Programming Project #3

Chris Wells

# Introduction

The purpose of this project is to implement both classification and regression trees and test their performance on each of the six data sets, both before and after pruning.

In the first case, after pre-processing I split the datasets into five partitions and conduct five-fold cross validation on each dataset. For each fold, I build a decision tree based on the training data using four of the partitions, and test on the fifth partition.

In the second case, after pre-processing I pull 20% of the entries from the dataset for use as a pruning set. I then split the remaining 80% of the entries and perform five-fold cross validation on them like so: First, I generate a decision tree on four partitions, then I prune the tree using the pruning set, and then test on the remaining fifth partition.

Since I will be training the decision trees to a theta of zero, I anticipate that the unpruned trees will be prone to overfit the data and perform less well in testing than the pruned trees. In addition, given the specificity of the predictions in the unpruned tree the results will be less stable over five-fold cross validation and give swings in accuracy dependent on chance, as one partition might fit the tree much better than the others. I expect that the pruned decision tree will give more stable accuracy and mean squared error (MSE) scores across the folds.

# Algorithm

The decision trees were implemented as Python classes: DecisionTreeClassifier and DecisionTreeClassNode for the classifier, and DecisionTreeRegressor and DecisionTreeRegNode for the regressor.

The DecisionTreeClassNode and DecisionTreeRegNode classes represent each node and leaf of the decision trees and contain the following attributes:

* data – a string containing either the feature that the node will split on or the class or *r* value for a leaf
* df – a pandas DataFrame containing all the entries of the dataframe with the training set that have entered this node. This is used when pruning.
* type – whether the node’s feature is categorical or numerical. This is used when splitting the node at generation or when traversing the tree to know how to interpret the node’s children.
* children – a dictionary mapping a branch name (either a class value for categorical or midpoint for numerical) to a child node. For categorical features an additional ‘default’ child node is added for feature classes that were not trained on and for numerical features an additional ‘upper’ child node is added for any feature values greater or equal than the greatest midpoint.

The DecisionTreeClassifier and DecisionTreeRegressor classes represent the tree itself. They contain the attributes top\_node, representing the root node of the tree and theta, representing the theta value to use when generating the tree (defaults to 0). The functions in both classes include:

* generate\_tree() – recursively-called method used to generate the decision tree from a training set.
* traverse\_tree() – recursively-called method to traverse the tree for a single row of testing or pruning data.
* find\_subtree() – returns a list of nodes in the tree with children. Used to identify subtrees for potential pruning.
* prune\_subtrees() – for each subtree in the node, replace with the majority (classification) or average (regression) value in the subtree root’s associated dataframe and tests this modified tree with the pruning set. If the modified tree performs worse, “roll back” the prune.

## Tree Generation

For Decision Tree Classifiers, I calculate the entropy for the node, then the gain for each feature using this entropy value. If the node entropy is zero or there is no feature with a higher gain than 0 I set this node as a leaf, with the most common class in the associated dataframe as the value.

If there is a feature with a highest gain, I then split the node on this feature. For categorical features I simply create a child and map the child for one of the feature’s possible classes. In addition, I create a ‘default’ child for any feature classes not trained on and set this to be a leaf with the most common class in the dataframe as the value.

For numerical features I sort the node’s dataframe on that feature, then return each midpoint in the dataframe between class changes. I then split the node on each midpoint, with the branch being taken when the feature value < midpoint. In addition, I create an ‘upper’ child for feature values >= the highest midpoint.

For each of the children created above, I then call the tree generation method, passing in an updated dataframe with this node’s feature removed and the branch condition applied. In this way, the tree is recursively generated.

Decision Tree Regressors are generated in much the same way, but using MSE instead of entropy when determining which feature to assign to the node. After splitting all of the features (either by class for categorical or the midpoint of the range for numerical) and calculating the MSE of each node in the split, I find the feature with the lowest MSE and choose this to split for the node.

## Tree Traversal

Tree traversal is handled recursively. For both types of trees, if the node is not a leaf (that is, has children) the type is checked. For nodes of a categorical feature the test example’s feature corresponding to the node’s data attribute is checked among the key values of the children dictionary (the branches). If a value is found the branch is taken. If not (the testing set contains a class the tree wasn’t trained for) the ‘default’ node is taken.

For nodes of a regression feature the test example’s feature is compared to the lowest branch value. Successive branches are checked until the feature value is less than the branch value. If the feature value is higher than all the ‘upper’ branch is taken.

