**Uncovering Sentiments using EDGAR Datasets**

# **Group 6**

## **Data source:**

Call transcript (Q4 2018 for the following stocks Google(1), Amazon(2), Facebook(3), Netflix(4), Microsoft(5), Tesla(6), Walmart(7), Kroger(8), Goldman sachs(9), NVIDIA(10)).

**Manual Labeling data**

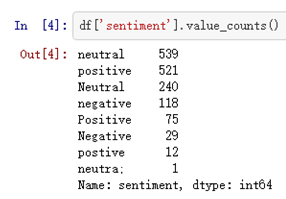
How we define sentiments(positive/negative/neutral):

Positive: if one paragraph talks something good to/about the company

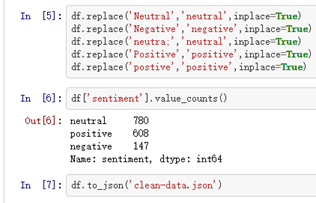
Negative: if one paragraph talks something bad to/about the company

Neutral: neither any of above.

**Data cleaning**

****

There are lots of mistakes in our dataset, so before we do experiments, we need to clean data first.



## **Experiments:**

### Experiment 1:

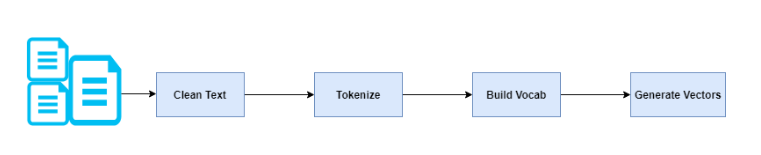
In this part, we used ​NLTK​ package, and mainly applied English stop words.

#### Bag Of Words:

Bag of Words (BOW) is a method to extract features from text documents. These features can be used for training machine learning algorithms. It creates a vocabulary of all the unique words occurring in all the documents in the training set.

it’s a collection of words to represent a sentence with word count and mostly disregarding the order in which they appear.

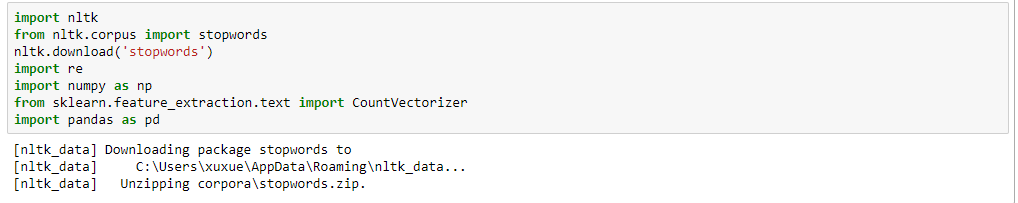
At a very high level, the diagram below shows the steps that you have to go through in order to implement the bag of words model.



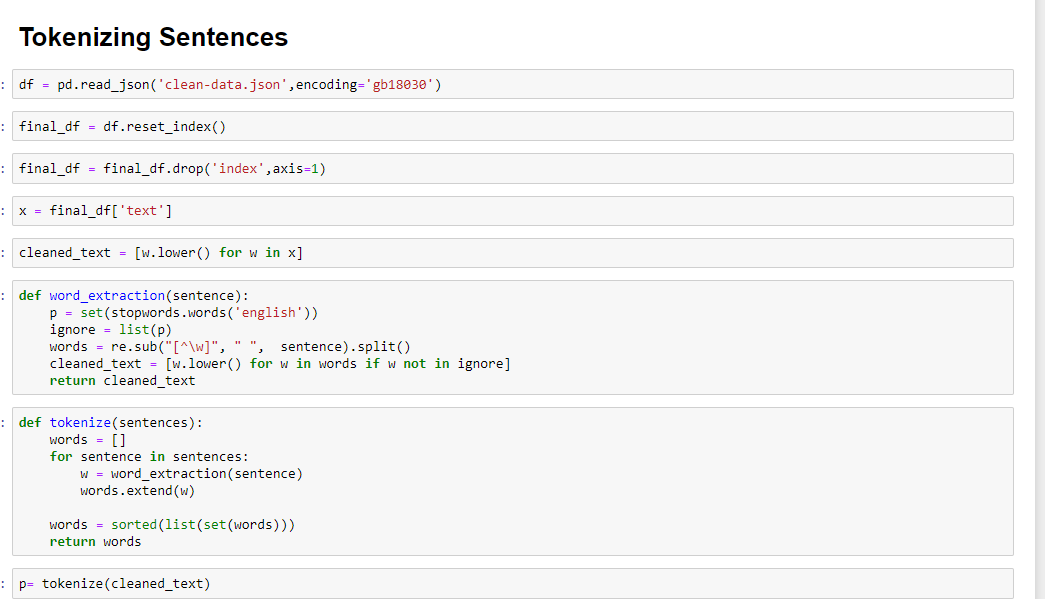
In theory, here is how it works for sentiment analysis.

* Imagine that you have a dataset broken down into sentences and that each sentence has a sentiment associated with it. In this case the sentence becomes your input feature (X) while the sentiment becomes the target variable (Y).
* We remove stopword from each sentence. Stopwords are words which do not contain enough significance to be used without our algorithm. We did this using the nltk python library.
* We tokenize each sentence by breaking up a sequence of strings into pieces such as words, keywords, phrases, symbols and other elements called tokens.
* We construct a document vocabulary by extracting all the words from all sentences. With this step, we can be able to create a set of all unique words that occur in the whole document. The output is an array whose size is equal to the size of all unique words that occur in the document.
* For each sentence in the original dataset, we create an array whose size is equal to the size of the array created in step 4 above. As you can see, each sentence was compared with our word list generated in Step 4. Based on the comparison, the vector element value may be incremented. For every sentence in the dataset, the vectors generated form a set of features which becomes our X variable, while the sentiment became our Y variable. The vector was used in several machine learning algorithms to make predictions.
* The snapshots below show the breakdown of our experiments for the Bag of Words approach.

Importing libraries



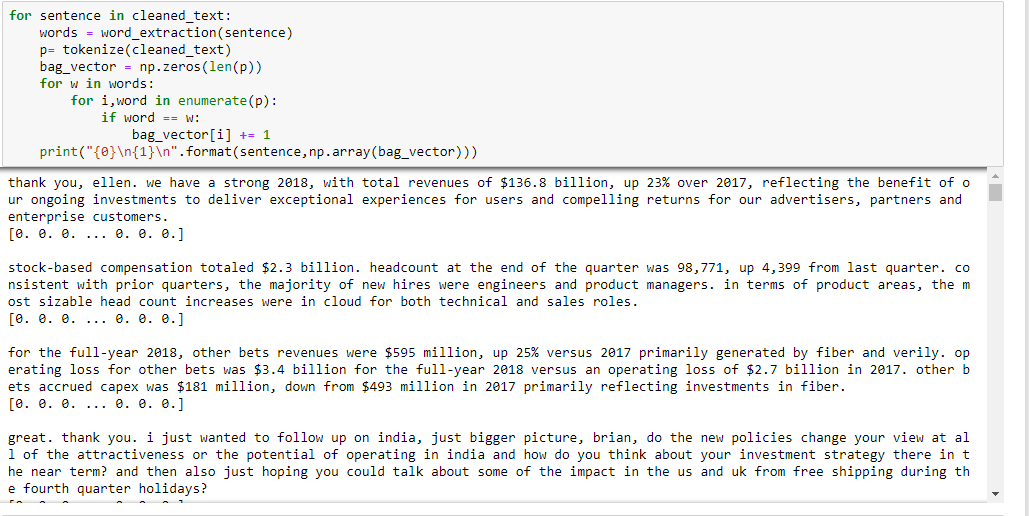
Loading and tokenizing the dataset



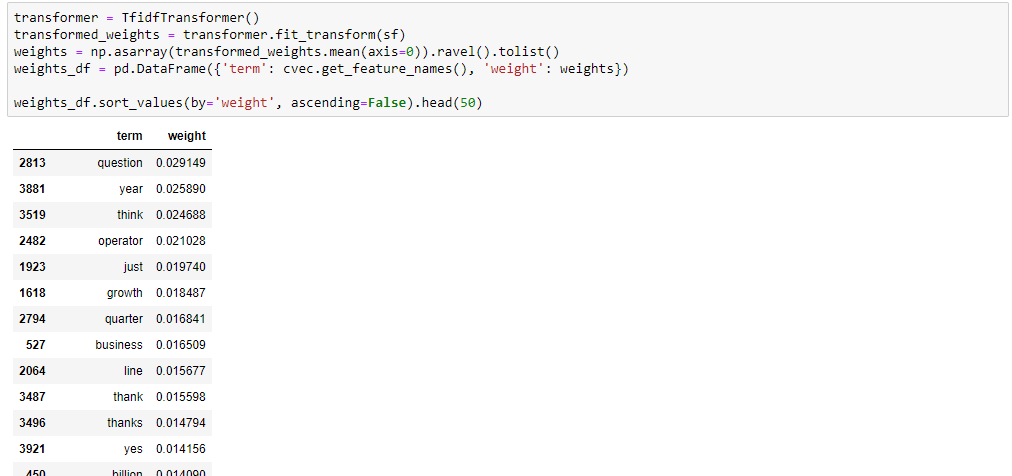
Sample unique vector (Bag of words)



Sentence tokenization

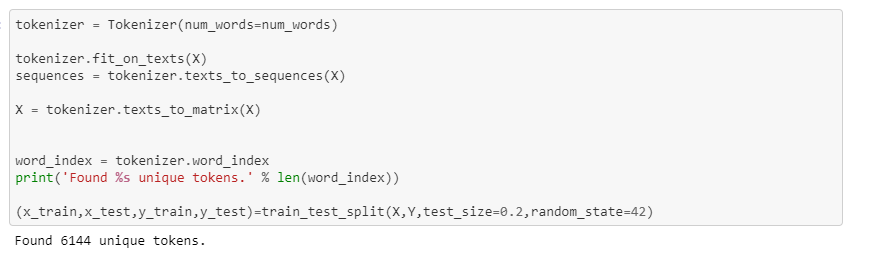


Computing the weights of the words in the document



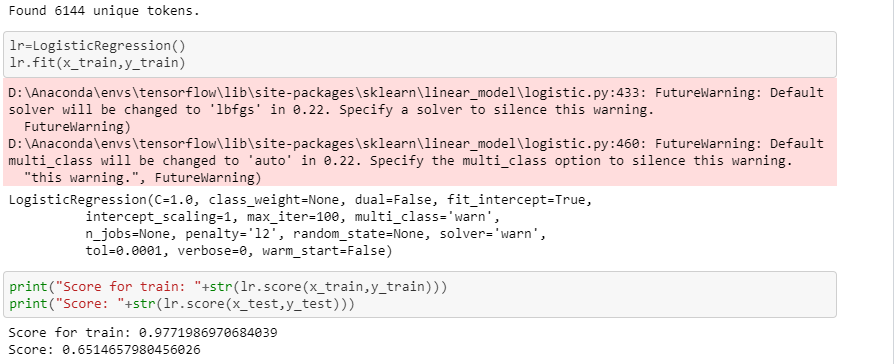
Applying models to the vector generated using the bag of words approach;

After tokenization, we split the dataset into both the training and testing datasets in the ratio of 80:20 as shown below;

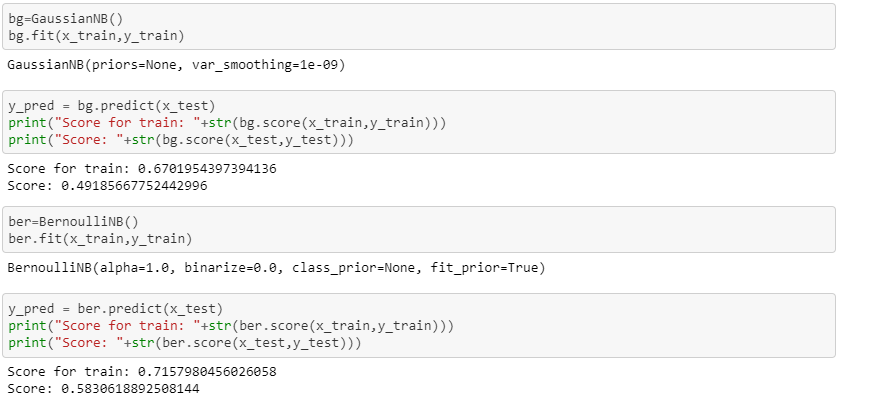


We applied several models to the dataset and the results that we got are shown below;

