# Decision Trees

(following AIxploratorium)

You want to predict whether your basketball team is going to win or lose, based on data collected from previous game, and what you know about the upcoming game. In particular, from previous games you have collected:

Win or loss?
Home or away?
Time of game (5, 7, or 9)?
Does Fred start?
Is Joe forward or center?
Does Joe defend forward or center?
Is opponent team's center tall or (relatively) short?

For the upcoming game, you know: Away, 9pm, Fred does not start, Joe is center, Joe defends forward, and opponent center is tall.

So, what do you think? Well, in total they have lost 10 and won 10, so that doesn't help. They also have lost exactly half of home games and half of away games, so that doesn't help either. But, they have lost all 4 of their 9pm games, so that doesn't bode well. Then again, when Joe plays offense, they won 8 out of 11. Hmm…. Maybe some combination of factors is responsible: maybe Joe plays well playing a short opponent center whenever playing center himself, … though maybe only at home games. Etc.

OK, what we need is a model: Something that is a little more systematic! Something that explains the data from the 20 games. And something that can provide predictions to any other future game. Well, decision trees are like that. So, let's make a decision tree:

(run applet with splitting -> random and pruning -> none settings a couple of times)

Notice that we get different trees each time. They all provide perfect explanations of the data collected, because we made sure to set the 'Win' or 'Lose' leaves at the ends of the branches in accordance with the data. However, we also notice that some trees predict a win for the upcoming game, while others predict a loss. So, how are we to trust one tree over another? Can we maybe say that one tree is better than another?

Well, one thing to notice is that some decisions (predictions) are based on more past games than others. That is, sometimes we set a leaf to 'Win' because there were 4 games of the particular type belonging to that branch of the tree, and they all led to a win, while there are other leafs of the tree for which there was only 1 corresponding game in the data set. And, intuitively, the former decision is a more reliable decision than the latter (from statistics: the greater your sample, the better!). So, the more games from your data set correspond to a leaf, the better. In terms of whole trees, this translates to: the fewer the number of leafs, the better. Or, simply put: the smaller tree, the better!

OK, so one thing we can do is to generate a bunch of random trees and pick one that is small. In fact, we could even try and generate all possible decision trees, and pick the very smallest one (assuming there is one 'smallest'). Will that be a good (or possibly the best) tree? Well, it is quite possible that the original problem of basing predictions on a very small number of games remains, even for the smallest tree. Indeed, maybe some predictions are still going to be based on a single past game, which is not a good thing.

So, another thing we can do is to prune a tree. That is, once a tree has been generated (through whatever method), we may want to check the leafs and make sure that they are based on some minimum number of games (say, at least 4). If not, we go to an earlier decision point, and possibly make a decision there. For example, if there were 4 games associated with some node, and 3 out of those 4 led to a win, then we may want to change that node into a 'Win' prediction, and thus pruning the branches that led to 'unanimous' outcomes of less than 4 games.

(run applet with splitting -> random and pruning -> pessimistic a couple of times)

You'll notice that when you prune a tree, you no longer get 100% accuracy on the data set. This is the price you pay for pruning a tree, but also think of it this way: trying to have your model explain every single bit of data should really not be required:

1. First of all, remember that we're dealing with basketball games here: the outcome of basketball games is determined by a very large number of factors, including just some pure luck. So, it is unlikely that the features about which we have collected our data are all the relevant features. It also means that we could have multiple games with the exact same features but which led to different outcomes, in which case a 100% accurate explanation of the facts in terms of the features looked at is simple impossible (this is actually not something that ever happens in the datasets that come with the applet, but it certainly can happen!)
2. Second, the collected data is always subject to noise: maybe something went wrong in the measurement or recording of the data. This is something that can happen even if you are looking at all the relevant features, and so once again a 'perfect' explanation is simply out of the question.

Making sure that your predictions are based on a larger number of games is actually a good way to deal with these problems. So: keeping a tree small (or pruning a large tree) is in several ways a good idea. In fact, the larger the tree, the more danger there is that you are 'overfitting the data'. That is, as you go further down the tree, and as the tree gets more and more complicated, you are dealing with fewer and fewer games, and hence the particular results for those few games may be 'idiosyncratic' to those very few games, i.e. you may be going into details that are not at all part of a theory that is true in general. Indeed, this is exactly why Ockham's Razor says to choose the 'simplest' theory when presented with several theories that all explain the data equally well. In fact, for all the reasons stated above, it can make perfect sense to prefer a simpler theory that isn't quite as good at explaining the available data over a more complicated theory that explains all!

