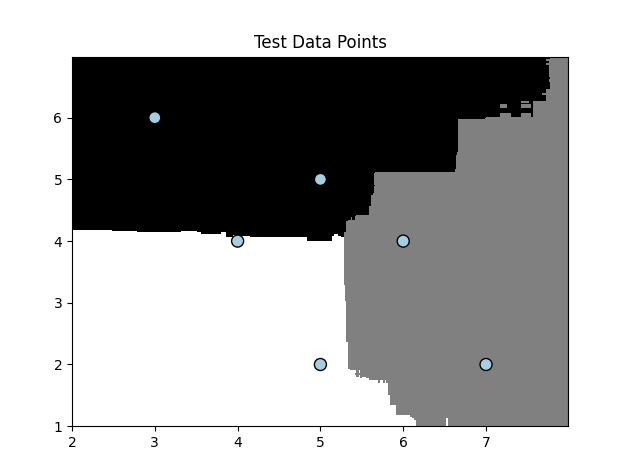
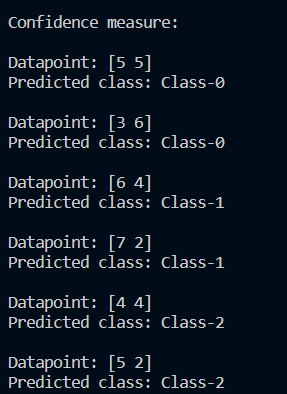
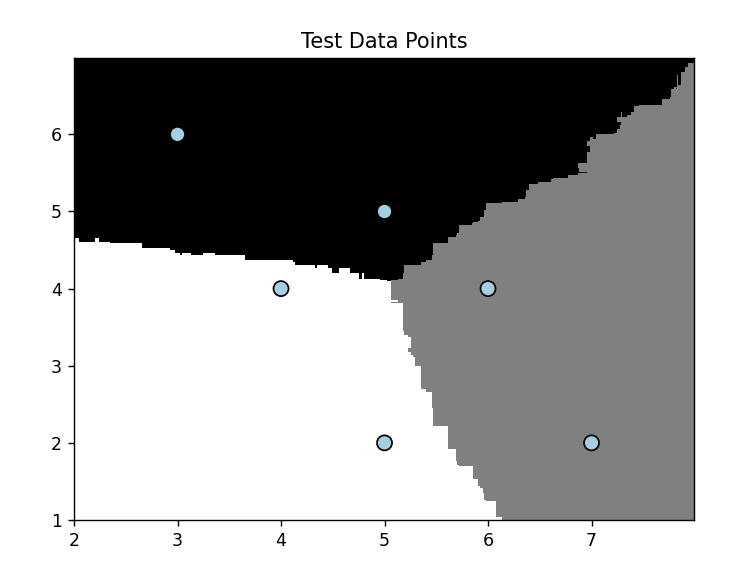


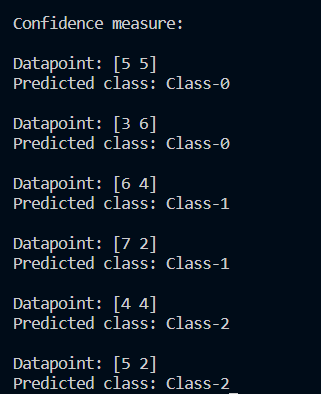
RF:





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

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 of classifier to use: 'rf' (Random Forest) or 'erf' (Extra Random Forest)")

return parser

*def* load\_data(*input\_file*):

data = np.loadtxt(*input\_file*, *delimiter*=',')

X, y = data[:, :-1], data[:, -1]

return X, y

*def* main():

# Parsing arguments

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

classifier\_type = args.classifier\_type

# Load dataset

input\_file = 'data\_random\_forests.txt'

X, y = load\_data(input\_file)

# Input data visualization

plt.figure()

markers = ['s', 'o', '\*']

for i, label in enumerate(np.unique(y)):

class\_data = X[y == label]

plt.scatter(class\_data[:, 0], class\_data[:, 1], *s*=75, *facecolors*='white',

*edgecolors*='black', *linewidth*=1, *marker*=markers[i], *label*=*f*'Class-{int(label)}')

plt.title('Input Data')

plt.legend()

plt.show()

