Лабораторна робота №5

ДОСЛІДЖЕННЯ МЕТОДІВ АНСАМБЛЕВОГО НАВЧАННЯ

Мета: використовуючи спеціалізовані бібліотеки та мову програмування Python дослідити методи ансамблів у машинному навчанні.

Хід роботи:

Завдання 5.1. Створення класифікаторів на основі випадкових та гранично випадкових лісів.

Лістинг програми (файл LR_5_task_1.py):

```
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; 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_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])
```

Лістинг програми (продовження LR_5_task_1.py):

```
    plt.figure()
    plt.scatter(class_0[:, 0], class_0[:, 1], s=75,
                facecolors='white', edgecolors='black',
                linewidth=1, marker='s', label='Class 0')
    plt.scatter(class_1[:, 0], class_1[:, 1], s=75,
                facecolors='white', edgecolors='black',
                linewidth=1, marker='o', label='Class 1')
    plt.scatter(class_2[:, 0], class_2[:, 1], s=75,
                facecolors='white', edgecolors='black',
                linewidth=1, marker='^', label='Class 2')
    plt.title('Input data')
    plt.legend()
    plt.show()

    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')

    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:")
    y_pred_points = []
    for datapoint in test_datapoints:
        probabilities = classifier.predict_proba([datapoint])[0]
        predicted_class = np.argmax(probabilities)
        y_pred_points.append(predicted_class)
```

```
        print('\nDatapoint:', datapoint)
        print('Predicted class: Class-', predicted_class)
        print('Probabilities:', probabilities)

    visualize_classifier(classifier, test_datapoints, y_pred_points, 'Test
datapoints')
    plt.show()
```

Лістинг програми (файл utilities.py):

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

def visualize_classifier(classifier, X, y, title):
    x_min, x_max = X[:, 0].min() - 1.0, X[:, 0].max() + 1.0
    y_min, y_max = X[:, 1].min() - 1.0, X[:, 1].max() + 1.0
    step_size = 0.01
    x_values, y_values = np.meshgrid(np.arange(x_min, x_max, step_size),
                                     np.arange(y_min, y_max, step_size))

    output = classifier.predict(np.c_[x_values.ravel(), y_values.ravel()])
    output = output.reshape(x_values.shape)

