

Decision Trees

*Professor: Dan Roth**Scribe: Ben Zhou, C. Cervantes*

Overview

- Decision Tree
- ID3 Algorithm
- Overfitting
- Issues with Decision Trees

1 Decision Trees

1.1 Introduction

In the previously introduced paradigm, feature generation and learning were decoupled. However, we may want to learn directly from the data. In the badges game, for example, it was helpful to consider the notion of vowels, which in previously introduced algorithms could be considered a feature to be introduced during generation. Decision trees, however, can learn this notion from the data itself.

1.2 Representing Data

Assume a large table with N attributes, and we want to know something about the people represented as entries in this table. Further assume that a label for people:

Own an expensive car

Do not own an expensive car

A simple way to represent this table is through a histogram on the attribute "own", later on, we may add another attribute gender, which makes the histogram into 4 entries (own & male, own & female, not own & male, not own & female).

Let's say there are 16 attributes in total. If we only consider one attribute (1-d histograms), there will be 16 entries in the histograms. Similarly, 2-d histogram

has 120 numbers, 3-d has 560 numbers. As number of attributes increases, the problem will be impossible to solve. Thus, we need a way to represent the data more meaningfully such that we can determine which are the most important attributes.

1.3 Definitions

Decision tree is a hierarchical data structure that represents data through a divide and conquer strategy. In this class we discuss decision trees with categorical labels, but non-parametric classification and regression can be performed with decision trees as well.

In classification, the goal is to learn a decision tree that represents the training data such that labels for new examples can be determined.

Decision trees are classifiers for instances represented as feature vectors (e.g. color=?; shape=?; label=?;). Nodes are tests for feature values, leaves specify the label, and at each node there must be one branch for each value of the feature.

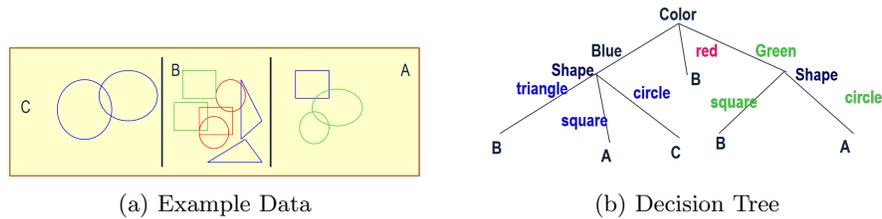


Figure 1: Decision Tree Example

From the example in Figure 1, given a new shape, we can use the decision tree to predict its label.

1.4 Expressivity

As previously discussed, not all Boolean functions can be expressed as linear functions. Decision trees, however, can represent any linear function.

Decision trees can be thought of as a disjunction of conjunctions, or rewritten as rules in Disjunctive Normal Form (DNF).

For example, one could rewrite the decision tree in Figure 1 with only two labels, as in Figure 2.

This decision tree could then be expressed as the following disjunction

$$green \wedge square \vee blue \wedge circle \vee blue \wedge square$$

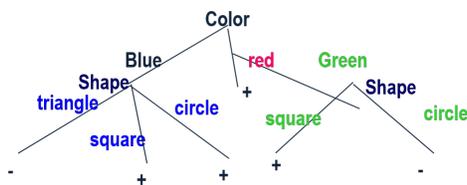


Figure 2: Decision Tree with two labels

Decision trees' expressivity is enough to represent any binary function, but that means in addition to our target function, a decision tree can also fit noise or overfit on training data.

1.5 History

- Hunt and colleagues in Psychology used full search decision tree methods to model human concept learning in the 60s
- Quinlan developed ID3, with the information gain heuristics in the late 70s to learn expert systems from examples
- Breiman, Freidman and colleagues in statistics developed CART (classification and regression trees simultaneously)
- A variety of improvements in the 80s: coping with noise, continuous attributes, missing data, non-axis parallel etc.
- Quinlans updated algorithm, C4.5 (1993) is commonly used (New: C5)
- Boosting (or Bagging) over DTs is a very good general purpose algorithm

