ЛАБОРАТОРНА РОБОТА № 5

ДОСЛІДЖЕННЯ МЕТОДІВ АНСАБЛЕВОГО НАВЧАННЯ

Мета: використовуючи спеціалізовані бібліотеки та мову програмування Python дослідити різні ансаблів у машинному навчанні.

Варіант 12

Хід роботи:

Завдання 1

Створення класифікаторів на основі випадкових тагранично випадкових лісів

Програмний код:

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

matplotlib.use('TkAgg')


def build_arg_parser():
    parser = argparse.ArgumentParser(
        description='Classify data using Ensemble Learning techniques'
    )
    parser.add_argument(
        '--classifier-type',
        dest='classifier_type',
        required=False,
        default='rf',
        choices=['rf', 'erf'],
        help="Type of classifier to use; can be either 'rf' or 'erf'"
    )
    return parser


def load_data(input_file):
    data = np.loadtxt(input_file, delimiter=',')
    X, y = data[:, :-1], data[:, -1]
    return X, y
```

```
def plot_input_data(X, y):
    classes = np.unique(y)
    markers = ['s', 'o', '^']
    plt.figure()
    for idx, cls in enumerate(classes):
        class_data = X[y == cls]
        plt.scatter(
            class_data[:, 0], class_data[:, 1],
            s=75, facecolors='white', edgecolors='black',
            linewidth=1, marker=markers[idx], label=f'Class {int(cls)}'
        )
    plt.title('Input Data')
    plt.legend()
    plt.show()


def get_classifier(classifier_type, params):
    if classifier_type == 'rf':
        return RandomForestClassifier(**params)
    elif classifier_type == 'erf':
        return ExtraTreesClassifier(**params)


def plot_decision_boundaries(classifier, X, y, title):
    h = 0.01
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    Z = classifier.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    plt.figure()
    plt.contourf(xx, yy, Z, alpha=0.8, cmap=ListedColormap(['#FFAAAA', '#AAFFAA',
'#AAAAFF']))
    markers = ['s', 'o', '^']
    for idx, cls in enumerate(np.unique(y)):
        class_data = X[y == cls]
        plt.scatter(
            class_data[:, 0], class_data[:, 1],
            s=75, edgecolor='black', marker=markers[idx], label=f'Class
{int(cls)}'
        )
    plt.title(title)
    plt.legend()
    plt.show()


def main():
    args = build_arg_parser().parse_args()
    classifier_type = args.classifier_type
    params = {'n_estimators': 100, 'max_depth': 4, 'random_state': 0}

    input_file = 'data_random_forests.txt'
    X, y = load_data(input_file)
    plot_input_data(X, y)
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.25, random_state=5
    )
    classifier = get_classifier(classifier_type, params)
    classifier.fit(X_train, y_train)
```

```
    plot_decision_boundaries(classifier, X_train, y_train, "Training Set")
    plot_decision_boundaries(classifier, X_test, y_test, "Test Set")

    class_names = [f'Class-{int(cls)}' for cls in np.unique(y)]
    print("\n" + "#" * 40)
    print("Classifier performance on training dataset\n")
    print(classification_report(y_train, classifier.predict(X_train), tar-
get_names=class_names))
    print("#" * 40 + "\n")
    print("Classifier performance on test dataset\n")
    print(classification_report(y_test, classifier.predict(X_test), tar-
get_names=class_names))
    print("#" * 40 + "\n")

    test_datapoints = np.array([[5, 5], [3, 6], [6, 4], [7, 2], [4, 4], [5, 2]])
    print("Confidence measure:")
    for datapoint in test_datapoints:
        probabilities = classifier.predict_proba([datapoint])[0]
        predicted_class = f"Class-{np.argmax(probabilities)}"
        print('\nDatapoint:', datapoint)
        print('Predicted class:', predicted_class)

    plot_decision_boundaries(classifier, np.vstack((X_test, test_datapoints)),
                             np.hstack((y_test, [0] * len(test_datapoints))),
"Test Points with Predictions")


