

ЛАБОРАТОРНА РОБОТА № 5

ДОСЛІДЖЕННЯ МЕТОДІВ АНСАМБЛЕВОГО НАВЧАННЯ

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

Сахно Андрій, 18 варіант

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.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier
from sklearn.metrics import classification_report
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', 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__':
    # Вилучення вхідних аргументів
    args = build_arg_parser().parse_args()
    classifier_type = args.classifier_type

    # Завантаження вхідних даних
    input_file = '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])

    # Віртуалізація вхідних даних
    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('Input data')

# 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}
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, 'Тестовий набір даних')

# Перевірка роботи класифікатора
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")

# 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('\nDatapoint:', datapoint)
    print('Predicted class:', predicted_class)

# Віртуалізація точок даних
visualize_classifier(classifier, test_datapoints,
[0]*len(test_datapoints), 'Тестові точки даних')
plt.show()

```

python random_forests.py --classifier-type rf

```
PS C:\Users\User\Desktop\ua-un\7th semester\CWI\projs\lab5> python random_forests.py --classifier-type rf
```

```
#####
```

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

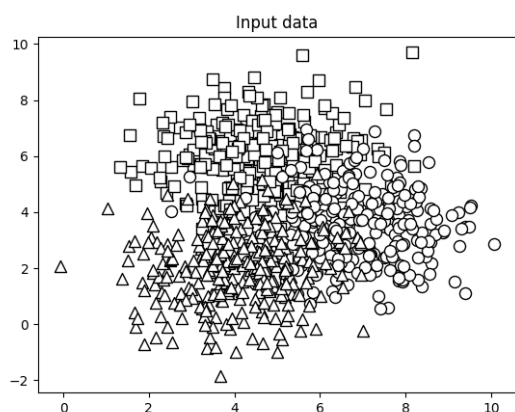
Datapoint: [4 4]

Predicted class: Class-2

Datapoint: [5 2]

Predicted class: Class-2

Figure 1




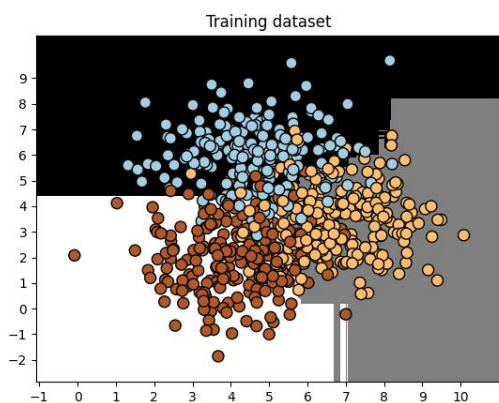
 (x, y) = (-0.18, 5.85)

Figure 2




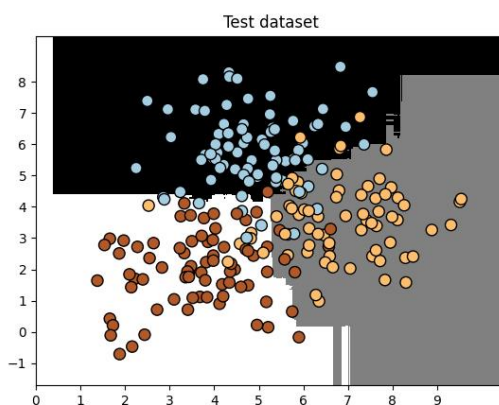

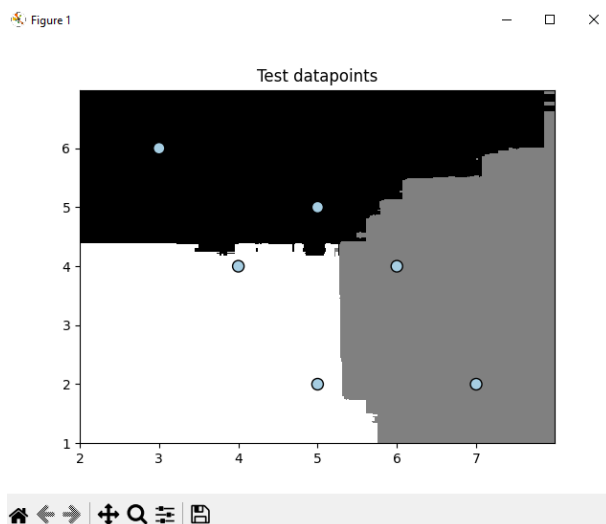


Figure 1







Використовуючи випадковий ліс (Random Forest) для класифікації даних, модель досягла хороших результатів як на навчальному, так і на тестовому наборах. На навчальному наборі точність моделі становить 87%, з високими показниками для всіх класів: клас 0 — 91% точність, клас 1 — 84%, клас 2 — 86%. Для тестового набору точність класифікації також складає 87%, що вказує на хорошу здатність моделі до узагальнення. Показники precision, recall та f1-score для кожного класу майже однакові, що свідчить про збалансовану класифікацію для всіх трьох класів.

Крім того, рівні довіри для кожної точки даних показують точне передбачення класу, що підтверджує ефективність моделі. Для всіх тестових точок, модель правильно визначила клас з високою ймовірністю. Отже, класифікатор на основі випадкового лісу продемонстрував стабільну роботу і хороші результати при класифікації трьох класів.

python random_forests.py --classifier-type erf

