Implementation:

1. **Pre-processing the data:** In this step I am removing all the unnecessary words like **stop words, Xref, Path, From** and other **header information** in the document. I am also removing the digits, non English characters and also the special characters from the documents used for training. The primary reason to remove these words is they don't add much value to the classifier for classifying various documents and the second reason is to make the dictionary smaller which helps to **improve efficiency** of the classifier. Note: For the stop words I am using a small list named **"stopwords.txt"**.
2. **Creating a Word count dictionary:** I am creating a separate dictionary of word counts for each of the 20 class labels.
3. **Removing the common words in all the labels:** After obtaining the dictionaries for each class label. I am deleting the set of all common words from each class label dictionary.
4. **Apply Naïve Bayes Rule with an assumption on Laplace smoothing**: The assumption I made is that the laplace smoothing numerator alone can be used instead of the original equation. I made this assumption because denominator is the "**number of unique words + number of repetitions of each word in the new document in all the 20 class labels**". This value is huge when compared to the numerator making the probability approximately 0(something like e^(-100)). Also I am using log for the probabilities so that the numerator doesn't explode to **+infinity**. Hence I made that assumption. Using this equation I am calculating the naïve bayes equation which is **p(Class Label = 'c' | New Document) = P(New Document | Class Label='c') * P(Class Label = 'c')**. Applying log on this equation will result in **log(p(Class Label = 'c' | New Document)) = log(P(New Document | Class Label='c'))+log(P(Class Label = 'c')).** Finally, the equation will become, in **log(p(Class Label = 'c' | New Document)) = log(P(Word1, W2, W3,…….,Wn | Class Label='c'))+log(P(Class Label = 'c'))** . This will turn into **log(p(Class Label = 'c' | New Document)) = log(P(Word1 | Class Label='c'))+ log(P(W2 | Class Label='c'))+ log(P(W3 | Class Label='c'))+……+ log(P(Wn | Class Label='c'))+log(P(Class Label = 'c')).**

Code Structure:

- A function to pre-process the data named "**preprocessWord(word)**": This function will clean the word and remove all the special characters from it.
- A function to fetch word count dictionary from a document named "**extractDictionary(root,files)**": This function will get the unique word counts from the preprocessed document. **Note:** The dictionaries formed are separate for each category(Class Label).
- Shuffle the files. (I've done this to avoid biased distribution problems which come by not shuffling the data well.)
- Read all the train files and then form dictionary from it. (Using 50% of the data as train)
- Test on unseen data. (Using 50% of the data as test)

Results:

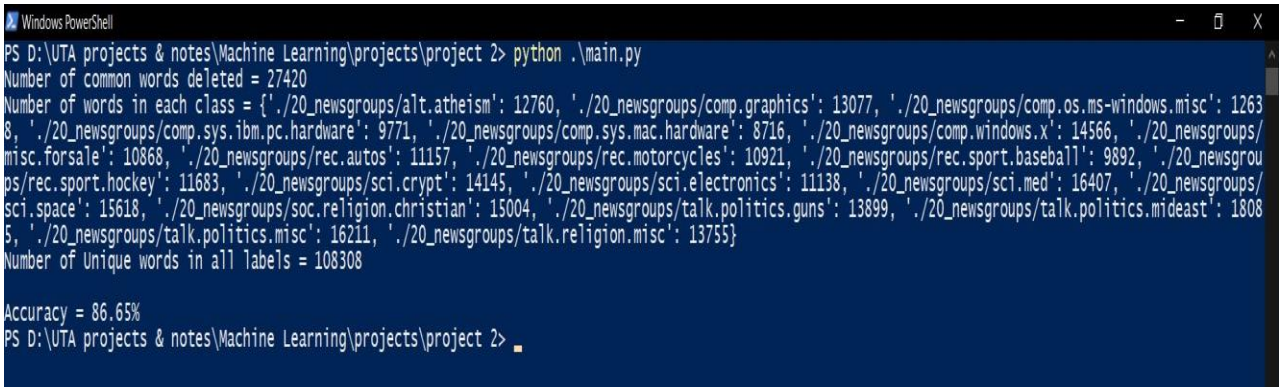
Obtained Accuracy of 84.09%.

