Project 1 :
Decision Tree

Girish Kasiviswanathan

February 25, 2016

1 Goal

The goal of this project is to implement and test a decision tree classifier on standard data-sets, and also measure the effectiveness of pruning algorithms by performing k-fold validation.

2 Datasets

The datasets selected from the UC Irvine Machine Learning Repository for this project are, *Car, Breast Cancer, Pima Indians Diabetes, Phising, Iris, and Mushroom.*

2.1 Preprocessing

The preprocessing accepts a control file that has information about the dataset location, class and attribute column numbers, names and types(discrete or continuous). The control file also has information such as separator character used in the raw data, and supports other necessary metadata we might require in future. It then converts the raw input into the format $< dataset, labels >$, and creates a metadata object that is visible throughout the program.

2.1.1 Missing Values

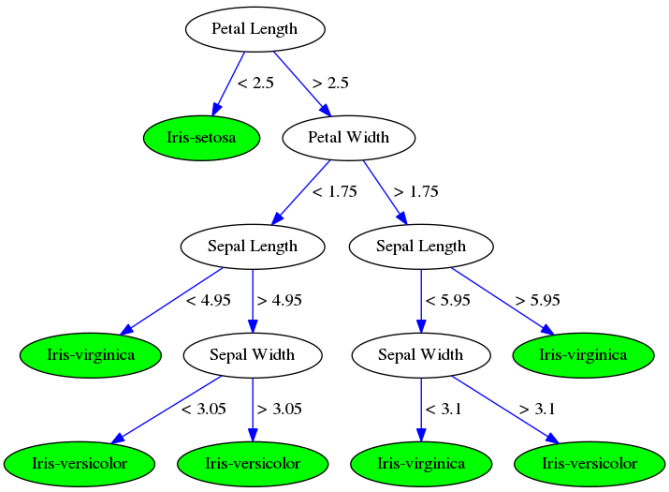
Missing values are replaced by the mode of the distribution in case of discrete attributes, and by the mean of the distribution in the case of continuous attributes.

2.1.2 Control Files

The control files for the datasets mentioned in this report, have already been generated in the pre-processing phase. Refer to `control.json` within the respective data folders for the template of control files. We can also add more information such as distribution statistics, thresholds, and additional data parsing parameters. The mandatory parameters in the control file include `class name, class position, data location, attribute types and attribute names`

3 Tree Building and Usage

The tree structure adopted is of the following form, and is constructed recursively by splitting examples and passing them down to their respective subtrees.



The members of the node class are :

Next splitting attribute : Holds the next attribute to split on

Parent split point : This holds the splitting value of the edge from the parent to this node (discrete attribute value in case of discrete values, and the splitting point for continuous attributes)

Class label : Leaves have a terminal class value. For other nodes, this is set to NULL

IsLeaf : Denotes if a node is a leaf or not

Majority class : Holds the majority class of the examples at that node

3.1 Stopping Criteria

The decision construction stops, and creates a leaf with the class label set to the majority class label at that point, when it encounter one of these cases:

1. All the examples in a node belong to the same class
2. There are no attributes left to split on
3. All the attributes left in the attribute set, has only one value left in their domain (*i.e* A case where all the examples would go down a single branch)

3.2 Splitting Criteria

The splitting criteria used is information gain. At each iteration of building a subtree with the remaining attributes, we identify the attribute that gives us the largest information gain. Next, this attribute with maximum gain is removed from the list of attributes that are sent down along with the partition of examples, to the next iteration of the subtree construction.

Information Gain $= S - S_{A_i}$

,where S is the entropy of the dataset prior to splitting, and S_{A_i} is the weighted entropy of the all the partitions obtained by splitting on A_i.

3.2.1 Discrete Attributes

For discrete attributes, to find the weighted entropy, we simply create one branch for each value in the domain of that attribute, and pass the examples down their respective branches.

