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

March 29, 2019

Notes from ask2256

1 Learning by Example

How did we (as humans) resolve spam vs. real emails when cleaning up an inbox?

- Key-word searches
- Look for email contacts, but maybe there are necessary unknown contacts
- Use what is already in the inbox
- Did the email go out to a large group of people, BUT lots of email lists are legit

We have a lot of options, but they are weakly indicative of what is or is not spam. How do we make a direct rule out of this? We could use a bunch of if-then statements, then that could get really complicated because each statement has to be weighted differently. This is the idea of building a **classifier**. We learn the rules from the data.

2 Exercise: Diagnoses a la Bayes

Suppose we are testing for a rare disease. 1% of the population is infected, and we have a highly sensitive and specific test that yields 99% of sick patients test positive, 99% of healthy patients test negative. Given a patient tests positive, what is the probability the patient is sick?

3 Natural frequencies a la Gigerenzer

Claim: early screening of breast cancer lowers chance of breast cancer by 20%. What does this mean? Imagine there are a thousand women who do not get screened and a thousand women who do get screened. 5 women who did not get screening died of breast cancer. 4 women who did get screening died of breast cancer, so we are comparing $\frac{5}{1000}$ and $\frac{4}{1000}$. Also note that the number of women who died from all types of cancer is equal (21 overall).

If a woman does not get screening, then there is no cost. If you do get screening there is a different cost (what if it is a false alarm?). After perhaps surgery a lot of women learned they didn't have breast cancer. If the test is not reliable enough, it can cause a lot of false positives and unnecessarily scare people. Note that the guidelines of early screening have changed. Genetic markers and other indicators means screening is recommended.

4 Inverting conditional probabilities

Let us start with the **product rule**

$$p(y|x)p(x) = p(x,y) \qquad\qquad p(x|y)p(y) = p(x,y).$$

Equating both sides we get

$$p(y|x)p(x) = p(x,y) = p(x|y)p(y).$$

If we divide both sides by $p(x)$, we get

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

which is **Bayes' Theorem**.
Bayes' Theorem states

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

where $p(x) = \sum_{y \in \Omega_y} p(x|y)p(y)$ is the normalization constant.
There is a truth and then there is an indicator (positive or negative)

5 (Super) Naive Bayes

In the email, we define 'truth' as whether or not an email is spam. We use indicators tell us which is which. Bayes' Rule allows us to do this. Using the branches from previously, we let $p(y)$ denote the 'positive' branches.

Going back to the suggestion of looking at different emails and see how often different words occur. Bayes' rule let's us do that. Look among all the emails that are spam and search for a key word. This is Super Native Bayes, where we classify the emails based on one word. We take all the spam documents containing the keyword and divide by all the spam docs. We need the probability of seeing the words among both spam and non-spam documents. Note, we call non-spam 'ham'. Let h denote ham, s denote spam, and w denote all the words. So we have:

$$p(w) = p(w|s)p(s) + p(w|h)p(h)$$

Note: we care about how often does the word occurs overall, not just how many documents it appears in. Even though we only care about spam, we have to look in both spam and ham. We ask the question: what are the sizes of that term relative to the other sizes?

We can also use shell to do this by looking at released Enron employee emails to build our script, which does everything from get the data to build the classifier.

```
$ ./enron_naive_bayes.sh meeting
1500 spam examples
3672 ham examples
16 spam examples containing meeting
153 ham examples containing meeting

estimated P(spam) = .2900
estimated P(ham) = .7100
estimated P(meeting|spam) = .0106
estimated P(meeting|ham) = .0416

P(spam|meeting) = .0923
```

Let's say we want to know how often the word 'money' appears, we can use grep. We get multiple responses per files, we just want to count the matching files. So we retrieve a list of all the files and the use wc $-$ l to count them.

```
$ ./enron_naive_bayes.sh enron
1500 spam examples
3672 ham examples
0 spam examples containing enron
1478 ham examples containing enron

estimated P(spam) = .2900
estimated P(ham) = .7100
estimated P(enron|spam) = 0
estimated P(enron|ham) = .4025

P(spam|enron) = 0
```

Notice that the word ENRON does not appear in the spam classifier. If the spammers just added this word, they would be able to get through. So do we keep it?

