Лабораторна робота №5

Ресурси Keras. TensorFlow. Навчання лінійної регресії

Виконав: ІП3-21-3 Осипчук Антон Олексійович

Github: https://github.com/AntonOsypchuk1/ai_lab/tree/main/lab5

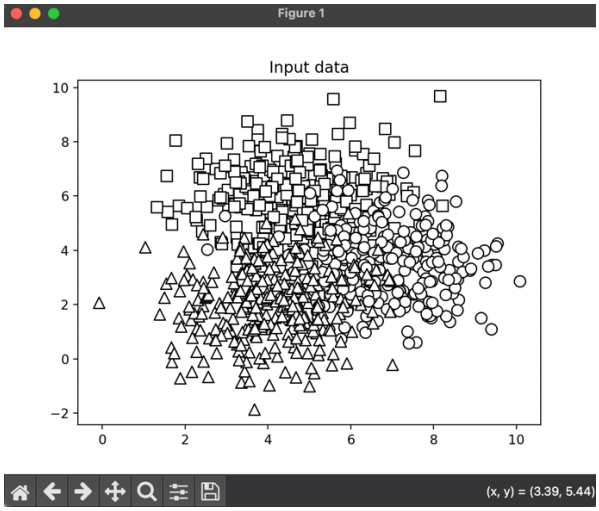
Завдання 1. Створення класифікаторів на основі випадкових та гранично випадкових лісів.
Лістинг програми:

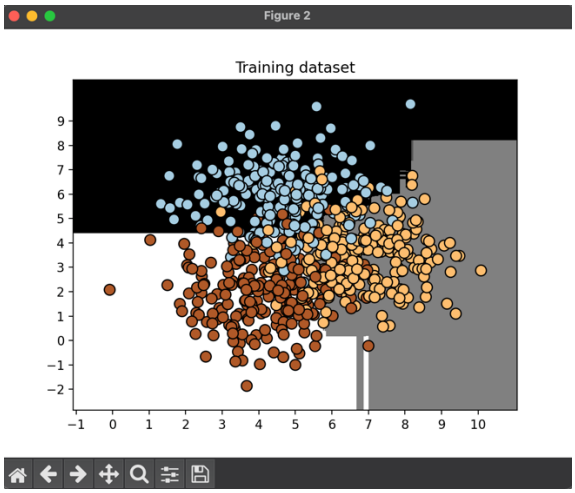
```
import argparse
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier
from sklearn.metrics import classification_report
from 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')
```

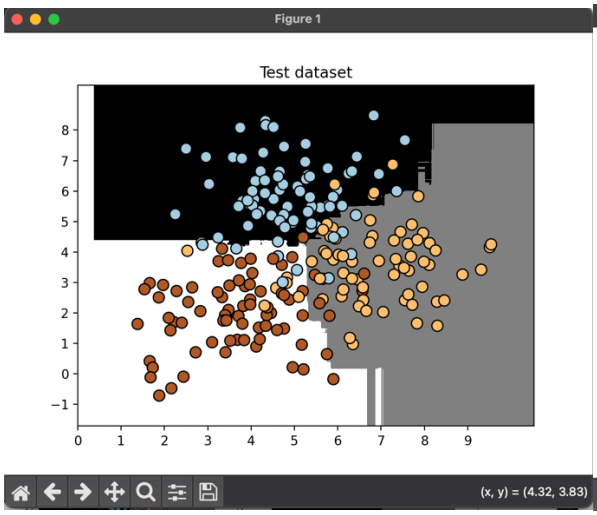
```
    plt.scatter(class_2[:, 0], class_2[:, 1], s=75, facecolors='white',
edgecolors='black', linewidth=1, marker='^')
    plt.title('Input data')

    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()
```

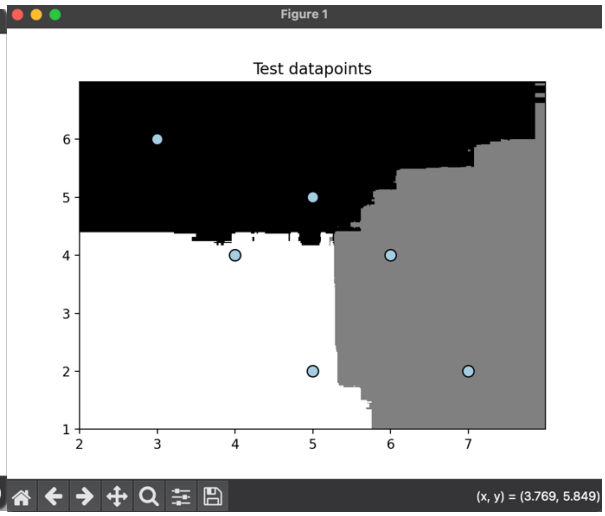
Результат виконання:

--classifier-type rf





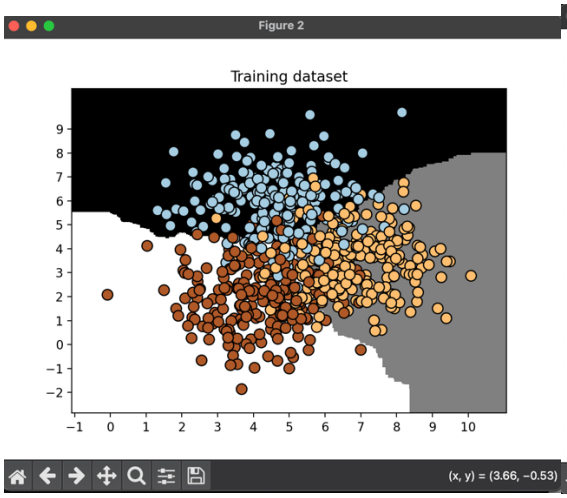


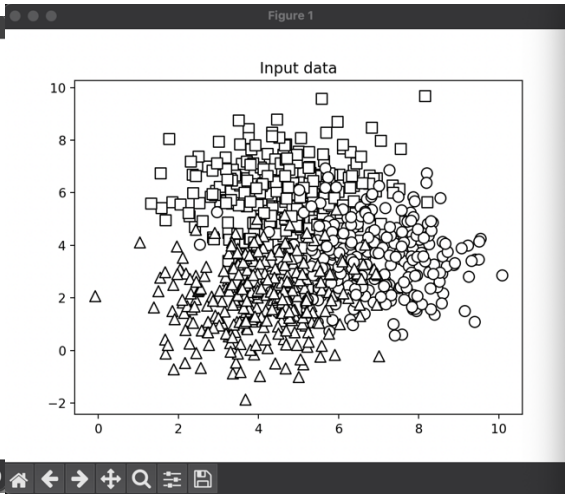


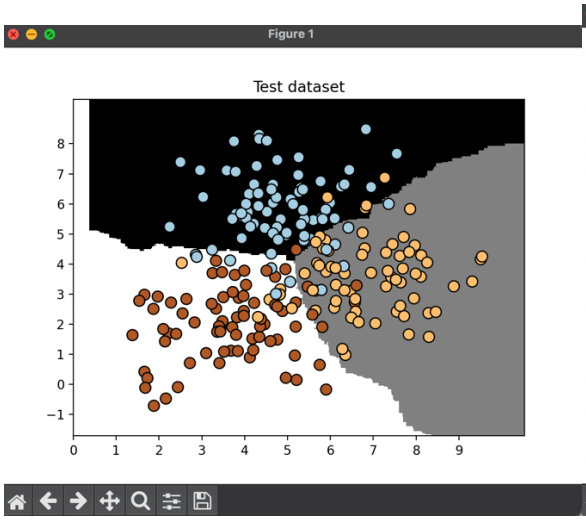

```
########################################

Classifier performance on training dataset

              precision    recall  f1-score   support

     Class-0       0.91      0.86      0.88       221
     Class-1       0.84      0.87      0.86       230
     Class-2       0.86      0.87      0.86       224

    accuracy                           0.87       675
   macro avg       0.87      0.87      0.87       675
weighted avg       0.87      0.87      0.87       675

########################################

########################################

Classifier performance on test dataset

              precision    recall  f1-score   support

     Class-0       0.92      0.85      0.88        79
     Class-1       0.86      0.84      0.85        70
     Class-2       0.84      0.92      0.88        76

    accuracy                           0.87       225
   macro avg       0.87      0.87      0.87       225
weighted avg       0.87      0.87      0.87       225

########################################
```

