# DECISION TREE LEARNING ALGORITHMS

LECTURER: AMOS FIAT SCRIBE: ALEXANDER F. AVIDAN R.

ABSTRACT. This lecture examines the classification problem and discusses decision trees. The discussion focuses on decision trees building algorithms and the problem of building minimal size decision tree. Some commercial heuristics like C4.5, CART are also mentioned.

## 1. Classification Problem

1.1. **Definition.** The aim of classification problem is to learn a specific criteria of the data from a given sample data set. The data set sample consists of a number of data samples, each sample has a number of attributes that describe the data sample and whose values belong to some unknown distribution $D$. Let's call each attribute $x_i$. The attribute may have either numeric or non-numeric value. For the purpose of this discussion the attributes are supposed to be numeric binary values, this greatly simplifies further approach by allowing the use of binary trees for the algorithms.

One of the attributes is the criteria that the classification problem aims to learn, let's call it $C$ - classification and the binary classification call $T$ - target. Data sample that is formed from a numerous attributes can be called a vector $x_1 ... x_n$. Thus the whole data set sample is a set of vectors $x_1 ... x_n, T$.

The problem is to learn for each vector its target / classification column, so that given a new input vector $x_1 ... x_n$, the algorithm could predict the value of $T$ with some small error.

Sample data from distribution $D$

| $x_1$ | $x_2$ | $x_3$ | $\dots$ | $x_n$ | T |
|---|---|---|---|---|---|
| 0 | 0 | 1 | ... | 0 | 1 |
| 0 | 1 | 1 | ... | 0 | 1 |
| 1 | 0 | 0 | ... | 1 | 0 |

1.2. **Example.** As an example for classification problem, suppose that some bank is interested to know if the new customer is reliable and if it is safe to give him a loan. From past experience the bank has collected a number of parameters characterizing it's customers and their reliability value. Those customers that returned their loans are defined as reliable and those that posed some difficulties are defined as unreliable. The bank is interested to give a loan only to reliable customers, so it is interested to study his customers in order to be able to predict if the new customer is expected to be reliable or not.

Below is a table that describe customers' characteristics and their reliability (target column).

| Age (under 30) | Sex (male) | Married | Employed | Reliable (Target) |
|---|---|---|---|---|
| + | - | + | - | 1 |
| - | - | + | + | 0 |
| + | - | - | + | 1 |
| + | + | - | - | 1 |
| - | - | + | - | 0 |

## 2. Decision Trees

2.1. **Approaches for classification.** A number of approaches are used to solve classification problem. Among them three widely used ones are : decision trees, regression, neural networks. The first of the three is in scope of this discussion. Decision trees are easy to use and visualize in practise, however they are hard to build.

2.2. **Definition.** A **decision tree** over the boolean variables $x_1 ... x_n$ is a binary tree $T$, whose leaves $l$ are decision bits (the target column) and the nodes contain functions h : $\overline{x} \rightarrow \{0, 1\}$ on the input. The path from each node is determined by the value of function on the node, for zero go to the left node, for one go to the right node.

The set of functions $h$ defines a *node function class* $F$ of allowed functions for activation on internal nodes.

2.3. **Discussion of decision trees.** Decision tree is used to predict classification of the input vector by passing the nodes of the tree until the leaves that hold the classification results (decision bits). The path is determined by the value of functions $h$ on the nodes when input vector is applied on them. For example of decision tree see **Figure 1**.

Building of decision tree is not trivial. First, it has to be consistent (with a possible small error) with the data samples from which it is built. Second, it is expected to be as small as possible. Third, the error of prediction should be bounded by some constant.

**Question 1** : Which functions $h \in F$ on the nodes of decision tree are allowed? When input vector contains of boolean variables $x_i \in \{0, 1\}$ then the following functions are allowed : $\{x_1? = 1,\ x_2? = 1\ , ...,\ x_i? = 1\}$
It is allowed to use a linear combination of the above functions from class $F$.

