# ЛАБОРАТОРНА РОБОТА № 5

## Дослідження методів ансамблевого навчання

**Мета:** 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.

Використовувати файл вхідних даних: data_random_forests.txt, побудувати класифікатори на основі випадкових та гранично випадкових лісів.

```python
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 utilities import visualize_classifier


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


if __name__ == '__main__':
    # Вилучення вхідних аргументів
    args = build_arg_parser().parse_args()
    classifier_type = args.classifier_type

# Завантаження вхідних даних
input_file = 'Task/data_random_forests.txt'
data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1]

# Розбиття вхідних даних на три класи
class_0 = np.array(X[y == 0])
class_1 = np.array(X[y == 1])
class_2 = np.array(X[y == 2])

# Візуалізація вхідних даних
plt.figure()
scatter_params = {'s': 75, 'facecolors': 'white', 'edgecolors': 'black', 'linewidths': 1, 'marker': 's'}
plt.scatter(class_0[:, 0], class_0[:, 1], **scatter_params)
plt.scatter(class_1[:, 0], class_1[:, 1], **scatter_params)
plt.scatter(class_2[:, 0], class_2[:, 1], **scatter_params)
plt.title('Вхідні дані')
plt.show()

# Розбивка даних на навчальний та тестовий набори
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=5)

# Класифікатор на основі ансамблевого навчання
params = {'n_estimators': 100, 'max_depth': 4, 'random_state': 0}
```

Рис. 1. Код програми

```python
# Класифікатор на основі ансамблевого навчання
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')

# Перевірка роботи класифікатора
class_names = ['Class-0', 'Class-1', 'Class-2']
print('\n' + '#' * 40)
print('Classifier performance on training dataset')
print(classification_report(y_train, classifier.predict(X_train), target_names=class_names))
print('\n' + '#' * 40)
print('Classifier performance on test dataset')
print(classification_report(y_test, y_test_pred, target_names=class_names))

# Обчислення параметрів довірливості
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('Datapoint:', datapoint)
    print('Predicted class:', predicted_class)

# Візуалізація точок даних
visualize_classifier(classifier, test_datapoints, [0]*len(test_datapoints), 'Test data points')
```

Рис. 2. Код програми

Дані з типом класифікатора rf:

