ЛАБОРАТОРНА РОБОТА № 5

ДОСЛІДЖЕННЯ МЕТОДІВ АНСАМБЛЕВОГО НАВЧАННЯ

Мета роботи: 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.

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

from utilities import visualize_classifier

# 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

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()
```

					Державний університет "Житомирська політехніка"	
Змн.	Арк.	№ докум.	Підпис	Дата		
Розроб.		Дроботун Д. Я.			Звіт з практичної роботи	Літ. / Арк. 1 / Аркушів
Перевір.						
Керівник						
Н. контр.						Гр. ІПЗК-19-1
Зав. каф.						

```
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()
```

Рис. 1.1 Код програми

```
PS E:\ЖДУ\Python\Work_Drobotun_5> python LR_5_task_1.py --classifier-type rf
```

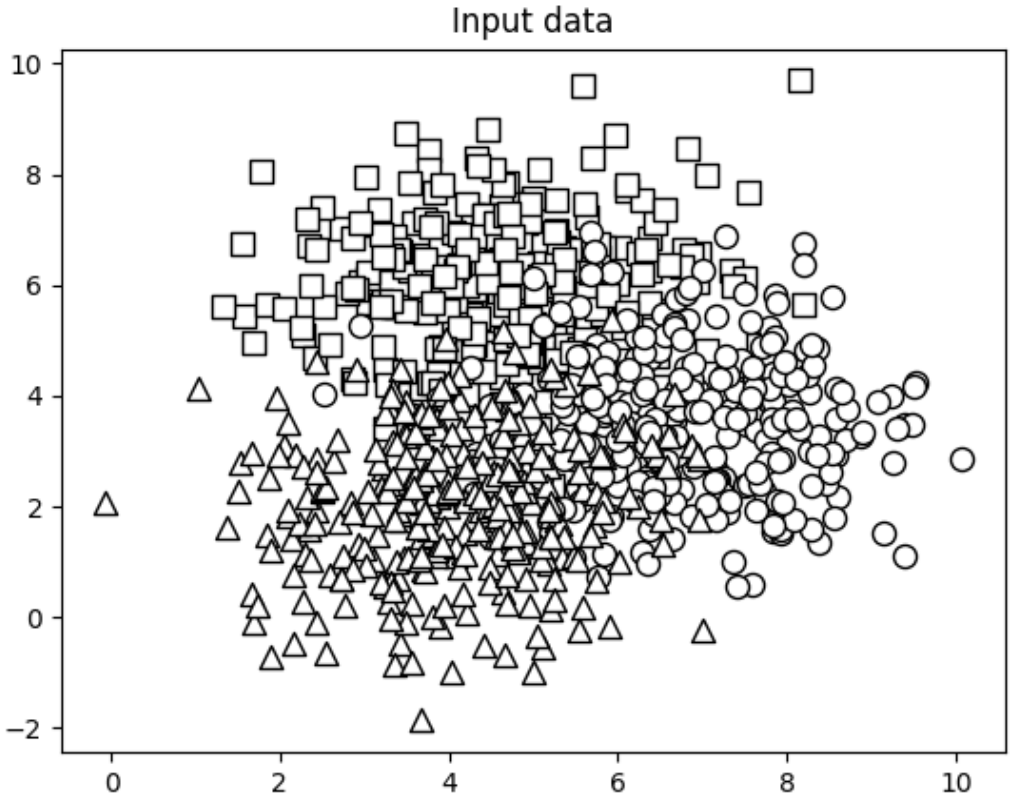

Рис. 1.2 Результат виконання програми

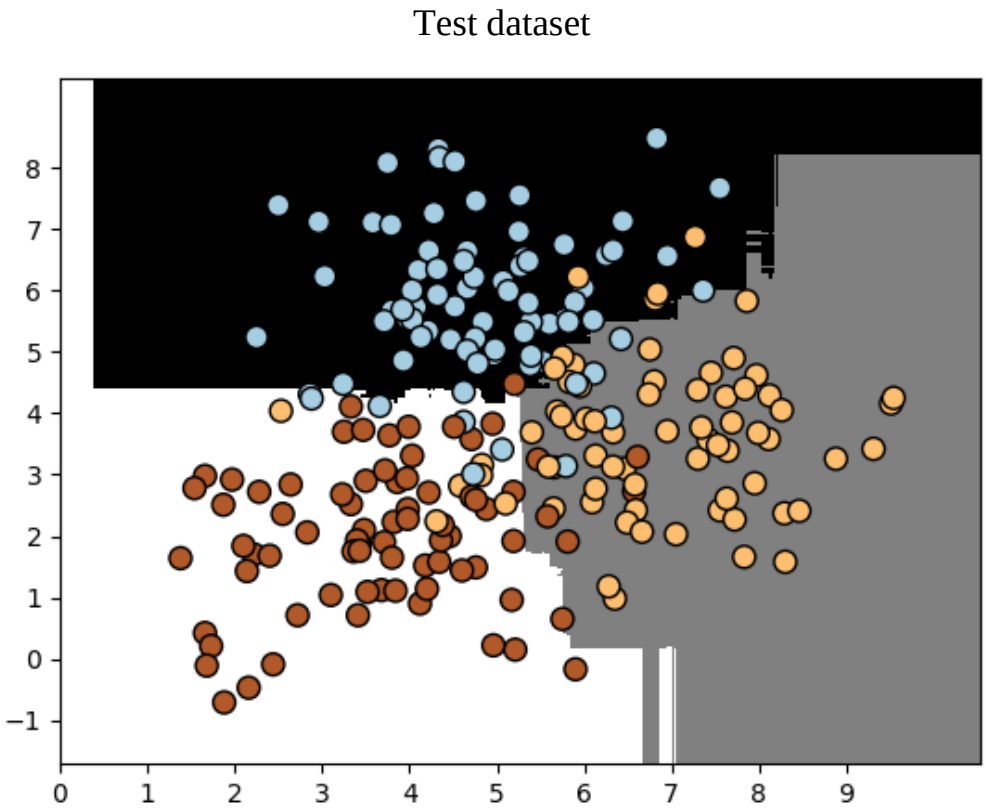

Рис. 1.3 Результат виконання програми

