ЛАБОРАТОРНА РОБОТА № 5

ДОСЛІДЖЕННЯ МЕТОДІВ АНСАМБЛЕВОГО НАВЧАННЯ

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

Хід роботи

GitHub репозиторій: https://github.com/VovaYanko/SAI

Завдання 1

Код програми:

```
import argparse
from cgi import test
import numpy as np
import matplotlib.pyplot as plt
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

def build_arg_parser():
    par-
ser = argparse.ArgumentParser(description='Classify data using Ensemble Learni
ng techniques')
    parser.add_argument('--classifier-
type', dest='classifier_type', required=True,
        choic-
es=['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])

    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('Вхідні дані')

    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)
    visual-
ize_classifier(classifier, X_train, y_train, 'Тренувальний набір даних')

    y_test_pred = classifier.predict(X_test)
    visualize_classifier(classifier, X_test, y_test, 'Тестовий набір даних')

    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_n
ames = 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_name
s))
    print("#"*40 + "\n")

    test_datapoints = np.array([[5, 5], [3, 6], [6, 4], [7, 2], [4, 4], [5, 2]
])
    print('\nConfidense 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)
```

```
    visual-
ize_classifier(classifier, test_datapoints, [0] * len(test_datapoints), 'Тестові точки даних')
    plt.show()
```

Отримані графіки та результати для випадкового лісу наведені на рисунках 1.1-1.6.

