PY-599 (Fall 2018): Applied Artificial Intelligence
Naïve Bayes Classifier for Document Classification
Homework Assignment
Deadline to Submit: Tuesday 3:00 PM, Sept 25th (right before the Tuesday session)
Submission Method: Please share your Colab notebook
Group Submission is allowed

Implement a Naïve Bayes Classifier for Document Classification:

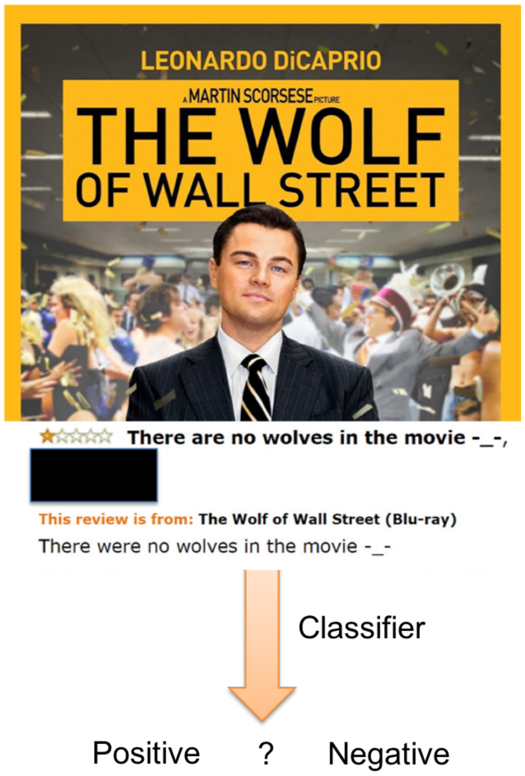
Document classification covers many different applications including, but not limited to:

- Spam/email detection
- Sentiment Analysis (is the document has a positive note or negative?)
- Subject identification of a document. Is the document about Math? History? Or Physics?
- Authorship identification? Who could be the author of this document based on the document's tone, choise of words, etc.?
- …

For a human, it is usually easy to read a document and determine for example whether the document has a positive tone or negative. But this is not a trivial task for a computer to accomplish.

How can we develop a computer program that reads a document to determine the tone (sentiment) of the document? As an example, imagine the document under examination is a movie review. How a computer program is supposed to decide the sentiment of a review?

One method would be we to equip the computer program with some rules to apply in order to make this decision. For example, we give the program a set of key sentences or words to look for in a document as the "flags" or "identifiers" of the sentiment. Although this can somewhat work, in practice the task of creating and maintaining such an inclusive and comprehensive set of rules is not an easy job. Granted you can come up with many "key sentences" or "expressions" that can be associated with a positive review or a negative review. But can you create a comprehensive rule set? And can you keep updating this set of rules over time as the reviewers might use different expressions to express themselves? This doesn't sound like a practical engineering solution to this task.



There are no wolves in the movie -_-,

[redacted]

This review is from: The Wolf of Wall Street (Blu-ray)

There were no wolves in the movie -_-

Instead of us giving the computer program all the “key sentences or flags” to use as rules to decide the sentiment of a review, we can let the computer *learns* on its own what a positive or a negative review IS by showing it a lot of labeled positive and negative reviews, and after learning the computer program will hopefully should be able to determine the sentiment of new unlabeled reviews on its own.

The dataset for reviews comes from IMDB. In the colab file that I shared with you, I introduced two methods to download this dataset. You can use Keras’ built-in function:

```
(x_train, y_train), (x_test, y_test) = imdb.load_data(path="imdb.npz",
                                                      num_words=None,
                                                      skip_top=0,
                                                      maxlen=None,
                                                      seed=113,
                                                      start_char=1,
                                                      oov_char=2,
                                                      index_from=3)
```

This will give you your training dataset x_train, which contains the reviews, and their labels y_train. After you train your classifier with x_train and y_train, you will test the performance of the classifier by giving it x_test, and compare the outputs it produce against the y_test. Count how many misclassifications your classifier make and report it. The error of the Bayesian Classifier that I developed was about 20%. This is what you can expect to get (plus minus a few points).

According to Keras website, “*for convenience, in Keras dataset, words are indexed by overall frequency in the dataset, so that for instance the integer "3" encodes* *the 3rd most frequent word in the data*." Do **not** confuse this with the frequency of the words among positive reviews and negative reviews. Words can have (and usually do have) different distributions between positive and negative reviews, and that is basically how we are going to do the classification. This means that we still need to calculate the frequency of words (here numbers) among positive reviews and the negative reviews separately.

