Programming Project #03 – Decision Trees

Jacob M. Lundeen JLUNDEE1@JHU.EDU

Department of Data Science
Johns Hopkins University
Baltimore, MD 21218

Abstract

This paper describes six experiments (three regression and three classification) using a Decision Tree algorithm coded from scratch. All six experiments utilize 5x2 cross validation and the Decision Tree utilizes early stopping mechanics and Reduced Error Pruning. The results show that the algorithm was not coded correctly as the Mean Square Error and F_1 scores show poor performance across the respective experiments.

1. Introduction

Decision Tree (DT) learning is a supervised learning method that generates a "tree" that can be used for both classification and regression. At the leaf nodes (the nodes where the predictions are made) the output is a plurality vote of the remaining examples for classification. For regression, the mean value of the remaining examples is the output. DTs are extremely popular in Machine Learning (ML) due to their simplicity and explainability. The explainability is helped in large part due to DTs lending themselves to visualizations where a person can see the path of how an example was predicted.

Along with the standard DT implementation, early stopping is implemented to keep the DT from growing too large. As well, to improve accuracy and reduce complexity, Reduced Error Pruning (REP) will be implemented.

Problem Statement

The purpose of this experiment is to see how well the DT algorithm performs in classification and regression on six data sets (described later). My hypothesis is that the DT will perform better at classification than at regression. REP will improve the performance of both.

2. Algorithms and Experimental Methods

Data Sets

The data sets used are the same six used previously: Abalone, Breast Cancer, Car, Forest Fires, House Votes, and Machine.

The abalone data set is a regression set used to predict the age of an abalone from its physical characteristics.

The breast cancer set is a classification set used to classify a tumor as benign or malignment based on characteristics of the tumor.

The car dataset is a classification problem to determine acceptability of a car based on certain specifications.

The forest fire data set is a regression problem looking to predict the burned area of forest fires utilizing meteorological and other data.

The house votes data set is a classification problem attempting to classify congresspeople as republican or democrat.

The machine data set is a regression problem to estimate performance of computer hardware.

Pre-Processing

Pre-processing of the data is managed the same as in the first programming assignment. The data is read in, and missing values are managed for the Cancer data set. Regression data sets are normalized. Unlike the previous assignments, categorical variables are not encoded for DTs because the algorithm can manage them. A new feature added for this assignment was making sure the class and target variables of the data sets are set as the last column in the data to eliminate the need to identify the class/target variable for the functions. Finally, the forest data set's target variable 'area' is skewed, so the variable is log transformed to remove the skew (this is done before normalization of the data set).

Experimental Approach

As stated earlier, DTs are supervised learning predictive models that use a set of binary rules to calculate, or predict, a target value (Drakos, 2020). A tree contains branches, nodes, and leaves. Figure 1 shows what a basic DT looks like:

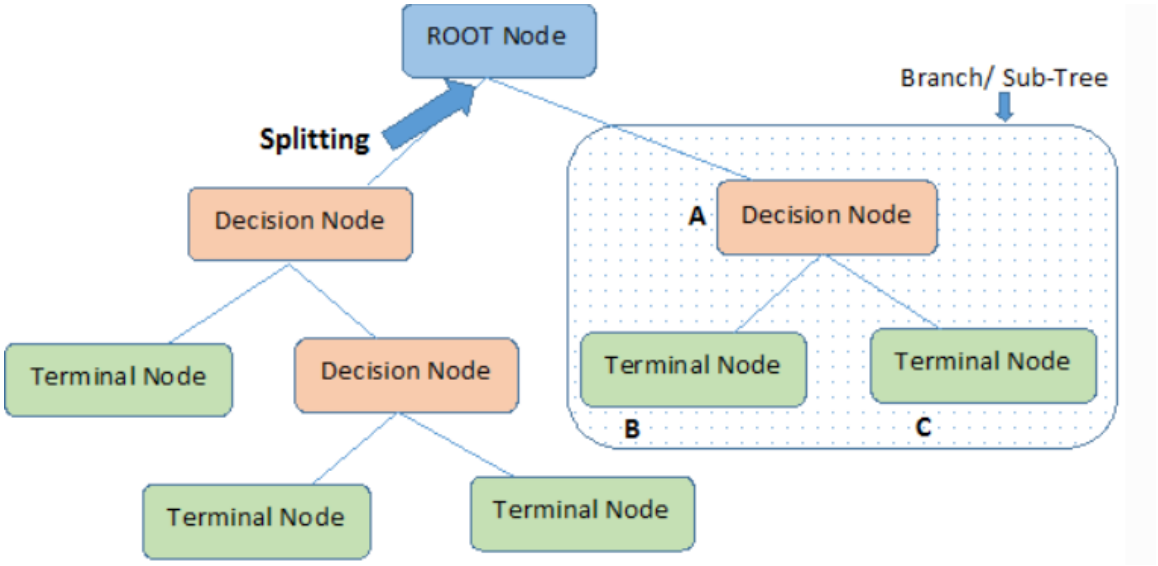

Figure 1 - Naive Decision Tree Structure (Drakos, 2020)

A DT starts at the root node and splits along branches to internal, or decision, nodes which eventually terminate in terminal nodes, or leaves. Each internal node and its corresponding branches can be considered a sub-tree. These are particularly useful when pruning a tree, which is discussed later.

As one traverses down the tree, the internal notes dictate the path, making the decision based on True/False questions (e.g., $X \geq 0.1$ or $X < 0.1$). Once a data point has traversed far enough down the path, the DT is confident enough to make a prediction on what the data point is. The splitting criteria differs between classification and regression. For classification, entropy and information gain is used. For regression, the Mean Squared Error (MSE) is used.

Classification

Classification works as follows: starting at the root node, we calculate the Entropy (E) and the Information Gain (IG) of the data set and of each feature and chooses the feature with the largest IG. The data set is then split on that feature by the unique values of that feature. It then recursively continues to split down each branch based on the features not previously selected. At the end, each internal node represents a feature that was split on, and the leaf nodes represent the class label of the final subset of the branch.

An issue that arises with classification is when a feature possesses many unique values. When this occurs, the algorithm tends to split along a high number of those unique values, creating a very wide tree that overfits the data. To counteract that, the idea of Gain Ratio (GR) was developed. GR penalizes features with a high unique value count by dividing the IG by the Intrinsic Value (IV) of the feature.

To calculate these, let ***p*** and ***n*** be the positive and negative (respectively) examples for the j^{th} value of the feature (or data set), m_i is the size of the domain for the feature. Following are the corresponding equations:

$$I(p,n) = \frac{-p}{p+n}\log_2\frac{p}{p+n} - \frac{-n}{p+n}\log_2\frac{n}{p+n}$$

$$E_\pi(f_i) = \sum_{j=1}^{m_i}\frac{p_\pi^j + n_\pi^j}{p_\pi + n_\pi} I(p_\pi^j, n_\pi^j)$$

$$IG(f_i) = I(p_\pi, n_\pi) - E_\pi(f_i)$$

$$IV(f_i) = -\sum_{j=1}^{m_i}\frac{(p_j + n_j)}{(p+n)}\log_2\frac{(p_j+n_j)}{(p+n)}$$

$$GR = \frac{IG(f_i)}{IV(f_i)}$$

Regression

For regression, the process is the same except for the splitting criterion. Starting at the root, the feature and threshold value are determined by sorting the data, iterating through the values, and calculating the MSE at each binary split. This process is done for all possible binary splits and summed up. The feature that minimizes the MSE is selected as the splitting feature, and then the data set is split on the median value of that feature. The equation for minimizing the MSE is:

$$Err'_\pi = \frac{1}{|D_\pi|}\sum_j\sum_t \left(r^t - \hat{r}_{\pi j}^t\right)^2 b_{\pi j}(x^t)$$

where r^t is the ground truth response value for $\mathbf{x}^t$, $\hat{r}_{\pi j}^t$ is the predicted response value for $\mathbf{x}^t$ in partition π for following branch j, and:

$$b_{\pi j}(x^t) = \begin{cases} 1 \; if \; x^t \in D_\pi \\ 0 \; otherwise \end{cases}$$

Early Stopping

For both classification and regression, early stopping is implemented to keep the tree from growing too wide and/or too deep. This is done by setting the max depth of the tree and setting a threshold for how small the partitions can be. For example, if the threshold θ is set to five, then once the number of examples in a partition are less than or equal to five, that partition is automatically turned into a leaf node.

Pruning

To also help prevent overfitting, we implemented REP as an option. REP is a simple form of post-pruning, where each node, starting at the leaves, is replaced with its most popular class. If the accuracy of the DT is not affected or improved, the change is kept

Cross-Validation

Kx2 cross-validation is utilized for this experiment with k=5 (for a total of 10 experiments run each time). For classification, the folds (including the validation/pruning set that is created first) are stratified.

