National University of
Sciences & Technology
School of Electrical
Engineering and
Computer Science
Department of Computing

SE : Design and Analysis of
Algorithms(3+0): BESE Spring
2023

Name	CMS IDs
Sohaib Ahmed	**373937**
Hira Toheed Butt	**367864**
Umar Abdullah	**369741**

Project Report

Instructor: Dr. Zuhair Zafar

Introduction:

Project Background:

The project focused on credit risk assessment using machine learning models based on decision trees. Decision tree was chosen to be the base algorithm because research paper as mentioned in our project proposal published a research that credit risk assessment got best results with decision trees and its modifications. So, we chose decision tree and applied its modifications.The purpose of the project was to develop an effective solution for determining whether a person should be granted a loan or not. The project aimed to explore the performance of different decision tree algorithms and their modifications in solving the credit risk assessment problem.

Project Scope:

The project involved applying various decision tree algorithms, including gradient boost, decision stump, and others, to build ML models for credit risk assessment. The scope of the project encompassed analyzing the accuracy, time complexity, and space complexity of these models. Additionally, the project aimed to evaluate the trade-off between accuracy and computational complexity.

Project Objectives:

Following are objectives of our project:

- Build ML models based on decision trees for credit risk assessment.
- Apply modifications of decision trees, such as boosting techniques, to enhance model performance.
- Analyze the accuracy, time complexity, and space complexity of the models.
- Assess the trade-off between accuracy and computational complexity.
- Compare the performance of different decision tree algorithms and their modifications.
- Provide insights and recommendations based on the analysis conducted.

By addressing these objectives, the project aimed to provide a comprehensive understanding of decision tree algorithms and their applicability in credit risk assessment.

Project Timeline:

Day 1-3: Data Preprocessing:

- Collect two datasets from Kaggle
- Perform data preprocessing

Day 4-7: ML Model Development

- Apply different ML models, including decision trees, LP Boost, etc.
- Evaluate the performance of each model using appropriate evaluation metrics
- Select the most promising ML models for credit risk assessment

Day 8-10: Analysis and Reporting

- Divide the team into subgroups to analyze specific ML algorithms
- Analyze the performance, strengths, and weaknesses of each ML algorithm
- Compare the accuracy, time complexity, and space complexity of the models
- Compile the analysis and findings into a comprehensive report

Contribution:

Work Performed:

- Initially, the project used a decision tree algorithm as the base model for credit risk assessment.
- Several modifications, including boosting algorithms such as LPboost, Totalboost, Brownboost, Gradientboost, and XGBoost, were implemented to improve accuracy.

Among the modifications, Gradientboost was chosen as the final algorithm due to its highest accuracy.

Analysis and Results:

- The analysis compared the performance of the decision tree with the modified algorithms.
- Gradientboost showed the highest accuracy but had higher time and space complexity compared to the decision tree.
- This tradeoff between accuracy and complexity was observed across the modifications.

<u>Equation</u>:

<u>Decision Tree Algorithm :</u>

A decision tree is a supervised machine learning algorithm used for both classification and regression tasks. It creates a model that predicts the value of a target variable based on several input features. The decision tree algorithm builds a tree-like model of decisions and their possible consequences.
The basic concept of a decision tree is to split the data based on different attributes or features in a hierarchical manner, creating a tree structure.

Entropy:

Entropy is the measure of the degree of randomness or uncertainty in the dataset. In the case of classifications, It measures the randomness based on the distribution of class labels in the dataset.
The entropy for a subset of the original dataset having K number of classes for the ith node can be defined as:

$$H_i = -\sum_{k \epsilon K}^{n} p(i,k) \log_2 p(i,k)$$

Where, S is the dataset sample. k is the particular class from K classes and p(k) is the proportion of the data points that belong to class k to the total number of data points in dataset sample S.

$$p(k) = \frac{1}{n} \sum I(y = k)$$

Important points related to Entropy:

- The entropy is 0 when the dataset is completely homogeneous, meaning that each instance belongs to the same class. It is the lowest entropy indicating no uncertainty in the dataset sample.

- When the dataset is equally divided between multiple classes, the entropy is at its maximum value. Therefore, entropy is highest when the distribution of class labels is even, indicating maximum uncertainty in the dataset sample.
- Entropy is used to evaluate the quality of a split. The goal of entropy is to select the attribute that minimizes the entropy of the resulting subsets, by splitting the dataset into more homogeneous subsets with respect to the class labels.