```
PS C:\Users\User\Desktop\ua-un\7th semester\CWI\artificial-intelligence\lab5> python random_forests.py --classifier-type erf

#####

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          0.85          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          0.87          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
```

Figure 1

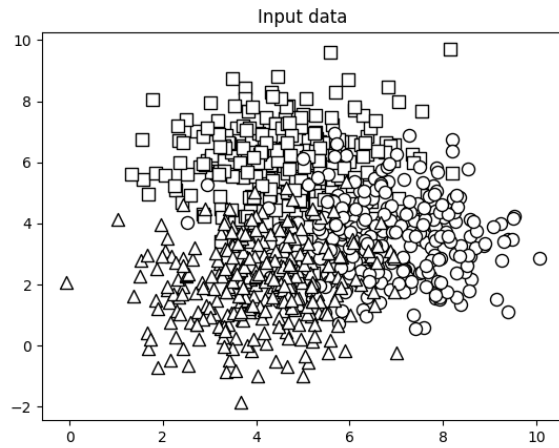


Figure 2

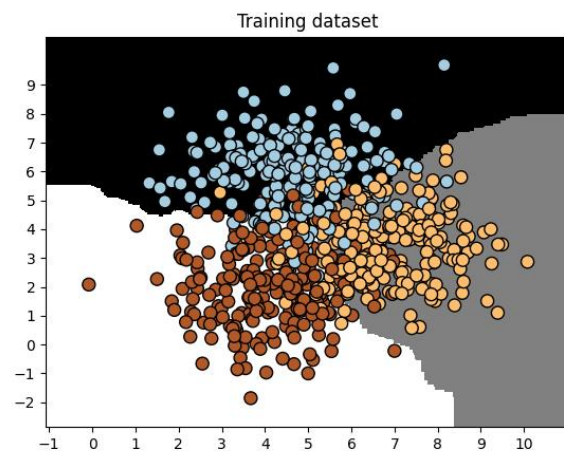
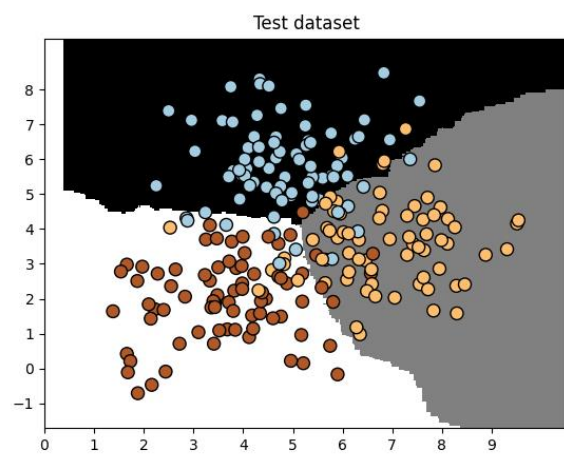
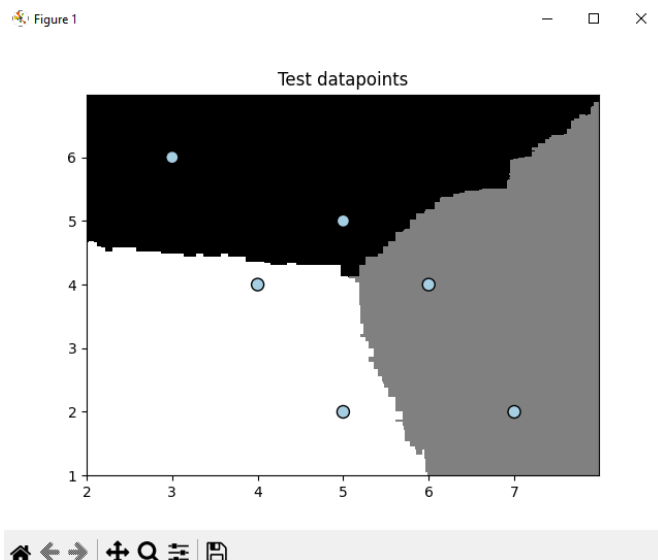


Figure 1





Використовуючи гранично випадковий ліс (Extremely Randomized Forest) для класифікації даних, модель показала добрі результати, хоча й трохи гірші порівняно з випадковим лісом. Точність на навчальному наборі становить 85%, з дещо нижчими показниками для класів: клас 0 — 89% точність, клас 1 — 82%, клас 2 — 83%. Для тестового набору точність класифікації складає 87%, що вказує на хорошу здатність моделі до узагальнення, хоча і з деякими відмінностями в порівнянні з результатами випадкового лісу. Показники precision, recall та f1-score на тестовому наборі для класів 0 і 2 мають майже однакові значення, але для класу 1 спостерігається невелике зниження. Рівні довіри для кожної точки даних також підтверджують правильність прогнозів, хоча загалом точність моделі на тестових точках виявилася дещо меншою. Таким чином, класифікатор на основі гранично випадкового лісу продемонстрував добрі результати, хоча і з меншим рівнем точності на навчальних даних порівняно з випадковим лісом.

Завдання 2.2. Обробка дисбалансу класів