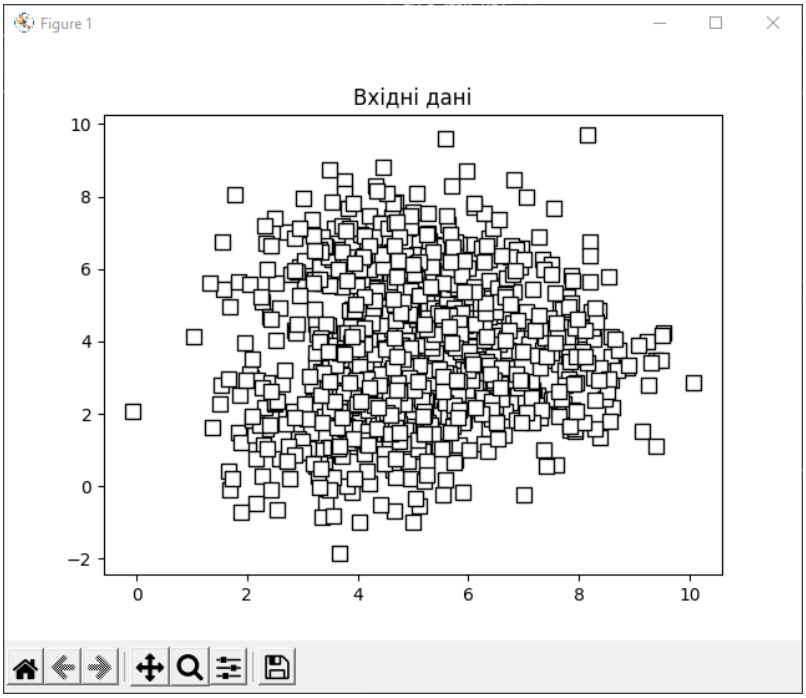


Рис. 3. Графік вхідних даних

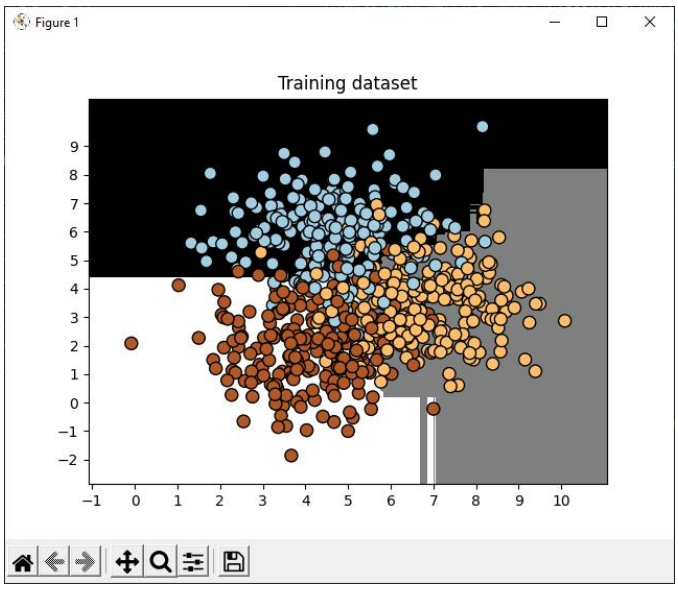


Рис. 4. Графік тренувальних даних

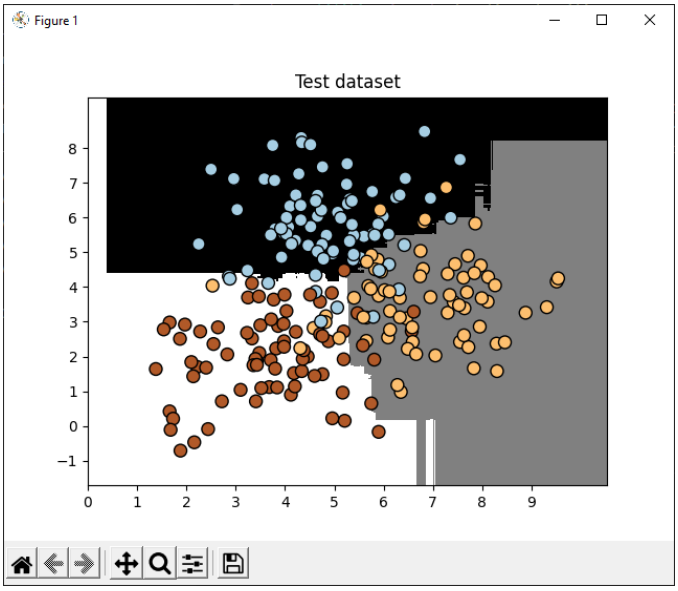


Рис. 5. Графік тестових даних

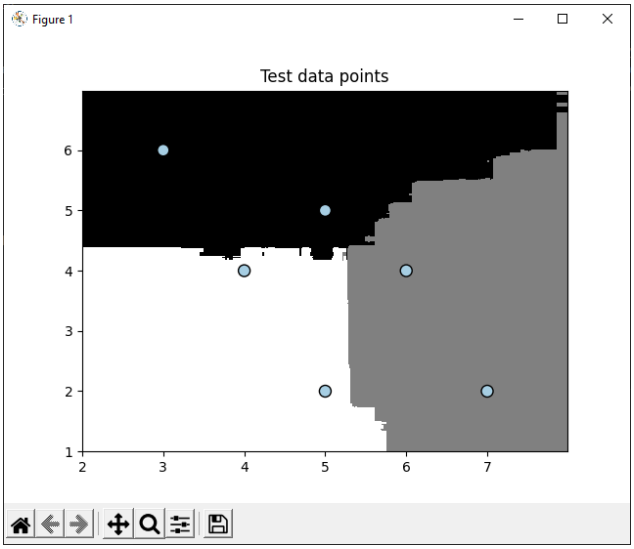


Рис. 6. Тестові точки

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

Рис. 7. Рівні довіри та оцінка класифікатора

```
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
```

Рис. 8. Передбачені класи

Дані з типом класифікатора erf:

