Green University Of Bangladesh

Department Of Computer Science and Engineering (CSE)

Faculty of Sciences and Engineering

Semester: (Fall, Year: 2023), B.Sc. in CSE (DAY)

LAB REPORT NO - 06

Course Title: Data Mining Lab

Course Code: CSE-436 **Section:** D2

Lab Experiment Name: Machine Learning Classification: Decision trees using Python

Student Details

	Name	ID
1	Shamim Ahmed	201902067

Lab Date : 27th October 2023

Submission Date : 3rd November 2023

Course Teacher Name : Rezwanul Haque

Lab Report Status

Mark:..... **Signature:.....**

Comments:..... **Date:.....**

1 INTRODUCTION

In this lab report we are going to develop a Colab using Machine Learning Classification: Decision trees using Python. We will Choose an appropriate dataset Then we will implement the Machine Learning Classification: Decision trees without pruning and also with pruning and examined the result visualize with appropriate plot.

2 OBJECTIVE

The aim of this lab is to learn about Machine Learning Classification: Decision trees without pruning and with pruning.Also to understand the importance of pruning in decision trees and to evaluate its impact on the model's complexity, interpretability, and predictive performance. This experiment aims to showcase the practical implementation of decision tree pruning in a real-world dataset, providing insights into the process and its implications for decision tree-based machine learning models..

3 THEORY

3.1 Decision Tree

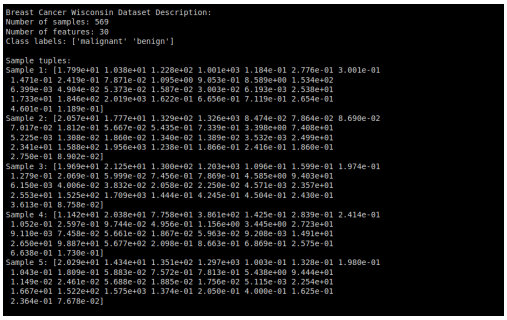
A decision tree is a supervised learning algorithm used for both classification and regression tasks. It works by recursively partitioning the data into subsets based on the value of a particular attribute. The tree is constructed using a top-down, greedy approach, where the most significant attribute is selected as the root node and the dataset is split based on its values. Pruning is a technique used to reduce the size of a decision tree by removing nodes that do not provide significant predictive power, thus preventing overfitting.

3.2 Pruning Reasons and Types

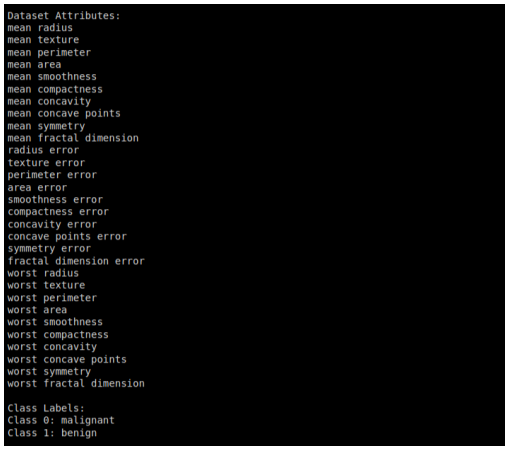
Pruning is essential to prevent overfitting, which occurs when a decision tree captures noise in the training data, leading to poor generalization to unseen data. Pruning helps to simplify the model, making it less complex and more interpretable, while maintaining or even improving its predictive performance. There are two types of pruning:

1. `Pre-Pruning:` In pre-pruning, the tree is pruned during the construction phase by setting constraints on the tree growth, such as setting a maximum depth or requiring a minimum number of samples to split a node. This helps avoid overfitting by stopping the tree from becoming too complex.

2. `Post-Pruning:` Post-pruning, also known as backward pruning or cost-complexity pruning, involves growing the tree to its maximum size and then pruning the nodes that provide little information gain. This is done by removing nodes with low impurity reduction or low information gain.