```
import sys
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
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, '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()

```

python -W ignore LR_5_task_2.py

```
PS C:\Users\User\Desktop\va-un\7th semester\CWI\artificial-intelligence\lab5> python -W ignore LR_5_task_2.py
```

```
#####
```

```
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

```
#####
```

```
#####
```

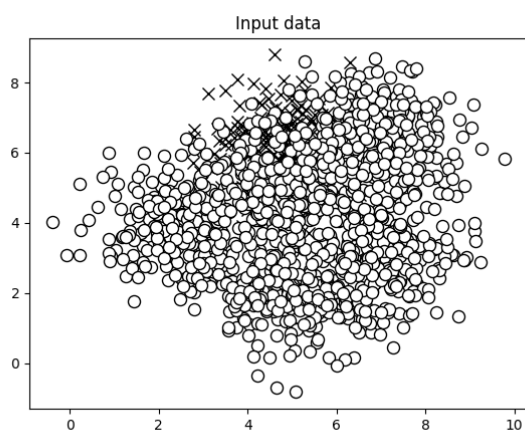
```
Classifier performance on test dataset
```

	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

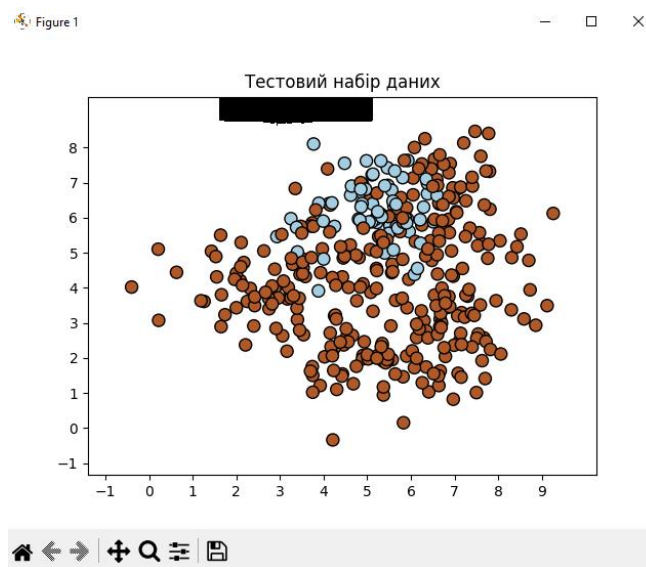
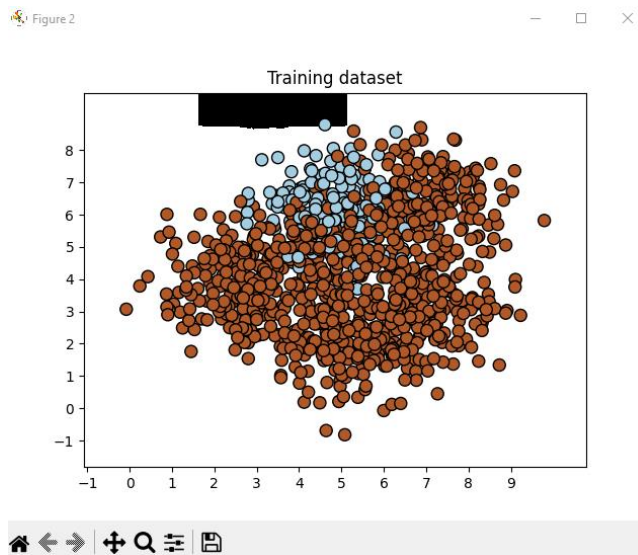
```
#####
```

