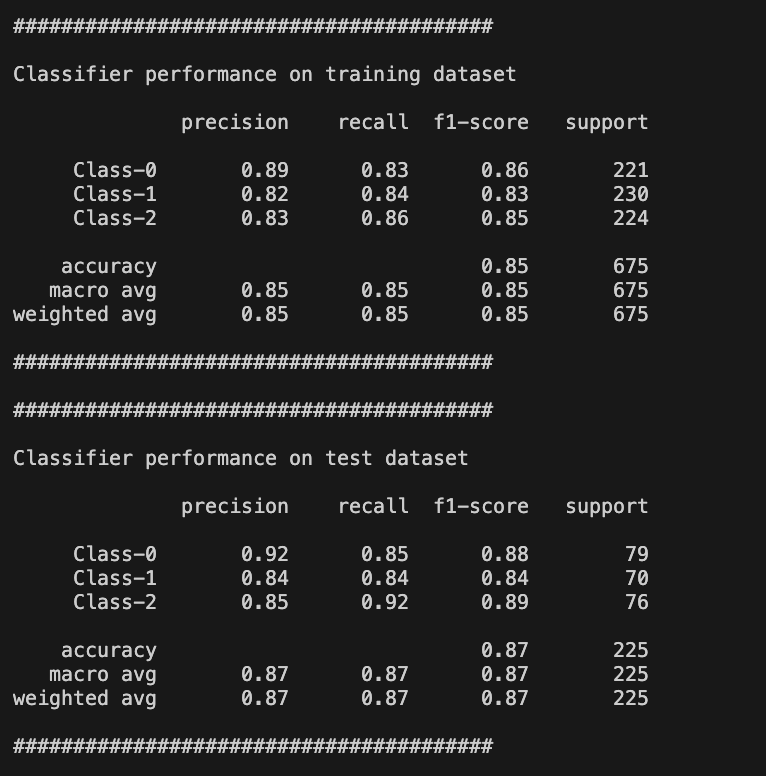




--classifier-type erf







RandomForestClassifier будує набір підвибірок за допомогою методу бутстрапу. Для кожної підвибірки створюється окреме дерево рішень. Для прогнозування класу об'єкта алгоритм RandomForestClassifier використовує голосування дерев, де кожне дерево визначає свій клас. Цей алгоритм має високу точність, добре справляється з шумом у даних і є стійким до перенавчання.

ExtraTreesClassifier схожий на попередній алгоритм, але має кілька відмінностей. При вибірці ознак він використовує всі p ознак, а не випадкову підмножину з m ознак. Крім того, при виборі порогів розщеплення він випадково обирає поріг для кожної ознаки, замість того, щоб шукати й обирати найкращий поріг. Це робить алгоритм більш ефективним для класифікації даних з великою кількістю ознак.

**Завдання 2.** Обробка дисбалансу класів.

Лістинг програми:

import sys

import numpy as np

import matplotlib.pyplot as plt

from sklearn.ensemble import ExtraTreesClassifier

from sklearn import model\_selection as cross\_validation

from sklearn.metrics import classification\_report

from utilities import visualize\_classifier

input\_file = 'data/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')

X\_train, X\_test, y\_train, y\_test = cross\_validation.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 = {'n\_estimators': 100, 'max\_depth': 4, 'random\_state': 0, '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, 'Training dataset')

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

visualize\_classifier(classifier, X\_test, y\_test, 'Test dataset')

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")

