Naive Bayes Email Classification

Chase Perry

October 7, 2019

1 Problem Definition

The 21st century has come with plenty of technological advances that have benefited society and made us more efficient in numerous ways. Centuries ago we have physical mail that could take weeks to reach its recipient, advances in transportation shortened shortened that wait. Decades ago we developed emails, a near instantaneous mode of communication far superior to any other modes available at the time but as what happens with any technological advance it soon became abused. Due to the style of the technology, the cost to abuse it is near minimal. With the right knowledge and a couple lines of code thousands of emails could be sent to people across the globe, emails trying to convince the recipients of buying some product, sharing their information, or even sending money around the world.

The problem that many email providers face now, is how to correctly identify these emails so that their users only see emails actually intended for them and sent with good intentions and hiding those sent by ill-intending individuals. This is obviously a tough issue, as these spam emails aren't titled as being spam. In fact, many of these emails make the attempt to seem as legitimate as possible so that the recipients are more likely to trust and consider the topic of the email. So, the question then becomes: How can we identify spam emails (those being generic emails sent to many people as either scams or advertisements) and ham emails (legitimate emails that users may actually care about)?

2 How Naive Bayes Can Help

If you took a technology magazine, such as 'Wired', and compared the words throughout the magazines to those used in a publication similar to 'People' or 'Time' you'd likely fine a distinct difference in the diction used between the two. If that can be said for different types of magazines, why not different types of emails? Different classes of people, publications, websites, even video games use words differently to convey their ideas and intentions. The words used are choices, choices effected by the background of those creating the materials and by the type of material being produced. People obviously don't use the same kind of words for text messages that they would for a scholarly article, so could the same be said for a spam/ham email?

Each word has its own choice, and thus can be considered as a feature or data point of the email, each with it's own probability of being used in spam emails and a probability for being used in ham emails. By combining these probabilities in some manner, we can get a value that helps use determine how we should classify this email. Thus, Naive Bayes classification can help us determine an emails class and improve the experience of users, as whatever provider can best screen out their users spam emails will likely have more users as they can provide a better experience.

3 Data Set Source

The data used for this project is publicly available and was produced by Mr. Tasdik Rahman, with 250 different emails each marked with either a 1 or -1 to denote the class it falls into. A 1 indicates that the email is a valid email, which will be referred to as ham, and -1 indicates a spam email. An email can only fall into these two classes and can only belong to one class, as marking an email as being both a spam email and marking it as a ham email is not possible.

The data set has a 50/50 split between the ham and spam emails, working to the benefit of our classifying by providing both a unique and varying source. By having this split, we are able to get clear and diverse dictionary of words used throughout both classes of emails in contrast to so a 70/30 split where we'd likely suffer from a shortage of words used in one of the classes. The emails used also vary vastly in length and number of words, which is important for two reasons. For each word we essentially have a data point by which to classify the email, and the more data points to make that decision the better. The other reason being that with a varying length comes a difference in the words used as there's less to get the point across, thus potentially exposing scam emails as they will have to be more direct with their "request" or advertisement.

4 Proposed Solution

As stated above, each email has a number of words with each of those words having some probability of being used in either a ham email or a spam email. Using Naive Bayes and the overall probability of an email being in a class, regardless of it's words, we can combine these probabilities to create an overall probability that an email resides in one class or the other. However, the question then becomes: How best can we classify emails using this method? Is it necessary to use every word in an email? Only those from the first say 40%? Or perhaps only those in the last 30% of the email?

When asking how "best" we can classify emails, we must look at accuracy as well efficiency. While only looking at say 10% or 20% allows us to save on the amount of calculations we need to make, it may hurt our accuracy. Thus, we must also examine how our accuracy changes with the more words we consider when trying to classify an email. The assumption here would be that the more features (words in this case) by which to classify the email we have the better, but just how much of an accuracy improvement is there and is that difference worth the increase in calculations.