Information Gain:

Information gain measures the reduction in entropy or variance that results from splitting a dataset based on a specific property. It is used in decision tree algorithms to determine the usefulness of a feature by partitioning the dataset into more homogeneous subsets with respect to the class labels or target variable. The higher the information gain, the more valuable the feature is in predicting the target variable.

The information gain of an attribute A, with respect to a dataset S, is calculated as follows:

$$\text{Information Gain(H, A)} = H - \sum \frac{|H_V|}{|H|} H_v$$

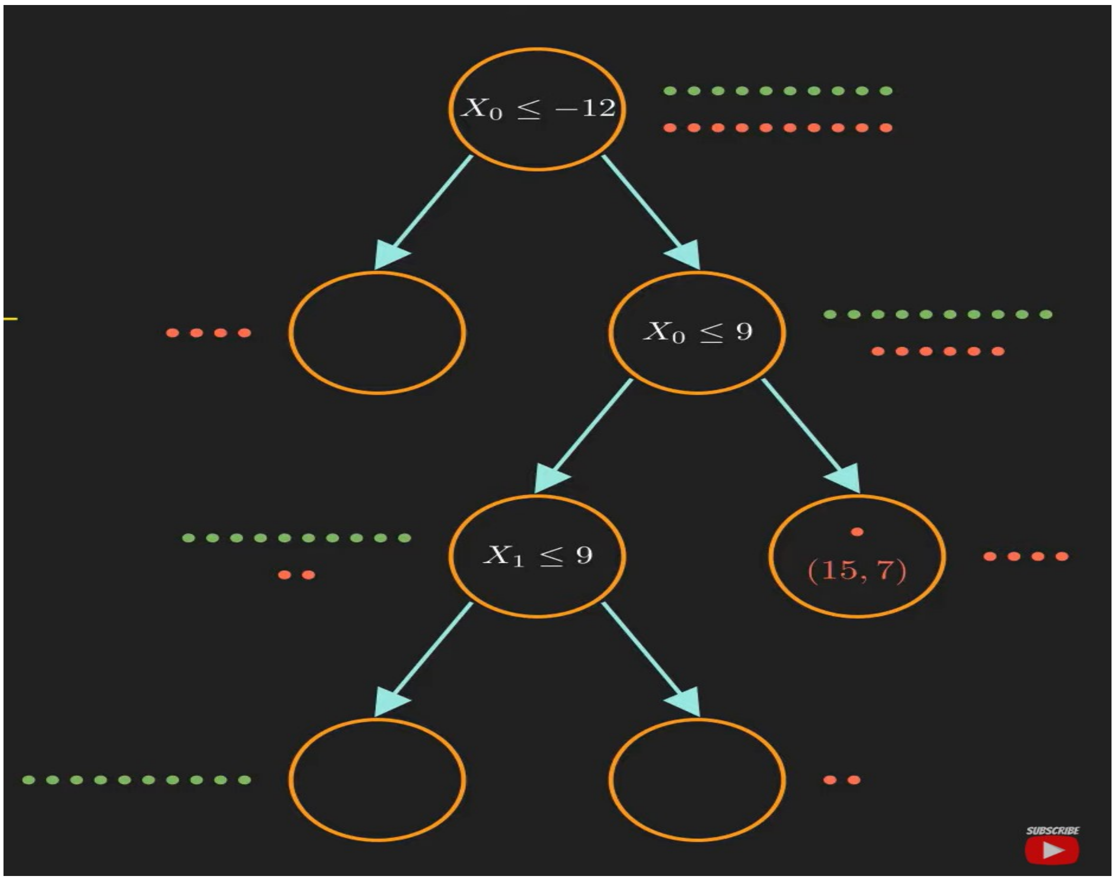
where

- A is the specific attribute or class label
- |H| is the entropy of dataset sample S
- |HV| is the number of instances in the subset S that have the value v for attribute A

Information gain measures the reduction in entropy or variance achieved by partitioning the dataset on attribute A. The attribute that maximizes information gain is chosen as the splitting criterion for building the decision tree.

Diagram:

It builds a flowchart-like tree structure where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label. It is constructed by recursively splitting the training data into subsets based on the values of the attributes until a stopping criterion is met, such as the maximum depth of the tree or the minimum number of samples required to split a node.