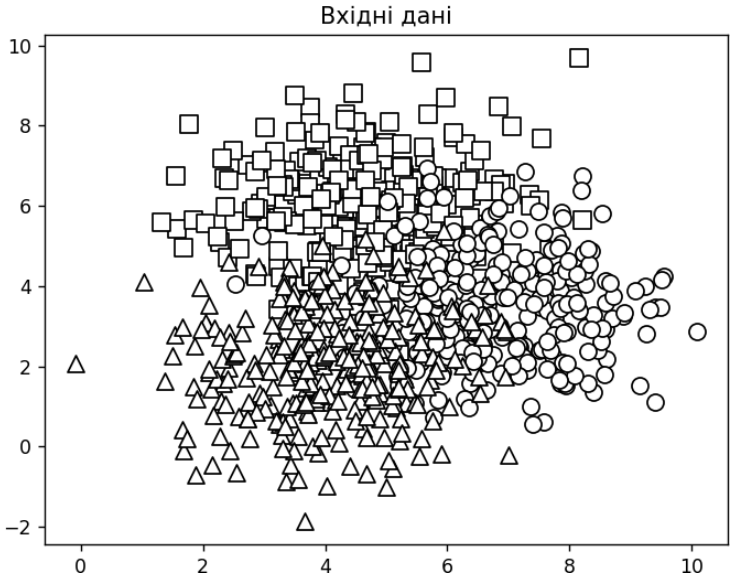

Рис. 1.1. Графік розподілу вхідних даних

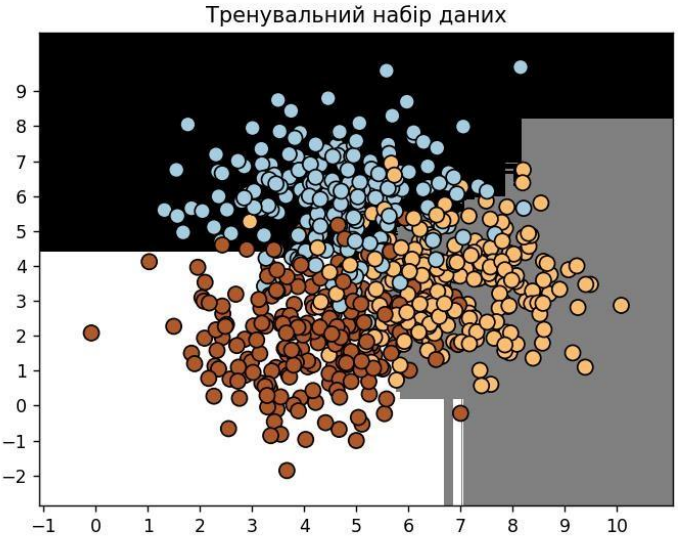

Рис. 1.2. Графік результату навчання тренувального набору

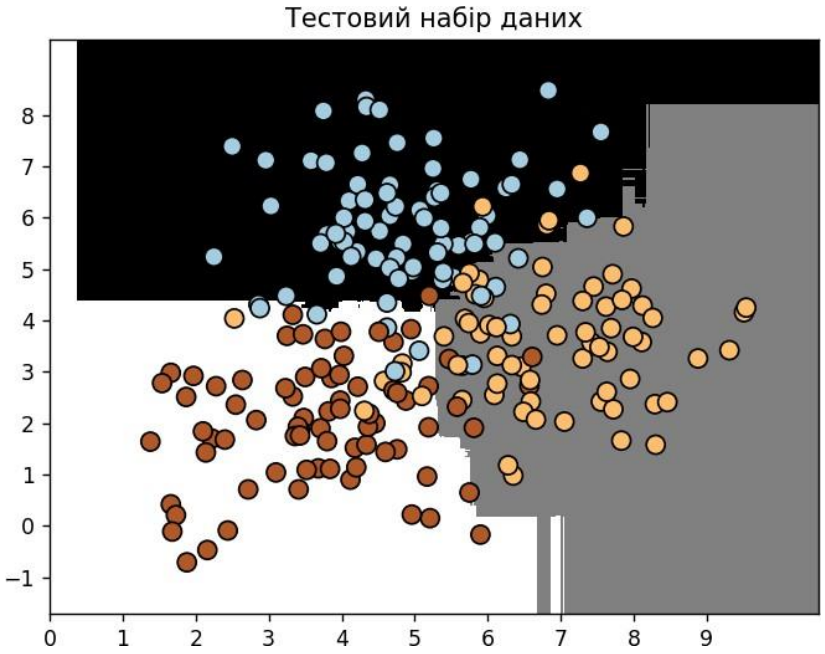

Рис. 1.3. Графік результату навчання тестувального набору

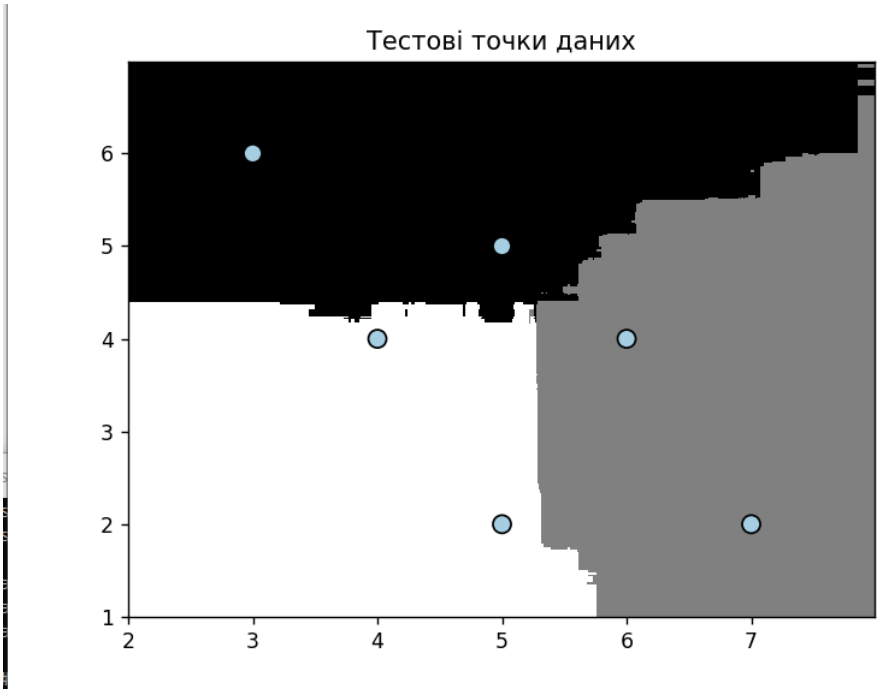

Рис. 1.4. Графік результату навчання для тестових точок

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

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. 1.5. Результат класифікатора для тренувальних та тестових наборів даних

```
Confidense 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. 1.6. Результат передбачення класів на тестових точках

Отримані графіки та результати для гранично випадкового лісу наведені на рисунках 1.7-1.12.

Рис. 1.7. Графік розподілу вхідних даних

Рис. 1.8. Графік результату навчання тренувального набору

Рис. 1.9. Графік результату навчання тестувального набору

Рис. 1.10. Графік результату навчання для тестових точок

Рис. 1.11. Результат класифікатора для тренувальних та тестових наборів даних

Рис. 1.12. Результат передбачення класів на тестових точках

В результаті виконання даного завдання було досліджено два види класифікаторів: випадкового лісу та гранично випадкового лісу, наочно представлено їх ефективність у вигляді графіків, результатів класифікації та метрик.

Завдання 2

Код 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, facecolor = 'black',
    edgecolors = 'black', linewidths = 1, marker = 'x')

plt.scatter(class_1[:, 0], class_1[:, 1], s = 75, facecolor = 'white',
    edgecolors = 'black', linewidths = 1, marker = 'o')

plt.title('Вхідні дані')

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, ra
ndom_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, 'cla
ss_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, 'Тренувальний набір даних')
```

```
y_test_pred = classifier.predict(X_test)
visualize_classifier(classifier, X_test, y_test, 'Тестовий набір даних')

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()
```

Отримані графіки та результати для гранично випадкового лісу на дисбалансних даних наведені на рисунках 1.13-1.16

Рис. 1.13. Графік розподілу вхідних даних

Рис. 1.14. Графік результату навчання тренувального набору

Рис. 1.15. Графік результату навчання тестувального набору

Рис. 1.16. Результат класифікатора для тренувальних та тестових наборів даних

Отримані графіки та результати для гранично випадкового лісу на дисбалансних даних з врахуванням дисбалансу наведені на рисунках 1.17-1.19

Рис. 1.17. Графік розподілу вхідних даних

Рис. 1.18. Графік результату навчання тренувального набору

Рис. 1.19. Графік результату навчання тестувального набору

Рис. 1.20. Результат класифікатора для тренувальних та тестових наборів даних

В результаті виконання даного завдання було досліджено вплив враування дисбалансу в гранично випадковому лісі при використанні несбалансованих даних.