From this, the general probability that an email resides in a class can be found below in Figure 1. Where 'C_k' denotes a specific class from the set of classes the email could be placed in, in this case ham or spam, 'x' references an email, 'x_i' is a specific word from the set of words in the email, and '$p(C_k — x)$' is the probability that the email resides in that class C_k. By computing this value for both classes, we can place an email within a class based on whichever probability is higher and compare this against the true class it resides in to determine our accuracy with the placement.

$$p(C_k|x) \propto p(C_k) * \Pi_{i=1}^{n} p(x_i|C_k) \tag{1}$$

5 Solution

To begin, the data set was split 80%/20% such that the 80% part could be used for training and the remaining entries could be used for testing. From this training set two dictionaries were created, whose keys were the unique words in all emails across the data set. One dictionary, contained the probability of each words occurring in a spam email and the other contained the probability of each word occurring in a ham email. These probabilities were obtained by dividing the number of occurrences of a word in ham/spam emails by its total number of occurrences across the entire training data set. This work was done largely using the code found in Figure 5, though the code in the figure is an 'abbreviated' version of what was actually used. The probabilities for an email being in a class, regardless of the words used, was simply obtained by dividing the number of ham_emails/spam_emails by the overall number of emails in the training data set. While this was close to a 50/50 chance between the two, it wasn't perfectly at that figure so these probabilities were kept since they wouldn't cancel out.

```
master_words, ham_words, spam_words = {}

for i in range(0, x_train.size):
  email = x_train[i].split(' ')
  for ix in range(0, len(email)):
    master_words.update({email[ix]: 0})

ham_words.update(master_words)
spam_words.update(master_words)

ham_emails = x_train[y_train==1]
spam_emails = x_train[y_train==-1]

for word in master_words:
  for i in range(0, ham_emails.size):
    if word in ham_emails[i]:
      master_words[word] += 1
      ham_words[word] += 1
  for i in range(0, spam_emails.size):
    if word in spam_emails[i]:
      master_words[word] += 1
      spam_words[word] += 1

for word in master_words:
  ham_words[word] = ham_words[word] / master_words[word]
  spam_words[word] = spam_words[word] / master_words[word]

hguess_train = ham_emails.size / x_train.size
sguess_train = spam_emails.size / x_train.size

return master_words, ham_words, spam_words, hguess_train, sguess_train
```

Once the probabilities were obtained from the training data set, it was then necessary to decide how to classify the testing emails. There were three main attempts to classify the testing set, which varied purely on which words were used when making the decision of which class the email belonged to. These includes using the words at the beginning of the email, at the end of the email, or all words in the email. With the exception of using all the words, attempts made using the first/last words varied also on how many words were used as it was necessary to test both the accuracy and efficiency in both styles. The number of words used was not set for all emails, for instance at 50 or 100 words but rather at a percentage of the words in the email. This was done to avoid a bias towards the shorter emails, because if an attempt was made to classify an email of 200 words and an email of 1000 words using only the first/last 100 words we'd be throwing away more information about the longer email than the shorter when making our decision.

It was decided to also test accuracy/efficiency between the attempts using the first vs. the last words in order to see if there was perhaps some defining characteristics elsewhere in each email. Perhaps one class of emails started or ended in a specific manner? If only the first OR last words were used then we could potentially miss out on see this information, and thus miss a defining characteristic of a class and would never know to look for it. The percentages of the emails used in these attempts ranged from 10-90% increasing by 10 each time the attempt was made.

