Machine Learning: Foundations	**Spring semester, 1996/7**
Lecture 9 **Decision tree pruning**	May 21, 1997
Lecturer: Yishay Mansour	*Scribe: Mariano Schain*

10.1 Introduction

In a previous lecture we examined methods to build decision trees based on given examples. While building the trees we tried to keep its size as small as possible, yet, the resulting tree classified correctly *all* the examples. As seen in the previous lecture, this property might cause an over fitting effect (the resulting tree adapts itself to the *noise* in the sample set rather than the true underlying structure, and will probably fail to classify unseen instances).

The goal of this lecture is to present methods to *prune* the unnecessary nodes of the resulting decision tree. The problem is actually of *model selection* - the model size parameter is the tree size and the trade off is with the accuracy of the tree on the given sample set. We will examine four methods, each facing the problem in a different manner.

10.2 Reduce Error Pruning

The first method splits the sample set into a training set and a test set. The training set is used to build the decision tree $\mathbf{T}$. And the test set is used to decide whether a node of $\mathbf{T}$ should be pruaned or not (The pruning of an internal node v is done by replacing the subtree of v by a leaf). In this section we describe how the decision is done.

First, we determine for every internal node v of $\mathbf{T}$ the label of v, if pruned. We set the label of the pruned v to be the majority class among the examples in the training set that reach v. We now use the test set to compute for each node v the number of examples that reach v, and the number of errors if the classification in v is done using the label of a pruned v.

The algorithm basically checks for each internal node v if pruning it improves the generalization ability of the tree. Specifically, we compare the computed number of errors over the test set of the tree pruned at v to the number of errors over the test set of all the subtree rooted at v (which is the sum of the errors of all the leaves of the subtree).

Intuitively, if a subtree represents over fitting, then the true error (approximated by using the test set) of v as a pruned leaf will be significantly smaller than the true error of the subtree rooted at v. We can therefore conclude the pruning rule: If the number of classification errors over the test set of v as a pruned leaf is significantly smaller than the error of the subtree rooted at v then we choose to prune v. If the number of classification errors over the test set of v as a pruned leaf is significantly larger than the error of the subtree rooted at v then we choose not to prune v. Otherwise (when there is no significant difference in the number of errors) we can make arbitrary choice (It is natural that we will prefer smaller trees).

The order in which nodes are checked is from the leaves up towards the root. While checking, the tree changes and the counts are updated accordingly.

To illustrate the algorithm we will use as an example the decision tree $\mathbf{T}$ in Figure 10.1. As the tree is not pruned yet, the leaf label is the class of all the examples in the training set that reach the leaf. Note that we can not assume anything like that for the examples in the test set.

We first examine the bottom right dotted subtree whose root is the node labeled X_4. In the test set there are 5 examples reaching the subtree, 2 of them to the left leaf and 3 to the right leaf. Assume that both examples that reach the left leaf have class label 1 and that among the three examples that reach the right leaf, two have class label 1 and one has class label 0 (This is indicated in the figure by the numbers below the subtree.) The subtree error is therefore 2, while the error of X_4 as a pruned leaf is 1 (The label of the pruned leaf X_4 is 1 since in the training set 3 of the examples reaching X_4 have class label 1 and only one example has class label 0). We can therefore prune $\mathbf{T}$ at X_4. A similar analysis is done for the subtrees whose roots are X_6 and X_3 (Note that when considering the X_3 subtree the counters are changed since we already pruned at X_4). The resulting tree is presented in Figure 10.2.

Another possible pruning method is illustrated in figure 10.3 Where an internal node is replaced by one of its subtrees. The pruning rule is similar to the one described above.

It can be shown that if the size of the test set is large enough then the error count at each node is close to the true error. Therefore by pruning we truly reduce (or not increase) the classification error.

This method (similar to CV) requires that we use only part of the given examples to build the decision tree (the rest is used as a test set).

10.3 Structural Risk Minimization

We will use dynamic programming to find for every k the best (with minimum number of errors) decision tree T_k of size k that is a pruned version of the given decision tree T. Obviously, the number of errors decreases as we increase k. We can apply the method of Structural Risk Minimization to chose the best T_k. We can alternatively use Cross Validation to test each T_k on an independent test set, and choose among the set of 'best' trees the tree with the least number of errors.