Завдання 3

Код програми:

```
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]
```

```
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, ra
ndom_state = 5)

parameter_grid = [ {'n_estimators': [100], 'max_depth': [2, 4, 7, 12, 17] },
    {'max_depth': [4], 'n_estimators': [25, 50, 100, 250] }]

metrics = ['precision_weighted', 'recall_weighted']

for metric in metrics:
    print('\n#### Searching for optimal parameters for', metric)

    classifi-
er = 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 range(0, len(classifier.cv_results_['params'])):
        print(classifier.cv_results_['params'][i], '--
>', classifier.cv_results_['rank_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 пошуку оптимальних параметрів (рис. 1.21-1.22).

Рис. 1.21. Результат пошуку оптимальних параметрів

Рис. 1.22. Результат пошуку оптимальних параметрів

Під час виконання даного завдання ми досліили процес оптимізації параметрів класифікатора у відповідності до певної метрики.

Завдання 4

Код програми:

```
from ctypes import util
import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn import datasets, preprocessing, utils
from sklearn.metrics import mean_squared_error, explained_variance_score
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle

label_encoder = preprocessing.LabelEncoder()
housing_data = datasets.load_boston()
X, y = shuffle(housing_data.data, label_encoder.fit_transform(housing_data.tar
get), random_state = 7)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, ran
dom_state = 7)

regres-
sor = 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 squarred error =', round(mse, 2))
print('Explained variance error =', 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('Оцінка важливості ознак з використанням регресора AdaBoost')
plt.show()
```

Рис. 1.23. Діаграма оцінки важливості ознак

Рис. 1.24. Результат виконання програми

Відповідно до отриманої діаграми, найбільш важливими ознаками є LSTAT (відсоток малозабезпеченого населення) та RM (середня кількість кімнат), а знехтувати можна CHAS (чи межує з річкою).

В результаті виконання даного завдання ми навчилися аналізувати важливість характеристик датасету за допомогою регресора AdaBoost.

Завдання 5

Код програми:

```
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 ExtraTreesClassifier
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, ra
ndom_state = 5)
```

```
params = { 'n_estimators': 200, 'max_depth': 15, 'random_state': 0 }
regressor = ExtraTreesClassifier(**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:
        encoder = label_encoder[count]
        test_datapoint_encoded[i] = int(encoder.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]
))
```

Рис. 1.25. Результат виконання програми

Висновки: на даній лабораторній роботі ми дослідили методи ансамблів у машинному навчанні використовуючи спеціалізовані бібліотеки та мову програмування Python.