Classification Trees

36-350, Data Mining

29 Octber 2008

Classification trees work just like regression trees, only they try to predict a discrete category (the class), rather than a numerical value. The variables which go into the classification — the inputs — can be numerical or categorical themselves, the same way they can with a regression tree. They are useful for the same reasons regression trees are — they provide fairly comprehensible predictors in situations where there are many variables which interact in complicated, nonlinear ways.

1 An Example: Mushrooms

A field guide to mushrooms lists 5416 varieties, with 22 categorical attributes for each, as well as the class, which is "edible" or "poisonous". The first row:

```
cap.shape cap.surface cap.color bruises odor gill.attachment gill.spacing
   convex     fibrous       red bruises none            free        close
gill.size gill.color stalk.shape stalk.root stalk.surface.above.ring
    broad     purple    tapering    bulbous                   smooth
stalk.surface.below.ring stalk.color.above.ring stalk.color.below.ring
                  smooth                   gray                   pink
veil.type veil.color ring.number ring.type spore.print.color population
  partial      white         one   pendant             brown    several
habitat   class
  woods edible.
```

Despite all the data, the classification tree is simple (Figure 1). It manages to classify perfectly using an OR-rule: if a mushroom smells bad, OR it has green spores, OR it grows in clusters, then it is poisonous.[1]

2 Growing Classification Trees

We find classification trees in almost the same way we found regression trees: we start with a single node, and then look for the binary distinction which gives

[1]Apparently-perfect classifications like this should make one suspicious. I have *not* run this by a mycologist to see if it makes sense, though it is the kind of result which is *worth* running by a mycologist. Anyway, if you go mushroom-hunting on this basis and get sick and die, don't blame me.

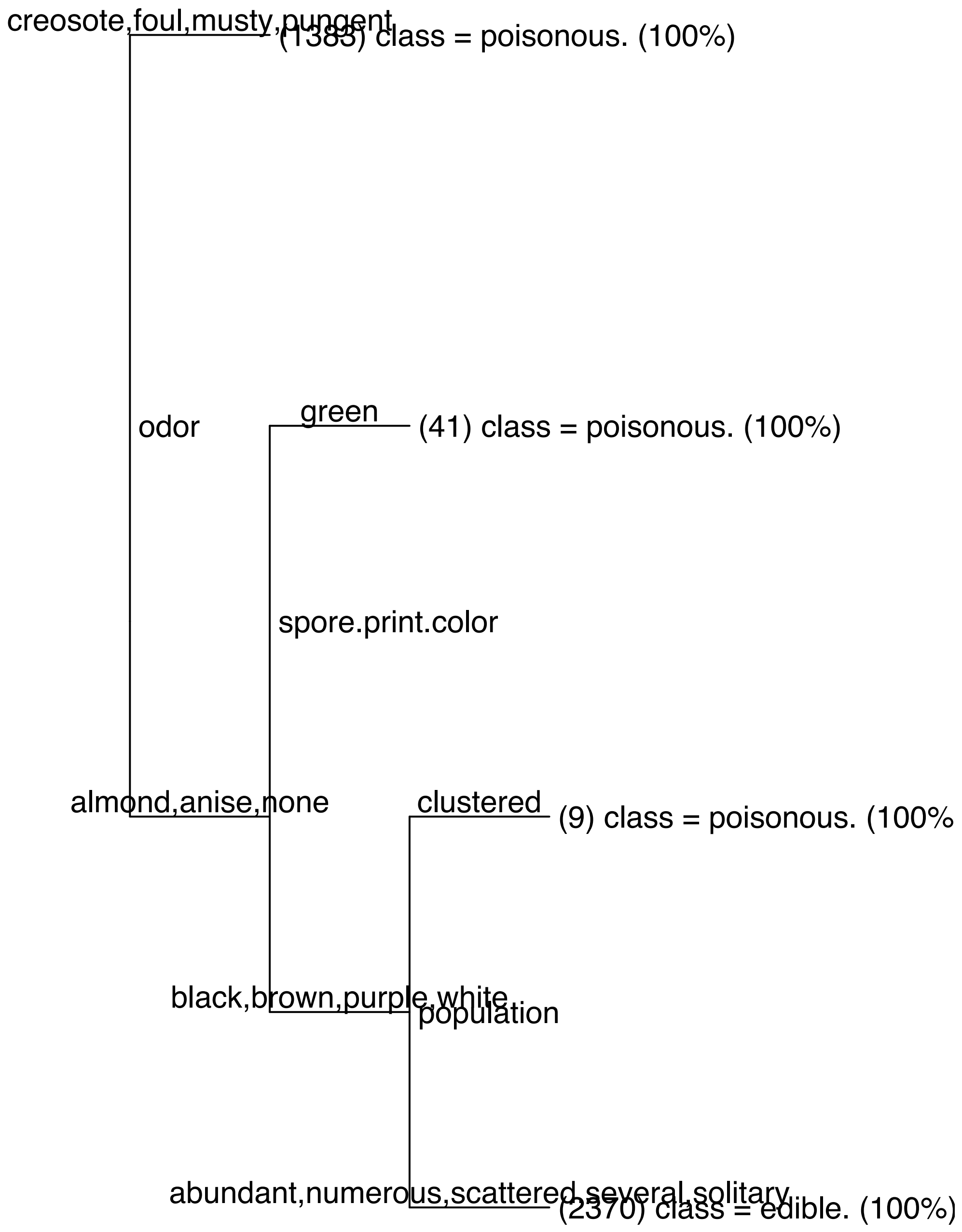
creosote,foul,musty,pungent
(1383) class = poisonous. (100%)
odor
green
(41) class = poisonous. (100%)
spore.print.color
almond,anise,none
clustered
(9) class = poisonous. (100%
black,brown,purple,white
population
abundant,numerous,scattered,several,solitary
(2370) class = edible. (100%)