**Question 2** : What is the error of the decision tree?
The error depends on the distribution of the input $D$. The family of functions in

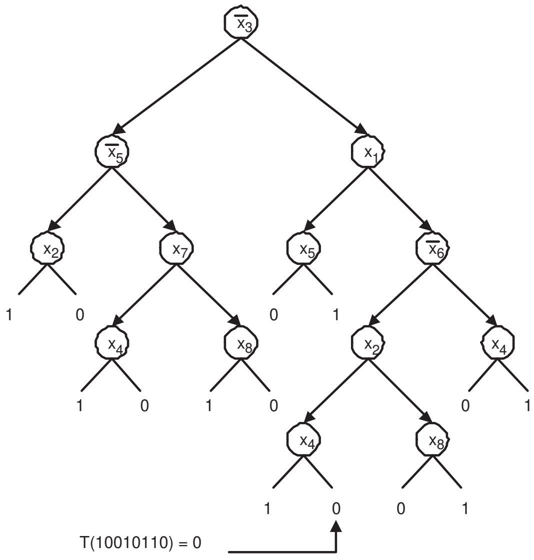


FIGURE 1. A decision tree and the path followed by an input

$F$ - $(x_1? = 1,\ x_2? = 1,\ ...,\ x_n? = 1) \in D$. If $D$ is known then the error is the sum of the probabilities for error on all the leaves $l$. An error on the leaf $l$ is when traversing the tree leads to the leaf with a wrong label.

$$Error \ = \ \sum_{l \in leaves(T)} Prob_D(X \Longrightarrow l \bigcap classification\ is\ wrong)$$

Because the whole distribution is usually unknown there is only an assumption to the above sum.

The target is to build a minimal decision tree with a minimal error, given a class of functions $F$ and data samples. This problem is $NP - hard$ and there is no polynomial time algorithm that builds decision tree of polynomial size from the optimal tree.

**Theorem of Ming Lee** - Given a minimal decision tree $T$ of size $t$, it is impossible to find in a polynomial time a tree of size less than $t^2$.

2.4. **Background to algorithms.** In spite of their exponential complexity, decision trees are quite often used in practise. There are heuristics that used by commercial applications and build decision trees of size expo(t) from the optimal tree. ( C4.5, C5.0, CART )

The popularity of such heuristics is due to their efficiency, simplicity, interpretability to humans and good generalization ability.

These algorithms grow a tree from the root to the leaves by repeatedly replacing an existing leaf with an internal node, thus *splitting* the data reaching this leaf into two new leaves, and reducing the error on the given sample. These algorithms are so called *top down* algorithms.

## 3. Weak learners and boosting

3.1. **Hoonta function.** Suppose there is a binary function, call it *Hoonta*, whose value is determined by only $k$ from $n$ its input parameters $(x_1...x_n)$.

$$f(x_{i1}, x_{i2}, \ ... \ , x_{ik}) \rightarrow 0{,}1 \text{ where } x_{ij} \in 0{,}1$$

The problem is to find which of $n$ parameters are relevant and which are not. The trivial solution is simply try all possible variations and check which one of them is correct. It obviously has the complexity of $\binom{n}{k}$. Recently this result was improved to $n^{3/4k}$, but the complexity still remains exponential.

There is a similarity between the *Hoonta* function and classification problem. The value of the *target* function depends on some $m$ of $n$ boolean variables of the input vector $\vec{x}$. Perhaps $m = n$, but it is not known. Therefore finding those $m$ relevant variables and the relationship between them to the value of the *target* function is at least as hard as to solve the *Hoonta* problem.