# Splitting the dataset

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

# Classifier initialization

params = {'n\_estimators': 100, 'max\_depth': 4, 'random\_state': 0}

if classifier\_type == 'rf':

classifier = RandomForestClassifier(\*\*params)

elif classifier\_type == 'erf':

classifier = ExtraTreesClassifier(\*\*params)

# Training the classifier

classifier.fit(X\_train, y\_train)

# Training dataset visualization

visualize\_classifier(classifier, X\_train, y\_train, 'Training Dataset')

# Test dataset visualization

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

# Training performance

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

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

print(classification\_report(y\_train, classifier.predict(X\_train), *target\_names*=['Class-0', 'Class-1', 'Class-2']))

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

# Test performance

print("#" \* 40)

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

print(classification\_report(y\_test, classifier.predict(X\_test), *target\_names*=['Class-0', 'Class-1', 'Class-2']))

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

# Test data points visualization

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 = *f*"Class-{np.argmax(probabilities)}"

print(*f*"\nDatapoint: {datapoint}")

print(*f*"Predicted class: {predicted\_class}")

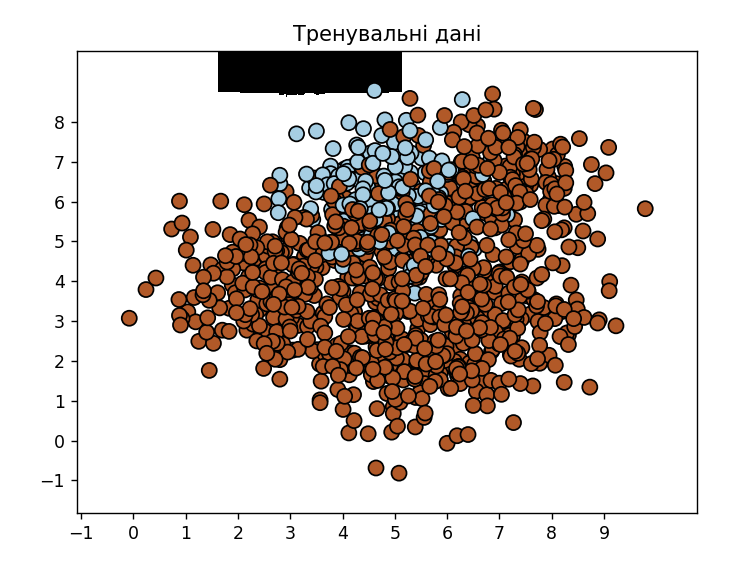
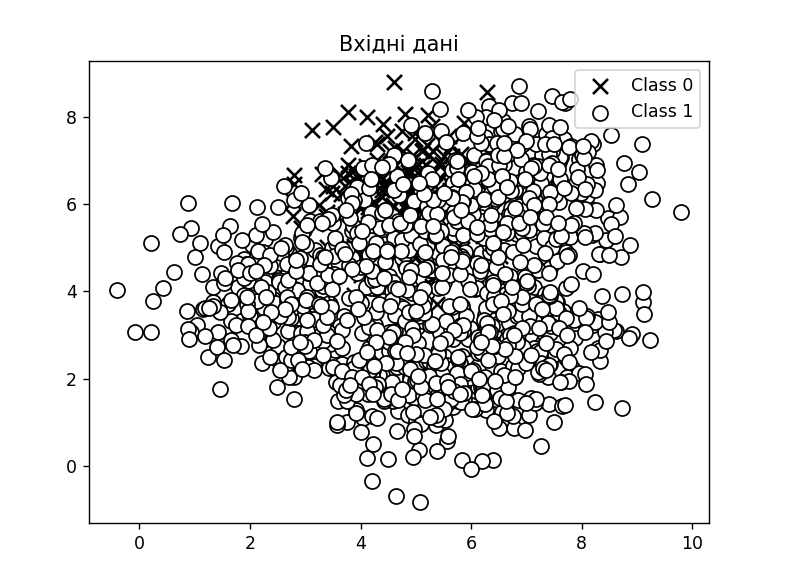
visualize\_classifier(classifier, test\_datapoints, [0] \* len(test\_datapoints), 'Test Data Points')