if __name__ == '__main__':
    main()
```

Результат виконання:

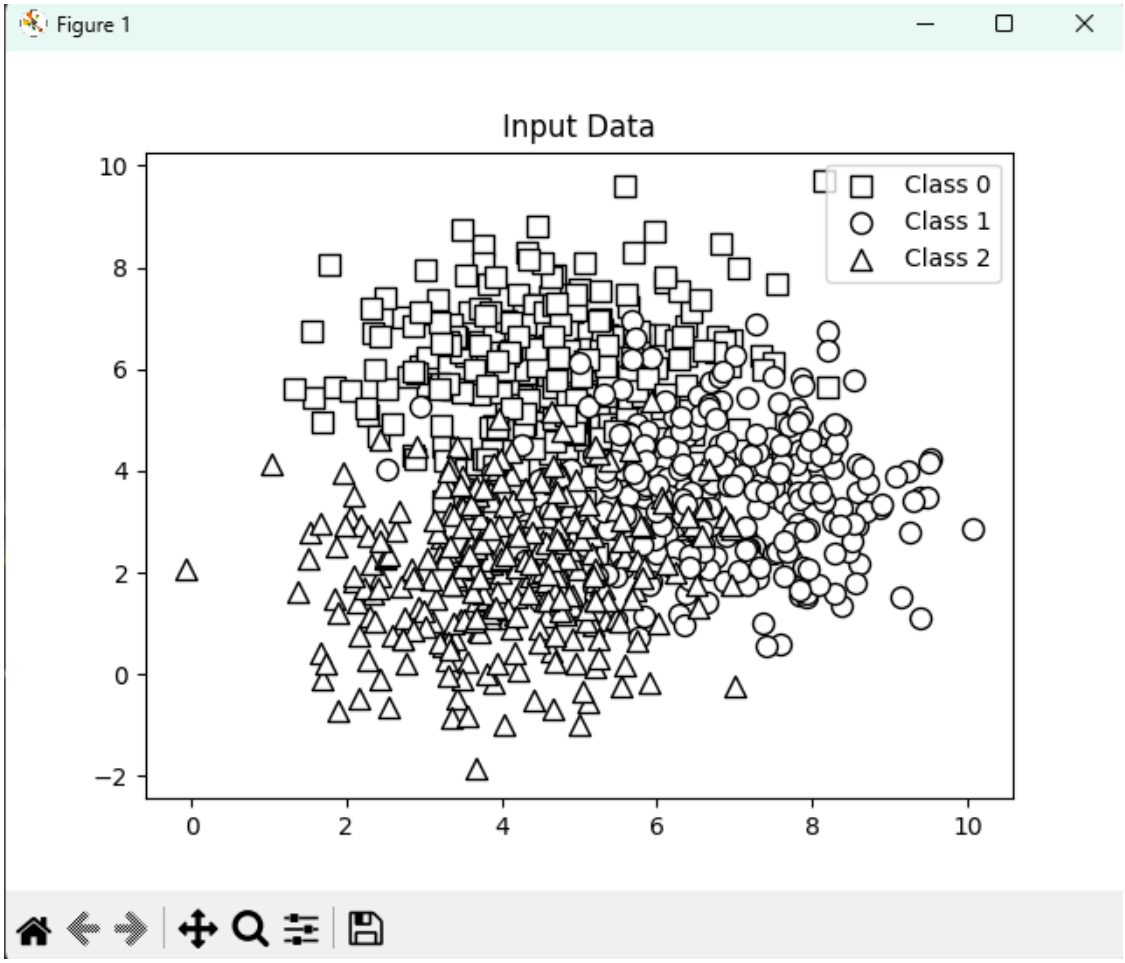


Figure 1
Input Data
Class 0
Class 1
Class 2
10
8
6
4
2
0
−2
0
2
4
6
8
10

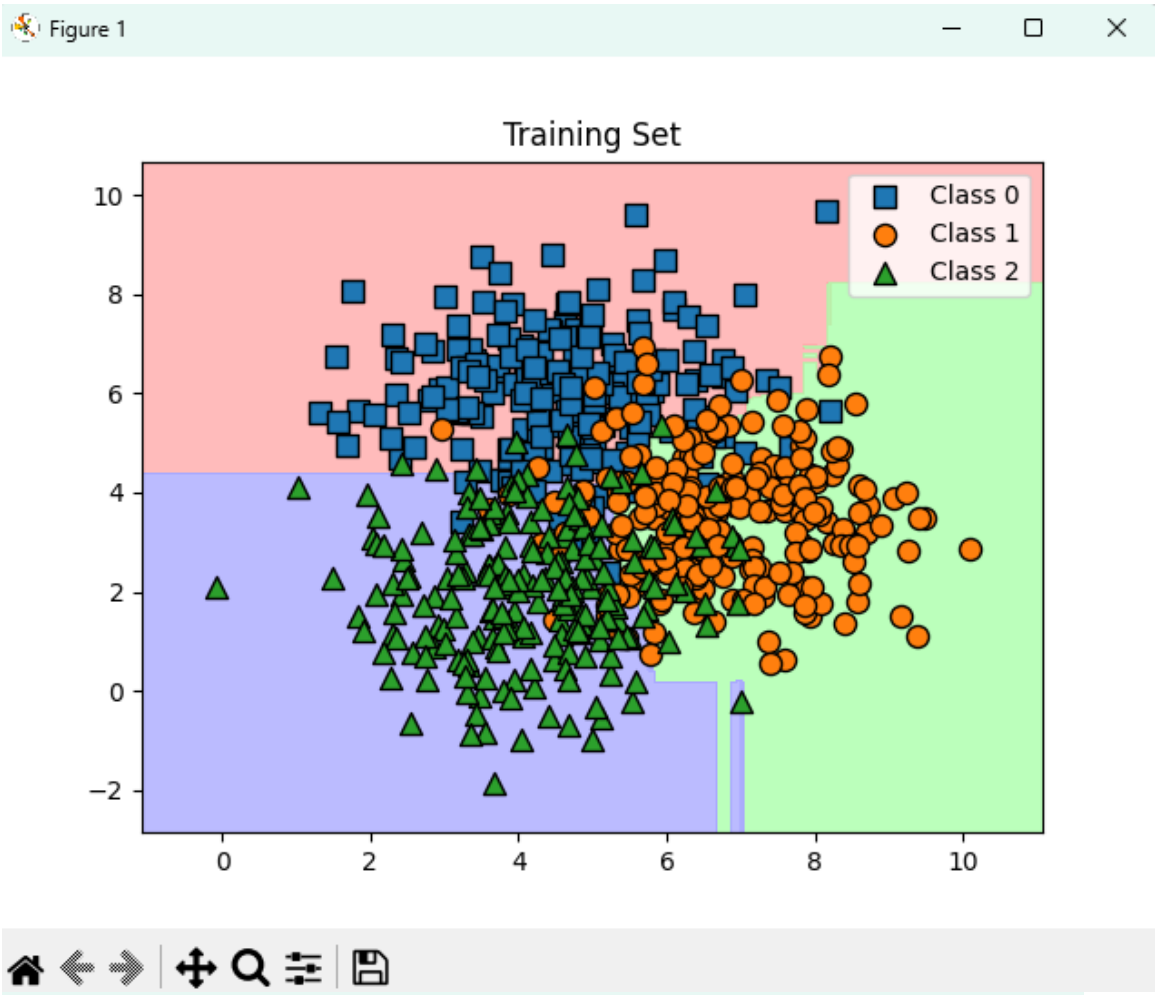


Figure 1
Training Set
Class 0
Class 1
Class 2
10
8
6
4
2
0
−2
0
2
4
6
8
10

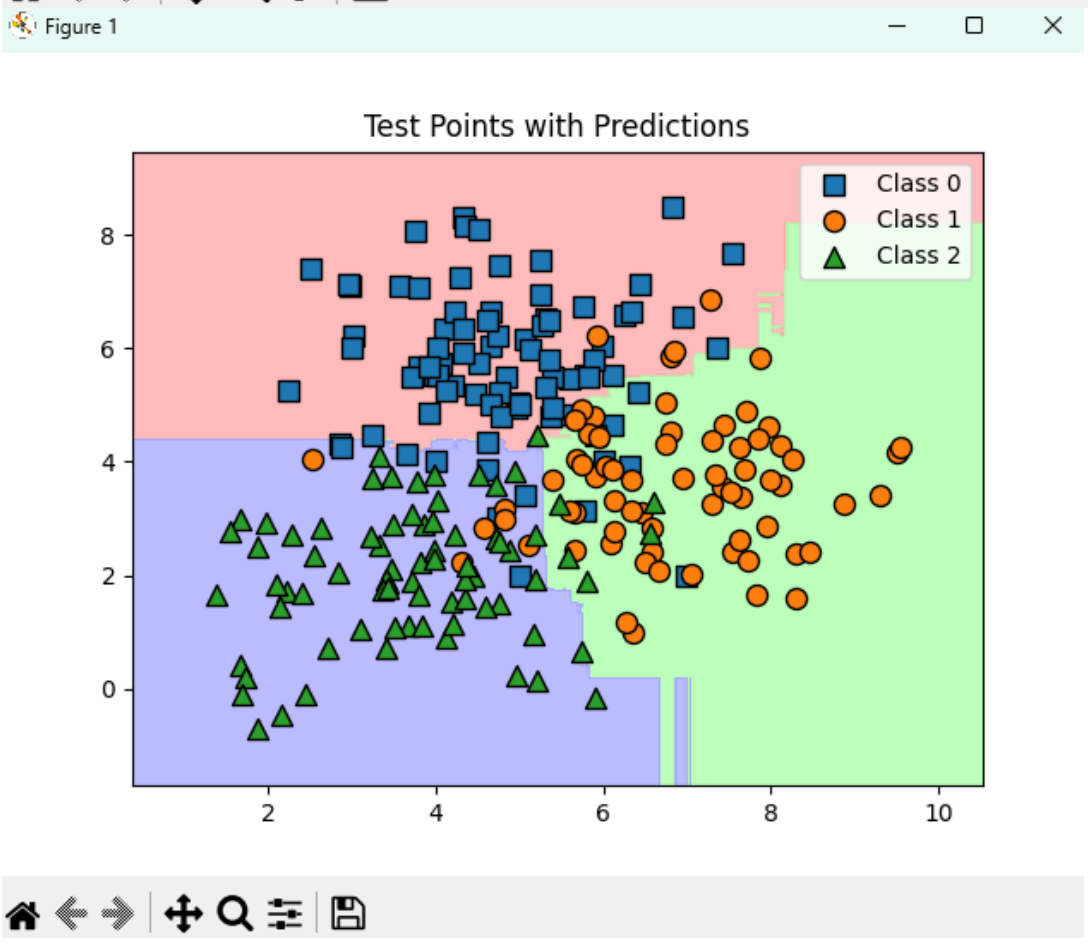


Figure 1
Test Points with Predictions
Class 0
Class 1
Class 2
8
6
4
2
0
2
4
6
8
10

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

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

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

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

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

Datapoint: [5 2]
Predicted class: Class-2

Process finished with exit code 0
```

