Assignment 5: Part 2

CSCI2244-Randomness and Computation
Part 2 due Monday, April 1 at 11:59

This part of the assignment has you implement a Naïve Bayes Classifier. The intended use is to distinguish which of two newspapers a given newspaper headline comes from, or to tell whether a headline is from the Sports or the Politics section of the newspaper.

I have posted several different datasets (described below). I did a lot of cleaning up of the original data, so that in the end, each dataset is an ordinary text file with one headline per row. I have also provided some Python code for reading in the file and doing some pre-processing. So most of the messy uninteresting work has already been done for you. Your job is to implement the classifier: this will separate the data into training and test sets, build probability models from the training sets, and evaluate the performance of the classifier on the test sets.

1 The theory.

(You might want to read the Example below first!)

Let's suppose we want to use the classifier to distinguish whether a brief snippet of a news story is from the Sports section or the Politics section. (We'll assume that we know it is one of these two.)

If you choose a word at random from a Sports headline, what is the probability that this word is, say, 'prize'? We write this as

$$P(\texttt{`prize'}|\texttt{Sports}).$$

In doing so, we conceive of Sports headlines as a giant 'bag of words', in which each word appears with a particular probability, much like the urns of M&Ms described in the lecture notes. And we think of Politics headlines as another such bag.

A headline D is a sequence of words

$$D = (w_1, \ldots, w_r).$$

We think of D as being generated by reaching into one of the bags r times, and pulling out the word w_i on the i^{th} draw. Thus

$$P(D|\texttt{Sports}) = \prod_{i=1}^{r} P(w_i|\texttt{Sports}).$$

This is the naïve part of Naïve Bayes—in effect we're treating the sequence of words

> Boston is dominating, and even with Chris Sale headed to the disabled list, the gap is wide enough that the Red Sox mostly watched as other teams scrambled.

(one of the Sports items in our dataset) as just as likely to be generated as the nonsensical string

> Sale Boston teams scrambled even as dominating disabled watched gap, and with Chris headed to Sox the list, the enough that the Red mostly other is is wide.

While this assumption is obviously false, in practice it often leads to effective results.

The task of classification is to take a message D and decide whether it is from the Sports or the Politics section; that is, we want to decide which of the two values

$$P(\texttt{Sports}|D), \quad P(\texttt{Politics}|D)$$

is larger. By Bayes's Theorem,

$$P(\texttt{Sports}|D) = \frac{P(D|\texttt{Sports})}{P(D)} \cdot P(\texttt{Sports}), \quad P(\texttt{Politics}|D) = \frac{P(D|\texttt{Politics})}{P(D)} \cdot P(\texttt{Politics}).$$

We can ignore the common denominator $P(D)$ on the right-hand sides, and apply the naïve assumption above, so that the problem now is to determine which of the following is larger:

$$P(\texttt{Sports}) \cdot \prod_{i=1}^{r} P(w_i|\texttt{Sports}), \quad P(\texttt{Politics}) \cdot \prod_{i=1}^{r} P(w_i|\texttt{Politics}).$$

We call these two numbers the 'Sports score' and the 'Politics score' of the document D.

1.1 Estimating probabilities

To compute the scores, we need estimates for $P(w|\texttt{Sports})$ and $P(w|\texttt{Politicss})$ for each word w, as well as estimates of $P(\texttt{Sports})$ and $P(\texttt{Politics})$. The simplest idea is to gather up all the Sports items in our training data, count the total number $N_{\texttt{Sports}}$ of words, and the total number $N_{w,\texttt{Sports}}$ of occurrences of w among these words, and estimate

$$P(w|\texttt{Sports}) \approx \frac{N_{w,\texttt{Sports}}}{N_{\texttt{Sports}}}.$$

For example, if there are a total of 6000 distinct words in the Sports messages in the training data, and 150 of them are the word 'Mets', then we would approximate

$$P(\text{`}\texttt{Mets}\text{'}|\texttt{Sports}) \approx 150/6000 = 0.025.$$

Of course, we do the same thing to estimate $P(w|\texttt{Politics})$.

We also need to have estimates of $P(\texttt{Sports})$ and $P(\texttt{Politics})$. These are our 'priors'. Here we require some kind of global estimate of the fraction of messages that are Sports. For purposes of this problem, you can use the proportions in the datasets. The Sports dataset contains 1204 items, and the Politics dataset 3290 items, so we estimate

$$P(\texttt{Sports}) \approx \frac{1204}{1204 + 3290} = 0.366.$$

1.2 Weeding out stop words

Common English words like 'the' and 'and' can predominate in our text samples, and their distribution tends to be similar in the two classes of documents. So it is a common practice to remove these from the document before doing the analysis.

