Project 2 – Karan Pankaj Makhija & Jeet Bhavesh Thakur

Introduction:

The following experiment was based on classifying an unlabeled post to its correct subreddit. The task is quite common and of utmost importance when it comes to large pools of mismanaged data centers. Reddits are a way people communicate to the larger ideas world-wide, the sub-reddit's are a medium through which each of these main reddit's are spawned into various groups or forums, however, still many posts coming in to the system at times are unrelated to the subreddit that they belong or are clueless about the many other subreddits that this post could be affecting apart from the prospective main idea it belongs to. Thus we tried here to come up with an algorithm which could classify each and every post labeled/ un-labeled in the future could be well identified and be classified accurately.

To achieve this herculean task, we have used the advance Machine Learning library of Sckilit Learn in python, which has a plethora of algorithms embedded in the para body of the function such that a user could use them at an instance by just mentioning the name followed by the exact parameters the function uses. We have chosen 3 subreddits to work on performing a one to many class classifications. We have produced our own learning knowledge base at the split ratio of 75% to 25%, SVM and Random Forest algorithms are used to train and equip our model with knowledge. The outputs on the 25% random split test data produced accurate accuracy around 85-95% on average.

Let us learn more about the details of our approach in the later chapters of this report.

Methods:

The project revolves around classifying unlabeled posts to its accurate reddit's, for which we used 3 subreddits quite distinct In nature to allow ourselves to understand the nuances behind the process in which the algorithm's work.

Subreddit #1: Data Science
Subreddit #3: Cooking
Subreddit #3: GRE

The vast popularity of these subjects is the notion behind their selection, each of them individually are topics people love to discuss and post comments on, however each of them are such where the plague to identify their true class towards a remark is quite troublesome. Thus we took it upon us to try and resolve this issue via means of an effective ML module to accurately weigh in the writeup and predict whether the text should belong to Data Science / cooking / GRE. However, the bigger picture here is that this algorithm can be scaled very rapidly to fit various subtopics for these main reddit's. Lets not waste time and learn the step by step making of our model.

- **Step 1: Extracting data and storing them:-**
  - **Library used:** Praw, Pandas
  - **Storing format:** Pandas DataFrame
  - **Converted DF into:** JSON
  - **Data Collected:** 350 comments per subreddit

- **Selection Procedure for Data:-**
  - The data from each subreddit was taken only if it contained a valid body with at least a single character and not being blank. { if len(self.text.body) > 1: extract text }

- **How was our data used ?**
  - We first upon collection cleaned our data or the extra unwanted information the data comprised itself of, like for example: we deleted column Time-Stamp, score, Title, ID, URL, No.of punctuations which were of no use towards our experiment.
  - So now we were left with just the comment per each transaction.
  - We thus introduced a column of data ourselves which was the class label: '0','1','2' respectively for each class it belonged.
  - So now our data looked something like this:
    { comment: "Where there is a will, there is a way", class-label: 0 }
  - We performed an action of feature extraction on our comments to make meaningful vectors which can help us for the machine learning procedure since the ML algorithms take only numeric input values and not string data.

Let us now move forward to our model training procedure

- **What is feature extraction ?**
  - Feature extraction is a process in which we mine specific knowledge out of written English text to convert them into suitable numeric count vectors or data points, this helps us to visualize text better and assists us to perform various applications revolving around learning from the semantics of written text scripts.
- **How have we implemented Feature extraction ?**
  - **Libraries used:** sklearn.feature_extraction.text, TfidVectorizer, CountVectorizer
  - **Code Info:** The TF-IDF converts each text para-phrases into Document Term matrix which is then used to identify the exact importance of that word para-phrase towards the class label classification of that comment.
- **What we achieve post feature extraction ?**
  - We are ready with a perfectly filtered data which is Learn ready, having only columns with data points of interest.
  - We understand the variations in the data stream which can help us to produce various plots that help us to better visualize the data using various plotting techniques.
  - A better normalized structure to be smooth enough to have higher accuracy and better statistical relevance when the same is used over for visualization.
- **What Algorithms used and how ?**
  - The model data is ready we now just need to split and feed our well set feature extracted and normalized data inputs to our model suited fit for learning to produce results that we wished for at the very beginning of our report.
  - The model was trained to work on SVM and Random Forest both supervised learning algorithms which help in defining the cluster labels based on the factors we have extracted and assigned weight depending on the various methods chosen, Eg. TF-IDF, Word Count Vectorizer.
  - The SVM class label predictor has 2 kernel modes each suited to serve a different purpose, however, normally the SVM kernel is biased towards two class label split. We changed the kernel setting to a one to multi class split which helped our visualization better.