    plt.figure()
    plt.pcolormesh(x_values, y_values, output, cmap=plt.cm.gray, shading='auto')
    plt.scatter(X[:, 0], X[:, 1], c=y, s=75, edgecolors="black", linewidth=1,
cmap=plt.cm.Paired)
    plt.title(title)
    plt.show()
```

Для початку запустимо програму з прапорцем rf.

Результат виконання програми:

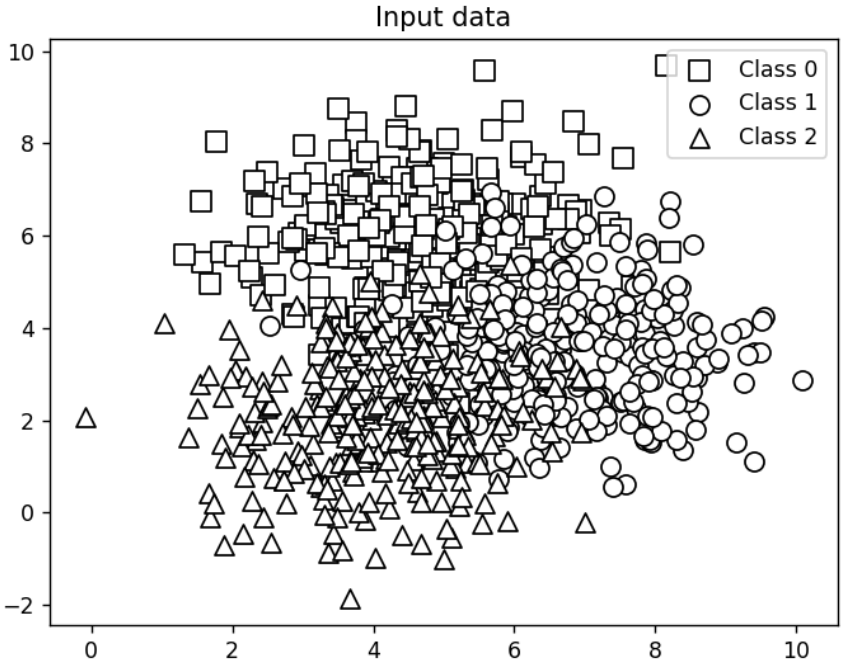

Рис. 1. Результат виконання програми

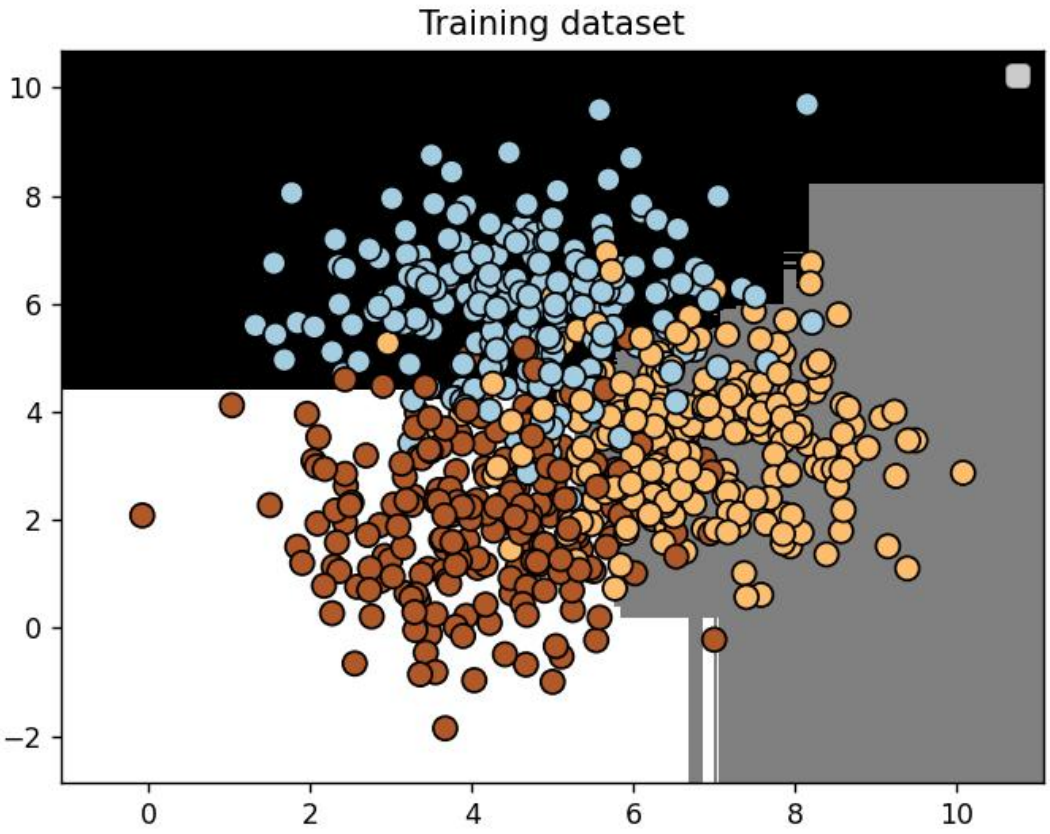

Рис. 2. Результат виконання програми

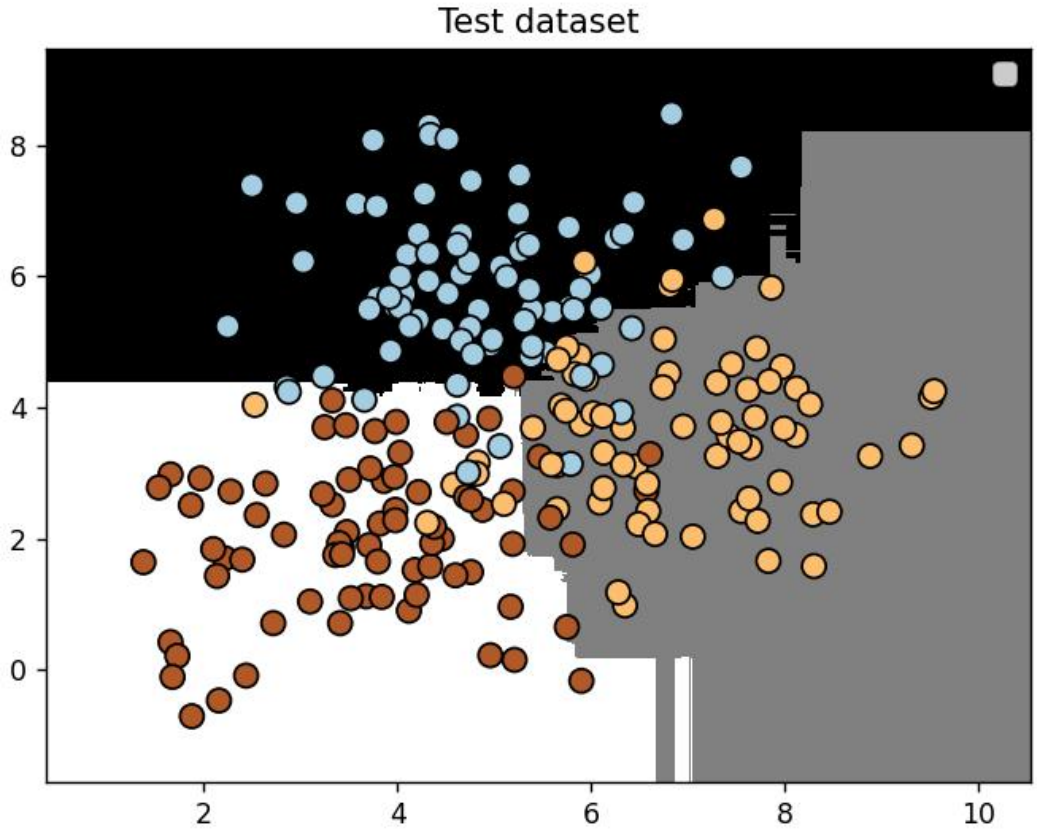

Рис. 3. Результат виконання програми

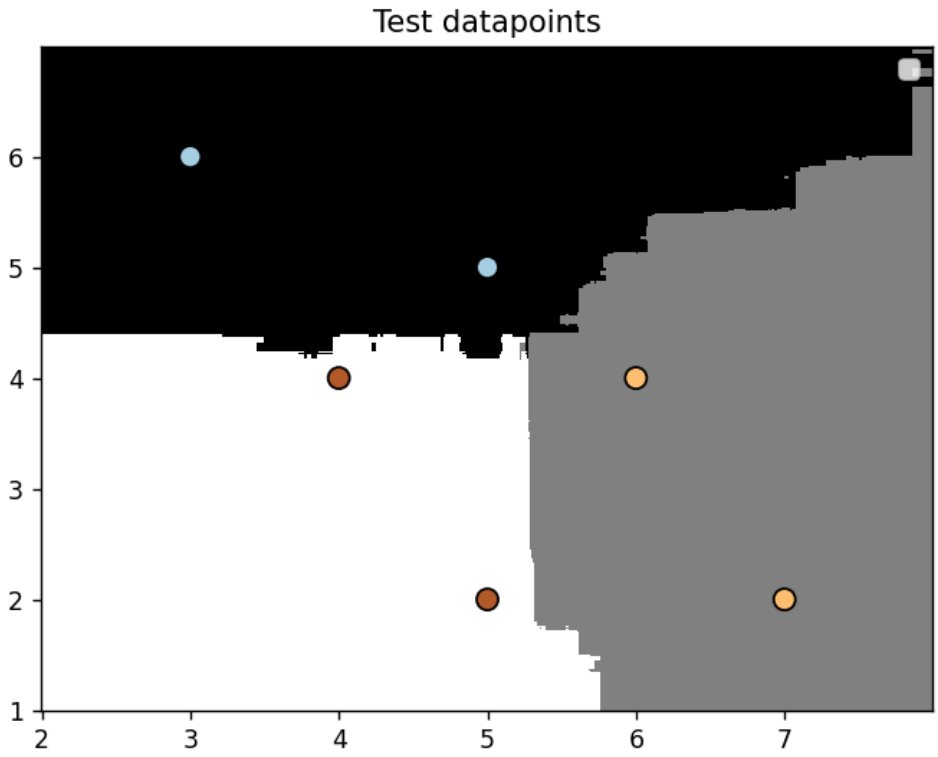

Рис. 4. Результат виконання програми

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

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. 5. Результат виконання програми

```
Confidence measure:

Datapoint: [5 5]
Predicted class: Class- 0
Probabilities: [0.81427532 0.08639273 0.09933195]

Datapoint: [3 6]
Predicted class: Class- 0
Probabilities: [0.93574458 0.02465345 0.03960197]

Datapoint: [6 4]
Predicted class: Class- 1
Probabilities: [0.12232404 0.7451078  0.13256816]

Datapoint: [7 2]
Predicted class: Class- 1
Probabilities: [0.05415465 0.70660226 0.23924309]

Datapoint: [4 4]
Predicted class: Class- 2
Probabilities: [0.20594744 0.15523491 0.63881765]