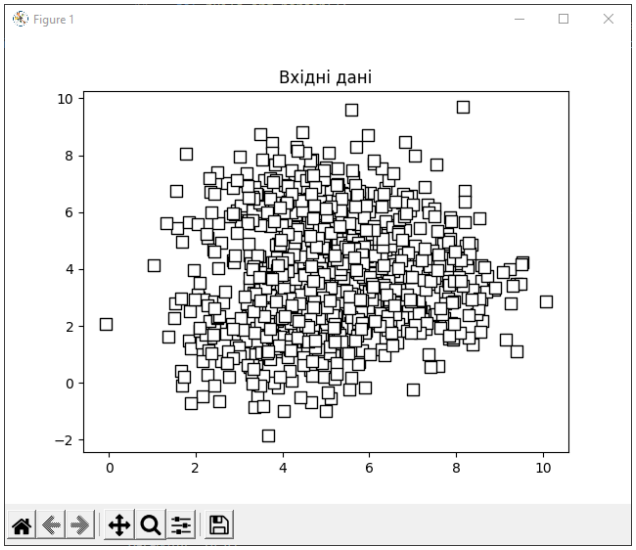


Рис. 9. Графік вхідних даних

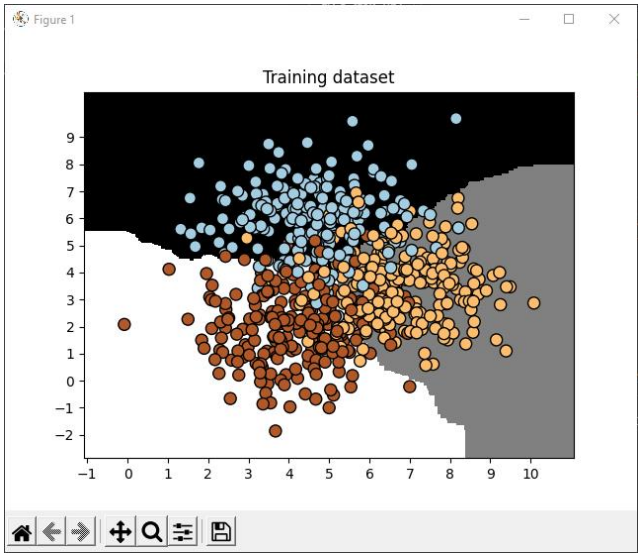


Рис. 10. Графік тренувальних даних

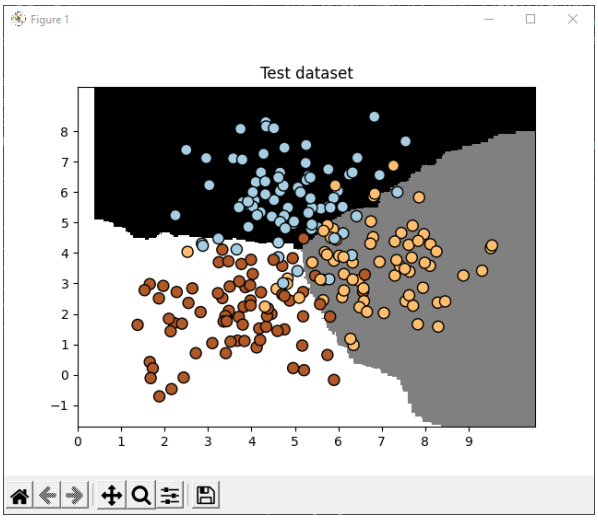


Рис. 11. Графік тестових даних

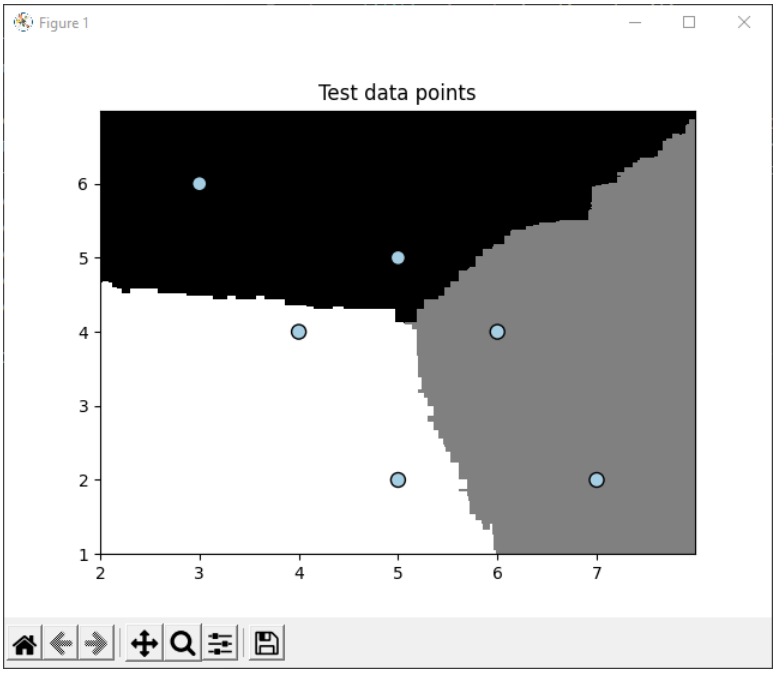


Рис. 12. Тестові точки

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

Рис. 13. Рівні довіри та оцінка класифікатора

```
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
```

Рис. 14. Передбачені класи

За графіками можна зробити висновок, що при типі класифікатора erf присутні більш плавні переходи, але рівень довіри та оцінки вищі для типу rf. Класи визначені однаково і обох випадках.

**Завдання 2.2.** Обробка дизбалансу класів.

Використовуючи для аналізу дані, які містяться у файлі data_imbalance.txt проведіть обробку з урахуванням дисбалансу класів.

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

# Завантаження вхідних даних
input_file = 'Task/data_imbalance.txt'
data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1]

# Поділ вхідних даних на два класи на підставі міток
class_0 = np.array(X[y == 0])
class_1 = np.array(X[y == 1])

# Візуалізація вхідних даних
plt.figure()
scatter_params = {'s': 75, 'edgecolors': 'black', 'linewidths': 1}
plt.scatter(class_0[:, 0], class_0[:, 1], facecolors='black', marker='x', **scatter_params)
plt.scatter(class_1[:, 0], class_1[:, 1], facecolors='white', marker='o', **scatter_params)
plt.title('Вхідні дані')

# Розбиття даних на навчальний та тестовий набори
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=5)

# Класифікатор на основі гранично випадкових лісів
params = {'n_estimators': 100, 'max_depth': 4, 'random_state': 0}
if len(sys.argv) > 1:
    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 data')
```

Рис. 15. Код програми