(a) Tuples



(b) Attributes and Class

Figure 1: Breast Cancer Wisconsin (Diagnostic) dataset

4 IMPLEMENTATION

4.1 DATASET

For this experiment, I used the "Breast Cancer Wisconsin (Diagnostic)" dataset, obtained from the UCI Machine Learning Repository. This dataset includes various features computed from a digitized image of a fine needle aspirate (FNA) of a breast mass, and the task is to predict whether the mass is benign or malignant. `Number of tuples:`569 `Number of attributes:`30 (including the class attribute) `Class:`M (Malignant) and B (Benign)

4.2 CODING

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.tree import DecisionTreeClassifier, export_graphviz,
    export_text
from IPython.display import Image, display
import graphviz
import matplotlib.pyplot as plt

# Load the Breast Cancer dataset
data = load_breast_cancer()
X, y = data.data, data.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    random_state=42)

# Initialize the unpruned Decision Tree Classifier
```

```
dt_classifier = DecisionTreeClassifier(random_state=42)

# Perform 15-fold cross-validation on unpruned classifier
cv_scores_before = cross_val_score(dt_classifier, X, y, cv=15)
mean_cv_score_before = cv_scores_before.mean()

# Train the unpruned Decision Tree classifier
dt_classifier.fit(X_train, y_train)

# Print the unpruned decision tree
print("Unpruned Decision Tree:")
text_representation_before = export_text(dt_classifier)
print(text_representation_before)

# Plot the unpruned decision tree
dot_data_before = export_graphviz(dt_classifier, out_file=None,
    feature_names=data.feature_names, class_names=data.target_names, filled
    =True, rounded=True, special_characters=True)
graph_before = graphviz.Source(dot_data_before)
display(Image(graph_before.pipe(format='png')))

# Apply pruning on the decision tree
path = dt_classifier.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas, impurities = path.ccp_alphas, path.impurities

clfs = []
for ccp_alpha in ccp_alphas:
    clf = DecisionTreeClassifier(random_state=0, ccp_alpha=ccp_alpha)
    clf.fit(X_train, y_train)
    clfs.append(clf)

# Remove the last element in clfs
clfs = clfs[:-1]
ccp_alphas = ccp_alphas[:-1]

# Perform 15-fold cross-validation on pruned classifiers
cv_scores_after = [cross_val_score(clf, X, y, cv=15).mean() for clf in
    clfs]

# Find the index of the maximum accuracy
max_accuracy_index = cv_scores_after.index(max(cv_scores_after))
best_clf = clfs[max_accuracy_index]

# Print the pruned decision tree
print("\nBest Pruned Decision Tree:")
text_representation_after = export_text(best_clf)
print(text_representation_after)

# Plot the best pruned decision tree
dot_data_after = export_graphviz(best_clf, out_file=None, feature_names=
    data.feature_names, class_names=data.target_names, filled=True, rounded
```

```
    =True, special_characters=True)
graph_after = graphviz.Source(dot_data_after)
display(Image(graph_after.pipe(format='png')))