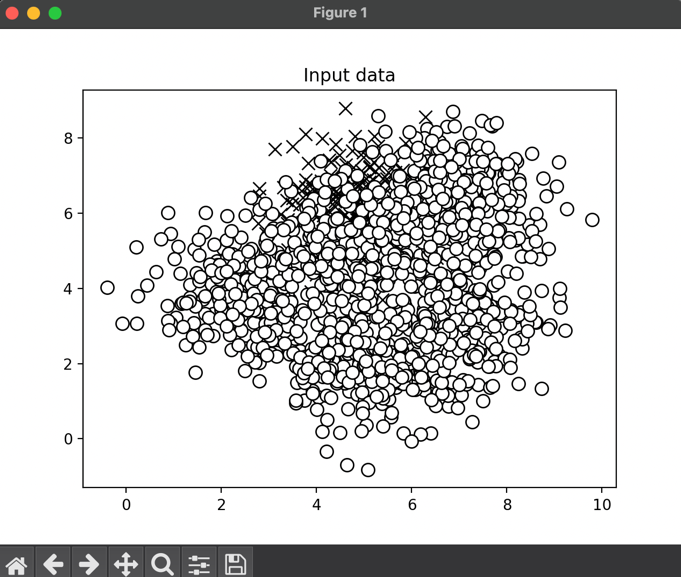
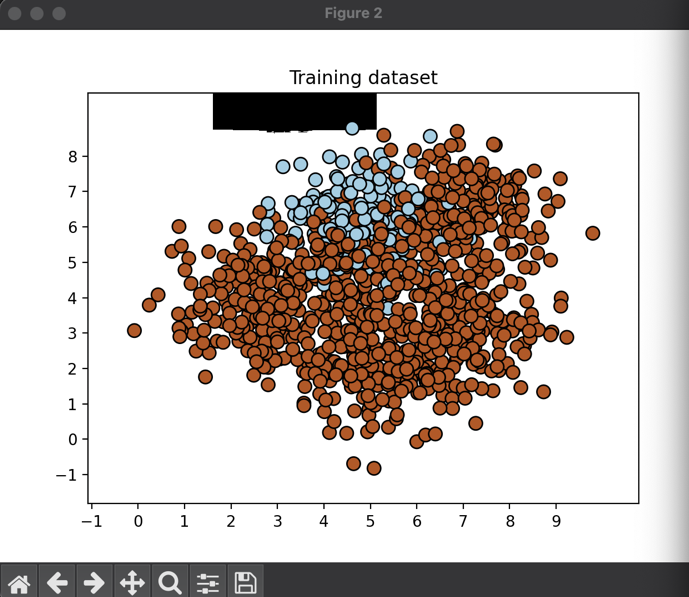
print(classification\_report(y\_test, y\_test\_pred,

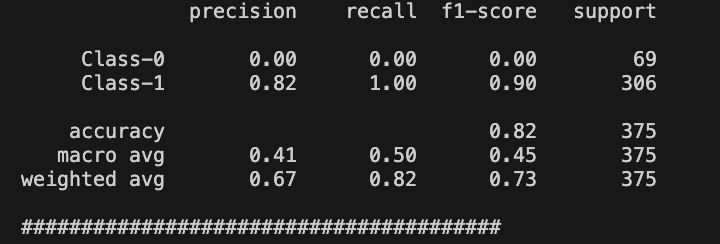
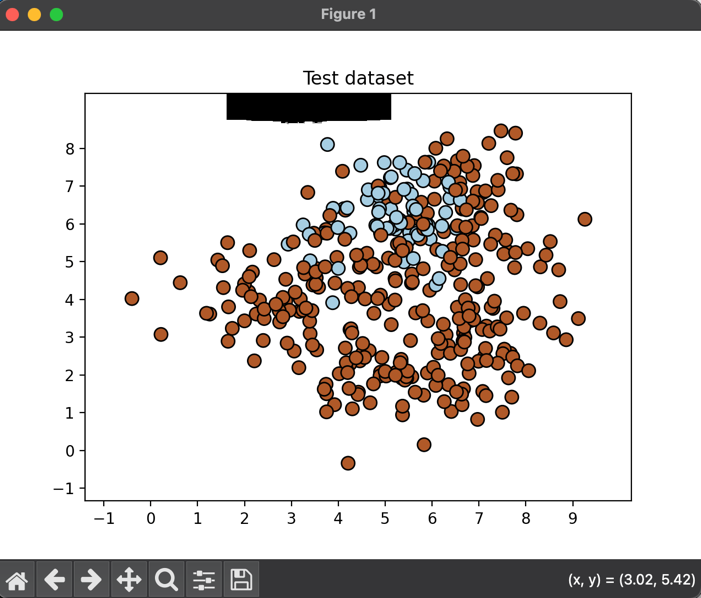
target\_names=class\_names))