The main problem is finding for every k the best decision tree T_k of size k. Note that this problem is equivalent to finding for any e the smallest tree with no more than e errors.

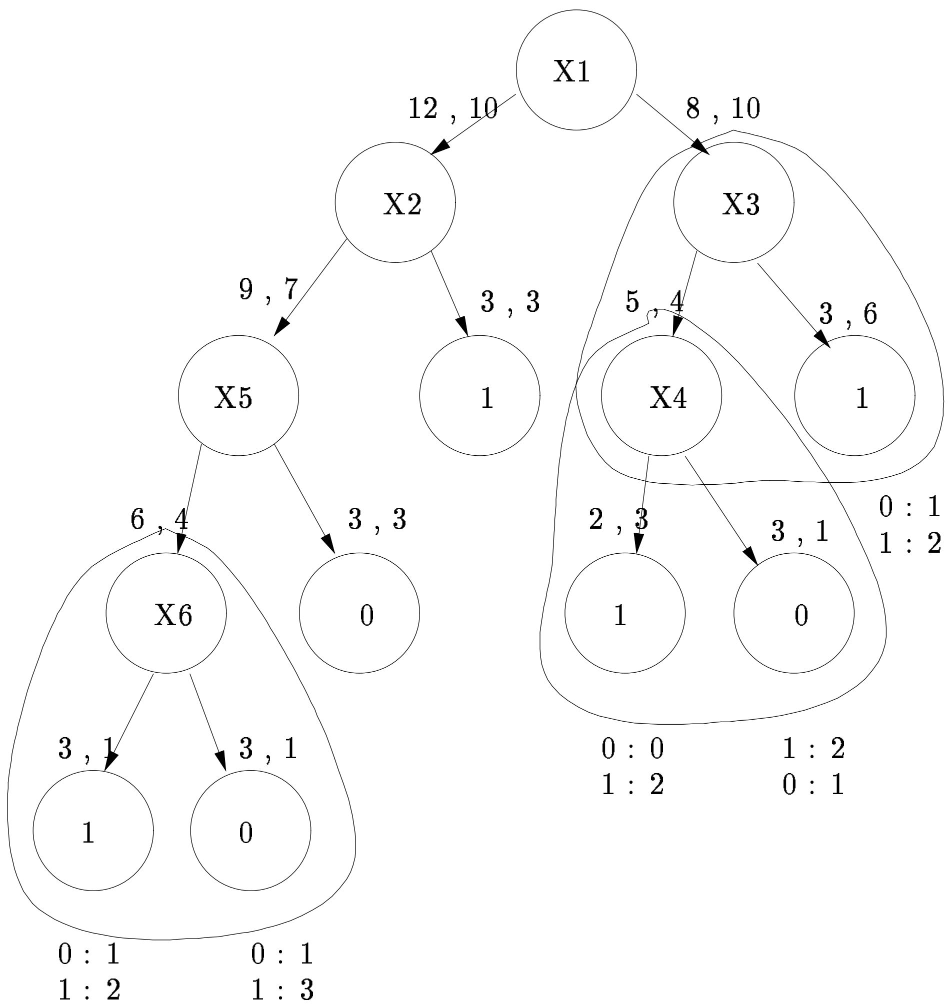

Figure 10.1: An initial decision tree **T**, about to be pruned. The numbers next to each edge are the number of examples that reach the edge when classified using **T**. (The left number is the count in the training set, and the right number is the count in the test set.) The X_i variable in each internal node is the attribute examined while making a decision in the node, and the number in each leaf is the classification made for examples reaching the leaf.