Obviously that each one of $m$ boolean variables that determine the value of the target function have some correlation with it. Therefore there is some correlation between functions in $F$ and the value of the *target* function. Even if correlation is very small - just a bit over 1/2, it is still may be assumed that the functions $h \in F$ are *weak learners.*

3.2. **Definition of weak learner.** *Let $f$ be any boolean function over an input space $X$. Let $F$ be any class of boolean functions over $X$. Let $\gamma \in$ (0,1/2]. We say that $f\gamma$ - satisfies the Weak Hypothesis Assumption with respect to $F$ if for any distribution $D$ over $X$, there exist a function $h \in F$ such that $Prob_D[h(x) \neq f(x)] \leq 1/2 - \gamma$. If* D *is a class of distributions, we say that $f$ $\gamma$ - satisfies the Week Hypothesis Assumption with respect to $F$ over* D *if the statement holds for any $D \in$* D. *The $\gamma$ parameter is called advantage.*

3.3. **Introduction to Boosting.** The class of boolean functions $h \in F$ as they are defined above are *weak learners*. Weak learners can be transformed to the model of PAC learning by *boosting* process.

Boosting extends the correlation of weak learners until achieving the PAC learning. The boosting process allows to build a function not in $F$ that has any desired $\epsilon$ error even if original functions $h \in F$ have an error that is only slightly better than random guessing. Article [2] proves that in the language of weak learning, the standard top-down decision tree algorithms are in fact *boosting* algorithms.

3.4. **Example of Boosting.** As an example, let's assume that each function $f_i \in F$ has a mistake of less than $\frac{1}{2} - \epsilon$. Let's assume that $\epsilon = \frac{1}{4}$, then each $f \in F$ has an error of less than $\frac{1}{4}$. The aim is to achieve a function with an error that is less than $\frac{1}{4}$.

Let $h_1 \in F$, $h_2 \in F$, $h_3 \in F$. All the three functions may have an error on the same quarter of the distribution. In order to achieve a function with an error that is less than $\frac{1}{4}$, a new distribution is being built.

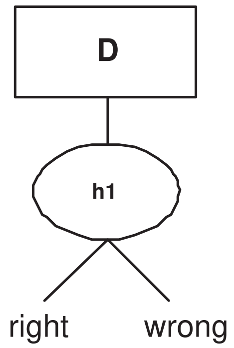


FIGURE 2. Original distribution

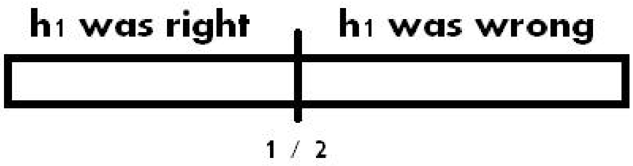


FIGURE 3. Building distribution $D_2$

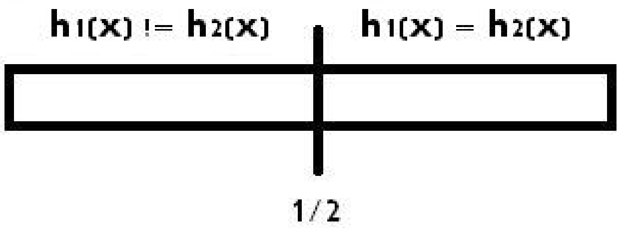


FIGURE 4. Building distribution $D_3$

Let $D$ be the original distribution. The function $h_1$ accepts random elements $x_i$ of distribution $D$. $h_1$ is wrong in less then $\frac{1}{4}$ cases and is right in the rest. See **Figure 2**.

The new distribution $D_2$ is being built. Half of it's elements are the elements that $h_1$ was right about them, and half are the elements that $h_1$ was wrong about them.

Now $h_2$ function accepts the elements from the new distribution $D_2$. $h_2$ will be wrong in less than $\frac{1}{4}$ of the cases. Now there is a new information on the cases that $h_1$ was wrong about them. See **Figure 3**.

Then another distribution $D_3$ is being built. Half of it's elements are the elements that $h_2(x) \neq h_1(x)$, and half are the elements that $h_2(x) = h_1(x)$. Now function $h_3$ accepts elements from distribution $D_3$. See **Figure 4**.