You can also directly get the raw IMDB reviews that are not converted to numbers. There should be no difference between the logic of your program regardless of which dataset you use. The only difference would be how you get the data and read them into the arrays. Other than that, they are the same. You can find the dataset on Stanford group's website: http://ai.stanford.edu/~amaas/data/sentiment/. Download this file, unzip it, look at the folders and see how they are organized.

You can upload this zipped dataset to colab by this simple code and unzip it there:

```
from google.colab import files
uploaded = files.upload()
!tar xzvf aclImdb_v1.tar.gz >/dev/null
```

When you execute this code, a browser window will pop up, which guides you to locate and upload the zip file.

Let's get back to how to solve the machine learning problem.

To put this problem in probability theory language, given a review document *d*, the aim is to calculate $P(c_P|d)$ and $P(c_N|d)$ and decide the sentiment is Positive if $P(c_P|d) > P(c_N|d)$, but negative if otherwise. c_P and c_N represent positive sentiment class and negative sentiment class respectively.

So now the problem of sentiment analysis is formulated as two conditional probabilities that we have to calculate and compare. The question is how? How can we calculate these probabilities using the provided labeled, training reviews?

Bayes Theorem says that these two conditional probabilities can be rewritten as:

$$P(c_P|d) = \frac{P(d|c_P)P(c_P)}{P(d)}$$

$$P(c_N|d) = \frac{P(d|c_N)P(c_N)}{P(d)} \qquad (1)$$

This is much better! At the beginning we had $P(c_P|d)$ and $P(c_N|d)$ and we were not sure how to calculate them. But Bayes rule flips these conditional probabilities on their head. We now need to have $P(d|c_N)$, $P(d|c_P), P(c_N)$, and $P(c_P)$. These are probability functions that we can *try* to *estimate* from the provided, labeled reviews (the training data). Estimating $P(c_P)$, and $P(c_N)$ from the training data (labeled reviews) is easy. Just count the number of positive reviews and divide it with the total number of the reviews and it gives an estimate for $P(c_P)$, and repeat the same process for the negative reviews to obtain an estimate for $P(c_N)$.

Notice that since both conditional probabilities in equation (1) share the same denominator, *P(d*), you don't have to calculate *P(d)* in order to find the greater conditional probability. The one with greater numerator is going to be the one with higher probability. So we just need to calculate the numerator.

Furthermore, note that in our IMDB dataset, we have an equal number of positive and negative reviews among our training data. So, both $P(c_P)$ and $P(c_N)$ are going to be 0.5. So, there is no point in calculating the prior probabilities ($P(c_P)$ and $P(c_N)$) and you can only rely on likelihood functions, $P(d|c_N)$ and $P(d|c_P)$, to decide whether the review is positive or negative. As a result, the entire task comes down to estimating $P(d|c_N)$ and $P(d|c_P)$ from the training data.

Let's step back and look at these two conditional probabilities that we can estimate them. We need to estimate a probability distribution function for positive reviews, $\hat{P}(d|c_P)$ and a probability distribution function for the negative reviews, $\hat{P}(d|c_N)$. This task is easy and doable for low dimensional datasets. For example, in session 6, we had a 1-D data set for dogs and cats. The dataset contains the weights of the animals as a feature. We simply plotted and used histogram of the training data for the cats to obtain an estimate for P(weight|cat) and do the same for the dogs. You can find the code on the course website (Session 6: Probability and Statistics for AI & Machine Learning).

But here this task of estimating probability distribution function is not easy because we are dealing with a very high dimensional dataset. A document is a series of words. In this homework we are going to represent a document as a collection of words, where the order at which they appear in the document doesn't matter. This model of representation for a document is called the "bag of words." Let's assume $x_1, x_2, \ldots, x_n$ is the collection of *n* words appearing in document *d*. Each word belongs to the vocabulary, and of course we can have repetitive words in this set. We can rewrite the conditional probabilities that we like to estimate as following based on the bag of words representation:

$$P(d|c_N) = P(x_1, x_2, \ldots, x_n\,|c_N) \qquad (2)$$
$$P(d|c_P) = P(x_1, x_2, \ldots, x_n|c_P)$$

A document, for example a review, is composed of many words, hundreds or thousands of words. The size of the document in terms of the number of its words is the dimensionality of the document. As an example, the order of a documents that contain 100 words is 100. There are 100 degrees of freedom in this document. Each word can be anything from the vocabulary, and you have 100 of them in the document. Long story short, a document is a high dimensional data and estimating probability distribution of such high dimensional dataset from the training data is not an easy job (The Curse of

Dimensionality.) We need exponentially many more data as the dimensionality goes higher in order to obtain meaningful estimation.

