Machine Learning: Foundations	Spring semester, 2001/2
Lecture 10 **Decision tree pruning**	June 13, 2002
Lecturer: Yishay Mansour	*Scribe: Lior Robshitz Evyatar Shoresh*

10.1 Introduction

In lecture number 5 we introduced the model of *decision trees* for learning. As in other learning models we faced the problem of over-fitting, meaning: producing a predicator (in our case a decision tree) that would "over fit" the seen samples and would fail to achieve good results with high probability on the real distribution. In order to overcome this problem we added some arbitrary rules to the decision trees' building algorithm, such as, "don't create leaves that predict less that x samples". These arbitrary rules seemed to lack the ability of taking into consideration the structure of the specific problem, in fact they were absolutely general. In this lecture we'll introduce a different approach: after receiving a decision tree that describes the samples exactly, we'll start a procedure called **pruning** which will reduce the side of the tree based on the structure (distribution) of the specific problem. The pruning 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'll examine few pruning algorithms and finally show an efficient, linear time, algorithm for pruning.

10.2 Pruning overview

Aforesaid pruning is a procedure done on trees to reduce their size, with the thought that a smaller model has a better generality capability. A pruning procedure gets as input a sample S and decision tree T and outputs a new decision tree T'. This is correct with a small reservation: T' must be some transformation on T or otherwise this cannot be considered pruning. Apart from that T' can be any transformation on T, for example a simple pruning would output a sub-tree of T by replacing some of its inner nodes with leaves.

10.3 Reduce error pruning

The first pruning algorithm we'll introduce is called *reduce error pruning*. The *reduce error pruning* algorithm splits the sample given S into two samples S_1 and S_2. S_1 is used for building the decision tree T and S_2 is used for decisions of how to prune it. The pruning of an internal node v is done by replacing the sub-tree rooted at v by a leaf - v. In the first part of the algorithm we build the decision tree T using the algorithm presented in lecture number 5 without any additional rules. This will produce a tree that reflects the sample S_1 accurately. In the second part we traverse the tree bottom-up and on each internal node v we check whether we want to replace the sub-tree originated by it with a leaf or not. The decision of whether to prune the node v or not is rather simple: we sum the errors of the leaves in the sub-tree rooted by node v and the errors in case we replace it with a leaf. If replacing the sub-tree rooted at v will reduce the total error on S_2 we prune v. In case the number of errors is the same we also prune the node in order to minimize the outputted model size. Notice that on each step in the second part of the algorithm we examine the sub-tree that is relevant to the current stage in the algorithm, meaning that if we pruned part of the sub-tree rooted in v on previous iterations than when we check v we examine it against its current sub-tree and not the original one.

To illustrate the algorithm we will use as an example the decision tree T in Figure 10.1. As the tree is not pruned yet, the leaf label is the class of all the examples in S_1 that reach the leaf. Note that we can not assume anything like that for the examples in S_2.

We first examine the bottom right dotted subtree whose root is the node labelled 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 S_1 3 of the examples reaching X_4 have class labelled 1 and only one example has class labelled 0). We can therefore prune 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.

10.4 Structural Risk Minimization And Cross Validation

In this section we'll present an algorithm that produces few pruning possibilities. The algorithm will be wrapped by Structural Risk Minimization and Cross Validation that choose one of the pruning possibilities produced by the underline algorithm. Generally the core algorithm will be given a decision tree T and target number of errors ε and will produce the smallest pruned trees that has such errors rate.