3.2.2 Continuous Attributes

To handle continuous attribute, we have to find the point at which at which we can generate two disjoint intervals within the range of the continuous attributes in the training data. To find this point, the domain of the attribute, as observed in training data, is sorted, and we treat the mid point of every pair of consecutive values as a prospective split point. By iterating over all the possible mid point, we can determine the one that gives us the smallest entropy.

3.3 Classification

Classification on test examples is done by traversing the tree until it hits a leaf node, upon which the class label of the leaf is returned. If there is no branch corresponding to the test example's attribute value (this happens in case of discrete values), then we simply return the majority label of that node.

4 Pruning

4.1 Classic Holdout

In the classic holdout method, we train the tree on 70% of the examples, and test it on the remaining 30%. This is a simplistic approach, and there is no pruning performed.

4.2 10-fold Validation with Pruning

In 10-fold validation, we make 10 splits on the data. We then iteratively choose each split as a test set, while using the other 9 splits for training.

Now, within these 9 splits dedicated for training, we reserve $\frac{1}{3}$ of it for validation, *i.e* for improving the learnt decision tree model by mitigating the effect of over fitting. The remaining $\frac{2}{3}$ is used for training for constructing the initial

decision tree model, which is vulnerable to overfitting. For example, with 100 examples, we would use 10 for testing, 60 for training, and 30 for pruning/validation.

For reduced error pruning, we perform multiple pruning iterations until we reach a point where the tree's accuracy on the validation set cannot be improved any further. At each round of pruning, the pruning algorithm tries to 'leafify' every node (i.e temporarily convert it to a leaf that returns in majority class as its class label), and checks if the accuracy improves on the validation/pruning set. It finally selects the node that gave the highest improvement (or reduction in error), and pruned it away permanently, and moves on to the next round. It terminates when all possible pruning options cause an increase in error.

Figure 1 and Figure 2 show the unpruned and pruned versions respectively, for one of the folds of running the algorithm on the Breast Cancer dataset.

