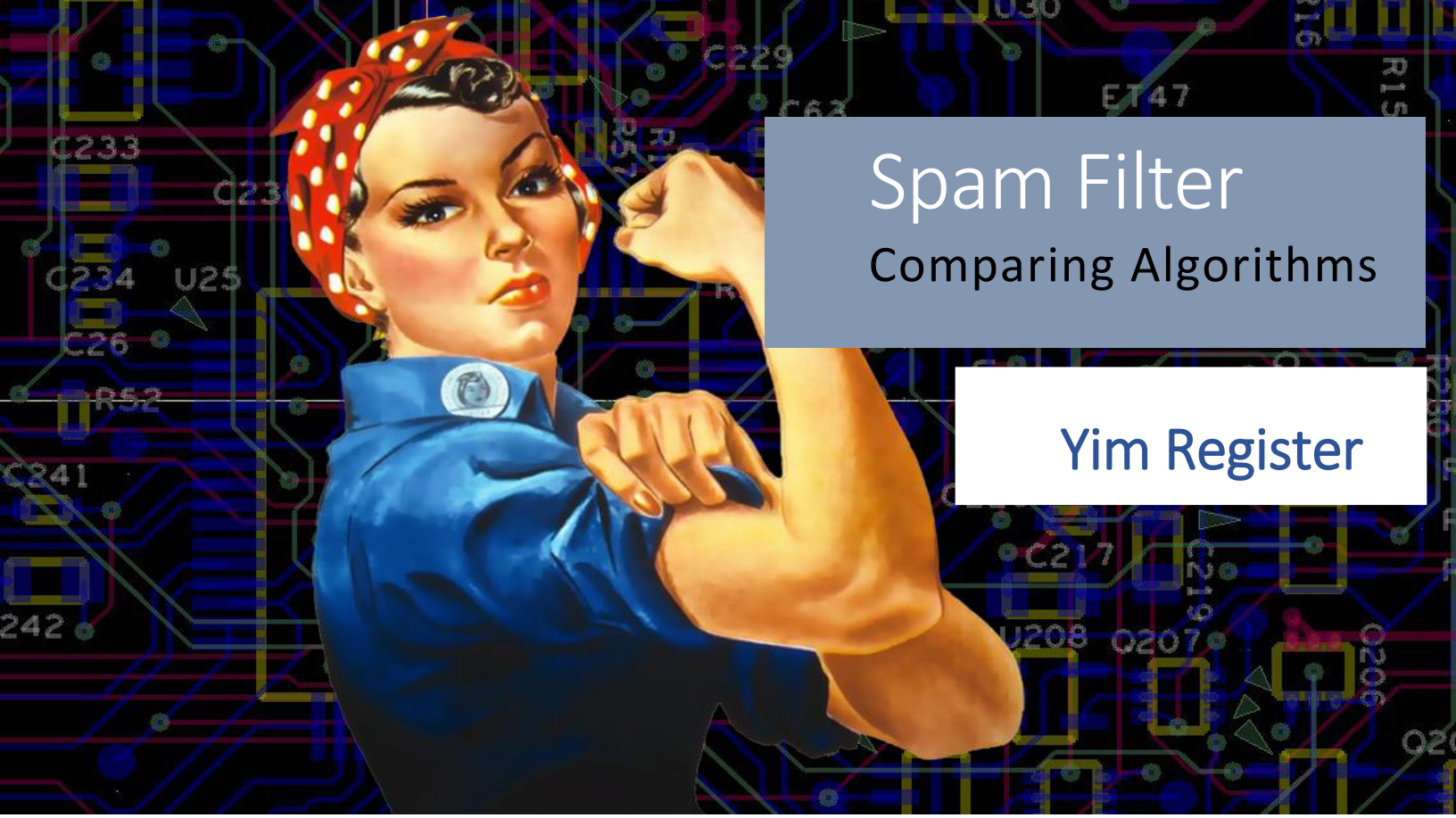


Spam Filter

Comparing Algorithms

Yim Register

01

My Data

02

Logistic Regression

03

Naïve Bayes

Different algorithms have different strengths

In this report I explore the strengths, weaknesses, and performance of two different algorithms on a spam filtering task on my own emails.

My Data

Setting up how we read the data is a critical part of any model building. We walk through all the choices I made for **formatting the data**; an important transparency step that should be included when we talk about machine learning.

Logistic Regression

We begin with **logistic regression**, perhaps the most immediate algorithm to come to mind for 0/1 classification.

Naïve Bayes

In my experience, **Naïve Bayes** is the common sense algorithm for spam filtering, and we take a detailed look at it's nuances, strengths, weaknesses, assumptions, and take a small detour into **Information Theory** to explain the value of conceptualizing entropy to better understand predictive models.

My Data

For any dataset used in machine learning, we must make some choices in how we read in, format, process, and manipulate the data. As the data scientist in charge here, I've already noticed patterns in my data that I want my model to pick up on. For instance, my spam emails contain a lot of links! But if I were to read each link in exactly as written, they would *all* be different. My theory as the data scientist is that the content of the links matters, but the *presence* of links at all is more indicative that a message will be spam. So I have replaced links with the **string** "featurlink". This way, my algorithms will be able to regard any urls as a common feature that can be aggregated together. I've used a bit of a funky word because I also employ a word stemmer (Porter Stemmer), and I didn't want the word "link" to get stored with instances of the actual word "link". The Porter Stemmer transforms all text into lowercase letters, and allows "eat" to be the same as "ate" or "eating". When working with text, this is usually the right way to go. Here is a summary of the choices I made:

Porter Stemmer

- Words are reduced to their stems, making 'eat' the same as 'ate','eating', and 'eats'. This is to get at the actual content of the message, as opposed to artifacts of natural language.

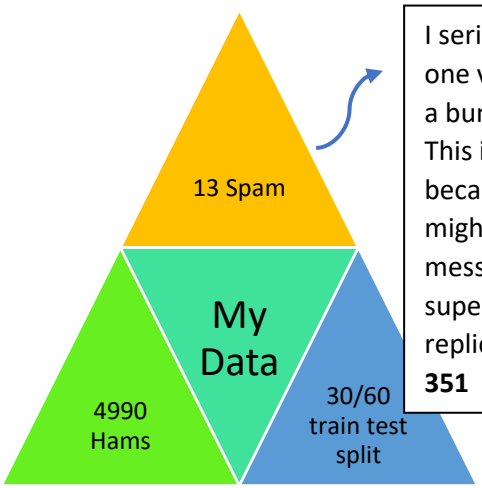
Links

- Links are replaced with the string 'featurlink' so that they are all regarded for what they *are* and not for the specifics of the link address (all regarded the same)

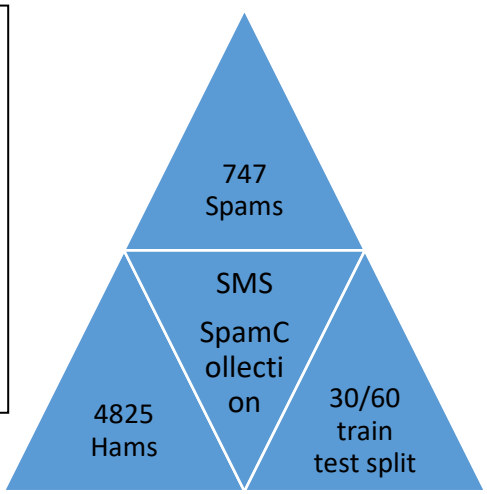
Extraneous Characters

- Some spam filters may choose to ignore common characters in English text, but I actually chose to keep punctuation. I think there actually could be a difference in things like dollar signs ($), exclamation points (!) and other symbols when it comes to distinguishing between Spam or Not Spam

Unfortunately, I had to explore several of my own email archives, because I simply did not have enough spam! For my account, I only had 13 spam messages! Through this project I've really come to see the value in "Big" Data. I tried the algorithms on my own data, and include it in the writeup, but I had to validate my work on an official Spam dataset from the web, which I have also included.



I seriously only had **13** spams. In one version, I cheat by replicating a bunch of the spam messages. This is a terrible way to do things, because then my train/test data might be the exact same message. Performance inflates superficially when I artificially replicate spam messages to **size = 351**



Yim Register

Logistic Regression

How it Works

Since we have two (binary) categories for this problem, we might immediately think of logistic regression. We know that linear regression can help us combine predictor values to provide a numerical outcome, but logistic regression is the perfect opportunity to **classify** things into "Yes" or "No". In this case, it's "Spam" or "Not Spam". The big conceptual perk of logistic regression is that it smooths the boundary between "Yes" and "No". Data that looks like Figure 1 isn't too helpful for prediction, due to that sharp boundary. Instead, we want a smooth curve of the probability that something will be "Yes" or "No" based on its features.

