CS 13: Mathematical Foundations of Computer Science Fall 2023

Classification: Application 02 (due November 19th)

Directions: *Write up carefully argued solutions to the following problems. Your solution should be clear enough that it should explain to someone who does not already understand the answer why it works. You may use results from lecture and previous homeworks without proof. Your solutions* must *be written in LaTeX using our homework template.* ***No solution to a single part may be more than one page.***

A common machine learning problem we encounter is wanting to classify documents of words into different categories. It would be great, for example, to be able to classify an email as "spam" without reading it, or classify a tweet's emotion instantly. This is exactly what the Naive Bayes algorithm allows us to do!

0. Deriving Naive Bayes (10 points)

Naive Bayes is what is called a "bag of words" approach, which reduces a document to just a set of words (meaning that we are completely ignoring the order of the words) and works with that. This seems like it loses the majority of the information—and it does—but it turns out to work decently well despite this.
Let's say we have a document with n distinct words, which we enumerate arbitrarily as $W = \{w_i\}_{i=0}^{n-1}$ (note that this is not necessarily the order of the words in the document). We can imagine that the document (defined for our purposes by the set W) was pulled from some ideal distribution of documents, and let $B_{\texttt{word}}$ be the event that a document pulled from this distribution contains `word`. Let A be the event that our document is spam. In order to classify our document as spam or not spam, we want to find $\Pr(A \mid B)$, where

$$B = \bigcap_{i=0}^{n-1} B_{w_i}$$

is the event that a document drawn from the distribution of documents contains all of the words W.
Applying Bayes' Theorem gives us

$$\Pr(A \mid B) = \frac{\Pr(B \mid A)\Pr(A)}{\Pr(B)}.$$

Note that since $\Pr(B)$ is a scaling factor, we will omit it from our calculations (your job in this part is to justify why).
Thus,

$$\Pr(A \mid B) \propto \Pr(B \mid A)\Pr(A)$$

In Bayesian statistics, the right hand side of this expression is called the "likelihood" of A given B.
Making the additional (naive) assumption that the B_{w_i}'s are conditionally independent with respect to A, which means that for each $0 \le i, j < n$, whenever $i \ne j$ we have $\Pr\big(B_{w_i} \cap B_{w_j} \mid A\big) = \Pr(B_{w_i} \mid A)\Pr\big(B_{w_j} \mid A\big)$, we get:

$$\Pr\left(\bigcap_{i=0}^{n-1} B_{w_i} \mid A\right) = \prod_{i=0}^{n-1} \Pr(B_{w_i} \mid A)$$

Plugging this into our previous equation gives:

$$\Pr(A \mid B) \propto \left(\prod_{i=0}^{n-1} \Pr(B_{w_i} \mid A)\right)\Pr(A)$$

In Naive Bayes classification, we estimate the probabilities $\Pr(B_{w_i} \mid A)$ and $\Pr(A)$ relative to a *corpus* of documents which have already been labeled with categories $C_1, \ldots, C_k$ – in our case, we only have $k = 2$

categories, spam ($C_1 = A$) and ham (that is, not spam, or in other words, $C_2 = \overline{A}$, the complement of A). By taking the category C_i with the largest likelihood, we can give our document a single, most likely classification.

(a) [10 Points] Explain why just a proportional likelihood value suffices for classification purposes and we don't need to calculate the actual value of $\mathsf{Pr}(A \mid B)$.

1. Uh Oh! Underflow! (70 points)

Imagine we have 150 documents, all spam, each of a single distinct word. Then, $\mathsf{Pr}(B_{w_i} \mid A) = \dfrac{1}{150}$.
So,

$$\mathsf{Pr}(A \mid B) \propto \left(\frac{1}{150}\right)^{150} \times 1 \approx \texttt{2.5923E-326}$$

Unfortunately, Python helpfully tells us that this quantity is zero:

```
>>> (1/150)**150
0.0
```

It turns out, real numbers are not stored exactly on computers. Instead, computers use a system called "floating point numbers" which is a standardized approximation of the real numbers and used everywhere. At a high level, floating point numbers are basically a binary version of scientific notation.

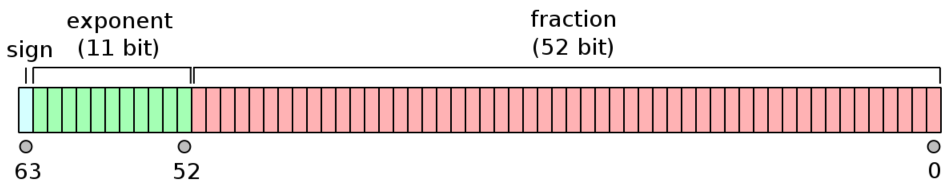
To fix this underflow problem, we're going to be taking the log of the probabilities and using those instead, like so:

$$\begin{aligned}
\mathsf{Pr}(A \mid B) &\propto \left(\prod_{i=0}^{n} \mathsf{Pr}(B_{w_i} \mid A)\right) \mathsf{Pr}(A) \\
\log \mathsf{Pr}(A \mid B) &= \log\left(\left(\prod_{i=0}^{n} \mathsf{Pr}(B_{w_i} \mid A)\right) \mathsf{Pr}(A)\right) \\
&= \left(\sum_{i=0}^{n} \log(\mathsf{Pr}(B_{w_i} \mid A))\right) + \log(\mathsf{Pr}(A))
\end{aligned}$$

But, we'd like a justification for why this works. So, we'll delve into floating point a little bit.