We could say

$$\hat{p}(w|s) = \frac{n_{w,s} + \alpha}{N_s + \beta}$$

, where $\alpha = 1$ and $\beta = 1000$.

How do we actually combine this process for a lot of words? That brings us to **Naive Bayes**.

6 Naive Bayes

We represent each document by a binary vector and each word is independent. That means that if we see the word MONEY, the probability of seeing the word FREE has nothing to do with seeing the word MONEY.

Suppose there are $100,000$ words. We ascribe a 'coin' to them and then flip that coin and record the ones that turn up 1 or 0. We can think of it as follows:

X is a big matrix where each row is composed of 1 or 0 to represent if a word occurs or not

$$X = \begin{bmatrix} 0 & 1 & \cdots & 1 & 0 \\ 1 & 0 & \cdots & 1 & 1 \\ \vdots & \vdots & \ddots & \vdots & \\ 1 & 1 & \cdots & 0 & 1 \\ 0 & 1 & \cdots & 0 & 0 \end{bmatrix}$$

Now we want to apply **Bayes' Rule**. Let $\vec{x}$ represent each row, x_j each column. θ_c represents the probability of observing a document of class c.

$$p(\vec{x}|c) = \Pi_j \theta_j^{x_j c} e^j (1 - \theta_{jc}^{1-x_j c}.$$

Let us take the $\log$ of both sides so we can solve for $p(\vec{x}|c)$.

$$\log(p(\vec{x})|c) = \sum_j log \left[\theta_j^{x_j c} (1 - \theta_{jc})^{1-x_{jc}} \right] = \sum_j \left[x_j \log \theta_{jc} + (1 - x_j) log(1 - \theta_{jc}) \right]$$

Aside Think about why are we adding a million floats if the documents only has 10 words.

Let's derive the results from the slides:

$$\log(p(\vec{x}|c) = \sum_j \left[x_j \log \theta_{jc} + (1 - x_j) log(1 - \theta_{jc}) \right]$$

and group like terms together:

$$\log(p(x|c) = \sum_j x_j \log \frac{\theta_{jc}}{1 - \theta_{jc}} + \sum_j \log(1 - \theta_{jc}).$$

The first summation represents one term for each non-zero word. The second sum term represents the constant. This is where we get the result seen in the slides.

Notice,

$$\sum_j x_j w_j + w_o$$

which is just

$$\log p \ \vec{w}\vec{x} + \text{intercept},$$

where the intercept is just x_0. Notice we have just a linear model.

We can eliminate $p(\vec{x})$ by calculating the log-odds, which gives us a linear classifier of the above form:

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

where $w_j = \log \frac{\theta_{j1}(1-\theta_{j0}}{\theta_{j0}(1-\theta_{j1}}$ and $w_0 = \sum_j \log \frac{1-\theta_{j1}}{1-\theta_{j0}} + \log \frac{\theta_1}{\theta_0}$. This gives us our linear classifier of the form $\vec{w}\dot{\vec{x}} + w_0$.

Basically, we are using Bayes rule to get to a linear model. $y = mx + b$

We train by counting words and documents within classes to estimate θ_{jc} and θ_c:

$$\hat{\theta}_{jc} = \frac{n_{jc}}{n_c}$$

$$\hat{\theta} = \frac{n_c}{n},$$

which gives us

$$\hat{w}_j = \log \frac{\hat{\theta}_j(1-\hat{\theta}_{j0}}{\hat{\theta}_{j0}(1-\hat{\theta_{j1}}}$$

$$\hat{w}_0 = \sum_j \log \frac{1-\hat{\theta}_{j1}}{1-\hat{\theta}_{j0}} + \log \frac{\hat{\theta}_1}{\hat{\theta}_0}$$

This means we predict by adding the weights of the words that appear int he document to the bias term.