2 Id3 Algorithm

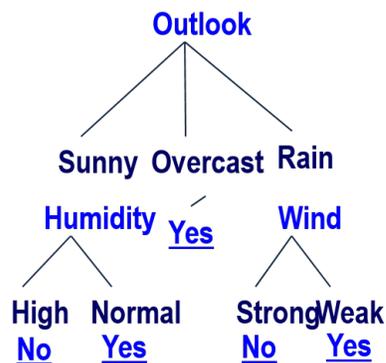
2.1 Tennis Example

Assume we are interested in determining whether to play tennis (+/-) given certain nominal features, below:

- Outlook: S(ununny), O(vercast), R(ainy)
- Temperature: H(ot), M(ild), C(ool)
- Humidity: H(igh), N(ormal), L(ow)
- Wind: S(trong), W(eak)

	O	T	H	W	Play?
1	S	H	H	W	-
2	S	H	H	S	-
3	O	H	H	W	+
4	R	M	H	W	+
5	R	C	N	W	+
6	R	C	N	S	-
7	O	C	N	S	+
8	S	M	H	W	-
9	S	C	N	W	+
10	R	M	N	W	+
11	S	M	N	S	+
12	O	M	H	S	+
13	O	H	N	W	+
14	R	M	H	S	-

(a) Example Data



(b) Decision Tree

Given these features, let's further assume example data, given in Figure 3a.

In learning a decision tree, we must first choose a root attribute and then recursively decide sub-roots, building the decision tree in a top-down fashion. Using the given data, one possible decision tree is shown in Figure 3b.

2.2 Id3 Procedure

Assume a set of examples S , and a set of attributes (A ; attribute $a \in A$; value $v \in a$) and a target label (L) corresponding to the examples in S . The Id3 procedure for building decision trees is given by Algorithm 1

It is important to note that Algorithm 1 adds a leaf node when S_v is empty. This is to provide predictions for future unseen examples that fall into that category.

2.3 Determining the Root Attribute

When building a decision tree, the goal is to produce as small of a decision tree as possible. However, finding a minimal decision tree that is consistent with the

```

Data:  $S, A, L$ 
Result: Decision tree
if All examples' labels =  $L$  then
  | return single node tree with  $L$ 
else
  |  $a \leftarrow$  attribute that best classifies  $S$ 
  | for  $v \in a$  do
  |   | Add new tree branch  $a=v$ 
  | end
  |  $S_v \leftarrow$  subset of examples in  $S$  such that  $a = v$ 
  | if  $S_v$  is empty then
  |   | Add leaf node with most common value of label in  $S$ 
  | else
  |   | Add subtree  $Id3(S_v, A - \{a\}, L)$ 
end

```

Algorithm 1: Id3 procedure

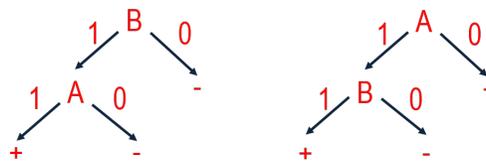
data is NP-hard. We thus want to use heuristics that will produce a small tree, but not necessarily the minimal tree.

Consider the following data with two Boolean attributes (A,B), which attribute should we choose as root?:

- < $(A = 0, B = 0)$, - >: 50 examples
- < $(A = 0, B = 1)$, - >: 50 examples
- < $(A = 1, B = 0)$, - >: 0 examples
- < $(A = 1, B = 1)$, + >: 100 examples

If we split on A, we get purely labeled nodes. If $A=1$, the label is +, - otherwise. If we split on B, we do not get purely labeled nodes.

However if we change the number of $(A = 1, B = 0)$ from 0 to 3, we will no longer have purely labeled nodes after splitting on A. Choosing A or B gives us the following two trees:



(a) Splitting on B

(b) Splitting on A

We want attributes that split the examples to sets into relatively homogenous subsets, with respect to their label, such that we get closer to a leaf node.

2.4 Information Gain

The key heuristic to determine on which attribute to split relies on information gain, and is called Id3.

In order understand information gain, we must first define *entropy*: a measure of disorder in a probability distribution. Entropy for a set of examples, S , can be given as follows, relative to the binary classification task.

$$Entropy(S) = -p_+ \log(p_+) - p_- \log(p_-)$$

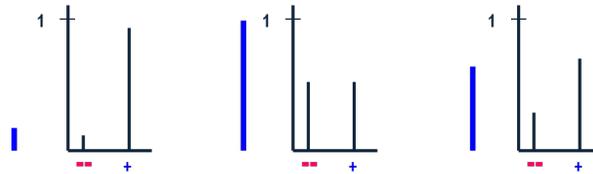
where p_+ is the proportion of positive examples in S and p_- is the proportion of negatives.

If all examples belong to the same label, entropy is 0. If all examples are equally mixed, entropy is 1. In general, entropy represents a level of uncertainty.

In general, when p_i is the fraction of examples labeled i :

$$Entropy(\{p_1, p_2, \dots, p_k\}) = - \sum_{i=1}^k p_i \log(p_i)$$

For example, in a set with two labels, left blue bar show the Entropy and the black bars are proportions of each label.



(a) Entropy Intuition

Given attribute, we can now understand the notion of *information gain*: the expected reduction in entropy caused by partitioning on a given attribute. Information gain is given as follows

$$Gain(S, a) = Entropy(S) - \sum_{v \in values(a)} \frac{|S_v|}{|S|} Entropy(S_v)$$

where S_v is the subset of S for which attribute a has value v , and the entropy of partitioning the data is calculated by weighing the entropy of each partition by its size relative to the original set.

2.5 Tennis Example (Continued)

In the example defined in Section 2.1, the starting entropy is given by $p = \frac{9}{14}, n = \frac{5}{14}$

$$H(Y) = -\frac{9}{14} \log_2\left(\frac{9}{14}\right) - \frac{5}{14} \log_2\left(\frac{5}{14}\right) = 0.94$$

We can now compute information gain for the attributes. For example, the information gain for outlook is given by:

O is sunny, $p = \frac{2}{5}, n = \frac{3}{5}, H_s = 0.971$

O is overcast, $p = \frac{4}{4}, n = 0, H_o = 0$

O is rainy, $p = \frac{3}{5}, n = \frac{2}{5}, H_R = 0.971$

Expected entropy, then, is given by $\frac{5}{14} * 0.971 + \frac{4}{14} * 0 + \frac{5}{14} * 0.971 = 0.694$ and $Gain_{Outlook} = 0.940 - 0.694 = 0.246$.

Similarly, we can calculate the information gain for all other attributes, $Gain_{Humidity} = 0.151, Gain_{Wind} = 0.048, Gain_{Temperature} = 0.029$. Comparing them, we decide to split on Outlook.

After making Outlook the current root, the decision tree looks like

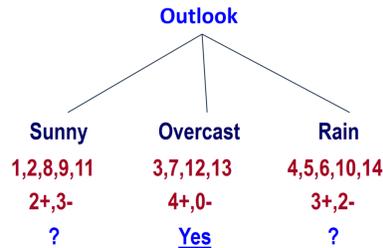


Figure 6: Decision Tree after splitting at Outlook

The final decision tree can then be built by repeating these steps.

2.6 Hypothesis Space

One important thing to remember is that the decisions at the bottom of the tree are less statistically robust than those at the top because the decisions are based on less data.

3 Overfitting

Consider the following decision tree, which has been trained on some data.

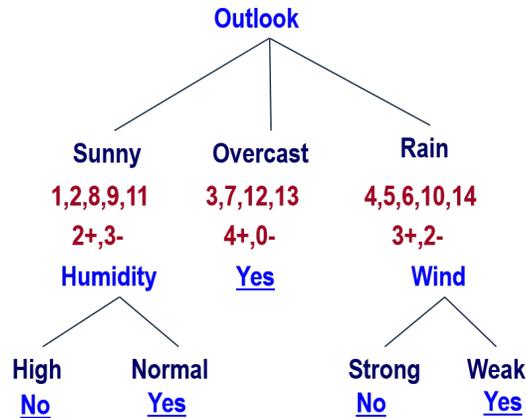


Figure 7: A decision tree that has overfit to its training data

Suppose the existence of a new example

(*Outlook = Sunny, Temp = Hot, Humidity = Normal, Wind = Strong, Label = No*)

The tree in Figure 9a will incorrectly classify this example.

