UGBA 198 - Lecture 6

Somani, Neel - Bao, Jason

February 27th, 2018

1 MAP: Maximum a posteriori estimation

Let's recall our ordinary least squares model:

$$x_i * w = y_i$$

If we have a set of $X_1, ...X_n$ and a set of corresponding $Y_1, ..., Y_n$, then we can solve for the value of w that best fits this training data.

But what if we don't want the value of w that best fits the training data? For example, what if we have some prior knowledge about where w should be located? In the case that our data is sparse or noisy, this kind of prior knowledge can lead to a far more generalizable model.

If this is the case, then we might want to impose some *bias* in our model, as we've reviewed previously. We can statistically justify this approach with *maximimum a posteriori estimation*, or MAP estimation.

While MLE solved for the value of w that maximized:

$$arg\,max_w Pr[Y_1, ..., Y_n|w]$$

MAP instead seeks to maximize:

$$arg\,max_w Pr[w|Y_1, ...Y_n] \tag{1}$$

This way, we can impose a *prior distribution* on w. i.e., we can statistically weight evidence in a way that is optimal, given our prior knowledge of where w is likely to be located.

1.1 Simplifying the expression

Recall Bayes' rule:

$$Pr[A|B] = \frac{Pr[B|A] * Pr[A]}{Pr[B]} \tag{2}$$

We rewrite the MAP expression above:

$$arg\,max_w Pr[w|Y_1, ...Y_n]$$

$$= arg\,max_w \frac{Pr[Y_1, ..., Y_n|w] * Pr[w]}{Pr[Y_1, ..., Y_n]}$$

$$= arg\,max_w Pr[Y_1, ..., Y_n|w] * Pr[w] \tag{3}$$

where the last line follows since the unconditional probability $Pr[Y_1, ...Y_n]$ does not vary with respect to w.

1.2 Using MAP estimation: Justifying Ridge Regression

What's a reasonable prior distribution to impose on w? Well, let's suppose:

$$w \sim N(0, \frac{1}{\lambda}) \tag{4}$$

and

$$y_i|x_i, w \sim N(x_i * w, 1) \tag{5}$$

where the latter equation comes from our ordinary least squares white noise assumption (see the lecture note on MLE if this is unclear). The reasoning for the choice of $Var(w) = \frac{1}{\lambda}$ will become clear in this section.

Then:

$$arg\,max_w Pr[Y_1, ..., Y_n|w] * Pr[w]$$

$$= arg\,max_w (\sum_{i=1}^{n} log(Pr[Y_i|w])) + log(Pr[w])$$

$$= arg\,max_w (\sum_{i=1}^{n} log(\frac{1}{\sqrt{2\pi}} e^{\frac{-(y_i - x_i * w)^2}{2}})) + log(\frac{1}{\sqrt{2\pi \frac{1}{\lambda}}} e^{-\frac{w^2}{\frac{1}{\lambda}}})$$

$$= arg\,max_w (\sum_{i=1}^{n} -(y_i - x_i * w)^2) + \lambda(-w^2)$$

$$= arg\,min_w (\sum_{i=1}^{n} (y_i - x_i * w)^2) + \lambda(w^2) \tag{6}$$

This is the optimization equation for ridge regression. Just as MLE statistically justified the optimization equation for ordinary least squares, MAP justifies ridge regression (and other types of regression).