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

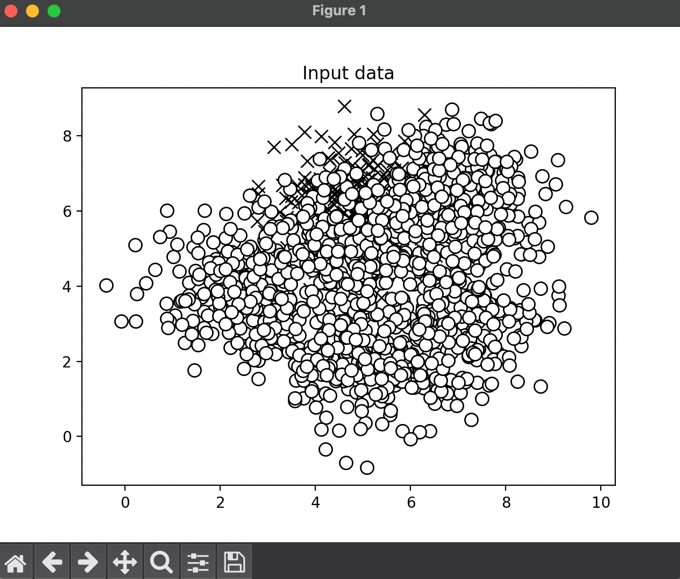
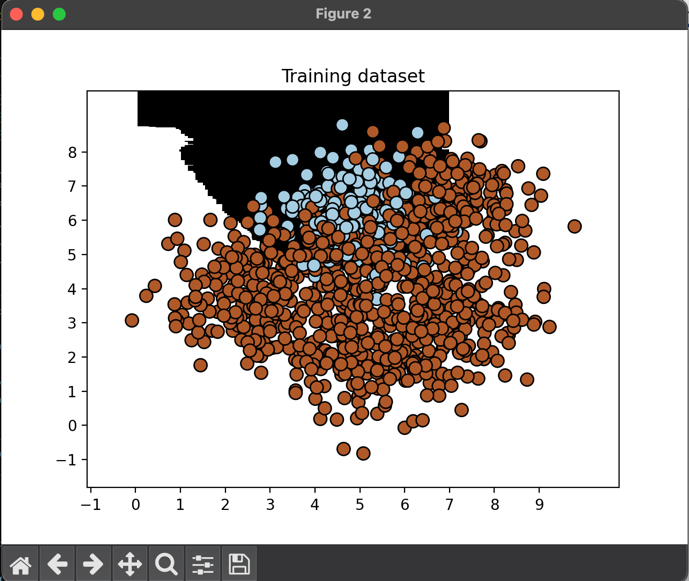
plt.show()

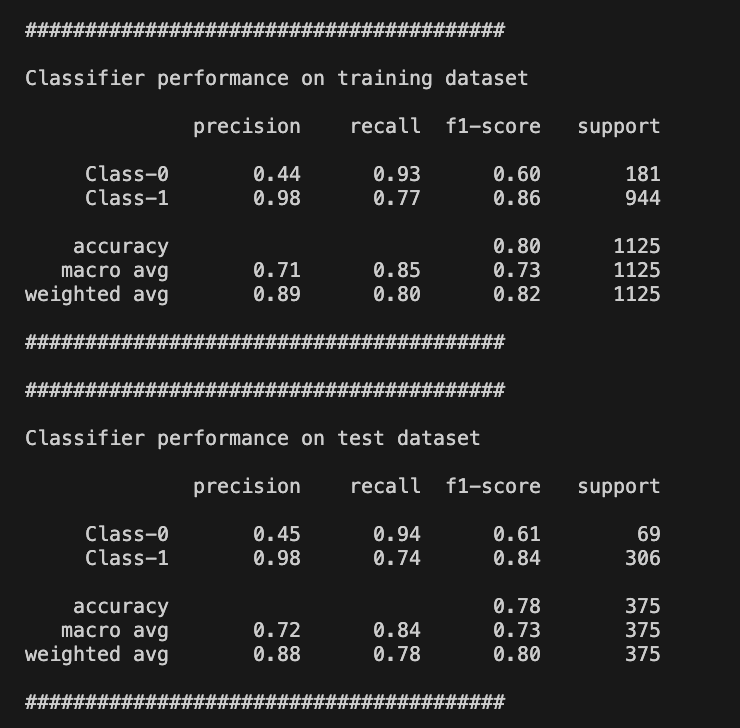
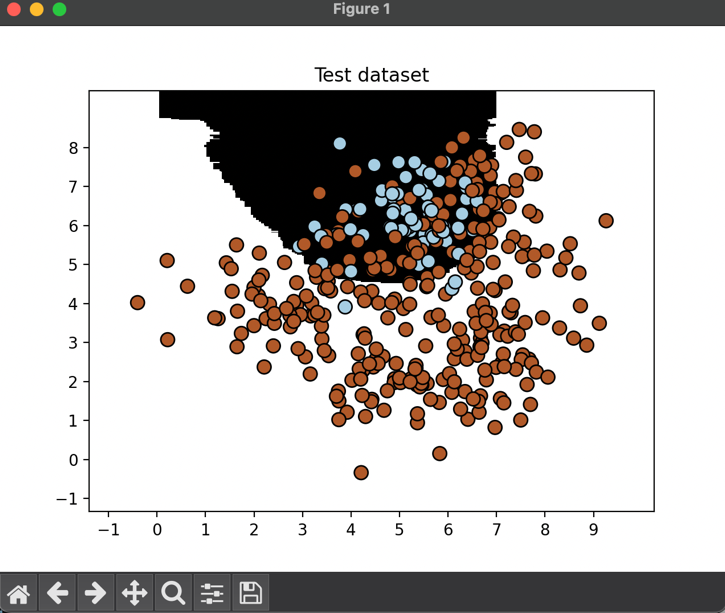
Результат виконання:





Balance:





При встановленні аргументу `class\_weight` в ExtraTreesClassifier на значення `balanced`, алгоритм автоматично розраховує ваги класів на основі їхньої кількості в даних. Це дозволяє компенсувати значні відмінності в кількості представників різних класів, що може покращити точність класифікації для менш представлених класів.

**Завдання 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 utilities import visualize\_classifier

input\_file = 'data/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 i in classifier.cv\_results\_['params']:

print(i, '-->', round(classifier.cv\_results\_['mean\_test\_score'][classifier.cv\_results\_['params'].index(i)], 3))

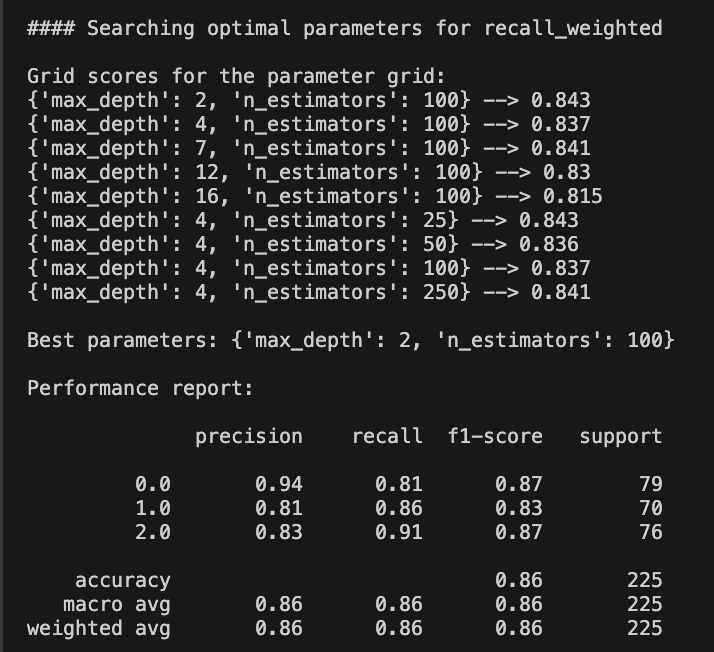
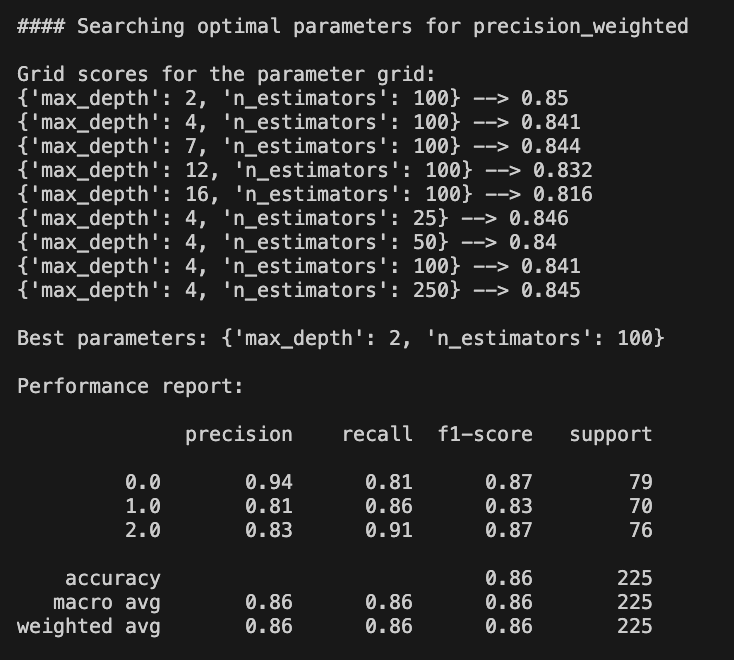
print("\nBest parameters:", classifier.best\_params\_)