Aside Idiot's Bayes–Not so stupid after all? (https://pdfs.semanticscholar.org/bd6a/9d35dabbba8132f48835f636cf7c3b3e9c80.p

Training is computationally cheap and scalable, and the model is easy to update given new observations. If we get new data, and we have trained the model based on previous data, we just have to update the counts. Performance varies with document representations corresponding to likelihood models. The big thing is that when we are estimating the θ's, we have to be careful to cross-validate over the parameters α and β.

It is often important to smooth out these parameter estimates to avoid over-fitting as follows:

$$\hat{\theta}_{jc} = \frac{n_{jc}+\alpha}{n_c+\alpha+\beta}.$$

Going back to log-odds calculations, let us take

$$\log \frac{p(s|\vec{x}}{p(h|\vec{x})} = \vec{w}\dot{\vec{x}}$$

$$\log \frac{p}{1-p} = \vec{w}\dot{\vec{x}}.$$

This is a form of a classifier.

How do we estimate the w's?

$$\frac{p}{1-p} = e^{\vec{w}\dot{\vec{x}}}$$

$$p = (1-p)e^{w\dot{x}}$$

$$p(1+e^{w\dot{x}}) = e^{w\dot{x}}$$

$$p = \frac{e^{w\dot{x}}}{1+e^{w\dot{x}}}e^{-w}$$

x$\frac{}{e^{-w\dot{x}}=\frac{1}{1+e^{-w}}}$.

When we plot we get something that looks like this:

INSERT DRAWING

As the function becomes increasingly negative, the likelihood of an email being spam decreases.

Look at the probabilities of seeing a set of examples (x_i, y_i), where y_i is $\{0, 1\}$.

The product of those probabilities is as follows:

$$L = \Pi_i p_i^{y_i}(1-p_i)^{1-y_i},$$

where $p_i = p(x_i|\vec{w})$.

so we get

$$\mathcal{L} = \log L = \sum_i \left[y_i \log p_i + (1 - y_i) \log(1 - p_i) \right]$$

The best $\vec{w}$ is found as:

$$argmax_{\vec{w}}(p(D|\vec{w}) \log \frac{1}{1 + e^{-wx_i}} = -\log(1 + e^{\vec{w}\dot{x}})$$

Then putting this into our previous statement

$$\mathcal{L} = \sum_j \left[y_i \log \frac{p_i}{1 - p_i} + \log(1 - p_i) \right],$$

where $\vec{w}\dot{x}_i = \log \frac{p_i}{1-p_i}$.
It follows,

$$\mathcal{L} = y_i(w\dot{x}_i) - \log(1 + e^{wx_i}).$$

Differentiating both sides:

$$\frac{d\mathcal{L}}{dw_j} = \sum_j \left[y_i x_{ij} - \frac{1}{1 + e^{wx_i}} e^{w\dot{x}_i} x_{ij} \right]$$

where $p_i = \frac{1}{1+e^{w\dot{x}_i}} e^{w\dot{x}_i} x_{ij}$.
It follows

$$0 = \sum_j (y_i - p_i) x_{ij}$$

We can't invert these matrices because we are not dealing with a linear system.

This is actually a **Gradient Descent Method**. We can think of y_i as the actual value and p_i as the predicted, and then scale it by the weight x_{ij}.

In the i^{th} document if the j^{th} word appears, and we are wrong we either increase or decrease our prediction. Gradient descent is just guess , check update. too high? lower guess. Too low? higher guess. All of the stuff we had before for linear regression follows through, but we are dealing with non linear prediction, not linear. The purpose is to minimize the error ϵ. We can't minimize this error in stone step (e.g. it's not quadratic), so the surface actually looks quite weird. We are dealing with non-linear Euclidean geometry.