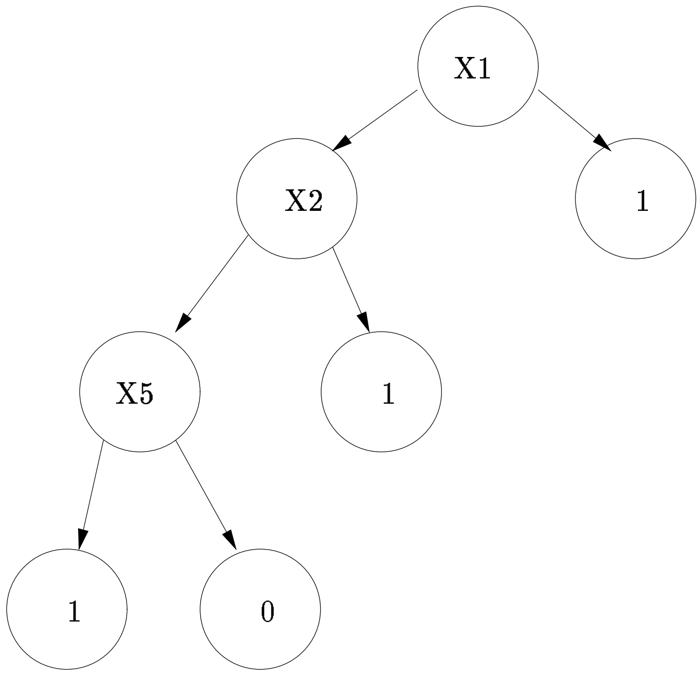

Figure 10.2: The resulting pruned decision tree

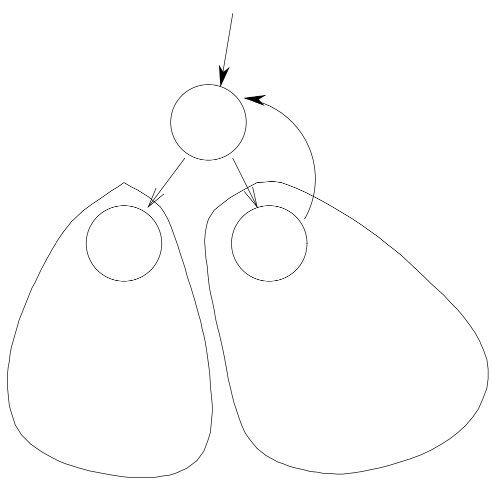

Figure 10.3: Another pruning method

Once we solve the latter problem it is only a matter of reversing a table to solve the former.

Given e, the number of errors, we want to find the smallest pruned version of T that has at most e errors. We give a recursive algorithm with exponential complexity. We use $\mathbf{T}[0]$ and $\mathbf{T}[1]$ to represent the left and right child subtrees of $\mathbf{T}$ respectively. root($\mathbf{T}$) is the root node of the tree $\mathbf{T}$. We also define $\mathbf{maketree}(r, \mathbf{T_0}, \mathbf{T_1})$ to be the tree formed by making the subtrees $\mathbf{T_0}$ and $\mathbf{T_1}$ the left and right child of the root node r. For every node v of $\mathbf{T}$, $\mathbf{Errors}(v)$ is the number of classification errors on the sample set of v as a leaf.

compute(input : k:numOfErrs,$\mathbf{T}$:tree,
    output : s:treeSize,$\mathbf{P}$:prunedTree)
    If **IsLeaf**($\mathbf{T}$)
        If $\mathbf{Errors}(\mathbf{T}) \leq k$
            $s = 1$
        Else
            $s =$infinity
        $\mathbf{P} = \mathbf{T}$
        return
    If $\mathbf{Errors}(\text{root}(\mathbf{T})) \leq k$
        $s = 1$
        $\mathbf{P} = \text{root}(\mathbf{T})$
        return
    For $i = 1 \ldots k$
        Call $\mathbf{compute}(i,\mathbf{T}[0],s_i^0,\mathbf{P}_i^0)$
        Call $\mathbf{compute}(k-i,\mathbf{T}[1],s_i^1,\mathbf{P}_i^1)$
    $I = \arg\min_i\{s_i^0 + s_i^1\}$
    $s = s_I^0 + s_I^1 + 1$
    $\mathbf{P} = \mathbf{maketree}(\text{root}(\mathbf{T}),\mathbf{P}_I^0,\mathbf{P}_I^1)$