Figure 1: Classification tree for North American wild mushrooms.

us the most information about the class. We then take each of the resulting new nodes and repeat the process there, continuing the recursion until we reach some stopping criterion. The resulting tree will often be too large (i.e., over-fit), so we prune it back using (say) cross-validation. The differences from regression-tree growing have to do with (1) how we measure information, (2) what kind of predictions the tree makes, and (3) how we measure predictive error.

2.1 Measuring Information

The response variable Y is categorical, so we can use information theory to measure how much we learn about it from knowing the value of another discrete variable A:

$$I[Y;A] = \sum_{a} \Pr\left(A=a\right) I[Y;A=a] \tag{1}$$

where

$$I[Y;A=a] = H[Y] - H[Y|A=a] \tag{2}$$

and you remember the definitions of entropy $H[Y]$ and conditional entropy $H[Y|A=a]$.

$I[Y;A=a]$ is how much our uncertainty about Y decreases from knowing that $A=a$. (Less subjectively: how much less variable Y becomes when we go from the full population to the sub-population where $A=a$.) $I[Y;A]$ is how much our uncertainty about Y shrinks, on average, from knowing the value of A.

For classification trees, A isn't (necessarily) one of the predictors, but rather the answer to some question, generally binary, about one of the predictors X, i.e., $A = \mathbf{1}_{\mathcal{A}}(X)$ for some set $\mathcal{A}$. This doesn't change any of the math above, however. So we chose the question in the first, root node of the tree so as to maximize $I[Y;A]$, which we calculate from the formula above, using the relative frequencies in our data to get the probabilities.