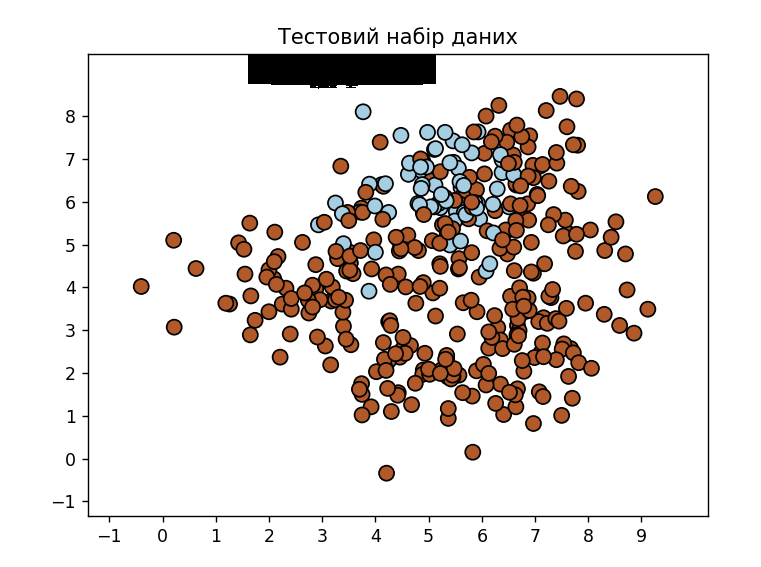
plt.show()

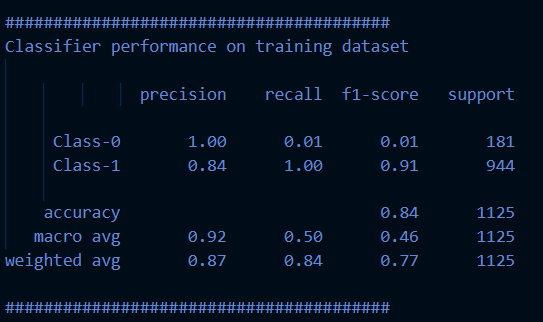
if \_\_name\_\_ == '\_\_main\_\_':

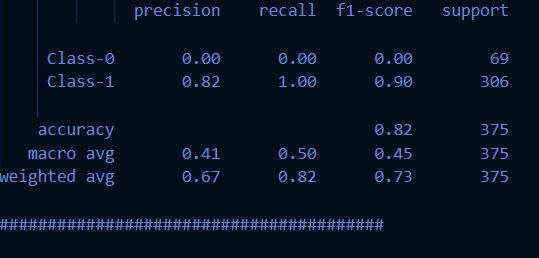
main()