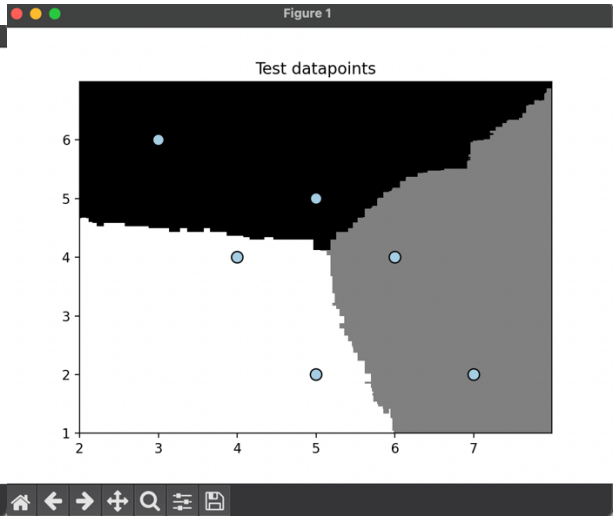
```
Confidence measure:

Datapoint: [5 5]
Predicted class: Class-0
Probability: 0.814275318675482

Datapoint: [3 6]
Predicted class: Class-0
Probability: 0.9357445782050678

Datapoint: [6 4]
Predicted class: Class-1
Probability: 0.7451078021772837

Datapoint: [7 2]
Predicted class: Class-1
Probability: 0.7066022643054627

Datapoint: [4 4]
Predicted class: Class-2
Probability: 0.6388176473387874

Datapoint: [5 2]
Predicted class: Class-2
Probability: 0.8528526682816628
```

--classifier-type erf










```
#####################################

Classifier performance on training dataset

              precision    recall  f1-score   support

     Class-0       0.89      0.83      0.86       221
     Class-1       0.82      0.84      0.83       230
     Class-2       0.83      0.86      0.85       224

    accuracy                           0.85       675
   macro avg       0.85      0.85      0.85       675
weighted avg       0.85      0.85      0.85       675

#####################################

#####################################

Classifier performance on test dataset

              precision    recall  f1-score   support

     Class-0       0.92      0.85      0.88        79
     Class-1       0.84      0.84      0.84        70
     Class-2       0.85      0.92      0.89        76

    accuracy                           0.87       225
   macro avg       0.87      0.87      0.87       225
weighted avg       0.87      0.87      0.87       225

#####################################
```

```
Confidence measure:

Datapoint: [5 5]
Predicted class: Class-0
Probability: 0.4890441872546998

Datapoint: [3 6]
Predicted class: Class-0
Probability: 0.6670738316255893

Datapoint: [6 4]
Predicted class: Class-1
Probability: 0.49535143822679545

Datapoint: [7 2]
Predicted class: Class-1
Probability: 0.6246676978884234

Datapoint: [4 4]
Predicted class: Class-2
Probability: 0.4512103944624855

Datapoint: [5 2]
Predicted class: Class-2
Probability: 0.5256798857566317
```

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.

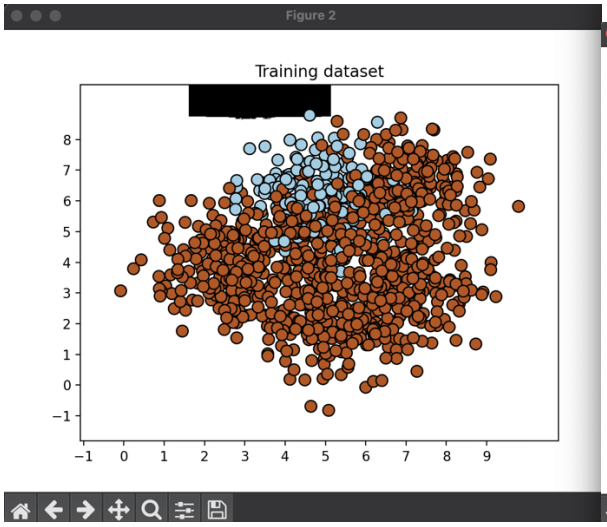
ExtraTreesClassifier схожий на попередній алгоритм, але має кілька відмінностей. При вибірці ознак він використовує всі p ознак, а не випадкову підмножину з m ознак. Крім того, при виборі порогів розщеплення він випадково обирає поріг для кожної ознаки, замість того, щоб шукати й обирати найкращий поріг.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

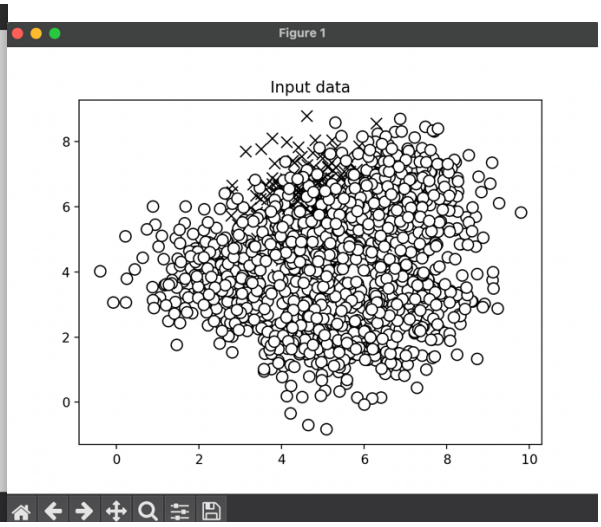
Завдання 2. Обробка дисбалансу класів.
Лістинг програми:

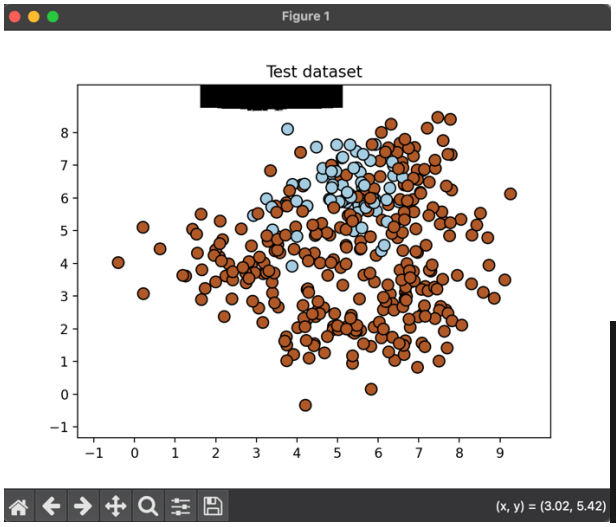
```
import sys
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import ExtraTreesClassifier
from sklearn import model_selection as cross_validation
from sklearn.metrics import classification_report
from utilities import visualize_classifier

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:

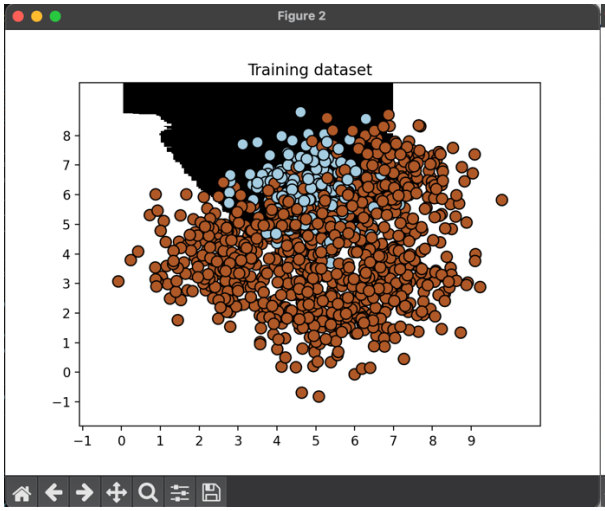


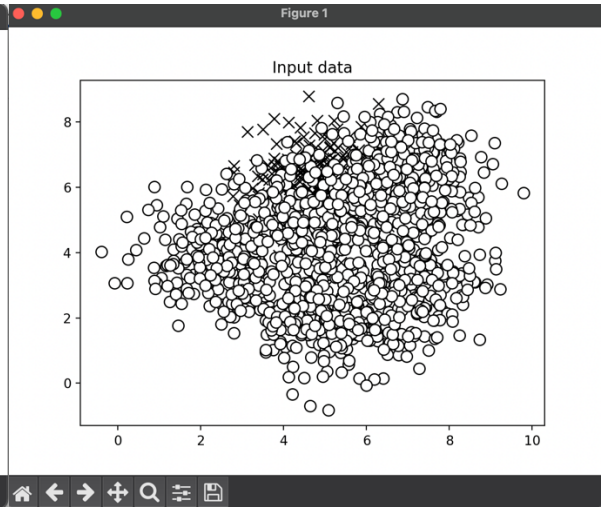


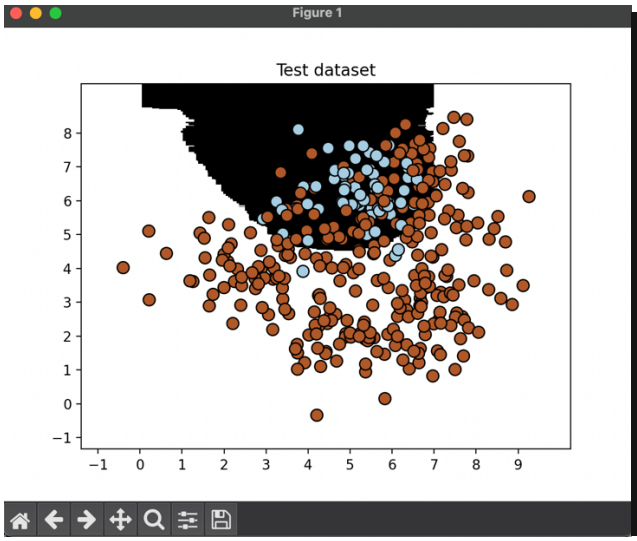



