

Decision Trees

We've spent a lot of time talking about linear models—models which are parameterized by a weight vector of equal dimension to the input vectors. These are nice, but limited. Here, we briefly consider a very different technique: decision trees (think: playing twenty questions).

Idea: suppose we could only use one feature to make a classification decision. Let's choose that feature. Now, look at all example for which this feature is on and choose a single feature to make a classification decision. Then look at all for which the first feature was off. Recurse until no data left.

Two issues: (a) how to choose a single feature, (b) how to choose to stop.

The simplest way to choose a feature is to choose the feature that gives you the *lowest classification error* (on the training data). There are more complicated ways (one of the more popular, ID3, uses the entropy of the distribution of labels—equivalent to doing a maximum information gain step) but those are minor tweaks.

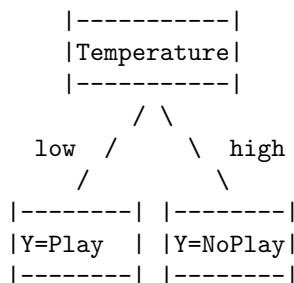
Let's consider our play/no-play data set:

Class	Outlook	Temperature	Windy?
Play	Sunny	Low	Yes
No play	Sunny	High	Yes
No play	Sunny	High	No
Play	Overcast	Low	Yes
Play	Overcast	High	No
Play	Overcast	Low	No
No play	Rainy	Low	Yes
Play	Rainy	Low	No

We have three features we could split on:

- Outlook: In this case, on the “Sunny” branch we'd classify as “No play” and we'd have 1 error (the play example), on the “Overcast” branch we'd classify as “Play” have 0 errors and on the “Rainy” branch we'd classify as either and get 1 error. Thus, Outlook yields an error rate of two.
- Temperature: On the “Low” branch, we'd classify as “Play” and have one error; on the “High” branch, we'd classify as “No Play” and also have one error. Thus, Temperature also yields an error rate of two.
- Windy: On “Yes”, we'd classify as either and get two errors; on “No” we'd say “Play” and get one error. Thus, Windy yields three errors.

So we arbitrarily choose between Outlook and Temperature. Let's say we choose temperature. We now have a tree that looks like:



Now, we can recurse. For the left-child (“low”), we have the following filtered data:

Class	Outlook	Temperature	Windy?
Play	Sunny	Low	Yes
Play	Overcast	Low	Yes
Play	Overcast	Low	No
No play	Rainy	Low	Yes
Play	Rainy	Low	No

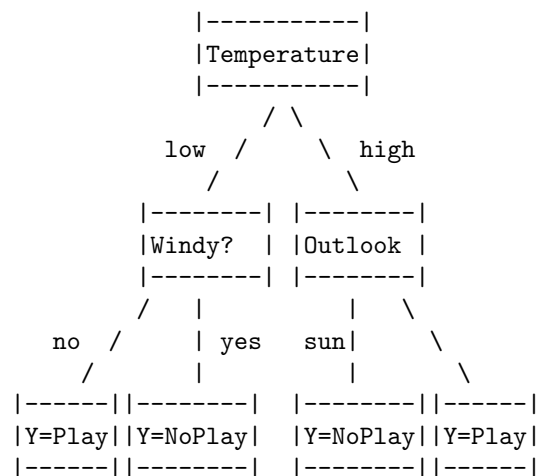
We can no longer split on temperature. If we split on outlook, we’ll make one error (rainy); if on windy, we’ll make one error (yes). This is better than we’re doing right now, which is making two errors. Say we split on windy.

For the right child (“high”) we have the following filtered data:

Class	Outlook	Temperature	Windy?
No play	Sunny	High	Yes
No play	Sunny	High	No
Play	Overcast	High	No

Now it’s obvious: we should split on outlook.

Doing so yields the following tree.



All of the children are done except for low/yes, which would like to split again (on outlook) to achieve perfect classification.

(Note: I did this in breadth-first manner; a real implementation would probably be depth-first.)

Stopping: use threshold of either number of elements in leaf or misclassification rate of leaf.

Dealing with real-valued features: if X is real-valued, consider all possible split locations ($X \leq z$ and $X > z$) and find the best z to split. Only need to search over splits that exist in training data.