Introduction to Machine Learning **Fall Semester, 2013**

Lecture 11: December 29

Lecturer: Yishay Mansour *Scribe: ym*

11.1 Decision Tree Learning Algorithms

11.1.1 What is a decision tree?

A decision tree is a tree whose nodes are labeled with predicates and whose leaves are labeled with the function values. In Figure 11.1 we have Boolean attributes. In this case we can have each predicate to be simply a single attribute. The leaves are labeled with $+1$ and -1. The edges are labeled with the values of the attributes that will generate that transition.

Given an input x, we first evaluate the root of the tree. Given the value of the the predicate in the root, x_1, we continue either to the left sub-tree (if $x_1 = 0$) or the right sub-tree (if $x_1 = 1$). The value of the tree is the value of the leaf we reach in the computation. Any computation defines a path from the root to a leaf.

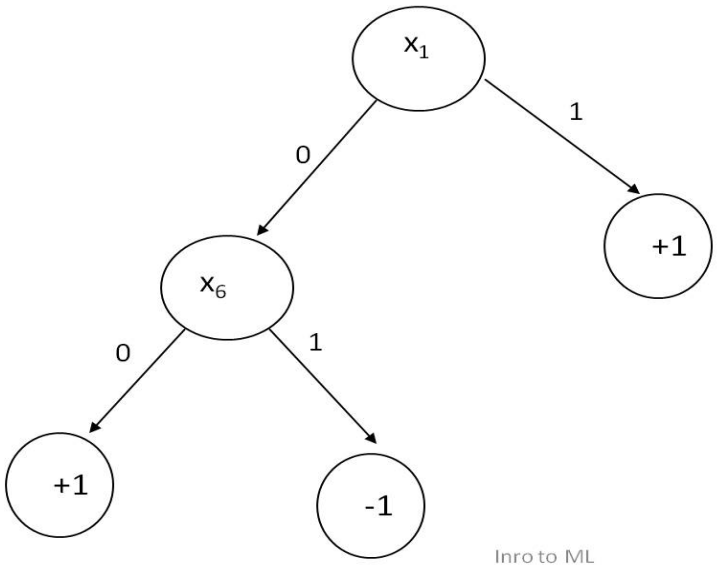

Figure 11.1: Boolean attributes decision tree

When we have continuous value attributes, we need to define some predicate class that will induce binary splits. A common class is *decision stumps* which compare a single attribute

to a fixed value, e.g., $x_1 > 5$. The evaluation, again, starts by evaluating the root, and follows a path until we reach a leaf. (See Figure 11.2)

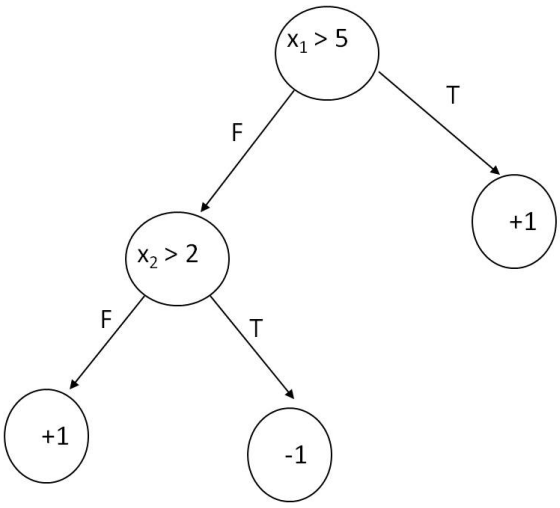

Figure 11.2: A decision tree with decision stumps

When we consider decision stumps, the boundary for decisions are axis parallel, as shown in Figure 11.3.