Завдання 2.2. Обробка дисбалансу класів

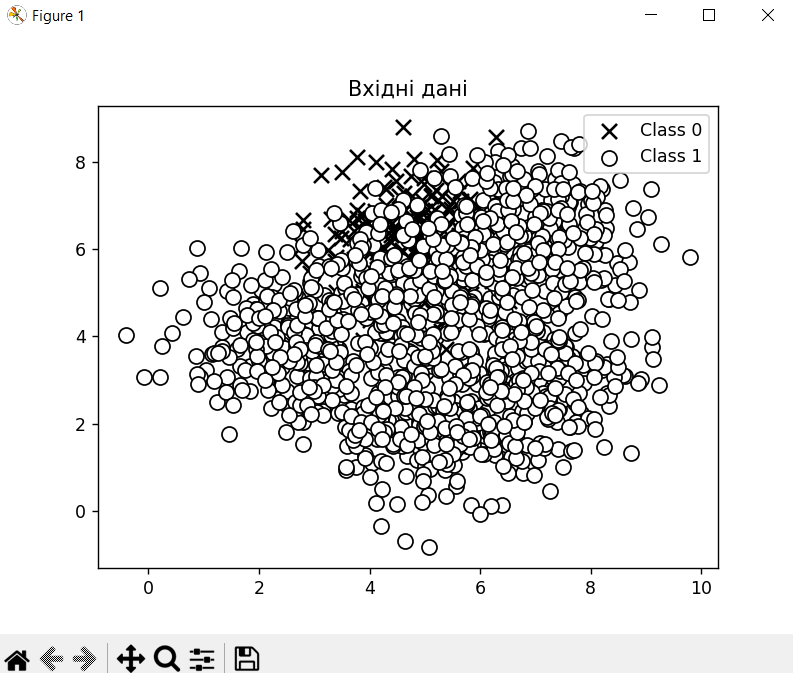


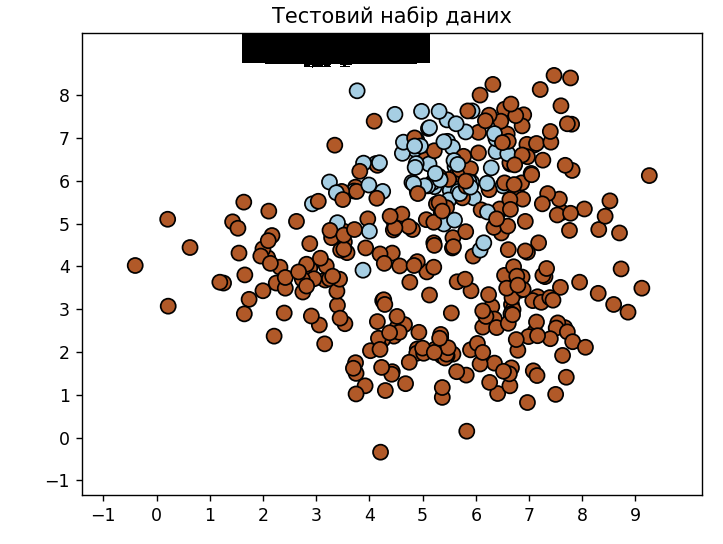
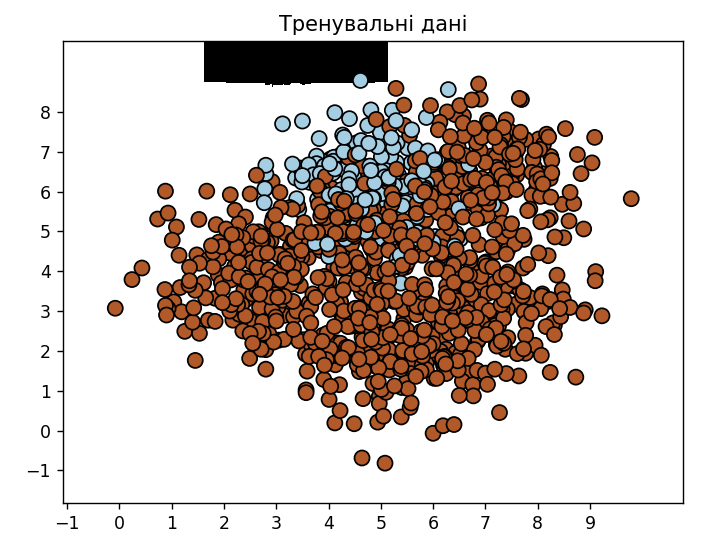