10.4.1 Cross Validation algorithm

· Split the sample S to S_1 and S_2
· Build a tree using S_1

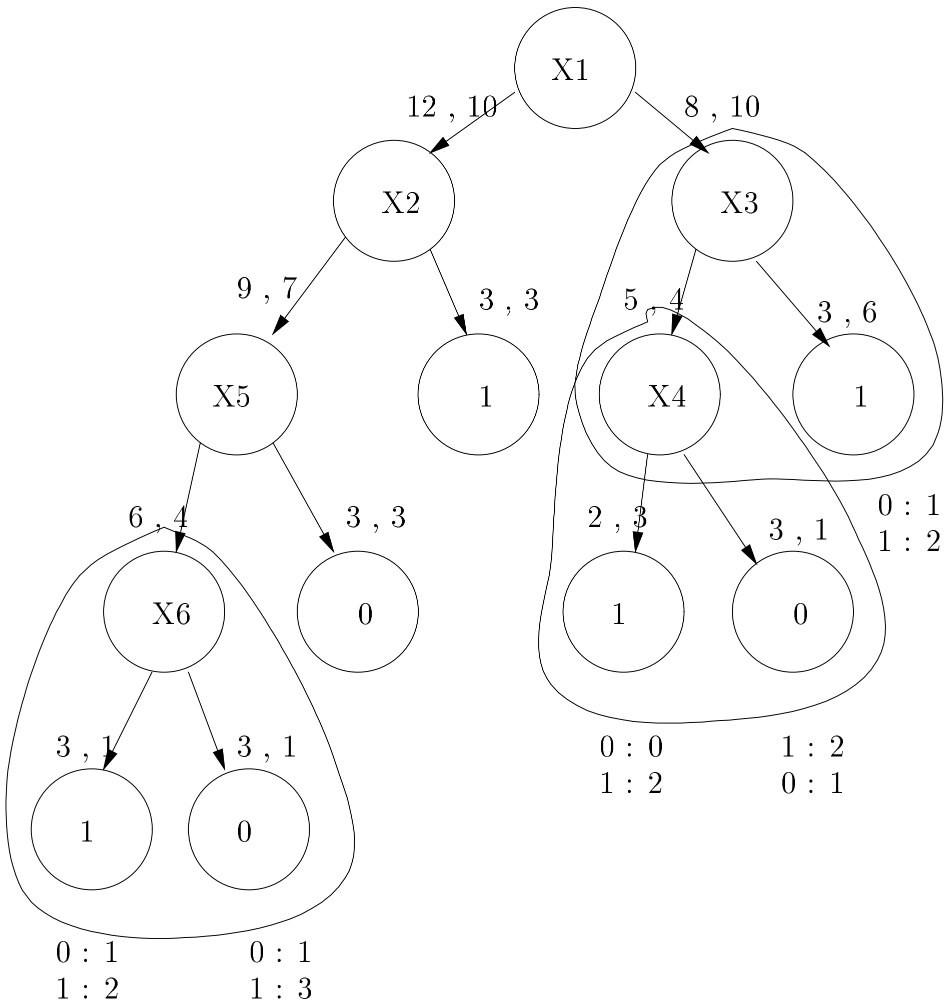

Figure 10.1: An initial decision tree T, about to be pruned. The numbers next to each edge are the number of examples that reach the edge when classified using T. (The left number is the count in the training set, and the right number is the count in the test set.) The X_i variable in each internal node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

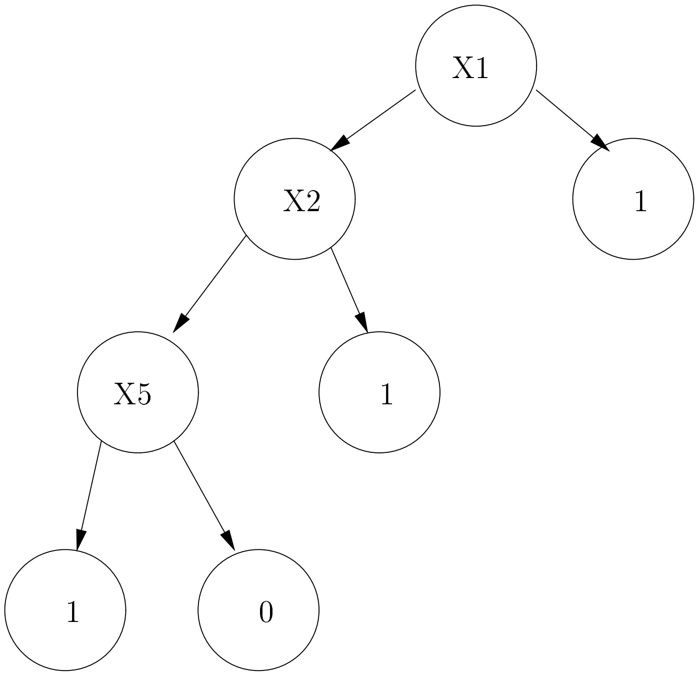

Figure 10.2: The resulting pruned decision tree

· Compute the candidate pruning
· Select one pruning using S_2
· Output the tree with the smallest error on S_2

10.4.2 Structural Risk Minimization algorithm

· Build a tree T using S
· Compute the candidate pruning
· K_d the size of pruning with d errors
· Select one pruning using **SRM** formula

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

Where k_d is the size of the T and m the size of S.

