Generalization Error on Pruning Decision Trees

Ryan R. Rosario
Computer Science 269 – Fall 2010

A decision tree is a predictive model that can be used for either classification or regression [3]. Decision tree construction algorithms take a set of observations \( \mathbf{X} \) and a vector of labels \( \mathbf{Y} \), and constructs a tree that attempts to build a model that describes the data based on specific splits in values of the input variables. Each interior node of such a tree represents a single input variable and each leaf of the tree represents the predicted class label of an observation whose values follow the path from root to that leaf.

One of the major problems with decision trees is the tendency for a raw decision tree to overfit the training data. In these cases, the decision tree contains many paths from the root to each leaf that are not of any statistical validity, and provide little predictive power to the model [2]. That is, the decision tree model does not generalize well to new, unseen values of \( \mathbf{X} \). If \( T \) is a decision tree and \( S \) is the sample of observations used to build \( T \), then pruning is the method by which one discovers a subtree of \( T \) that provides a generalization error bound not much worse than the optimal pruning that actually minimizes the generalization error associated with \( T \) [4].

In this paper, I describe two benchmark pruning methods whose error bounds depend on the data itself, and not just the size of the tree or sample. The layout of this paper is as follows. In section 1, a short taxonomy of decision trees and the mechanics behind their construction is presented. In section 2, two methods for pruning decision trees are discussed as well as an analysis of the generalization error associated with each. In section 3, I wrap up with a comparison of the algorithms and the generalization error between the pruning methods.

1 Introduction to Decision Trees

In the decision tree problem, we have a set of attributes \( \mathbf{X} \) and a set of discrete labels \( \mathbf{Y} \). In this paper, I focus on classification trees, where each observation has
associated with it a categorical label. Another form of decision trees, the regression
tree, attempts to predict a real number label from each observation [3].

There are several ways to construct decision trees and different metrics that
can be used. At each iteration of a decision tree algorithm, the algorithm chooses
an attribute to use to split the data and assigns it to a node. There are several
different loss functions that can be used to determine which variable to assign to a
node including entropy [7], and Gini [6]. Some common algorithms for constructing
decision trees are ID3 [7], CART [1] and C4.5 [8]. Figure 1 shows an example of a
trivial decision tree that classifies weather outlooks into weather for playing tennis
or not playing tennis.

\[
\forall \delta S \forall T \epsilon(T) \leq \hat{\epsilon}(T) + \sqrt{\frac{\left(\log 2\right)|T| + \log \frac{1}{\delta}}{2|S|}} \tag{1}
\]

Figure 1: a decision tree that takes as input a weather outlook and outputs whether

2 Pruning Decision Trees

Much research has been done in the pruning of decision trees to prevent the overfitting
problem, where one uses too many parameters (variable splits) to build a model.
According to [5], the standard technique for dealing with this problem is to prove
a bound on the generalization error as function of both the training error, and the
concept size or VC dimension. Then, we wish to select the concept minimizing this
error bound. A typical error bound, which was proven in [5] is given by
which basically states that for any arbitrary concept class $\mathcal{C}$ where every concept $T$ can be represented by some bitstring of length $|T|$, with probability $1 - \delta$ over the sample $S$ we have that each concept have their true error is bounded above by the sum of the training error $\hat{\epsilon}$ and some penalty that depends on the sizes $|S|, |T|$ and $\delta$.

2.1 Pruning by Finding Optimal Root Fragments

The major problem with the bound presented in (1) is that it relies only on the size of $T$ and the number of observations $|S|$. Statisticians, particularly those that study non-parametric statistics, would argue that such error bounds should be constructed from the data itself, and should not solely depend on the size of the model or the size of the data alone.