In both cases a leaf will be reached eventually. Then the node’s data will be returned as a prediction to be scored.

## Tree Pruning

Because each node has an associated dataframe containing the dataset that enters the node, pruning is comparatively simple.

After identifying all the nodes with children (sub-trees), the performance value of the tree is first checked against the pruning set. Then the root node of the subtree is replaced with a leaf containing the majority (classification) or mean (regression) value of the associated dataframe. Now modified, the tree is again checked against the pruning set.

If the accuracy is greater with the pruned set the next sub-tree is checked, but if not, we take back the change by replacing the modified node with a copy of the original node. By swapping out the individual nodes this way, the tree is quickly and easily pruned.

# Results

## Pre-Processing

After pre-processing each dataset possessed the following dimensions. Since Decision Trees should be able to handle categorical or numerical data none of the features were encoded:

|  |  |  |
| --- | --- | --- |
| **Dataset** | **# Features** | **# Rows** |
| Abalone | 8 | 4,177 |
| Breast Cancer | 9 | 699 |
| Car | 6 | 1,728 |
| Forest Fires | 12 | 517 |
| House Votes | 16 | 435 |
| Machine | 8 | 209 |

## Accuracy – Unpruned and Pruned

After completing setup, I created Decision Trees for all data sets and tested them using five-fold cross validation. I then re-ran, pruning the Decision Trees after creation and again testing them with five-fold cross validation. Results are shown below compared to Project 1’s Null Model:

|  |  |  |  |
| --- | --- | --- | --- |
| **REGRESSION** |  |  |  |
| **Dataset/Fold** | **MSE** | **MSE Pruned** | **Null Model MSE** |
| **Abalone** |  |  |  |
| **Fold 1** | 20.18 | 10.67 | 11.50 |
| **Fold 2** | 16.05 | 9.19 | 9.39 |
| **Fold 3** | 11.04 | 9.51 | 10.13 |
| **Fold 4** | 12.71 | 11.04 | 10.31 |
| **Fold 5** | 19.05 | 9.20 | 9.90 |
| **Avg** | **15.81** | **9.92** | **10.25** |
| **Forest** |  |  |  |
| **Fold 1** | 576.73 | 209.80 | 495.00 |
| **Fold 2** | 6,244.70 | 877.83 | 1,522.41 |
| **Fold 3** | 1,326.04 | 688.15 | 7,159.40 |
| **Fold 4** | 1,033.38 | 7,600.29 | 14,423.31 |
| **Fold 5** | 11,560.28 | 912.90 | 1,338.95 |
| **Avg** | **4,148.23** | **2,057.79** | **4,987.82** |
| **Machine** |  |  |  |
| Fold 1 | 45,205.09 | 49,035.12 | 11,583.53 |
| Fold 2 | 17,574.17 | 17,603.44 | 17,743.86 |
| Fold 3 | 60,941.44 | 28,347.94 | 15,501.21 |
| Fold 4 | 13,759.26 | 24,141.11 | 77,886.21 |
| Fold 5 | 34,369.96 | 39,774.06 | 26,855.64 |
| **Avg** | **34,369.96** | **31,780.33** | **29,914.09** |
| **CLASSIFICATION** |  |  |  |
| **Dataset/Fold** | **% Acc** | **% Acc Pruned** | **Null Model % Acc** |
| **Breast** |  |  |  |
| **Fold 1** | 34.29 | 65.18 | 65.18 |
| **Fold 2** | 65.71 | 65.77 | 65.77 |
| **Fold 3** | 65.47 | 65.18 | 65.18 |
| **Fold 4** | 65.71 | 66.07 | 66.07 |
| **Fold 5** | 35.00 | 65.18 | 65.18 |
| **Avg** | **53.24** | **65.47** | **65.47** |
| **Car** | **%** | **% Pruned** | **Null Model** |
| **Fold 1** | 22.25 | 70.29 | 70.29 |
| **Fold 2** | 22.25 | 70.04 | 70.04 |
| **Fold 3** | 22.25 | 69.93 | 69.93 |
| **Fold 4** | 22.03 | 70.04 | 70.03 |
| **Fold 5** | 70.14 | 69.93 | 69.93 |
| **Avg** | **31.79** | **70.04** | **70.04** |
| **House** | **%** | **% Pruned** | **Null Model** |
| **Fold 1** | 60.92 | 61.43 | 61.43 |
| **Fold 2** | 61.36 | 61.43 | 61.43 |
| **Fold 3** | 61.63 | 61.43 | 61.43 |
| **Fold 4** | 61.36 | 61.76 | 61.76 |
| **Fold 5** | 38.37 | 61.43 | 61.43 |
| **Avg** | **56.73** | **61.50** | **61.50** |

## Using Median instead of Mean Forest Fire and Machine MSE

The results for Forest Fire and Machine show that the MSE is still subject to “swings” in accuracy even after pruning, making the average MSE suffer during five-fold cross validation. In the case of Machine, it is enough to make the Decision Tree performance worse than the Null Model.

