Project 3: Decision Tree for Classification

Nazanin Yari
Johns Hopkins University

Introduction to Machine Learning - Section 8VL

Abstract

The general purpose of pruning a decision tree is increasing the performance, or increasing the accuracy. While pruning a decision tree might improve performance by reducing the size of the tree, it might reduce the accuracy by eliminating some useful information. In this project, the effects of using reduced pruning technique is studied in detail. The results of the experiments with and without using pruning strategies are compared. The analysis of the algorithm over only one dataset showed that pruning improved the performance by reducing the size of the tree significantly while resulting in almost the same classification accuracy rate.

Problem Statement

The purpose of this project is to examine whether pruning the decision tree improves the performance of the data classification due to reducing overfitting. A complete tree is created by memorizing a set of training dataset so that the generalization power could suffer for the unseen or test dataset. This is because the created tree might be too complex. Post pruning of a tree is a technique that can be used after the tree is built in order to improve the prediction. It results in a smaller tree with fewer splits which might lower the variance at the cost of some bias. The hypothesis is that the classification error rate of a pruned decision tree would be less than or equal to the classification error rate of a complete decision tree.

Algorithm Implementations

ID3 Based Decision Tree. For this assignment, the ID3 (Iterative Dichotomiser) algorithm is used to generate a decision tree. This algorithm iteratively divides the features into two groups, the first group consists of the dominant attribute and the other group is used to construct the tree. It first calculates the entropy and information gains for each attribute. The decision node would be the dominant attribute with the highest information gains. This dominant attribute would be removed from the dataset, and the above process would be repeated for the remaining features.[1] The procedure continues until reaching a classification for each branch.

Reduced Error Pruning. The reduced error pruning is a post pruning technique which reduces the size of a decision tree to improve generalization. Starting from the leaves, each subtree below any non-leaf node is temporarily replaced with a leaf labeled with the current majority class at that node. The accuracy of the pruned decision tree is measured over a validation set. If the performance of the pruned tree has not been decreased, the node would be permanently pruned. This process continues until further pruning of the tree reduces the performance.

Experimental Approach

ID3 Based Decision Tree. This algorithm can be found in the decision_tree.py file.

1. Creating a Tree Using Training Set: The function `decision_tree()` uses a training dataset to learn the classification. It first finds the index of the dominant feature using the function `best_ig()`. It then creates a list with feature names as its first element, and an empty dictionary as its second element. The function iterates over the list of the unique values of the selected dominant feature, and creates a subset of the dataset with the unique values. It then stores the unique values of the class column

[1] "A Step by Step ID3 Decision Tree Example - Sefik Ilkin Serengil." 20 Nov. 2017, https://sefiks.com/2017/11/20/a-step-by-step-id3-decision-tree-example/. Accessed 21 Nov. 2020.

with the counts of each class label in a tuple. For each node and arcs of the tree, the y distribution or the counts of each class label is stored as a value in the tree dictionary. The tree reaches the leaf node if the data is pure, meaning it reaches a single classification. If no more features are left, it picks the most frequent class as the leaf node. The function recursively calls itself using the subset of the dataset until it reaches a leaf node.

The function `best_ig()` is used to find a feature with the highest information gain. It first calculates the entropy of the whole dataset. It then computes the entropy of each feature and the information gain using the calculated entropy and the entropy of the dataset. Finally, it returns the value and the index of the feature with the highest information gain.

2. Classification Using Test Set: The function `predict_batch()` creates an array which stores the predicted classifications for the test set. It computes the class labels for each data point by calling the function `predict_row()`. This function takes a single data point or row and the decision tree as its arguments and returns a class label by recursively calling itself using the inner node values as its tree argument. The function takes a default value in case a subtree does not exist in the decision tree. This value is set equal to the majority class in the corresponding node.

Reduced Error Pruning. This algorithm can also be found in decision_tree.py file. It consists of three functions, `compute_feature_depths()`, `prune_single_feature()`, `prune()`.

The function `compute_feature_depths()` computes the depth of the features in the tree. The function alters its argument feature_with_depths by traversing the tree and adding a subtree and its corresponding depth to the feature_with_depths list. When the leaves are not reached yet, the function recursively calls itself with the updated feature_with_depths list and incrementing the depth by 1.

The function `prune_single_feature()` prunes the tree and replaces a single subtree with a leaf node. It takes a sorted version of the feature_with_depth list, and picks the first element (with maximum depth) as a pruning node. It then removes this node from the feature_with_depth list. Finally, it replaces the subtree with the index of the majority counts of the distribution stored for each class value in the tree.

The function `prune()` prunes the whole tree based on the reduced error pruning technique. It first computes the feature_with_depths by calling the function `compute_feature_depths()`. It then sorts the list based on the first element, or depth. Then it calculates the error rate of the classification using the pre-pruned tree on the validation set. It then prunes the tree on the feature with maximum depth and computes the classification error of the pruned tree using validation set. This process continues until the error of the pruned tree gets larger than the tree, or until the root node is reached.

Classification Error Rate. The decision tree performance is measured by the function `error_rate()`. It compares the predicted class array to the actual classification array and computes an error percent where the algorithm misclassified data points.

Dataset

The dataset used for this project was the Car Evaluation Dataset from UCI Machine Learning Repository.[2] The dataset includes 6 categorical features: buying price, maintenance price, number of

[2] "UCI Machine Learning Repository: Data Set." http://archive.ics.uci.edu/ml/datasets/Car+Evaluation). Accessed 21 Nov. 2020.

doors, capacity in terms of persons, size of luggage boot, and safety. The class values are unacceptables, acceptable, good, and very good.

The dataset has been preprocessed and no missing values have been found. In order to create a decision tree, prune the tree and compare the performance of the trees, the dataset is randomly divided into three parts. A block of 60% of the data is the training set used to build the tree, the block of 30% of the data is the validation set used to prune the data, and the remaining block of 10% of the data is the test set used to compare the performance of the trees. The function `data_train_validation_test()` creates these three blocks by calling `train_test_split()` function twice.

Results

The classification error from the complete tree over the test set was 28.902%. The list of the features of the tree with their corresponding depth can be found in the output folder submitted along with this project. It can be observed that all of the branches with depth 5 or the maximum depth have been pruned making the maximum depth of the pruned tree equal to 4. The classification error of the pruned tree over the test set was 28.323% resulted in 0.05% reduction in error compared to the complete tree. Therefore, pruning the tree improved the performance of the tree by size and overfitting reduction while resulting in almost the same classification error rate.

Discussion

The pruning technique is used to reduce the size of a decision tree by removing some subtrees. It also improves generalization by reducing the complexity of the classifier. Although, reducing overfitting of the tree can generally improve the prediction accuracy, the error rate of the pruned tree only decreased by 0.05%. Therefore, pruning did not hurt with misclassification, but it resulted in a very simpler tree.

Summery

Pruning the tree decreased the size of the tree significantly and improved the generalization without increasing the misclassification error.