Observations made:

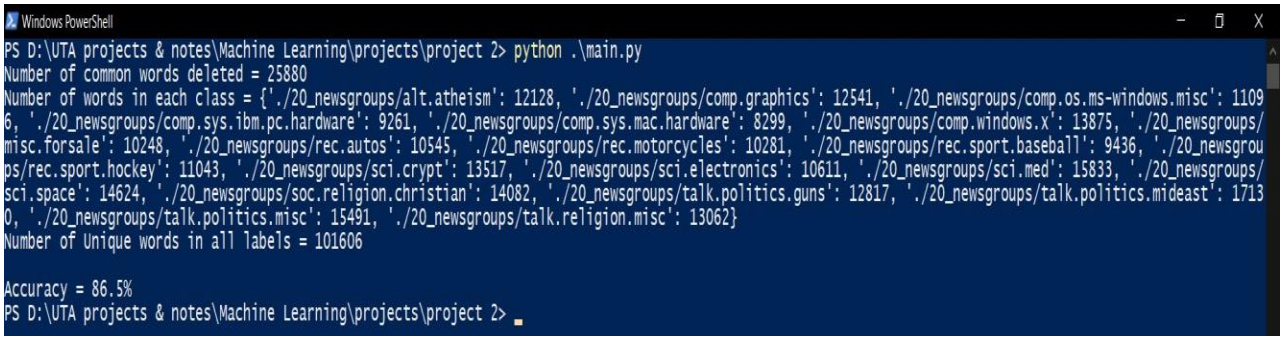
- I have also noticed that all the incorrect predictions are not totally bad. My model predicts **comp.windows.x as comp.os.ms-windows.misc, comp.os.ms-windows.misc as comp.sys.ibm.pc.hardware, sci.crypt as sci.electronics** and so on. The point is these categories all have something in common i.e., **comp.graphics, comp.os.ms-windows.misc, comp.sys.ibm.pc.hardware, comp.sys.mac.hardware, comp.windows.x** all belong to the **"comp"**(computers) category. Similarly, rec.motorcycles, rec.baseball, rec.hockey, rec.autos belong to the "**rec"** cateogry and so on. Model predictions are not very misleading, they are somewhat correct, rather than predicting something totally different. For example: Predicting **"sci.space"** as **"rec.autos"** which are unrelated at all.
- Removing the frequent words improved the accuracy of the classifier. Because few words like **"okay","yes"** etc don't add much value to the classification but are there in the documents. Such words can be removed for better results.
- Removing the common words from the class label dictionaries improved the accuracy by a lot. I checked the common words present in all the 20 dictionaries (20 because there are 20 class labels) and they came to be around **933**. Means **933*20=18660** words removed from the entire word dictionary. The reason for removing all these common words is they don't have much information to distinguish the between various categories(Class Labels). This boosted the accuracy of the classifier by a large percent(13-15%).
- I noticed the importance of Preprocessing the data in this project. There were a lot of un-necessary words and symbols which don't have a meaning with the class label. Without preprocessing the data the classifier was so inefficient and inaccurate in making predictions. The words that I had to store without preprocessing was extremely large making the computation very slow.

Tweaking the code for obtaining various results:

1. Providing 10% of the data for testing and 90% of data for training improved classifier accuracy by 2.65%. But there is a huge increase in the number of words that we have to store in order to get this 2.65% accuracy improvement. **Note**: Here, the variable test_size = 0.1



2. Providing 20% of the data for testing and 80% of data for training improved classifier accuracy by 2.5%. But there is a huge increase in the number of words that we have to store in order to get this 2.5% accuracy improvement. Also, notice that the number of words to accuracy ratio is not very wide when compared to test_size=0.1(10%). There is only a subtle difference in the accuracy, but the number of words we had to store is significantly lesser. **Note:** Here, the variable test_size = 0.2



3. Providing 30% of the data for testing and 70% of data for training improved classifier accuracy by 2%. But there is a huge increase in the number of words(Around 25k words) that we have to store in order to get this 2% accuracy improvement. **Note**: Here, the variable test_size = 0.3

Note: All the above modifications can be easily done. Please look for the variable at the top of the "main.py" file. Under the comment tweakable variables on line 5, there is a variable test_size = 0.5 (Because in description it was mentioned to keep 50% for test and remaining 50% for train). You can change this value to whatever proportion (In between 0 to 1) you want and it will fetch the results accordingly.

4. Setting up deleteThreshold improved computing time of the program but reduced the accuracy by a bit (like 3-5 %). There are 2 variables with names **deleteThresholdUpperBound, deleteThresholdLowerBound.** These variables remove the words (which are greater the upper bound or lesser than the lower bound). I tried to remove all the variables which appeared only once in the entire class label(only on the train data, of course) and noticed that the number of words for the entire dictionary came around 1000. Removing more than 1,00,000 words, this decreased the accuracy by a lot. The accuracy when I did this was 3-4%(which is pretty bad). So, I commented the logic for this. The upperBound threshold was useful to remove greater frequency words (say words repeated more than 100 times). This helped in improving efficiency and also accuracy few times.

Important Notes:

- For detailed explanation of what the code is doing look for the comments in the **"main.py"** file.
- The project was entirely based on computation time, memory usage & accuracy tradeoff.
- I used the word **'categories'** and the word **'class labels'** interchangeably.
- My program takes around 2-3 minutes to execute.