Let us now view our logical steps / procedure through which we performed our experiment using some pseudo codes to assist our work.

Pseudo codes and working explanation of Method listed above:

Step 1 : Extraction and Data collection:

```
Library Praw

Praw.reddit(client-id:"your_id", client_secret: "your_secret_key", pass:"your_pass", user-agent:
"your_api_name", username:"your_username")

Reddit_name_1 = reddit.subreddit("Your_Topic_here") …. /// can add as many as required
```

The above makes sure you are well connected to the reddit app online and have virtual access to all the available subreddits that reddit has on offer to prawl and extract data using the key words you wish.

Step 2: Converting the same to a data frame:

```
Library Pandas

# convert the above prawled data into a suitable format to store, here we have considered a
dictionary however one can use multiple list objects to save the same as comma separated
values.

# Converting here into a data frame from dict

Dataframe = pd.DataFrame("your_object")
```

Step 3: Inserting class labels:

```
Dataframe["your_class_label_columnName"] = <your assigned class label – numeric>
```

Step 4: Data Pre-processing:

# Read your input JSON / DF as it is from the previous step, here we have converted our DF into a JSON Object for academic reasons.

# Converting to lower case for **A Grade**

- Data.iloc[:,"your_column_number"] = apply lambda (x: " ".join(x.lower))

# removing punctuations for **A Grade**

- Str.replace(pattern with words)

# Extracting STEM words

- **Library used:** from NLTK.stem import Porterstemmer

# Removing all the stop words

- **Library used:** from NLTK.corpus import stopwords

Step 5: Splitting data into 50% - 25% - 25%: As training, validation(development), test data

- **Library used:** from sklearn.model_selection import train_test_split

Step 6 Feature extraction:

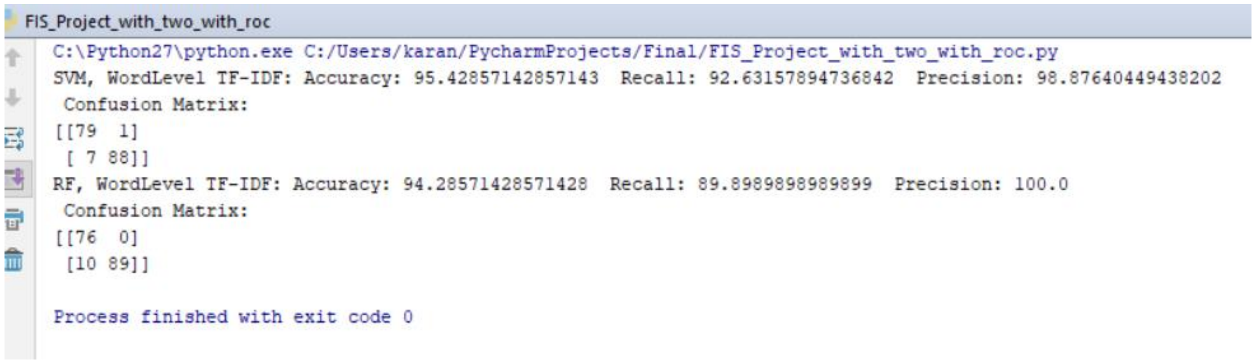
- **Library used:** from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer as mentioned above.
- We here fit the entire training set of split based on randomness into our training environment using .fit()
- Transform each row of the train split using .transform()

Step 7 Model Learning:

- Using sklearn we run the model of learning input to first our validation set, using: svm.SVC (kernel = linear) this kernel helps us perform one to class classification. Fit this model with your training data and its respective class label.
- Perform the same above step for Random forest using: ensemble.RandomForest(n_estimators = 100) #global constant value is 100.

Step 8 Prediction:

- **Our model is ready to predict the test labels with the highest amount of accuracy:** Using the above " classifier from the .fit() step above "
  this is performed using the model trained above on the test split data which was performed way above in step 5
- We are done with our prediction and we see that our model was accurate in and around 85-95% on various random splits. You don't believe us ? well we have something to show you then :



```
FIS_Project_with_two_with_roc
C:\Python27\python.exe C:/Users/karan/PycharmProjects/Final/FIS_Project_with_two_with_roc.py
SVM, WordLevel TF-IDF: Accuracy: 95.42857142857143  Recall: 92.63157894736842  Precision: 98.87640449438202
 Confusion Matrix:
[[79  1]
 [ 7 88]]
RF, WordLevel TF-IDF: Accuracy: 94.28571428571428  Recall: 89.8989898989899  Precision: 100.0
 Confusion Matrix:
[[76  0]
 [10 89]]

Process finished with exit code 0
```