Висновок: Класифікатори сконструйовані на основі випадкового і гранично випадкового лісу схожі між собою, але класифікатор на основі випадкового лісу виконує свою роботу дещо краще за інший варіант.

Завдання 2 Обробка дисбалансу класів

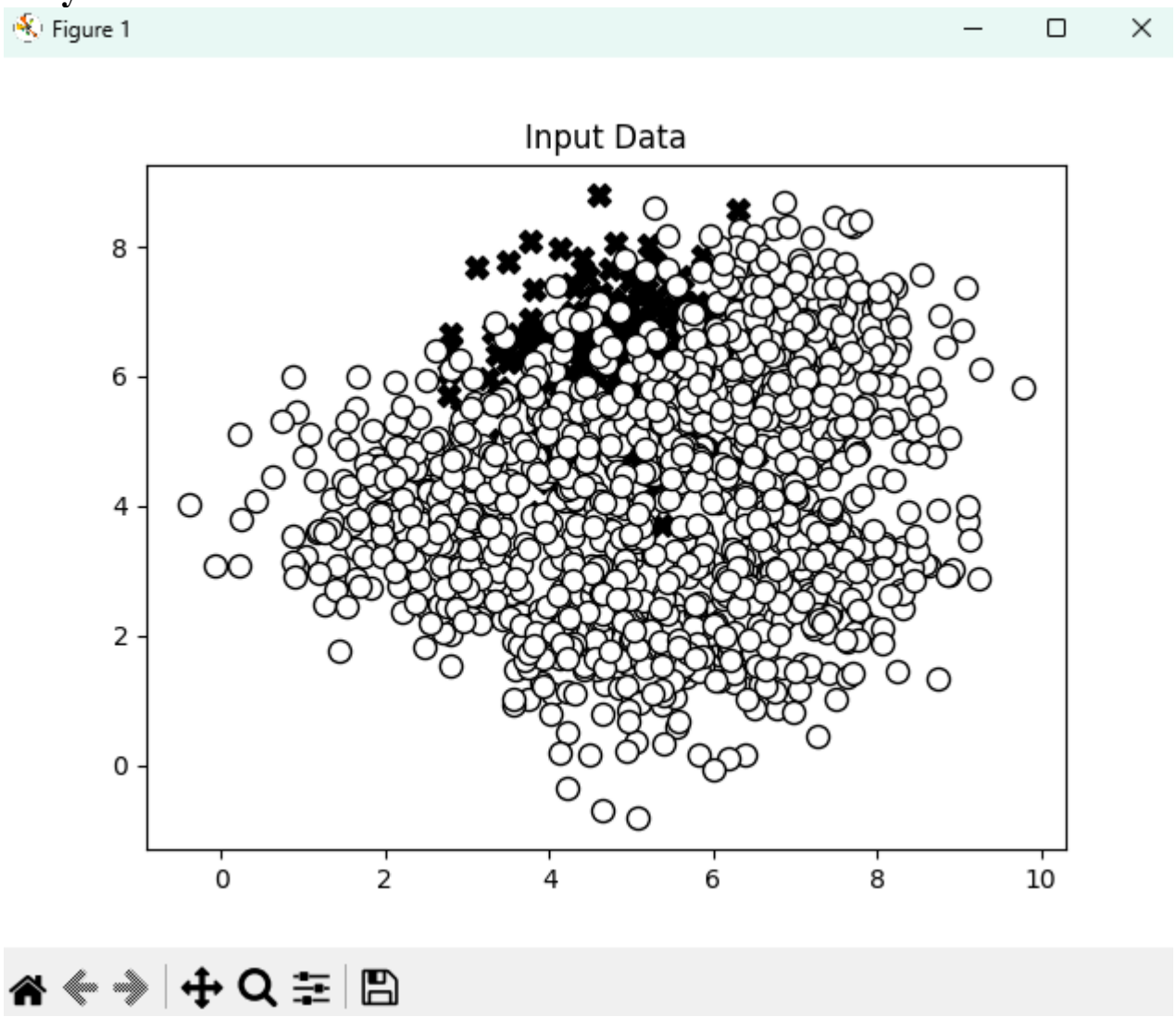
Програмний код:

```
import sys
import numpy as np
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from matplotlib.colors import ListedColormap
import matplotlib
import matplotlib.pyplot as plt

matplotlib.use('TkAgg')


def visualize_classifier(classifier, X, y, title):
    h = 0.01
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
```

```
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    Z = classifier.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    plt.figure()
    plt.contourf(xx, yy, Z, alpha=0.8, cmap=ListedColormap(['#FFAAAA',
'#AAAAFF']))
    markers = ['X', 'o']
    colors = ['black', 'white']
    for idx, cls in enumerate(np.unique(y)):
        plt.scatter(
            X[y == cls, 0], X[y == cls, 1],
            c=colors[idx], edgecolor='black', marker=markers[idx], s=75,
            label=f'Class-{int(cls)}'
        )
    plt.title(title)
    plt.legend()
    plt.show()


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', edgecol-
ors='black', linewidth=1, marker='X')
plt.scatter(class_1[:, 0], class_1[:, 1], s=75, facecolors='white', edgecol-
ors='black', linewidth=1, marker='o')
plt.title('Input Data')
plt.show()

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, ran-
dom_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 TypeError("Invalid input argument; should be 'balance'")

classifier = ExtraTreesClassifier(**params)
classifier.fit(X_train, y_train)

visualize_classifier(classifier, X_train, y_train, title="Training Set")
visualize_classifier(classifier, X_test, y_test, title="Test Set")

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), tar-
get_names=class_names))
print("#" * 40 + "\n")
print("#" * 40)
print("\nClassifier performance on test dataset\n")
```

```
print(classification_report(y_test, classifier.predict(X_test), tar-
get_names=class_names))
print("#" * 40 + "\n")
```

Результат виконання:



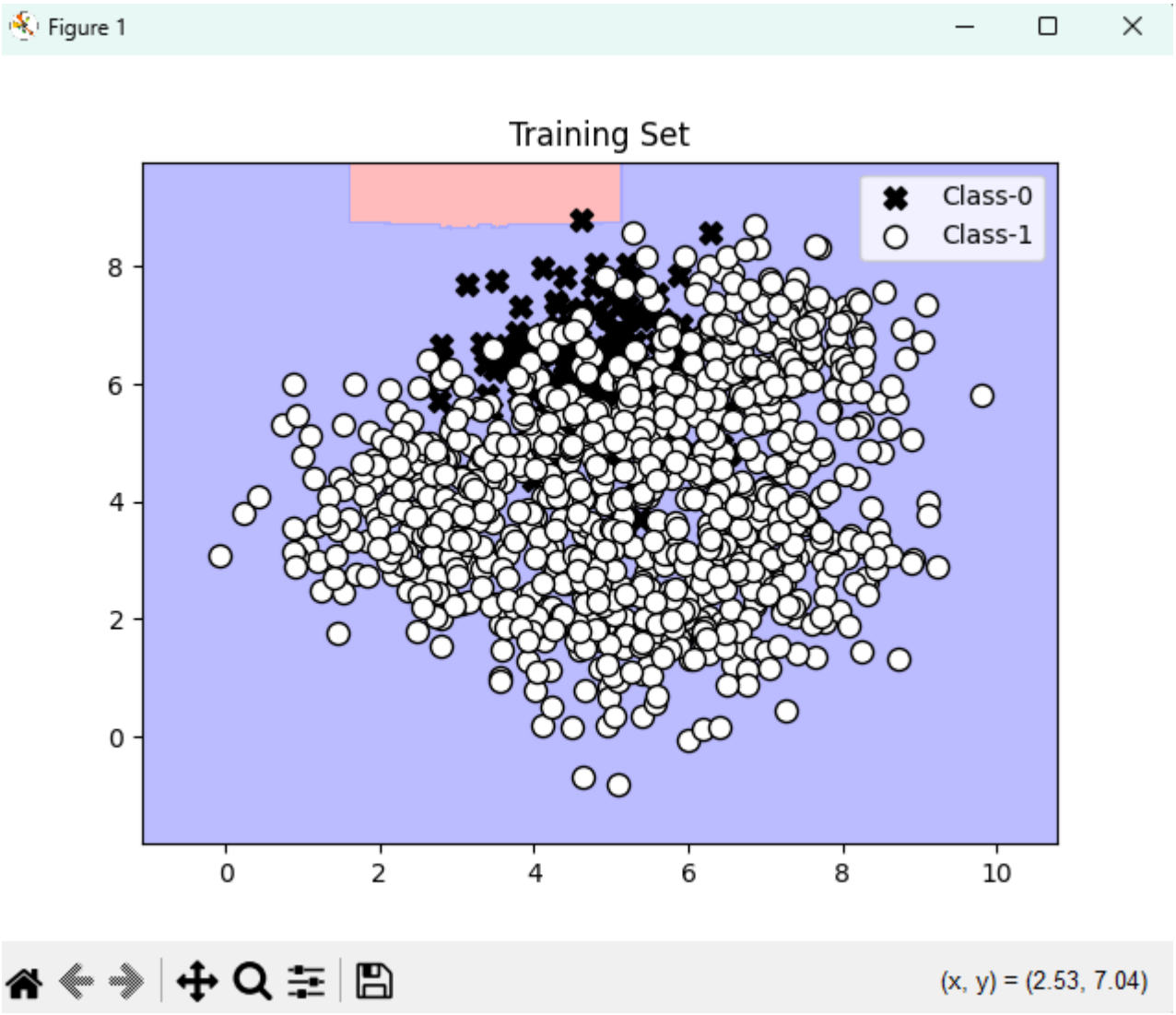
