ID3 Algorithm for Decision Trees

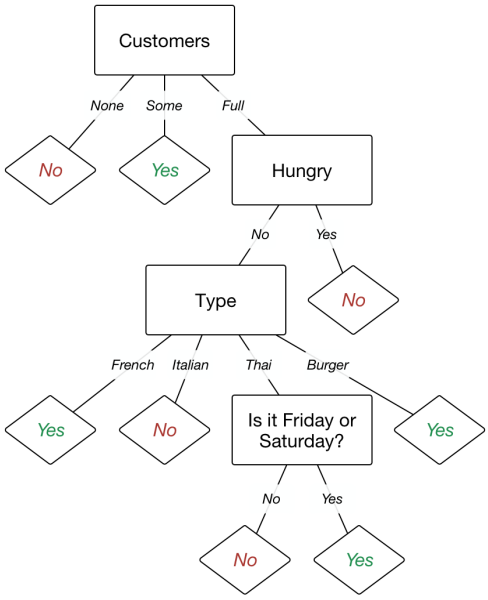
The purpose of this document is to introduce the ID3 algorithm for creating decision trees with an in-depth example, go over the formulas required for the algorithm (entropy and information gain), and discuss ways to extend it.

Overview and Motivation:
Introduction
The ID3 algorithm
- Summary:
- Pseudocode:
- Detail:

Extending the ID3 algorithm
- Continuous attributes:
- Missing attribute values:
- Avoiding overfitting the data:
- Changing the information gain formula:

Overview and Motivation:

Decision tree learning algorithms generate decision trees from training data to approximate solutions to classification or regression problems. In the case of classification, trees are typically represented by a set of if-then rules culminating in a decision:



A decision tree about restaurants[1]

To make this tree, a decision tree learning algorithm would take training data containing various permutations of these four variables and their classifications (*yes*, eat there or no, don't eat there) and try to produce a tree that is consistent with that data.

This document will cover classification trees by introducing one of the classic small data sets in the literature: the *PlayTennis* data set. Then, a popular algorithm used to take training data and produce a decision tree, the ID3 algorithm, will be discussed in detail. Finally, we will discuss potential pitfalls when using the data on real data sets and explain workarounds and solutions to them.

Introduction

Let's pretend for a moment that you like to play tennis. On a particular day—say, a random Sunday morning—how would you decide whether or not you would head to the nearest tennis court for a few sets? Perhaps you would look outside and check to see if it's cloudy or raining. Maybe you'd even step outside to see how hot (or cold) it is. Then, you'd use all of this information to inform your decision. If you took that even further, you could record the choices you made on

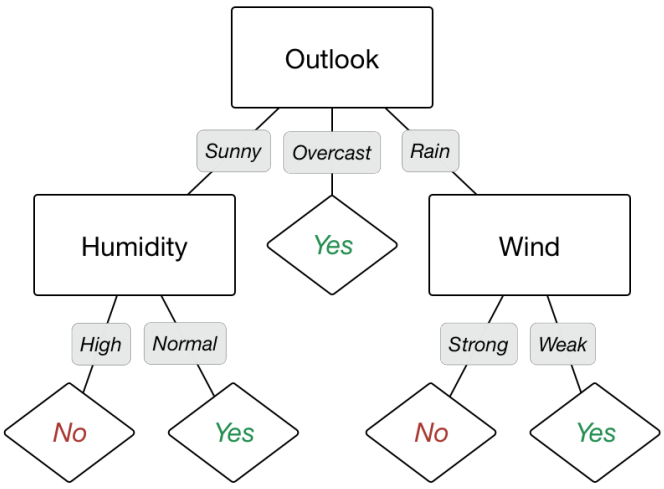
[1] Image adapted from: Stuart J. Russell and Peter Norvig. 2003. Artificial Intelligence: A Modern Approach (2 ed.). Pearson Education. Chapter 18.3.

different days (and all of the variables you took into account to make them) into a table like the one below[2]:

Day	Outlook	Temp.	Humidity	Wind	Play?
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

With this table, other people would be able to use your intuition to decide whether they should play tennis by looking up what you did given a certain weather pattern, but after just 14 days, it's a little unwieldy to match your current weather situation with one of the rows in the table. A decision tree would be a great way to represent data like this, because the data takes certain weather patterns and classifies them into a few buckets: tennis-playing weather and not tennis-playing weather. A decision tree for this data allows you to make a decision by following a graph, rather than by looking up your particular situation in a table:

[2] Data set found in: Tom Mitchell. 1997. Machine Learning. McGraw-Hill. Chapter 3.