Figure 1

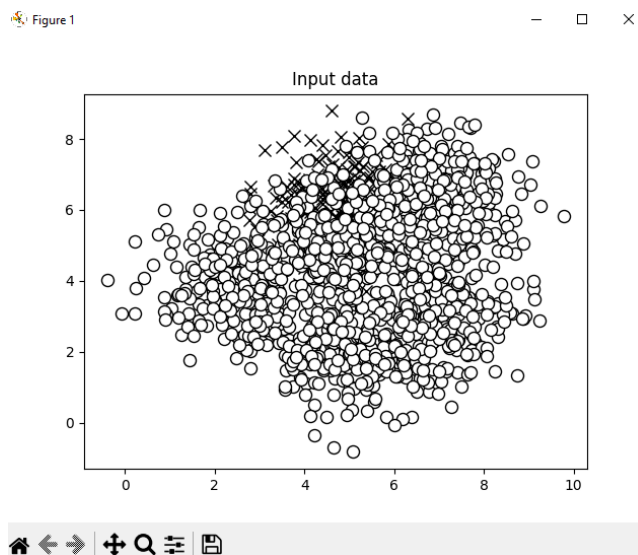
— □ ×

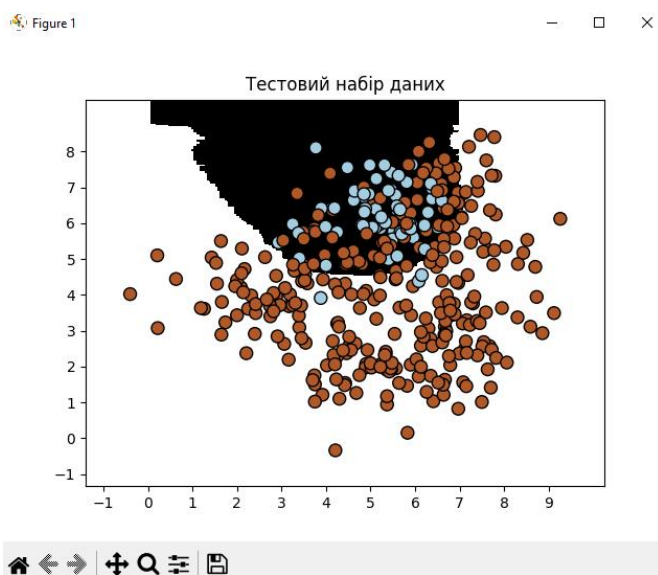
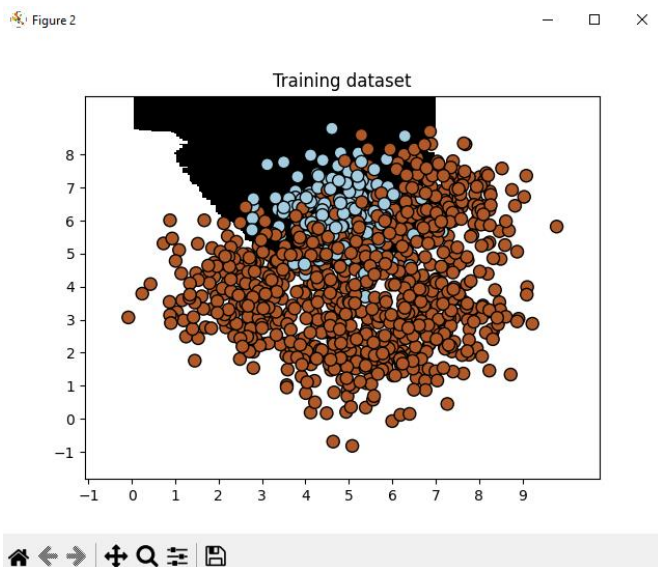


Navigation icons: home, back, forward, search, zoom, and save.



python LR_5_task_2.py balance





Результати класифікації показують, що модель без балансування класів демонструє високу точність для класу 1 (0.98) та низьку для класу 0 (0.44), що свідчить про сильний дисбаланс у даних. При використанні параметра `'class_weight="balanced"'` точність для класу 0 зростає до 0.45, а для класу 1 знижується до 0.98. Це забезпечує кращу рівновагу між класами, хоча загальна точність для тестового набору залишається на рівні 0.78, що все ще вказує на деякі труднощі в класифікації через дисбаланс.