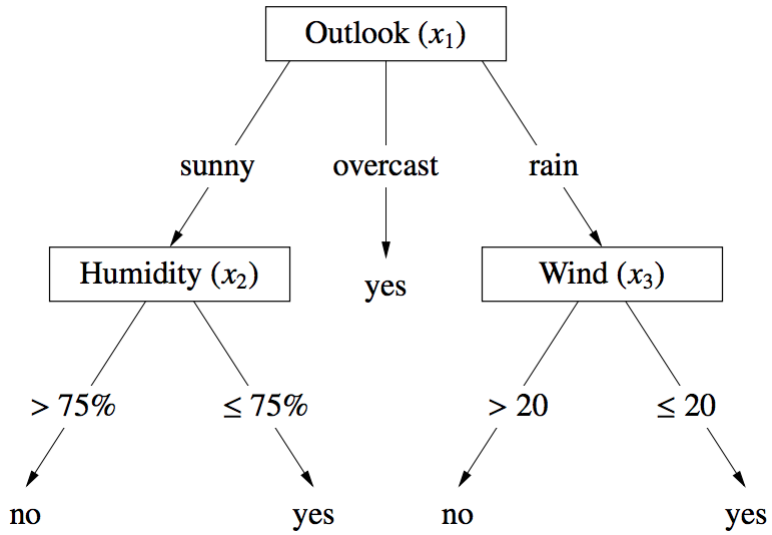
2 Decision Trees

Decision trees are a non-linear machine learning model that are popular because of their easy interpretability. You're probably used to using decision trees that have already been created. In this section, we will show you how to create an optimal decision tree by splitting based on the best features.

The general idea is that given a set of data that we have feature values and labels for, we solve for the features that we should split on in order to create the strongest model, where we will define the model's strength in this section.

2.1 Example

Here is an example of a decision tree for: "Whether or not a couple went on a picnic."



An example of a conclusion we could make from this image is that, given it was sunny and had $> 75\%$ humidity, our model would predict that the couple did go on the picnic.

2.2 Algorithm

Here is the general pseudocode for our algorithm:

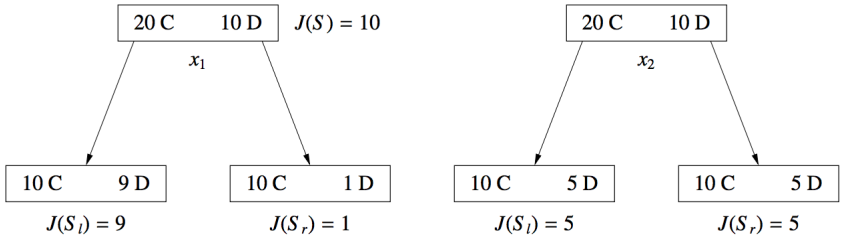
1. Find the feature that minimizes $J(dataleft) + J(dataright)$ where $J(d)$ is some cost function that we will discuss later and $dataleft$ and $dataright$ are the data points that belong to each side of the feature (e.g. whether it is $>$ or $<= 75\%$ humidity).
2. Repeat algorithm on the data that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

2.3 Cost function

Now we have to think of some cost function that gives us a powerful split. We can measure **information gain** as the difference between our cost function without the split and with the split.

2.3.1 A bad cost function

A naive idea would be to measure the features that get mislabeled at each split. Here is a pictorial reason that hopefully illuminates why this does not work:



Notice that our average information gain, $J(S) - (J(S_l) + J(S_r))/2$, on each of these splits are the same even though, intuitively, we clearly prefer x_1 as a split over x_2.

2.3.2 A good cost function

Instead, we'll introduce the idea of something called **entropy**. We'll first define surprise as $-log_2(Pr[Y = C])$. Notice here that if $Pr[Y = C] = 1$ then we have 0 surprise and that if $Pr[Y = C] = 0$ we have ∞ surprise.

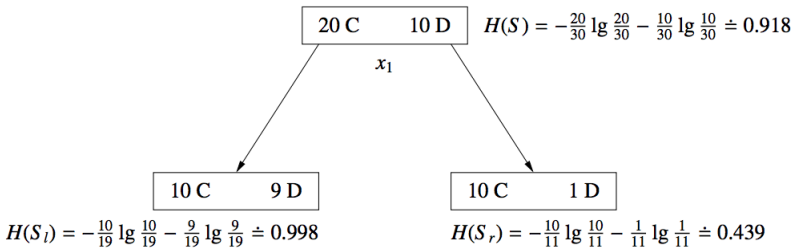
Now we can define entropy as

$$entropy = H(S) = -\sum_{C} p_C log_2(p_C) \tag{7}$$

where p_C = the number of items classified on this side with class label C divided by the total number of items classified on this side (e.g. on our previous example p_C on the left hand side would equal 10 / 19)

Now using this new split function, our decision tree with its respective entropy

values is:



We will continue this analysis in the next lecture.