The exponential complexity of the algorithm can be reduced by implementing the same idea in a different way. Note that we can compute all the values for a node if we know all the values for its two child subtrees. The basic idea is therefore to do the computation from the leaves up towards the root of the tree. If we have all the computations done for the subtrees $\mathbf{T_0}$ and $\mathbf{T_1}$ (We have computed s_i and the suitable tree $\mathbf{P_i}$ for every $i = 1 \ldots k$) then we can compute for $\mathbf{T}$ by scanning the already computed values k times ($i = 1 \ldots k$):

$I \;= arg\min_i\{s_i^0 + s_{k-i}^1\}$ ## this takes O(k)
$s_i \;= s_I^0 + s_I^1 + 1$ ## this takes constant time
$\mathbf{P}_i = \mathbf{maketree}(\text{root}(\mathbf{T}),\, \mathbf{P}_I^0,\mathbf{P}_I^1)$ ## this takes constant time

We also have to initialize all the values in the leaves of $\mathbf{T}$. To do that we must find for every leaf the exact number of errors over the sample set (if we have l errors in a leaf then we set $s_l = 1$ and all other s_i to infinity). The total time complexity is therefore $O(k^2 * s) + O(m * depth(\mathbf{T}))$ where s is the size of $\mathbf{T}$ and m is the size of the sample set. Since $k = O(m)$ we have total time complexity $O(m^2 * s)$.

Now that we have a way to compute for every k the best decision tree T_k of size k, we have to choose one of them as the best hypothesis. It is actually a question of Model Selection (k being the model size), and we can use the method of Structural Risk Minimization that was discussed in the previous lecture. As the VC-dimension d of the class of decision trees with k nodes is at least k (H.W. 3) we can choose according to the following rule:

$$T_* = arg\min_d\{err(T_d) + \sqrt{\frac{k}{m}}\}$$

As mentioned before, we can also use Cross Validation to test each T_k on an independent test set (not the one used to build the decision trees) and choose among the set of 'best' trees the tree with the least number of errors.

10.4 Pessimistic Pruning

We now present a pruning method that does not require a separate test set nor high time complexity. The idea is to make a single pass from the leaves of the decision tree up towards the root, and at every internal node make a decision wether to prune or not. The decision is based on a comparison of the error at the node (if pruned) to the error of the node subtree. Since the true error can only be approximated we make a pessimistic approximation.

Let H_k be the set of all decision trees with at most k nodes. For any boolean function f and $h \in H_k$ Define $e(f,h)$ to be the true classification error and $\hat{e}(f,h)$ be the error as measured on the given sample set.

We show now an upper bound on the probability that there is some $h \in H_k$ with $e(f,h)$ significantly different from $\hat{e}(f,h)$.

$$\begin{aligned} Pr[\exists h \in H_k \;\; s.t. \;\; |e(f,h) - \hat{e}(f,h)| > \Gamma_k] &\leq |H_k| Pr[|e(f,h) - \hat{e}(f,h)| > \Gamma_k] \\ &\leq 2|H_k| e^{-2\Gamma_k^2 n} \stackrel{def}{=} \delta \end{aligned}$$

Where the last inequality is Chernoff bound, and n is the size of the sample set.

We have

$$\Gamma_k = \sqrt{\frac{\ln 2|H_k| + \ln\frac{1}{\delta}}{2n}}$$

The above lemma implies that for any decision tree with k nodes, with high probability the true error is Γ_k-close to the observed error.

We now proceed to bound $|H_k|$, This will let us get a bound on Γ_k. Let $k = 2k^{`} + 1$ ($k^{`}$ internal nodes and $k^{`} + 1$ leaves). Ignoring the labeling of leaves and internal nodes, it is easy to see that there are at most 2^k different decision trees. (This is because we can map any such decision tree to a vector of k bits in BFS order, representing the internal nodes by 0 and leaves by 1). There are at most $d^{k^{`}}$ different labelings for internal nodes (We may choose any of the $d$ attributes $x_1 \ldots x_d$ to be tested at any internal node), and $2^{k^{`}+1}$ for the leaves. We conclude with the required bound

$$|H_k| = O((4d)^k).$$

Let n_v be the number of examples that arrive to node v. Let s be the size of the subtree rooted at v. Let e_1 be the number of errors at the subtree, and let e_2 be the number of errors if v is replaced by a leaf. By the above bound, with high probability, the true error of the subtree is at most $\frac{e_1}{n_v} + \Gamma_k$. (This is actually a pessimistic estimation of the true error of the subtree, hence the algorithm name). We will decide to prune only if the estimated error of v, if replaced by a leaf, is smaller than the pessimistic estimated error of the subtree:

$$\frac{e_2}{n_v} < \frac{e_1}{n_v} + \Gamma_s$$

The following claims justify the above rule.