Why is this different from Naive Bayes? The predictor has the same form. The difference is how we estimate w_j. In this case, we are using all the w_j's, then update them to correct the prediction. If things do co=occur, we can learn to share the weight.

With **Naive Bayes**, we just count one word. In this one, we are actually, we are taking the derivative with respect to w, but it's not separate. When we make the prediction, we are using all the w's. This involves making the prediction, so we are using all the w's and updating. That's how it learns to share between the weights.

7 Linear Regression Models

Let us start with some coding. Going back to our model idea, let us think of each row as a word, each column as a probability of being spam or ham?

We can fit a **Naive Bayes** classifier in R. This gives us the number of words, the class, and what type of character we're dealing with. R contains native Naive Bayes functions, as follows:

```
model <- naiveBayes(xTrain, yTrain)
summary(model)
```

```
          Length Class  Mode
apriori    2     table  numeric
tables    57     -none- list
levels     2     -none- character
isnumeric 57     -none- logical
call       3     -none- call
```

Then we construct the confusion matrix. The confusion matrix gives a summary of the classifiers performance, with the actual label determining the row, ad the predicted label giving the column. Here is how we do it in R:

```
table(df)
```

```
        pred
actual   email spam
  email    139  116
  spam      12  194
```

which gives us the following table:

```
df <- data.frame(actual = yTest,
                 pred = predict(model, xTest))
head(df)
```

actual	pred
spam	spam
spam	spam
spam	spam
spam	spam
spam	spam
spam	spam

When we predict and it is spam, we get a **True Positive**. If something is predicted as spam and actually not spam, this is a **False Positive** (**Type I Error**). If something is predicted as not spam and ends up being spam, we get a **False Negative** (**Type II Error**). If something is predicted as not spam and ends up not being spam actually, it is a **True Negative**. This is gives us the concept of **accuracy**

Precision is the fraction of predicted spam that is actually spam.

So this would be true positive over true positive plus false positives. Basically, we want true positives over everything that is predicted as being spam.

We could optimize the w's with precision, but that would be very hard. Instead, we can perform the analysis, then adjust for precision.

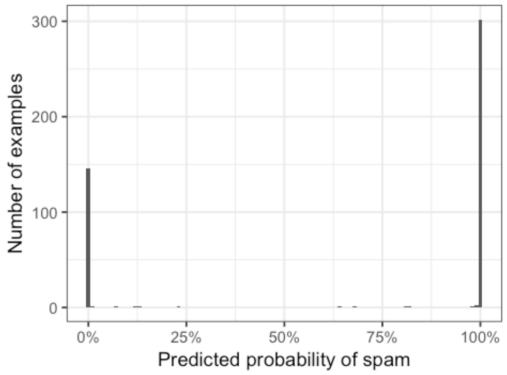
Recall: the fraction of all spam that is predicted to be spam is taken as the true positive over false negatives. We are focusing on what was predicted as spam and actually spam over what is predicted as not being spam but ends up being spam. That would be

$$\frac{\text{true positive}}{\text{false negative}}.$$

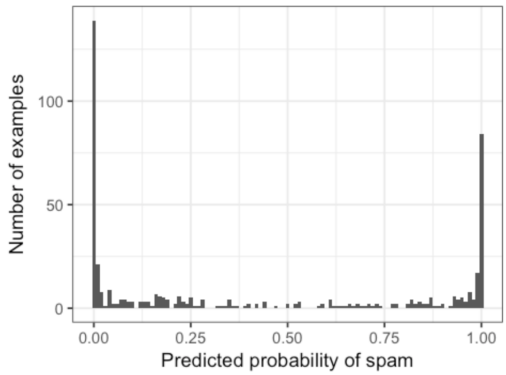
If we want a high recall, then precision increases.

The false positive rate is the fraction of legitimate email that is predicted to be spam. So we take the false positive (predicted as spam ends up not being spam) and divide by true negatives.