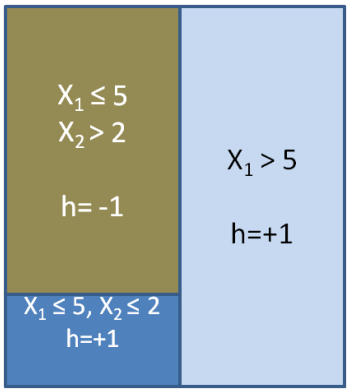

Figure 11.3: Decision boundary for decision stumps.

11.1.2 Decision Trees - Basic setup

The decision tree would have a class of predicates H. This class of predicates can be decision stumps, single attributes (in case of Boolean attributes) or any other predicate class. The

predicate class H can be highly complicate (e.g., hyperplanes) but in most algorithm simple classes are used, since we like to bound the decision trees from simple classifiers. Technically, we can have in each node even a large decision tree.

The input to the decision tree algorithm is the sample $S = \{(x, b)\}$, which as always includes examples of an input and its classification.

The output of the decision tree algorithm is a decision tree where each internal node has a predicate from H and each leaf node has a classification value.

The goal is to output a small decision tree which classifies all (or most) of the examples correctly. This is inline with Occam's Razor, which states that low complexity hypotheses have a better generalization guarantee.

11.1.3 Decision Trees - Why?

One of the main reasons for using decision trees us the human interpretability. Humans find it much easier to understand (small) decision trees. The evaluation process is very simple and one can understand fairly simply what the decision tree will predict.

There is a variety of efficient decision tree algorithms, and there is are many commercial software packages that learn decision trees, such as C4.5 or CART. Even MATLAB has a standard implementation of a decision tree algorithm.

The performance of decision trees is reasonable, probably slightly weaker than SVM and AdaBoost, but comparable on many data sets. Decision forests, which use many small decision trees have a general performance comparable to the best classification algorithms (including SVM and AdaBoost).

11.1.4 Decision trees - Construction

There is a very natural greedy algorithm for constructing a decision tree given a sample. We first decide on a predicate $h_r \in H$ to assign to the root. Once we selected h_r, we split the sample S to two parts. In S_0 (S_1) we have all the examples where $h_r(x) = 0$ ($h_r(x) = 1$). Given S_0 and S_1 we can continue recursively to build a subtree for S_0 and a subtree for S_1.

The time complexity for such a procedure would be

$$Time(m) = O(|H|m) + Time(m_0) + Time(m_1) = O(m^2)$$

where $m = |S|$, $m_0 = |S_0|$ and $m_1 = |S_1|$. In most case the running time is $O(m \log m)$ since most of the splits will be approximately balanced.

The main issue that we need to resolve is how to select the predicate h_r given a sample S. Clearly, if all the samples have the same label we can select a leaf and mark it with that label. Note that we are building a decision tree that would classify all the sample correctly.

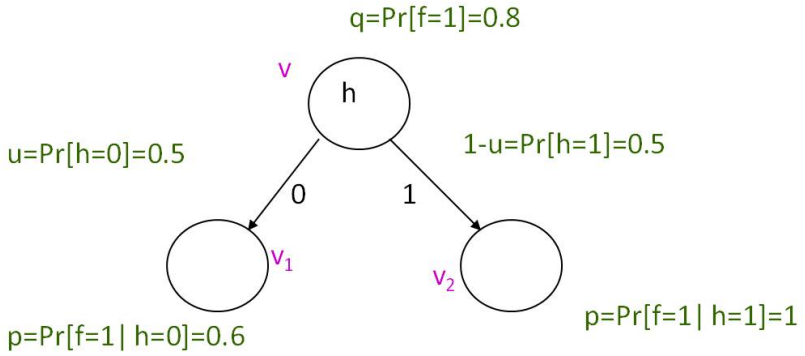

Figure 11.4: An example of a split for the potential equal to the observed error

11.1.5 Selecting predicates - splitting criteria

Given the outline of the greedy algorithm, the main remaining task is to select a predicate, assuming that not all the examples have the same label. We like a simple local criteria that would be base only on the parameters observed at the node.