Pseudo-Code:

```
import numpy as np
from collections import Counter
class Node:
    def __init__(self, feature=None, threshold=None, left=None,
right=None,*,value=None):
    def is_leaf_node(self):
        # Check whether the node is a leaf node or not
        return self.value is not None
class DecisionTree:
    def __init__(self, min_samples_split=2, max_depth=100,
n_features=None):
        # Initialize DecisionTree class with min_samples_split,
max_depth, and n_features
    def fit(self, X, y):
        # Fit the model on the training data
        self.n_features = X.shape[1] if not self.n_features else
min(X.shape[1],self.n_features)  #in this line its checking that
the n_features should exceed the total no of features
        self.root = self._grow_tree(X, y)
    def _grow_tree(self, X, y, depth=0):
        Grow the tree recursively .
Find best split and call function again on left and right indexes

    def _best_split(self, X, y, feat_idxs):
        for feat_idx in feat_idxs:
            X_column = X[:, feat_idx]
            thresholds = np.unique(X_column)
           for thr in thresholds:
                # calculate the information gain
                gain = self._information_gain(y, X_column, thr)
                if gain > best_gain:
                    best_gain = gain
                    split_idx = feat_idx
                    split_threshold = thr
        return split_idx, split_threshold
    def _information_gain(self, y, X_column, threshold):
        Calculate information gain through entropy and return it
    def _split(self, X_column, split_thresh):
      splits dataset into left and right indexes and return them
    def _entropy(self, y):
        hist = np.bincount(y)
        ps = hist / len(y)
        return -np.sum([p * np.log(p) for p in ps if p>0])
    def _most_common_label(self, y):
        Find most common label and retrun it
    def predict(self, X):
        return np.array([self._traverse_tree(x, self.root) for x
in X])
    def _traverse_tree(self, x, node):
        if node.is_leaf_node():
            return node.value
        if x[node.feature] <= node.threshold:
            return self._traverse_tree(x, node.left)
        return self._traverse_tree(x, node.right)
```

Time complexity :

Best case:

The best case time complexity of decision tree is O(n*m*logn) times where m is number of features and n is number of samples. This is explained later.

Average case :

The average case time complexity of decision tree is O(n*m*logn).

Explanation of best and average case:

In daily life, we have most of the datasets satisfying average case of decision tree. Average cases and best case are such that split of decision tree gets all pure nodes at almost logn height. At each level of tree, we have two nested for loops outer one running m times where m is number of features and inner one running n times for each outer one where n is number of samples. That's why total time complexity in these cases is O(n*m*logn).

Worst case:

The absolute maximum depth would be N−1, where N is the number of training samples. You can derive this by considering that the least effective split would be peeling off one training example per node. So maximum depth will be O(n) and for each one ,nm times loop runs. So, total time complexity is O(m*n^2).

Test time complexity of all cases is height of trees for them i.e. O(logn) , O(logn) and O(n) for best, average and worst cases respectively.

<u>Space complexity:</u>

Best case and Average case:

The best and average case time complexity of decision tree is O(nodes).This happens when our data is such that we find a pure best split at almost logn height such that all leaf nodes have yes or no answers of target variable. We know that decision tree is a complete binary tree so at logn height, we have 2n-1 nodes. So we can write space complexity here as O(n).

Worst case:

The worst case time complexity of decision tree is O(nodes). Since in worst case, typically height of tree is n and for n height we have 2^(n+1)-1 nodes. So we can write above space complexity as O(2^n).

Test space complexity for all cases is same since testing also only involves storing extra nodes.

Gradient Boosting Algorithm :

Gradient boosting is a machine learning technique used for both regression and classification tasks. It is an ensemble method that combines multiple decision trees, to create a strong predictive model.

The basic idea behind gradient boosting is to iteratively build a sequence of weak models, where each subsequent model is trained to correct the mistakes made by the previous models.

To improve the robustness and generalization of the gradient boosting algorithm, additional techniques can be applied, such as subsampling the training data, introducing regularization parameters, and adjusting the learning rate.

General Steps For Gradient Boosting :

Step 1: Initialize the Ensemble:

The process starts by initializing the ensemble with a simple model. For regression tasks, this can be a model that predicts the mean or median of the target variable

Step 2: Iterative Training:

For each iteration:

Compute the residuals with respect to the current ensemble's predictions.

The weak model is typically trained on a subset of the data.

Add the weak model to the ensemble

Step 3: Update Ensemble Predictions:

After each iteration, update the ensemble's predictions by summing the predictions from all the models in the ensemble, using the weights assigned to each model.

Step 4: Repeat Iterations:

Repeat the iterative training process for a predefined number of iterations or until a stopping criterion is met.

Step 5: Final Prediction:

The final prediction is obtained by aggregating the predictions of all the weak models in the ensemble, often using a weighted average.

