Decision Tree Derivations

Mickey Chao

February 2016

1 Intuition Behind Decision Trees

To better fit training data, we have previously been applying feature mappings or kernels to learning our classifiers in higher dimensional feature spaces. One of the drawbacks is that it may not be as clear how the features interact with each other. For example, some kernels map to infinite-dimensional spaces, so we can't even specify what the feature mapping is - let alone determine what features are important. Even methods like linear or logistic classification may have countless features with small coefficients and it is difficult to figure out exactly what combinations of features are important.

We will look for a better, white-box method of learning a classifier where we can more-easily see how different interactions between features will help us classify our training data.

2 Entropy

Suppose we have a training set where the labels are $+$ and $-$. Let $P_\oplus$ denote the proportion of positive training examples and $P_\ominus$ denote the proportion of negative training examples. If we're in this situation have no additional information about our training data, our best chance at classifying new test examples is to guess proportionally to $P_\oplus$ and $P_\ominus$.

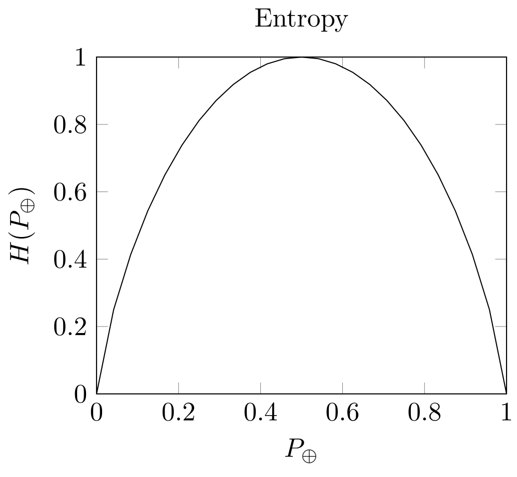
If conveniently $P_\oplus = 1$ and $P_\ominus = 0$, then we can be very certain that by guessing that all future test samples are positive, we'll classify them correctly. On the other hand, is $P_\oplus = 0.5$ and $P_\ominus = 0.5$, then we are very uncertain and might as well flip a coin to guess the classification.

We'd like to quantify the amount of uncertainty in a given situation. Consider the entropy, H, of a system defined as follows:

$$H = -\sum_P -P \log_2(P)$$

where the P are the proportions of the different classes that training examples may take.

For two classes, $\oplus$ and $\ominus$ where $P_\oplus + P_\ominus = 1$, we get a graph of H as follows:



We see that the graph of H has some of the properties we've described above. For example, as the proportions between $\oplus$ and $\ominus$ becomes more evenly split, our entropy peaks at 1. However, when the proportions become heavily biased towards $\oplus$ or $\ominus$ the entropy decreases towards 0.

Note that $\lim_{x \to 0} x \log_2(x) = 0$. We see this by applying L Hopital's rule:

$$\lim_{x \to 0} x \log_2(x) = \lim_{x \to 0} \frac{x}{\frac{1}{\log_2(x)}} = \lim_{x \to 0} \frac{1}{\frac{1}{x}} = \lim_{x \to 0} x = 0$$

so H is defined at $P_\oplus = 0$ and $P_\ominus = 0$.

Conditional Entropy and Information Gain

Oftentimes, when we are working with a dataset, the training dataset has many features. We want to look at which features best separate the training examples into different classes. Therefore, it becomes important for us to develop a measure of how well a feature is at separating training examples into different classes.

Recall the example of where $P_\oplus = P_\ominus$. If we select a feature that splits the training data into two subsets that both have an equal number of positive and negative examples, then that feature is not very good. As a concrete example, if our training data was initially [20+, 20-] and by splitting on feature f, we get $f^+ = [10+, 10-]$ and $f^- = [10+, 10-]$, then we haven't really learned much by splitting the dataset on feature f.

Also recall the example of where $P_\oplus = 1$. If we select a feature that perfectly separates the positive and negative examples, then that feature is very good. For example, if our training data was initially [20+, 20-] and by splitting on feature f, we have $f^+ = [20+, 0-]$ and $f^- = [0+, 20-]$, then we expect to be able to predict all test examples perfectly just by looking at the presence of feature f.

In both scenarios, we need to be able to quantify how much information we gained from splitting the training dataset on feature f.

Now, we can extend the definition of entropy to define conditional entropy, which is the entropy of the entire system given some feature. We can then compare the entropy given some feature with the entropy before knowing that feature. The difference would intuitively correspond to how much information we can gain by knowing some feature.

Let Y denote the label and X denote the feature. Simply applying the definition of entropy, we have

$$H(Y = y|X = x) = -P(Y = y|X = x) \log_2 P(Y = y|X = x)$$

We can condition over all possible values of X and Y and have

$$H(Y|X) = \sum_{y \in Y} \sum_{x \in X} H(Y = y|X = x)$$

$$= \sum_{y \in Y} \sum_{x \in X} -P(Y = y|X = x) \log_2 P(Y = y|X = x) P(X = x)$$

Finally, we can define the information gain to be

$$IG(X, Y) = H(Y) - H(Y|X)$$

which is basically how much the entropy decreases when we know feature X.

Decision Trees

Decision Trees are a straightforward application of information gain. The idea is to repeatedly split on the feature that gives us the most information gain until a stopping criterion has been met.

The three main situations in which we cannot split the tree anymore are when all the labels are identical, when all the training examples are the same, or when the maximal information gain is 0.

Here is some pseudocode:

```
BuildTree( dataset , nodeToExpand ):
    if $y^{(i)} == y$ for all $y \in$ dataset then
        add child $y$ to nodeToExpand
    if $x^{(i)} == x$ for all $x \in$ dataset then
        add majority label to nodeToExpand
    else
        splitFeature = $\text{argmin}_x H(Y|X)$
        for class in splitFeature:
            subset = {all $(x^{(i)}, y^{(i)}) \in$ dataset with splitFeature == class}
            BuildTree( subset , nodeToExpand )

main():
    BuildTree( dataset , empty node )
```

Generally, if left unattended, decision trees may grow to huge depths and overfit the data severely. To prevent overfitting, we apply any of the following techniques:

1. set a max depth
2. measure the cross validation performance and stop once it worsens
3. grow the entire tree and post-prune by greedily removing nodes that don't significantly impact cross-validation performance.

A single decision tree may do a good job on some data. However, we realize that it splits on features in a certain order. Sometimes, the features in some data have much more complex interactions. For example, it may be the case that f_1 leads to f_2 or f_3, but it may also be the case that f_4 leads to f_1, f_2, or f_3. We may not capture all interactions between with a single decision tree. So, we can turn towards methods that utilize more trees with more variation to make better hypotheses.

3 Bagging

Previously, we stated that a single decision tree may not capture complex interactions between features since the decision tree depends on the order in which we split on features. We now consider a more sophisticated method where we apply a diverse population of decision trees. The idea is that the wisdom of the crowd may be better than the wisdom of an individual.