```
PS E:\ЖДУ\Python\Work_Drobotun_5> python LR_5_task_1.py --classifier-type erf
```

Test dataset

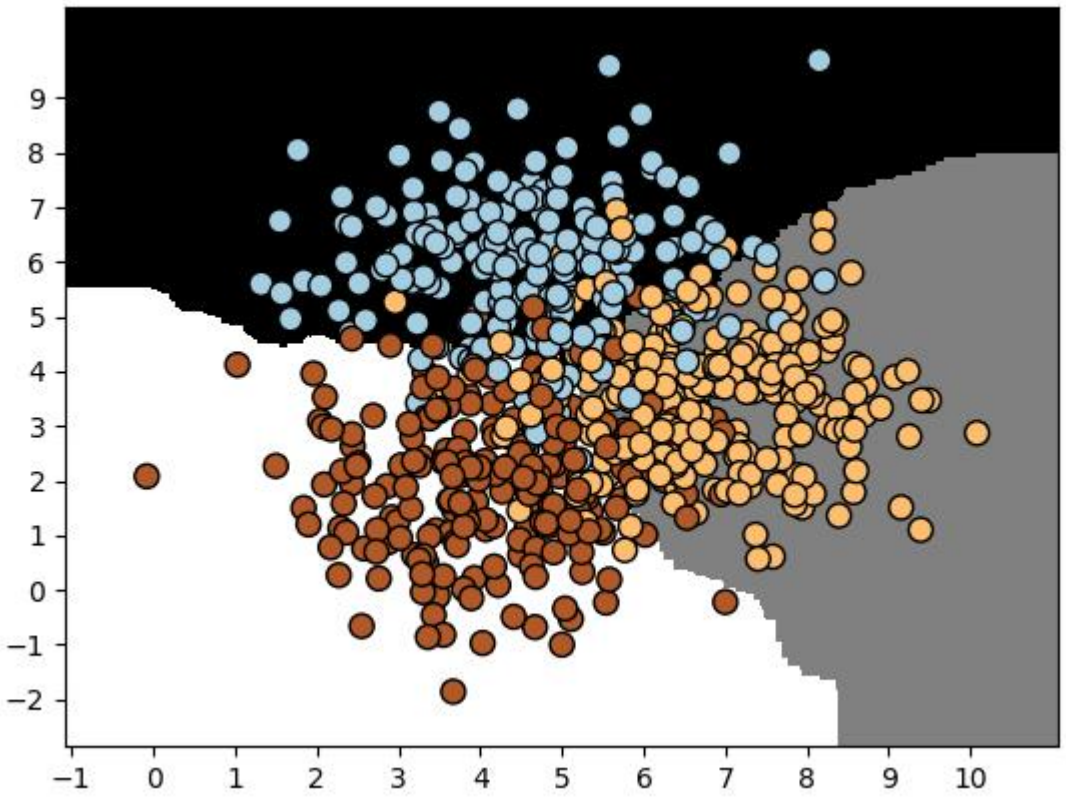

Рис. 1.4 Результат виконання програми

Завдання 2.2 Обробка дисбалансу класів