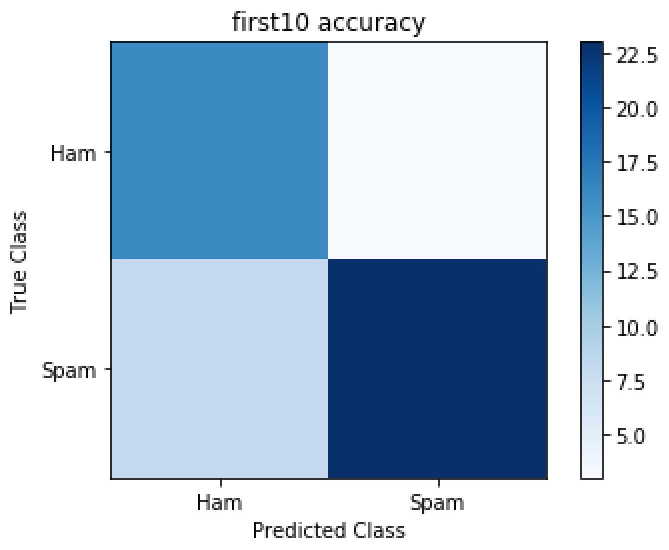
After deciding which words would be used, an attempt was made to get the probability that the word occurred in either class and multiply that by the current product probability for the email. For any word that was not found, meaning it had never been seen before in the training set, it was skipped along with any word that had a probability of existing in a class of zero. This process can be seen in Figure 5. While this means that certain words didn't get their "voice" into the mix to determine the class of an email it kept the math cleaner and less complicated.

```
for word in words:
      try:
        hprob_word = hwords[word] #dictionary of words & their ham ...
            probabilities
        sprob_word = swords[word] #dictionary of words & their spam ...
            probabilities

        if hprob_word == 0:
          hprob_word = 1
        if sprob_word == 0:
          sprob_word = 1

        ham_prob *= hprob_word #multiplying the current email ham prob. ...
            by the prob of the current word
        spam_prob *= sprob_word #same as above but for spam

      except KeyError:
        continue

    ham_prob *= hguess_train #multiplying the probability of the entire ...
        ham class
    spam_prob *= sguess_train #multiplying the probability of the ...
        entire spam class
```

The results of each attempt at classification resided in a two by two matrix similar to that of a general confusion matrix, where the first row and column signified the ham class while the second row and column signified the spam class. The final results of each attempt were then saved into a dictionary with a key describing that attempt for later analysis of accuracy. Each attempt at placing an email into a class consisted of comparing the product of the probabilities of all words chosen being in the ham/spam class and the overall probability of an email being in that class. Whichever had the higher resulting probability was the class the email was placed into, and in the unlikely event these probabilities were equal the classification was made by comparing the straight probabilities of an email being in either class obtained from the training data.

6 Evaluation

Generally speaking, using Naive Bayes and the procedure described above to classify emails as being spam or ham was rather successful based on the results of this project. With overall accuracy ranging from 82% to 96% most emails were placed into the correct class, with attempts using more of the words in an email to decide being more accurate (generally speaking). While results will obviously differ based on the data trained and tested against, as well as the size of the split between the training and test, there seemed to be next to no discernable difference between using the first or last percentage of the email, nor between

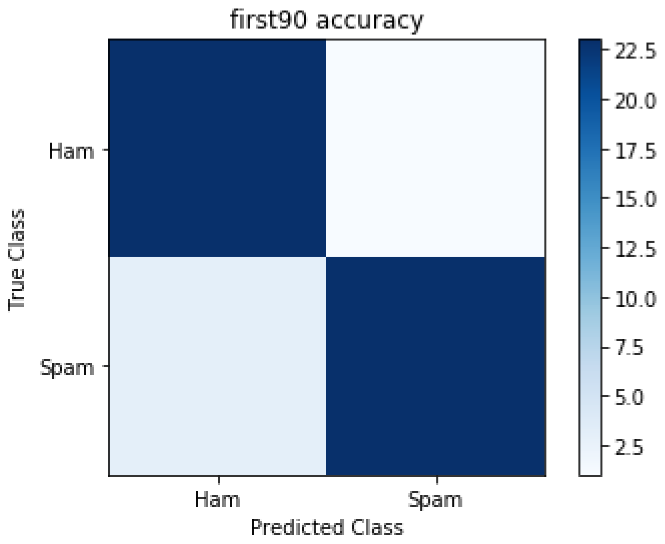


First 10% Of Emails Classification Attempt

using the first or last 90% or all words in an email. However, it does appears that it was more likely that the error appeared from classifying a spam email as being ham, rather than classifying ham as spam.