10.4.3 Finding the minimum pruning

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 **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 $\mathbf{T}$, **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 **Errors**($\mathbf{T}$) $\leq k$
    $s = 1$
  Else
    $s =$infinity
  $\mathbf{P} = \mathbf{T}$
  return
If **Errors**(root($\mathbf{T}$)) $\leq k$
  $s = 1$
  $\mathbf{P} =$ root($\mathbf{T}$)
  return
For $i = 1 \ldots k$

Call **compute**(i,$\mathbf{T}[0]$,s_i^0,$\mathbf{P}_i^0$)
Call **compute**($k-i$,$\mathbf{T}[1]$,s_i^1,$\mathbf{P}_i^1$)
$I = \arg\min_i\{s_i^0 + s_i^1\}$
$s = s_I^0 + s_I^1 + 1$
$\mathbf{P} =$ **maketree**(root($\mathbf{T}$),$\mathbf{P}_I^0$,$\mathbf{P}_I^1$)

The exponential complexity of the algorithm can be reduced by using dynamic programming method. 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 ($i = 1 \ldots k$):

$I \quad = arg\min_i\{s_i^0 + s_{k-i}^1\}$ ## O(k)
$s_i = s_I^0 + s_I^1 + 1$ ## O(1)
$\mathbf{P}_i =$ **maketree**(root($\mathbf{T}$), $\mathbf{P}_I^0$,$\mathbf{P}_I^1$) ## O(1)

We also have to initialize all the values in the leaves of 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 $O(k^2 * s) + O(m * depth(T))$ where s is the size of T and m is the size of the sample set. Since $k = O(m)$ we have total time complexity $O(m^2 * s)$.

10.4.4 Summery

We saw an algorithm for finding a small model using pruning by traversing a tree top-down and finding sub-trees that has at most error of some given ε. We also saw how to use **SRM** and **CV** methods in order to choose the best sub-tree among the outputs of the core algorithm. The main problem we had in this section is the complexity of the core algorithm. In the last section we'll introduce a similar algorithm traversing the tree bottom-up and running at linear time.

10.5 Linear time Pruning Algorithm

We want to use a pruning algorithm that will run in linear time (the naive Reduce Error Pruning algorithm in 10.1 has linear time, but its performance is not guaranteed).The algorithm that we shall describe will use a single bottom up pass on the tree. It will make its pruning decisions based on local infromation (of the subtree in question). Finally We shall prove a strong performance guarantee for the generalization error of the pruned tree achieved with this algorithm.

10.5.1 description

The algorithm is given the random sample S and a tree T as input (We do not assume the independence of T and S, actually we may assume S was used to build T therefore $T = T(S)$). The high-level structure of the algorithm is quite straightforward: the algorithm makes a single "bottom-up" pass through T, and decides for every node v whether to leave the subtree currently rooted at v in place (at least for the moment), or whether to delete this subtree. The algorithm only considers pruning at a node v once it has first considered pruning at all nodes below v; this simply formalizes the standard notion of "bottom-up" processing. When moving up the tree the relevant information is passed from node v to its parent upon pruning. Thus when we reach a node we have all the relevant information for the decision making.

Two observations are in order here. First, the algorithm considers a pruning operation only once at each node v of T. Second, the subtree rooted at v at that time when considering pruning may be *different* than T_v (the original subtree of T rooted at v), because parts of T_v may have already been deleted. We shall use the notation T_v^* to denote the subtree that is rooted at v when considering pruning. It is T_v^* that our algorithm decides whether to prune. We will use T^* to denote the final pruning of T output by our algorithm.

It remains only to describe how our algorithm decides whether or not to prune T_v^*. For this we need some additional notations: S_v will denote the set of all $x \in S$ that reach v , $m_v = |S_v|$, s_v denotes the number of nodes in T_v^*, and ℓ_v denotes the depth of the node v in T. We shall also use $\hat{\epsilon}_v(T_v^*)$ as the fraction of errors that T_v^* makes on the local sample S_v, and $\hat{\epsilon}_v(\emptyset)$ is the fraction of errors the best leaf (a constant 0 or 1 function replacing v) makes on S_v. Then our algorithm will replace T_v^* by this best leaf if and only if

$$\hat{\epsilon}_v(T_v^*) + \alpha(m_v, s_v, \ell_v, \delta) \geq \hat{\epsilon}_v(\emptyset) \qquad (10.1)$$

where $\delta \in [0, 1]$ is a confidence parameter.