In this case, you're asked a number of questions about your current weather situation that will result in a yes (let's play tennis!) or no (let's stay indoors) answer. So, how did this tree result from the training data? Let's take a look at the ID3 algorithm.

The ID3 algorithm

Summary:
The ID3 algorithm builds decision trees using a top-down, greedy approach. Briefly, the steps to the algorithm are:

1. Start with a training data set, which we'll call *S*. It should have attributes and classifications. The attributes of PlayTennis are outlook, temperature humidity, and wind, and the classification is whether or not to play tennis. There are 14 observations.
2. Determine the best attribute in the data set *S*. The first attribute ID3 picks in our example is outlook. We'll go over the definition of "best attribute" shortly.
3. Split *S* into subsets that correspond to the possible values of the best attribute. Under outlook, the possible values are sunny, overcast, and rain, so the data is split into three subsets (rows 1, 2, 8, 9, and 11 for sunny; rows 3, 7, 12, and 13 for overcast; and rows 4, 5, 6, 10, and 14 for rain).
4. Make a decision tree node that contains the best attribute. The outlook attribute takes its rightful place at the root of the PlayTennis decision tree.
5. Recursively make new decision tree nodes with the subsets of data created in step #3. Attributes can't be reused. If a subset of data agrees on the classification, choose that

classification. If there are no more attributes to split on, choose the most popular classification. The sunny data is split further on humidity because ID3 decides that within the set of sunny rows (1, 2, 8, 9, and 11), humidity is the best attribute. The two paths result in consistent classifications—sunny/high humidity always leads to no and sunny/normal humidity always leads to yes—so the tree ends after that. The rain data behaves in a similar manner, except with the wind attribute instead of the humidity attribute. On the other hand, the overcast data always leads to yes without the help of an additional attribute, so the tree ends immediately.

Pseudocode:

This pseudocode assumes that the attributes are discrete and that the classifications are either yes or no. It deals with inconsistent training data by choosing the most popular classification label whenever a possible conflict arises.

```
def id3(examples, classification_attribute, attributes):
    create a root node for the tree
    if all examples are positive/yes:
        return root node with positive/yes label
    else if all examples are negative/no:
        return root node with negative/no label
    else if there are no attributes left:
        return root node with most popular
classification_attribute label
    else:
        best_attribute = attribute from attributes that best
                         classifies examples
        assign best_attribute to root node
        for each value in best_attribute:
            add branch below root node for the value
            branch_examples = [examples that have that value
                               for best_attribute]
            if branch_examples is empty:
                add leaf node with most popular
                    classification_attribute label
            else:
```

```
add subtree id3(branch_examples,
                classification_attribute,
                attributes - best_attribute)
```

If there's an attribute for the data to be split on, the algorithm calls itself recursively, with the original set of examples being split into groups based on the value of the best attribute and the set of available attributes to split on having the best attribute removed from it. Because this algorithm is a recursive one, the base cases: all examples having the same classification, no attributes being left, or no examples remaining, are tested first.

Detail:
Now that we have a high-level picture of the ID3 algorithm, let's fill in some of the gaps!

First, the ID3 algorithm answers the question, "are we done yet?" Being done, in the sense of the ID3 algorithm, means one of two things:

1. All of the data points to the same classification. This allows ID3 to make a final decision, since all of the training data will agree with it.
2. There are no more attributes available to divide the data. ID3 only uses each attribute a maximum of one time per path through the tree[3]. Once it's reached that maximum, if the remaining data doesn't point to the same classification, the algorithm is forced to make a final decision, which usually ends up being the most popular classification, because it can't split up the data any longer.