When we want to get good questions at subsequent nodes, we have to take into account what we know already at each stage. Computationally, we do this by computing the probabilities and informations using only the cases in that node, rather than the complete data set. (Remember that we're doing *recursive* partitioning, so at each stage the sub-problem looks just like a smaller version of the original problem.) Mathematically, what this means is that if we reach the node when $A=a$ and $B=b$, we look for the question C which maximizes $I[Y;C|A=a,B=b]$, the information *conditional* on $A=a$, $B=b$. Algebraically,

$$I[Y;C|A=a,B=b] = H[Y|A=a,B=b] - H[Y|A=a,B=b,C] \tag{3}$$

Computationally, rather than looking at all the cases in our data set, we just look at the ones where $A=a$ and $B=b$, and calculate as though that were all the data. Also, notice that the first term on the right-hand side, $H[Y|A=a,B=b]$, does not depend on the next question C. So rather than maximizing $I[Y;C|A=a,B=b]$, we can just minimize $H[Y|A=a,B=b,C]$.

2.2 Making Predictions

There are two kinds of predictions which a classification tree can make. One is a **point** prediction, a single guess as to the class or category: to say "this is a flower" or "this is a tiger" and nothing more. The other idea is to give a *probability* for each of the classes. This is slightly more general, because if we need to extract a point prediction from a probability forecast we can always do so, but we can't go in the other direction.

For probability forecasts, each terminal node in the tree gives us a distribution over the classes. If the terminal node corresponds to the sequence of answers $A = a, B = b, \ldots Q = q$, then ideally this would give us $\Pr(Y = y | A = a, B = b, \ldots Q = q)$ for each possible value y of the response. A simple way to get close to this is to use the empirical relative frequencies of the classes in that node. E.g., if there are 33 cases at a certain leaf, 22 of which are tigers and 11 of which are flowers, the leaf should predict "tiger with probability 2/3, flower with probability 1/3". This is the *maximum likelihood* estimate of the true probability distribution, and we'll write it $\widehat{\Pr}(\cdot)$.

(Incidentally, while the empirical relative frequencies are consistent estimates of the true probabilities under many circumstances, nothing particularly *compells* us to use them. When the number of classes is large relative to the sample size, we may easily fail to see any samples at all of a particular class. The empirical relative frequency of that class is then zero. This is good if the actual probability is zero, not so good otherwise. (In fact, under the negative log-likelihood error discussed below, it's infinitely bad, because we will eventually see that class, but our model will say it's impossible.) The empirical relative frequency estimator is in a sense too reckless in following the data, without allowing for the possibility that it the data are wrong; it may under-smooth. Other probability estimators "shrink away" or "back off" from the empirical relative frequencies. One popular approach is to add 1/2 to the empirical count of each class (even for classes not observed in the sample), and then divide by the sum of these inflated counts. EXERCISE: Can you show, using the law of large numbers, that this is also consistent?)

For point forecasts, the best strategy depends on the loss function. If it is just the mis-classification rate, then the best prediction at each leaf is the class with the highest conditional probability in that leaf. With other loss functions, we should make the guess which minimizes the expected loss. But this leads us to the topic of measuring error.

2.3 Measuring Error

There are three common ways of measuring error for classification trees, or indeed other classification algorithms: misclassification rate, expected loss, and normalized negative log-likelihood, a.k.a. **cross-entropy**.

2.3.1 Misclassification Rate

We've already seen this: it's the fraction of cases assigned to the wrong class.

2.3.2 Average Loss

The idea of the average loss is that some errors are more costly than others. E.g., for the mushrooms, the possible values of Y were "edible" and "poisonous". If we think an edible mushroom is poisonous, we don't eat it and miss out on a tasty treat. The consequences of thinking a poisonous mushroom is edible are much worse! There will be a different cost for each combination of the real class and the guessed class; write L_{ij} for the loss we incur by saying that the class is j when it's really i.

For an observation x, the classifier gives class probabilities $\Pr(Y = i|X = x)$. Then the expected cost of predicting j is:

$$\text{Loss}(Y = j|X = x) = \sum_i L_{ij} \Pr(Y = i|X = x)$$

Suppose the cost matrix is biased against class "edible":

	prediction: "poisonous"	prediction: "edible"
truth "poisonous"	0	10
truth "edible"	1	0

We run an observation through the tree and wind up with class probabilities $(0.4, 0.6)$. The most likely class is "edible", but it is not the most cost-effective decision. The expected cost of predicting "poisonous" is $0.4 * 0 + 0.6 * 1 = 0.6$, while the expected cost of predicting "edible" is $0.4 * 10 + 0.6 * 0 = 4$. The probability of $Y =$ "edible" must be 10 times higher than that of $Y =$ "poisonous" before "edible" is a cost-effective prediction.