There are packages built into R that let us calculate all of these rates. The biggest problem with the Native Bayes classifier is overconfidence. But maybe it's not? Maybe everything in the right curve is actually spam in this plot:



Let's do this on a fractional basis. Then we get the below plot



Of the things we were certain were spam, only 75% of them turned out to be spam, of the things that we were certain were not spam, about 5% turned out to not be spam. But this plot does not look very good.

We can fit a linear regression model by taking the data and saying spam is a function of all the words. We use GLM to do that. We tell it that the outcome is 'binomial' (the $\vec{w}\vec{x}$ corresponds to a coin flip).

Next, we fit a logistic regression model using the following code.

```
model <- glm(spam ~ ., data=spam[ndx, ], family="binomial")
model
#summary(model)
```

which yields the following results

```
Coefficients:
(Intercept)          A.1          A.2          A.3          A.4          A.5
 -1.641e+00   -3.008e-01   -1.854e-01    1.518e-01    2.867e+00    5.747e-01
        A.6          A.7          A.8          A.9         A.10         A.11
  8.963e-01    2.172e+00    5.103e-01    1.093e+00    2.304e-01   -2.654e-01
       A.12         A.13         A.14         A.15         A.16         A.17
 -1.801e-01    2.335e-02    1.361e-01    1.034e+00    1.089e+00    1.081e+00
       A.18         A.19         A.20         A.21         A.22         A.23
  1.027e-01    5.755e-02    8.663e-01    2.544e-01    2.620e-01    2.128e+00
       A.24         A.25         A.26         A.27         A.28         A.29
  7.238e-01   -1.863e+00   -8.376e-01   -1.156e+01    3.950e-01   -3.891e+00
       A.30         A.31         A.32         A.33         A.34         A.35
 -4.095e-01   -1.832e-01    2.639e+00   -9.168e-01    1.450e+00   -1.835e+00
       A.36         A.37         A.38         A.39         A.40         A.41
  1.043e+00    1.744e-01   -6.226e-01   -1.018e+00   -3.384e-01   -4.235e+01
       A.42         A.43         A.44         A.45         A.46         A.47
 -2.633e+00   -2.373e+00   -1.345e+00   -8.769e-01   -1.462e+00   -2.263e+00
       A.48         A.49         A.50         A.51         A.52         A.53
 -4.540e+00   -1.417e+00   -2.375e-01   -4.739e-01    3.366e-01    5.838e+00
       A.54         A.55         A.56         A.57
  2.481e+00    6.462e-03    9.955e-03    7.337e-04

Degrees of Freedom: 4139 Total (i.e. Null);  4082 Residual
Null Deviance:      5530
Residual Deviance: 1594         AIC: 1710
```

Coefficients greater than 0 indicate spam. Coefficients less than 0 indicate ham. When we compute all the summary stats, we notice that the False Positive Rate has been greatly reduced from approximately 40% to 5%! Now 5% of regular emails end up in the spam box. This is much better!

Notes from cyt2113

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.

Notes from jm4303

1 Learning by example

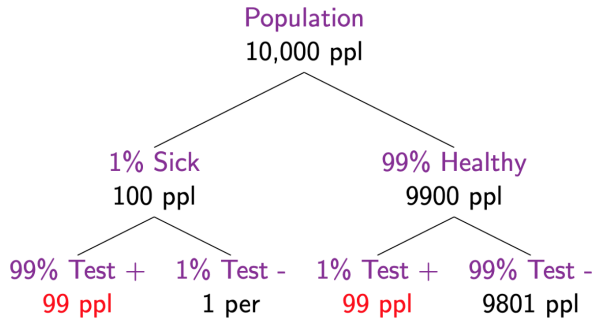
Spam vs Ham

- We can separate spam from non-spam emails by looking at various indicative things
- For example, words such as 'special offer' are highly indicative of spam, whereas words like 'supercomputing cluster' are much less indicative.
- A potential problem is with ubiquitous words like 'the'.
- It's hard to write down all rules, so we learn them from data instead.