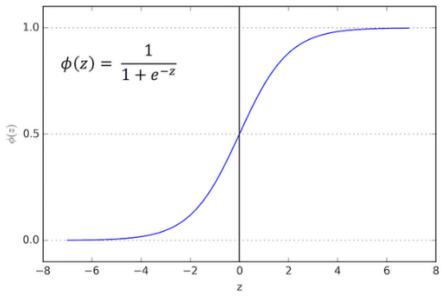


Figure 1 Activation function for logistic regression. Instead of a harsh 'Yes' or 'No', we use a smooth boundary with a threshold.

You might ask, why do we need it to be smooth? Especially if we are making a "Spam" or "Not Spam" judgment? Do we really want a category that is *almost* spam? Well, no. We don't want a separate category for almost spam, but let's walk through an example to understand why we might want a smoother boundary for "Yes" and "No" in this example. Here's an email I *actually* sent out the other day, trying to recruit undergraduates to a research study: *Participate in a research study! $20 gift card!* It was entirely in earnest, but after I sent it, I realized it might be chucked into a Spam bucket because it sounds so desperate. The presence of a dollar sign, exclamation points, the call for participation... it all seems like it might fit nicely with what we typically classify as "Spam". Instead of a hard boundary, I might want the system to flag this email as *possible* spam as opposed to *definitely* spam, because there are other features about it that indicate it is "Not Spam" (like the mention of a research study, or the fact that it's coming from a uw.edu email address). In order to come up with a model that can classify new emails as "Spam" or "Not Spam", our task is to discover the proper **weights** on the **features** that are contributing to something being labeled as spam or not in our **training set**. You might already even be able to think of a few features that would indicate "Spam"! Just thinking of your own emails, you've learned that features like "free", "win", "iPhone", "vacation", "$", "Congratulations!" often indicate that the message is spam. Our task is to discover just how much each of those kinds of words play a role in signaling that a message is spam or not, and to create a model that can help us automatically classify *new* messages based on what we learned in training. This reasoning is common to the following algorithm (**Naïve Bayes**) as well, so I'll spend less time explaining this kind of overview in the following sections.

Performance

My Data

Accuracy: 99.7% **(MISLEADING! It labels everything 'ham')**
Spam detection: 0%

•	precision	recall	f1-score	support
•ham	1.00	1.00	1.00	1248
• spam	0.00	0.00	0.00	3
•avg / total	1.00	1.00	1.00	1251

SMS Collections Data

Accuracy 97%
Spam detection: 80%

•	precision	recall	f1-score	support
•ham	0.97	1.00	0.98	1203
•spam	0.99	0.78	0.87	190
•avg / total	0.97	0.97	0.97	1393

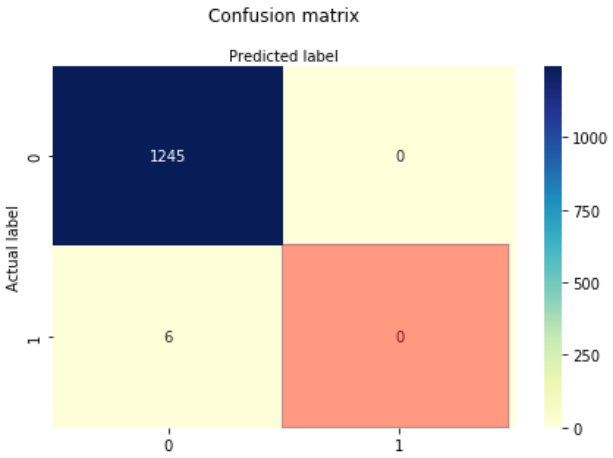


Figure 3: Literally my logistic model got ZERO correct for the spams. It predicted everything as 'ham' because it didn't have enough data.