Tuning

The max depth and θ hyperparameters are tuned using the pruning set that is 20% of the original data set. Both are tuned using a grid search method where max depth is set between three and ten and θ is set between one and ten (all integers). The values that are returned are based on highest F_1 score for classification and smallest MSE for regression.

Metrics

For regression: MSE is the primary metric, but R^2 and Pearson's Correlation are also reported.
For classification: F_1 is the primary metric, but Precision and Recall are also reported.

3. Results

Table 1 shows the results for both classification and regression on the six data sets without pruning:

Table 1 - Experiment Results without REP

	Parameters		**Metrics**					
	Max Depth	**θ**	**R^2**	**MSE**	**Pearson**	**Precision**	**Recall**	**F_1**
Abalone	3	1	0	0	0			
Breast Cancer	3	1				0.164	0.25	0.13
Cars	3	1				0.164	0.25	0.13
Forest	3	1	-0.549	1.039	0			
House	3	1				0.164	0.25	0.13
Machine	3	1	-0.847	1.801	0			

Table 2 shows the results for both classification and regression on the six data sets with pruning:

Table 2 - Experiment Results with REP

	Parameters		Metrics					
	Max Depth	**θ**	**R^2**	**MSE**	**Pearson**	**Precision**	**Recall**	**F_1**
Abalone	3	1	0	0	0			
Breast Cancer	3	1				0.164	0.25	0.13
Cars	3	1				0.164	0.25	0.13
Forest	3	1	-0.549	1.039	0			
House	3	1				0.164	0.25	0.13
Machine	3	1	-0.847	1.801	0			

All metrics are the mean of the respective score from 5x2 cross validation.

4. Discussion

No REP

When REP was not used, the DT algorithm did not perform well for either regression or classification. In fact, the same Precision, Recall, and F_1 scores were found for all three classification data sets (0.164, 0.25, and 0.13 respectively). Additionally, the same values for the max depth and θ were used for not just all three classification experiments, but for all six experiments as well. This clearly indicates an issue with the coding for either growing the tree and/or traversing the tree when predicting the test set.

For the three regression experiments, we see negative R^2 values, no Pearson's correlation value, and an MSE over one. As well, I got no scores for the Abalone data set at all. This further verifies that there are issues with the coding of the algorithm.

The coding issue resides in the prediction portion of the algorithm. I confirmed by hand that the splitting criterion algorithms were functioning correctly (Tables 3 and 4 show the Gain Ratio and MSE of two partitions, respectively).

Table 3 - Gain Ratio for a Cars Partition

Feature	Gain Ratio
clump_thick	0.023
unif_size	0.015
adhesion	0.019
epithelial_size	0.016
bland_chromatin	0.021
norm_nucleoli	0.019
mitosis	0.006

Table 4 - MSE for a Forest Partition

Feature	MSE
X	17.02
Y	12.56
FFMC	16.95
DMC	15.15
DC	13.11
ISI	15.85
temp	17.90
RH	17.55
wind	13.18
rain	10.08

Growing the tree looked to work correctly as well, but there were two key issues. First, tracking which node belonged to which parent and the branch of that parent did not work. After the root node, there was nothing keeping partitions to be split on the same feature (which is expected). However, this caused problems during prediction because the algorithm could not differentiate between which decision node it should be on if there were more than one node on a level that was split on the same feature.

Secondly, DTs are supposed to be recursive algorithms. I could not get the recursion to work, so everything is done iteratively which meant the experiment took much longer to run.

With REP

You will notice that all the metrics are the same with and without REP. That is because I was unable to get REP to work as intended, so no pruning was performed.

5. Conclusion

The results show that my hypothesis was incorrect. However, given the problems with the coding, I cannot say definitively that the DT algorithm performed poorly. If the DT were able to perform correctly, I believe my hypothesis would be proven correct.

Acknowledgements

I would like to acknowledge the love and support of my wife, Dr. Jordan S. Lundeen, and our wonderful family: Logan, Sydney, Mavis, Thea, Hermione, Ron, and Ginny. Without their unyielding support of my endeavors, I am not sure I would have made it this far. I would also like to thank Johns Hopkins for taking a chance on me.

References

Drakos, G. (2020, February 5). *Decision Tree Regressor explained in depth*. GDCoder. https://gdcoder.com/decision-tree-regressor-explained-in-depth/