Using the Logistic Regression model, the testing accuracy was about 65%;



Using the gaussian and bernoulli models, the accuracy was 49% and 58% respectively



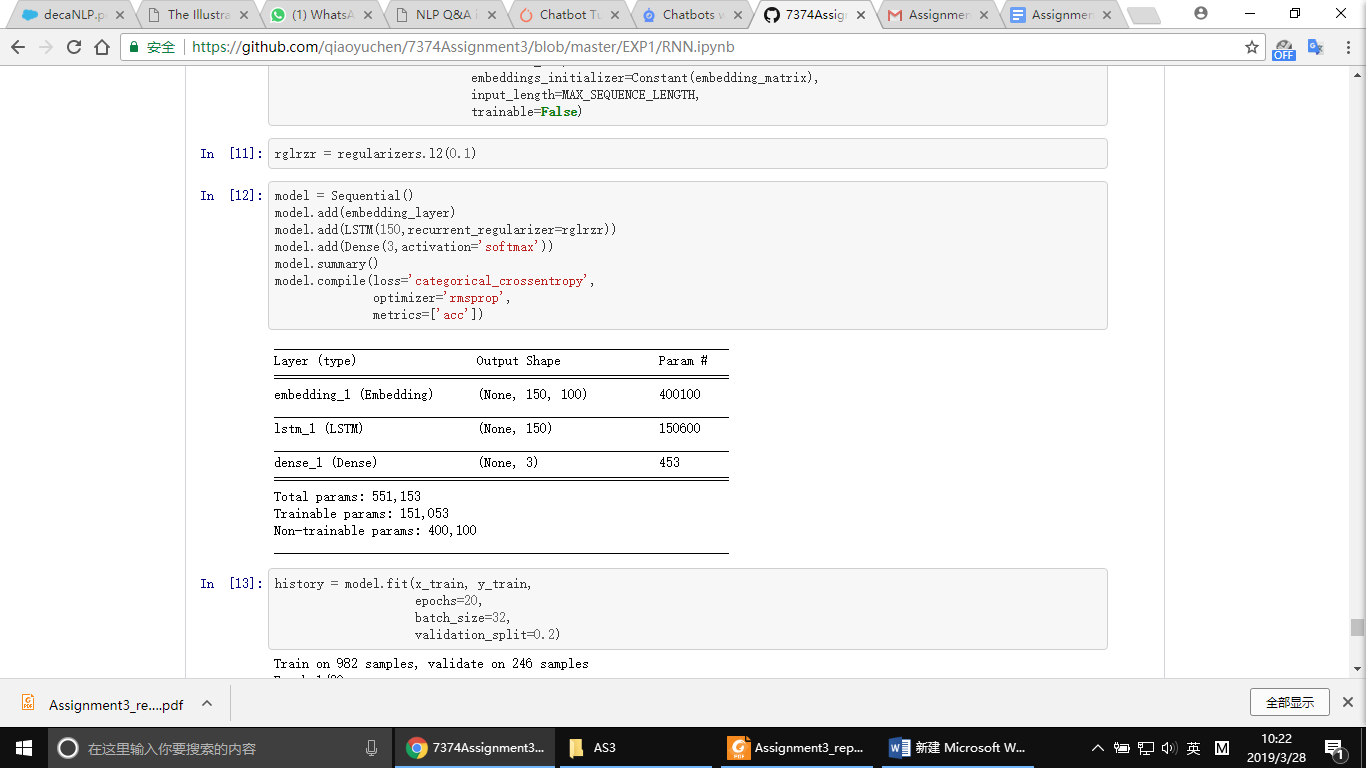
The confusion matrices for the 3 models above are shown below;

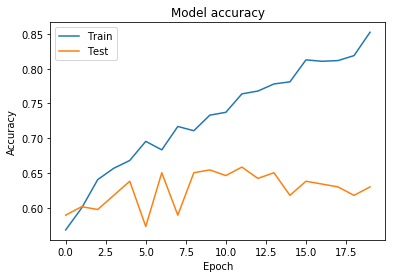


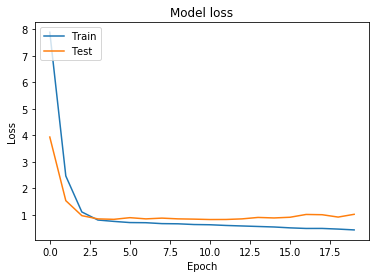
**Limitations of Bag of Words**

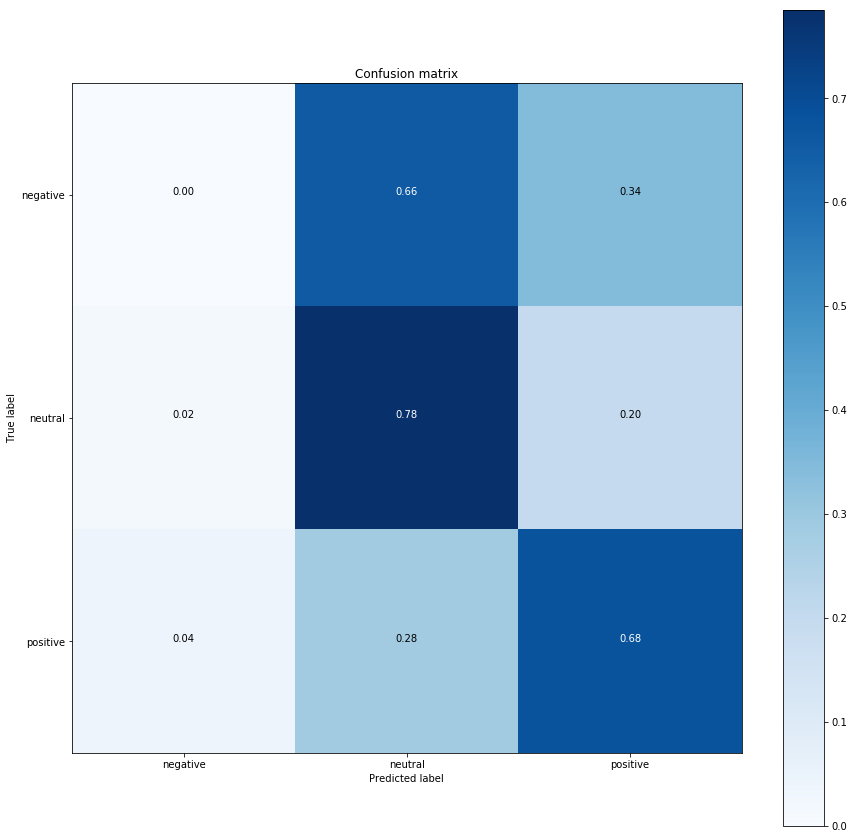
* Semantic meaning: the basic BOW approach does not consider the meaning of the word in the document and it completely ignores the context of the words, as such the meaning is lost.
* Vector size: For a large document, the vector size can be huge resulting in a lot of computation and time.