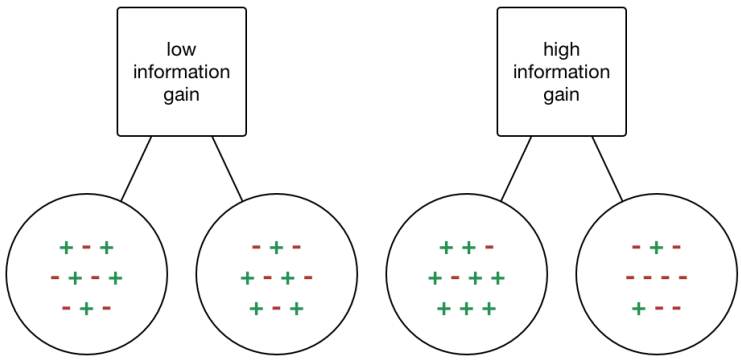
As you may have guessed, neither of these situations should apply at the very beginning. It would not be particularly useful to have an entire training set with the same classification. Imagine that our *PlayTennis* data set always told us to play tennis, regardless of the weather situation. Because ID3 would recognize that all of the data points to the same classification and, therefore, it could arrive at a final decision, the tree that would result would look like:

[3] This restriction might not apply to continuous attributes.

Similarly, it wouldn't be useful to have an entire training set that has no attributes. If our algorithm could only use classifications and, consequently, had no information except for the ratio of *yes* rows to *no* rows, it would be forced to pick the one that's most popular. There are 9 *yes* rows and 5 *no* rows, so the entire decision tree is, again:

Luckily, in this case, the training data is split into 9 *yes* rows and 5 *no* rows and there are four potential attributes we can use to split the data. The algorithm, sensing that it's not done yet, asks the following question: "which attribute should we divide the data with?" The answer is that it should divide the data by the best attribute, but what does "best" actually mean?

For ID3, we think of best in terms of which attribute has the most *information gain*, a measure that expresses how well an attribute splits the data into groups based on classification.



An attribute like the one on the left that splits the data into groups with relatively even distributions of positive and negative examples doesn't bring us any closer to a decision, whereas an attribute like the one on

the right that splits the data into groups with lopsided numbers of positive or negative examples does. The latter group scores better in terms of information gain.

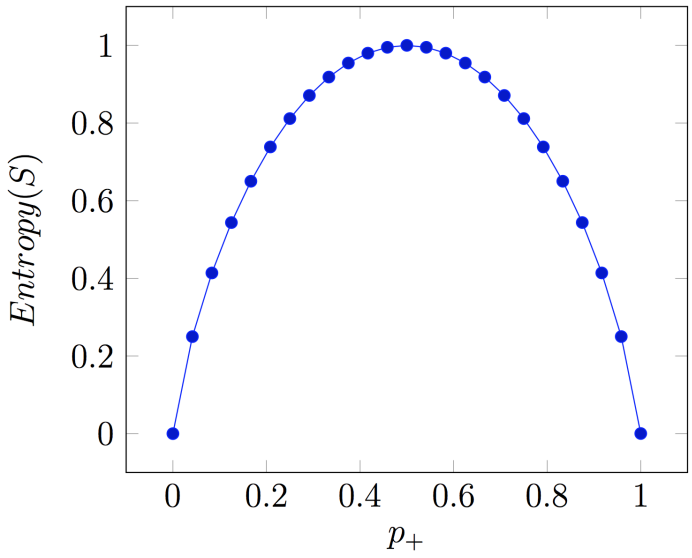
To actually calculate information gain, we must first take a look at another measure, *entropy*. Entropy, in an information theory and machine learning sense, measures the homogeneity of a data set *S*'s classifications. It ranges from 0, which means that all of the classifications in the data set are the same, to $\log_2$ of the number of different classifications, which means that the classifications are equally distributed within the data set. In our *PlayTennis* example, which has 2 different classifications (*yes* and *no*), the maximum entropy of the training data is $\log_2(2) = 1$. If all of the training data tells us yes, the entropy is 0. If all of it tells us no, the entropy is still 0. If there are equal numbers of yes and no examples, the entropy is 1. Since there are 9 yes examples and 5 no examples in our table, its entropy lies somewhere between 0 and 1. We'll have to calculate it using the formula for entropy, which is:

$$Entropy(S) = \sum_{i=1}^{c} -p_i \log_2 p_i$$

In this formula, *c* corresponds to the number of different classifications and p_i corresponds to the proportion of the data with the classification *i*. Because our example and the basic version of the ID3 algorithm both deal with the case where classifications are either positive or negative, we can simplify the formula to:

$$Entropy(S) = -p_+ \log_2 p_+ - p_- \log_2 p_-$$

Here, p_+ is the proportion of examples with a positive classification and p_- is the proportion of examples with a negative classification. A plot of p_+ against *Entropy(S)* demonstrates how entropy decreases to 0 as the proportion of negative or positive examples reaches 100% and peaks at 1 as the examples become more heterogeneous.