The exact choice of $\alpha(m_v, s_v, \ell_v, \delta)$ will depend on the setting, but in all cases can be thought of as a penalty for the complexity of the subtree T_v^*.

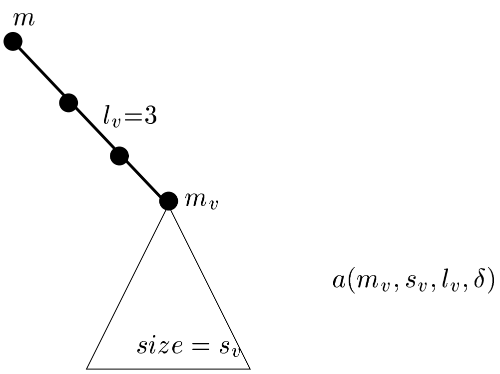

Figure 10.3: The basic setting for the algorithm

We use TREES$(\mathcal{T}, s)$ to denote the class of all binary trees with tests predicates from $\mathcal{T}$ and size at most s. And PATHS$(\mathcal{T}, \ell)$ to denote the class of all conjunctions of at most ℓ predicates from $\mathcal{T}$.

Let us first consider the case in which the class $\mathcal{T}$ of testing functions in the tree nodes is finite, in which case the class of possible path predicates PATHS$(\mathcal{T}, \ell_v)$ leading to v and the class of possible subtrees TREES$(\mathcal{T}, s_v)$ rooted at v are also finite. In this case, we would choose

$$\alpha(m_v, s_v, \ell_v, \delta) = c\sqrt{\frac{\log(|\text{PATHS}(\mathcal{T}, \ell_v)|) + \log(|\text{TREES}(\mathcal{T}, s_v)| + \log(m/\delta)}{m_v}} \qquad (10.2)$$

for some constant specific $c > 1$. Perhaps the most natural and common special case of this finite-cardinality setting is that in which the input space X is the boolean hypercube $\{0, 1\}^n$, and the test class $\mathcal{T}$ contains just the n single-variable tests x_i. Since $|\text{PATHS}(\mathcal{T}, \ell)| \leq n^\ell$ and $|\text{TREES}(\mathcal{T}, s)| \leq (an)^s$ for some constant a, Equation (10.2) specializes to

$$\alpha(m_v, s_v, \ell_v, \delta) = c'\sqrt{\frac{(\ell_v + s_v)\log(n) + \log(m/\delta)}{m_v}} \qquad (10.3)$$

for some specific constant $c' > 1$ determined by the analysis. To simplify the exposition and to make it more concrete, we will work with this particular choice of $\mathcal{T}$ in most of our proofs, but specifically point out how the analysis changes for the case of infinite $\mathcal{T}$, where the pruning rule is given by choosing

$$\alpha(m_v, s_v, \ell_v, \delta) = c'' \sqrt{\frac{(d_{\ell_v} + d_{s_v}) \log(2m) + \log(m/\delta)}{m_v}} \tag{10.4}$$

for some specific constant $c'' > 1$ determined by the analysis, where d_{ℓ_v} and d_{s_v} are the VC dimensions of the classes $\mathrm{PATHS}(\mathcal{T}, \ell_v)$ and $\mathrm{TREES}(\mathcal{T}, s_v)$, respectively.

Let us first provide some brief intuition behind the algorithm: At each node v, the algorithm considers whether to leave the current subtree T_v^* or to delete it. The basis for this comparison must clearly make use of the sample S provided. Furthermore, we could simply prefer whichever of T_v^* and the best leaf makes the smaller number of mistakes on S_v. This is clearly a poor idea, since T_v^* cannot do worse than the best leaf and may do considerably better but generalize poorly compared to the best leaf due to overfitting. Thus, a penalty on T_v^* for its complexity seems reasonable, which is exactly the role of the additive term $\alpha(m_v, s_v, \ell_v, \delta)$ above.

