CS 6140: Machine Learning **Spring 2018**
College of Computer and Information Science
Northeastern University
Lecture 1 **January, 8**
Instructor: Bilal Ahmed **Scribe: Bilal Ahmed & Virgil Pavlu**

Decision Trees: Pitfalls

1 Overfitting in Decision Trees

Overfitting occurs when the decision tree has low classification error on the training data, but considerably higher classification error on test (previously unseen) data. Figure-1 shows an example of overfitting and we can see that as the number of node in the tree grow its accuracy increases on training data but the accuracy on test data gets lowered. As the size of the tree grows, the decision tree starts to capture the peculiarities of the training data itself such as measurement noise instead of learning a general trend from the data which better characterizes the underlying target concept.

There are a number of methods that can be used to avoid growing larger and deeper trees.

- **Pre-Pruning (Early Stopping Criterion):**We can incorporate the preference of learning shorter trees within the tree growing process by imposing a limit on:
  1. Maximum number of leaf nodes
  2. Maximum depth of the tree
  3. Minimum number of training instances at a leaf node

  Remark: Care should be taken while setting these limit, because stringent limits can result in a shallow tree that is unable to adequately represent the target concept we are trying to

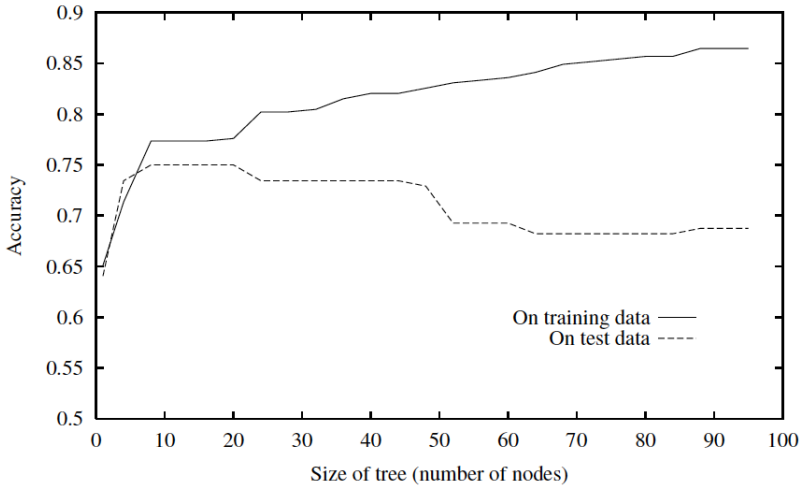

Figure 1: Overfitting.

learn. There are no general rules to how these limits should be set and usually these limits are dataset specific.

- **Reduced Error Post-Pruning:** Another strategy to avoid overfitting in decision trees is to first grow a full tree, and then prune it based on a previously held-out validation dataset. One pruning strategy is to propagate the errors from the leaves upwards to the internal nodes, and replace the sub-tree rooted at the internal node by a single leaf node. By labeling this new leaf node as predicting the majority class, we can calculate the reduction in validation error for every node in the tree. The subtrees rooted at the node with error reduction can be pruned and replaced by a leaf node. The algorithm is summarized as:

Data: Validation Data D_{vd} and fully grown tree T_{full}
Result: Pruned Decision Tree T_{pruned}
T=T_{full};
while *all nodes of T are tested* **do**
  select an inner node $t \in T$;
  construct T': replace t with a leaf node using training data;
  if $error(T', D_{vd}) \leq error(T, D_{vd})$ **then**
    $T = T'$;
  end
end

Algorithm 1: Algorithm for reduced error post-pruning of decision trees.

- **Rule-based Post-Pruning:** In rule-based pruning, a decision tree is converted into an equivalent set of rules. This is done by generating a separate rule for each path (root to leaf) in the decision tree. For example, in the case of the decision tree shown in Figure-**??** the right-most path translated into the rule:

IF (*Patrons* = 'Full') **AND** (*WaitEstimate* = '0-10') **THEN** (*WillWait* = 'Yes')

The pruning strategy is then to refine each rule by removing any preconditions or antecedents that do not increase the error on validation data. So, in the first step for the rule given above the following two rules will be tested:

1. **IF** (*Patrons* = 'Full') **THEN** (*WillWait* = 'Yes')
2. **IF** (*WaitEstimate* = '0-10') **THEN** (*WillWait* = 'Yes')

If any of these rules have an error that is lower than the original rule, then we would replace the original rule with the new pruned rule. The pruning process will recurse over the new (shorter) rules and halts when all the pruned rules are worst than their un-pruned version. At the end of the pruning procedure all the rules are sorted based on their accuracy over the validation data, and are used in this sequence to classify new instances.