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

print("\nPerformance report:\n")

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

Результат виконання:



використовується для автоматичного пошуку оптимальних параметрів моделі. Він здійснює систематичний пошук через задану сітку параметрів моделі, застосовуючи крос-валідацію для оцінки результатів і вибору найкращих значень параметрів.

**Завдання 4.** Обчислення відносної важливості ознак.

Лістинг програми:

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

from sklearn import preprocessing

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import AdaBoostClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_squared\_error, explained\_variance\_score

from sklearn.utils import shuffle

data\_url = "http://lib.stat.cmu.edu/datasets/boston"

raw\_df = pd.read\_csv(data\_url, sep="\s+", skiprows=22, header=None)

housing\_data = np.hstack([raw\_df.values[::2, :], raw\_df.values[1::2, :2]])

target = raw\_df.values[1::2, 2]

label\_encoder = preprocessing.LabelEncoder()

y = label\_encoder.fit\_transform(target)

X, y = shuffle(housing\_data, y, random\_state=7)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=7)

regressor = AdaBoostClassifier(DecisionTreeClassifier(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 = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT']

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')

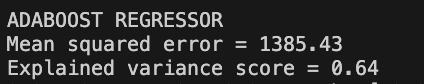
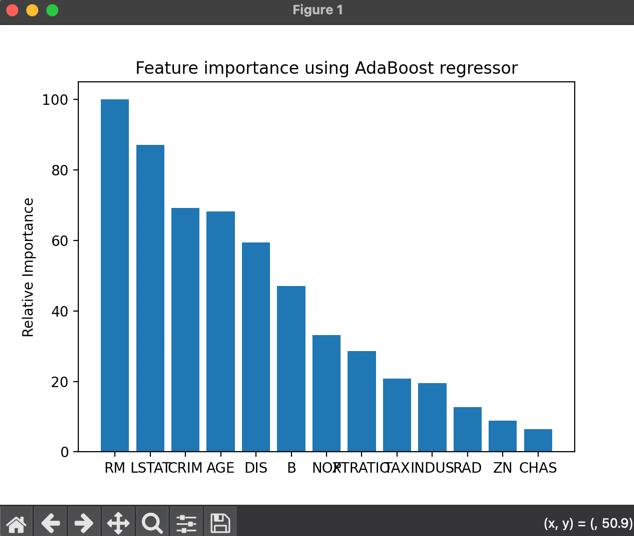
plt.xticks(pos, [feature\_names[i] for i in index\_sorted])

plt.ylabel('Relative Importance')

plt.title('Feature importance using AdaBoost regressor')

plt.show()

Результат виконання:



Діаграма показує величину впливу різних ознак на ціну житла (MEDV) в базі даних, що містить такі характеристики як рівень злочинності (CRIM), частка земель під житлову забудову (ZN), концентрація оксидів азоту (NOX), кількість кімнат на житло (RM) та інші. Відповідно до графіка, найбільший вплив на ціну житла мають ознаки CRIM, RM, PTRATIO та DIS. Водночас ознаки RAD, INDUS, TAX та CHAS мають дуже малий вплив і можуть бути проігноровані, оскільки їхня кореляція з ціною житла є незначною.

**Завдання 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 import preprocessing

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import classification\_report, mean\_absolute\_error

from sklearn.ensemble import ExtraTreesRegressor

input\_file = 'data/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([test\_datapoint[i]])[0])

count += 1

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

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

Результат виконання:

