Project II: Naïve Bayes Classifier

CSE 107 Probability and Statistics for Engineers
Textbook: Probability & Statistics with Applications to Computing, Alex Tsun
http://www.alextsun.com/files/Prob_Stat_for_CS_Book.pdf

Instructor: Chen Qian

We strongly recommend that you do NOT use C language for this project. Python and C++ are recommended. Using C would cost your extra time to configure based on different platforms (Windows, Linux, or Mac). If you really want to use C, please start your project early and visit TA's discussion sections for clarification.

1 Motivation

Even with the limited knowledge we learned from this course, we are able to write a simple yet useful machine learning program (probably your first one). In this project, we are going to discover one way to solve one extremely important type of machine learning task: classification. In particular, we'll learn how to take in an email (a string/text), and predict whether it is "Spam" or "Ham". We will discuss this further shortly!

2 Naïve Bayes Classifier

To summarize, our end goal is to write a function which takes in an email (a string type) and returns either that it is "SPAM" or "HAM". The function in Python may look something like this.

```
def classify(email:str):
    # Some Code Here
    if some_condition:
        return SPAM
    else:
        return HAM
```

So how do we write the code to make the decision for us? In the past, people tried writing these classifiers with a set of rules that they came up themselves. For example, if it is over 1000 words, predict "SPAM". Or if it contains the word 'millionaire', predict that it is "SPAM". This leads to code which looks like a ton of if-else statements, and is also not very accurate. In machine learning, we come up with a model that learns a decision-making rule for us!

2.1 Prepossessing

Handling text can be very messy. People misspell words, use slang that isn't in the vocabulary, have bad grammar, use tons of punctuation, and so on. When we process our emails, we will employ the following approach:

1. Ignore Duplicate Words.
2. Ignore Punctuation.
3. Ignore Casing.

That is, we will reduce an email into a Set of lowercase words and nothing else! We'll see a potential drawback to this later, but despite these strong assumptions, the classifier still does a really good job!

2.2 Decision Rule

For this section, we'll use the example of classifying the email "Wanna be millionaire!". The representation we have after processing is wanna, be, millionaire. Here's the approach of the Naive Bayes classifier. We will compute and compare the following two quantities (which must add to 1):

$$\mathbb{P}(spam|\{wanna, be, millionaire\}) \text{ and } \mathbb{P}(ham|\{wanna, be, millionaire\})$$

This is because, for a particular email, it is either spam or ham, and so the probabilities must sum to 1. In fact, because they both sum to 1, we can just compute one of them (let's say the rfist), and predict SPAM if $\mathbb{P}(spam|\{wanna, be, millionaire\}) > 0.5$ and HAM otherwise. Note that if it is exactly equal to 0.5, we will predict HAM (this is arbitrary - you can break ties however you want).

2.3 Learning from Data

WARNING: This is the heaviest math section, which draws from all ideas of our lectures about conditional probability. Let's try Bayes Theorem with the Law of Total Probability and see where that gets us! Let E be the event that an email includes 'wanna', F be the the event that an email includes 'be', and G be the event that an email includes 'millionaire'. The following notations are the same: $\mathbb{P}(spam|E \cap F \cap G)$=$\mathbb{P}(spam|\{E, F, G\})$= $\mathbb{P}(spam|\{wanna, be, millionaire\})$

$$\begin{aligned}
\mathbb{P}(spam|\{E,F,G\}) &= \frac{\mathbb{P}(\{E,F,G\}|spam)\mathbb{P}(spam)}{\mathbb{P}(\{E,F,G\})} && \text{Bayes} \\
&= \frac{\mathbb{P}(\{E,F,G\}|spam)\mathbb{P}(spam)}{\mathbb{P}(\{E,F,G\}|spam)\mathbb{P}(spam) + \mathbb{P}(\{E,F,G\}|ham)\mathbb{P}(ham)} && \text{LTP}
\end{aligned}$$

Let's assume we were given five examples of emails with their labels (Spam or Ham) to learn from. We call these emails as the **training data**.

Email	Label
Wanna be millionaire?	Spam
I wanna go there. Thanks!	Ham
You are a millionaire!	Spam
Millionaire is possible!	Spam
Yao's millionaire problem	Ham

Based on the data only, what would you estimate $\mathbb{P}(spam)$ to be? That is 3/5, because

$$\mathbb{P}(spam) \simeq \frac{\#\text{of spam emails}}{\#\text{ of total emails}}$$

Similarly, we might estimate

$$\mathbb{P}(ham) \simeq \frac{\#\text{of ham emails}}{\#\text{ of total emails}}$$

to be 2/5 in our case.

The *Naive Bayes* name comes from two parts. We've seen the Bayes part because we used Bayes Theorem to (attempt to) compute our desired probability. We are at a roadblock now, and now we will make the "naive" assumption that: words are conditionally independent GIVEN the label. That is,

$$\mathbb{P}(\{wanna, be, millionaire\}|spam) \simeq \mathbb{P}(wanna|spam)\mathbb{P}(be|spam)\mathbb{P}(millionaire|spam)$$

So now, we might estimate

$$\mathbb{P}(wanna|spam) \simeq \frac{\#\text{of spam emails with you}}{\#\text{ of spam emails}}$$

What should this quantity be? It is 1/3: there is just one spam email out of three which contains the word "wanna". In general,

$$\mathbb{P}(word|spam) \simeq \frac{\#\text{of spam emails with word}}{\#\text{ of spam emails}}$$

Now, Let's combine what we had earlier (Bayes+LTP) with the (naive) conditional independence assumption. Recall we let E be the event that an email includes 'wanna', F be the the event that an email includes 'be', and G be the event that an email includes 'millionaire'.

Solution:

$$\begin{aligned}
&\mathbb{P}(spam|\{E,F,G\})\frac{\mathbb{P}(\{E,F,G\}|spam)\mathbb{P}(spam)}{\mathbb{P}(\{E,F,G\}|spam)\mathbb{P}(spam)+\mathbb{P}(\{E,F,G\}|ham)\mathbb{P}(ham)} \\
&= \frac{\mathbb{P}(E|spam)\mathbb{P}(F|spam)\mathbb{P}(G|spam)\mathbb{P}(spam)}{\mathbb{P}(E|spam)\mathbb{P}(F|spam)\mathbb{P}(G|spam)\mathbb{P}(spam)+\mathbb{P}(E|ham)\mathbb{P}(F|ham)\mathbb{P}(G|ham)\mathbb{P}(ham)}
\end{aligned}$$

We need to compute a bunch of quantities, but notice the left side of the denominator is the same as the numerator, so we need to compute 8 quantities, 3 of which we did earlier!

$$\begin{aligned}
\mathbb{P}(spam) &= \frac{3}{5} & \mathbb{P}(ham) &= \frac{2}{5} \\
\mathbb{P}(E|spam) &= \frac{1}{3} & \mathbb{P}(E|ham) &= \frac{1}{2} \\
\mathbb{P}(F|spam) &= \frac{1}{3} & \mathbb{P}(F|ham) &= \frac{0}{2} \\
\mathbb{P}(G|spam) &= \frac{3}{3} & \mathbb{P}(G|ham) &= \frac{1}{2}
\end{aligned}$$

Once we plug in all these quantities, we end up with a probability of 1, because the $\mathbb{P}(F|ham) = 0$ killed the entire right side of the denominator! It turns out then we should predict spam because $\mathbb{P}(spam|\{wanna, be, millionaire\}) = 1 > 0.5$, and this is correct! We still don't ever want zeros though, so we'll see how we can fix that soon!

Notice how the data (example emails) completely dictated our decision rule, along with Bayes Theorem and Conditional Independence. That is, we learned from our data, and used it to make conclusions on new data! One last final thing, to avoid zeros, we will apply the following trick called "Laplace Smoothing". Before, we had said that

$$\mathbb{P}(word|spam) = \frac{\#\text{of spam emails with word}}{\#\text{ of spam emails}}$$

We will now pretend we saw TWO additional spam emails: one which contained the word, and one which did not. This means instead that we have

$$\mathbb{P}(word|spam) = \frac{\#\text{of spam emails with word + 1}}{\#\text{ of spam emails+2}}$$

This will ensure that we don't get any zeros! For example, $\mathbb{P}(buy|ham)$ was $\frac{0}{2}$ previously (none of the two ham emails contained the word "buy"), but now it is $\frac{0+1}{2+2} = \frac{1}{4}$.

We do not usually apply Laplace smoothing to the label probabilities $\mathbb{P}(spam)$ and $\mathbb{P}(ham)$ since these will never be zero anyway.

2.4 Evaluating Performance

Let's say we are given 1000 emails for learning our spam filter using Naive Bayes. How should we measure performance? We could check the accuracy, which is exactly what you think it is:

$$accuracy = \frac{\text{\# of emails calssified correctly}}{\text{\# of total emails}}$$

However, if we trained/learned from these 1000 emails, and measure the accuracy, surely it will be very good right? It's like taking a practice test and then using that as your actual test - of course you'll do well! What we care about is how well the spam filter works on NEW or UNSEEN emails. Emails that the spam filter was not allowed to see/use when estimating those probabilities. This is fair and more realistic now right? You get practice exams, as many as you want, but you are only evaluated once on an exam you (hopefully) haven't seen before!