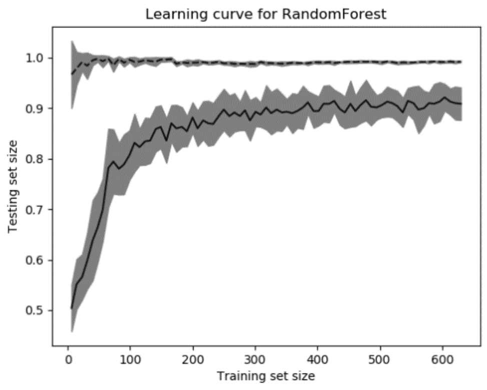
Here you go, 95% accuracy achieved on SVM and around the same for RF.

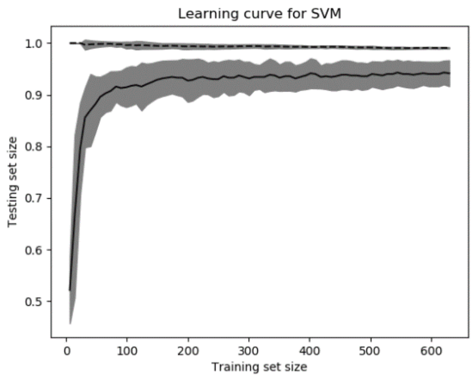
With this we end our Method's section and this picture takes us to a section full of images the Result section, followed by our conclusion.

Results:

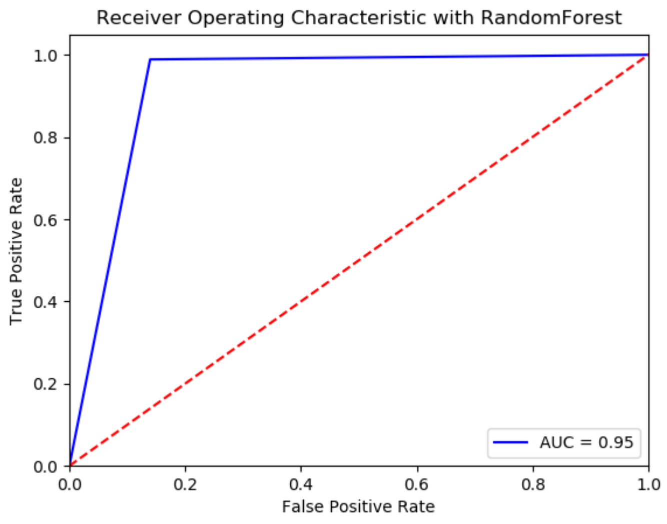
#0 Learning Curves:

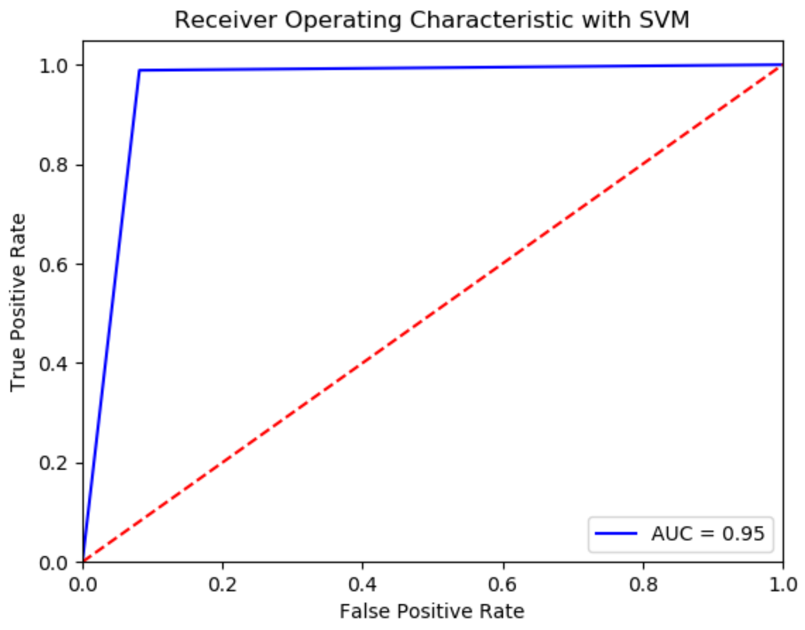
Learning curve for RandomForestClassifier and Learning curve for SVM