#### RNN

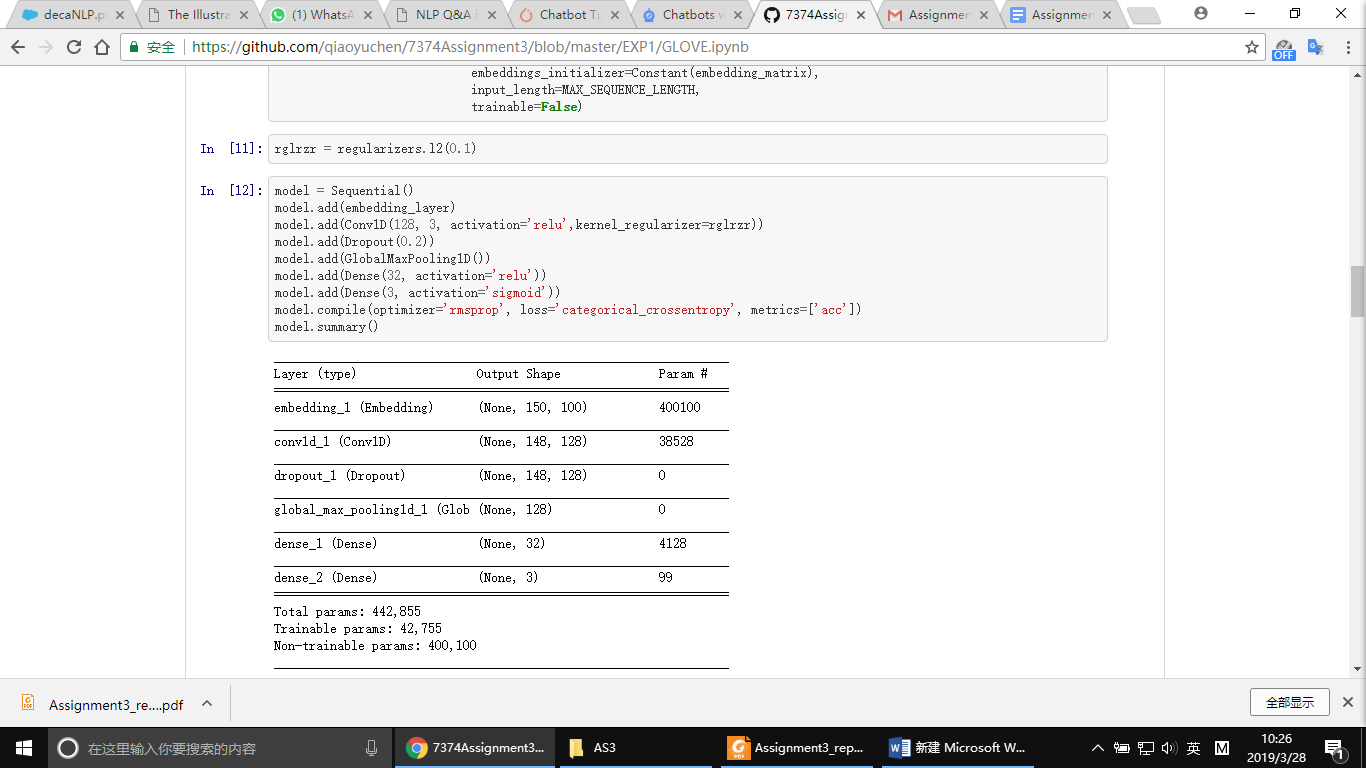


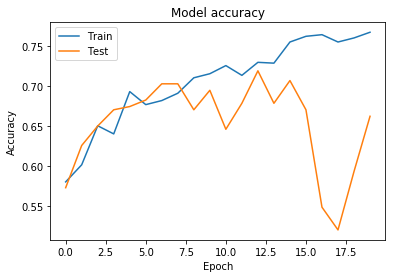


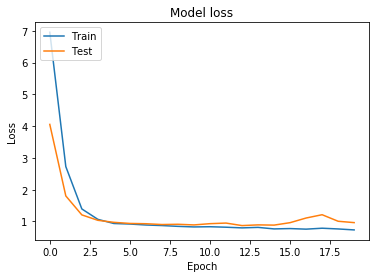


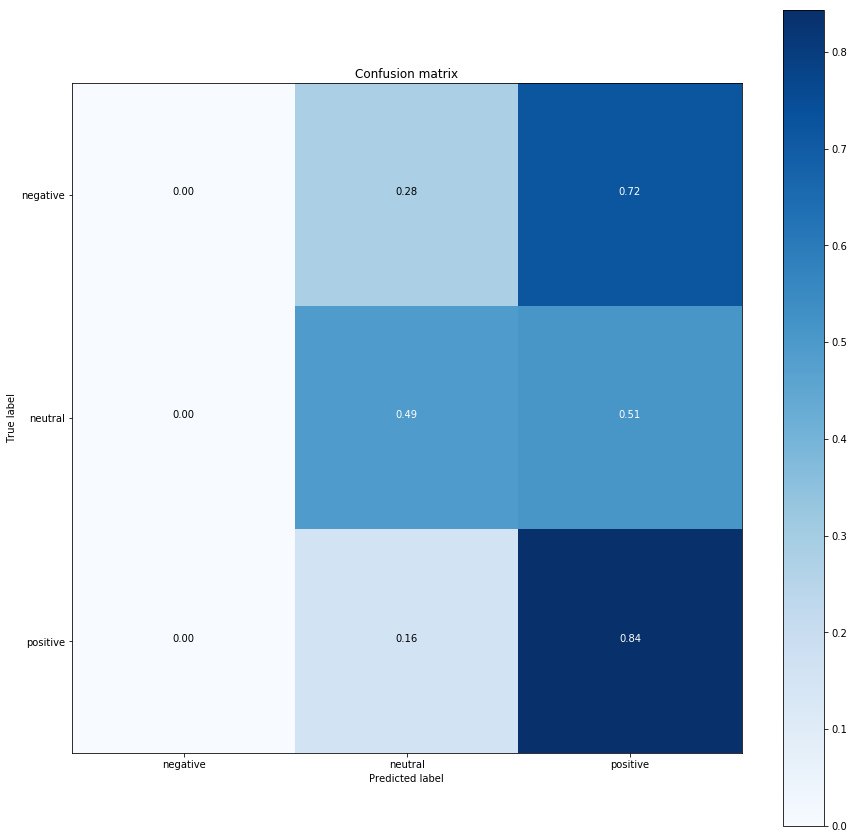


#### GLOVE









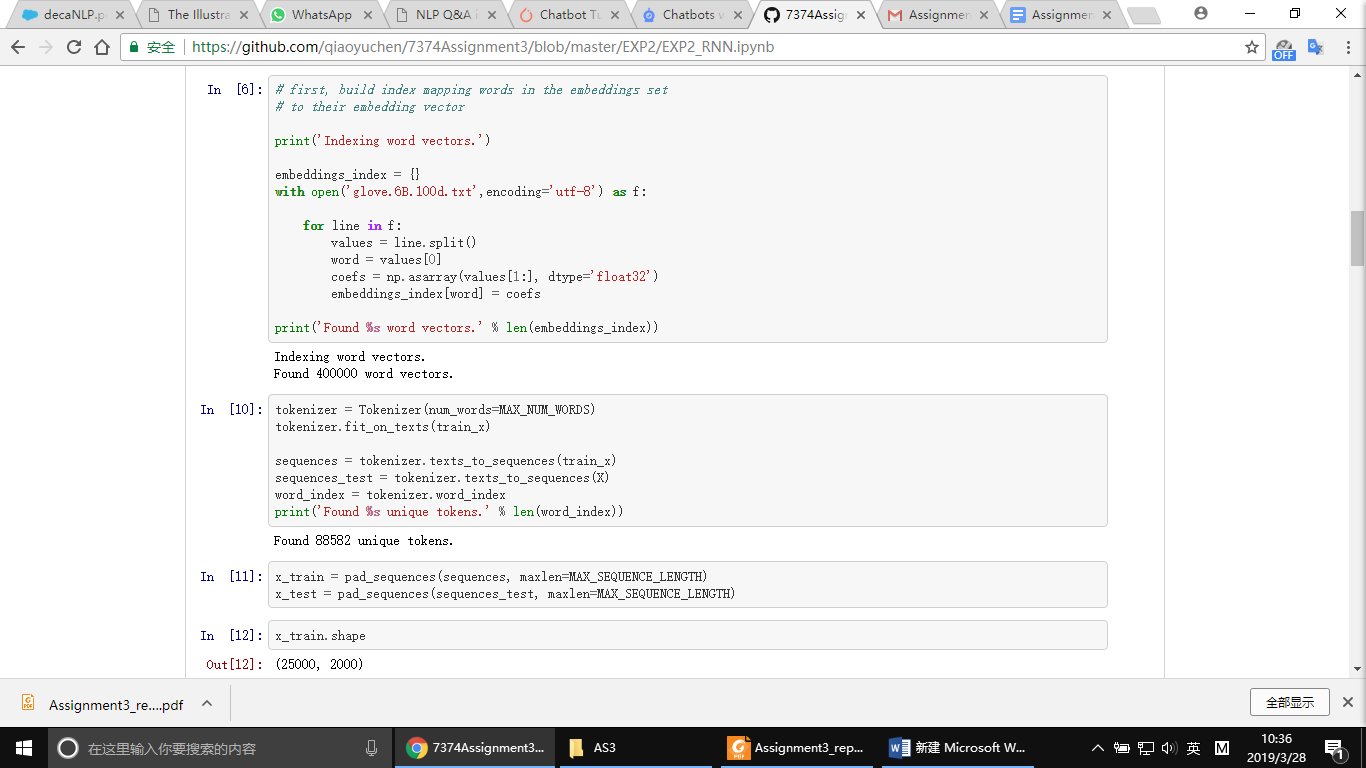
### Experiment2:

#### BOW

#### RNN

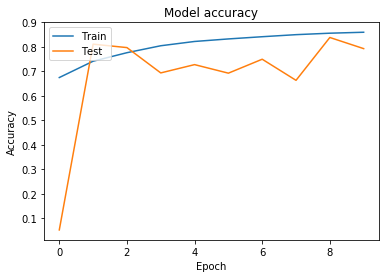
RNN(LSTM) on IMDB dataset, then train the mode with transcripts data.

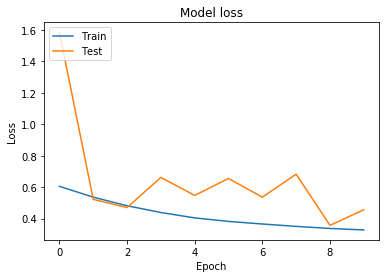
Padding to categorical:

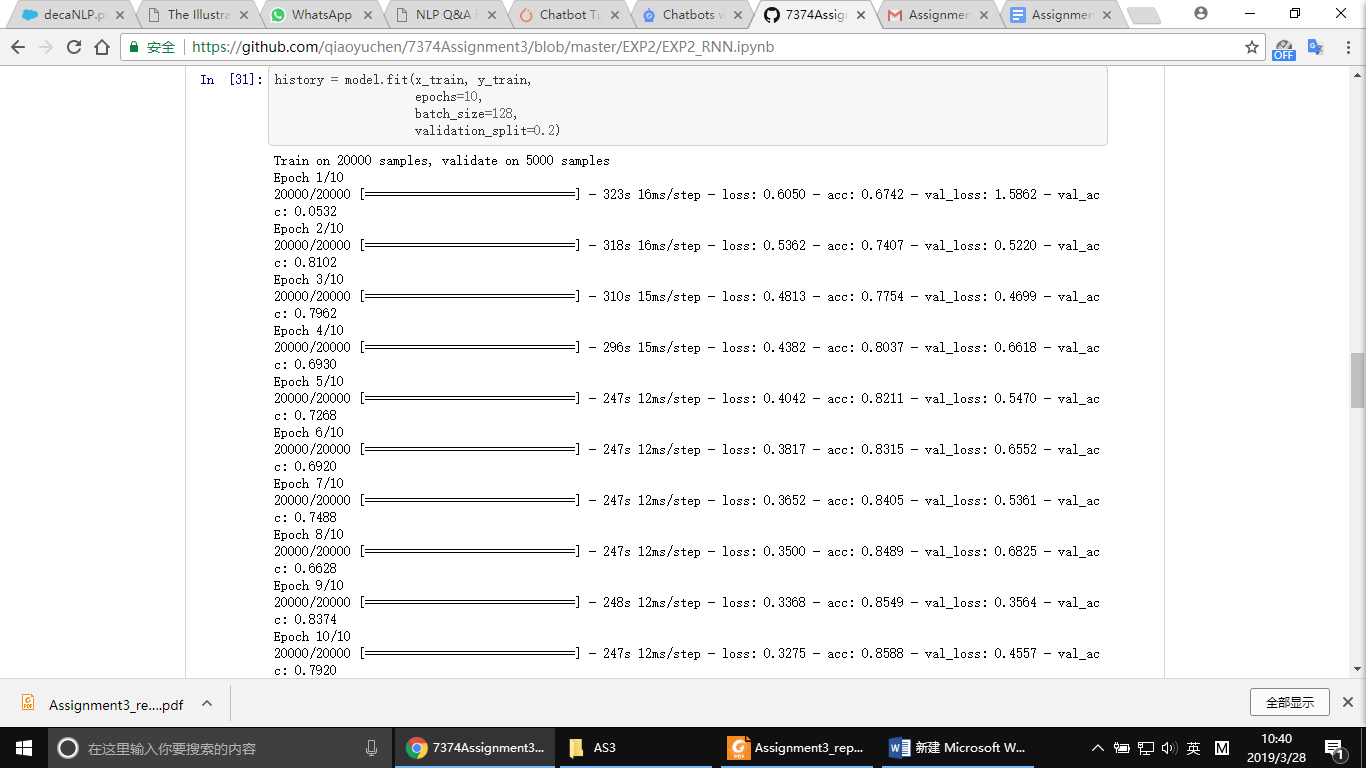


Build LSTM model:



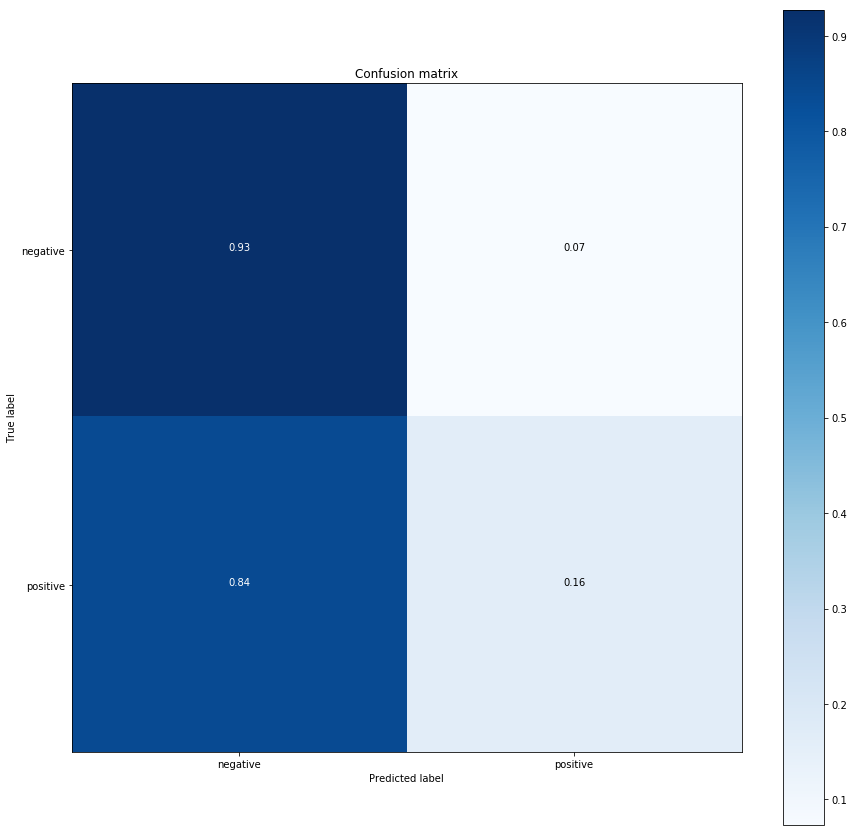






The accuracy is 0.792

Because there is no ‘neutral’ class in imdb dataset, we decided drop all those neutral sentiment in our dataset.