How can we proceed? Let us make an assumption; a very rough and naïve assumption! Let's assume the words in a document are conditionality independent from each other given the class that the document belongs to:

$$P(x_1, x_2, \ldots, x_n \mid c_N) = P(x_1|c_N)P(x_2|c_N)\ldots P(x_n|c_N) \quad (3)$$
$$P(x_1, x_2, \ldots, x_n \mid c_P) = P(x_1|c_P)P(x_2|c_P)\ldots P(x_n|c_P)$$

The resulting classifier is named Naïve Bayes classifier because it is based on a naïve assumption. With this assumption our n-dimensional probability functions - that are very hard to estimate - are reduced to n separate 1-dimensional probability functions that we can easily estimate given adequate amount of training data. This is fantastic! But note that we accomplish this simplification using an assumption! Does this assumption always hold true? The answer is no. As a matter of fact, in most cases this assumption is not correct. However, this assumption gives us a path forward to estimate and approximate probability functions that otherwise we wouldn't be able to directly estimate or calculate. The results of Naïve Bayes classifiers can be sometimes surprisingly good even though the assumption was not true for that specific case. We tend to make this conditional independence assumption without assessing whether it is true or not, and proceed with it. At the end, we look at the performance of the classifier and we let it be the judge. And this is what we are doing in this homework. We "assume" this conditional independence assumption is true, and we go with it, and apply naïve Bayes classifier to classify the documents.

Now we need to estimate P($x_i|c_P$) and P($x_i|c_N$) for each word. We are going to do simple counting to obtain the frequency of appearance of word x_i in positive and negative reviews. Estimating these conditional probabilities for each individual word is easy and straightforward.

1- Count how many times word x_i shows up in positive reviews.
2- Count the number of words in all positive reviews. If the same word shows up multiple times keep counting it again and again.
3- Divide the result of step 1 by the result of step 2. This gives us $\hat{P}(x_i|c_P)$; an estimate for $P(x_i|c_P)$.

Repeat the process above for all words x_i. And then repeat the same for the negative reviews to obtain $\hat{P}(x_i|c_N)$.

Let us demonstrate this with a simple example. Assume below is the collection of all positive reviews (supposedly all written by the beloved Groot!):

I am Groot. I am Groot. I am Groot. We are Groot.

The conditional probabilities for each word would be:

$$\hat{P}(I|c_P) = \frac{3}{12}$$
$$\hat{P}(am|c_P) = \frac{3}{12}$$
$$\hat{P}(Groot|c_P) = \frac{4}{12}$$
$$\hat{P}(we|c_P) = \frac{1}{12}$$
$$\hat{P}(are|c_P) = \frac{1}{12}$$

Now imagine we are given a new review as below and our task is to decide whether it is positive or not according to the provided training data:

I can do this all day.

So we need to calculate P(*I can do this all day*|c_P). Naïve Bayes rule simplifies this as:
P(*I can do this all day*|c_P)≈ P(I|c_P) P(can|c_P) P(do|c_P) P($this$|c_P) P(all|c_P) P(day|c_P).

We have an estimate for P(I|c_P) from the training data. But what about the rest? In our new test reviews that we are trying to classify, we might encounter words that we had not observed in our training data. And their frequency is therefore going to be zero. And this is problematic because anything multiplied by zero is going to become 0, and therefore the entire probability is going to be zero just because of one unseen word in training positive reviews!
To solve this issue we do a simple trick (which is called Laplace smoothing). Always add one to the number of occurrences of any word in the reviews, and add *V,* the size of vocabulary to the denominator. This will insure that even when a word that had never appeared in the training data, its conditional probability would not become zero. Let's assume the size of vocabulary is 10000. After Laplace smoothing, our estimated probabilities from the collection of positive reviews would be:

$$\hat{P}(I|c_P) = \frac{3+1}{12+10000}$$
$$\hat{P}(am|c_P) = \frac{3+1}{12+10000}$$
$$\hat{P}(Groot|c_P) = \frac{4+1}{12+10000}$$
$$\hat{P}(we|c_P) = \frac{1+1}{12+10000}$$
$$\hat{P}(are|c_P) = \frac{1+1}{12+10000}$$

And the probability of unseen new words in the new review would be:

$$\hat{P}(can|c_P) = \frac{0+1}{12+10000}$$
$$\hat{P}(do|c_P) = \frac{0+1}{12+10000}$$

$$\hat{P}(this|c_P) = \frac{0+1}{12+10000}$$
$$\hat{P}(all|c_P) = \frac{0+1}{12+10000}$$

$$\hat{P}(day|c_P) = \frac{0+1}{12+10000}$$

As a result, the probability of the new review being positive will become:

P(*I can do this all day*|c_P)= P(*I*|c_P) P(*can*|c_P) P(*do*|c_P) P(*this*|c_P) P(*all*|c_P) P(*day*|c_P)

$$=\frac{3+1}{12+10000}\times\frac{0+1}{12+10000}\times\frac{0+1}{12+10000}\times\frac{0+1}{12+10000}\times\frac{0+1}{12+10000}\times\frac{0+1}{12+10000}$$

But now we are going to have another issue. We are multiplying very small numbers, and the result an underflow to zero will occure. To solve this problem, we use another trick. We calculate log(P(*I can do this all day*|c_P)) instead, resulting:

log(P(*I can do this all day*|c_P))=$\log(\frac{3+1}{12+10000})+\log(\frac{0+1}{12+10000})+\log(\frac{0+1}{12+10000})+\log(\frac{0+1}{12+10000})+\log(\frac{0+1}{12+10000})+\log(\frac{0+1}{12+10000})$

You can implement Naïve Bayes for document classification in Python with many different techniques, data structures, modules, etc. But choosing the right data structure and method can make life much easier. I would strongly suggest you write your own code instead of trying to follow mine. It is simpler to write your own instead of trying to complete my code. That being said, I am showing some parts of my code in order to give you an idea of how some important bits and pieces of this code can work. I followed the second method to download IMDB dataset, therefore I was working with real actual words. You still can use the code below even if you used the Keras dataset. Your words are going to be numbers though.

I combined all positive reviews in one string, named str. I split str to a list of word and stored them in `wordListPos` using the instruction below:

```
import re
wordListPos = re.sub("[^\w]", " ",  str).split()
```

I used the *Collection* module and *Counter* function that comes with it.

Please see the website for *Collection* module to see how to use it.

```
import collections
cntPos = collections.Counter()

for word in wordListPos:
  cntPos[word] += 1
```

This piece of code counts and keeps records of how many times different words occur in the long string that contains all positive reviews.

Make sure that your Indentation is correct for "for loop block" when you are coping it into colab.

This is what Python gives me when I print cntPos

```
Counter({'the': 150000, 'and': 86484, 'a': 80152, 'of': 75899, 'to':
65951, 'is': 56763, 'br': 49234, 'in': 46986,…
```

The nice thing about this Counter object is that you can use words as an index and it will return back how many times that word has occurred in the long string of positive reviews. For example, the instruction below:

```
print(cntPos["good"])
```
gives me:

```
7344
```

meaning that the word "good" occurred 7344 times among all positive reviews.
You can do the same for the negative reviews, and get `cntNeg`.

Now let's jump forward and see how we can calculate P(d|c_P) for a test review. In Keras dataset, test reviews are x_test. In this code the test review is stored in a list named `this_review`:

```
wordListReview = re.sub("[^\w]", " ",  this_review).split()
ReviewWordCount=len(wordListReview)
cntReview = collections.Counter()
  for word in wordListReview:
     cntReview[word] += 1
  ReviewKeys=cntReview.keys()
  LoglikelihoodPos=0
  for word in ReviewKeys:
    LoglikelihoodPos=
LoglikelihoodPos+cntReview[word]*math.log((float(cntPos[word])+1)/(word
_count_Pos+len(Voc)))
  LoglikelihoodNeg=0
  for word in ReviewKeys:
    LoglikelihoodNeg=
LoglikelihoodNeg+cntReview[word]*math.log((float(cntNeg[word])+1)/(word
_count_Neg+len(Voc)))
```

Couple of quick notes:

1- When a word is repeated *k* times in a test review, of course we are going to add the conditional probability of that word *k* times when we are in the log world. That is why you can see "`cntReview[word]*`" term.

2- `cntPos[word]`says how many times a word showed up in the positive reviews and `cntNeg[word]` says how many times in the negative reviews. And we divide them with (`word_count_Pos`), the total number of words in all positive reviews, or (`word_count_Neg),` the total number of words in all negative reviews.
3- We add 1 to the numerator and add Voc, the size of vocabulary, to denominator for the purpose of Laplace smoothing. You can simply assume `Voc=10000.`

Then we check which likelihood is greater, `LoglikelihoodPos` or `LoglikelihoodNeg.` If `LoglikelihoodPos` is greater, it means that the Naïve Bayes says the review is positive, otherwise it says negative. Now compare the result with the actual labels of the test reviews, which are stored in y_test. Count how many misclassifications your naiive Bayes make. Report it as your error rate. We expect an error rate of 20% for this method and for this dataset (plus minus a few points.)