`doubles` are stored in 64 bits, broken up into three parts:

- s: the sign (first bit)
- e: the exponent (next 11 bits)
- $f = b_{51}b_{50}\cdots b_0$: the fraction (final 52 bits)



To calculate the value of a floating point number, we use the following rules:

- If $e \neq 0$ and $e \neq \texttt{0x7ff}$ (all 1s), we say the number is "normal" and use the following formula:

$$(-1)^s \times 2^{e-1023} \times (1.b_{51}b_{50}\cdots b_0)_2$$

i.e.

$$(-1)^s \times 2^{e-1023} \times \left(1 + \sum_{i=1}^{52} b_{52-i}2^{-i}\right)$$

- If $e = 0$ and $f = 0$, then the number represented is +0 or -0
- If $e = 0$ and $f \neq 0$, we say the number is "subnormal" and we use an alternate formula:

$$(-1)^s \times 2^{-1022} \times (0.b_{51}b_{50}\cdots b_0)_2$$

- If $e =$ `0x7ff`, then the number represented is NaN or an infinity.

(a) [10 Points] Use Python to determine the smallest representable floating point number greater than 0. (Hint: Use powers of two.)

(b) [10 Points] Prove that floating-point addition is not associative. That is, prove there exist x, y, z such that $(x + y) + z \neq x + (y + z)$ when $+$ is floating point addition.

(c) [5 Points] What is the smallest normal value greater than 0?

(d) [10 Points] How many values (both normal and subnormal) can be represented in the range $(0, 1)$ (exclusive)?

(e) [10 Points] How many values can be represented in the range $(-\infty, 0)$ (exclusive of $-\infty$ and 0)?

(f) [15 Points] Prove that $\ln(x)$ is an increasing function. You may assume the mean value theorem and that $\ln x$ is continuous and differentiable in its entire domain, with a derivative of $\frac{1}{x}$. However, you may not use any other theorems from calculus.

(g) [5 Points] Explain without proof why (d) and (e) together show that the number of probabilities we can represent increases when we take the $\log$.

(h) [5 Points] Explain why it's necessary for $\log$ to be monotonic for this transformation to be valid.

Because taking the $\log$ greatly increases our space of representable probabilities, and turns hard-to-compute multiplications into easy-to-compute additions, it is often used in machine learning algorithms such as Naive Bayes!

2. Implement Naive Bayes (20 points)

In this part, you will implement Naive Bayes as described above to classify some documents and tweets into different categories.

Laplace Smoothing

One final "gotcha" you will encounter is when you are processing words in the classification files that aren't present in the training set. We use *Laplace Smoothing* to fix this issue.

In essence, to ensure that each word in the document appears at least once as each category, we add a phantom document for each word in the document which contains just that word and belongs to the category. **Do not actually add documents!**

Instead, for each word, when calculating Pr$(B_{w_i} \mid A)$ you should add 1 to the numerator (since there is one more phantom document containing the word in the category) and n in the denominator (since there are n more phantom documents total in the category).

This will prevent us getting a division-by-zero error in this special case.

The Data

In this problem, we provide you two distinct data sets to practice training on. They are "emails" and "tweets", respectively.

The Code

You are responsible for filling in the probability/likelihood calculations in the code. We've handled everything else for you. Note that these functions are the core of Naive Bayes as they correspond directly to the quantities in the formula.

When you clone or download the starter repository, you will find several files inside. While you are welcome to look around if you are curious, **the only file you will submit, and thus the only file you should edit, is `bayes.py`**.

The driver code in `main.py` will call the `train` function on the data set and then the `predict` function on each document. The assignment can be completed by editing only `pr_category`, `pr_word_given_category`, and `log_likelihood_category_given_words`, but you are allowed to edit or create others if you wish.

Our reference solution adds less than twenty lines to the starter code, though a solution which includes more may well be correct.

Grading

We will check your classifications against the provided classifications given in the "validate" files. Here are our results for the two data sets we've given you:

tweets_validate
correctly classed as:
negative: 1166
positive: 780
incorrectly classed as:
positive: 420
negative: 216

emails_validate
correctly classed as:
spam: 145
ham: 326
incorrectly classed as:
spam: 0
ham: 29

You should expect something roughly similar to these results. **We will also be running your code on a private test file, so these results alone are not enough to guarantee full credit.**