Decision Trees (cont.), Classification

Somani, Neel - Bao, Jason

March 6th, 2018

1 Reviewing the Decision Tree Algorithm

We started today's lecture by reviewing why decision trees were powerful: they're interpretable, and can effectively model complex decision boundaries. (This has its disadvantages, too, if you recall from the bias-variance decomposition.)

First, let's review the decision tree algorithm that we outlined in the previous lecture:

1. Find the feature that minimizes $J(dataleft) + J(dataright)$ where $J(d)$ is some cost function that we will discuss later and $dataleft$ and $dataright$ are the data points that belong to each side of the feature (e.g. whether it is $>$ or $<=$ 75% humidity).
2. Repeat algorithm on the data that got split to the right and the data got split to the left.
3. End at a desired depth.

We want to pick a good cost function $J(data)$.

Recall the definition of entropy:

$$entropy = H(S) = -\sum p * log(p) \tag{1}$$

where p is some probability. We can view this through another lens. Let:

$$surprise = -log(p) \tag{2}$$

Notice that $surprise$ is infinite is $p = 0$ (as expected, since a zero probability event occurred), and $surprise = 0$ when $p = 1$. Now we can define entropy as:

$$entropy = E[surprise] = \sum_C p_i * -log(p_i) \tag{3}$$

where p_i are the probabilities of the events in the event space. While we won't go into detail about entropy in this course, we do note that entropy will be the cost function that we use to decide which feature to split on for our decision tree, as discussed in the previous note.

Specifically, for each split, we aim to maximize:

$$H(S) - H_{after}(S)$$

a.k.a., *information gain.* That is equivalent to minimizing H_{after}.

1.1 Picking a split

For a binary variable, it's easy to split the data into two groups and determine the entropy before and after the split.

For discrete random variables, we can split between the values observed.

How should we handle a continuous random variable? Answer: In practice, we treat it the same way as the discrete case, by picking discrete values in the range of the continuous random variable to potentially split on, and picking the value that maximizes the information gain.

1.2 Avoiding overfitting

Here are some ways that we can avoid overfitting.

- Design some function/threshold that operates on the set of training data that decides whether or not you should split. (see the next two bullet points)
- Threshold on information gain (once splitting gives you less than some threshold of information gain, stop splitting)
- Threshold on class frequency (once the classes have fewer than some number of data points, stop splitting)
- Validate with cross-validation
- Limit depth of the tree
- Have a lot of trees

These are all great ideas. To elaborate on the last point, what would happen if we ran our decision tree algorithm on our set of data multiple times? Would we get different trees? Answer: No; the algorithm will produce the exact same tree every time, since this is the tree that uniquely maximizes information gain.

To produce different trees, we can train each tree on a different subset of training data or features. This is the general premise behind a method called *random forests.* At test time, we'll return the mode/average of the tree's output.

This is a very common and effective technique in practice. In industry, if we use decision trees, we generally do use random forests of some type. They're even used in self-driving cars (although the field is moving progressively toward deep learning).

2 Application of MLE

We continued today's lecture with an example of an application of maximum likelihood estimation. Consider a classification problem where we want to model S such that:

$$Pr[S = 1] = p_S, Pr[S = 0] = 1 - p_S$$

We observe $Y = S + N$, where $N \sim N(0, 1)$. In other words, we observe noisy estimates of S. Maybe we have some group of people, and we're trying to determine the probability that they'll default on a loan, or something like that.

What is the PDF of $Y|S$? $Y \sim N(S, 1)$, so:

$$f_{Y|S=1} = \frac{1}{\sqrt{2\pi}} e^{\frac{-(Y-1)^2}{2}}$$

$$f_{Y|S=0} = \frac{1}{\sqrt{2\pi}} e^{\frac{-(Y)^2}{2}}$$

Now we want to determine if $Pr[S = 1|Y] > Pr[S = 0|Y]$. Note that:

$$Pr[S = k|Y] = Pr[Y|S = k] * Pr[S = k]/Pr[Y]$$

This leads us to the inequality:

$$Pr[Y|S = 1] * Pr[S = 1]/Pr[Y] > Pr[Y|S = 0] * Pr[S = 0]/Pr[Y]$$

$$Pr[Y|S = 1] * Pr[S = 1] > Pr[Y|S = 0] * Pr[S = 0]$$

$$e^{\frac{-(Y-1)^2}{2}} * p_S > e^{\frac{-(Y)^2}{2}} * (1 - p_S)$$

$$log(e^{\frac{-(Y-1)^2}{2}} * p_S) > log(e^{\frac{-(Y)^2}{2}} * (1 - p_S))$$

$$-\frac{Y^2}{2} + \frac{Y}{2} - \frac{1}{2} + log(p_S) > -\frac{Y^2}{2} + log(1 - p_S)$$

$$Y > \frac{1}{2} + log(1 - p_S) - log(p_S)$$

$$Y > \frac{1}{2} + log(\frac{1 - p_S}{p_S})$$

3 Classification

Now we'll continue on to the problem of classification. Given a set of data $X_1, ...X_n$ and categorical labels $Y_1, ..., Y_n$ where $Y_i \in \{0, 1\}$, we want to estimate $Pr[y_i = 0|x_i]$. Just as we used decision trees for classification, we can use a method called *logistic regression.*

3.1 Logistic Regression

Just as we did for the derivation of MAP for conditionally independent samples, we can write the probability of observing data from any distribution:

$$Pr[Y_1, ..., Y_n|X_1, ..., X_n] = \prod_{i=1}^{n} Pr[Y_i|X_i]$$

Since $Y_i \in \{0, 1\}$, we can write this compactly:

$$Pr[Y_1, ..., Y_n|X_1, ..., X_n] = \prod_{i=1}^{n} Pr[Y_i = 1|X_i]^{Y_i} * Pr[Y_i = 0|X_i]^{1-Y_i}$$

$$= \prod_{i=1}^{n} Pr[Y_i = 1|X_i]^{Y_i} * (1 - Pr[Y_i = 1|X_i])^{1-Y_i}$$

If we have some distribution P with parameters we want to maximize, this is equivalent to maximizing:

$$max \prod_{i=1}^{n} P(Y_i = 1|X_i)^{Y_i} * (1 - P(Y_i = 1|X_i))^{1-Y_i}$$

$$= max \sum_{i=1}^{n} Y_i * log(P(Y_i = 1|X_i)) + (1 - Y_i) * log(1 - P(Y_i = 1|X_i))$$

Now we need to define P. Let's use the *logistic function*:

$$S(\gamma) = \frac{1}{1 + e^{-\gamma}}$$

This takes any real number, and outputs $S(\gamma) \in (0, 1)$. Note that:

$$S'(\gamma) = S(\gamma) * (1 - S(\gamma))$$

Let's finally define P as:

$$P(Y_i = 1|X_i) = S(w^T * X_i)$$

where w is some parameter that we want to learn. So we rewrite our objective function:

$$argmax_w \sum_{i=1}^{n} Y_i * log(S(w^T * X_i)) + (1 - Y_i) * log(1 - S(w^T * X_i))$$

Finally, we need to learn w. Unfortunately, we can't use a clean closed form solution like with OLS. We'll have to use *gradient descent.* The logistic function is fortunately convex, as covered in our earlier lectures; this means that gradient descent is guaranteed to return the global optimum.