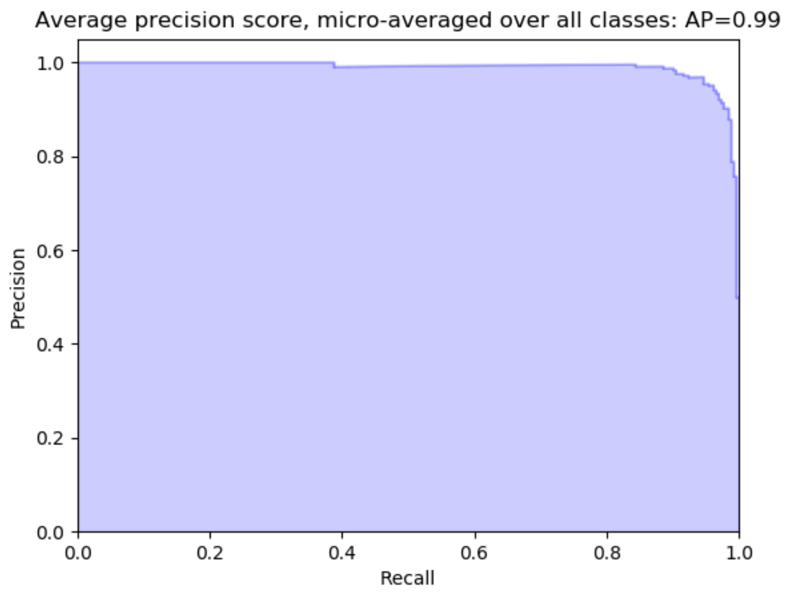
#1: ROC Curves:



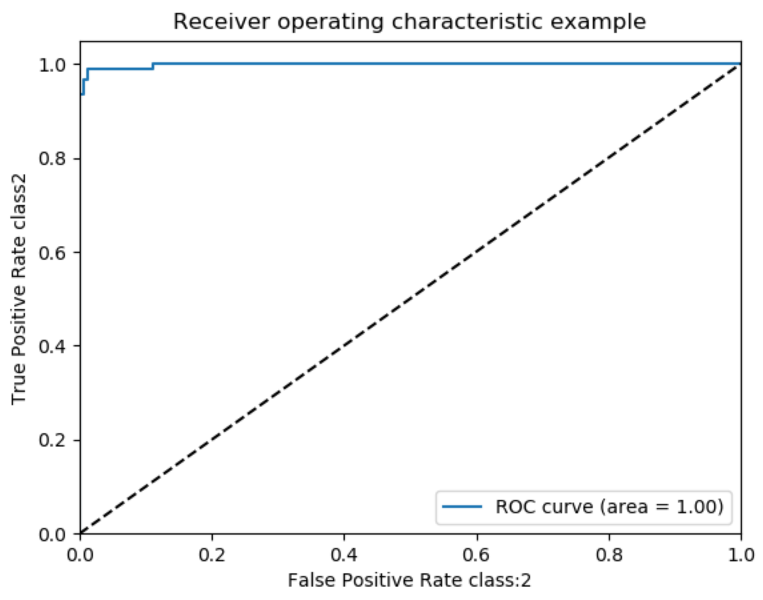


The above graphs are plotted against 2 classes : Data Science and Cooking, as you see the AUC is 95% that depicts an accuracy of prediction by our algorithm is around 95%.

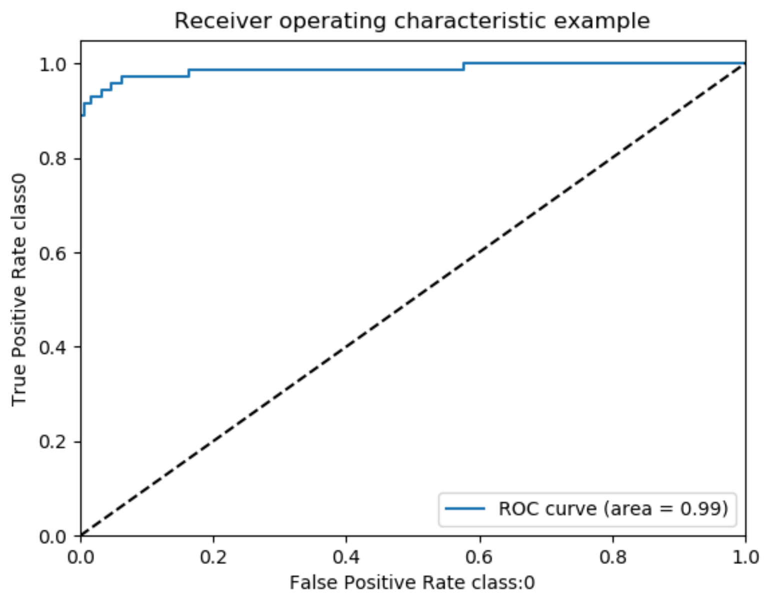
Now the next set of graphs are for SVM Algorithm towards the 1 vs Rest classifier:



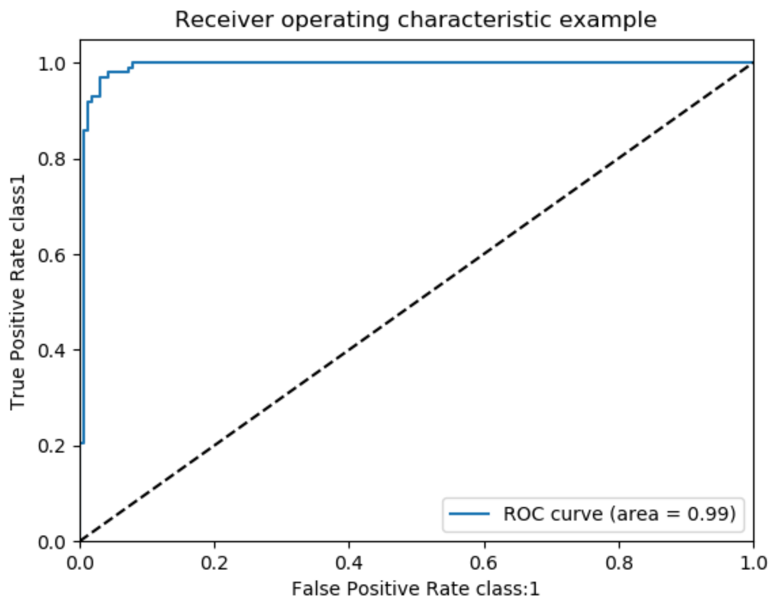
The above is Precision Vs Recall for the overall experiment



This above one is class label : 2 VS combination of classes : 0, 1



This above one is class label : 0 VS combination of classes : 2, 1

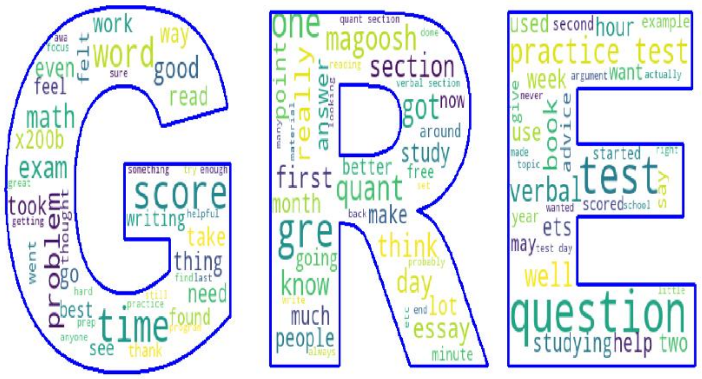


This above one is class label : 1 VS combination of classes : 0, 2

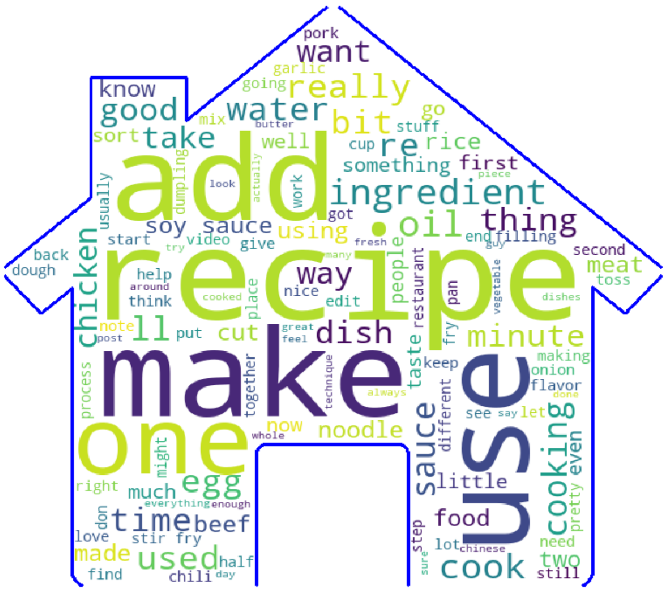
Now the next set is for **A Grade:**

Word Clouds:

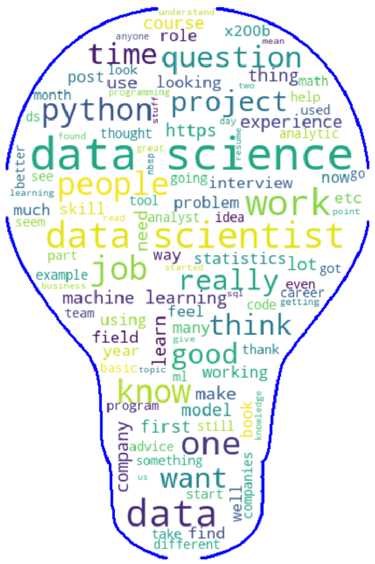
#1: Word cloud for subreddit 1, we have tried to depict the clouds as real to the scenario for which we have created them, eg. GRE for GRE.