We would like to use a potential function $val(\cdot)$ to guide our selection. First let us define val for a leaf v with a fraction of 1 equal to q_v as $val(q_v)$. Next for an inner node v which has two leaf sons, we define $val(v) = u \cdot val(p) + (1-u)val(r)$, where u, p and r are define as follows. Let u be the fraction of examples such that $h(x) = 1$, let p be the fraction of examples for which $b = 1$ (the target function is labeled 1) out of the examples with $h(x) = 1$, and let r be the fraction of examples for which $b = 1$ out of the examples with $h(x) = 0$. Finally, for decision tree T we define $val(T) = \sum_{v \in Leaves} p_v val(q_v)$, where p_v is the fraction of samples that reach v.

Given the function val we need to define how we select the predicate $h \in H$ to assign to node v. The idea is to consider the split induced by h, and to compute the value of the potential val if we stop immediately after that split. Namely, we have $val(v,h) = u \cdot val(p) + (1-u)val(r)$, where u, p, and r are define as above. We select the predicate h that minimizes $val(v,h)$. (Note that the potential will never increase, since we can simply select not to split.)

What remains now is to define $val(q)$. Since we would like to minimize the overall error, it seems reasonable to select $val(q) = \min(q, 1-q)$. The reasoning is the following. When we fix v to be a leaf, then the label that would minimize the fraction of error at v is the one that is more likely, and the fraction of errors would be $\min(q, 1-q)$. It is important to understand why this choice is problematic.

Consider the example in Figure 11.4. In this example we have that before the split we have 0.20 fraction of errors and after the split we have $0.5 * 0.4 + 0.5 * 0 = 0.20$ fraction of

errors, so the potential did not decrease. This implies that the potential of the observed error would prefer any other predicate over this split. On the other hand, if we look closely at the split, it looks like an excellent split. We have *half* of the examples perfectly labeled,and we are left with building a decision tree for only half of the sample.

The reason why the observed error potential failed is that it was not able to quantify the progress we make when the observed error does not decrease. In order to overcome this difficulty we would like the following to hold:

1. Every Change in an improvement. We will be able to achieve this by using a strictly convex function.

2. The potential is symmetric around 0.5,namely, $val(q) = val(1-q)$.

3. When zero perfect classification. This implies that $val(0) = val(1) = 0$.

4. We have $val(0.5) = 0.5$.

The important an interesting part is the convexity. The convexity will guarantee us that any split will have a decrease in the potential. The reason is that

$$val(q) > u \cdot val(p) + (1-u)val(r)$$

when $q = up + (1-u)r$, due to the strict convexity.

The other conditions are mainly to maintain normalization. Using them we can ensure that $val(T) \geq error(T)$, since at any leaf v we will have $val(v) > error(v)$.

11.1.6 Splitting criteria

We have reduced the discussion to selecting a splitting criteria which is a strictly convex function. Several potential functions are suggested in the literature (see Figure 11.5) :