One simple way to correct for this is by adding a sub-tree under (*Humidity = Normal*) that classifies the example to be Yes if the Wind is Weak, No otherwise.

The problem is that we can always correct the tree using this method until the tree agrees on all the examples. Thus the tree may fit noise or other coincidental regularities.

3.1 Definition

Learning a tree that classifies the training data perfectly may not lead to the tree with the best generalization performance because there may be noise in the training data the tree is fitting and the algorithm might be making decisions based on very little data.

A hypothesis h is said to overfit the training data if there is another hypothesis h' , such that h' has a smaller error than h on the training data but h' has larger error on the test data than h .

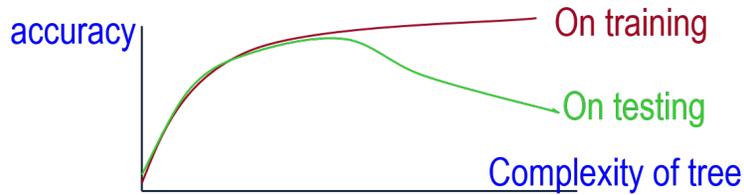


Figure 8: Decision tree accuracy, as tree complexity increases

3.2 Reasons for Overfitting

- Too much variance in the training data such that data is not a representative sample of the instance space and we split on irrelevant features
- Too much noise in the training data: incorrect featured or class labels

In both cases, we want to minimize the empirical error when we learn, we can do so with decision trees.

3.3 Avoiding Overfitting

In order to avoid overfitting with decision trees, we want to *prune* the decision tree: remove leaves and assign the majority label of the parent to all items.

We can do this in two ways.

Pre-pruning refers to when we stop growing the tree at some point during construction when it is determined that there is insufficient data to make reliable decisions.

Post-pruning refers to the process of growing the full decision tree and removing nodes for which we have insufficient evidence

One mechanism for doing so is to prune the children of S if all children are leaves and the accuracy on the validation set does not decrease if we assign the most frequent class label to all items at S .

There are various methods for evaluating which subtrees to prune.

Cross-validation: Reserve hold-out set to evaluate utility

Statistical testing: Test if the observed regularity can be dismissed as likely to occur by chance

Minimum Description Length: Is the additional complexity of the hypothesis smaller than remembering the exceptions?

3.4 The i.i.d. assumption

In order to talk about learning, we need to make some assumptions. Primarily, we assume that there is some distribution that governs the generation of the data.

We assume a distribution $P(X, Y)$ from which the data $D = \{(x, y)\}$ is generated, and we assume that the training and test data are sampled from the same distribution $P(X, Y)$ such that

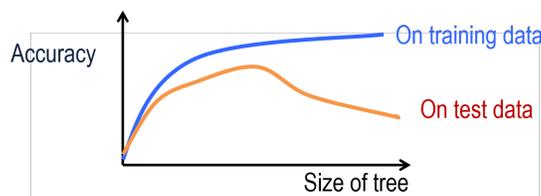
train and test data are identically distributed

each data point (X, Y) is drawn independently from $P(X, Y)$

The notion of training and test items being independently and identically distributed (i.i.d.) is important because the identically distributed assumption ensures our evaluations are valid, and the independence assumption makes it possible to analyze learning algorithms. It is important to note, however, that in many cases the examples are not actually independently drawn.

3.5 Overfitting Accuracy

We talked about a decision tree overfits the training data when its accuracy on the training data goes up but its accuracy on unseen data goes down.



(a) Decision Tree Overfitting

The curves in Figure 9a shows that the training accuracy increases as the size of the tree grows until the tree fits all the training data (assuming no noise).

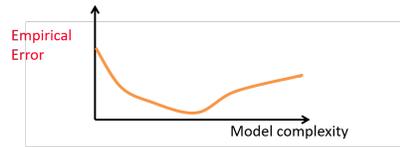
On the test data, however, the accuracy increases only to a point. At the beginning, the tree is not large enough to explain the test data, but as the tree grows it begins to memorize the training data. This means that the tree will only predict positive when an instance is one of the positive examples in the training set. A sufficiently large tree cannot generalize to unseen test data, and thus the test accuracy starts to decrease.