Word cloud for Cooking– Subreddit

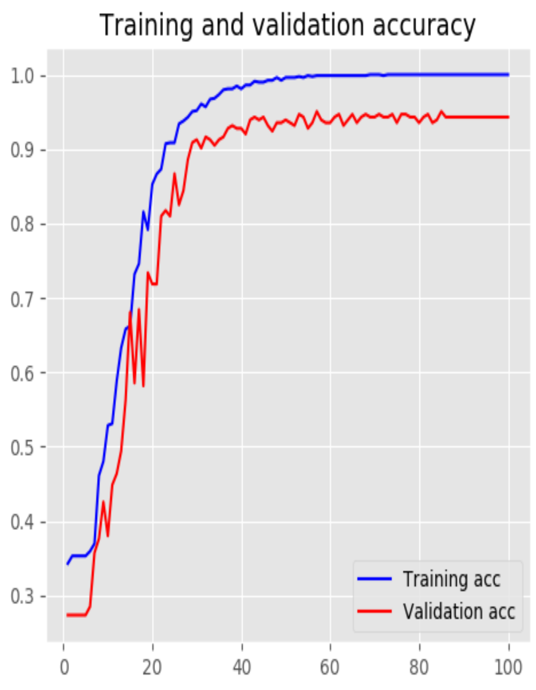


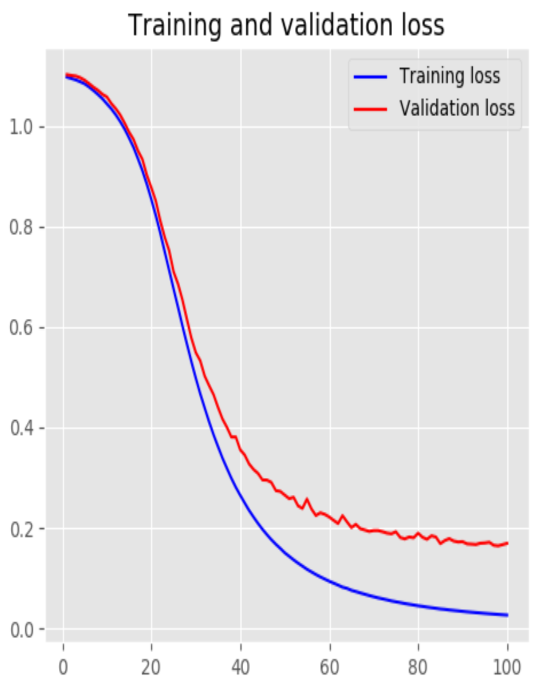
Word cloud for Data Science – Subreddit



Now our next set of pictures depict something of great importance:

LSTM using TensorFlow and Kernas:





The above graph is towards the content of A Grade:

We have taken 3 layers of activation functions, out of which 1,2 are Relu and the last one is softmax.

The reason for which is we are performing multiple classification and we need softness in our result.

Next up we are fitting our model for 100 episodes.

In the above graph we can see that around 50th episode, the graph becomes stable for our training accuracy. The accuracy for training is 100%, the accuracy for validation is 94.3333%

Our actual predictions were around 95% for SVM and 94.5% for Random Forest which is exactly what we see above so we can say our model goes hand-in-hand with the above mentioned models.