```
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('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 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()
```

Рис. 1.5 Код програми

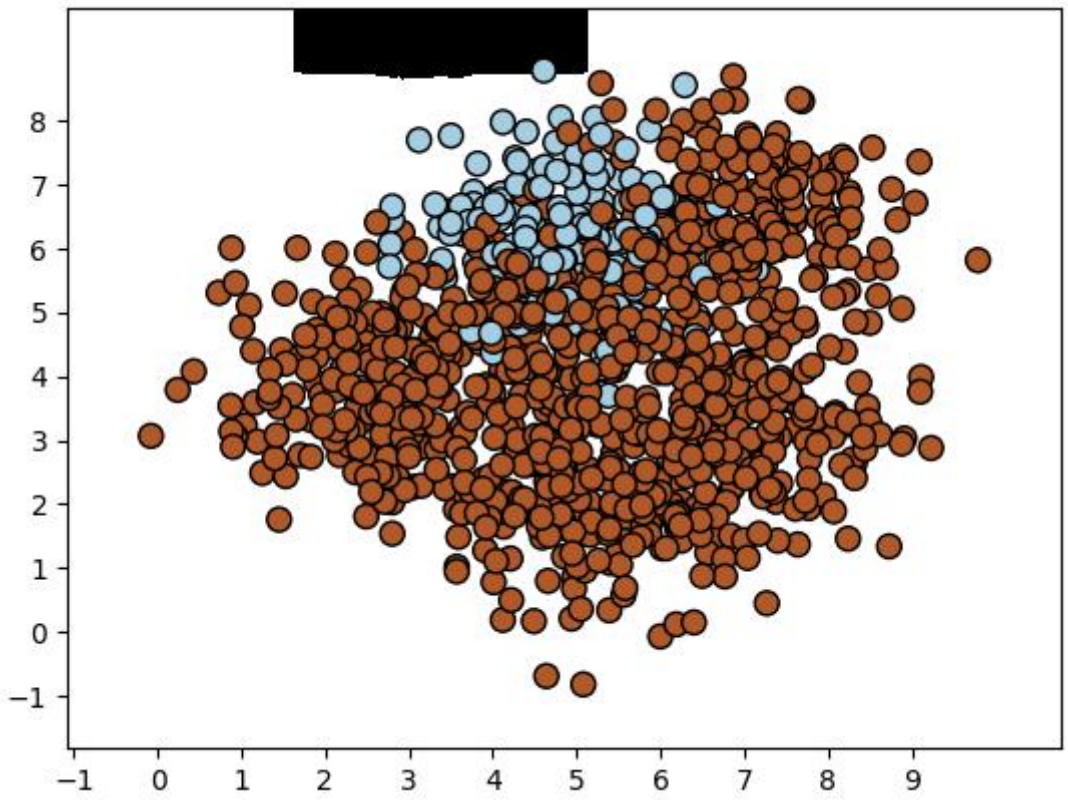

Рис. 1.6 Результат виконання програми

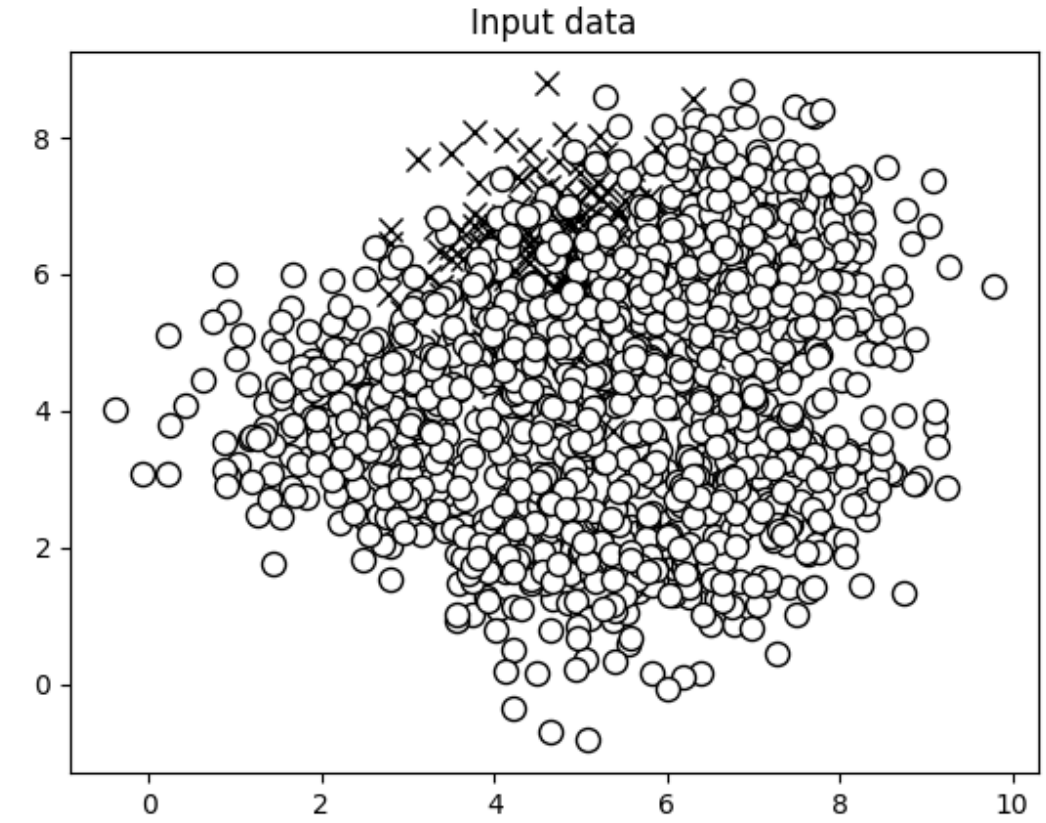

Рис. 1.7 Результат виконання програми

Test dataset

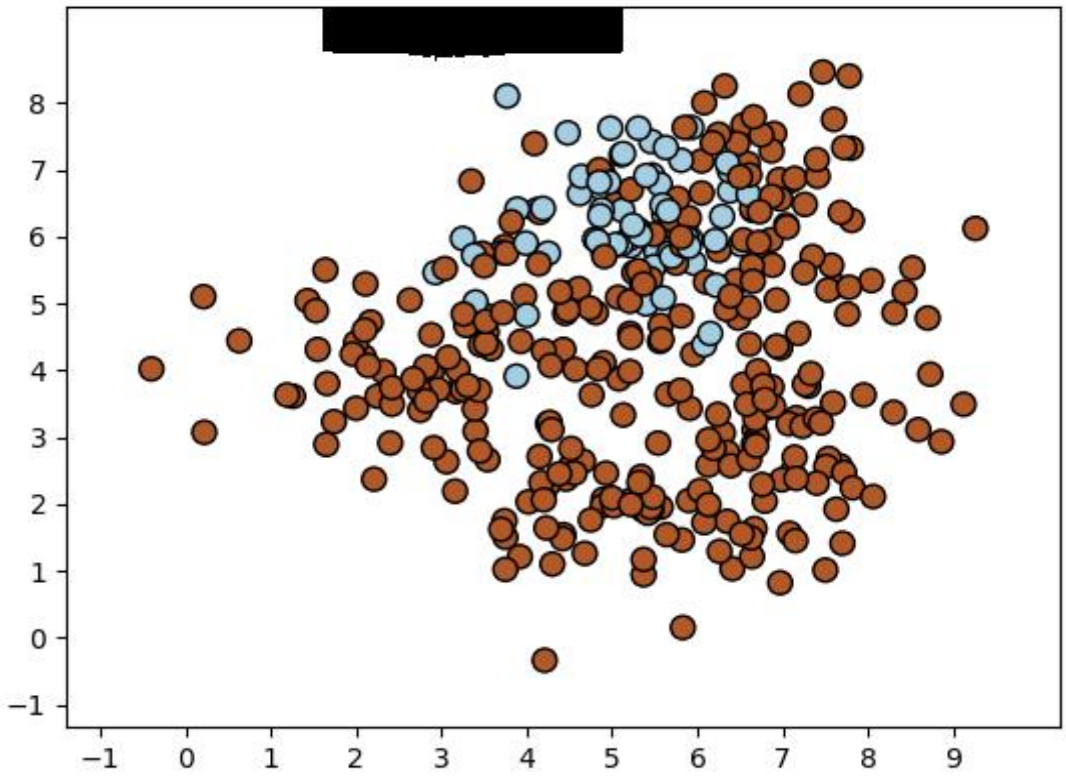

Рис. 1.8 Результат виконання програми

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

Classifier performance on training dataset

              precision    recall  f1-score   support

     Class-0       1.00      0.01      0.01       181
     Class-1       0.84      1.00      0.91       944

    accuracy                           0.84      1125
   macro avg       0.92      0.50      0.46      1125
weighted avg       0.87      0.84      0.77      1125

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

