Lecture 9: Classification: Naive Bayes, Logistic Regression
Modeling Social Data, Spring 2019
Columbia University

Calvin Tong

March 29th 2019

1 Introduction

In this class we talk about the classification problem and two general strategies for solving in, Naive Bayes and Logistical Regression. We end with a discussion of different metrics that help us optimize our classifier. Through out the lecture we will use the spam filtering problem as the main example. The problem is simply, given a set of emails how do you filter the spam emails from the ham (not spam) emails.

2 Setup and Initial Thoughts

We know that humans can classify spam fairly well, but we also know that they'll quickly get bored if they have to do it a million times. So the question becomes, "Can we teach a computer to classify spam for us?" A logical first step is to try to be formal about the heuristics we use when we classify spam. For example, we can look at the sender to see if we trust them or we can look at special keywords to see if they match our prior spam emails or we could look if we have received similar emails in the past and how we reacted to them. The issue with this approach is that it is extremely hard to write down a full set of rules and none of the rules apply universally.

So the brute force approach doesn't work that well, but what's a better approach to solving this? It turns out we can learn these rules from the data and make them fuzzy by introducing probability distributions. We will formalize this below.

3 Super Naive Bayes

Here we will develop a simple model that we will call Super Naive Bayes. We're being more naive that Naive Bayes, but the model will still work fairly well and the math will be simple and easy to implement.

3.1 Review: Bayes Rule

The main tool we will use for our first basic model is Bayes rule. We can derive this from equating the left and right hand sides of the product rule

$$p(x, y) = p(y|x)p(x) = p(x|y)p(y) \tag{1}$$

Rearranging we arrive at the infamous (and useful) Bayes rule

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)} \tag{2}$$

The tricky thing to remember is finding $p(x)$ involves summing over all instances of x given y. With this in hand we can begin building our model

3.2 Single Word Probabilities

The first thing we have to do is define the probability an email is spam if it contains a certain word. Applying Bayes rules, we see in math speak this is

$$p(\text{spam}|\text{word}) = \frac{p(\text{word}|\text{spam})p(\text{spam})}{p(\text{word})} \tag{3}$$

We can estimate these probabilities with ratios. This is super naive (hence the name), but super easy to implement. We can define more complicated ways of calculating these probabilities, but the math is the same. So let's define:

$$\hat{p}(\text{word}|\text{spam}) = \frac{\#\text{ spam docs containing word}}{\#\text{ spam docs}} \tag{4}$$

$$\hat{p}(\text{word}|\text{ham}) = \frac{\#\text{ ham docs containing word}}{\#\text{ ham docs}} \tag{5}$$

$$\hat{p}(\text{spam}) = \frac{\#\text{ spam docs}}{\#\text{ docs}} \tag{6}$$

$$\hat{p}(\text{ham}) = \frac{\#\text{ ham docs}}{\#\text{ docs}} \tag{7}$$

$$p(\text{word}) = p(\text{word}|\text{spam})p(\text{spam}) + p(\text{word}|\text{ham})p(\text{ham}) \tag{8}$$

Now we can just count up these quantities and we can easily find a the probability the email is spam or not if it contains a certain word.

3.3 Laplacian Smoothing

A big problem (among many) with this is that spammers can game this fairly easily. If they add a word that has never been in a spam message, like you company name, the classifier will predict a 0% probability of spam. Possible patch is to condition on a prior, which is called Laplacian smoothing. To do this, we introduce two constants α and β that "unzero" the probability and make us less confidant the words with no mentions in spam messages are not spam

$$\hat{p}(\text{word—spam}) = \frac{n_{word,spam} + \alpha}{N_{\text{spam}} + \beta} \tag{9}$$

How do we choose α and β? We use cross validation and a grid search of some kind. Remember if a classifier has no tuneable parameters, it's probably a bad classifier!

3.4 Full Email Probabilities

Okay now we have an easy way of calculating the probabilities for single words, but unfortunately most of the time emails have more than one word, so how the heck to do we deal with that? Answer: Math. But don't worry it's easy(ish) because we're going to be a bit naive. To start, we represent each document with a binary vector where $x_j = 1$ if the word appears in the document. Now we get to be naive: We will model the word occurrences in each email as independent. This simplifies the math greatly as we can write

$$p(\vec{x}|c) = \prod_j \theta_{jc}^{x_j}(1-\theta_{jc})^{1-x_j} \tag{10}$$

or the total probability of the email is just the product of the probabilities of the word in the document. Simple! To make this even easier to work with we're going to switch to a log space because doing so will make our products switch to sums and we like sums

$$\log p(\vec{x}|c) = \sum_j \log[\theta_{jc}^{x_j}(1-\theta_{jc})^{1-x_j}] \tag{11}$$

$$= \sum_j x_j \log \theta_{jc} + (1-x_j)\log(1-\theta_{jc}) \tag{12}$$

This formulation works, but in order to use it we'd have to sum over every word in the dictionary for every email. To avoid this we rewrite the equation as

$$\log p(\vec{x}|c) = \sum_j x_j \log \frac{\theta_{jc}}{1-\theta_{jc}} - \sum_j log(1-\theta_{jc}) \tag{13}$$