```
# Обчислення показників ефективності класифікатора
class_names = ['Class-0', 'Class-1']
print('\n' + '#' * 40)
print('Classifier performance on training dataset')
print(classification_report(y_train, classifier.predict(X_train), target_names=class_names))
print('\n' + '#' * 40)
print('Classifier performance on test dataset')
print(classification_report(y_test, y_test_pred, target_names=class_names))
```

Рис. 16. Код програми

Результати виконання програми з використанням параметру balance

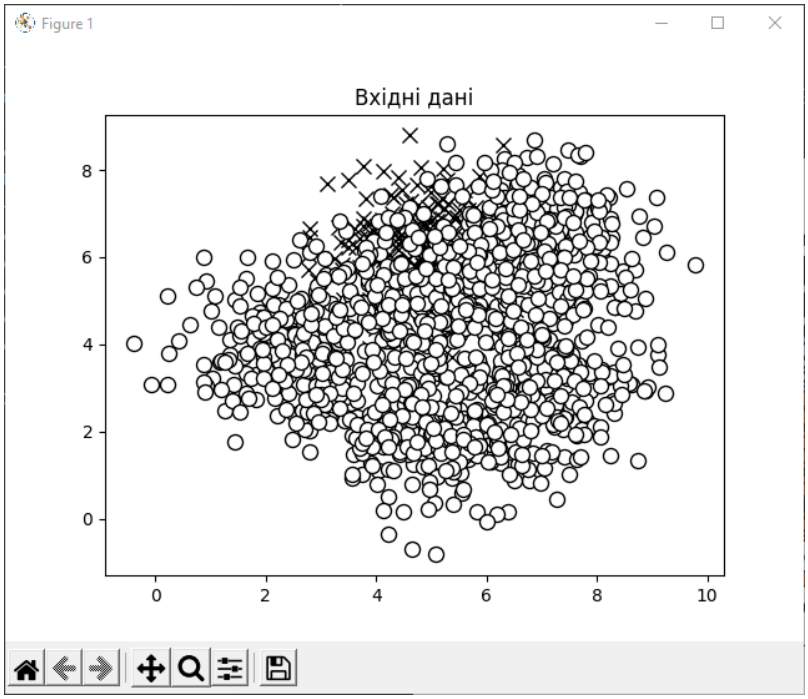


Рис. 17. Графік вхідних даних

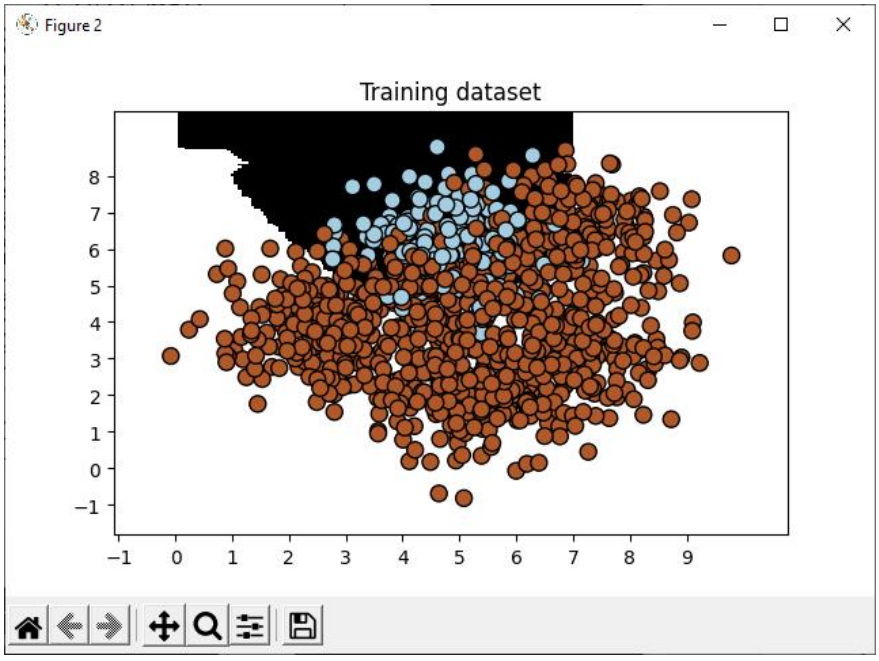


Рис. 18. Графік тренувальних даних

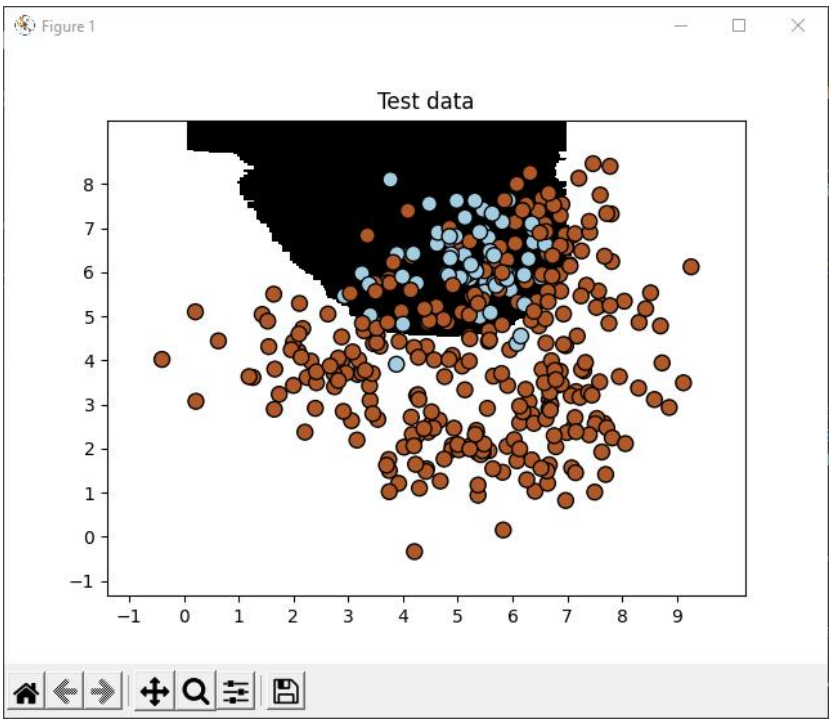


Рис. 19. Графік тестових даних

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

Рис. 20. Показники ефективності класифікатора

Результат виконання програми без параметру balance

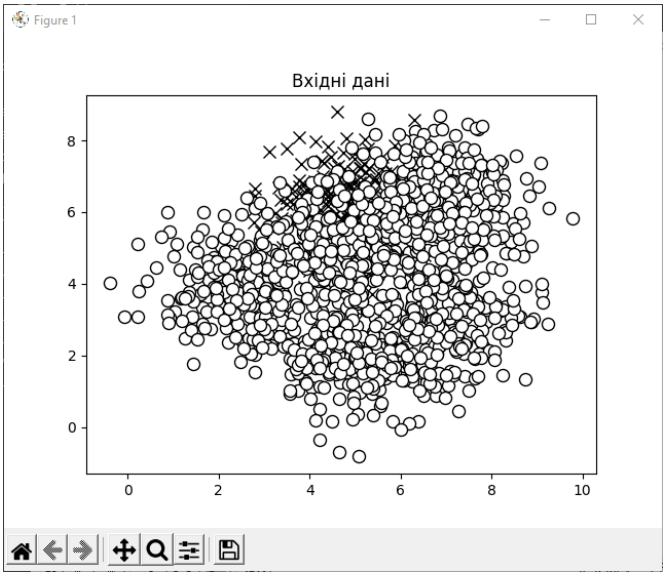


Рис. 21. Графік вхідних даних