2.1.1 Error Bound on Pruning

In [5] an error bound that depends on the concept $T$ as well as the sample $S$ was shown. First, they arbitrarily divide the nodes in some tree $T$ into shallow nodes and deep nodes. Take $R$ to be an arbitrary set of shallow nodes such that $R$ forms a subtree of $T$ where each node in $R$ is either a leaf, or a parent of two other nodes also in $R$. Also, let $L(R)$ be the set of all leaves of this subtree $R$. Note that $R$ is also called a root fragment. Then we have that

$$\forall \delta \forall T \in T(\mathcal{H}, S), \epsilon(T) \leq \hat{\epsilon}(T) + \min_R f(T, S, R, \delta)$$

(2)

where

$$f(T, S, R, \delta) = \sum_{v \in L(T)} \frac{|S_v|}{|S|} \left( \sqrt{\frac{|T_v| \log 2}{2|S_v|}} + 2 \sqrt{\frac{|A_v|}{|S_v|}} + 2 \sqrt{\frac{\log 2}{|S_v|}} \right)$$

Let $v$ represent some vertex in the tree, and $A_v$ represents the path from the root to $v$, and $|S_v|$ is the number of observations that reach $v$. Additionally, $|S|$ is the size of the sample, $|T_v|$ is the number of nodes contained in the subtree of $T$ rooted at $v$ and $|A_v|$ is the number of terms in the predicate that lead to $v$. To efficiently prune the decision tree $T$, we want to select trees $T_v$ that minimize $\hat{\epsilon}(T) + f(T, S, R, \delta)$ and we can optimize each tree $T_v$ independently. I feel that being able to optimize these subtrees independently is a huge win because these optimizations can be parallelized to speed up computation. The authors indicate that this penalty that they have devised is meant to penalize prunings that result in decision tree models that are
overly complex, and favors models that are smaller and less complex. The first term, \(|S_v|\) seems to have been chosen as to penalize trees where \(|S_v| > |S|\); the algorithm favors smaller trees. The first term in the parentheses adds an additional penalty on trees containing nodes where the size of the tree rooted at \(v\) is larger than the number of observations reaching node \(v\). If \(|S_v| \gg |T_v|\), only a small (perhaps negligible) amount is added to the penalty. The second term in parentheses penalizes subtrees containing nodes with a large number of predicates relative to the number of observations reaching \(v\). In statistical modeling, it is preferable to have \(n \gg p\) [3]. That is, we want the number of observations to be much greater than the number of parameters otherwise we suffer from overfitting. The requirement that \(n \gg p\) is analogous to \(|S_v| \gg |A_v|\). The final term relates the number of observations reaching \(v\) with \(\delta\). Since \(\delta\) is very small, this term would generally be negligible if \(|S_v|\) is large.

2.1.2 Choosing an Optimal Root Fragment

The error bound I have discussed is based on knowing some best root fragment \(R\). To take advantage of this error bound, we must find the best root fragment of \(T\). The authors note that the best root fragment of \(T\) is either the best root fragment of the left subtree of \(T\) combined with the right subtree of \(T\), or just contains the root itself. This observation yields a bottom-up algorithm for finding the optimal root fragment. Let \(\text{compute}_R(T_v, S, |A_v|, \delta)\) compute the best root segment of subtree \(T_v\) and let \(\text{eval}(T_v, S_v, |A_v|, \delta)\) be a function that computes the penalty incurred by terminating the root fragment at the root of \(T\). Then the authors define

\[
\text{compute}_R(\text{IF}(B, T_l, T_r), S, d, \delta) = \min \begin{cases} 
\text{eval}(\text{IF}(B, T_l, T_r), S, d, \delta) \\
\frac{|S_B|}{|S|} \text{compute}_R(T_l, S_B, d + |B| + 2, \delta) + \frac{|S_{\neg B}|}{|S|} \text{compute}_R(T_r, S_{\neg B}, d + |B| + 2, \delta) 
\end{cases}
\]

where \(B\) represents a predicate of the root, and \(T_l\) and \(T_r\) represent the left and right subtrees respectively. If one assumes that predicates in \(T\) can be computed in constant time, then this algorithm implemented as a recurrence relation requires time proportional to the sum of the number of observations reaching each node, computed over all nodes. Using this algorithm, we have the error bound (2).

2.2 Sample-Based Pruning based on Locality

Kearns and Mansour propose an alternate method in [4] where the decision to prune or not to prune a subtree is made based solely on the data reaching the root of the subtree, and provides a new penalty that depends only the size of the pruned tree. The developers have also proven a strict performance guarantee from the algorithm
stemming from their work. First, the authors hypothesize some optimal pruning of $T$, called $T_{\text{opt}}$ that minimizes the generalization error and to which all pruning candidates are considered. Iterating through each node in the tree, the question of interest is whether or not we prune $T_v$ and replace it with a majority vote of the data reaching the node. Let $P_v$ be the probability distribution on only the observations $x$ that form a path through $T$ and pass through $v$. The observed error incurred by replacing $T_v$ with its best leaf is given by