Datapoint: [5 2]
Predicted class: Class- 2
Probabilities: [0.05403583 0.0931115  0.85285267]
```

Рис. 6. Результат виконання програми

Далі запустимо програму з прапорцем erf.

Результат виконання програми:

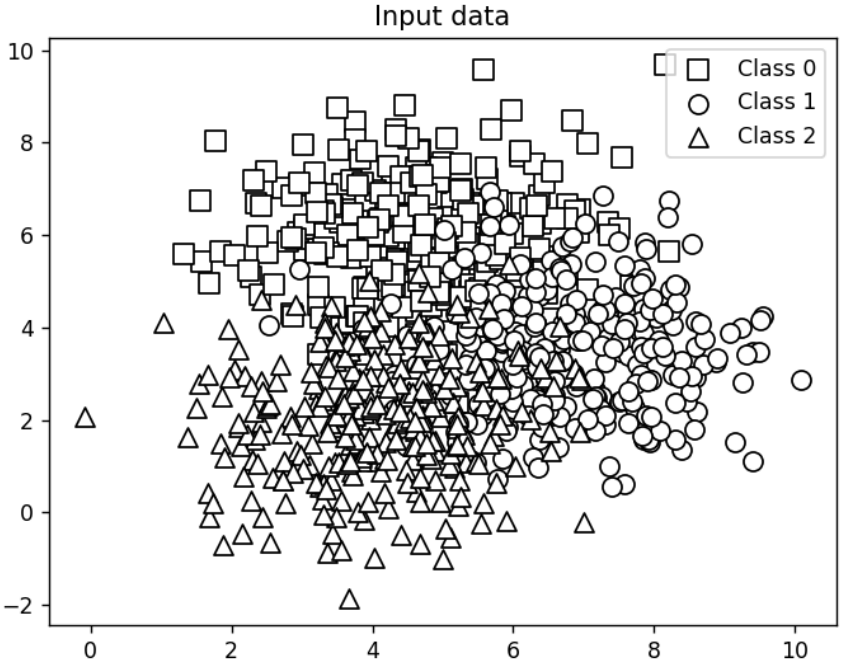

Рис. 7. Результат виконання програми

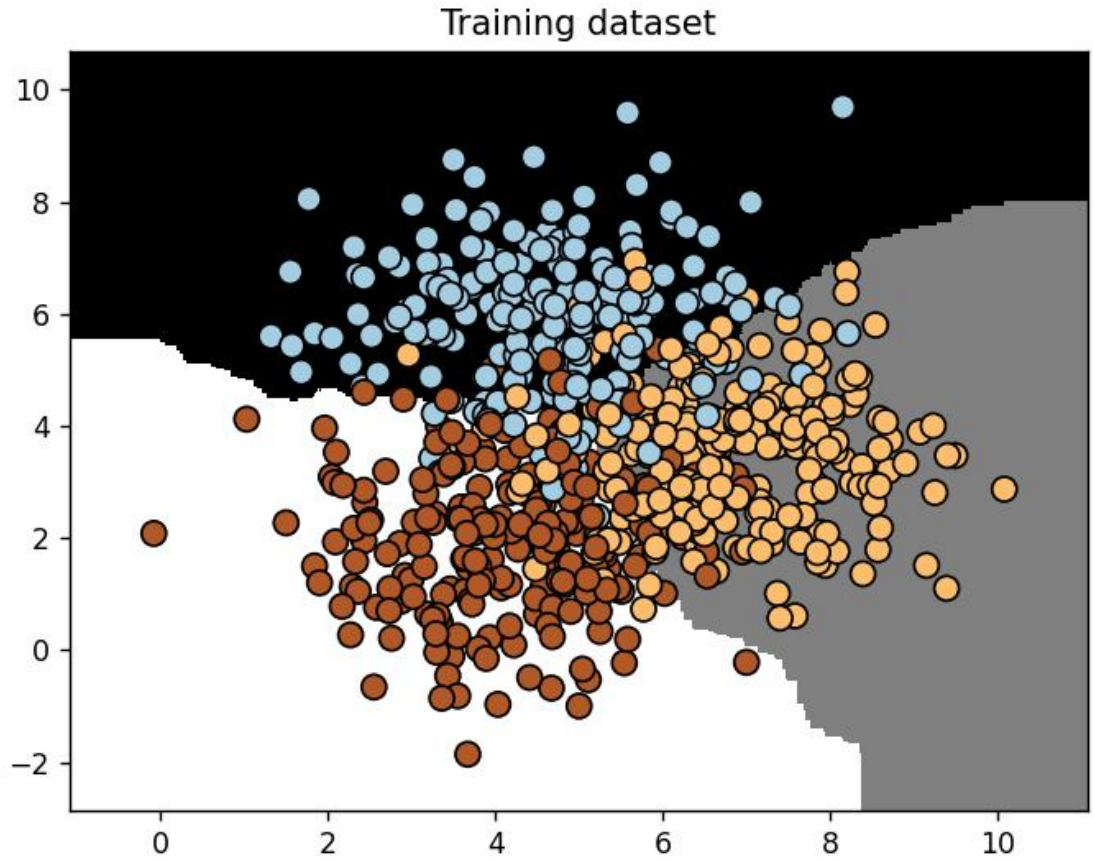

Рис. 8. Результат виконання програми

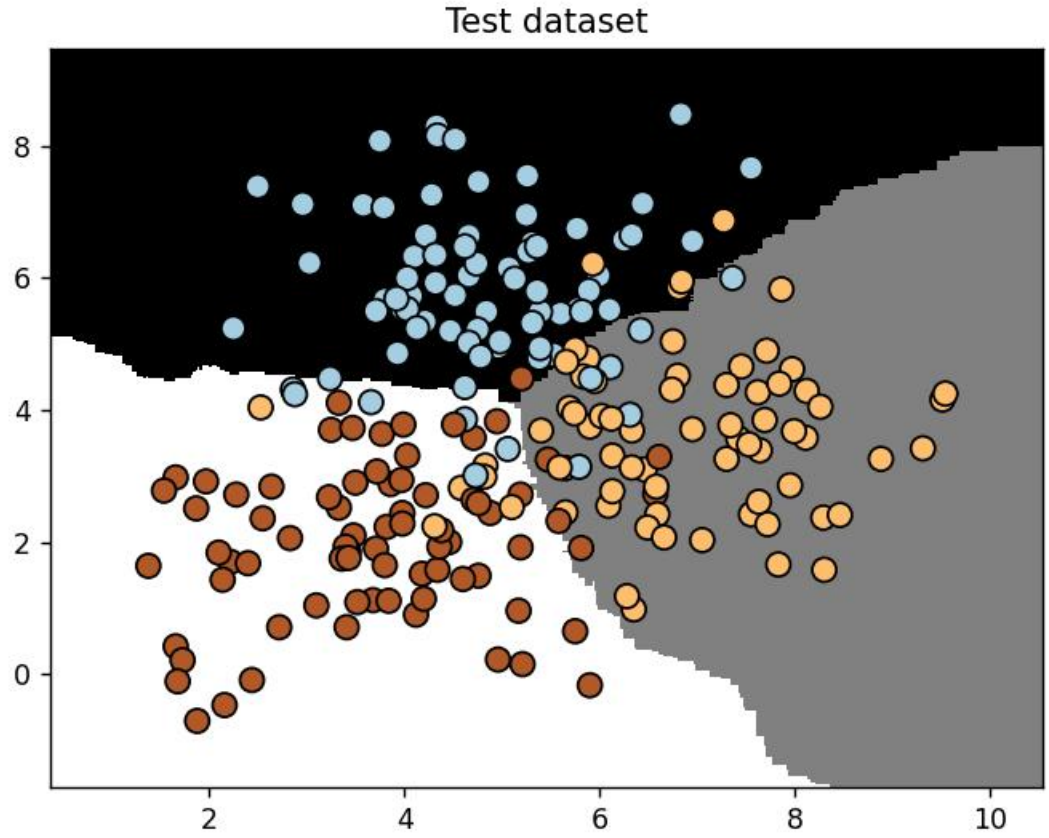

Рис. 9. Результат виконання програми

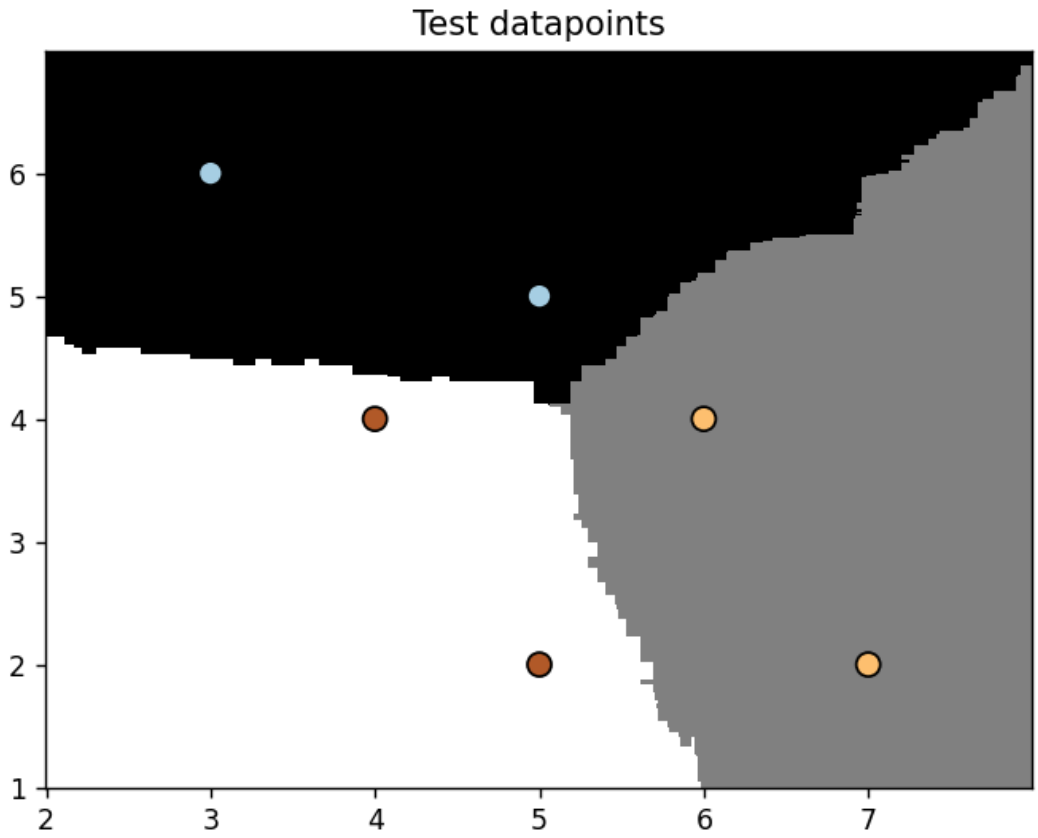

Рис. 10. Результат виконання програми

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

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. 11. Результат виконання програми

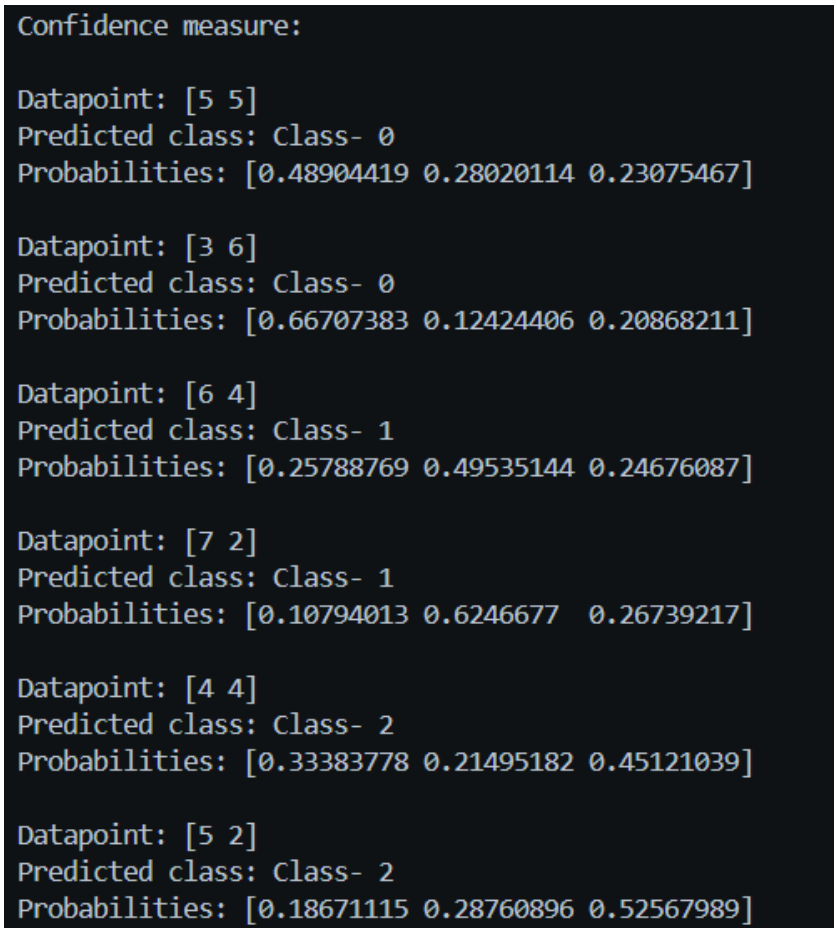

Рис. 12. Результат виконання програми

У завданні було реалізовано класифікатори двох типів: rf (випадковий ліс) і erf (гранично випадковий ліс). Ці типи є методами ансамблевого навчання, у яких індивідуальні моделі конструюються з використанням дерев рішень.

На рис. 1 і рис. 7 представлено графіки вхідних даних, які розділені на три класи (0, 1, 2), позначені різними маркерами. Точки на графіках частково перекривають одна одну, тому лінійні методи не змогли б добре розподілити дані.

Графік класифікації для rf (рис. 2 – 4) має три області, відповідно до трьох класів, межі яких мають «східчасту» форму. Графік класифікації для erf (рис. 8 – 10) аналогічно має три області, межі яких мають більш плавну форму порівняно з попереднім графіком. Це пов’язано з тим, що гранично випадкові ліси випадково вибирають і ознаки, і граничні значення, а тому зменшують варіативність моделі, що робить межі більш узагальненими.

За результатами вимірів продуктивності класифікаторів на навчальному та тестовому наборах даних (рис. 5, 11) можна зробити наступний висновок: моделі

непогано навчаються та класифікують дані, мають високу точність, ефективні для цього набору даних.

За результатами виведення мір впевненості (Confidence measure) для обох моделей (рис. 6, 12) можна сказати, що:

- модель rf досить «впевнено» класифікує точки, для кожної з них ймовірність спрогнозованого класу переважає над іншими доволі сильно;
- модель erf менш «впевнений» у класифікації точок, присутні невеликі розриви між ймовірностями кожної через плавні границі між класами.

Завдання 5.2. Обробка дисбалансу класів.

Лістинг програми:

```
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", label='Class 0')

plt.scatter(class_1[:, 0], class_1[:, 1], s=75,
    facecolors="white", edgecolors="black",
    linewidth=1, marker="o", label='Class 1')

plt.title("Input Data")
plt.legend()
plt.show()

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")
```

Спершу запустимо програму без балансування.

Результат виконання програми:

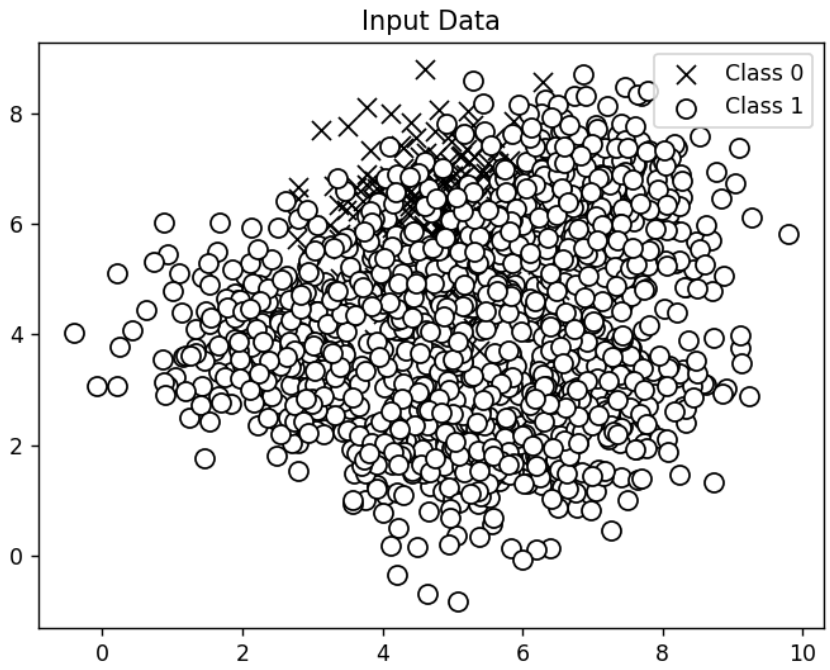

Рис. 13. Результат виконання програми

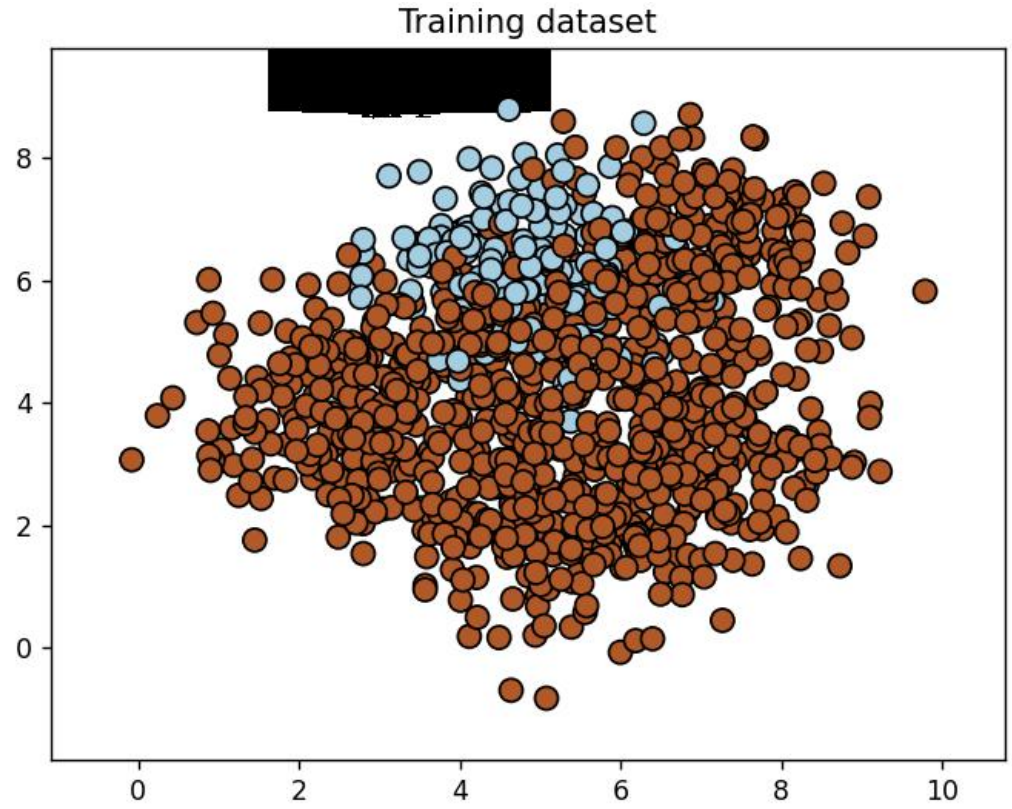

Рис. 14. Результат виконання програми

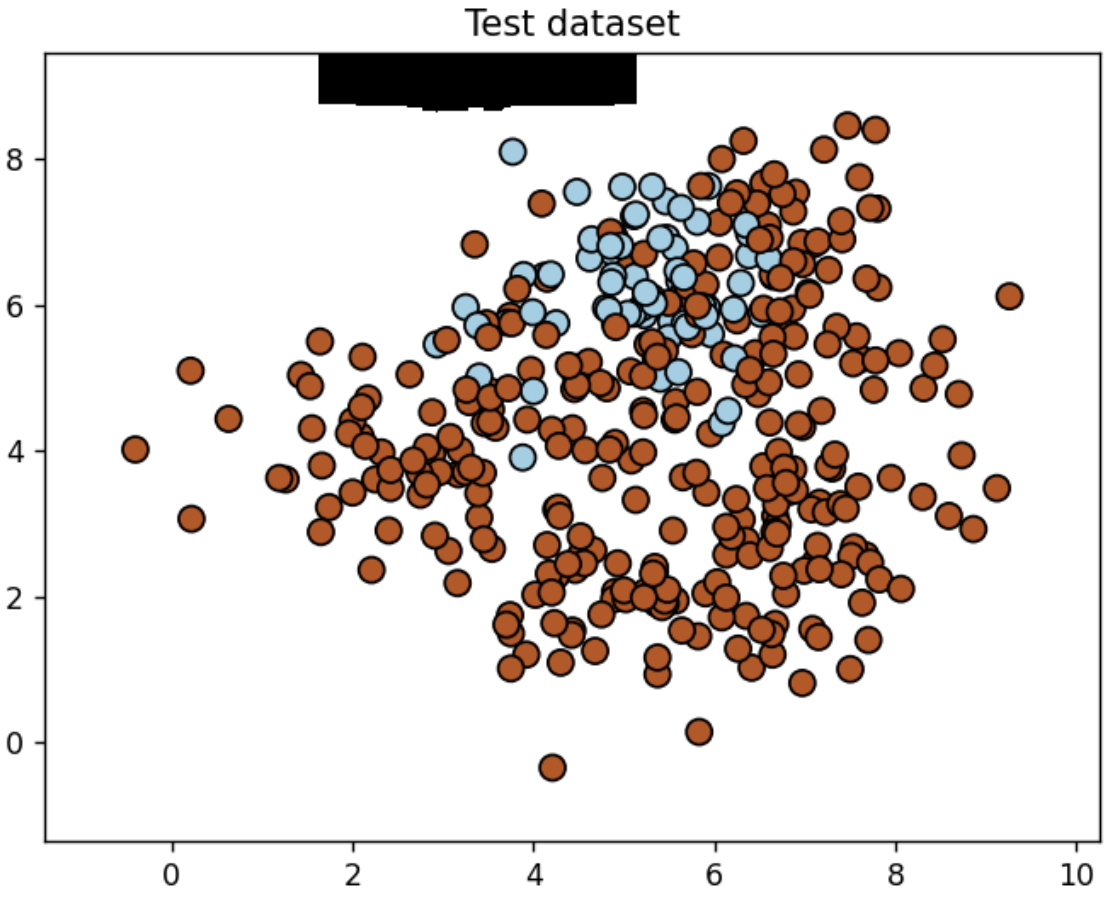

Рис. 15. Результат виконання програми

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

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

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

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

Classifier performance on test dataset

              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. 16. Результат виконання програми

Тепер запустимо програму з балансуванням.

Результат виконання програми:

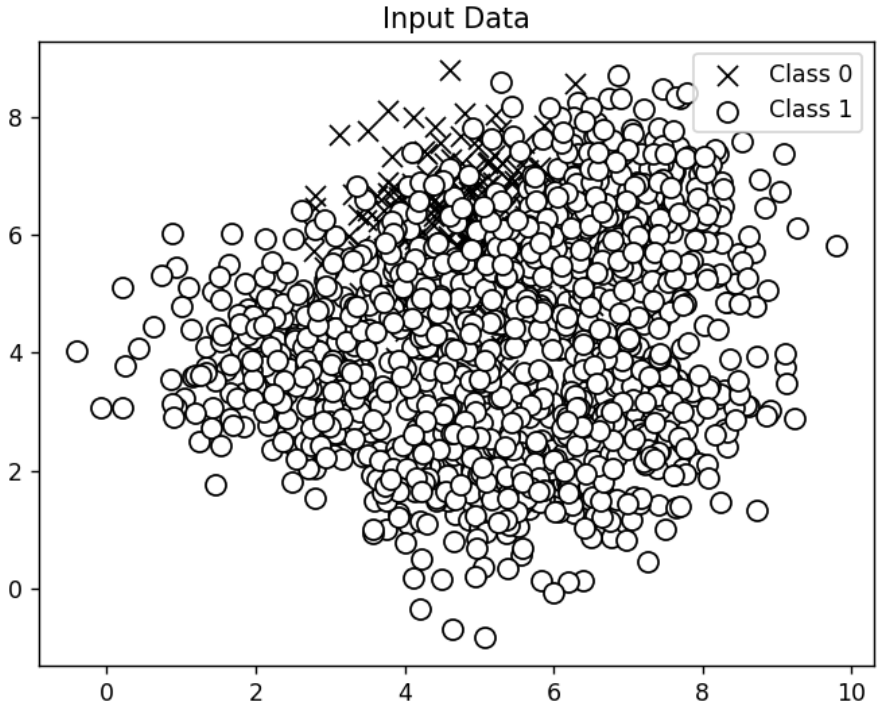