```
              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

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

Рис. 1.9 Результат виконання програми

Завдання 2.3. Знаходження оптимальних навчальних параметрів за допомогою сіткового пошуку

```
import numpy as np
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split, GridSearchCV

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 classifier.cv_results_:
        #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))
```

Рис. 2.1 Код програми

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

Grid scores for the parameter grid:

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:
```

```
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
```

Рис. 2.2 Результат виконання програми

Завдання 2.4. Обчислення відносної важливості ознак.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn import datasets
from sklearn.metrics import mean_squared_error, explained_variance_score
from sklearn.utils import shuffle

housing_data = datasets.load_boston()


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, feature_names[index_sorted])
plt.ylabel('Relative Importance')
plt.title('Feature importance using AdaBoost regressor')
plt.show()
```

Рис. 2.3 Код програми

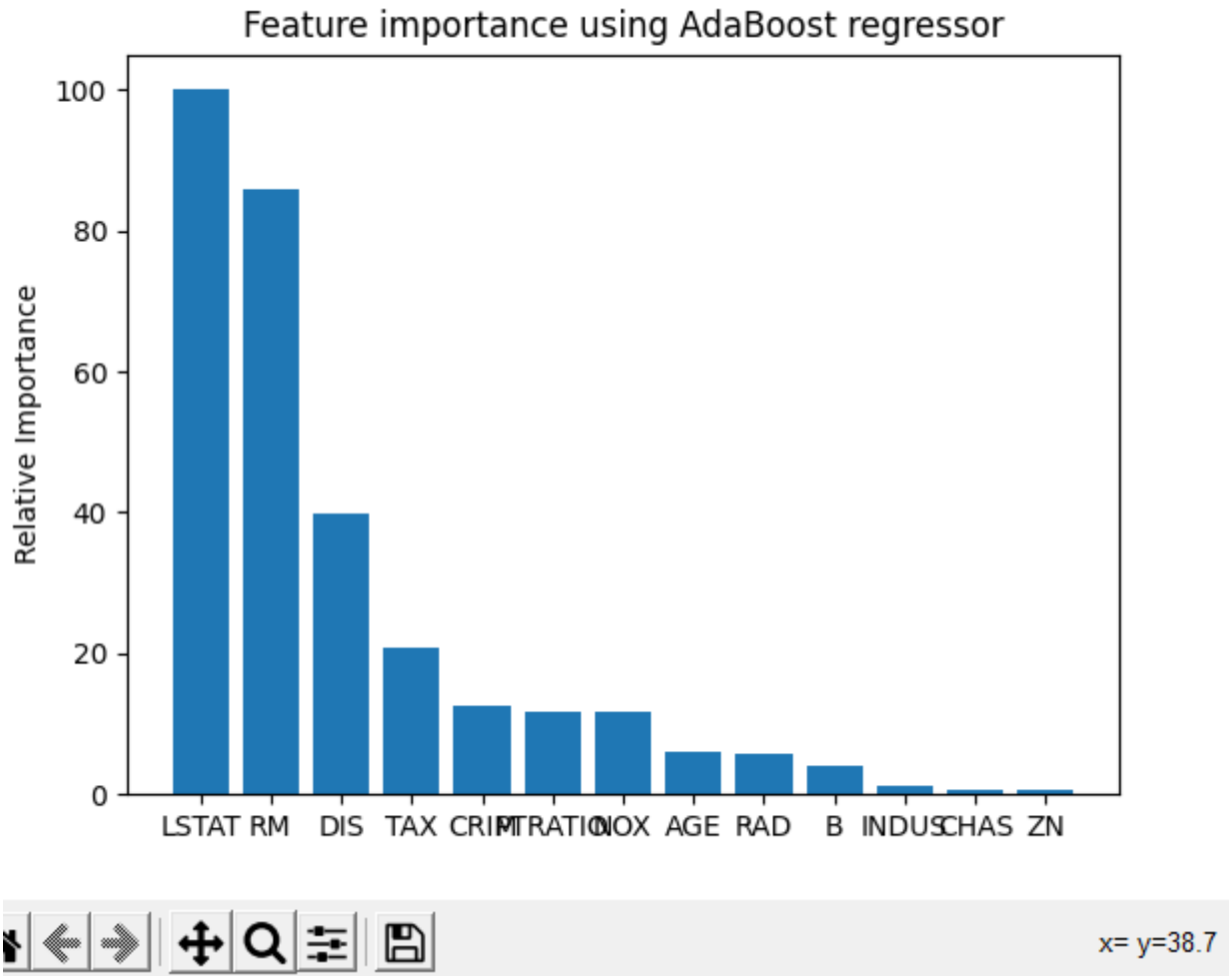

Рис. 2.4 Результат виконання програми

```
ADABOOST REGRESSOR
Mean squared error = 22.7
Explained variance score = 0.79
```

Рис. 2.5 Результат виконання програми

Висновок: для обчислення важливості ознак використовується регресор AdaBoost. Алгоритм AdaBoost це алгоритм машинного навчання, являється мета-алгоритмом, в процесі навчання будує композицію з базових алгоритмів навчання для покращення їх ефективності. AdaBoost є алгоритмом адаптивної роботи в тому сенсі, що кожен наступний класифікатор будується за об'єктом, які погано класифікуються попередніми класифікаторами.

AdaBoost викликає слабкий класифікатор у циклі. Після кожного виклику оновлюється розподіл весів, які відзначають важливість кожного з об'єктів навчального множини для класифікації.

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

```python
import numpy as np
from sklearn.metrics import mean_absolute_error
from sklearn import preprocessing
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.model_selection import train_test_split

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(test_datapoint[i]))
        count = count + 1

test_datapoint_encoded = np.array(test_datapoint_encoded)

print("Predicted traffic:", int(regressor.predict([test_datapoint_encoded])[0]))
```

Рис. 2.6 Код програми

```
Mean absolute error: 7.42
```

Рис. 2.7 Результат виконання програми

Висновок: я використовав спеціалізовані бібліотеки та мову програмування Python дослідивши методи регресії даних у машинному навчанні.