The analysis will proceed as follows. We shall first argue that any time our algorithm chooses *not* to prune T_v^*, then (with high probability) this was in fact the "right" decision, in the sense that the current tree T^* would be degraded by deleting T_v^*. This allows us to establish that our final pruning will be a subtree of the optimal pruning, so our only source of additional error results from those subtrees of this optimal pruning that we deleted. Further analysis allows us to bound this additional error relatively to the error of the optimal pruning by a quantity related to the size of the optimal pruning.

10.6 Local Uniform Convergence

To complete the analysis we need a tool for estimating the relation between the local observed error $\hat{\epsilon}_v$ of a subtree T_v and its local true error ϵ_v. We can use the following results from Local uniform convergence:

Lemma 10.1 *Let C and H be finite classes of boolean functions over X, let f be a target boolean function over x, and let P be a probability distribution over X. For any $c \in C$ and $h \in H$, let $\epsilon_c(h) = \mathbf{Pr}_P[h(x) \neq f(x) | c(x) = 1]$, and for any labeled sample S of $f(x)$, let $\hat{\epsilon}_c(h)$ denote the fraction of points in S_c on which h errs, where $S_c = \{x \in S : c(x) = 1\}$. Then the probability that there exists a $c \in C$ and an $h \in H$ such that*

$$|\epsilon_c(h) - \hat{\epsilon}_c(h)| \geq \sqrt{\frac{\log(|C|) + \log(|H|) + \log(1/\delta)}{m_c}} \tag{10.5}$$

is at most δ, where $m_c = |S_c|$.

*Proof:*Let us fix $c \in C$ and $h \in H$. For these fixed choices, we have for any value λ

$$\mathbf{Pr}_P[|\epsilon_c(h) - \hat{\epsilon}_c(h)| \geq \lambda] = \mathbf{E}_{m_c}[\mathbf{Pr}_{S_c}[|\epsilon_c(h) - \hat{\epsilon}_c(h)| \geq \lambda]]. \tag{10.6}$$

Here the expectation is over the distribution on values of m_c induced by P, and the distribution on S_c is over samples of size m_c (which is fixed inside the expectation) drawn according to P_c (the distribution P conditioned on c being 1). Since m_c is fixed, by standard Chernoff bounds we have

$$\mathbf{Pr}_{S_c}[|\epsilon_c(h) - \hat{\epsilon}_c(h)| \geq \lambda] \leq e^{-\lambda^2 m_c} \tag{10.7}$$

giving the bound

$$\mathbf{Pr}_P[|\epsilon_c(h) - \hat{\epsilon}_c(h)| \geq \lambda] \leq \mathbf{E}_{m_c}[e^{-\lambda^2 m_c}]. \tag{10.8}$$

If we choose

$$\lambda = \sqrt{\frac{\log(|C|) + \log(|H|) + \log(1/\delta)}{m_c}} \tag{10.9}$$

then $e^{-\lambda^2 m_c} = \delta/(|C||H|)$, which is a constant independent of m_c and thus can be moved outside the expectation. By appealing to the union bound ($\mathbf{Pr}[A \vee B] \leq \mathbf{Pr}[A] + \mathbf{Pr}[B]$), the probability that there is *some* c and h such that $|\epsilon_c(h) - \hat{\epsilon}_c(h)| \geq \lambda$ is at most $|C||H|(\delta/(|C||H|)) = \delta$, as desired.

Our use of Lemma 10.1 will be straightforward. Suppose we are considering some node v in a decision tree T at depth ℓ_v, and with a subtree T_v^* of size s_v rooted at v. Then we will appeal to the lemma choosing the conditioning class C to be the class $\textsc{paths}(\mathcal{T}, \ell_v)$, choosing H to be $\textsc{trees}(\mathcal{T}, s_v)$, and choosing δ to be δ'/m^2, where δ' is the overall confidence we desire. In this case, the local complexity penalty $\alpha(\ell_v, s_v, m_v, \delta)$ in Equation (10.2) and the deviation λ in Equation (10.9) coincide, and thus we can assert that with probability $1 - \delta'/m^2$ there is no leaf of depth ℓ_v and subtree of size s_v such that the local observed error of the subtree deviates by more than $\alpha(m_v, s_v, \ell_v, \delta)$ from the local true error. By summing over all m^2 choices for ℓ_v and s_v, we obtain an overall bound of δ' on the failure probability.