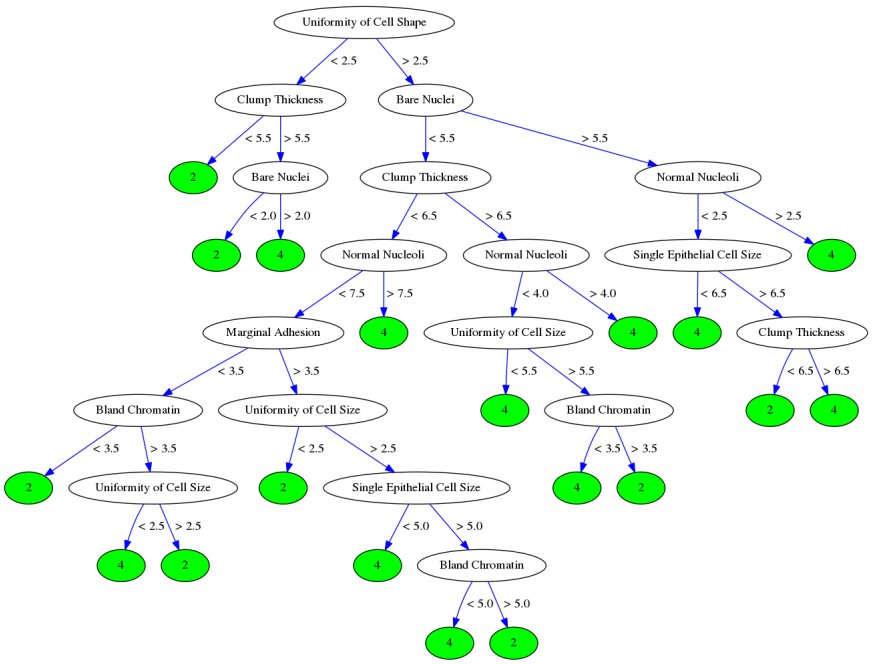


Figure 1: The original tree for Breast Cancer dataset

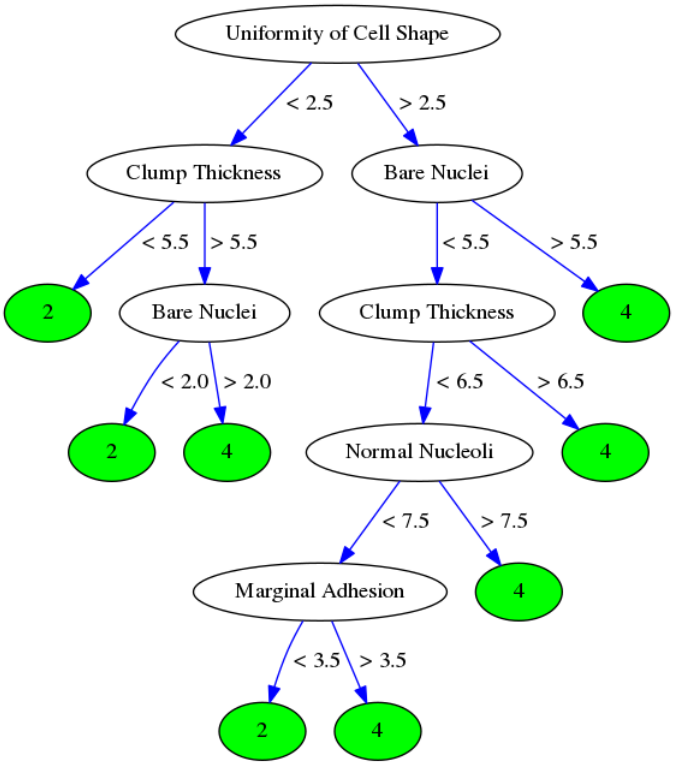


Figure 2: Pruned version of the tree above

5 Implementation

The implementation consists of 5 core modules:

- Preprocess : preprocess raw data
- Decision Tree : tree building and printing
- Classification : classify unseen data using built tree
- Pruning : prune a built tree using validation set
- Driver : splits data and runs the main k-fold validation proces

6 Results

For each of the datasets, we report the accuracies obtained on each fold using majority classifier, decision tree without pruning, and decision tree with reduced error pruning.

Refer to `Results.pdf` for the set of results. Also, refer to the folder `Tree Visualization` for images of the trees generated in the 10 folds, for each of the datasets.

7 Thoughts and Analysis

- On large trees such as that generated for the Phising Dataset (800 nodes), the pruning takes a very long time, as it removes just one node at a time. For k-fold validation, this gets worse, and takes almost 5 to 8 minutes for the process to complete on this dataset.
- For datasets such as Iris and Mushroom, which have clearly separable classes, the pruning improves the accuracy while also making it less complex. This clearly shows that training set has a severe over fitting issue. Occasionally, pruning causes a minor dip in the accuracy on the test set, such as in Phishing, but compensates by reducing the size of the tree. This shows that the concept being learnt needn't be as complex as one that fits the entire training set.
- For the Pima Indian Diabetes dataset, the class cannot be determined using a conjunctive hypothesis by our decision tree. This suggests that we might need to use a rule based pruning approach to account for disjunctive hypotheses also, which will great increase the time complexity of this algorithm. Reduced error pruning however, does significantly improve accuracy on this dataset, and yields a much simpler tree.
- The results are heavily dependant on the class distribution in each partition. It is observed that sometimes the tree is pruned to less thn $\frac{1}{10}$ of its original size, while in some folds, not even a single node is pruned. In both cases, the error reduction remains approximately same. Sub-sampling and informed partitioning technqies might be able to resolve this issue.
- Reduced errror pruning greedily increases the accuracy on the validation set (as it is a hill climbing algorithm), but this does **NOT** guarantee an increase on the test set.
- Reduced Error pruning can be made less greedy checking the statistical signifance of the improvements obtained on the validation set