So, the general slogan is: smaller trees are better! (of course, your trees can also become too small!) Now, other than pruning trees (or, what is effectively the same thing: making sure not to branch under certain conditions), is there something else we can do to prevent big trees? Yes! Consider a tree that starts out asking whether it was a home or away game. Notice that this question doesn't seem to really help, as they lost half of the home games, and half of the away games. However, other than this therefore being be a bit of a 'lost' or 'wasted' question, there is an even deeper problem here. Since this question splits the original 20 games into 12 and 8 games respectively, and what this means, is that subsequent questions (whether following the one branch, or the other) are going to be asked only of this subset of games, and that again decreases the likelihood that the data from those games generalize to all possible.  At the same time, since the answer to this question does not predispose the prediction to go one way or the other, we'll still need to ask a bunch of questions before we get to the leaves. So, however way you slice it, the resulting tree will not be very good.

What to do? Well, the above discussion reveals an obvious way to do this. Asking 'Where' at the beginning apparently doesn't help: it doesn't get us any closer to a decision; it has no 'gain'. Rather, we want to ask questions that do seem to get us closer to the answer, while keeping the number of games sizeable at the same time. Thus, we use a kind of 'gain' measure that tries to quantify this. I won't get into any details here (and indeed, there are many reasonable ways of defining such measures), but the applet has several of such measures. Let's see what happens.

(applet: use splitting -> gain, pruning -> none)

Since the algorithm simply picks the highest gain at each node, the tree generated will now be the same every time you run the program. Notice also that the generated tree is fairly small (or at least smaller than most trees generated randomly).  Of course, you can always prune the tree to make it even smaller, either by hand, or by using the automated pruning function.

(applet: use splitting -> gain, pruning -> pessimistic)

OK, is that it? No!  Emphatically no!

(run applet on Froglegs data set)

In the Froglegs dataset, the outcome of the game is supposed to be a result of whether the birthdays of 50 random spectators at the game fall within the first half of the year, or the second half… which we know is certainly not the case!  Using 'gains' or pruning or what have you is not going to solve the problem here, which is that while the tree may be a perfect fit of the data, it does not in any way generalize to any future games. Indeed, any tree generated for this data set is a tree that purely reflects idiosyncracies of the data set; a tree that overfits the data from the very start! Indeed, this is a perfect example of how in science you can always tell some story that explains all the data: that's not hard to do! What is hard, however, is to come up with a story that generalizes to other circumstances.

What to do? Well, you do what you always do when trying to give a model, explanation, theory, or hypothesis in science: you test your model! Indeed, while Ockham's Razor is nice and all, the far preferred way to choose between two theories is simple to have them both make predictions, and see which predictions come out true.

Unfortunately, in the basketball example, we may have only 1 game every week to test our tree: highly inconvenient! However, there is a smarter way of going about this: you simple take your data set, and split it into two: a training set and a validation (or testing) set. The training set will be used as before to generate the tree, but the validation or testing set will be used to see if your tree does indeed generalize. Let's see what happens:

(pick Froglegs data set. First, create a testing set. Then run algorithm (using whatever splitting or pruning settings))

You should find that the generated tree does well in explaining the training set (of course!), the accuracy for the testing set is basically the same as for just flipping a coin! So, we have now confirmed that indeed the tree has not picked up on anything that is predictive of the outcome of the game (of course: there were no such features in the first place!)

(pick some other data set, create testing set, run algorithm)

You should find that the testing set has better than chance accuracy, though for some datasets it's better than others, giving you some measure of the reliability of the tree for making predictions.

Splitting your data set into a training set and testing set is a standard thing to do when trying to generate models out of data. Indeed, it is basically the difference between doing good science (test your theories!) and bad science (just tell nice story that fits the facts). One common practice when doing so is to keep score of the error of the training and testing sets as the theory gets more and more elaborate. As you do so, you'll find that the training error may eventually go down to zero (i.e. your model explains all the data to a T), but that at some point the testing error no longer goes down: this tells you that from that point on, you're just fitting idiosyncracies, i.e. you're overfitting the data. In the case of generating a tree, we can consider what happens to the testing error when we just introduced any branch based on the training data: if this branch does not improve testing accouracy, then it's probably time to create a leaf and prune the tree right there. This is basically what the ReducedError option does:

(pick some data set, create testing set, run algorithm with pruning -> ReducedError)

What are still some drawbacks of decision trees?

1. *Garbage in* -> *Garbage out*: the algorithm works with the data set as provided: the algorithm will not realize that certain features that are not being considered may be important features for the model to consider.
2. *Forced ordering*: While using something like the gain will make clear what would be a sensible ordering in your consideration of features, in the end we still end up with some kind of ordering of features: we go into completely different parts of a tree based on a single feature at a time. Wouldn't it make sense to consider all the features at once?
3. *Features left out*: Some features may be left out. While the way that the tree is generated should ensure that important features will make it into the tree, it may leave out some not-so-important, but still telling features. Again, wouldn't it make more sense to consider all features?
4. *Continuous values*: the decision tree method works with features that have discrete values. How can it deal with continuous values? We could predetermine certain 'bins' or intervals of values, but how do you determine those? And does it make sense to use such 'bins' in the first place? In short, decision trees don't do well with features that can take on a continuous value.