Рис. 17. Результат виконання програми

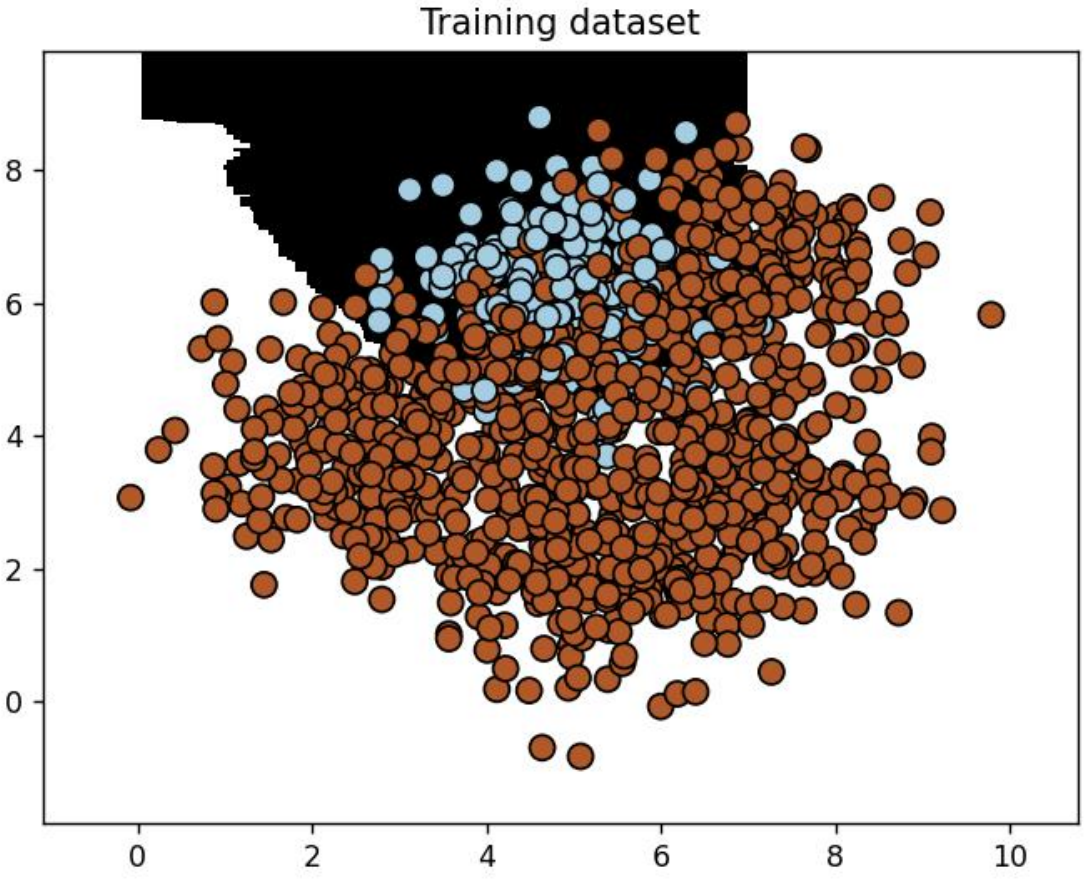

Рис. 18. Результат виконання програми

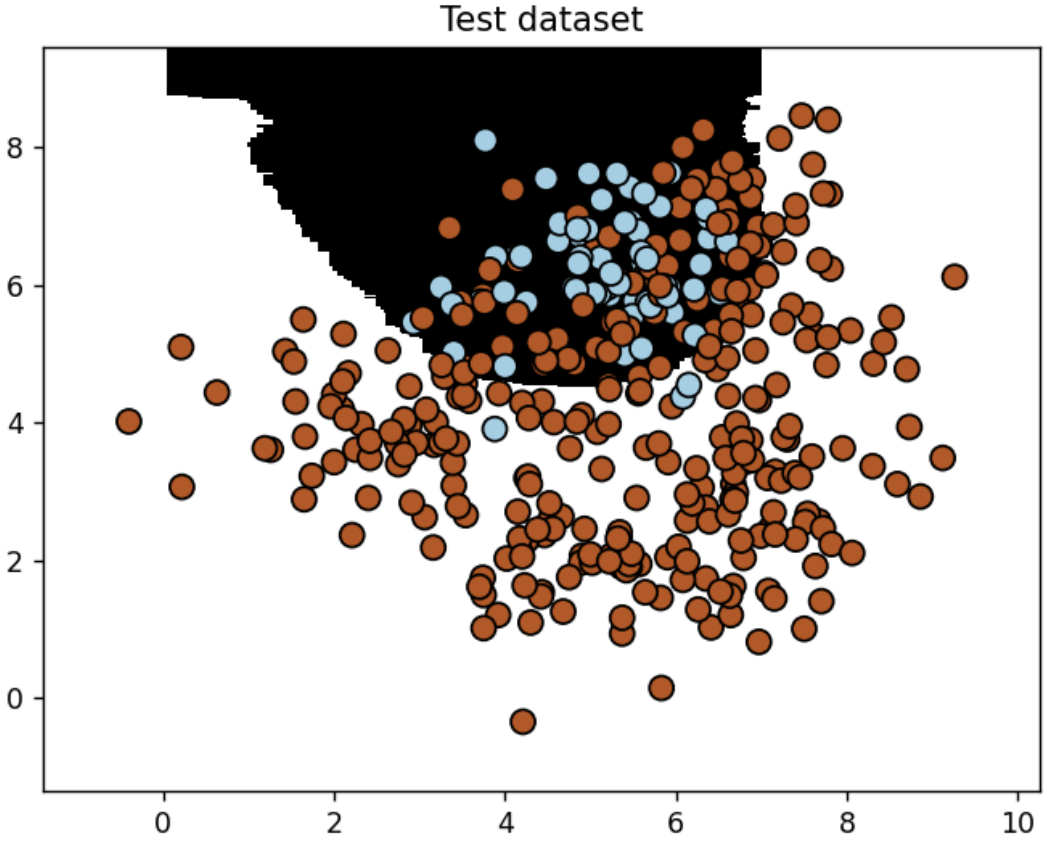

Рис. 19. Результат виконання програми

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

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. Результат виконання програми

У завданні було досліджено проблему дисбалансу класів, коли кількість точок даних для першого класу переважає над кількістю для другого.

При запуску програми без балансування ми отримали некоректний результат, оскільки модель віддавала перевагу більш численному класу. На графіках (рис. 14 – 15) це підтверджується зміщеними межами областей. За результатами виведення вимірів продуктивності класифікатора на навчальному та тестовому наборах даних (рис. 16):

- клас 0 (меншість) майже повністю ігнорується моделлю, його значення recall = 0.01 під час навчання і 0.00 під час тестування, тобто модель не вміє розпізнавати цей клас.

- клас 1 (більшість) має recall = 1.0, тому модель ідеально розпізнає цей його.
- значення accuracy = 0.82 є хибним, оскільки більшість точок, які розпізнаються належать до класу 1.

При запуску програми з балансуванням ми отримали правильний результат, оскільки ваги класів були збалансовані пропорційно до кількості точок у кожному з класів. Області класів на графіках (рис. 18 – 19) стали більш пропорційні до даних. За результатами виведення вимірів продуктивності класифікатора на навчальному та тестовому наборах даних (рис. 20):

- розпізнавання класу 0 (меншість) моделлю значно покращилося, його значення recall = 0.93 і 0.94 під час навчання і тестування відповідно.
- у класа 1 (більшість) трохи погіршився recall = 0.77 і 0.74.
- значення accuracy = 0.78 несуттєво знизилося, але тепер модель стала збалансованою та ефективнішою для даного датасету.

Завдання 5.3. Знаходження оптимальних навчальних параметрів за допомогою сіткового пошуку.

Лістинг програми:

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

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]},
    {"n_estimators": [25, 50, 100, 250], "max_depth": [4]},
]
```

```
metrics = ["precision_weighted", "recall_weighted"]
for metric in metrics:
    print(f"\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))
```