Notice that if our estimate of the class probabilities is very bad, we can go through the math above correctly, but still come out with the wrong answer. If our estimates were exact, however, we'd always be doing as well as we could, given the data.

You can show (and will, in the homework!) that if the costs are symmetric, we get the mis-classification rate back as our error function, and should always predict the most likely class.

2.3.3 Likelihood and Cross-Entropy

Finally, the normalized negative log-likelihood is a way of looking not just at whether the model made the wrong call, but whether it made the wrong call with confidence or tentatively. ("Often wrong, never in doubt" is *not* a good idea.) More precisely, this loss function for a model Q is

$$L(\text{data}, Q) = -\frac{1}{n} \sum_{i=1}^{n} \log Q(Y = y_i|X = x_i)$$

where $Q(Y = y|X = x)$ is the conditional probability the model predicts. If perfect classification were possible, i.e., if Y were a function of X, then the best classifier would give the actual value of Y a probability of 1, and $L = 0$. If there is some irreducible uncertainty in the classification, then the best possible classifier would give $L = H[Y|X]$, the conditional entropy of Y given the inputs X. Less-than-ideal predictors have $L > H[Y|X]$. To see this, try re-write L so we sum over values rather than data-points:

$$\begin{aligned}
L &= -\frac{1}{n}\sum_{x,y} N(Y=y, X=x)\log Q(Y=y|X=x) \\
&= -\sum_{x,y} \widehat{\Pr}\left(Y=y, X=x\right)\log Q(Y=y|X=x) \\
&= -\sum_{x,y} \widehat{\Pr}\left(X=x\right)\widehat{\Pr}\left(Y=y|X=x\right)\log Q(Y=y|X=x) \\
&= -\sum_{x} \widehat{\Pr}\left(X=x\right)\sum_{y}\widehat{\Pr}\left(Y=y|X=x\right)\log Q(Y=y|X=x)
\end{aligned}$$

If the quantity in the log was $\Pr\left(Y=y|X=x\right)$, this would be $H[Y|X]$. Since it's the model's estimated probability, rather than the real probability, it turns out that this is always *larger* than the conditional entropy. L is also called the **cross-entropy** for this reason.

There is a slightly subtle issue here about the difference between the in-sample loss, and the expected generalization error or risk. $N(Y = y, X = x)/n = \widehat{\Pr}\left(Y=y, X=x\right)$, the empirical relative frequency or empirical probability. The law of large numbers says that this converges to the true probability, $N(Y=y, X=x)/n \rightarrow \Pr\left(Y=y, X=x\right)$ as $n \rightarrow \infty$. Consequently, the model which minimizes the cross-entropy in sample may not be the one which minimizes it on future data, though the two ought to converge. Generally, the in-sample cross-entropy is lower than its expected value.

Notice that to compare two models, or the same model on two different data sets, etc., we do not need to know the true conditional entropy $H[Y|X]$. All we need to know is that L is smaller the closer we get to the true class probabilities. If we could get L down to the cross-entropy, we would be exactly reproducing all the class probabilities, and then we could use our model to minimize any loss function we liked (as we saw above).[2]

[2] Technically, if our model gets the class probabilities right, then the model's predictions are just as informative as the original data. We then say that the predictions are a **sufficient statistic** for forecasting the class. In fact, if the model gets the exact probabilities wrong, but has the correct partition of the feature space, then its prediction is still a sufficient statistic. Under any loss function, the optimal strategy can be implemented using *only* a sufficient statistic, rather than needing the full, original data. This is an interesting but much more advanced topic; see, e.g., David Blackwell and M. A. Girshick, *Theory of Games and Statistical Decisions* (New York: Wiley, 1954; Dover Books reprint, 1979) for details.