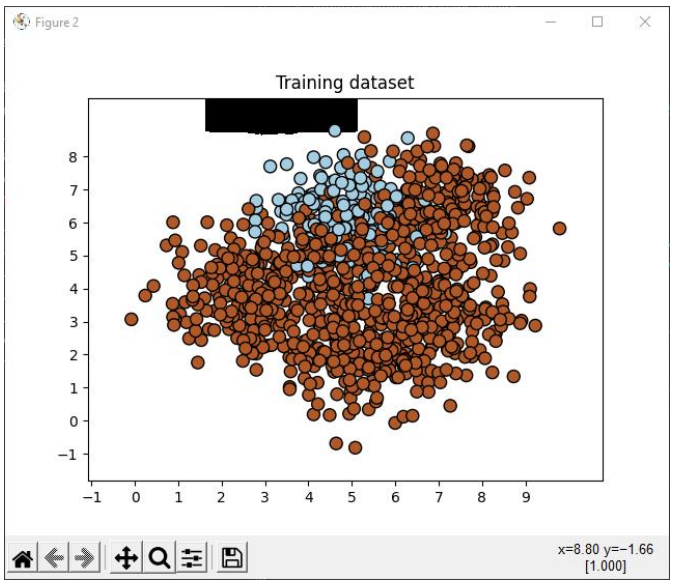


Рис. 22. Графік тренувальних даних

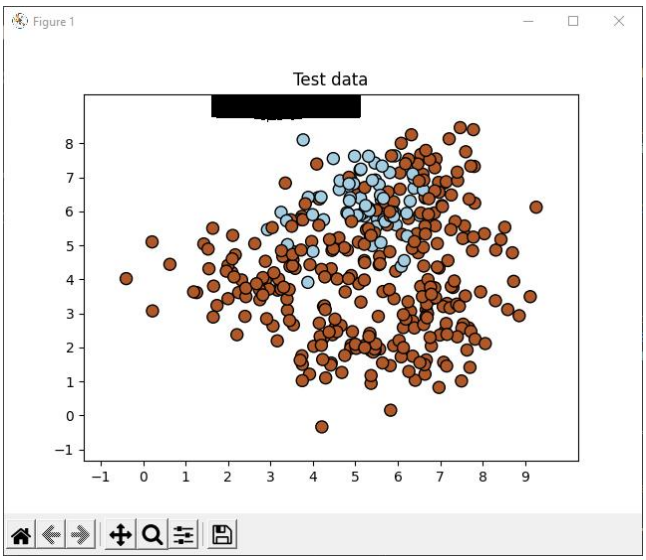


Рис. 23. Графік тестових даних

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

     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. 24. Показники ефективності класифікатора

При використанні класифікатора balance можемо бачити кращі показники ефективності, що також показують і графіки.

**Завдання 2.3.** Знаходження оптимальних навчальних параметрів за допомогою сіткового пошуку.

Використовуючи дані, що містяться у файлі data_random_forests.txt знайти оптимальних навчальних параметрів за допомогою сіткового пошуку.

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

# Завантаження вхідних даних
input_file = 'Task/data_random_forests.txt'
data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1]

# Розбиття вхідних даних на три класи
class_0 = np.array(X[y == 0])
class_1 = np.array(X[y == 1])
class_2 = np.array(X[y == 2])

# Розбивка даних на навчальний та тестовий набори
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=5)

# Класифікатор на основі ансамблевого навчання
parameter_grid = [
    {'n_estimators': [100], 'max_depth': [2, 4, 7, 12, 16]},
    {'n_estimators': [25, 50, 100, 250], 'max_depth': [4]}
]

metrics = ['precision_weighted', 'recall_weighted']
for metric in metrics:
    print('Searching optimal parameters for', metric)
    classifier = GridSearchCV(ExtraTreesClassifier(random_state=0), parameter_grid, cv=5, scoring=metric)
    classifier.fit(X_train, y_train)
    # !!! grid_scores_ is now deprecated
    '''
    print("Grid scores for the parameter grid:")
    for params, avg_score, _ in classifier.grid_scores_:
        print(params, '-->', round(avg_score, 3))
        print('Best parameters:', classifier.best_params_)
    '''

y_pred = classifier.predict(X_test)
print('Performance report')
print(classification_report(y_test, y_pred))
```