Результат виконання програми:

```
##### 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
```

Рис. 21. Результат виконання програми

```
##### 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
```

Рис. 22. Результат виконання програми

У завданні було знайдено оптимальні навчальні параметрів класифікатора ExtraTreesClassifier за допомогою сіткового пошуку. Для пошуку було використано параметри n_estimators і max_depth, визначено сітку їх значень та метричні характеристики precision_weighted та recall_weighted. GridSearchCV розглянув два словники в сітці, тобто протестував 9 комбінацій значень.

Для метрики precision_weighted (рис. 21) найкращою комбінацією є {'max_depth': 2, 'n_estimators': 100}. Для метрики recall_weighted (рис. 22) оптимальними є дві метрики {'max_depth': 2, 'n_estimators': 100} --> 0.843 і {'max_depth': 4, 'n_estimators': 25} --> 0.843, оскільки мають однаковий результат, але у таких випадках алгоритм автоматично обирає першу комбінацію у сітці.

У нашому випадку для обох метрик отримано однакові оптимальні комбінації, але зазвичай для різних метричних характеристик відрізняються і результати оптимальних навчальних параметрів.

Отже, за допомогою сіткового пошуку можна знаходити найкращі комбінації параметрів для цільових метрик, які необхідно оптимізувати.

Завдання 5.4. Обчислення відносної важливості ознак.

Лістинг програми:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.datasets import fetch_openml
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_openml(name="boston", version=1, as_frame=True)

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 = np.array(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(figsize=(10, 5))
plt.bar(pos, feature_importances[index_sorted], align='center')
plt.xticks(pos, feature_names[index_sorted])
plt.ylabel("Relative Importance")
plt.title("Feature importance estimation using the AdaBoost regressor")
plt.show()
```

Результат виконання програми:

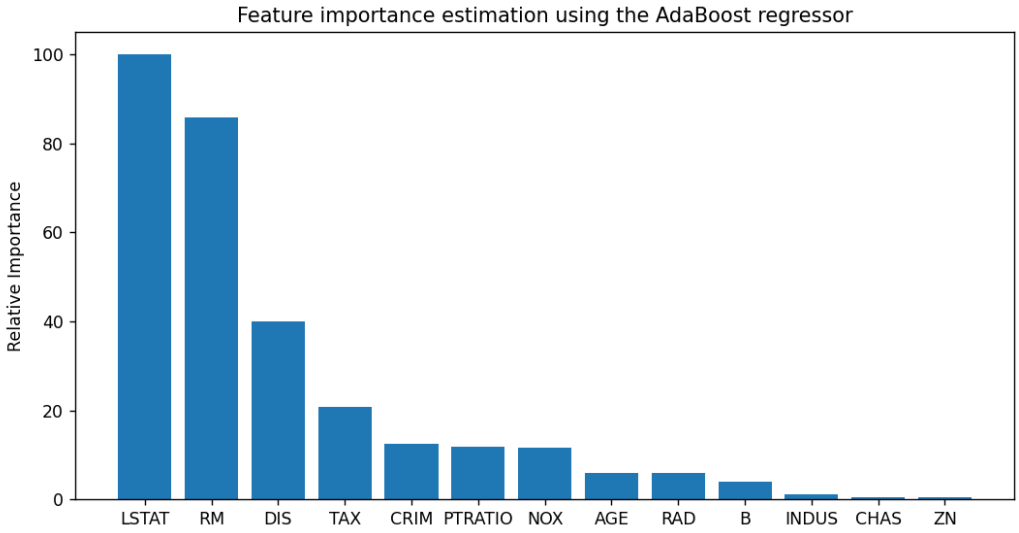

Рис. 23. Результат виконання програми

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

Рис. 24. Результат виконання програми

У завданні було проведено аналіз відносної важливості ознак, які впливають на ціну житла в Бостоні. На діаграмі (рис. 23) зображено ознаки та рівень їхньої важливості у %. Найважливішою ознакою для моделі є LSTAT (відсоток населення в районі, який належить до нижчого соціального класу), вона має значення 100%. Другою за важливістю є RM (середня кількість кімнат) ≈ 85%. DIS (відстань до центрів роботи) ≈ 40% також значно впливає на ціну житла. Найменш важливими ознаками є CHAS і ZN < 1%. За допомогою отриманих результатів можна спростити модель, залишивши ті ознаки, що мають відносну важливість понад ~10%. Це допоможе пришвидшити навчання моделі без істотної втрати точності.

Метрики ефективності регресора (рис. 24):

- Mean Squared Error = 22.7 має невелике значення і вказує на те, що відхилення прогнозованих значень від реальних даних незначні.
- Explained Variance Score (EVS) = 0.79 означає те, що модель добре узгоджується з даними, а тому дає якісний прогноз.

Завдання 5.5. Прогнозування інтенсивності дорожнього руху за допомогою класифікатора на основі гранично випадкових лісів.

Лістинг програми:

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

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]])[0])
        count = count + 1
```

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

Результат виконання програми:

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

Рис. 25. Результат виконання програми

У завданні було використано ExtraTreesRegressor, який прогнозує кількість транспортних засобів, що проїжджають дорогою. Метрика Mean Absolute Error (MAE) = 7.42, тобто в середньому прогнозовані значення відрізняються від реальних на ~ 7.42 одиниць або ж ~ 7 – 8 транспортних засобів (оскільки вони неподільні). Це досить помірна точність прогнозування.

Рядкові ознаки (день тижня, час доби, команда суперника та наявність матчу) успішно кодуються за допомогою LabelEncoder, щоб модель могла враховувати різні комбінації ознак, представлені у числовому вигляді, у прогнозуванні кількості транспортних засобів.

Модель було протестовано на точці даних ['Saturday', '10:20', 'Atlanta', 'no']. Вона спрогнозувала 26 транспортних засобів (рис. 25) і цей результат є ідентичним тому, що зазначений у методичних рекомендаціях до лабораторної роботи.

Отже, регресор на основі гранично випадкових лісів підходить для цих даних, оскільки ефективно працює з категоріальними та числовими ознаками і не перенавчається завдяки використанню різних випадкових підмножин.

Висновок: в ході виконання лабораторної роботи ми дослідили методи ансамблів у машинному навчанні, використовуючи спеціалізовані бібліотеки та мову програмування Python.

Репозиторій: https://github.com/SofiiaKozlyk/AI-systems