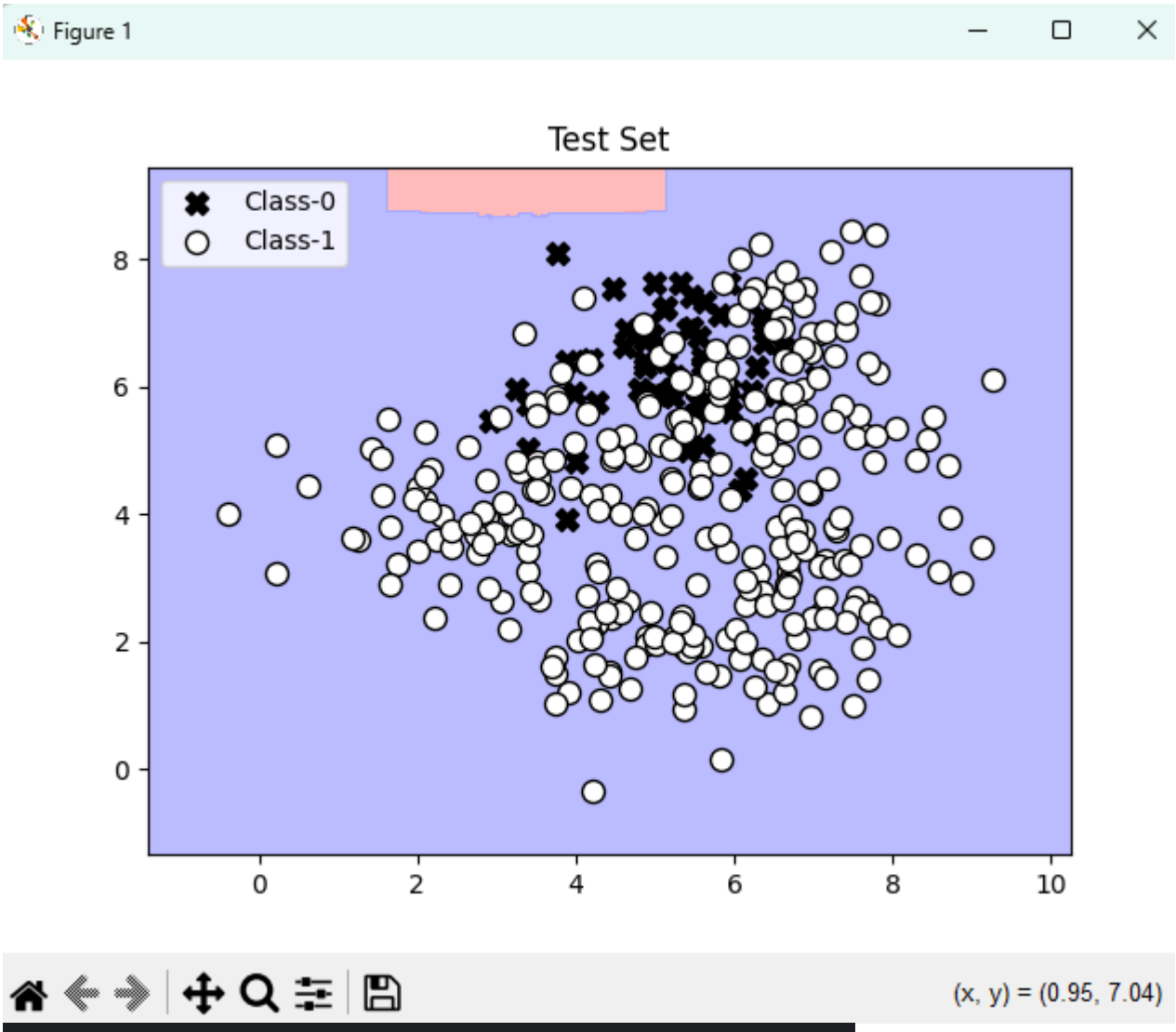