#### 

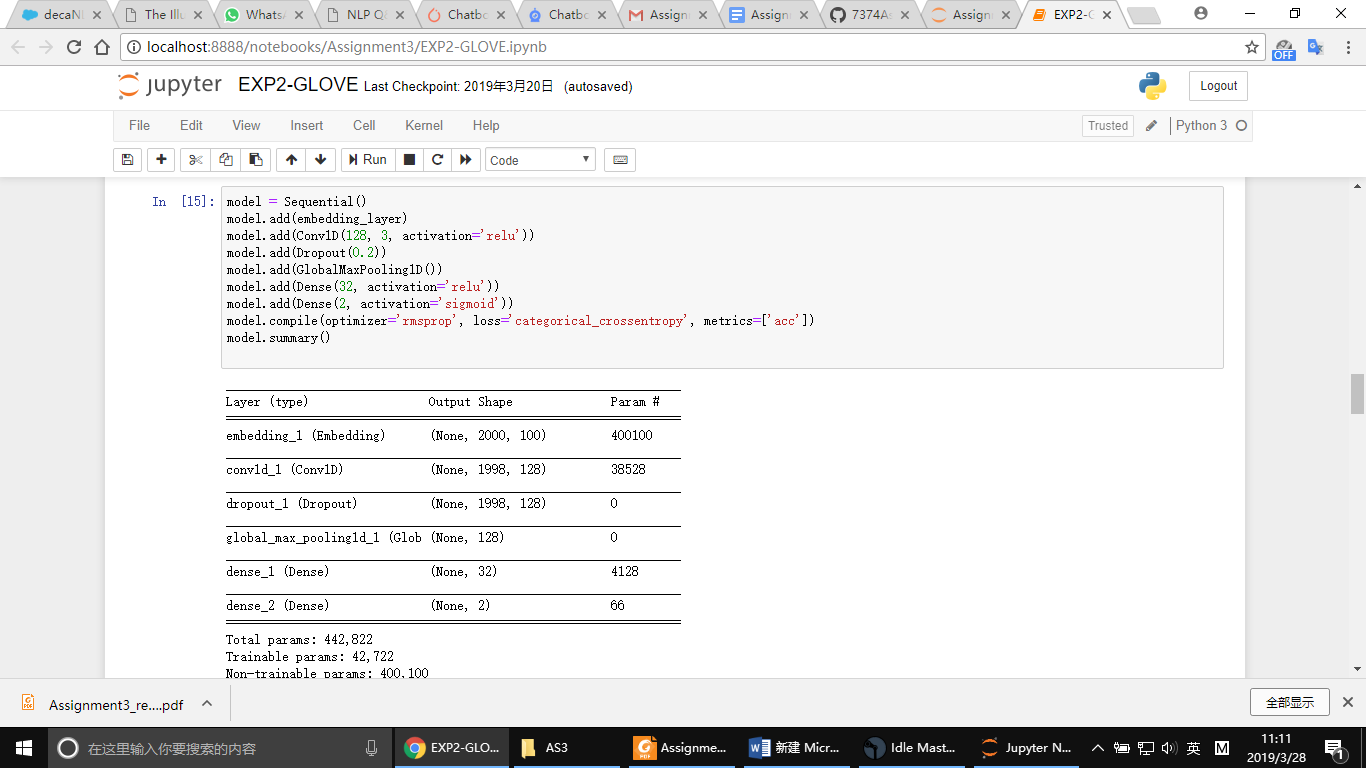
We used ‘glove.6B.100d.txt’ to pretrain the model.

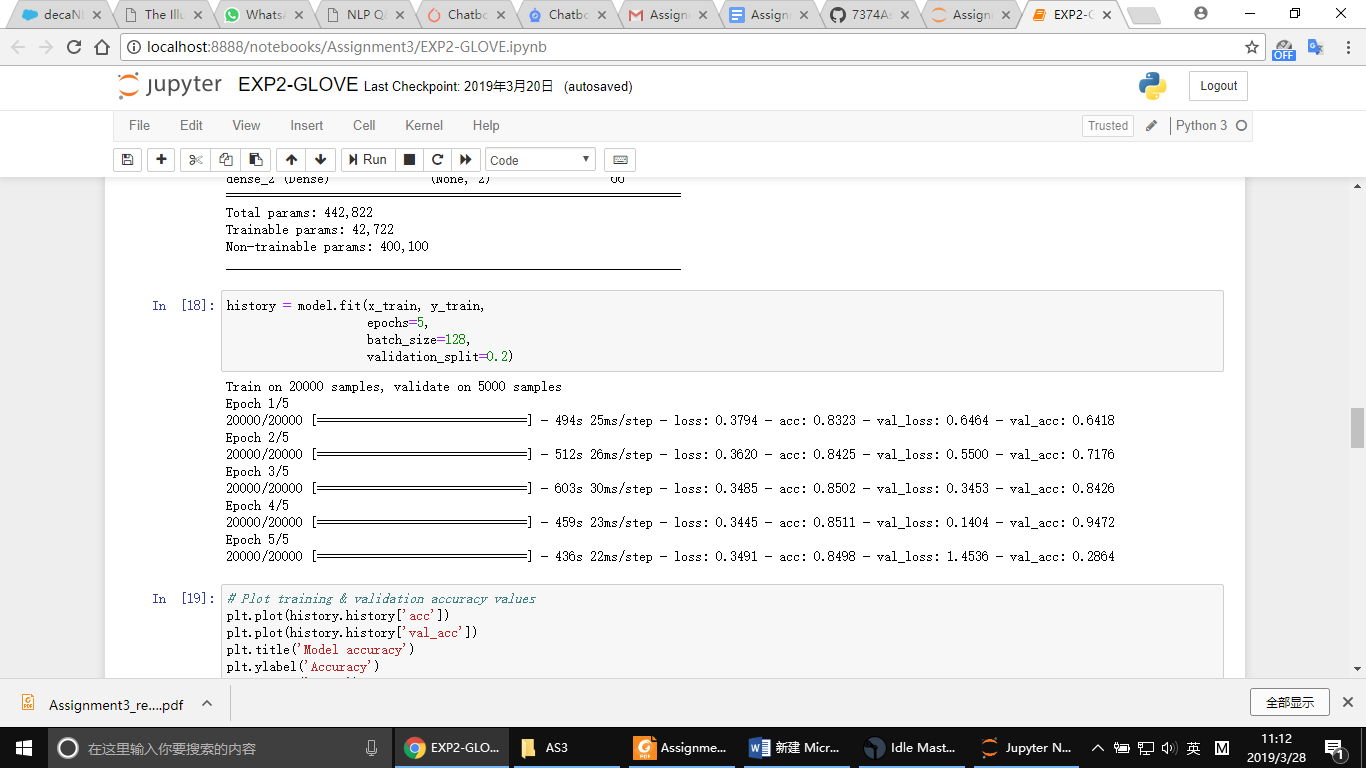
#### 

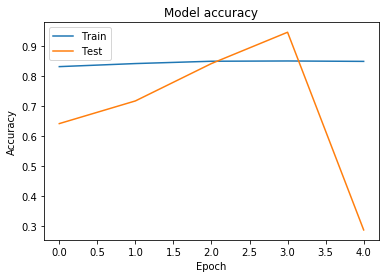
#### 

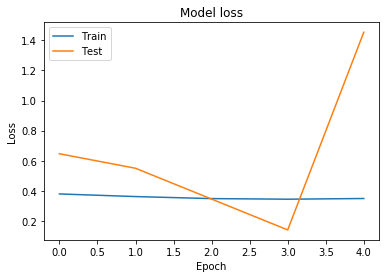
#### GLOVE

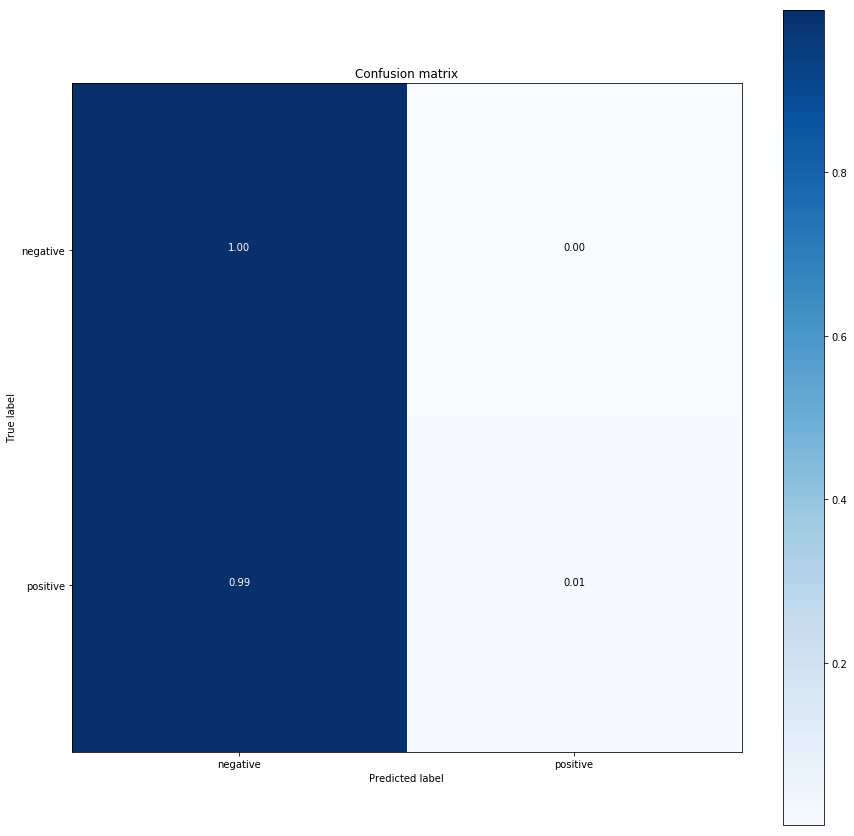
For this part, we used a simple neural network to perform the classification task. Then we used imdb data to train the simple network.