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

```
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, GridSearchCV

from utilities import visualize_classifier

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:")
    means = classifier.cv_results_['mean_test_score']
    params = classifier.cv_results_['params']
    for mean, param in zip(means, params):
        print(param, '-->', round(mean, 3))
    print("\nBest parameters:", classifier.best_params_)

y_pred = classifier.predict(X_test)
print("\nPerformance report:\n")
print(classification_report(y_test, y_pred))
```

```

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}

```

```

#### 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

У результаті сіткового пошуку для параметрів `n_estimators` і `max_depth` для метрик `precision_weighted` та `recall_weighted` найкращими виявилися параметри `{'max_depth': 2, 'n_estimators': 100}`. Обидва підбори показали схожі результати, де точність для різних комбінацій варіювалася в межах 0.81-0.85, а відносно значень `recall` — в межах 0.83-0.84.

На основі звіту про ефективність класифікатора, точність (precision) та відзив (recall) для трьох класів мають досить високе значення, зокрема для класу 0 точність досягає 0.94, а recall — 0.81. Для класу 1 і 2 також досягаються хороші результати, зокрема для класу 2 — 0.91 recall. Загальна точність класифікатора становить 0.86, що свідчить про хорошу роботу моделі з урахуванням обраних параметрів.

Вибір параметрів `max_depth=2` та `n_estimators=100` забезпечив найкраще співвідношення між точністю та відзивом для обраних метрик.

Завдання 2.4. Обчислення відносної важливості ознак

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
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.model_selection import train_test_split
from sklearn.utils import shuffle

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, 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
```

```
feature_names = np.array(housing_data.feature_names)

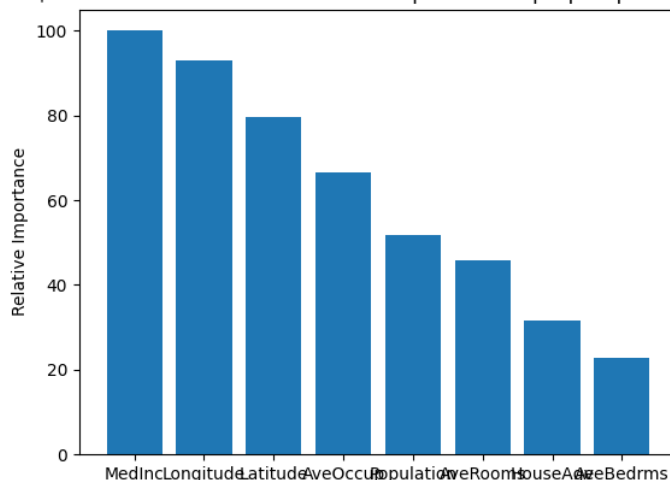
plt.figure()
plt.bar(pos, feature_importances[index_sorted], align='center')
plt.xticks(pos, feature_names[index_sorted])
plt.ylabel('Relative Importance')
plt.title('Оцінювання важливості ознак із використанням регресора AdaBoost')
plt.show()
```

```
C:\Users\User\AppData\Local\Programs\Python\Python313\python.exe "C:\Users\User\Desktop\ua-un\7th semester\CWI\artificial-intelligence\lab5\LR_5_task_4.py"

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

Process finished with exit code 0
```

Оцінювання важливості ознак із використанням регресора AdaBoost



Проаналізувавши діаграму, можна зробити висновок, що MedInc, Longitude, Latitude є найважливішими ознаками, а факторами якими ми можемо знехтувати відповідно є: AveBedrms, HouseAge, AveRooms, Population.

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

```
import numpy as np
from sklearn.metrics import classification_report, 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 += 1
test_datapoint_encoded = np.array(test_datapoint_encoded)
test_datapoint_encoded = test_datapoint_encoded.reshape(1, -1)

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

```

```

Mean absolute error: 7.42
Predicted traffic: 26

Process finished with exit code 0

```

Висновок: На лабораторній роботі, використовуючи спеціалізовані бібліотеки та мову програмування Python, дослідив різні методи регресії даних у машинному навчанні.

<https://github.com/andreylion06/artificial-intelligence>