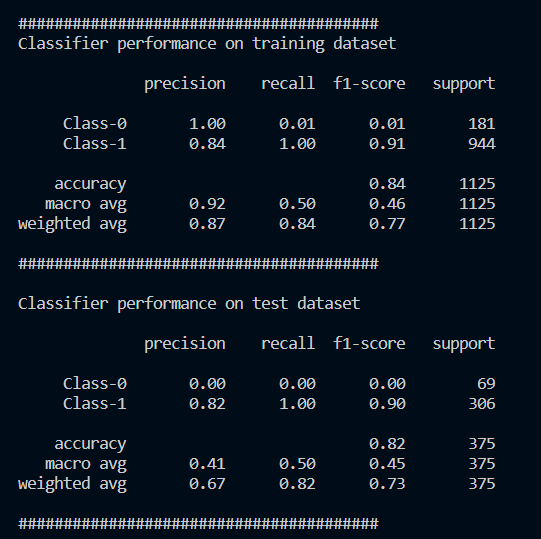




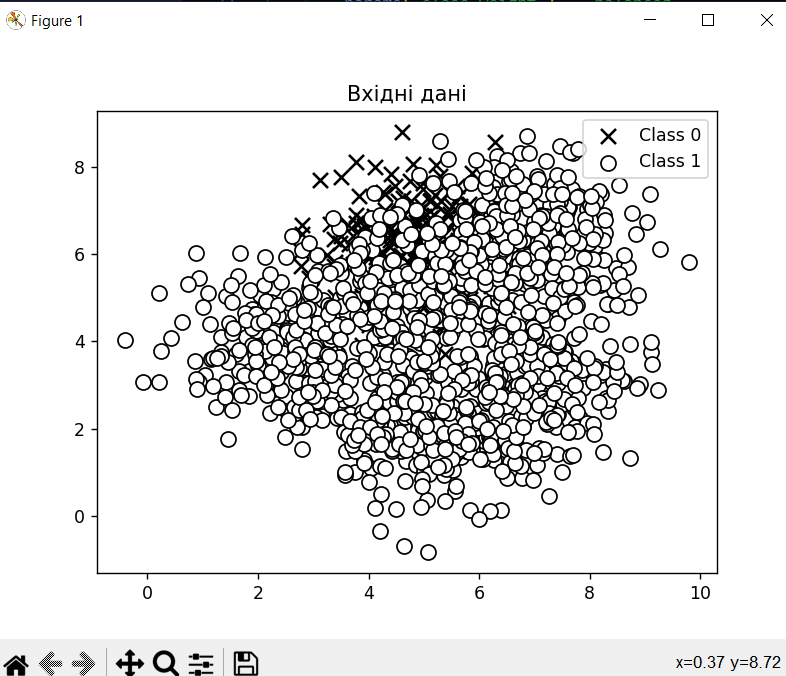
python3 –W ignore class\_imbalence.py:

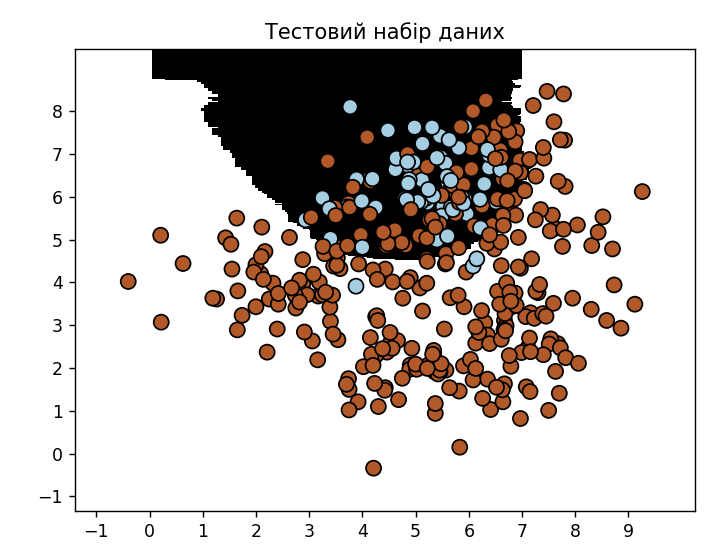
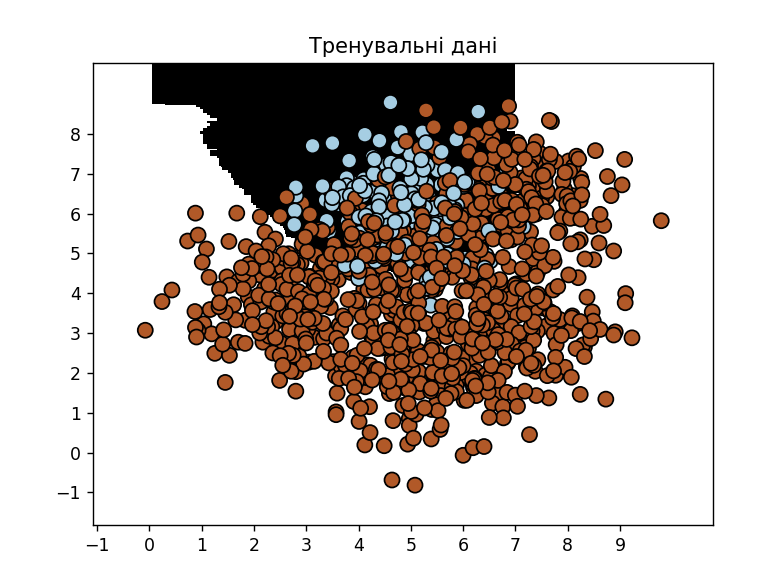