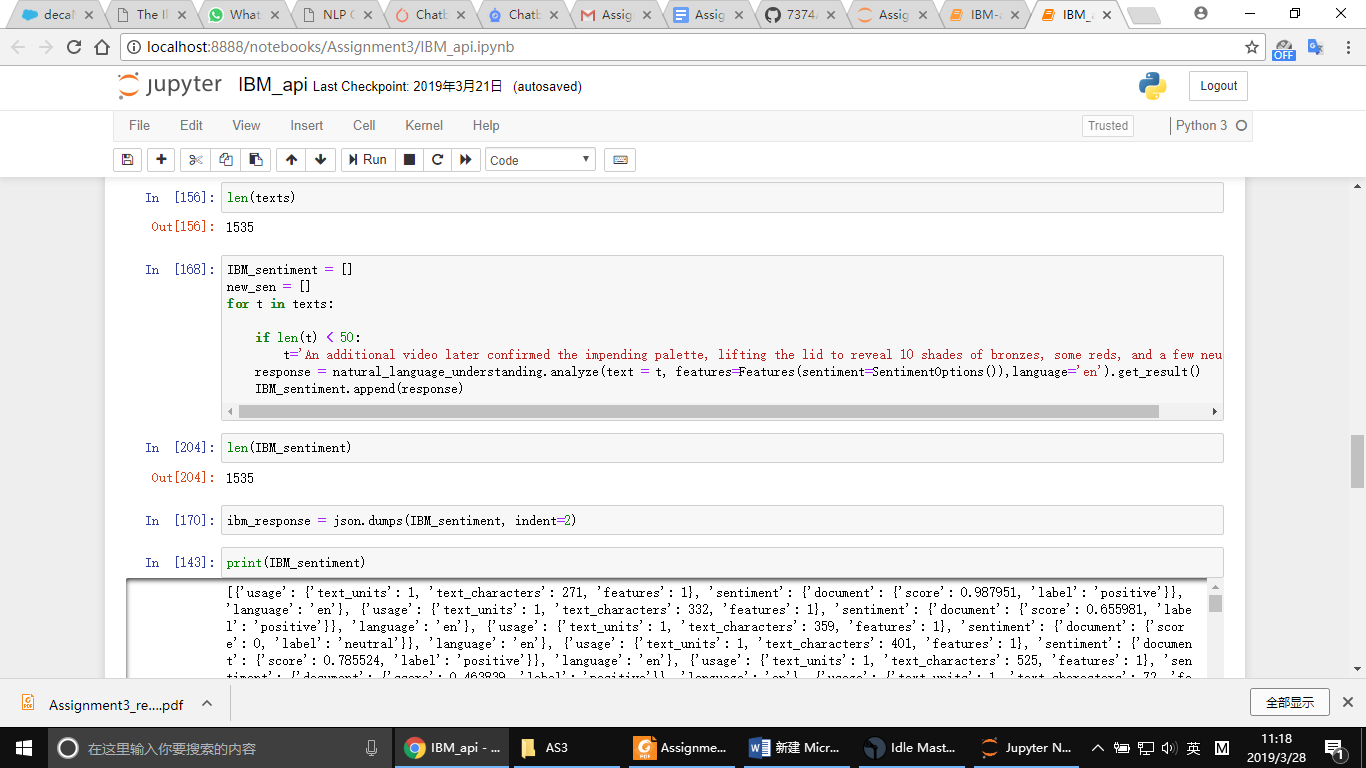




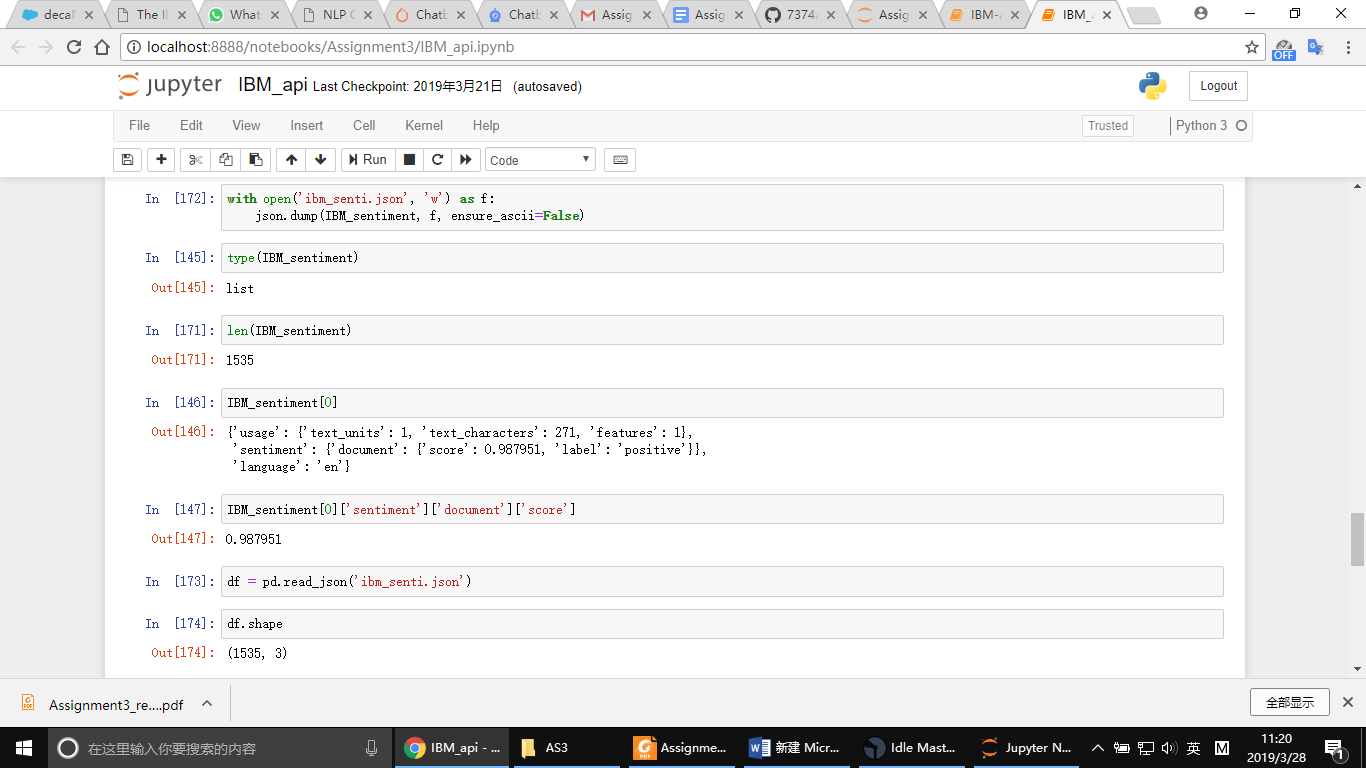
### Experiment 3:

In this part, we get sentiments by using Cloud APIs and calculate the confusion matrices for them.

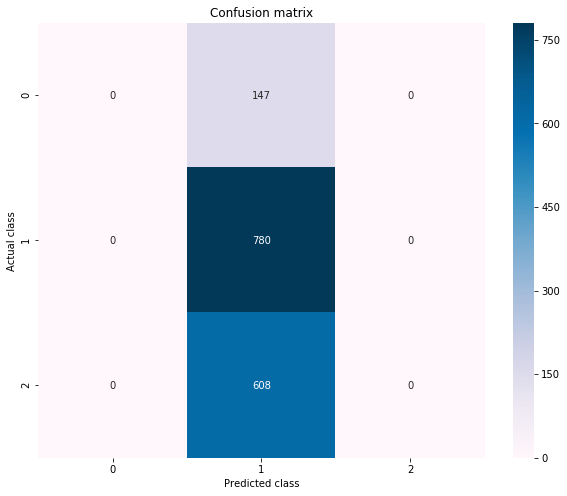
#### IBM api



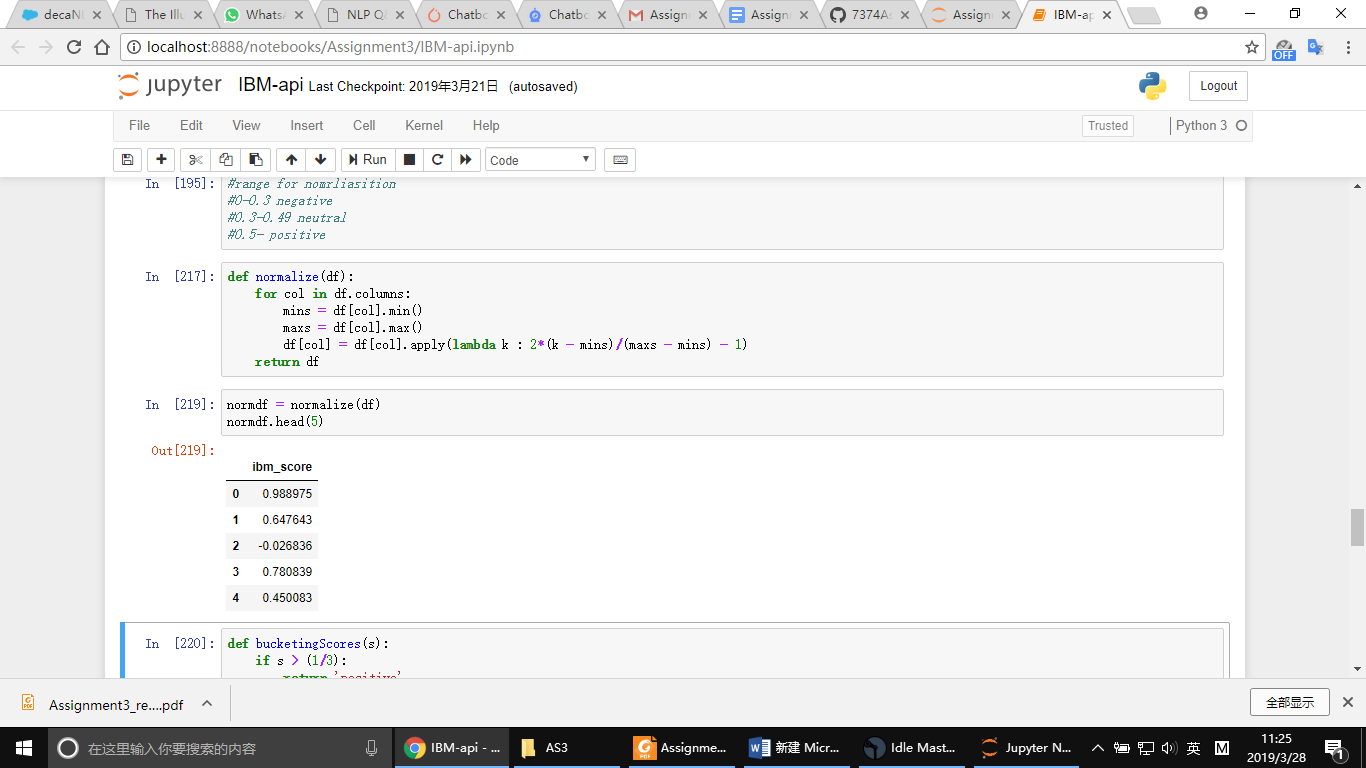
The sentiment scores of IBM are between -1 to +1 and it also gives the final sentiment label.



The confusion matrix as follow:



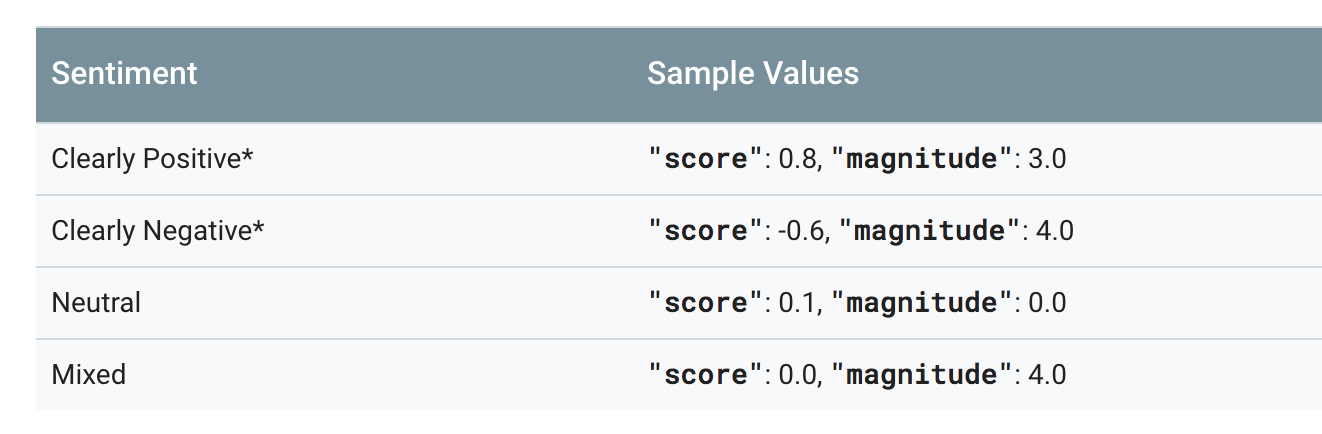
Normalization



#### Google API

Google Cloud Natural Language API is one of its suite of Machine Learning (ML) APIs that exposes powerful models that are just a REST call away. The Cloud Natural Language API helps us with text analysis. It can help us identify the entities in the text, sentiment analysis on the whole text and individual entities found in the text, syntax analysis, text content classification and more.

The API returns the Sentiment as a set of 2 values : **score** and **magnitude**. The score is a value between **-1.0** and **1.0**, with -1.0 indicating very negative and +1.0 indicating very positive. A value of 0.0 is a neutral sentiment. Each score is associated with a **magnitude value.** A higher value of magnitude indicates that the strength of the sentiment. The table below indicates how you could use the two values together to interpret the sentiment in a more meaningful and actionable manner.

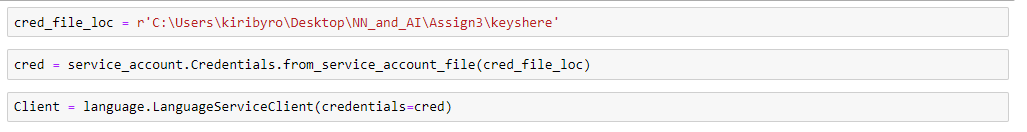


After signing up and enabling the API, we built a python script and we accessed the API using access keys as shown in the experiment below;

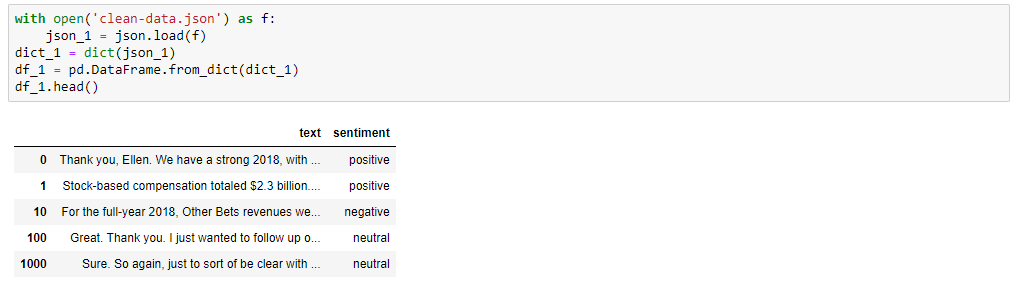
1. We imported important libraries;



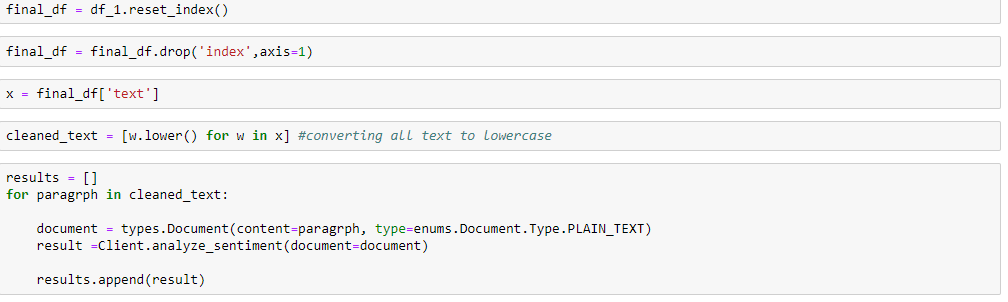
2. We used the security keys provided when we signed up to use the API to login and access the platform. The setup is shown below