Figure 1
Training Set
Class-0
Class-1
8
6
4
2
0
0
2
4
6
8
10
(x, y) = (2.53, 7.04)




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

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

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


Process finished with exit code 0
```

Висновок: З параметром «'class_weight': 'balanced'» класифікатор працює більш збалансовано і параметри класифікації не скачуть від одиниці до нуля.

Завдання 3

Знаходження оптимальних навчальних параметрів за допомогою сіткового пошуку

Програмний код:

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

input_file = 'data_random_forests.txt'
data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1]
# Розбиття даних на три класи на підставі міток
class_0 = np.array(X[y == 0])
class_1 = np.array(X[y == 1])
class_2 = np.array(X[y == 2])
# Розбиття даних на навчальний та тестовий набори
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, ran-
dom_state=5)
# Визначення сітки значень параметрів
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), parame-
ter_grid, cv=5, scoring=metric)
    classifier.fit(X_train, y_train)
print("\nGrid scores for the parameter grid:")
```

```
for i in range(len(classifier.cv_results_['params'])):
    print(classifier.cv_results_['params'][i], '-->', classifier.cv_re-
sults_['mean_test_score'][i])
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:

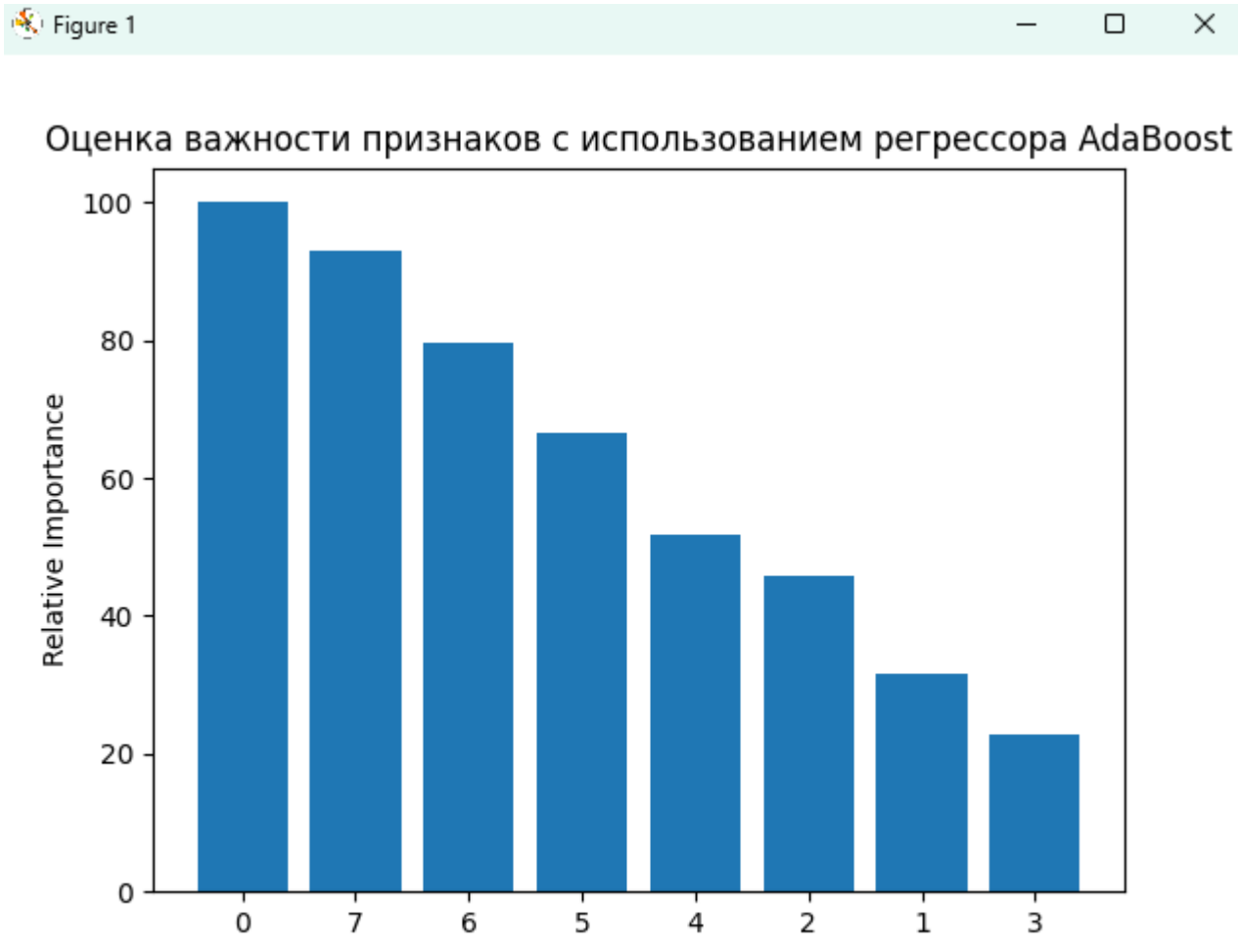
```
##### Searching optimal parameters for precision_weighted