1.3 Add-1 smoothing

What if a word, say, 'excite', appears once or twice in the Politics training set, but never in the Sports training set? $P(\text{`}\texttt{excite}\text{'}|\texttt{Sports})$ will be then be estimated as 0, while $P(\text{`}\texttt{excite}\text{'}|\texttt{Politics})$ will be positive. If we then try to evaluate a headline that happens to be from the Sports data and actually contains this word, then the Sports score will be 0, but the Politics score might be non-zero, so the document will be classified incorrectly as having come from the Politics section.

Many words, of course, will have this property. To get around this, we artificially add 1 to every count, pretending that every word encountered in the training sets appears in both sets of headlines: Let V be the set of distinct words appearing in the two training sets, from either of the two document classes, and set for each $w \in V$,

$$P(w|\texttt{Sports}) \approx \frac{N_{w,\texttt{Sports}} + 1}{N_{\texttt{Sports}} + |V|},$$

and likewise for Politics. If the document D contains words that appear in neither training set, then we just skip these words in computing the score. In this manner, every document will get a positive score for both classes.

1.4 Taking logs to avoid underflow

Typically, for any given word w, $P(w|\texttt{Sports})$ and $P(w|\texttt{Sports})$ will be quite small, in the neighborhood of 10^{-4} or less. If a document contains a few dozen words, we will wind up risking floating-point underflow when we try to evaluate the product

$$P(\texttt{Sports}) \cdot \prod_{i=1}^{r} P(w_i|\texttt{Sports}).$$

To avoid this, we take the logarithm of the product, which is the sum of the respective logarithms, and replace the score by

$$\ln(P(\texttt{Sports})) + \sum_{i=1}^{r} \ln(P(w_i|\texttt{Sports})).$$

Note that this will be a negative number, since we are taking logs of numbers less than 1. If you want, you can negate it and make the score positive, but just keep in mind that if you do this, you will want to find which of the two values gives a *lower* score.

1.5 Example

We give an example with artificially small numbers. Let's suppose we receive two packages of M&M's from two different factories, and use these as training data to construct a classifier. (Unlike the example in the notes, we don't assume any information about what the color distribution *should* be. Of course, this is a

ridiculously small sample to be used for a training set.) The color counts in the two packages are as follows:

	Factory 1		**Factory 2**	
Color	Absolute	Relative	Absolute	Relative
Red	21	0.456	12	0.255
Brown	14	0.304	8	0.170
Green	6	0.130	8	0.170
Yellow	0	0	2	0.043
Blue	5	0.109	17	0.362
Total	**46**	**1.00**	**47**	**1.00**

When we apply add-1 smoothing, we modify all the absolute counts:

	Factory 1		**Factory 2**	
Color	Absolute	Relative	Absolute	Relative
Red	22	0.431	13	0.25
Brown	15	0.294	9	0.173
Green	7	0.137	7	0.173
Yellow	1	0.020	3	0.058
Blue	6	0.118	18	0.346
Total	**51**	**1.00**	**52**	**1.00**

Now suppose we receive a new bag of M&Ms whose origin is unknown, in which the color distribution is as follows:

Color	Quantity
Red	18
Brown	11
Green	10
Yellow	4
Blue	1
Orange	2
Total	**46**

This is the test data to which we apply the classifier. We believe that Factory 1 M&Ms are twice as likely to have shown up in our new shipment than Factory 2 M&Ms, so we put $P(\text{Factory 1}) = \frac{2}{3}$, and $P(\text{Factory 2}) = \frac{1}{3}$. We then compute the Factory 1 score:

$$0.431^{18} \times 0.294^{11} \times 0.137^{10} \times 0.20^{4} \times 0.118^{1} \times \frac{2}{3} \approx 10^{-29}.$$

That's an awfully small number, so we redo the calculation using logarithms:

$$18\times\ln(0.431)+11\times\ln(0.294)+10\times\ln(0.137)+4\times\ln(0.2)+\ln(0.18)+\ln(\frac{2}{3}) = -66.73.$$

Note that the orange M&M in the test data is ignored, since it does not appear in either training set.

We compute the Factory 2 score analogously to get -75.36. As $-66.73 > -75.36$, the classifier answers Factory 1.