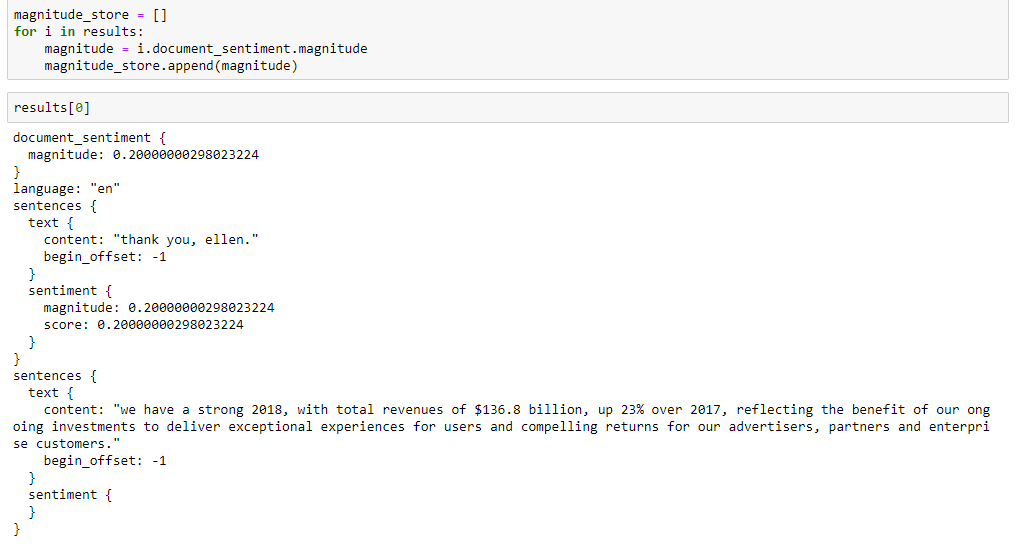
We read in the dataset and visualized the top 5 rows as shown below;



After cleaning the dataset, we allied google’s natural language API to our dataset as shown below;



The results are inform of json output. The result is in this format for every sentence is the dataset

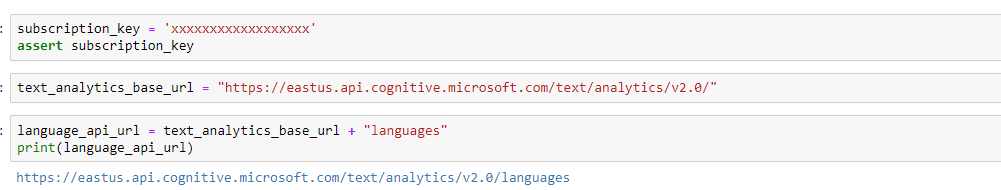


We saved the results as a CSV file for further analysis.

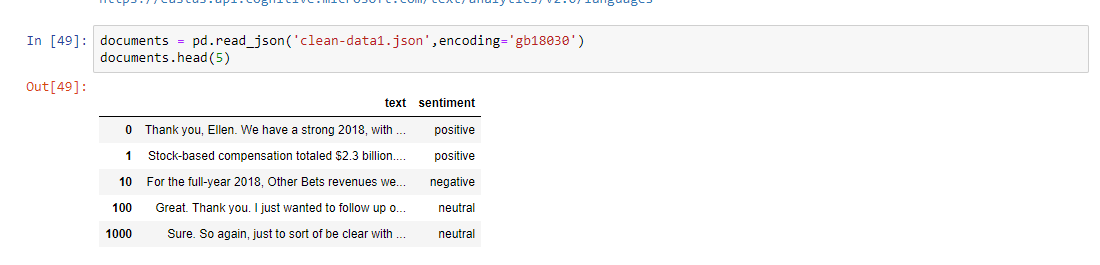


#### AZURE API

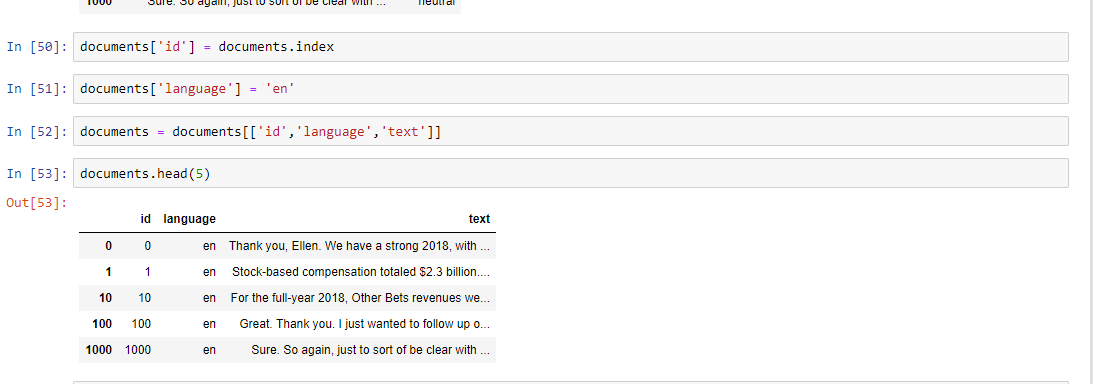
Firstly, we should set the configuration of the API which includes Azure user service group subscription key and base server URL:



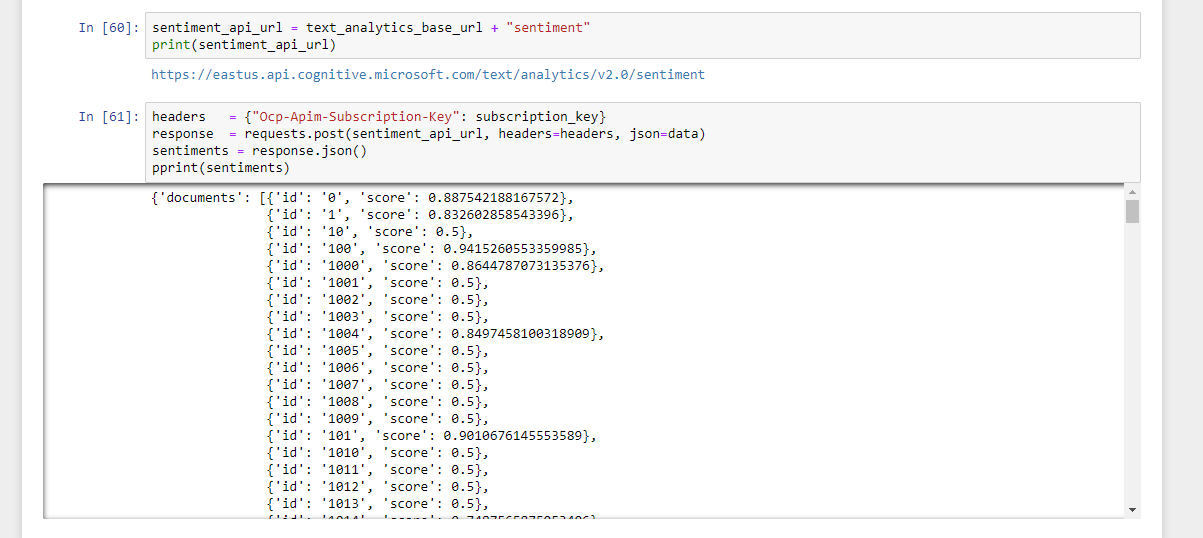
And then, we read the original JSON file:



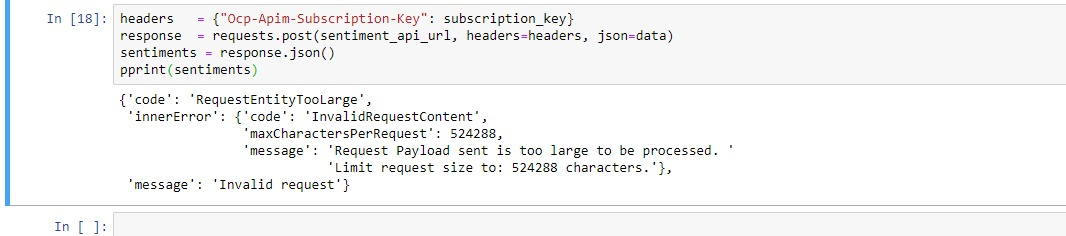
For the Azure API, It needs a special format of JSON fire which includes a list of language types because Azure can handle different language text for sentiment analyzing. Thus we should edit the file format we have now to the format suitable for Azure API :



At last, we use the API to process our normalized JSON file and we can get the result of each sentence:



There is a important problem for us to tackle. The problem is that the text file we use is too large to process :



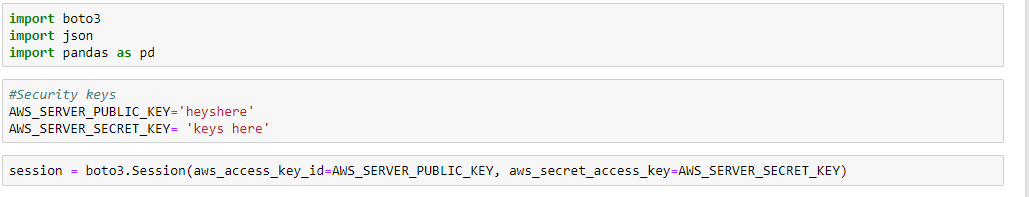
And we were trying to fix this issue by dividing the file into several small parts and process each part by a loop function and then add the results together, but because of lacking appropriate semantic skill in Python, we failed to finish that proceed. And we finally only processed half of the file.

#### Amazon API

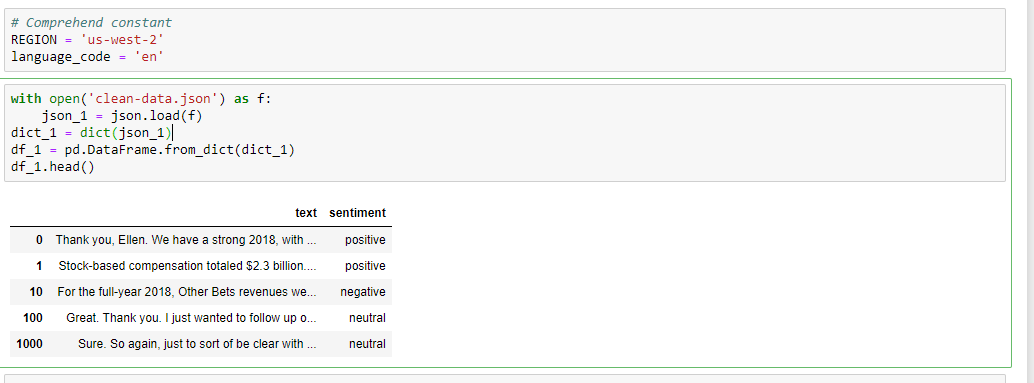
Amazon Comprehend is a natural language processing (NLP) service that uses machine learning to find insights and relationships in text. Amazon Comprehend uses machine learning to help you uncover the insights and relationships in an unstructured data. The service identifies the language of the text; extracts key phrases, places, people, brands, or events; understands how positive or negative the text is; analyzes text using tokenization and parts of speech; and automatically organizes a collection of text files by topic.

Setup

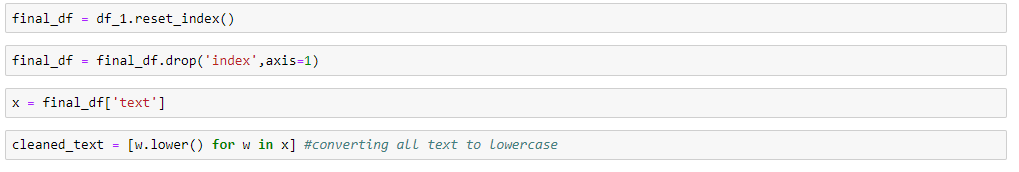
After signing up and creating public access keys, we were able to access Amazon’s platform using a jupyter notebook as shown below;



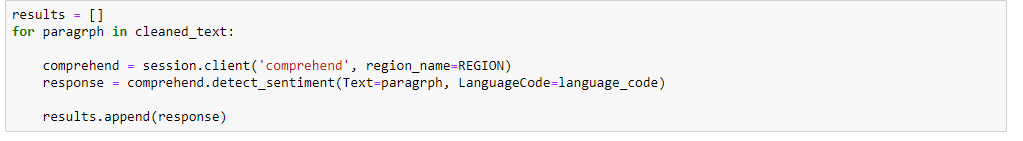
We then loaded the dataset as shown below and displayed the top 5 rows;



We reset and cleaned the datasets in the steps shown below;



We then applied Amazon’s natural language Comprehend service to each line in our dataset as shown below;



For each line of text in the original dataset, we saved the results in a list as shown above.