$$\hat{\epsilon}_v(\emptyset) = \frac{1}{|S_v|} \min \{|\{x \in S_v : f(x) = 0\}|, |\{x \in S_v : f(x) = 1\}|\}$$

which is essentially just a probability. Iteration through the tree occurs “bottom-up” and when node $v$ is reached, every node below $v$ has already been processed. The decision to prune is made if the following inequality holds

$$\hat{\epsilon}_v(T_v) + \alpha(m_v, s_v, l_v, \delta) \geq \hat{\epsilon}_v(\emptyset)$$

where $\hat{\epsilon}_v$ is the fraction of errors $T_v$ makes on $S_v$ and

$$\alpha(m_v, s_v, l_v, \delta) = \left\{ \begin{array}{ll} c\sqrt{\frac{(l_v + s_v)\log n + n\log \frac{m}{\delta}}{m_v}} & \text{for } T \text{ finite} \end{array} \right.$$ 

where $m_v = |S_v|$, $s_v$ is the number of nodes in $T_v$, and $l_v$ is the depth of node $v$ in $T$.

An important fact about this algorithm is that the comparison between the subtree $T_v$ and the best leaf is made based on how generalization error changes based on the decision to prune, and not on the entire tree. Using concepts of local uniform convergence, one gets a bound on the generalization error

$$\hat{\epsilon}(T^*) - \epsilon_{\text{opt}} = O\left(\left(\log \frac{s_{\text{opt}}}{\delta} + l_{\text{opt}}\sqrt{\log n + \log \frac{m}{\delta}}\right)\sqrt{\frac{s_{\text{opt}}}{m}}\right)$$

I see that this bound has two major results. First, as the depth of the decision tree gets deeper, the order of the generalization error increases. This makes sense because deeper trees are more complex than shallower trees, and overfitting is typically associated with overly complex, and perhaps overly deep trees. Second, it appears that the larger the sample size $m$ is, the smaller the generalization error. This again reminds of the classic model fitting rule in statistics, that we want to have $n \gg p$, that is, we want a large sample size, and a smaller set of parameters.
3 Discussion and Conclusion

In this paper, I presented two benchmark papers in the analysis of pruning for decision trees. Both methods used pruning as a vehicle for reducing the generalization error and reducing cases of overfitting the data \( S \) with some model \( T \). Both methods feature a “bottom-up” sweep of the nodes in \( T \) in linear time. Also, both methods prune the decision tree in real time, and compute some loss for choosing when to prune. There are some subtle differences between both methods.

The method proposed by Mansour and McAllester relies on first computing the optimal “root fragment” \( R \) of \( T \) which is some abstract method called \texttt{compute}_R that accepts a subtree \( T_v \) as an input. The method moves through the tree and computes a penalty \texttt{eval} for terminating the root fragment at \( T \). In this sense, the algorithm has a generative feel; it is building up a subtree and terminating when some condition is met. The bound on the generalization error has embedded in it several penalties: a penalty on complicated predicates, a penalty on the size of the subtree versus the number of elements reaching a particular node \( v \), and a holistic penalty on proportion of the observations reaching a node \( v \).

The method proposed by Kearns and Mansour does not rely on finding any root fragment. Rather, the algorithm makes a decision at each node whether or not to replace the current node with the best leaf. Unlike Mansour and McAllester, the loss is also based on the depth of the decision tree in addition to the size of the tree. In this sense, the algorithm has a more destructive feel; it collapse subtrees into leaf nodes when a certain condition is met. One important difference between these two approaches is the amount of information on which the error bound depends. \cite{5} uses a lot more information about the intricacies of the data in the decision tree, whereas \cite{4} relies on only the sample size and the size of the tree. I suspect that over the set of all trees \( T \), the error bound presented in \cite{5} is very “jagged” and getting a handle on the numerical bound would be somewhat unpredictable if data analysis is not performed beforehand. On the other hand, the error bound presented in \cite{4} requires less information, appears to be “smoother”, and does not place such an emphasis on the intricacies of the data. From these observations, I suspect that Mansour and McAllester’s method would yield superior performance on small decision trees, whereas Kearns and Mansour’s method would yield superior performance on much larger decision trees with larger samples.
References


