Decision Trees

This is a heirarchical model where features that are higher up on the tree are considered more relevant. Each node tests an attribute and each branch represents an outcome of that test. The leaf nodes represent class labels.

If we have D binary features and a binary classification system then there are 2^{2^D} possible unique decision trees. However, we will explore the way to come up with the best decision tree for a given dataset.

If our data is noiseless and without and randomness then there is at least one trivially consistent decision tree for each data set (fits the data perfectly). There are many potential ones, but we want the one with the **smallest height** for both efficiency and generalization. Finding the optimal decision tree is NP-complete, so we will use a greedy algorithm to approximate it.

Creating a Decision Tree

```
function DTL(X,Y,features,probs)
    if X is empty then return a leaf node with probs as the probability for each class.
    elseif all Y have same values, return a leaf node with value Y_1
    elseif features is empty return a leaf node with probabilities for each class.
    else
        best = ChooseAttribute(X,Y,features)
        tree = new internal node testing feature best
        for each value v_i in best do
            [X^(i), Y^(i)] = elements of [X,Y] with best = v_i
            subtree = DTL(X^(i), Y^(i), features − best, prob(Y))
            add a branch to tree with label v_i and subtree subtree
        return tree
```

Figure 1: X is the observable data, Y is the class labels, probs is the class priors

initial call with be $DTL(Xtrain, Ytrain, [0 : D-1], probs(YTrain))$ where probs is the class prior distribution.

Choosing the Best Feature

The best case is where all of the data "agrees" with the class label. This means that the entropy is 0. The worst case is where the data is evenly split between the classes, meaning the entropy is 1. We can use this to choose the best feature to split on at each node.

The entropy of a set S is defined as:

$$H(P_{v_1}, \ldots, P_{v_K}) = \sum_{i=1}^{K} (-P(v_i) \log_k P(v_i))$$

. This builds what is known as the ID3 decision tree

For example, if there is a feature with 3 classes and a 1/5, 2/5, 2/5 split then the entropy is:

$$H = -(\frac{1}{5} \log_3 \frac{1}{5} + \frac{2}{5} \log_3 \frac{2}{5} + \frac{2}{5} \log_3 \frac{2}{5}) = 0.876$$

We will use the features with the lowest weighted average entropy to build out decision tree.

Example

Consider this sample data:

Looking at feature 1:

$$X = \begin{bmatrix} 1 & 1 \\ 1 & 3 \\ 2 & 2 \\ 1 & 2 \\ 3 & 2 \\ 2 & 2 \end{bmatrix}, Y = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

When the value is 1, there is a $\frac{2}{3}$ $\frac{1}{3}$ split, so the entropy is:

$$H = -\frac{2}{3}\log_2\frac{2}{3} - \frac{1}{3}\log_2\frac{1}{3}$$

When the value is 2, the split is 50/50 so the entropy is 1.

When the value is 3, it is deterministic so the entropy is 0.

The weighted average for feature 1 is

$$\frac{3}{6}(-\frac{2}{3}\log_2\frac{2}{3} - \frac{1}{3}\log_2\frac{1}{3}) + \frac{2}{6}1 + \frac{1}{6}0$$

Looking at feature 2:

When the value is 1, it is deterministic, so the entropy is 0. There is one instance of the value 1 in feature 1.

When the value is 2, there is a $\frac{1}{4}$ $\frac{3}{4}$ split, so the entropy is

$$H = -\frac{1}{4}\log_2\frac{1}{4} - \frac{3}{4}\log_2\frac{3}{4}$$

. There are 4 instances of the value 2 in feature 2.

When the value is 3, it is also deterministic, so the entropy is 0. There is 1 instance of the value 3 in feature 2.

The weighted average for feature 2 is

$$\frac{1}{6}0 + \frac{4}{6}(-\frac{1}{4}\log_2\frac{1}{4} - \frac{3}{4}\log_2\frac{3}{4}) + \frac{1}{6}0$$

Eyeballing this, feature 2 has a lower weighted average entropy because there are two deterministic splits, so we will start with feature 2 first to build the decision tree.

Predicting

```
function predict(x,node)
    if isLeaf(node)
        return node.label
    else
        child = node.child[x[node.feature]]
        return predict(x,child)

yhat = predict(x,root)
```

Overfitting

Decision trees that are fully built out might overfit. Use a validation set to determine when to stop growing the tree. If adding a new node does not improve validation accuracy, then stop growing that branch.

You can also go the other way by using pruning. This is where you build the full tree and then remove nodes that do not improve validation accuracy.