2 Missing Values

In most real world datasets, there are missing feature values in the data. There are multiple reasons that can account for missing data, for example device failure, unanswered survey question, etc. In the case of missing feature values in our training data, we can:

- throw away incomplete instances (if we have enough training data)
- throw away the feature if a very high number of instances (e.g., 70%) have missing feature values
- fill in the missing values based on the overall feature mean
- fill in the missing values based on the class mean for the missing feature
- in certain cases where a missing value has its own specific meaning for example in the case of a medical test where a certain hormone is not detected we can create a new feature value to represent that the value is missing (not measured)
- if we come to an internal node and the feature value is missing, we can distribute the instance to all the child nodes with diminished weights that reflect the proportion of instances going down to each child

3 Different misclassification costs

Consider the task of classifying between spam and non-spam email messages. In this case classifying something that is non-spam as spam entails a higher cost than labeling a spam message as non-spam. Similarly, for problems arising in the medical diagnosis/evaluation domain false negatives have a higher cost than false positives. Assume that we have a binary classification task, and have used a decision tree to predict the labels for our training dataset. The results can be summarized using a confusion matrix as shown in Figure 2. Usually the confusion matrix is used in the context of binary classification problems (one of the classes is designated as negative and the other is designated as positive), however we can extend the notion of a confusion matrix to more than two classes as well. For m classes we will have a mxm matrix that can interpreted in a manner analogous to the two class problem.
So far, we have dealt with uniform costs i.e., the cost of predicting a false positive or a false negative is equal (unit cost, every mistake counts as one). However, in domains where we care about what errors are more costlier than others, we need to incorporate this information into the tree induction process, so that the final decision tree gives more attention to reducing the types of error that have a high cost. To this end we can specify the misclassification costs for a $k-$class classification problem using a $(k \times k)$ loss/cost matrix that is defined as follows:

$$\begin{bmatrix} 0 & L_{12} & L_{13} & \dots & L_{1k} \\ L_{21} & 0 & L_{23} & \dots & L_{2k} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ L_{k1} & L_{k2} & L_{k3} & \dots & 0 \end{bmatrix} \tag{1}$$

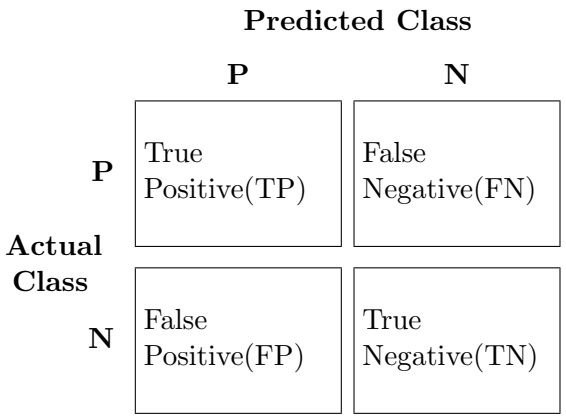

Figure 2: Confusion matrix for a binary classification task. We have two classes positive (P) and negative (N).

where, L_{ij} represent the cost of misclassifying an instance belonging to the i^{th} class as belonging to class j. The diagonal entries represent the cost associated with predicting the correct label which is zero. These costs can be integrated into Gini index, which for a node q in the decision tree is given as $Gini(q) = \sum_{k=1}^{M} p_{qk}(1 - p_{qk})$ for a classification problem with M classes. Note, that we can equivalently represent Gini index as $Gini(q) = \sum_{k \neq k'} p_{qk} p_{qk'}$ (*Homework problem*), and then modify it as:

$$Gini(q) = \sum_{k \neq k'} L_{kk'} p_{qk} p_{qk'} \tag{2}$$

This approach works only for classification problems that have more than two classes.

3.1 Instance Weighting

Another method to incorporate different misclassification costs is to associate a weight w_i with each instance x_i. Instead of counting instances to estimate class frequencies (for estimating information gain), instance weights are summed to represent class frequencies. For example, consider a classification task where we have to predict whether a patient is at a higher risk of a heart attack or not. In this case, the cost assoicated with a false negative (a patient who is high risk is classified as low risk) carries a much higher weight than a false positive (a low risk patient classified as high risk). Let's assume the cost associated with false negatives is ten times the cost associated with false positives. In this case, we can assign a weight of ten to all positive (high-risk patients) instances in our training data, and unit weights to the negative (low-risk patients) instances. Note, that if instance weighting is used then all the calculations must use instance weights i.e., while inducing, pruning and testing the tree.