2 Diagnoses a la Bayes

- You're testing for a rare disease:
  - 1% of the population is infected
- You have a highly sensitive and specific test:
  - 99% of sick patients test positive
  - 99% of healthy patients test negative
- Given that a patient tests positive, what is probability the patient is sick?

2.1 Method 1



So given that a patient tests positive (198 ppl), there is a 50% chance the patient is sick (99 ppl)!

2.2 Method 2

We know from the given that:

$$P(sick) = 0.01$$
$$P(healthy) = 0.99$$
$$P(+|sick) = 0.99$$
$$P(+|healthy) = 0.01$$

According to Bayes' Theorem (proven later):

$$P(sick|+) = \frac{P(+|sick)P(sick)}{P(+)} = \frac{P(+|sick)P(sick)}{P(sick)P(+|sick) + P(healthy)P(+|healthy)} = \frac{(0.99)(0.01)}{(0.01)(0.99) + (0.99)(0.01)} = \frac{1}{2}$$

3 Natural Frequencies a la Gigenrenzer

- Compared to the 1000 women who didn't have screening, the 1000 women with screening suffered 100 cases of false-positive negative results and 5 cases of unnecessary treatments.
- Thus, although the claim is that screening reduces the number of patients who died from breast cancer by 20% (5 deaths vs 4 deaths), there's a risk present for error for women with screening that is not present for those without.

4 Inverting Conditional Probabilities

Bayes' Theorem

We know that

$$P(x, y) = P(x|y)P(y) = P(y|x)P(x)$$

because $P(x, y) = P(y, x)$

Divide to get the probability of y given x from the probability of x given y:

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)}$$

where $P(x) = \sum_{y \in \Omega_Y} P(x|y)P(y)$

5 (Super) Naive Bayes

Idea: use Bayes' Rule to build a one-word spam classifier:

$$P(spam|word) = \frac{P(word|spam)P(spam)}{P(word)}$$

Estimate each term with ratio of counts:

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

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

$$\hat{P}(spam) = \frac{\#\text{ spam docs}}{\#\text{ docs}}$$

$$\hat{P}(ham) = \frac{\#\text{ ham docs}}{\#\text{ docs}}$$

Examples from running bash script:

```
$ ./enron_naive_bayes.sh money
1500 spam examples
3672 ham examples
194 spam examples containing money
50 ham examples containing money

estimated P(spam) = .2900
estimated P(ham) = .7100
estimated P(money|spam) = .1293
estimated P(money|ham) = .0136

P(spam|money) = .7957
```

```
$ ./enron_naive_bayes.sh enron
1500 spam examples
3672 ham examples
0 spam examples containing enron
1478 ham examples containing enron

estimated P(spam) = .2900
estimated P(ham) = .7100
estimated P(enron|spam) = 0
estimated P(enron|ham) = .4025

P(spam|enron) = 0
```

Note that the probability of an email with the word 'enron' being classified as spam is 0 because none of the spam examples has 'enron'. This is probably not what we want.

A possible way around this is:

$$\hat{P}(word|spam) = \frac{n_{word,spam} + \alpha}{N_{spam} + \beta}$$

where the best α, β combination on the test data can be found through grid search.

6 Naive Bayes

- 'Naive' in the sense that all words in a document are seen as independent given the class label of the document.
- Represent each document by a binary vector $\vec{x}$ where $x_j = 1$ if the j-th word appears in the document ($x_j = 0$ otherwise).

- If we model each words as *independent* Bernoulli random variable, the probability of observing document $\vec{x}$ of class c is:

$$P(\vec{x}|c) = \prod_i \theta_{jc}^{x_j}(1-\theta_{jc})^{1-x_j}$$

where θ_{jc} is the probability that the j-th word occurs in a document of class c.