Рис. 25. Код програми

```
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. 26. Метрики класифікації

**Завдання 2.4.** Обчислення відносної важливості ознак

```python
# Завантаження даних із цінами на нерухомість
housing_data = datasets.load_boston()

# Перемішування даних
X, y = shuffle(housing_data.data, housing_data.target, random_state=7)

# Розбиття даних на навчальний та тестовий набори
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=7)

# Модель на основі регресора AdaBoost
regressor = AdaBoostRegressor(DecisionTreeRegressor(max_depth=4), n_estimators=400, random_state=7)
regressor.fit(X_train, y_train)

# Обчислення показників ефективності регресора AdaBoost
y_pred = regressor.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
evs = explained_variance_score(y_test, y_pred)
print('Adaboost regressor')
print('Mean squared error =', round(mse, 2))
print('Explained variance score =', round(evs, 2))

# Вилучення важливості ознак
feature_importances = regressor.feature_importances_
feature_names = housing_data.feature_names

# Нормалізація значень важливості ознак
feature_importances = 100.0 * (feature_importances / max(feature_importances))

# Сортування та перестановка значень
index_sorted = np.flipud(np.argsort(feature_importances))

# Розміщення міток уздовж осі X
pos = np.arange(index_sorted.shape[0]) + 0.5

# Побудова стовпчастої діаграми
plt.figure()
plt.bar(pos, feature_importances[index_sorted], align='center')
plt.xticks(pos, feature_names[index_sorted])
plt.ylabel('Relative importance')
plt.title('Оцінка важливості ознак, використовуючи регресор AdaBoost')
plt.show()
```

Рис. 27. Код програми

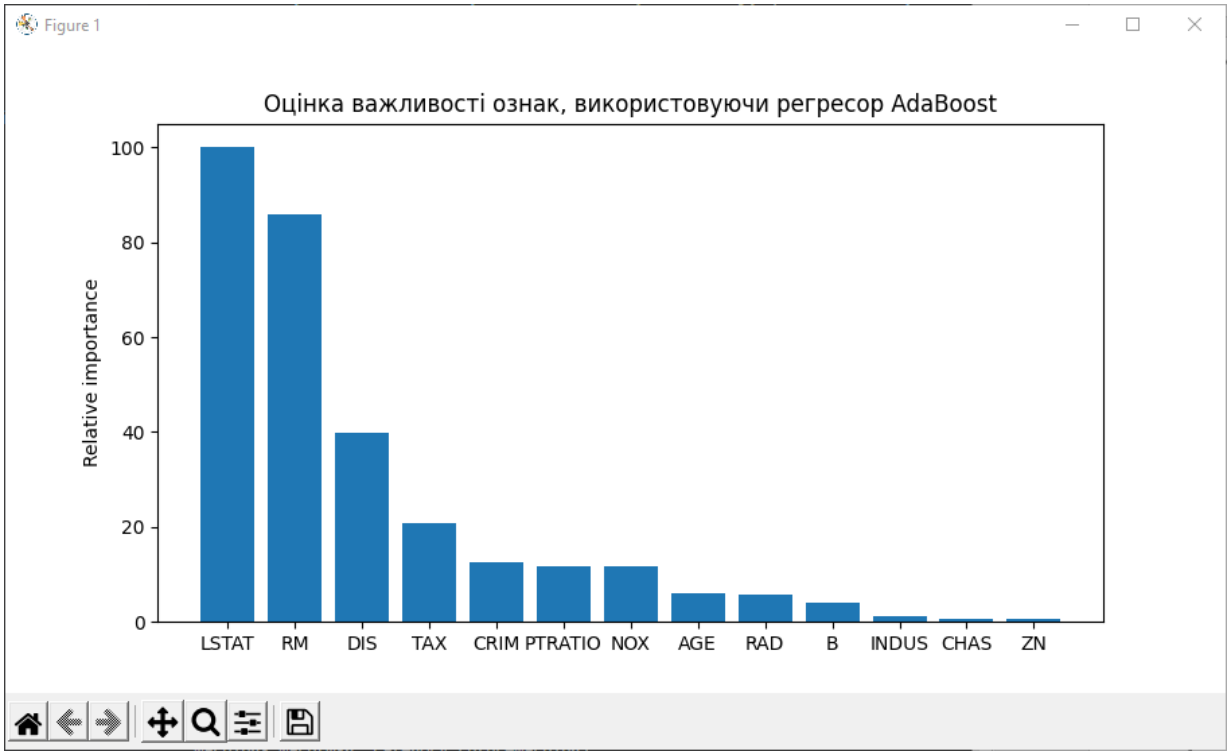


Рис. 28. Результат виконання програми

З отриманого графіку можна зробити висновок, що найбільш важливими є ознаки LSTAT та RM, ознаками середньої вартості є DIS, TAX, CRIM, PTRATIO та NOX, всі інші є неважливими.

```
Adaboost regressor
Mean squared error = 22.7
Explained variance score = 0.79
```

Рис. 29. Показники

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

Проведіть прогнозування інтенсивності дорожнього руху за допомогою класифікатора на основі гранично випадкових лісів.

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

# Завантажимо дані із файлу
input_file = 'Task/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)

# Перетворення рядкових даних на числові
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)

# Розбиття даних на навчальний та тестовий набори
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=5)

# Регресор на основі гранично випадкових лісів
params = {'n_estimators': 100, 'max_depth': 4, 'random_state': 0}
regressor = ExtraTreesRegressor(**params)
regressor.fit(X_train, y_train)
```

Рис. 30. Код програми

```python
# Обчислення характеристик ефективності регресора на тестових даних
y_pred = regressor.predict(X_test)
print('Mean absolute error =', round(mean_absolute_error(y_test, y_pred), 2))

# Тестування кодування на одиночному прикладі
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)

# Прогнозування результату для тестової точки даних
print('Predicted traffic:', int(regressor.predict([test_datapoint_encoded])[0]))
```

Рис. 31. Код програми

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

Рис. 32. Результат виконання програми

***Висновок:*** в ході виконання лабораторної роботи було досліджено методи ансамблів у машинному навчанні за допомогою мови програмування Python.

GitHub