As stated above, increasing the portion of the email that was used for classification improved the accurate classification for the testing set. For instance, in Figure 1 we can see that more spam emails were classified as ham, and that while accurate there was some error to be found in the process. However, by increasing the portion of the email to 90% we have a resulting confusion matrix similar to that found in Figure 2, which shows nearly the same accuracy between accurate classifications in the true classes of ham and spam.

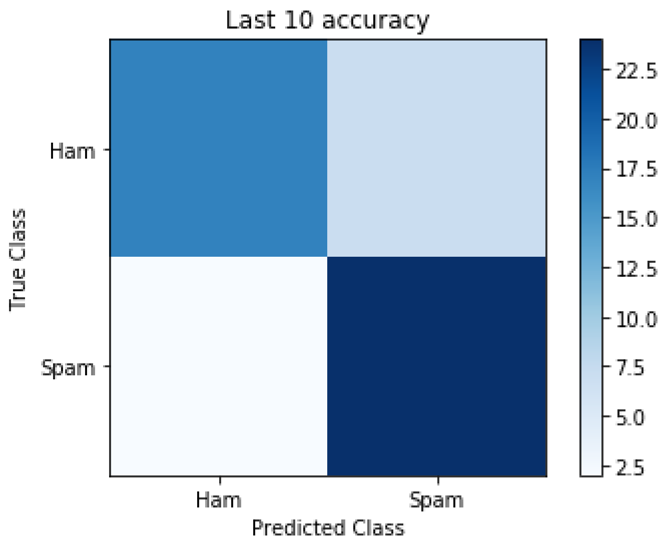
Similarly, by using the last 10% of an email for classification we have around 82% accuracy with our classifications, as can be seen in Figure 3. In this figure however, we're also able to see that more ham emails were incorrectly classified as being spam, something the 'last' attempts seemed to struggle more heavily with than the 'first' attempts. As with the 'first'



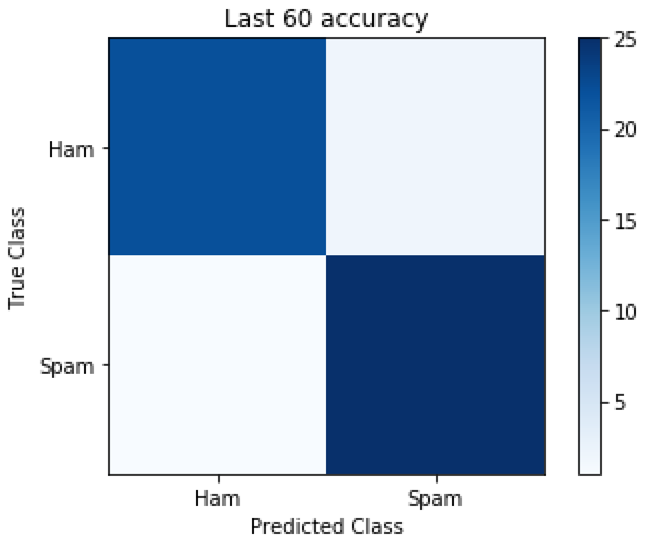
First 90% Of Emails Classification Attempt

attempts, including more of the email assisted with more accurate classification though there was a slight difference. While the 'first' attempts seemed to change more in accuracy when including different amounts of the email the 'last' attempts seemed to reach a point where the accuracy simply held steady. For instance, regardless of if the last 60% of an email was used or if the last 90% was used both had an error of 94%, as did the 70% and 80% attempts (this can be view in Figures 4 and 5).