##### Searching optimal parameters for recall_weighted

Grid scores for the parameter grid:
{'max_depth': 2, 'n_estimators': 100} --> 0.842962962962963
{'max_depth': 4, 'n_estimators': 100} --> 0.837037037037037
{'max_depth': 7, 'n_estimators': 100} --> 0.8414814814814815
{'max_depth': 12, 'n_estimators': 100} --> 0.8296296296296297
{'max_depth': 16, 'n_estimators': 100} --> 0.8148148148148149
{'max_depth': 4, 'n_estimators': 25} --> 0.842962962962963
{'max_depth': 4, 'n_estimators': 50} --> 0.8355555555555556
{'max_depth': 4, 'n_estimators': 100} --> 0.837037037037037
{'max_depth': 4, 'n_estimators': 250} --> 0.8414814814814815

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


Process finished with exit code 0
```

Завдання 4 Обчислення відносної важливості ознак

Програмний код:

```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt

matplotlib.use('TkAgg')
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
from sklearn.model_selection import train_test_split
from sklearn.datasets import fetch_california_housing
import pandas as pd
import numpy as np

housing_data = datasets.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, ran-
dom_state=7)

regressor = AdaBoostRegressor(DecisionTreeRegressor(max_depth=4), n_estima-
tors=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, index_sorted)
plt.ylabel('Relative Importance')
plt.title('Оценка важности признаков с использованием регрессора AdaBoost')
plt.show()
```

Результат виконання:




```
C:\Users\User\AppData\Local\Programs

ADABOOST REGRESSOR
Mean squared error = 1.18
Explained variance score = 0.47

Process finished with exit code 0
```

Завдання 5 Прогнозування інтенсивності дорожнього руху задопомогою класифікатора на основі гранично випадкових лісів

Програмний код:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, 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.strip().split(',')
        data.append(items)
data = np.array(data)

label_encoders = []
X_encoded = np.empty(data.shape, dtype=object)

for i in range(data.shape[1]):
    if data[:, i][0].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, ran-
dom_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 = []

for i, item in enumerate(test_datapoint):
    if item.isdigit():
        test_datapoint_encoded.append(int(item))
    else:
        encoder = label_encoders[i]
        test_datapoint_encoded.append(encoder.transform([item])[0])

test_datapoint_encoded = np.array(test_datapoint_encoded)

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

Результат виконання:

```
C:\Users\user\AppData\Local\Programs
Mean absolute error: 7.42
Predicted traffic: 26

Process finished with exit code 0
```

Висновки: в ході виконання лабораторної роботи було використано спеціалізовані бібліотеки та мову програмування Python та дослідив методи ансамблів у машинному навчанні.