3.6 Bias, Variance, and Error

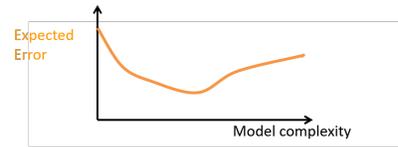
Empirical error on a given data set is the percentage of items in this data set are misclassified by the classifier f .

Informally, **model complexity** is the number of parameters that we have to learn. For decision trees, complexity is the number of nodes.

We refer to **expected error** as the percentage of items drawn from $P(X, Y)$ that we expect to be misclassified by f .



(a) Empirical error for Figure 9a

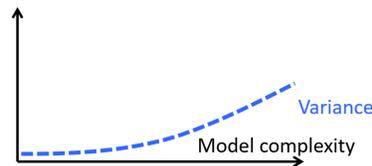


(b) Expected error for Figure 9a

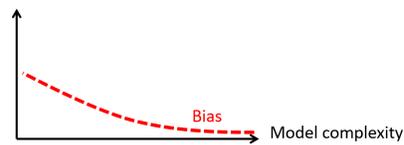
To explain this error, we need to define two notions.

The **variance** of a learner refers to how susceptible the learner is to minor changes in the training data. Variance increases with model complexity.

If, for example, you have a hypothesis space with one function, you can change the data however you want without changing the chosen hypothesis. If you have a large hypothesis space, however, changing the data will result in a different chosen hypothesis.



(a) Learner Variance



(b) Learner Bias

More accurately, assume that for each data set D , you will learn a different hypothesis $h(D)$, that will have a different true error $e(h)$. Here, we are looking at the variance of this random variable.

The second notion we need to understand is the **bias** of the learner, or the likelihood of the learner identifying the target hypothesis. If you have a more expressive model, it is more likely to find the right hypothesis. The larger the hypothesis space, the easier it is to find the true hypothesis; more expressive models have low bias.

More accurately, for data set D , hypothesis $h(D)$, and true error $e(h)$, the bias refers to the difference of the mean of this random variable from the true error. This difference goes down as the size of the hypothesis space goes up.

Expected error is related to the sum of the bias and the variance.

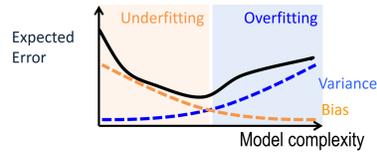


Figure 12: Impact of Bias and Variance on Expected Error

Since bias is a measure of closeness with the true hypothesis and variance refers to the variance of the models, simple models – high bias, low variance – tend to underfit the data, while complex models – low bias, high variance – tend to overfit. The point where these curves intersect is where balance can be found between these two extremes, as shown in Figure 12.

Everything expressed so far can be made more accurate if we fix a loss function. We will also develop a more precise and general theory that trades expressivity of models with empirical error.

3.7 Pruning Decision Trees as Rules

Decision trees can be pruned as they are. However, it is also possible to perform pruning at the rules level, by taking a tree, converting it to a set of rules, and generalizing some of those rules by removing some condition¹.

Pruning at the rule level, however, enables us to prune any condition in the rule without considering the position in the tree. When pruning the tree as-is, the operation must be bottom-up (only leaves are pruned), but in the rule setting, any condition can be chosen and eliminated.

4 Issues with Decision Trees

4.1 Continuous Attributes

Up until this point, decision trees have been discussed as having discrete-valued attribute (outlook = {sunny, overcast, rain}). Decision trees can handle continuous attributes, and one way to do so is to discretize the values into ranges (ex. {big, medium, small}).

Alternatively, we can split nodes based on thresholds ($A < c$) such that the data is partitioned into examples that satisfy $A < c$ and $A \geq c$. Information gain for

¹Many commercial packages can do this

these splits can be calculated in the same way as before, since in this setup each node becomes a Boolean value.