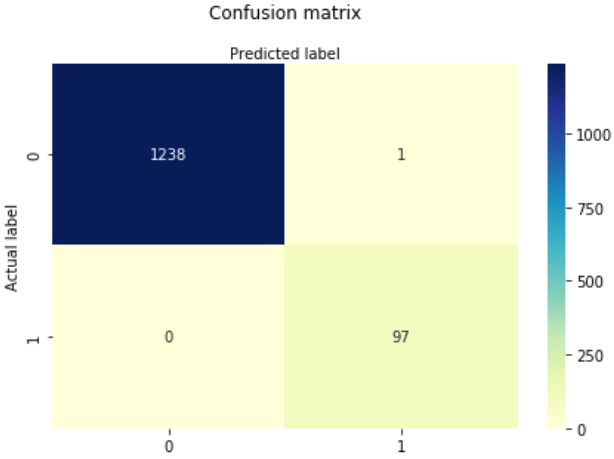
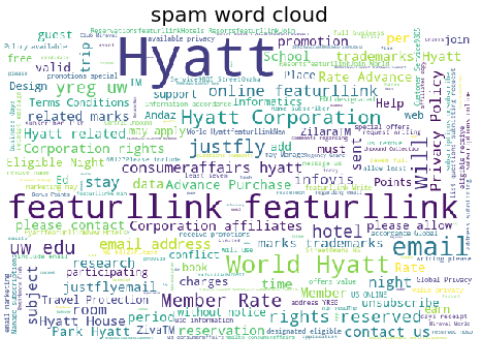


Figure 2: This is from the well-known dataset SMS Spam Collections just to show that my code would work on better data.



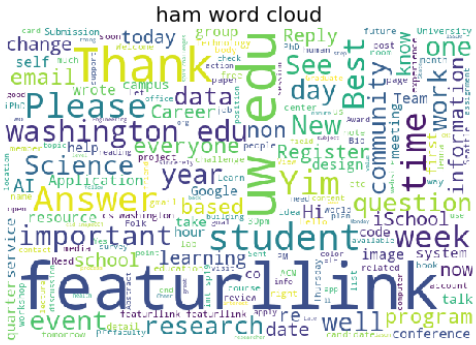


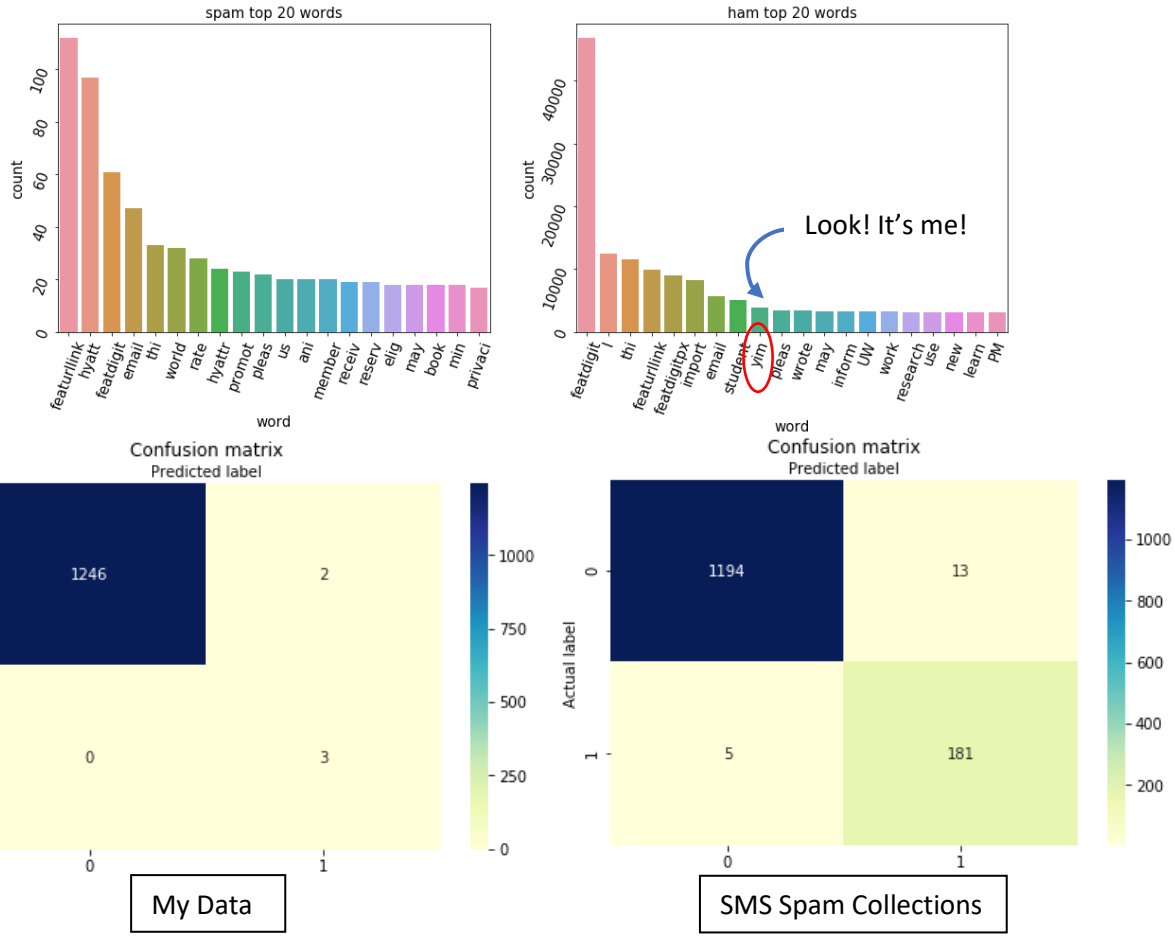
Figure 4: I didn't mind sharing some of the differences between my Spam and Ham. My Spam has a lot of things from the Hyatt apparently. While my school email talks about research, science, time, iSchool, AI, etc. Both have many links