- Using this likelihood in Bayes' Rule and taking the logarithm, we have:

$$logP(c|\vec{x}) = log\frac{P(\vec{x}|c)P(c)}{P(\vec{x})} = \sum_j x_j log(\frac{\theta_{jc}}{1-\theta_{jc}}) + \sum_j log(1-\theta_{jc}) + log\frac{\theta_c}{P(\vec{x})}$$

where θ_j is the probability of observing a document of class c and $logP(\vec{x}|c)$ is calculated as follows:

$$logP(\vec{x}|c) = \sum_j log[\theta_{jcj}^{x}(1-\theta_{jc})^{1-x_j}] = \sum_j x_j log\theta_{jc} + (1-x_j)log(1-\theta_{jc}) = \sum_j x_j log(\frac{\theta_{jc}}{1-\theta_{jc}}) + \sum_j log(1-\theta_{jc})$$

Note that the second term $\sum_j log(1-\theta_{jc}$ is a constant because it doesn't depend on $\vec{x}$. It can be interpreted as the base rate of class c for an empty document. Recall that θ_{jc} is the probability that the j-th word occurs in a document of class c.

We can remove $P(\vec{x})$ by calculating the log-odds:

$$log\frac{P(1|\vec{x})}{P(0|\vec{x})} = \sum_j x_j \underbrace{log\frac{\theta_{j1}(1-\theta_{j0})}{\theta_{j0}(1-\theta_{j1})}}$$

Form of classifier:

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

Solve for p:

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

$$p = \frac{e^{\vec{w}\cdot\vec{x}}}{1+e^{\vec{w}\cdot\vec{x}}}\cdot\frac{e^{-\vec{w}\cdot\vec{x}}}{e^{-\vec{w}\cdot\vec{x}}} = \frac{1}{1+e^{-\vec{w}\cdot\vec{x}}} = P(\vec{x}|\vec{w})$$

The probability of seeing data $(\vec{x_i}, y_i)$ is:

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

Note that $y_i \in 0, 1$ and $p_i = P(\vec{x_i}|\vec{w})$.

Take the log of L:

$$\ell = logL = \sum_i y_i log(p_i) + (1-y_i)log(1-p_i) = \sum_i y_i log\frac{p_i}{1-p_i} + log(1-p_i) = \sum_i y_i(\vec{w}\cdot\vec{x_i}) - log(1+e^{\vec{x}\cdot\vec{x_i}})$$

Note that $log(1-p_i) = log(1-\frac{1}{1+e^{-\vec{w}\cdot\vec{x}}}) = log(\frac{1}{1+e^{\vec{w}\cdot\vec{x}}}) = -log(1+e^{\vec{w}\cdot\vec{x}})$.

We want to find the $\vec{w}$ that gives the highest likelihood to the data that we have i.e. the $\vec{w}$ that maximizes $P(D|\vec{w})$. This is equivalent to finding a $\vec{w}$ that maximizes $logP(D|\vec{w})$. Note that this log probability is ℓ, what we found previously. So we take the derivative of ℓ with respect to $\vec{w}$:

$$\frac{d\ell}{dw_j} = \sum_j y_i x_{ij} - \frac{1}{1+e^{\vec{w}\cdot\vec{x}_i}} e^{\vec{w}\cdot\vec{x}_i} x_{ij} = \sum_j y_i x_{ij} - p_i x_{ij} = \sum_j (y_i - p_i) x_{ij}$$

note that $p_i = \frac{1}{1+e^{\vec{w}\cdot\vec{x}_i}} e^{\vec{w}\cdot\vec{x}_i}$ (see previous calculation).
We use this gradient in gradient descent to find the best $\vec{w}$.

- Naive Bayes gives extreme estimates because it doesn't take the weight of other words into account.
- Logistic Regression, however, allows for the sharing of weights because p_i is calculated with $\vec{w}$.