```
                precision    recall  f1-score   support

       Class-0       0.00      0.00      0.00        69
       Class-1       0.82      1.00      0.90       306

      accuracy                           0.82       375
     macro avg       0.41      0.50      0.45       375
  weighted avg       0.67      0.82      0.73       375

########################################
```

Balance:








```
#####################################

Classifier performance on training dataset

              precision    recall  f1-score   support

     Class-0       0.44      0.93      0.60       181
     Class-1       0.98      0.77      0.86       944

    accuracy                           0.80      1125
   macro avg       0.71      0.85      0.73      1125
weighted avg       0.89      0.80      0.82      1125

#####################################

#####################################

Classifier performance on test dataset

              precision    recall  f1-score   support

     Class-0       0.45      0.94      0.61        69
     Class-1       0.98      0.74      0.84       306

    accuracy                           0.78       375
   macro avg       0.72      0.84      0.73       375
weighted avg       0.88      0.78      0.80       375

#####################################
```

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:

```python
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_['pa
rams'].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:

```
#### Searching optimal parameters for precision_weighted

Grid scores for the parameter grid:
{'max_depth': 2, 'n_estimators': 100} --> 0.85
{'max_depth': 4, 'n_estimators': 100} --> 0.841
{'max_depth': 7, 'n_estimators': 100} --> 0.844
{'max_depth': 12, 'n_estimators': 100} --> 0.832
{'max_depth': 16, 'n_estimators': 100} --> 0.816
{'max_depth': 4, 'n_estimators': 25} --> 0.846
{'max_depth': 4, 'n_estimators': 50} --> 0.84
{'max_depth': 4, 'n_estimators': 100} --> 0.841
{'max_depth': 4, 'n_estimators': 250} --> 0.845

Best parameters: {'max_depth': 2, 'n_estimators': 100}

Performance report:

              precision    recall  f1-score   support

         0.0       0.94      0.81      0.87        79
         1.0       0.81      0.86      0.83        70
         2.0       0.83      0.91      0.87        76

    accuracy                           0.86       225
   macro avg       0.86      0.86      0.86       225
weighted avg       0.86      0.86      0.86       225
```

```
#### Searching optimal parameters for recall_weighted

Grid scores for the parameter grid:
{'max_depth': 2, 'n_estimators': 100} --> 0.843
{'max_depth': 4, 'n_estimators': 100} --> 0.837
{'max_depth': 7, 'n_estimators': 100} --> 0.841
{'max_depth': 12, 'n_estimators': 100} --> 0.83
{'max_depth': 16, 'n_estimators': 100} --> 0.815
{'max_depth': 4, 'n_estimators': 25} --> 0.843
{'max_depth': 4, 'n_estimators': 50} --> 0.836
{'max_depth': 4, 'n_estimators': 100} --> 0.837
{'max_depth': 4, 'n_estimators': 250} --> 0.841

Best parameters: {'max_depth': 2, 'n_estimators': 100}

Performance report:

              precision    recall  f1-score   support

         0.0       0.94      0.81      0.87        79
         1.0       0.81      0.86      0.83        70
         2.0       0.83      0.91      0.87        76

    accuracy                           0.86       225
   macro avg       0.86      0.86      0.86       225
weighted avg       0.86      0.86      0.86       225
```

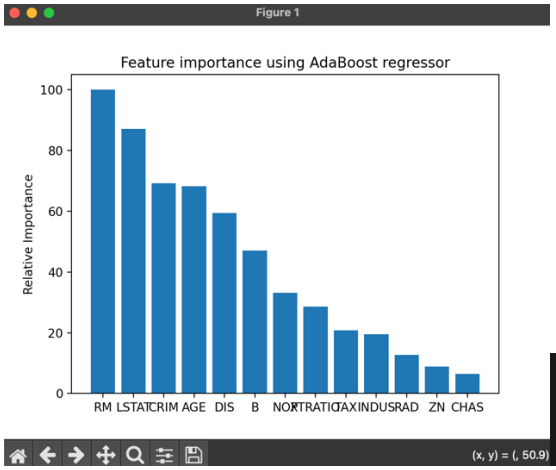
використовується для автоматичного пошуку оптимальних параметрів моделі. Він здійснює систематичний пошук через задану сітку параметрів моделі, застосовуючи крос-валідацію для оцінки результатів і вибору найкращих значень параметрів.

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:




```
ADABOOST REGRESSOR
Mean squared error = 1385.43
Explained variance score = 0.64
```

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:

```
Mean absolute error: 7.42
Predicted traffic: 26
```