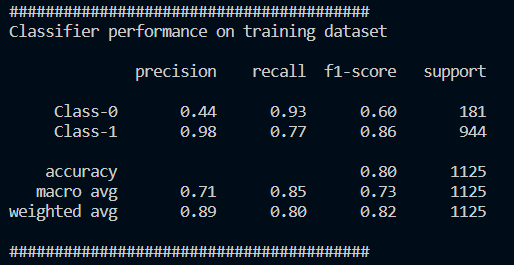


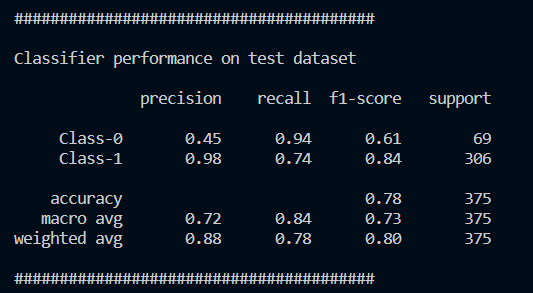


python3 class\_imbalence.py balance:









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

*def* load\_data(*input\_file*):

data = np.loadtxt(*input\_file*, *delimiter*=',')

X, y = data[:, :-1], data[:, -1]

return X, y

*def* plot\_data(*X*, *y*):

class\_0 = *X*[*y* == 0]

class\_1 = *X*[*y* == 1]

plt.figure()

plt.scatter(class\_0[:, 0], class\_0[:, 1], *s*=75, *c*='black', *marker*='x', *label*='Class 0')

plt.scatter(class\_1[:, 0], class\_1[:, 1], *s*=75, *c*='white', *edgecolors*='black', *marker*='o', *label*='Class 1')

plt.title("Вхідні дані")

plt.legend()

plt.show()

*def* train\_and\_evaluate(*X*, *y*, *balance*=False):

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(*X*, *y*, *test\_size*=0.25, *random\_state*=5)

# Define classifier parameters

params = {'n\_estimators': 100, 'max\_depth': 4, 'random\_state': 0}

if *balance*:

params['class\_weight'] = 'balanced'

# Train the classifier

classifier = ExtraTreesClassifier(\*\*params)

classifier.fit(X\_train, y\_train)

# Visualize the training and testing results

visualize\_classifier(classifier, X\_train, y\_train, 'Тренувальні дані')

visualize\_classifier(classifier, X\_test, y\_test, 'Тестовий набір даних')

# Evaluate classifier performance

class\_names = ['Class-0', 'Class-1']

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")

*def* main():

input\_file = 'data\_imbalance.txt'

X, y = load\_data(input\_file)

# Plot the input data

plot\_data(X, y)

# Check for optional argument

balance = len(sys.argv) > 1 and sys.argv[1] == 'balance'