Probability vs. entropy for a binary classifier[4]

In the case of *PlayTennis*, there are 9 *yes* rows and 5 *no* rows, which leads to an entropy of:

$$Entropy([9+,\ 5-]) = -\frac{9}{14}\log_2\frac{9}{14} - \frac{5}{14}\log_2\frac{5}{14} = .940$$

Information gain measures the reduction in entropy that results from partitioning the data on an attribute *A*, which is another way of saying that it represents how effective an attribute is at classifying the data. Given a set of training data *S* and an attribute *A*, the formula for information gain is:

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

The entropies of the partitions, when summed and weighted, can be compared to the entropy of the entire data set. The first term corresponds to the entropy of the data before the partitioning, whereas the second term corresponds to the entropy afterwards. We want to maximize information gain, so we want the entropies of the partitioned data to be as low as possible, which explains why attributes that exhibit high information gain split training data into

[4] Adapted from: Tom Mitchell. 1997. Machine Learning. McGraw-Hill. Chapter 3.

relatively heterogeneous groups. How do each of the four attributes in PlayTennis fare?

$$
\begin{aligned}
Values(Outlook) &= Sunny, Overcast, Rain \\
S &= [9+, 5-] \\
S_{Sunny} &= [2+, 3-] \\
S_{Overcast} &= [4+, 0-] \\
S_{Rain} &= [3+, 2-] \\
Gain(S, Outlook) &= Entropy(S) \\
&\quad - (5/14)Entropy(S_{Sunny}) \\
&\quad - (4/14)Entropy(S_{Overcast}) \\
&\quad - (5/14)Entropy(S_{Rain}) \\
&= .940 - (5/14).971 - (4/14)0 - (5/14).971 \\
&= .247
\end{aligned}
$$

$$
\begin{aligned}
Gain(S, Temperature) &= Entropy(S) \\
&\quad - (4/14)Entropy(S_{Hot}) \\
&\quad - (6/14)Entropy(S_{Mild}) \\
&\quad - (4/14)Entropy(S_{Cool}) \\
&= .940 - (4/14)1 - (6/14).811 - (4/14).971 \\
&= .029
\end{aligned}
$$

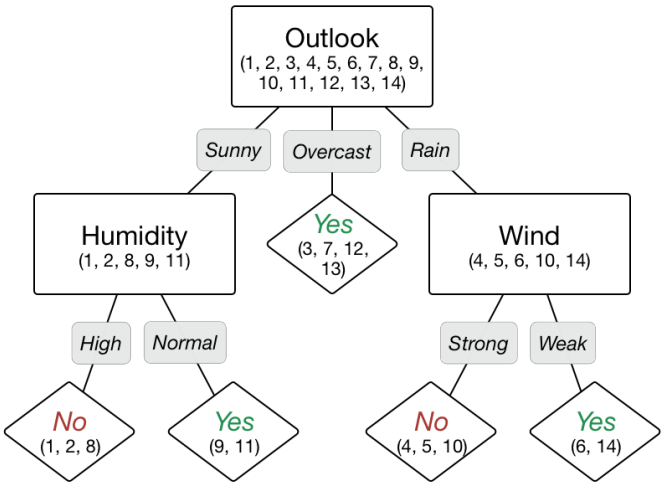
$$
\begin{aligned}
Gain(S, Humidity) &= Entropy(S) \\
&\quad - (7/14)Entropy(S_{High}) \\
&\quad - (7/14)Entropy(S_{Normal}) \\
&= .940 - (7/14).985 - (7/14).592 \\
&= .152
\end{aligned}
$$

$$
\begin{aligned}
Gain(S, Wind) &= Entropy(S) \\
&\quad - (8/14)Entropy(S_{Weak}) \\
&\quad - (6/14)Entropy(S_{Strong}) \\
&= .940 - (8/14).811 - (6/14)1 \\
&= .048
\end{aligned}
$$

The Outlook attribute wins pretty handily, so it's placed at the root of the decision tree.

Afterwards, the decision tree is expanded to cover Outlook's possible values. In *PlayTennis*, the outlook can be sunny, overcast, or rain, so

all of the examples that have a sunny outlook are funneled through the sunny path, the examples with an overcast outlook are diverted to the overcast path, and so on. The goal is to split the data at every step in such a way that consensus on a particular classification happens quickly. In this case, it only takes 2 levels of attribute nodes to decide what to do.