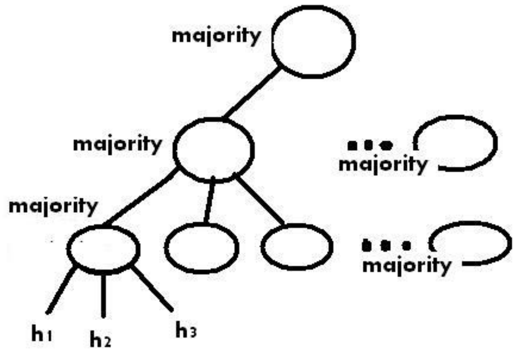


FIGURE 5. Recursive tree for boosting the error

Working with those three weak learner functions, and taking the majority of them, decreases the error. If the error was $\beta$ it will be $\beta^2$ after the process.

Operate the process in a recursive way as seen in **Figure 5**:

The tree depth is $\log \frac{1}{\epsilon}$.
The algorithm called Adaboost uses such a recursive tree.

## 4. Splitting Criterion $G$

As mentioned earlier, top-down decision tree algorithms build the tree by splitting the leaf into new leaves. The process of splitting continues until the required error constant is reached. The algorithms rely on some *splitting criterion* $G$ that determine which leaf should be split next, and which function $h \in F$ to use for the split.

The splitting criterion $G$ is a function $G:$ [0,1] $\rightarrow$ [0,1] having three properties:

- $G$ is symmetric about 1/2. Therefore, $G(x) = G(1-x)$ for any $x \in [0,1]$.
- $G$(1/2)=1 and $G$(0)=$G$(1)=0
- $G$ is concave.

The $G$ criterion function evaluates how good leaf's classification is. The example of $G$ function is shown in **Figure 6**.

Let $T$ be any decision tree whose internal nodes are labeled by functions in $F$, and whose leaves are labeled by values in 0,1. For each $l \in leaves(T)$, $w(l)$ (*weight* of $l$) is the probability that random $x$ reaches leaf $l$ in $T$, and $q(l)$ is the probability that $f(x) = 1$ ($f$ - *target function*) given that $x$ reaches leaf $l$ in $T$. Let's assume that each leaf $l$ in $T$ is labeled 0 if $q(l) \leq 1/2$, and is labeled 1 otherwise.

The previous $error - \epsilon(T)$ definition can be rewritten as:

$$Error \ = \ \sum_{l \in leaves(T)} \ w(l) min(q(l), 1-q(l)).$$

$G$ criterion is defined to be:

$$G(T) \ = \ \sum_{l \in leaves(T)} \ w(l) \ G(q(l)).$$

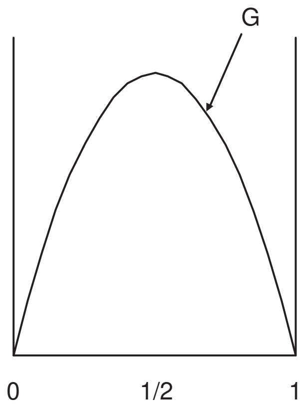


FIGURE 6. Example of $G$ criterion function

Actually $G(q(l))$ is an approximation of $\min(q(l), 1-q(l))$ and therefore $G(T)$ is an approximation of $\epsilon(T)$.

The top-down algorithms make local modifications to the current tree $T$ in order to reduce $G(T)$ and therefore to reduce $\epsilon(T)$ as well.

The choice of choosing $G$ function is upon the algorithm. For example, the popular C4.5 software package chooses $G(q)$ to be the binary entropy function:

$$G(q) = q \log \tfrac{1}{q} + (1-q) \lg \tfrac{1}{1-q} \ .$$

Similarly, the CART program uses the splitting criterion:

$$G(q) = 4q(1-q), \text{ known as the } \textit{Gini} \text{ criterion.}$$

The article [1] finds that there is a better splitting criterion function:

$$G(q) = 2\sqrt{q(1-q)}.$$

## REFERENCES

1. On the Boosting Ability of Top-Down Decision Tree Learning Algorithms / Michael Kearns, Yishay Mansour

2. An Introduction to Computational Learning Theory, Michael J. Kearns Umesh V.Vazirani