1. Gini Index used in CART:
$$G(q) = 2q(1-q)$$

2. Entropy used in C4.5
$$G(q) = \frac{1}{2}[q \cdot \log_2 \frac{1}{q} + (1-q) \cdot \log_2 \frac{1}{1-q}]$$

3. Variance function
$$G(q) = \sqrt{q(1-q)}$$

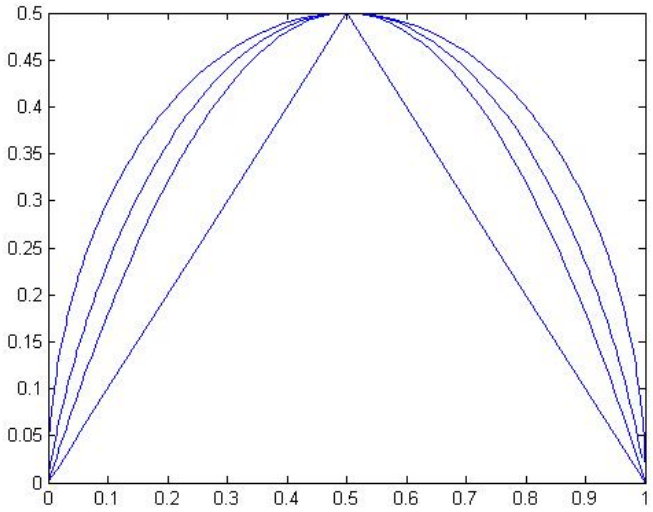

Figure 11.5: Relationships between different splitting criteria. The criteria, from inner to outer are observed error, Gini index, Entropy and Variance

We can now go back to the example of Figure 11.4, and consider the Gini index, i.e., $G = 2q(1-q)$. Before the split we have

$$G(0.8) = 2 \cdot 0.8 \cdot 0.2 = 0.32$$

and after the split we have

$$0.5G(0.6) + 0.5G(1) = 0.5 \cdot 2 \cdot 0.4 \cdot 0.6 = 0.24.$$

In this case, there is a drop in the potential, which is even significant. The drop is due to the fact that $G(\cdot)$ is strictly convex and not linear. Similar drops would be observed for the other two optional cost functions (see figure 11.5).

11.1.7 Decision tree construction - putting it all together

We define a procedure $DT(S)$, where S is the sample. The procedure return a tree that classifies S correctly.

Procedure DT(S) return a DT T
    If $\forall (x, b) \in S$ we have $b = 1$ Then
        Create a Leaf with label 1 and Return.

If $\forall(x,b) \in S$ we have $b = 0$ Then
  Create a Leaf with label 0 and Return.
For each $h \in H$ compute
  $S_h = \{(x,b) \in S | h(x) = 1\}$.
  $u_h = |S_h|/|S|$
  $S_{h,1} = \{(x,b) \in S_h | b = 1\}$
  $p_h = |S_{h,1}|/|S_h|$.
  $S_{h,0} = \{(x,b) \in S - S_h | b = 0\}$
  $r_h = |S_{h,0}|/|S - S_h|$.
  $val(h) = u_h val(p_h) + (1 - u_h) val(r_h)$,
Let $h' = \arg\min_h val(h)$
Call $DT(S_{h'})$ and receive T_1.
Call $DT(S - S_{h'})$ and receive T_0.
Return a decision tree with root labeled by $h\prime$, right subtree T_1 and left subtree T_0.

In class, in the slides we have an example of running the algorithm with the Gini index.

11.1.8 Decision tree algorithms - Guaranteed performance

We do not have any guarantee that the decision tree size produced by the greedy algorithm. In fact, if we consider the target function $x_1 \oplus x_2$ and a uniform distribution over d binary attributes then the greedy algorithm would create a very large decision tree. The reason is that when it considers any single attribute, the probability that the target is 1 or 0 is identical (until we select either x_1 or x_2). This implies that the decision tree will select attributes randomly, until we select either x_1 or x_2.

In fact, we can show that finding the smallest decision tree is NP-hard, which means that it is unlikely we will have a computationally efficient algorithm for computing the smallest decision tree given a sample.

We can perform an analysis that is based on the weak learner hypothesis. If we assume that for any distribution there is a predicate $h \in H$ which is a weak learner, i.e., has error at most $1/2 - \gamma$, then we can bound the decision tree size as a function of the parameter γ. Specifically,

1. For the Gini index,we have that the decision tree size is at most $e^{O(1/\gamma^2 1/\epsilon^2 log^2 1/\epsilon)}$.
2. For the Entropy index,we have that the decision tree size is at most $e^{O(1/\gamma^2 log^2 1/\epsilon)}$.
3. For the Variance index,we have that the decision tree size is at most $e^{O(1/\gamma^2 log 1/\epsilon)}$.