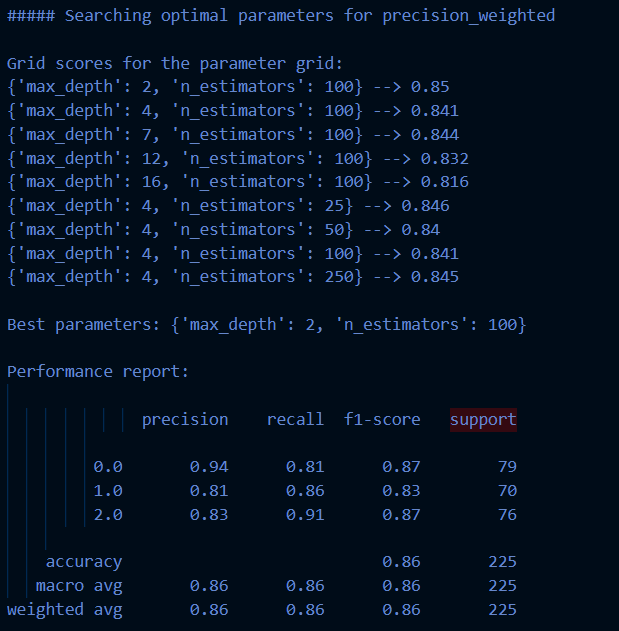
# Train and evaluate the classifier

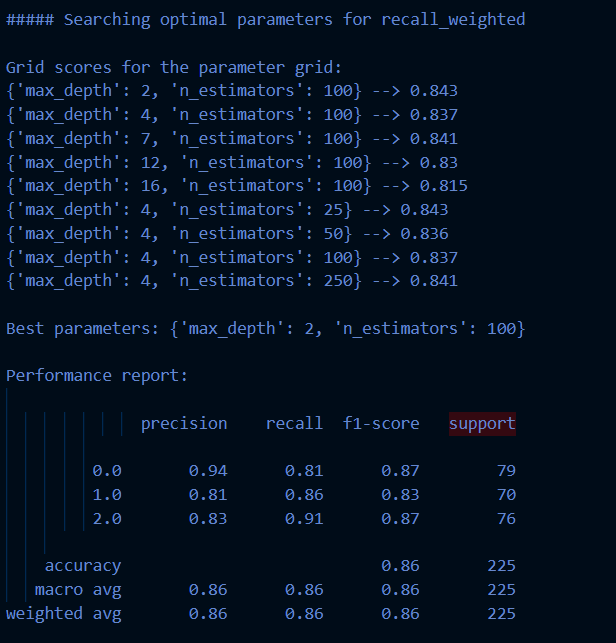
train\_and\_evaluate(X, y, *balance*=balance)

if \_\_name\_\_ == "\_\_main\_\_":

main()

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\_)

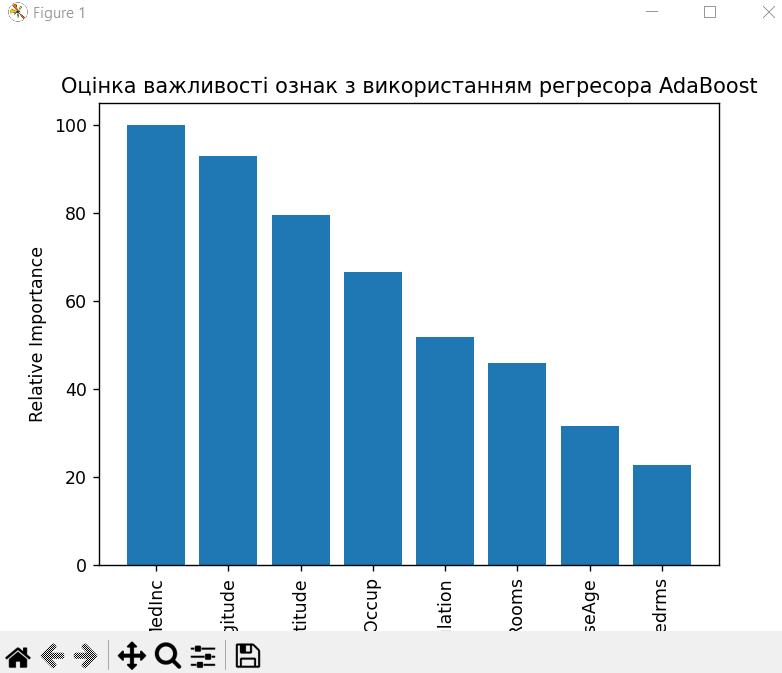
y\_pred = classifier.predict(X\_test)

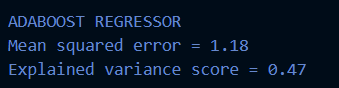
print("\nPerformance report: \n")

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

Завдання 2.4. Обчислення відносної важливості ознак.

load\_boston було видалено з нових версій scikit-learn, тому було викорстано: from sklearn.datasets import fetch\_california\_housing





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

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:

items = line.strip().split(',')

data.append(items)

data = np.array(data)

label\_encoders = []

X\_encoded = np.empty\_like(data, *dtype*=float)

for i, value in enumerate(data[0]):

if value.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)

mae = mean\_absolute\_error(y\_test, y\_pred)

print(*f*"Mean absolute error: {mae*:.2f*}")

test\_datapoint = ['Saturday', '10:20', 'Atlanta', 'no']

test\_datapoint\_encoded = []

for i, value in enumerate(test\_datapoint):

if value.isdigit():

test\_datapoint\_encoded.append(int(value))

else:

encoder = label\_encoders.pop(0)

test\_datapoint\_encoded.append(encoder.transform([value])[0])

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

predicted\_traffic = regressor.predict([test\_datapoint\_encoded])[0]

print(*f*"Predicted traffic: {int(predicted\_traffic)}")

Git: <https://github.com/PavlenkoOks/AI>