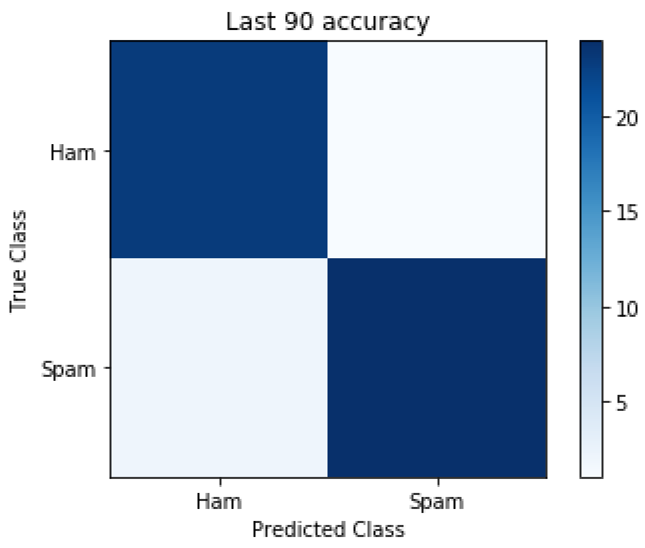
Finally, using all the words in an email produced similar results to using larger portions of the email for classification, unsurprisingly. This shows, that while using all the words for classification may have included more data point there could potentially be a drop off where including more words doesn't increase the accuracy of the classification. This potential was not investigated enough, however, to give an enforceable opinion on the change so I cannot say what the optimal portion of an email might be, so this would require further study.



Last 10% Of Emails Classification Attempt



Last 60% Of Emails Classification Attempt



Last 90% Of Emails Classification Attempt

7 Discussion

While accurate in it's predictions, roughly speaking, Naive Bayes makes several assumptions about an email that I'm not sure are safe. The largest of these assumptions being that there is no relation between any of the words in an email, which could cause disastrous effects when it comes to an emails classification. For instance, suppose a scammer sent an email and knowing the recipients email provider was using this method of classification and used a number of nonsense words at the bottom of the email to increase the chances it was get marked as ham and not spam. These words wouldn't have to mean anything together, as we're assuming there's no correlation between the words used, so they could have a simple string of words that would improve their chances like 'my heart birds feed doctor please' and could potentially skate right past the filter. Of course the recipient would find this odd and would likely delete it, but as our goal is to hide these emails from our users, we would have failed.

Regardless, given how simplistic Naive Bayes is and yet how accurately it predicted the classes of emails I would say that it is safe to use as a spam filter for production. As more emails are received and incorporated into its dictionaries to adjust the probabilities that a word exists in an email in either class, the process would naturally improve. While having more keys to search through, as each key would be the word added in, would increase the time to search I believe the accuracy gain would be enough to warrant this increase.

8 Future Work

In my opinion, the largest area for future work lies not much in changing the overall process but more-so in changing the selection of which words matter for classification. Specifically, I believe that selecting words with varying degrees of having the highest difference in probabilities between classes has the best chance of improving the accuracy of this method. For instance, if we searched through an entire email and got the words with the greatest difference between their probability of existing in an email in the ham versus spam class, we'd find the words that have the best chance to change that specific emails overall probability of belonging in either class.

As an example, if we found the word 'enlargement' had a probability of 1 for occurring in a spam email and a probability of 0 for occurring in a ham email, based on the training data set, then it would serve as a pretty clear indicator that it was a distinct marker for a spam email. Thus, by selecting the words that have greatest differences in probability we'd be selecting the key data points from each email to help identify them. Producing something that implemented the above strategy to test its effectiveness was attempted, but was unfortunately not prepared to be presented in this product.

9 What I've Learned

Throughout this project I've learned quite a bit on Python syntax, functions, and manipulations of objects, as well as how it deals with memory. As it has been some time since I've used Python thoroughly and to solve tougher problems its been some time since I've found myself stuck with how Python deals with certain things, and thus how to get around or deal with how Python attempts to go about managing something, such as a dictionary. I've also learned quite a bit on Naive Bayes classifier, as before this project I will admit I was quite confused as to how the process works but afterwards well, while I'm not ready to write poetry about it I do feel that I have a better grasp on how exactly it works.