Then we see that the second term is simply a constant and the first term allows us to only sum over the words in the current document. Now applying this to the full bayes rule, again working the log space

$$\log p(c|\vec{x}) = \log \frac{p(\vec{x}|c)p(c)}{p(\vec{x})} \tag{14}$$

$$= \sum_j x_j \log \frac{\theta_{jc}}{1-\theta_{jc}} + \sum_j \log(1-\theta_{jc}) + \log\left(\frac{\theta_c}{p(\vec{x})}\right) \tag{15}$$

The final step is to switch to calculating the log odds in order to eliminate $p(\vec{x})$ leaving us with the final expression of

$$\log \frac{p(1|\vec{x})}{p(0|\vec{x})} = \sum_j x_j \log \frac{\theta_{j1}(1-\theta_{j0})}{\theta_{j0}(1-\theta_{j1})} + \sum_j \log \frac{1-\theta_{j1}}{1-\theta_{j0}} + log\frac{\theta_1}{\theta_0} \tag{16}$$

This looks really heinous, but it's really just a linear model of form $y = \vec{w_1}x + w_2$. The only thing we have to do is count the word occurrences in the training set, calculate w_1 and w_2 and we're done! Turns out this approach works really well and was used for many years. It's also easy to implement and computationally efficient.

4 Logistic Regression

Okay that was great, but our method for learning the weights was super naive. Can we improve that? Of course we can. Enter Logistic Regression, which we will now derive. From (16), we see we really have classifier form of

$$log\frac{p}{1-p} = \vec{w}\cdot\vec{x} \tag{17}$$

We can do a litle algebra and solve for p to get

$$p = \frac{1}{1+e^{-\vec{w}\cdot\vec{x}}} \tag{18}$$

Which is the logistic function. So the logical next step is to take a certain predictor and try to learn w from the data. Formally, we look at the probability of seeing a set of examples $(\vec{x_i}, y_i)$ where $y_i \in \{0,1\}$. We see

$$L = \prod_i p_i^{y_i}(1-p_i)^{1-y_i} \tag{19}$$

Again there's a product, so we move into a log space

$$\log(L) = \sum_i [y_i \log p_i + (1-y_i)\log(p_i)] \tag{20}$$

Now the game is to find the best w vector, which is just a simple maximum likelihood w.r.t w. First we rewrite out log likelihood in terms of w

$$\log(L) = \sum_i y_i(w \cdot x_i - \log(1 + e^{w \cdot x_i}) \tag{21}$$

Now taking the derivative and setting equal to 0

$$\frac{\partial \log(L)}{\partial w_j} = \sum_i \left[y_i x_{ij} - \frac{1}{1 + e^{w \cdot x_i}} e^{w \cdot x_i} \cdot x_{ij} \right] \tag{22}$$

$$0 = \sum_i (y_i - p_i) x_{ij} \tag{23}$$

It turns out we can't solve this directly directly, but we can use out old friend gradient decent. If we examine the equation we see that x_{ij} is just an indicator for if the word is included in the email, so this really amounts to making our prediction p_i as close as possible to our actual y_i.

So now we can learn $\vec{w}$. Woohoo! But what have we actually gained and lost here? Now we can share the weights between classes when they co-occur. This improves performance, but it's annoying to train and update. However, it turns out this trade off is worth it and now logistic regression has replaced naive bayes in many places.

5 In Practice

With some theory developed now we can talk about some helpful things when trying to fit these classifiers. Take a look at the R code for a more detailed explanation, but here are some highlights

5.1 Metrics

In practice, the optimal classifier is dependant on the application. In order to better tune our parameters, we introduce a few quantities will help us get a better sense of how we are doing. Let's introduce the following

1. Accuracy: fraction of correct classifications
2. Precision: fraction of positive predictions that are true
3. Recall: the fraction of true examples that we predicted to be positive
4. False Positive Rate: The fraction of false examples predicted to be true

Clearly, there are going to be trade off between these things. For example, high precision will sacrifices recall. When evaluating a classifier one should decide what they want to optimize for and look at all of these metric together.

5.2 ROC and AOC curves

Another tool for tuning classifiers are the ROC and AOC curves. To understand this we define the True Positive Rate (TPR) and False Positive Rate (FPR)

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}} \text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}} \tag{24}$$

The ROC curve is the plot of TPR vs. FPR for different classification thresholds. The AUC curve is the area under the ROC curve and provides a measure aggregate performance across continuous classification thresholds (of course dependant on the resolution of the ROC curve). These tools are extremely useful for choosing an optimal classification threshold.