Claim 10.1 *When we decide NOT to prune in the internal node v, with high probability the error of the subtree rooted at v is smaller than the error at v as a leaf.*

Claim 10.2 *When we decide to prune a subtree of size k, the total error of the tree increases at most by Γ_k*

Note that all such errors occur at independent nodes, therefore the total increase in the error of the tree is an average of those increases and is therefore bounded by Γ_k as well.

Since Γ_k is in the order of $\sqrt{\frac{k}{n}}$, We can adjust the pruning decision test with another parameter c

$$\frac{e_2}{n_v} < \frac{e_1}{n_v} + c\sqrt{\frac{s}{n}}$$

The parameter c controls the tendency of the algorithm to prune.

10.5 Pruning With Experts

In a previous lecture we saw an on-line algorithm that combined experts predictions to predict assimptotically as well as the best expert. Every possible pruning of the decision tree represents an expert, so by adapting the algorithm to the pruning problem we are assured to perform as good as the best pruned subtree. The algorithm, however, cannot be directly used since there are too many experts. We show here a method to overcome that efficiency.

10.5.1 An inefficient algorithm

Recall from Lecture 5. The algorithm (*Hedge(β)*) based its prediction ($\hat{y}_t$) given the instance x^t in time t on a weighted average of the experts predictions $p(x^t)$.

$$\begin{aligned} r^t &= \frac{\sum_p w_p^t p(x^t)}{\sum_p w_p^t} \\ \hat{y}_t &= F_\beta(r^t) \end{aligned}$$

Where $\beta \in [0,1]$ is a parameter of the algorithm, w_p^t is the weight of expert p at time t, and F_β need only satisfy certain properties (See Lecture 5).

The weights of the experts are updated at time t after the true classification y^t is revealed

$$w_p^{t+1} = w_p^t U(|p(x^t) - y^t|)$$

The total loss of the algorithm is L_A

$$L_A = \sum_{t=1}^{T} |\hat{y}^t - y^t|$$

And if the loss of the best expert is L_P then we are assured that

$$L_A \le \frac{L_P \ln \frac{1}{\beta} - \ln \frac{1}{w_p^1}}{1-\beta}$$

Note that the running time of the algorithm is linear in the number of experts (i.e. prunings), which in this case is enormous.

10.5.2 An efficient implementation

The problem with the above algorithm is that we cannot maintain all of the weights w_p^t explicitly since there are far too many prunings to consider. Note, however that the prediction $\hat{y}^t$ is based directly on r^t. We show here a method to efficiently compute r^t. First we prove a general result, and then consider its application to our problem.

A general result

Definition Let T be a decision tree. The *size* of a pruning P of T, written $|P|$, is the number of internal nodes and leaves in P *minus* the number of leaves in P that are also leaves of T.

Let $g(s)$ be a function on the leaves of a decision tree. We define the following function on the nodes of T as a weighted average of the following form:

Definition

$$\overline{g}(u) \stackrel{def}{=} \sum_{P \ of \ T_u} 2^{-|P|} \prod_{s \in \mathrm{leaf}(P)} g(s)$$

Where we denote by T_u the subtree of T rooted at u, and the notation $\sum_{P \ of \ T_u}$ is used to indicate summation over all prunings of T_u.

Lemma 10.3 *1. if u is a leaf then $\overline{g}(u) = g(u)$*

2. if u is an internal node then $\overline{g}(u) = \frac{1}{2}g(u) + \frac{1}{2}\prod_{v \in son(u)} \overline{g}(v)$

Note that computing bottom up, the lemma provides a methods to compute $\overline{g}(u)$ in time proportional to the number of nodes in T.