To find the split with the highest gain for continuous attribute A, we would sort examples according to the value, and for each ordered pair (x, y) with different labels, we would check the midpoint as a possible threshold.

For example, consider the following data

Length (L): 10 15 21 28 32 40 50
 Class : - + + - + + -

Using the above, we would check thresholds $L < 12.5$, $L < 24.5$, $L < 45$, and could then use the standard decision tree algorithms. It is important to note, though, that domain knowledge may influence how we choose these splits; the midpoint may not always be appropriate.

4.2 Missing Values

In many situations, you do not have complete examples. It could be that some of the features are too expensive to measure, as in the medical domain.

In training process, we have to find a way to evaluate $Gain(S, a)$ where in some of the examples a value for a is not given.

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
8	Sunny	Mild	???	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes

Figure 13: Tennis Example with Missing Values

One way to handle missing values is to assign the most likely value of x_i to s . In the tennis example shown in Figure 13, the value for *Humidity* is missing. To compute $Gain(S_{sunny}, Humidity)$, then, we need to compute $argmax_k P(x_i = k)$ is *Normal*. That gives us $Gain = 0.97 - (3/5) * Ent[+0, -3] - (2/5) * Ent[+2, -0] = 0.97$

It is also possible to assign fractional counts $P(x_i = k)$ for each value of x_i to s . In order to compute $Gain(S_{sunny}, Humidity)$, we can say the label for the missing value is half *High* and half *Normal*. That is $0.97 - (2.5/5) * Ent[+0, -2.5] - (2.5/5) * Ent[+2, -5] < 0.97$

We could also identify each unknown value as a unique *unknown* label.

One possibility is to predict the missing value based on what we know about the other examples. Later in the course we'll discuss how to do this during learning, in effect predicting missing feature values with distributions while we're learning labels.

These different methods for handling missing values may be appropriate in different training contexts. We also want to be able to handle the same issue during testing.

For example, given {Outlook=Sunny, Temp=Hot, Humidity=???, Wind=Strong}, we need to predict the label using our decision tree. One simple way is to think in probabilistic context. Say if we have three possible values on *Humidity*, we can compute the label for each of the potential values and assign the most likely label

4.3 Other Issues with Decision Trees

Sometimes attributes have different costs. To account for this, information gain can be changed so that low-cost attributes are preferred.

When different attributes have different numbers of values, information gain tends to prefer attributes with more values.

Oblique Decision Trees – that is, where decisions are not axis-parallel – can learn linear combinations of features (ex. $2a + 3b$). Certain decision tree algorithms can handle this.

Incremental Decision Trees – that is, those that are learned and then updated incrementally – can be hard because an incoming example may not be consistent with the existing tree. Here, again, certain algorithms can handle this

5 Decision Trees as Features

One very important application of decision trees is to use them as features. Rather than using decision trees to represent one target function, we can use multiple small trees as features.

Learning one big decision tree often results in overfitting. Instead, we want to learn small decision trees with limited depth, using different subsets of the training set. Another option is to choose a subset of n features and learn a decision tree over that feature subset.

We can then treat these small decision trees as "experts"; they are correct only on a small region in the domain. Each of these experts knows how to take an example and return a $\{0, 1\}$ label. If, for example, we had 100 small decision trees, we would have 100 Boolean judgments on a given example which we could

then treat as feature vector. We could then pass this feature vector to another learning algorithm, and learn over these experts.

One way to combine these decision trees is with an algorithm called *boosting*, which we'll discuss later in the semester.

6 Experimental Machine Learning

Machine Learning is an experimental field, and one of the basics is in splitting the data into two or three sets: training data (70% to 90%), test data (10% to 20%) and development Data(10% to 20%).

Though test data is used to evaluate the final performance of your algorithm, it should never be viewed during the training process. Instead, intermediate evaluations – when trying to tune parameters, for example – should be done on a development set.

6.1 N-fold Cross Validation

Instead of splitting the data into one training set and one testing set, we can split data into N equal-sized parts as shown in Figure 14, where red sections represent test data.



Figure 14: N-fold Cross Validation data

In N-fold Cross Validation, we train and test N different classifiers and report the average accuracy.