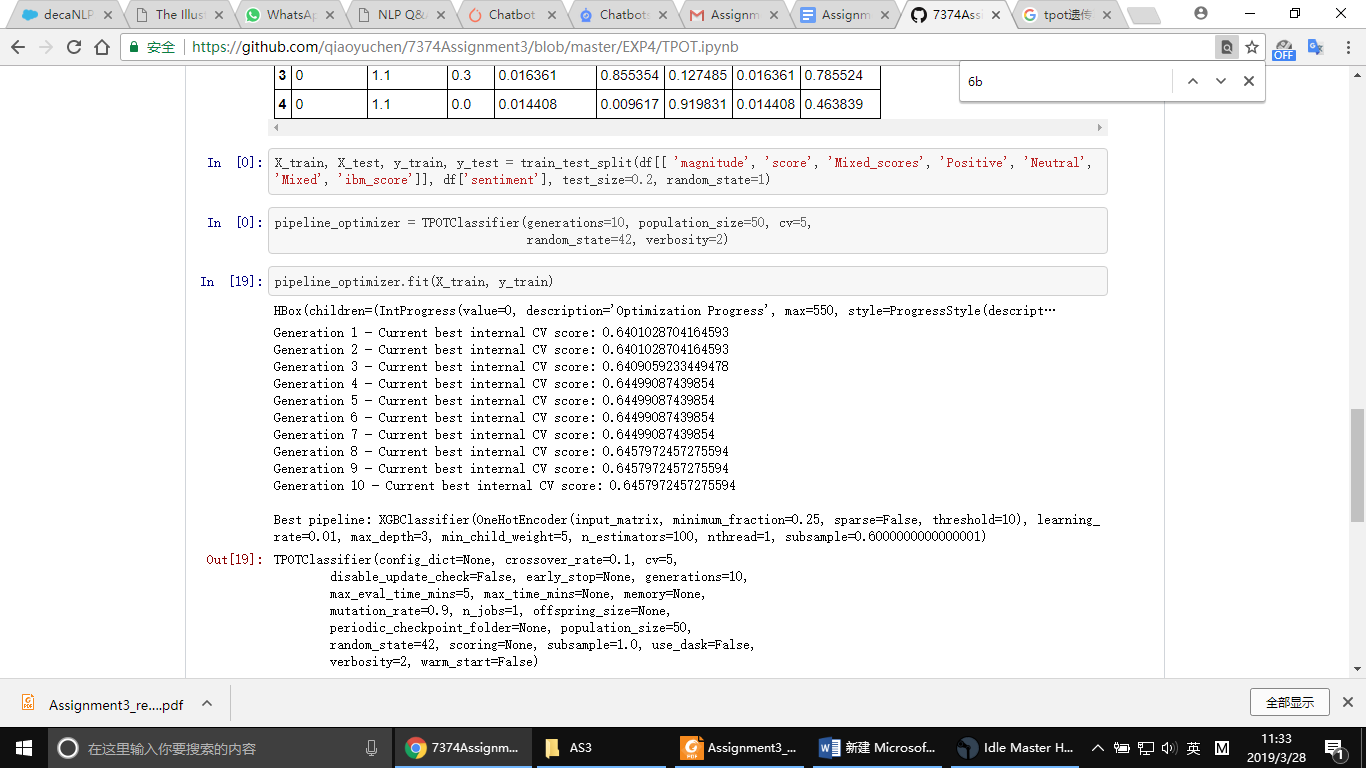
We then extracted the results from a json output and saved them in a csv file for future analysis as shown in the snapshot below



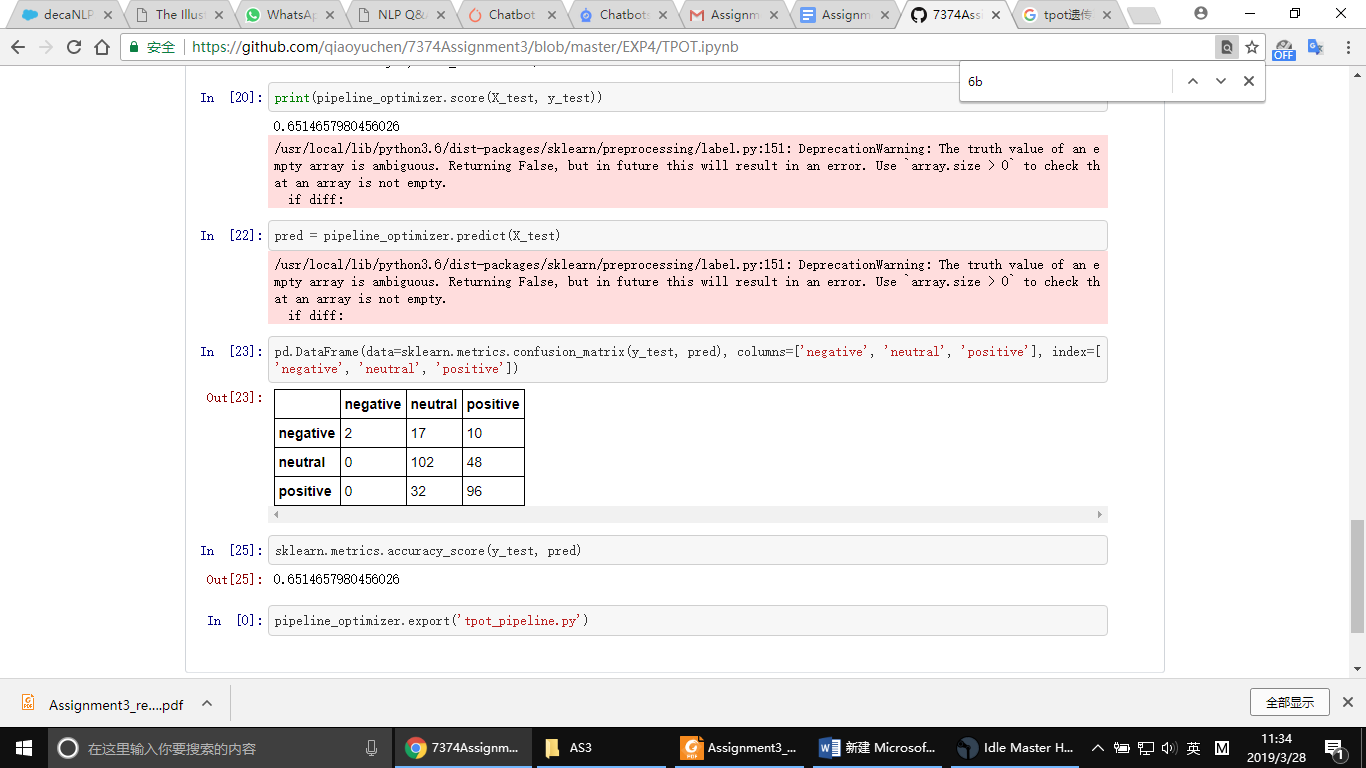
### Experiment 4:

#### TPOT

TPOT uses genetic algorithm to select the most suitable model for your data.



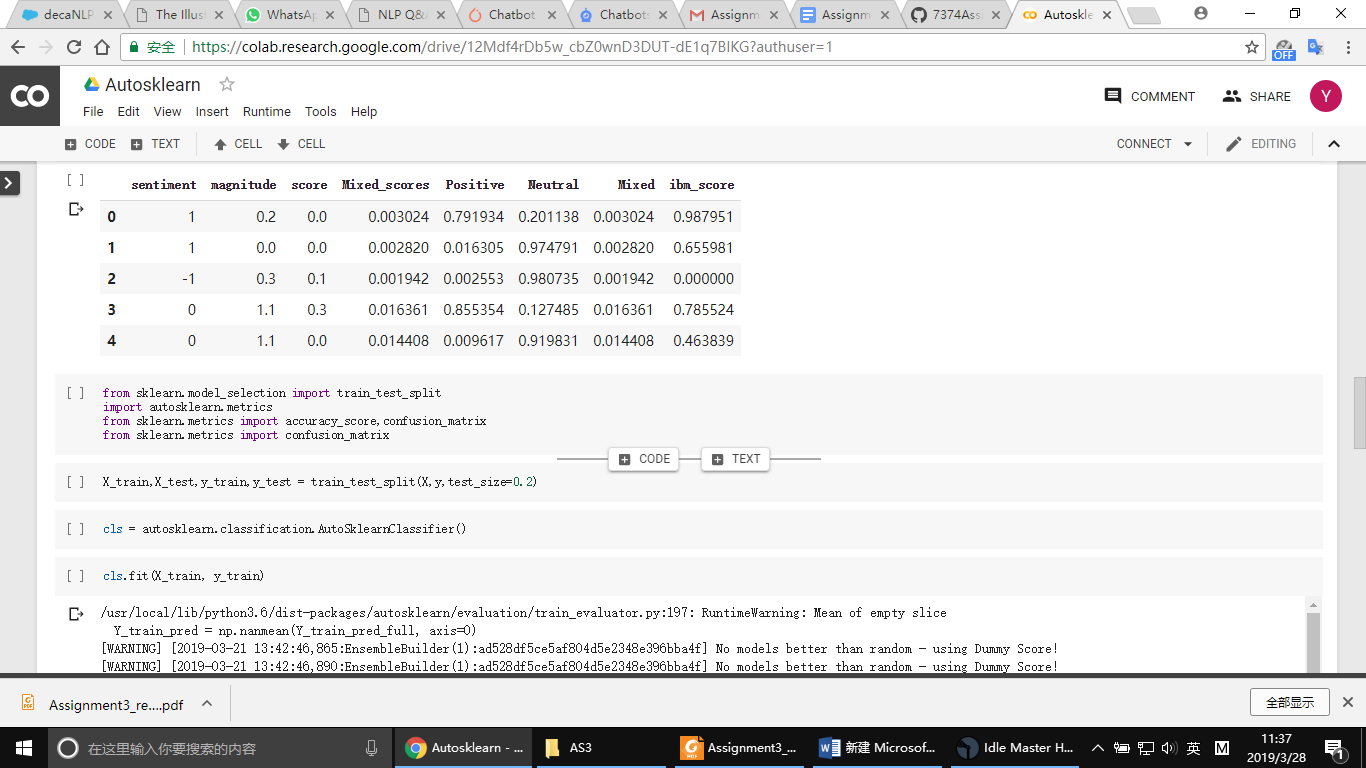
The confusion matrix as follow:

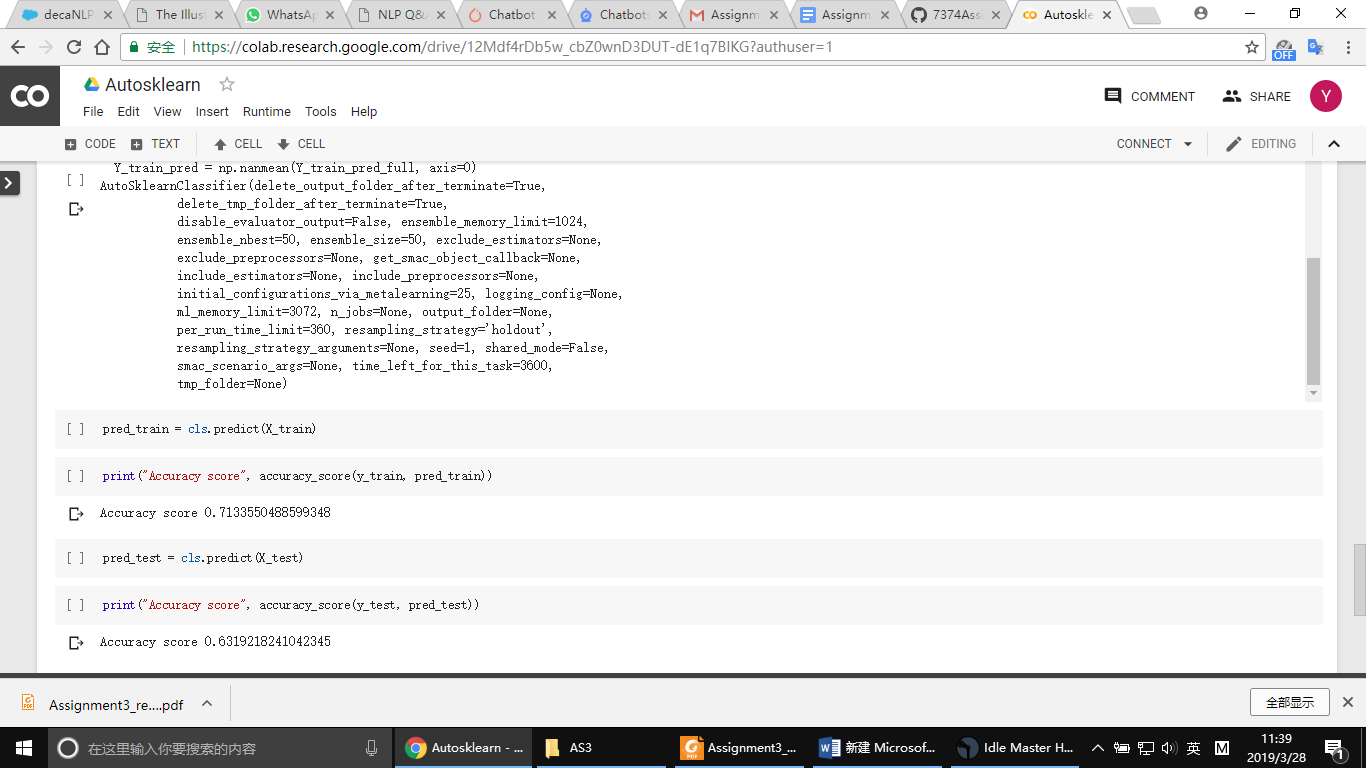


The best generation of TPOT had score of 0.65.