Given the large range of output values in both datasets that we do not see in the other regression set (Abalone), for these two sets I will try setting leaf values in the tree to the median of the associated nodes rather than the mean. Results of re-running for mean and median are shown below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Dataset/Fold** | **MSE (Mean)** | **MSE (Mean) Pruned** | **MSE (Median)** | **MSE (Median) Pruned** |
| **Forest** |  |  |  |  |
| **Fold 1** | 17,960.99 | 15,060.54 | 13,395.34 | 1,103.74 |
| **Fold 2** | 1,116.42 | 1,273.92 | 947.26 | 14,876.95 |
| **Fold 3** | 288.04 | 7,749.13 | 546.06 | 260.29 |
| **Fold 4** | 460.13 | 252.66 | 237.33 | 8,714.85 |
| **Fold 5** | 1,560.40 | 626.58 | 11,927.50 | 680.68 |
| **Avg** | 4,277.20 | 4,992.57 | 5,410.70 | 5,127.30 |
| **Machine** |  |  |  |  |
| Fold 1 | 46,727.10 | 11,829.50 | 65,357.69 | 63,940.06 |
| Fold 2 | 26,440.74 | 11,822.06 | 51,262.25 | 23,448.22 |
| Fold 3 | 13,978.90 | 44,920.32 | 4,442.48 | 10,216.19 |
| Fold 4 | 15,224.93 | 22,069.21 | 25,071.79 | 10,038.26 |
| Fold 5 | 48,288.51 | 32,619.89 | 10,827.07 | 30,092.88 |
| **Avg** | 30,132.03 | 24,652.19 | 31,392.26 | 27,547.12 |

The results show that mean is the better choice for generating the trees of both sets, since the MSE is greater for the median for both pruned and unpruned trees. Nor are the “swings” in the fold performances avoided for median, as some folds’ MSE is notably greater than others.

# Discussion

Across the board, Decision Tree accuracy was low, even compared to Project 1’s Null Model when the tree was unpruned, reflecting my hypothesis that it would overfit the data. Individual folds also had markedly different results when unpruned. Pruned, the performance of both trees increased and the folds showed much more stability.

## Abalone

After pruning the accuracy was greatly increased for this dataset and MSE values became much more stable after pruning. I believe this was due to the identical ranges among the features and the narrow range of the target value.

## Breast Cancer and House

## In both datasets the accuracy was generally high both pruned and unpruned, although individual folds could show a markedly lower performance. When pruned the high performance was maintained and the results across folds were much more stable.

## Forest Fire and Machine

## While on average, both datasets performed better when the decision tree was pruned versus unpruned, in both cases the folds were subject to ‘swings’ in value where the MSE would be much higher or lower than the fold average. Switching the calculated leaf values to use median instead of mean only worsened the performance.

## Complexity likewise does not explain the difference between these datasets and the more stable abalone, as the machine dataset, which has by far the highest MSE of the three regression sets, has the same number of features as abalone.

## Car

## In this dataset accuracy was generally low for unpruned and high when the tree was pruned, indicating that this particular dataset is prone to overfitting.

# Conclusion

In this project both Classification and Regression Decision Trees were successfully modeled and run on all the datasets. The resulting trees were then pruned and re-tested. Both tests were conducted using five-fold cross validation.

The results appear to have confirmed my hypothesis. The performance of both trees improved across all the datasets after pruning, indicating that the unpruned tree was prone to overfit the data. The individual folds’ performance was also more stable after pruning, with most of the datasets showing broadly similar fold performance as opposed to the more variable performance on the unpruned dataset.

Interestingly, the number of features correlates to the performance of the classification datasets, with the best-forming dataset (car) having the fewest features and the worst (house votes) having the most. Since this is not the case with the regression sets and the breast cancer dataset’s performance is close to house votes despite having only about half the features though, I do not think any broader conclusions should be drawn from decision tree performance and number of features from this.