Naïve Bayes

How it Works

I'm going to walk through how Naïve Bayes works, including why it includes the term "naïve" in it's name. Typically, when we think of "features" and "feature engineering", we think of variables like *price, weight, grade, height, gender, batting average, etc.* But for a spam filter, we don't have features in the traditional sense like that. What we have is all the words belonging to each document, and their counts. We can use the probability of each word showing up in a document to dictate our classification. The reason why it's called "Naïve" is because it makes a strong assumption that the words in each document are *independent*. We know that in language, that's just not true. Language patterns and syntax relate in intricate ways, and we often refer to **ngrams** to talk about **n** words in a series that occur together! But for Naïve Bayes, we ignore the series of words and treat each word independently. It tends to work in practice, especially with large datasets. I've laid out an example that was relevant for my own data, because for some reason I'm getting lots of spam from the Hyatt hotels. In our real analysis, we would use far more words than just "Hyatt" and "free" but I've demonstrated how you would calculate for more than one feature. Unfortunately in my case, the *P(spam)* was ridiculously low (13/5003).

$$\textbf{P(spam | "Hyatt", "free")} = \frac{P(\text{"Hyatt"}|spam) * P(\text{"free"}|spam) * P(spam)}{P(\text{"Hyatt"}) * P(\text{"free"})}$$



My Data

SMS Spam Collections

Performance

From what I've learned, logistic regression tends to be described as "discriminative" while Naïve Bayes is "generative". Basically speaking, this means that logistic regression uses the posterior as-is, and makes judgments from that. It's discriminating where a document belongs based on where it fell in the posterior we see in the world. With a small training set like mine, *it's going to perform poorly*. Naïve Bayes uses the probabilities of the features occurring in each class, and predicts the posterior from that. It's *generating* where a document belongs based on it's features. With a small data set, *it actually might perform better than logistic regression.* But on a larger set, logistic will actually do better than Naïve Bayes. This is exactly what happened in my own exploration. Naïve Bayes was actually able to capture 1-5 spams correctly (I ran the model several times to check)! Unlike logistic regression, which underfit drastically (labeling *everything* as hams). However, by looking at the actual large dataset SMS Spam Collections, I saw that Naïve Bayes isn't the automatic winner for all cases. Naïve Bayes had more false positives and false negatives than logistic regression did (which had only 1!), but had better overall accuracy and spam detection. This is an interesting exploration into how the features of your data matter, the size of your data matters, and the independence assumptions that you make might be good for one case but bad for another. I truly understand the tinkering aspects of ML through this project, especially because my dataset was so incredibly messy and sparse.

My Data

Accuracy: 99% (still misleading)
Spam detection: 58-66%

•	precision	recall	f1-score	support
•ham	1.00	1.00	1.00	1248
• spam	0.58	0.66	0.61	3
•avg / total	1.00	0.99	1.00	1251

SMS Collections Data

Accuracy 98%

Spam detection: 97%

•	precision	recall	f1-score	support
•ham	0.99	0.99	0.99	1191
•spam	0.93	0.95	0.94	202
•avg / total	0.98	0.98	0.98	1393