Time Complexity :

Best and Average Case:

O(M *n log n d) where M = Number of trees , n = Number Of Samples and d = Number Of Features

Cleary it involves M decision trees so it will have O(M*time complexity of single decision tree) and decision tree's time complexity is same as above.

Worst case:

O(M *m* n^2) where M = Number of trees and n = Number Of Samples

Cleary it involves M decision trees so it will have O(M*time complexity of single decision tree) and decision tree's time complexity is same as above.

Test time complexity of all cases is height of trees*number of trees for them i.e. O(M*logn) , O(M*logn) and O(M*n) for best, average and worst cases respectively.

Space Complexity :

Best and Average Case:

O(M*n + gamma m) where M=number of trees,N=number of samples and Gamma m =Results of each decision tree which are stored for calculating final results at end.

Number of nodes in each tree are O(n) and we have M number of trees. Also we have to store results of each decision tree which are stored for calculating final results at end. Hence our time complexity is O(M*n + gamma m).

Worst case:

O(M*2^n + gamma m) whereM=number of trees,N=number of samples and Gamma m =Results of each decision tree which are stored for calculating final results at end.

Number of nodes in each tree are O(2^n) in worst case and we have M number of trees. Also we have to store results of each decision tree which are stored for calculating final results at end. Hence our time complexity is O(M*2^n + gamma m).

Test space complexity for all cases is same since testing also only involves storing extra nodes.

Achievement:

- Successfully implemented and analyzed various decision tree modifications for credit risk assessment.
- Achieved the highest accuracy with the final algorithm, Gradientboost.

Effort:

- Dedicated significant time and effort to conducting thorough research on decision trees and their modifications.
- Invested extensive effort in implementing, fine-tuning, and optimizing the algorithms.
- Conducted rigorous testing, analysis, and documentation of the results.
- Ensured the algorithms' robustness in handling special cases and challenging scenarios.

Challenges and Hindrances:

- Encountered challenges related to handling imbalanced data, missing values, and outliers in the credit risk assessment dataset.
- Time and space complexity analysis posed difficulties due to the complexity of the implemented modifications.
- Balancing accuracy and complexity proved to be a challenging tradeoff.

Despite the encountered challenges and hindrances, the project's achievement lies in successfully implementing and analyzing decision tree modifications for credit risk assessment.

Conclusion:

The project achieved its primary goal of solving the credit risk assessment problem using decision trees and their modifications. The final algorithm, Gradientboost, demonstrated the highest accuracy among the tested modifications. The analysis provided valuable insights into the tradeoff between accuracy and time/space complexity.

The project's results were satisfactory, considering the successful implementation and analysis of multiple decision tree modifications. However, it is important to note that the project's scope was limited to decision trees and their modifications. Other advanced machine learning algorithms, such as neural networks or ensemble methods, could have been explored for comparison and potential performance improvements.

To further improve the project, several areas of future work could be considered:

- ***Exploration of advanced ML algorithms:***

Investigate the performance of neural networks, ensemble methods (e.g., random forests), or deep learning models to determine if they could provide better accuracy or efficiency compared to decision tree-based models.

.

REFERENCES:

Research Paper:

- https://www.researchgate.net/publication/343235514_Credit_Risk_Assessment_based_on_Gradient_Boosting_Decision_Tree

Datasets:

- https://www.kaggle.com/datasets/samuelcortinhas/credit-card-classification-clean-data
- https://www.kaggle.com/datasets/laotse/credit-risk-dataset

Code:

https://anderfernandez.com/en/blog/code-decision-tree-python-from-scratch/

Analysis:

- https://medium.com/analytics-vidhya/computational-complexity-of-ml-algorithms-1bdc88af1c7a

Reference to First decision tree research:

First article I've been able to find that develops a "decision tree" approach dates to 1959 and a British researcher, William Belson, in a paper titled *Matching and Prediction on the Principle of Biological Classification*, (*JRSS*, Series C, Applied Statistics, Vol. 8, No. 2, June, 1959, pp. 65-75), whose abstract describes his approach as one of *matching* population samples and developing criteria for doing so:

In this article Dr Belson describes a technique for matching population samples. This depends on the combination of empirically developed predictors to give the best available predictive, or matching, composite. The underlying principle is quite distinct from that inherent in the multiple correlation method. Links are as follows:

- https://www.jstor.org/stable/pdf/2985543.pdf?refreqid=excelsior%3A0fd3e55a5a27f0ac5cebba623f741da5&ab_segments=&origin=&initiator=&acceptTC=1
- https://ideas.repec.org/a/bla/jorssc/v8y1959i2p65-75.html