ID3 decides the best root attribute based on our entire data set (all 14 rows), but only uses the sunny outlook data (rows 1, 2, 8, 9, and 11) to decide the humidity node, the rain outlook data (rows 4, 5, 6, 10, and 14) to decide the wind node, and so on. Calculating and comparing the information gain values for the rest of the tree is left as an exercise for the reader.

The greedy part of the approach comes from the fact that it will decide which attribute should be at the root of the tree by looking just one move ahead. It compares all available attributes to find which one classifies the data the best, but it doesn't look ahead (or behind) at other attributes to see which combinations of them classify the data the best. This means that while the algorithm will, in most cases, come up with a good decision tree, a better one may exist. *PlayTennis* is a small enough and contrived enough example that the ID3 algorithm returns an optimal decision tree, but as the size of the training data and the number of attributes increase, it becomes likelier that running ID3 on it will return a suboptimal decision tree.

Extending the ID3 algorithm

While the *PlayTennis* example demonstrates that the ID3 algorithm works well with flawless training data—data with discrete attributes, no missing values, and no classification inconsistencies—the algorithm, with some help, is robust enough to handle much tougher situations. Small tweaks allow ID3 to handle continuous attributes, missing attribute values, data that doesn't work particularly well with information gain, and more.

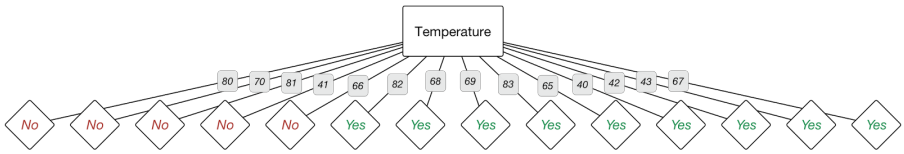
Continuous attributes:
PlayTennis' original temperature attribute has three possible values: cool, mild, and hot. If we had instead recorded the numerical temperature and tried to use the ID3 algorithm as-is on that data, it isn't difficult to imagine our new continuous temperature attribute providing the most information gain while giving us a decision tree that does not generalize particularly well.

Let's give our data set some temperature values besides Hot, Mild, and Cool:

Day	Outlook	Temp.	Humidity	Wind	Play?
1	Sunny	80	High	Weak	No
2	Sunny	81	High	Strong	No
3	Overcast	82	High	Weak	Yes
4	Rain	65	High	Weak	Yes
5	Rain	40	Normal	Weak	Yes
6	Rain	41	Normal	Strong	No
7	Overcast	42	Normal	Strong	Yes
8	Sunny	66	High	Weak	No
9	Sunny	43	Normal	Weak	Yes
10	Rain	67	Normal	Weak	Yes
11	Sunny	68	Normal	Strong	Yes
12	Overcast	69	High	Strong	Yes
13	Overcast	83	Normal	Weak	Yes
14	Rain	70	High	Strong	No

Information gain for the temperature attribute would be .940, the full entropy of *S*. This is because there is only one observation per temperature value, which means that for each value of the temperature attribute, the training data agrees unanimously on a

classification and the entropy for each temperature value is 0. The resulting tree would look like:



This tree is pretty short and it's consistent with the training data, but it's not particularly illuminating or helpful in real-life tennis-playing-deciding situations. One way to make the ID3 algorithm more useful with continuous variables is to turn them, in a way, into discrete variables. Instead of testing the information gain of the actual temperature values, we could test the information gain of certain partitions of the temperature values, such as temperature > 41.5. Typically, whenever the classification changes from no to yes or yes to no, the average of the two temperatures is taken as a potential partition boundary. Because 41 corresponds to no and 42 corresponds to yes, 41.5 becomes a candidate. If any of the partitions end up exhibiting the greatest information gain, then it is used as an attribute and temperature is removed from the set of potential attributes to split on.

Missing attribute values:

Because there are only 14 rows in the *PlayTennis* data set, it's not unrealistic to expect all our training data to have values for each of the attributes. However, in reality, data often comes from sources of varying quality, which makes the prospect of having to deal with missing values quite likely. If our data were missing certain values, what could we do?