In other words, if we limit our attention to the *local* errors ϵ_v (generalization) and $\hat{\epsilon}_v$ (observed), then with high probability we can assert that they will be within an amount (namely, $\alpha(m_v, s_v, \ell_v, \delta)$) that depends only on *local* quantities: the local sample size m_v, the length ℓ_v of the path leading to v, and the size of the subtree rooted at v.

A more complicated argument is needed to prove local uniform convergence for the case of infinite classes.

Lemma 10.2 *Let C and H be classes of boolean functions over X, let f be a target boolean function over x, and let P be a probability distribution over X. For any $c \in C$ and $h \in H$, let $\epsilon_c(h) = \mathbf{Pr}_P[h(x) \neq f(x)|c(x) = 1]$, and for any labeled sample S of $f(x)$, let $\hat{\epsilon}_c(h)$ denote the fraction of points in S_c on which h errs, where $S_c = \{x \in S : c(x) = 1\}$. Then the* *probability that there exists a* $c \in C$ *and an* $h \in H$ *such that*

$$|\epsilon_c(h) - \hat{\epsilon}_c(h)| \geq \sqrt{8\frac{(d_C + d_H)\log(2m) + \log(4/\delta)}{m_c}} \tag{10.10}$$

is at most δ*, where* $m_c = |S_c|$*, and* d_C *and* d_H *are the VC dimensions of* C *and* H*, respectively.*

We shall not discuss the infinite case here. see[1].

10.7 Global Analysis of the Pruning Algorithm

We can apply the tools of local uniform convergence to analyze the pruning algorithm given in Section 10.5. As mentioned earlier, for simplicity in exposition, we will limit our attention to the common case in which X is the boolean hypercube $\{0,1\}^n$ and the class $\mathcal{T}$ of allowed node tests is just the input variables x_i, in which case the pruning rule used by our bottom-up algorithm is that given by Equation (10.3). However, the analysis can easily generalizes to the more general algorithms given by Equations (10.2) and (10.4).

We shall compare the generalization error of the pruning T^* produced by our algorithm from S and T to the generalization error of T_{opt}, the pruning of T that minimizes the generalization error.

We will show that $\epsilon(T^*)$ is larger than $\epsilon_{opt} = \epsilon(T_{opt})$ by an amount that can be bounded by a function of the size s_{opt} and depth ℓ_{opt} of T_{opt}. Thus, if there is a reasonably small subtree of T with small generalization error, our algorithm will produce a pruning with small generalization error as well.

For the analysis, it will be convenient to introduce the notation

$$r_v = (\ell_v + s_v)\log(n) + \log(m/\delta) \tag{10.11}$$

for any node v, where ℓ_v is the depth of v in T, and s_v is the size of T_v^*. In this notation, the penalty $\alpha(m_v, s_v, \ell_v, \delta)$ given by Equation (10.3) is simply a constant (that we ignore in the analysis for ease of exposition) times $\sqrt{r_v/m_v}$. (We assume that $\sqrt{r_v/m_v} \leq 1$, since a penalty which is larger than 1 can be modified to a penalty of 1 without changing the results.)

Lemma 10.3 *With probability at least* $1-\delta$ *over the draw of the input sample* S*,* T^* *is a subtree of* T_{opt}*.*

*Proof:*Consider any node v that is a leaf in T_{opt}. It suffices to argue that our algorithm would choose to prune T_v^*, the subtree that remains at v when our algorithm reaches v. By Equation (10.3), our algorithm would *fail* to prune T_v^* only if $\hat{\epsilon}_v(\emptyset)$ exceeded $\hat{\epsilon}_v(T_v^*)$ by at least the amount $\alpha(m_v, s_v, \ell_v, \delta)$, but then using Lemma 10.1 and its application to the tree we can state that $\epsilon_v(T_v^*) < \epsilon_v(\emptyset)$ with high probability. In other words, if our algorithm fails to prune T_v^*, then T_{opt} would have smaller generalization error by including T_v^* rather than making v a leaf. This contradicts the optimality of T_{opt}.

Lemma 10.3 means that the only source of additional error of T^* compared to T_{opt} is through overpruning, not underpruning. Thus, for the purposes of our analysis, we can imagine that our algorithm is actually run on T_{opt} rather than the original input tree T (that is, the algorithm is initialized starting at the leaves of T_{opt}, since we know that the algorithm will prune everything below this frontier).