1.6 The datasets

The datasets are contained in a zipped folder. The two text files `sports.txt` and `politics.txt` are 'snippets'—usually the first few sentences—of stories that appeared, respectively, in the Sports section and the US section of *The New York Times* between August, 2018, and January, 2019. These were obtained using an API tool available from the `Times` website. The files `nytimes.txt`,`nyp.txt`, `guardian.txt`,`wapo.txt` consist of headlines from *The New York Times*, *The New York Post*, `The Washington Post`, and `The Guardian`. These were contained in a much larger dataset, that included many more publications as well as the complete articles—not just the headlines—posted at

`https://www.kaggle.com/snapcrack/all-the-news`

All of these are ordinary text files, with one item (headline, or snippet) per line. I got a bit lazy with the Sports and Politics snippets, and did not take care to weed out a few weird characters that are sprinkled through these files—this does not appear to affect the results.

1.7 The provided code.

The function `get_data(filename)` reads in the text file and returns a Python list of the lines in the file. The function `text_transform(text)` takes a string and returns the list of words in the string, converted to lower case and purged of stop words and punctuation. For example, a call to `get_data('sports.txt')` returns a list whose fifth item is

> The remarkably successful Ohio State football coach is learning that tolerance for off-the-field problems is at an all-time low.

Applying `text_transform` to this item returns the list

> ['remarkably', 'successful', 'ohio', 'state', 'football', 'coach', 'learning', 'tolerance', 'offthefield', 'problems', 'alltime', 'low']

1.8 What you should do

Write a function `build_models` that takes two lists—lists of strings from the two classes (*e.g.,* Sports and Politics, or New York Times and Washington Post) to be used as training data— and returns a pair of dictionaries giving the logs of the smoothed probabilities for the words in the two classes.

Then write a function `score(message,model,prior)` that takes a headline and computes its score relative to the model (which will be one of the dictionaries returned by `build_models`). The argument `prior` is the value of $P(\texttt{Sports})$ or $P(\texttt{Politics})$ that you use.

Finally, write a function `evaluate(d1,d2)` that takes two lists returned by `get_data()`, separates each into a training list and test list, and calls `build_models` to create the probability models. Then, repeatedly (say, one thousand times), sample a random item from the combined test sets, and use `score` to determine which of the two classes it belongs to. Evaluate how well the classification performs. Submit both your code and a brief writeup describing the results.

How should you do this evaluation? Treat the smaller of the two classes (which would be Sports in the Sports and Politics datasets) as the 'rare trait' we are testing for. Thus we treat a Sports story as 'positive' and a Politics story as negative. As we run the classifier on each item in the test set, we compare the classifier result to the correct answer, and keep four counts:

- TP–the number of true positives. In the Sports/Politics example, this will be the number of Sports stories that the classifier says are Sports.
- TN–the number of true negatives. This will be the number of Politics stories classified as such.
- FP–the number of false positives: Politics stories classified as Sports.
- FN–the number of false negatives: Sports stories classified as politics.

It is a common practice in machine learning to employ a number of different statistics to evaluate the performance of this kind of binary classifier. An obvious one is the *accuracy,* the proportion of answers that are correct:

$$\text{accuracy} = \frac{TP + TN}{TP + FP + TN + FN}.$$

As our drug testing example shows, accuracy is not the whole story. For instance, a diagnostic test for a very rare disease that consists in saying 'You don't have it!' to every subject, could be 99.9% accurate, but it would fail to diagnose any actual instance of the disease that it encountered. For this reason, other measures are also computed. The *precision* is the proportion of positive test results that are correct.

$$\text{precision} = \frac{TP}{TP + FP}.$$

This was the failing in our example of testing senior citizens for marijuana use: The accuracy was high, but the precision was only around 50%. The *recall* is the proportion of positive instances that are correctly diagnosed:

$$\text{recall} = \frac{TP}{TP + FN}.$$

The 'accurate' diagnostic test that always answers negative has zero recall!

You should compute these statistics for the various pairs of datasets to which you apply your classifier.

2 What I found

By the way, I implemented the classifier to see what kind of results I got. It performed very well with the Sports vs. Politics example, and so-so results when comparing two newspapers, say Washington Post versus New York Post. Try this out for different pairs of newspapers. I did not try (and am not asking you to try) the version of the problem where the original data represents selections from more than two classes, (*e.g.,* four different newspapers, five different sections of the newspaper).