Where do we get these new/unseen emails? We actually take our initial 1000 emails and do a train/test split (usually around 80/20 split). That means, we will use 800 emails to estimate those quantities, and measure the accuracy on the remaining 200 emails. The 800 emails we learn from are collectively called the **training set**, and the 200 emails we test on are collectively called the **test set**. This is good because we care how our classifier does on new examples, and so when doing machine learning, we ALWAYS split our data into separate training/testing sets!

2.5 Underflow Prevention

Computers are great, but sometimes they cause us problems. When we compute something like

$$\prod_{i=1}^{k} \mathbb{P}(w_i|spam)$$

we are multiplying a bunch of numbers between 0 and 1, and so we will get some very very small number (close to zero). When numbers get too large on a computer (exceeding around $2^{64} - 1$), it is called overflow, and results in weird and wrong arithmetic. Our problem is appropriately named underflow (when the exponent is < 128), as we can't handle the precision. For example, if we tried to represent the number 3.2×10^{-133}, this would be an underflow problem.

Remember that our two probabilities $\mathbb{P}(spam|\{w_1, ..., w_k\})$ and $\mathbb{P}(ham|\{w_1, ..., w_k\})$ summed to 1, so we only needed to compute one of them. Let's go back to computing both, and just comparing which is larger:

$$\mathbb{P}(spam|\{w_1, ..., w_k\}) = \frac{\mathbb{P}(spam)\prod_{i=1}^{k}\mathbb{P}(w_i|spam)}{\mathbb{P}(spam)\prod_{i=1}^{k}\mathbb{P}(w_i|spam) + \mathbb{P}(ham)\prod_{i=1}^{k}\mathbb{P}(w_i|ham)}$$

$$\mathbb{P}(ham|\{w_1, ..., w_k\}) = \frac{\mathbb{P}(ham)\prod_{i=1}^{k}\mathbb{P}(w_i|ham)}{\mathbb{P}(spam)\prod_{i=1}^{k}\mathbb{P}(w_i|spam) + \mathbb{P}(ham)\prod_{i=1}^{k}\mathbb{P}(w_i|ham)}$$

Notice the denominators are equal: they are both just $\mathbb{P}(\{w_1, ..., w_k\})$. So, $\mathbb{P}(spam|\{w_1, ..., w_k\}) > \mathbb{P}(ham|\{w_1, ..., w_k\})$ if and only the corresponding numerator is greater:

$$\mathbb{P}(spam)\prod_{i=1}^{k}\mathbb{P}(w_i|spam) > \mathbb{P}(ham)\prod_{i=1}^{k}\mathbb{P}(w_i|ham)$$

Recall the log properties:

$$log(xy) = log(x) + log(y)$$

and that both sides are simply a product of $k+1$ terms. We can take logs on both sides and this preserves order because log is a monotone increasing function (if $x > y$ then $log(x) > log(y)$):

$$log(\mathbb{P}(spam)) + \sum_{i=1}^{k} log(\mathbb{P}(w_i|spam)) > log(\mathbb{P}(ham)) + \sum_{i=1}^{k} log(\mathbb{P}(w_i|ham))$$

And that's it, problem solved! If our initial quantity (after multiplying 50 word probabilities) was something like $\mathbb{P}(spam)\prod_{i=1}^{k}\mathbb{P}(w_i|spam) \simeq 10^{-81}$, then $log\mathbb{P}(spam|\{w_1, ..., w_k\}) \simeq -186.51$. There is no chance of underflow anymore!

2.6 Project requirements:

1. Use the Naive Bayes Classifier to implement a spam filter that learns word spam probabilities from our pre-labeled training data and then predicts the label (ham or spam) of a set of emails that it hasn't seen before. Write your code for the following parts in the provided file: **naive_bayes.py** or **naive_bayes.cpp** or **naive_bayes.c**

2. Read about how to avoid floating point underflow using the log-trick in the above notes.

3. Make sure you understand how Laplace smoothing works.

4. Remember to remove any debug statements that you are printing to the output.

5. Needless to say, you should practice what you've learned in other courses: document your program, use good variable names, keep your code clean and straightforward, etc. Include comments outlining what your program does and how.

   Remember, it is not expected that Naive Bayes will classify every single test email correctly, but it should certainly do better than random chance! As this algorithm is deterministic, you should get a certain specific test accuracy around 90-95%, which we will be testing for to ensure your algorithm is correct.

Please include a pdf report that include

- Your name and contact information
- The instructions to compile/run your program
- Screenshot of the expected output of your program
- Your understanding of why your program works / does not work