Let $V = \{v_1, \ldots, v_t\}$ be the sequence of nodes in T_{opt} at which the algorithm chooses to prune the subtree $T_{v_i}^*$; note that $t \le s_{opt}$. We can now write:

$$\epsilon(T^*) - \epsilon_{opt} = \sum_{i=1}^{t} (\epsilon_{v_i}(\emptyset) - \epsilon_{v_i}(T_{v_i}^*)) p_{v_i} \tag{10.12}$$

where p_{v_i} is the probability under the input distribution P of reaching node v_i, that is, the probability of satisfying the path predicate $reach_{v_i}$. We can bound this difference:

$$\begin{aligned}
\epsilon(T^*) - \epsilon_{opt} &\le \sum_{i=1}^{t} \Big(|\epsilon_{v_i}(\emptyset) - \hat{\epsilon}_{v_i}(\emptyset)| + |\hat{\epsilon}_{v_i}(\emptyset) - \hat{\epsilon}_{v_i}(T_{v_i}^*)| + |\hat{\epsilon}_{v_i}(T_{v_i}^*) - \epsilon_{v_i}(T_{v_i}^*)| \Big) p_{v_i} \\
&\le \sum_{i=1}^{t} \left(\sqrt{\frac{(\ell_{v_i}+1)\log(n) + \log(m/\delta)}{m_{v_i}}} + \alpha(m_{v_i}, s_{v_i}, \ell_{v_i}, \delta) \right. \\
&\qquad \left. + \sqrt{\frac{(\ell_{v_i}+s_{v_i})\log(n) + \log(m/\delta)}{m_{v_i}}} \right) p_{v_i} \\
&\le 4 \sum_{i=1}^{t} \left(\sqrt{\frac{r_{v_i}}{m_{v_i}}} \right) p_{v_i}.
\end{aligned}$$

The first inequality comes from the triangle inequality. The second inequality uses two invocations of Lemma 10.1 in the special case form (equation 10.3) on the first and third arguments, and applies on the second argument the fact that our algorithm directly compares $\hat{\epsilon}_{v_i}(\emptyset)$ and $\hat{\epsilon}_{v_i}(T_{v_i}^*)$, and prunes only when they differ by less than $\alpha(m_{v_i}, s_{v_i}, \ell_{v_i}, \delta)$.

Thus, we would like to bound the sum $\Delta = \sum_{i=1}^{t} (\sqrt{r_{v_i}/m_{v_i}}) p_{v_i}$. We shall use the following lemma to break up the sum.

Lemma 10.4 *The probability, over the sample S, that there exists a node $v_i \in V$ such that $p_{v_i} > 12\log(t/\delta)/m$ but $p_{v_i} \ge 2\hat{p}_{v_i}$, is at most δ.*

*Proof:*We will use the relative Chernoff bound

$$\mathbf{Pr}[\hat{p}_{v_i} < (1-\gamma)p_{v_i}] \le e^{-mp\gamma^2/3} \tag{10.13}$$

which holds for any fixed v_i. By taking $\gamma = 1/2$ and applying the union bound, we obtain

$$\mathbf{Pr}[\exists v_i \in V : p_v \geq 2\hat{p}_v] \leq te^{-p_{v_i} m/12}. \tag{10.14}$$

Now we can use the assumed lower bound on p_{v_i} to bound the probability of the event by δ.

Let V' be the subset of V for which the lower bound $p_{v_i} > 12\log(t/\delta)/m$ holds. We divide the sum that describes Δ into two parts:

$$\Delta = \sum_{v_i \in V - V'} \left(\sqrt{r_{v_i}/m_{v_i}}\right) p_{v_i} + \sum_{v_i \in V'} \left(\sqrt{r_{v_i}/m_{v_i}}\right) p_{v_i} \tag{10.15}$$

The first sum is bounded by $12t\log(t/\delta)/m \leq 12s_{opt}\log(s_{opt}/\delta)/m$, since $\sqrt{r_{v_i}/m_{v_i}}$ is at most 1, and $t \leq s_{opt}$.