Day	Outlook	Temp.	Humidity	Wind	Play?
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	**?**	No
3	Overcast	Hot	**?**	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	**?**	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No

9	Sunny	Cool	Normal	Weak	Yes
10	**?**	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

Two simple ways of dealing with it involve choosing the most popular value in the training data. The first method does not take the classification into account, which leads to an outlook of sunny (5 examples), a temperature of mild (6 examples), a humidity of normal (7 examples), and a wind of weak (8 examples). The second method chooses the most popular value for the same classification, which leads to an outlook of overcast (4 examples), a temperature of mild (4 examples), a humidity of normal (6 examples), and a coin-flip for wind (2 examples each).

A more complicated way to deal with it involves assigning probabilities of each value of the attribute to the missing entries. As a result, day 10's outlook would be 5/13 sunny, 4/13 overcast, and 4/13 rain. When calculating information gain, these fractional examples are used. Here's what it looks like:

$$
\begin{aligned}
Values(Outlook) &= Sunny, Overcast, Rain \\
S &= [9+, 5-] \\
S_{Sunny} &= [2\frac{5}{13}+, 3-] \\
S_{Overcast} &= [4\frac{4}{13}+, 0-] \\
S_{Rain} &= [2\frac{4}{13}+, 2-] \\
Gain(S, Outlook) &= Entropy(S) \\
&\quad - \frac{5\frac{5}{13}}{14} Entropy(S_{Sunny}) \\
&\quad - \frac{4\frac{4}{13}}{14} Entropy(S_{Overcast}) \\
&\quad - \frac{4\frac{4}{13}}{14} Entropy(S_{Rain}) \\
&= .940 - \frac{5\frac{5}{13}}{14}(.991) - \frac{4\frac{4}{13}}{14}(0) - \frac{4\frac{4}{13}}{14}(.996) \\
&= .252
\end{aligned}
$$

Avoiding overfitting the data:
Because the ID3 algorithm continues splitting on attributes until either it classifies the data perfectly or there are no more attributes to split on, it's prone to creating decision trees that overfit by performing really well on the training data at the expense of accuracy with respect to the entire distribution of data.

There are two popular approaches to avoid this in decision trees: stop growing the tree before it becomes too large or prune the tree after it becomes too large. Typically, a limit to a decision tree's growth will be specified in terms of the maximum number of layers, or depth, it's allowed to have. The data available to train the decision tree will be split into a training set and test set and trees with various maximum depths will be created based on the training set and tested against the test set. Cross-validation can be used as part of this approach as well. Pruning the tree, on the other hand, involves testing the original tree against pruned versions of it. Leaf nodes are taken away from the tree as long as the pruned tree performs better against test data than the larger tree.

Changing the information gain formula:
The information gain formula used by the ID3 algorithm treats all of the variables the same, regardless of their distribution and their importance. This is a problem when it comes to continuous variables or discrete variables with many possible values because training examples may be few and far between for each possible value, which leads to low entropy and high information gain by virtue of splitting the data into small subsets, but results in a decision tree that might not generalize well.

One succesful approach to deal with this is using a formula called GainRatio in the place of information gain. GainRatio tries to correct for information gain's natural bias toward attributes with many possible values by adding a denominator to information gain called SplitInformation:

$$GainRatio(S,A) = \frac{Gain(S,A)}{SplitInformation(S,A)}$$

SplitInformation attempts to measure how broadly partitioned an attribute is and how evenly distributed those partitions are. The formula for data set *S* and an attribute *A* with *c* different partitions is:

$$SplitInformation(S,A) = -\sum_{i=1}^{c} \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}$$

If there's just 1 possible value for the attribute, then the formula equals $\log_2 1 = 0$. Luckily, we tend not to include attributes with 1 possible value in our training data because it is impossible to carry out the ID3 algorithm by splitting on an attribute with only 1 value, so GainRatio doesn't have to handle the possibility of a denominator of 0. On the other hand, our continuous temperature example has 14 possible values in our training data, each of which occurs once, which leads to $-\frac{1}{14}\log_2\frac{1}{14} * 14 = \log_2 14$. In general, the SplitInformation of an attribute with *n* equally-distributed values is $\log_2 n$. These relatively large denominators significantly affect an attribute's chances of being the best attribute after an iteration of the ID3 algorithm and help to avoid choices that perform particularly well on the training data but not so well outside of it.