Proof:

1. The size of a pruning of a leaf of T is 0. The result trivially follows from the definition of $\overline{g}$.

2. Note that any pruning P of the subtree T_u either contains only node u or can be decomposed into two subtrees, P_0 and P_1, rooted at the children u_0 and u_1 of u. Separating the above two cases and using the easy to prove fact $|P| = 1 + |P_0| + |P_1|$ we have

$$\begin{aligned}
\overline{g}(u) &= \frac{1}{2}g(u) + \sum_{P_0}\sum_{P_1} 2^{-(1+|P_0|+|P_1|)} \prod_{w_0 \in \mathrm{leaf}(T_{u_0})} g(w_0) \prod_{w_1 \in \mathrm{leaf}(T_{u_1})} g(w_1) \\
&= \frac{1}{2}g(u) + \frac{1}{2}\Big(\sum_{P_0} 2^{-|P_0|} \prod_{w_0} g(w_0)\Big)\Big(\sum_{P_1} 2^{-|P_1|} \prod_{w_1} g(w_1)\Big) \\
&= \frac{1}{2}g(u) + \frac{1}{2} \prod_{v \in son(u)} \overline{g}(v)
\end{aligned}$$

■

Efficiently computing r^t

We now show how Lemma 10.3 is used to compute r^t. We do so by writing r^t in a form equivalent to $\overline{g}$.

For any node u we define the weight at time t to be its past contribution to the weight decrease of any tree P that contains u as a leaf:

$$\mathrm{WEIGHT}^t(u) \stackrel{def}{=} \prod_{\substack{1 \leq t^{\text{`}} < t \\ x^{t^{\text{`}}} \text{ passes through } u}} U_\beta(|value_{t^{\text{`}}}(u) - y^{t^{\text{`}}}|)$$

Where $value_{t^{\text{`}}}(u)$ is the predicted value at node $u$ in time $t^{\text{`}}$. For any pruning p, we have

$$\begin{aligned} w_p^t &= 2^{-|p|} \prod_{1 \leq t^{\text{`}} < t} U_\beta(|p(x^{t^{\text{`}}}) - y^{t^{\text{`}}}|) \\ &= 2^{-|p|} \prod_{u \in \mathrm{leaf}(p)} \prod_{\substack{1 \leq t^{\text{`}} < t \\ \mathrm{leaf}_p(x^{t^{\text{`}}}) = u}} U_\beta(|p(x^{t^{\text{`}}}) - y^{t^{\text{`}}}|) \\ &= 2^{-|p|} \prod_{u \in \mathrm{leaf}(p)} \mathrm{WEIGHT}^t(u) \end{aligned}$$

Thus, the denominator of r^t is

$$\sum_p w_p^t = \sum_p 2^{-|p|} \prod_{u \in \mathrm{leaf}(p)} \mathrm{WEIGHT}^t(u),$$

which has the requested efficiently computable form.

For the numerator, we define

$$\mathrm{WP}^t(u) = \begin{cases} \mathrm{WEIGHT}(u)value(u) & \text{if } x^t \text{ passes through } u \\ \mathrm{WEIGHT}(u) & \text{otherwise.} \end{cases}$$

Then for any pruning p we have

$$\begin{aligned} w_p^t p(x^t) &= 2^{-|p|}\left(\prod_{u \in \mathrm{leaf}(p)} \mathrm{WEIGHT}^t(u)\right)p(x^t) \\ &= 2^{-|p|} \prod_{u \in \mathrm{leaf}(p)} \mathrm{WP}^t(u) \end{aligned}$$

Since x^t arrives to a single leaf of p. Thus,

$$\sum_p w_p^t p(x^t) = \sum_p 2^{-|p|} \prod_{u \in \mathrm{leaf}(p)} \mathrm{WP}^t(u),$$

which also has the requested form as above.

10.6 Bibliographic Notes

In addition to the lecture notes, the following articles were used in the preparation of this scribe.

1. *Pessimistic Decision tree pruning based on tree size* by Yishay Mansour.

2. *Predicting nearly as well as the best pruning of a decision tree* by David P. Helmbold and Robert E. Schapire.