#### AutoSKLearn

The data set looks like:





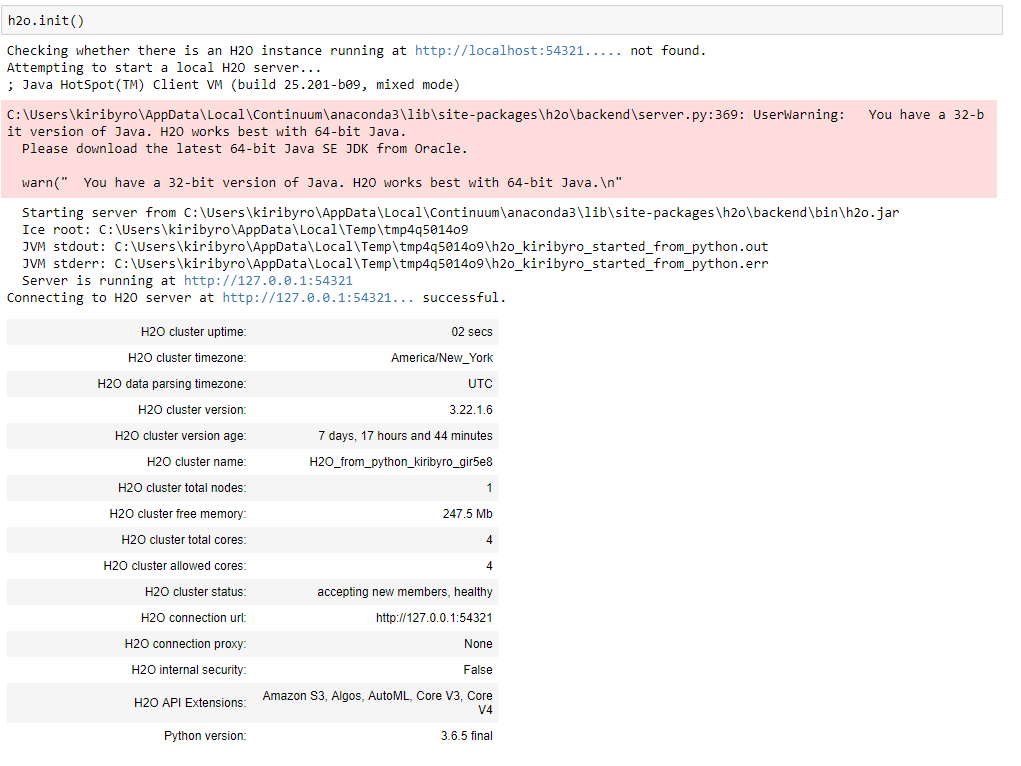
The test accuracy is 0.63.

#### H2O

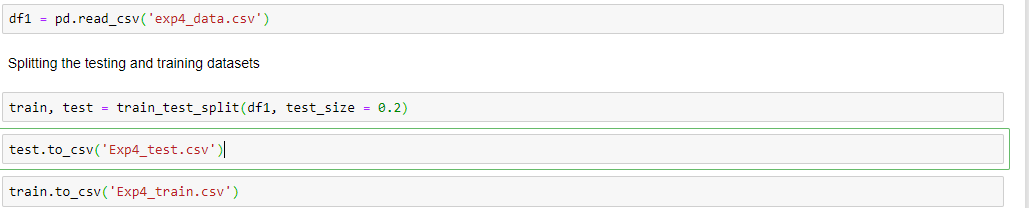
H2O is an open source, in-memory, distributed, fast, and scalable machine learning and predictive analytics platform that allows you to build machine learning models on big data and provides easy productionalization of those models in an enterprise environment. After computing the results from Experiment 3 into a complete dataset, and combining the results to the original sentences, we imported important libraries as shown below;



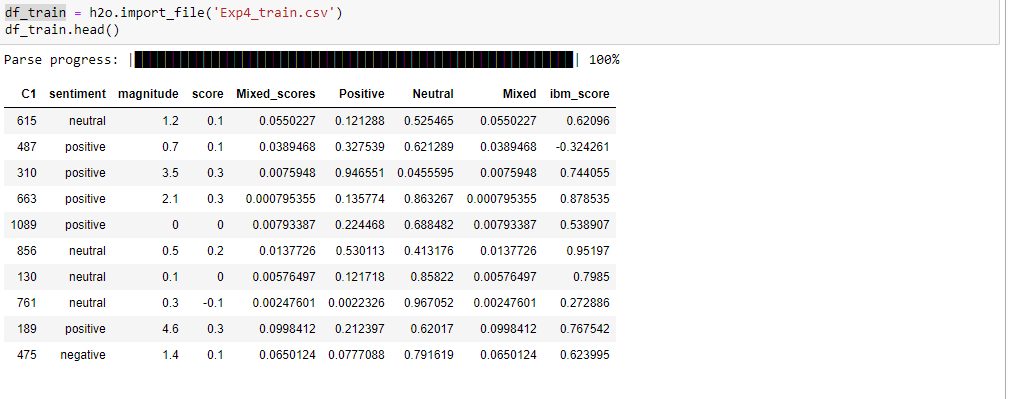
We initialized the H2O server as shown below;



We load the dataset to the environment as shown below and we split into train and test;



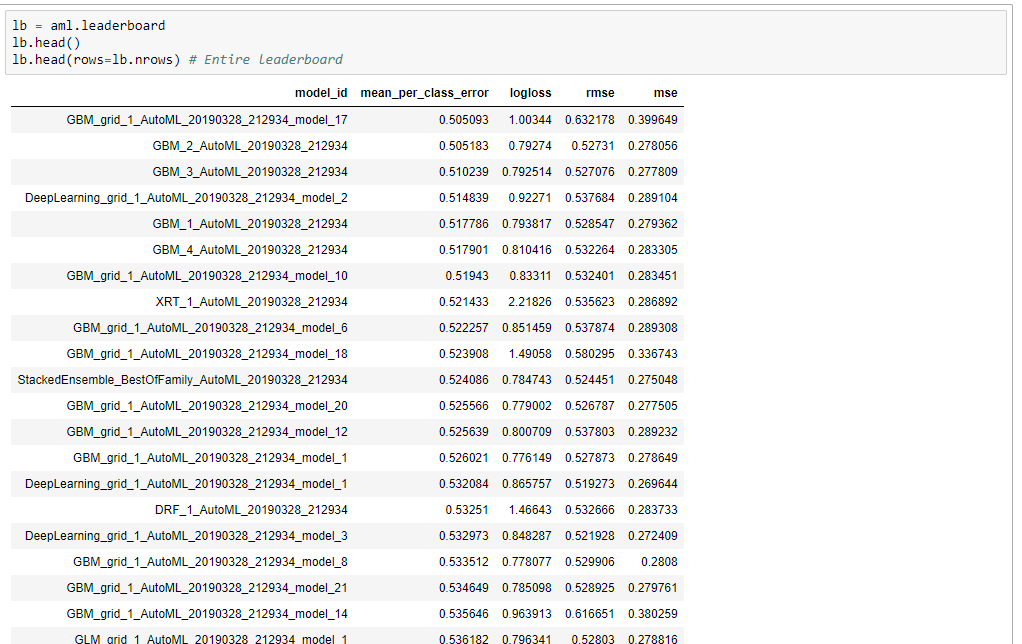
Visualizing the top rows of the training dataset.



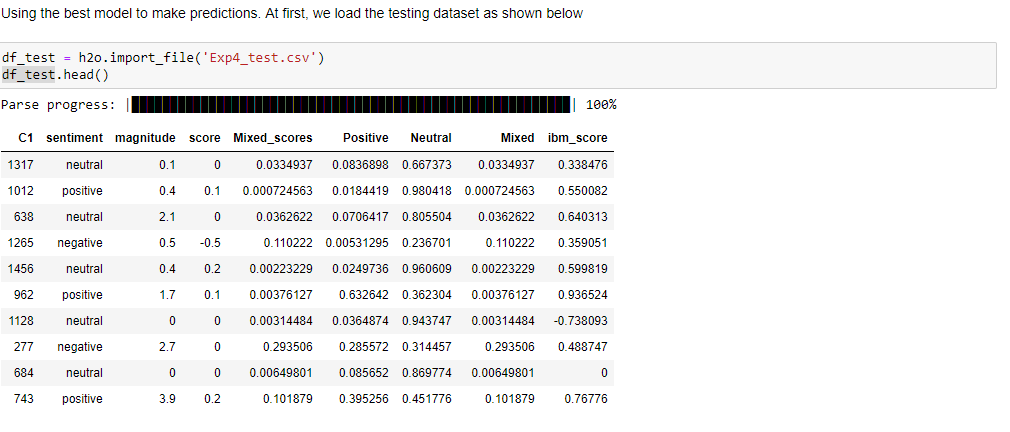
We split the training data into x and y and we test 100 models on the dataset as shown below;



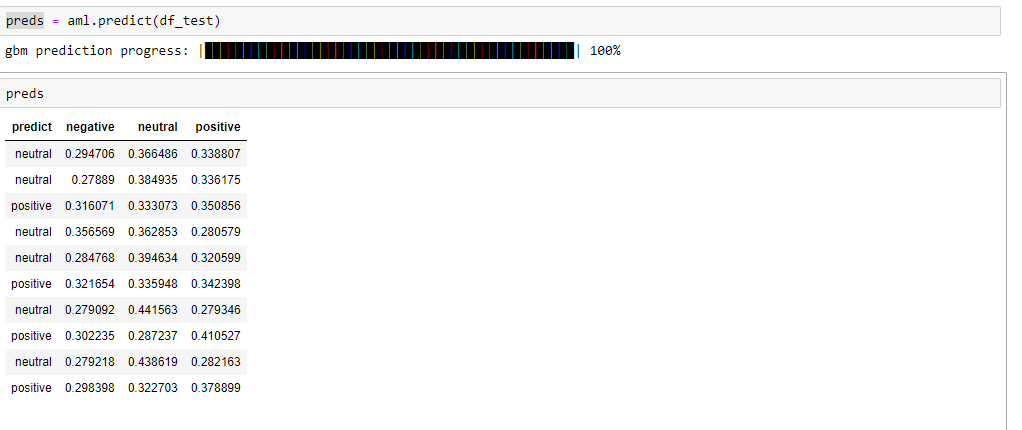
We visualize some of the top models as shown below;



We load the testing dataset to the H2O environment, and then we test the best model on the dataset as shown below;



Testing the best model on the test dataset



References

1. <https://medium.freecodecamp.org/an-introduction-to-bag-of-words-and-how-to-code-it-in-python-for-nlp-282e87a9da04>
2. <https://aws.amazon.com/comprehend/>
3. <https://medium.com/analytics-vidhya/gentle-introduction-to-automl-from-h2o-ai-a42b393b4ba2>