# Print results
print("\nUnpruned Decision Tree - Mean Cross-validation score before
    pruning:", mean_cv_score_before)
print("Best Pruned Decision Tree - Mean Cross-validation score after
    pruning:", cv_scores_after[max_accuracy_index])
```

Listing 1: Machine Learning Classification: Decision trees using Python

5 OUTPUT

5.1 Unpruned Decision Tree

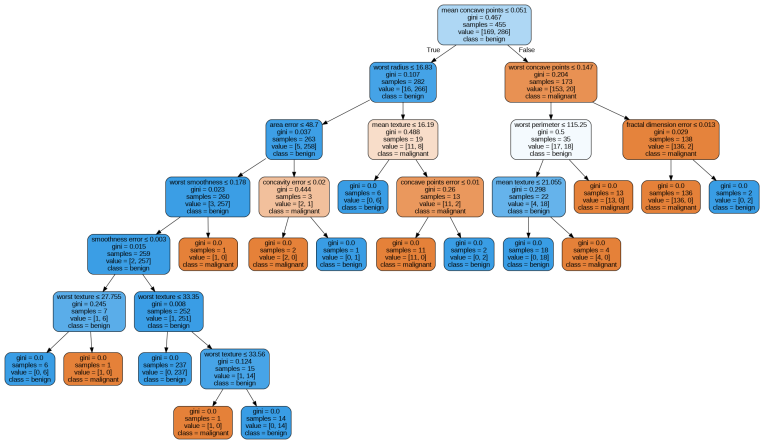


Figure 2: Unpruned Decision Tree

The unpruned decision tree, trained on the dataset, has a certain depth and structure based on the characteristics of the training data. This initial decision tree may be complex, potentially resulting in overfitting to the training data and poor generalization to unseen data. The tree's complexity might lead to a more intricate set of decision rules, potentially capturing noise or outliers in the training data.

Cross-validation scores were obtained before the pruning process, which serves as a measure of the model's performance before any simplification. These scores, including accuracy, precision, recall, and F1-score, provide insights into the initial effectiveness of the unpruned decision tree. By evaluating these metrics, we can determine how well the unpruned decision tree performs on unseen data and identify any potential issues related to overfitting or underfitting Mean Cross-validation score before pruning: 0.9332859174964437

5.2 After Pruning

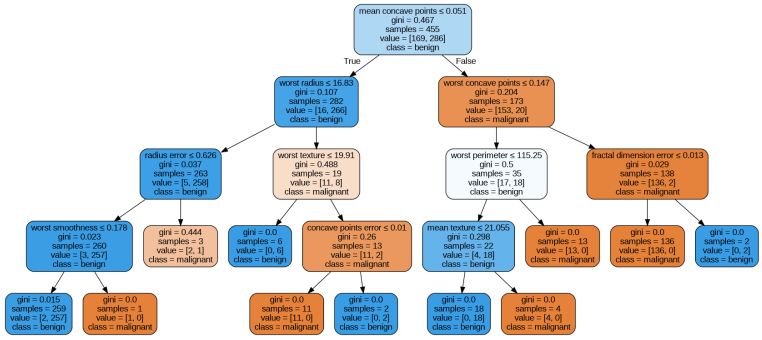


Figure 3: After Pruning

The best pruned decision tree represents a simplified version obtained through the pruning process, aiming to reduce overfitting and enhance the model's generalization capabilities. Pruning helps to eliminate unnecessary decision nodes and branches, leading to a more generalized and simplified tree structure that captures the most significant patterns in the data while reducing complexity.

Cross-validation scores were obtained after applying the pruning technique, providing insights into the pruned model's performance. Comparing these scores to the scores obtained before pruning allows us to evaluate the impact of pruning on the decision tree's effectiveness. Additionally, analyzing other metrics such as accuracy, precision, recall, and F1-score after pruning helps to determine the extent to which the pruned decision tree has improved its ability to generalize to unseen data.Mean Cross-validation score after pruning: 0.9333333333333332

```
Unpruned Decision Tree - Mean Cross-validation score before pruning: 0.9332859174964437
Best Pruned Decision Tree - Mean Cross-validation score after pruning: 0.9333333333333332
```

Figure 4: Mean Cross-validation score

6 DISCUSSION & ANALYSIS

In this lab report, we observed that pruning the decision tree led to a simpler model without significantly compromising its predictive performance. By removing unnecessary nodes and subtrees, the pruned tree became more interpretable and less likely to overfit to the training data.We saw also the difference in mean Cross-validation score,before pruning: 0.9332 and after pruning: 0.9333. So overall from this experiment we can analysis of the decision tree's performance before and after pruning allows for informed decision-making regarding the optimal balance between model complexity and generalization