# Project3 Report

Xueying Sui 1001682442

## Problem

We need to use Naive Bayes implement text articles classification. We have data sets, we use the half of data sets as training data, another half of data sets as test data. Firstly, we use the training data to train the Naive Bayes classifier, and then use test data to evaluate the classifier and get the accuracy.

## Data

We use the data set of NewsGroup Data
http://www.cs.cmu.edu/afs/cs/project/theo-11/www/naivebayes.html
(Newsgroup Data)
This data set include 20 groups of news, each group has about 1000 new texts. For train a Naive Bayes classifier, we use the first half of each group(about 500) of news is used as a training set, and the second half(about 500) is used as a test set.

## Method

### Load data(Common step)

The core code of this step:
We have training set and test set are both in one data file, so we just use glob method to get the name of all the texts, then put these files name into a list. Now we have a list that include all the files name, so we just need to read the text name and open it if we need to load the text.

### Process data and calculate

For Naive Bayes classifier, we use Y represents the group, xi(i=1,2...) represent the features of test text. Then our goal is calculate $P(Y = y_i \mid X = x)$. We can rewrite as:

$$P(Y = y_i \mid X = x) = \frac{P(Y = y_i, X = x)}{P(X = x)}$$

Now we just need to calculate $argmaxP(Y = y_i, X = x)$. We know that:

$$P(Y = y_i, X = x) = P(Y = y_i) \prod_{j=1}^{n} P(X^j = x^j \mid Y = y_i)$$

After determining the target, we need to process the training data of each group. Because this is text classification, we must first segment the text and get every word of this text(remove stop words). After we get the word sets of every group, we can calculate the probability of each word. Next we need to segment the test text and get every word, calculate the conditional probability of each word and multiply them. The group that has the biggest probability product is the prediction group.
Next I will show the two method that I applied. The first method is not correct, but it has a reference to the second method, so I also listed it.

### (1) Method 1: naive method

As the first attempt, when I process the training set, I just put the word into a dictionary and record it's appearances. In this step, I put the words that belong to the same group into the same dictionary. At last, I get 20 dictionary, each dictionary contains all words and occurrences in this group, except for stop words.
Then I use the same way to process the test text and calculate the probability for each test text. **Then I get a bad accuracy, just about 21%**. Consolation is that it is one percent higher than random guesses.

Then I found the problem, in the whole process, I only care about the probability of the words in this group, I ignored the probability of words in other groups. For example: there is a word ‘nice’, has a high probability of this group, but if it also appears in all other groups, **this phenomenon indicates that the word does not represent such text well and should be reduced in weight.**

Finally, I found a method names tf-idf, the idea of this method is also consider the frequency of occurrence of this word in this group and the number of occurrences in all other texts.

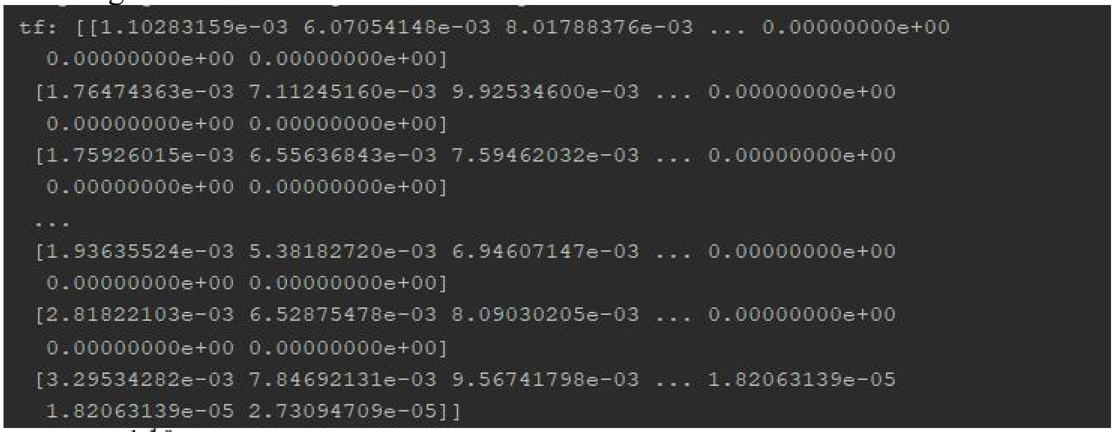
**(2) Method 2: improved method(tf-idf)**

In this method, I put all the words of all the training set(include all the training texts of all the groups) into a list as a general words dictionary.

We use $tf_{i,j}$ represents the frequency at which the word i appears in the text j, actually, we can directly calculate the frequency at which the word i appears in the group j:

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}}$$

We can get tf matrix after calculate:

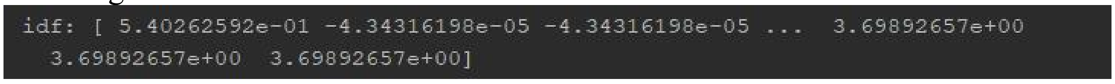


```
tf: [[1.10283159e-03 6.07054148e-03 8.01788376e-03 ... 0.00000000e+00
  0.00000000e+00 0.00000000e+00]
 [1.76474363e-03 7.11245160e-03 9.92534600e-03 ... 0.00000000e+00
  0.00000000e+00 0.00000000e+00]
 [1.75926015e-03 6.55636843e-03 7.59462032e-03 ... 0.00000000e+00
  0.00000000e+00 0.00000000e+00]
 ...
 [1.93635524e-03 5.38182720e-03 6.94607147e-03 ... 0.00000000e+00
  0.00000000e+00 0.00000000e+00]
 [2.81822103e-03 6.52875478e-03 8.09030205e-03 ... 0.00000000e+00
  0.00000000e+00 0.00000000e+00]
 [3.29534282e-03 7.84692131e-03 9.56741798e-03 ... 1.82063139e-05
  1.82063139e-05 2.73094709e-05]]
```

We use $idf_i$ represents the inverse document frequency of word i(D is the total number of training texts):

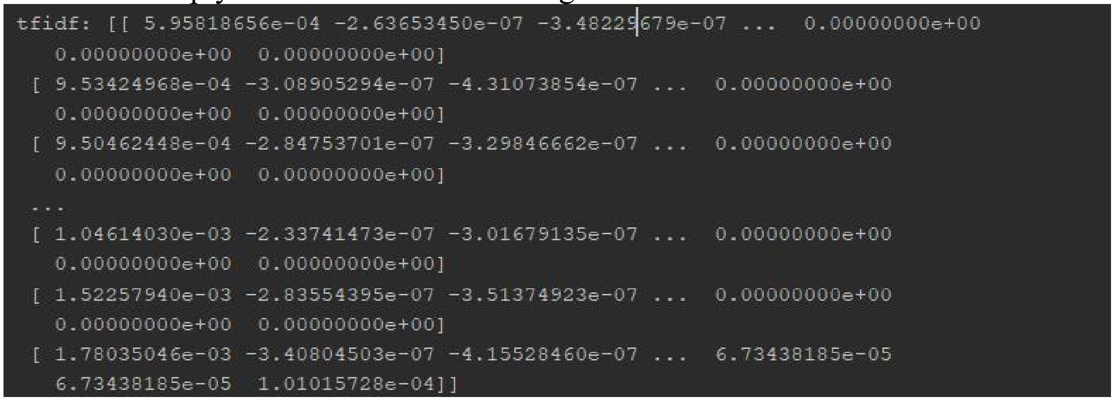
$$idf_i = log\frac{|D|}{|\{j : t_i \in d_j\}|}$$

We can get idf matrix after calculate:



```
idf: [ 5.40262592e-01 -4.34316198e-05 -4.34316198e-05 ...  3.69892657e+00
  3.69892657e+00  3.69892657e+00]
```

Then we multiply tf and idf as the final weight of one word.



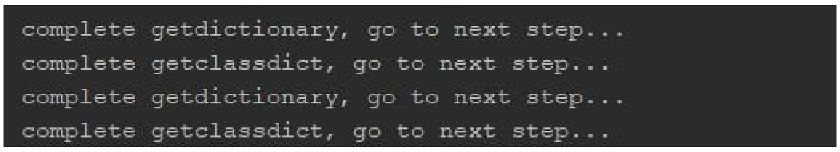
```
tfidf: [[ 5.95818656e-04 -2.63653450e-07 -3.48229679e-07 ...  0.00000000e+00
   0.00000000e+00  0.00000000e+00]
 [ 9.53424968e-04 -3.08905294e-07 -4.31073854e-07 ...  0.00000000e+00
   0.00000000e+00  0.00000000e+00]
 [ 9.50462448e-04 -2.84753701e-07 -3.29846662e-07 ...  0.00000000e+00
   0.00000000e+00  0.00000000e+00]
 ...
 [ 1.04614030e-03 -2.33741473e-07 -3.01679135e-07 ...  0.00000000e+00
   0.00000000e+00  0.00000000e+00]
 [ 1.52257940e-03 -2.83554395e-07 -3.51374923e-07 ...  0.00000000e+00
   0.00000000e+00  0.00000000e+00]
 [ 1.78035046e-03 -3.40804503e-07 -4.15528460e-07 ...  6.73438185e-05
   6.73438185e-05  1.01015728e-04]]
```

Now we can process the test text and make prediction according to the results. We know that we should multiply the probability of each word in the test set, but to avoid

underflow, we use the log function to process the probability result, in this process, we should add 1 to each probability result and then log them, in order to prevent the probability from being 0.
Now we complete the whole process of data process and calculate. Then we can get the accuracy of test texts.

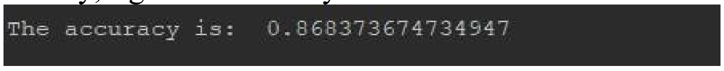
## Results

This is a long data processing process, I ran for about 10 hours or so. At the beginning of the procedure, there will show some progress tips and intermediate results, like this:



```
complete getdictionary, go to next step...
complete getclassdict, go to next step...
complete getdictionary, go to next step...
complete getclassdict, go to next step...
```

Finally, I got the accuracy of test texts:



```
The accuracy is:  0.868373674734947
```

The accuracy is about 86.8%.
I think this is a god result compare to my initial result. I believe I still need to improve operational efficiency and accuracy. I found the ‘stopwords’ file online, there are some common stop words, like “am”, “is”, “are” and so on. Maybe I can improve the accuracy through create a more efficient stop words.