For the second sum, we perform a maximization. By Lemma 10.4, with high probability we have that for every $v_i \in V'$, $p_{v_i} < 2\hat{p}_{v_i} = 2m_{v_i}/m$. Thus, with high probability we have

$$\begin{aligned} \sum_{v_i \in V'} \sqrt{\frac{r_{v_i}}{m_{v_i}}} p_{v_i} &< \sum_{v_i \in V'} \sqrt{\frac{r_{v_i}}{m_{v_i}}} \left(2\frac{m_{v_i}}{m}\right) & (10.16) \\ &= \frac{2}{m} \sum_{v_i \in V'} \sqrt{r_{v_i}}\sqrt{m_{v_i}} & (10.17) \\ &\leq \frac{2}{m}\sqrt{(\sum_{v_i \in V'} r_{v_i})(\sum_{v_i \in V'} m_{v_i})} & (10.18) \end{aligned}$$

To bound this last expression, we first bound $\sum_{v_i \in V'} r_{v_i}$. Recall that

$$r_{v_i} = (\ell_{v_i} + s_{v_i})\log(n) + \log(m/\delta). \tag{10.19}$$

Since for any $v_i \in V'$, we have $\ell_{v_i} \leq \ell_{opt}$, we have that $\sum_{v_i \in V'} \ell_{v_i} \leq t\ell_{opt} \leq s_{opt}\ell_{opt}$, since $|V'| \leq t \leq s_{opt}$. Since the subtrees $T^*_{v_i}$ that we prune are disjoint and subsets of the optimal subtree T_{opt}, we have $\sum_{v_i \in V'} s_{v_i} \leq |T_{opt} - T^*| \leq s_{opt}$. Thus

$$\sum_{v_i \in V'} r_i \leq s_{opt}((1 + \ell_{opt})\log(n) + \log(m/\delta)). \tag{10.20}$$

To bound $\sum_{v_i \in V'} m_{v_i}$ in Equation (10.18), we observe that since the sets of examples that reach different nodes at the same depth in the tree are disjoint (so their total in a given depth is at most m and there are at most ℓ_{opt} levels) , we have $\sum_{v_i \in V'} m_{v_i} \leq m\ell_{opt}$.

10.7.1 Putting it all together

Thus, with probability $1 - \delta$, we obtain an overall bound

$$\begin{aligned} \Delta &< 12\log(s_{opt}/\delta)\frac{s_{opt}}{m} + \frac{2}{m}\sqrt{s_{opt}((1 + \ell_{opt})\log(n) + \log(m/\delta))(m\ell_{opt})} & (10.21) \\ &= O\Big(\Big(\log(s_{opt}/\delta) + \ell_{opt}\sqrt{\log(n) + \log(m/\delta)}\Big) \times \sqrt{\frac{s_{opt}}{m}}\Big) & (10.22) \end{aligned}$$

This gives the following important result.

Theorem 10.5 *Let S be a random sample of size m drawn according to an unknown target function and input distribution. Let $T = T(S)$ be any decision tree, and let T^* denote the subtree of T output by our pruning algorithm on inputs S and T. Let ϵ_{opt} denote the smallest generalization error among all subtrees of T, and let s_{opt} and ℓ_{opt} denote the size and depth of the subtree achieving ϵ_{opt}. Then with probability $1-\delta$ over S,*

$$\epsilon(T^*) - \epsilon_{opt} = O\left(\left(\log(s_{opt}/\delta) + \ell_{opt}\sqrt{\log(n) + \log(m/\delta)}\right) \times \sqrt{s_{opt}/m}\right) \quad (10.23)$$

Roughly speaking, Theorem 10.5 ensures that the true error of the pruning found by our algorithm will be larger than that of the best possible pruning by an amount that is not much worse than $\sqrt{s_{opt}/m}$ (ignoring logarithmic and depth factors for simplicity). How good is this? Since we assume that T itself (and therefore, all subtrees of T) may have been constructed from the sample S, standard model selection analysis indicate that ϵ_{opt} may be larger than the error of the best decision tree approximation to the target function by an amount growing like $\sqrt{s_{opt}/m}$. (Recall that ϵ_{opt} is only the error of the optimal *subtree of* T — there may be other trees which are not subtrees of T with error less than ϵ_{opt}, especially if T was constructed by a greedy top-down heuristic.) Thus, if we only compare our error to that of T_{opt}, we are effectively only paying an additional penalty of the same order that T_{opt} pays. If s_{opt} is small compared to m — that is, the optimal subtree of T is small — then this is quite good indeed.

10.8 Bibliographic Notes

In addition to the lecture notes, the following article was used in the preparation of this scribe.

1. *A Fast, Bottom-Up Decision Tree Pruning Algorithm with Near-Optimal Generalization by Michael Kearns and Yishay Mansour.*