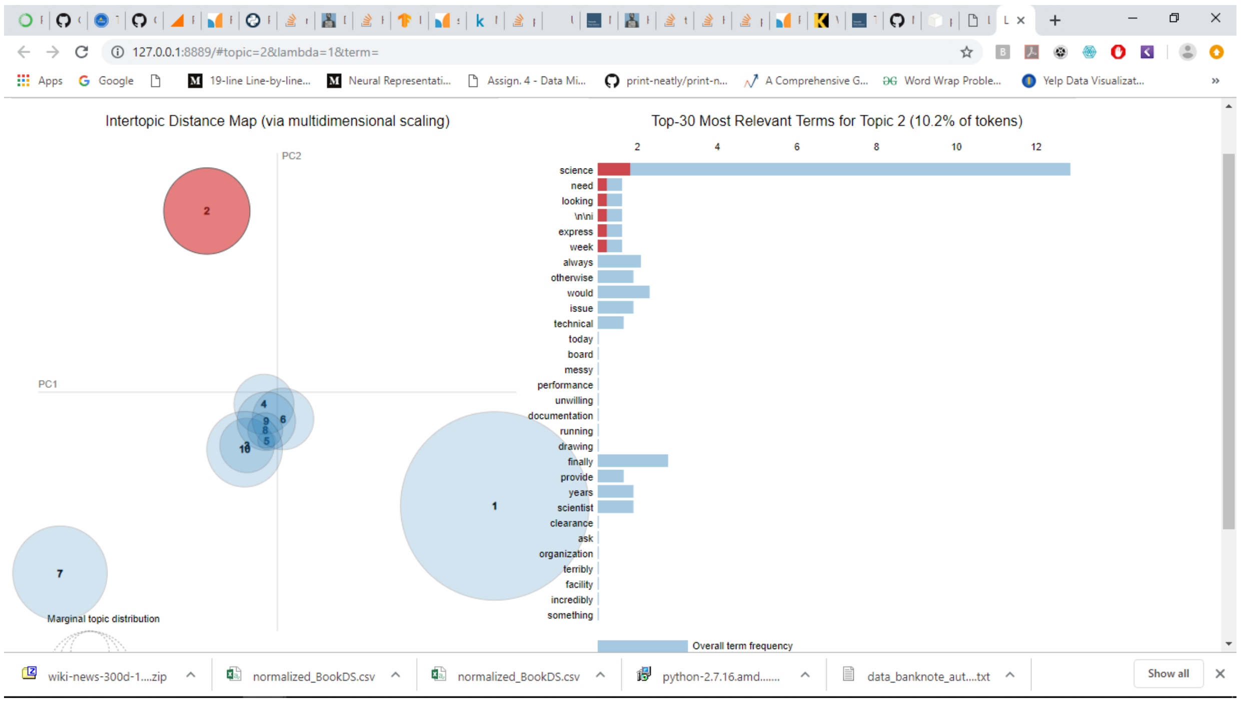
We know that LSTM models are overpower full in the longer run, so as to have our model in the same loop is quite enough to satisfy the near up approach method for the required data sets.

But in an event for more records it is required to have a model as robust as LSTM.

Loss for the training input was around 2% and test was around 20%

Topic mentioned under A Grade:

Intertopic distance map:

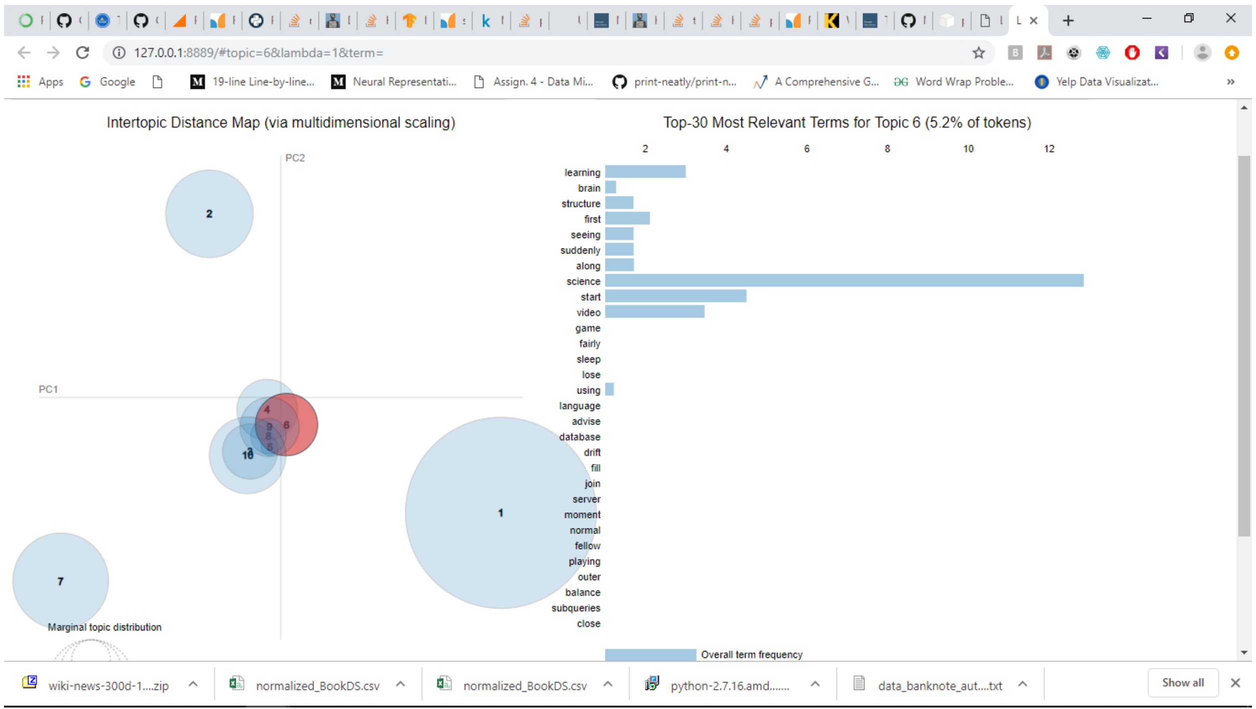


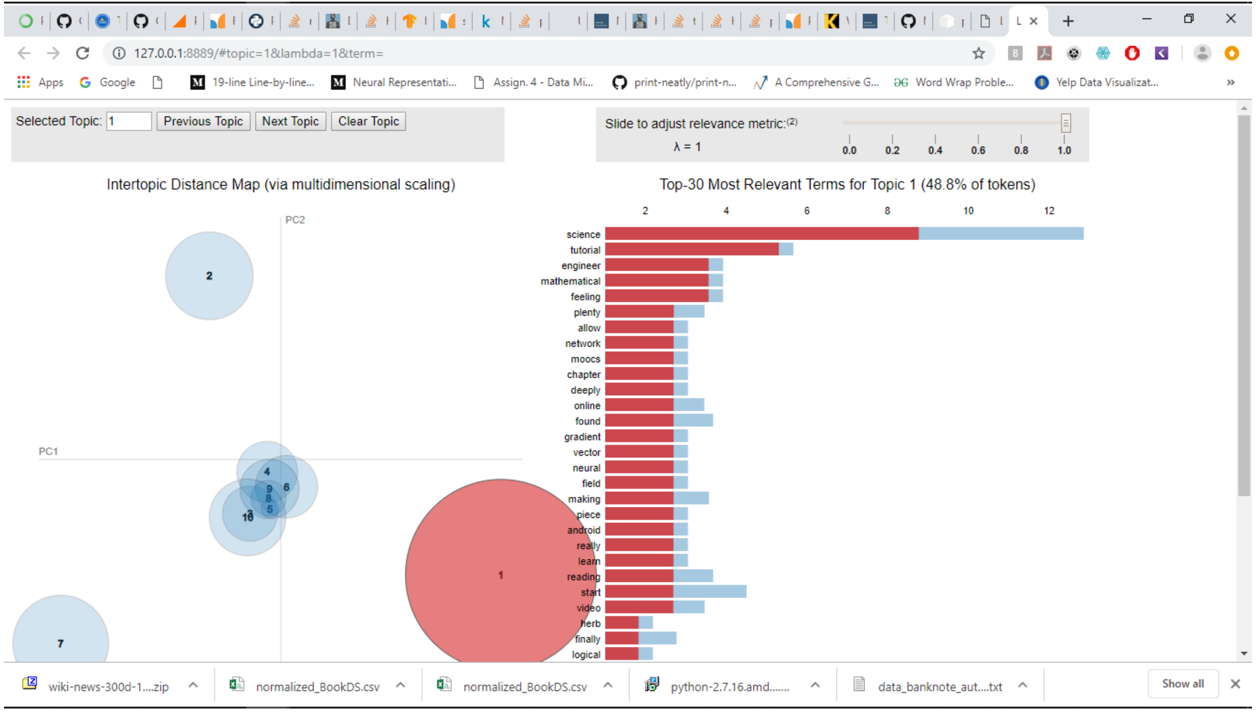
The above is a LDA diagram based on 10 chosen topics which are frequently identified in each of the respective documents on a higher ratio.

LDA is a form of unsupervised learning that views documents as bags of words (ie order does not matter). LDA works by first making a key assumption: the way a document was generated was by picking a set of topics and then for each topic picking a set of words. Now you may be asking “ok so how does it find topics?” Well the answer is simple: it reverse engineers this process.

The below pictures also depict the same.

Project 2 – Karan Pankaj Makhija & Jeet Bhavesh Thakur





Conclusion:

All in all to conclude the project in a nutshell, we used 3 subreddits, quite distinct in nature to assist our prediction better. We prawed, cleaned, processed and normalized our data to our needs.
Extracted features and used SVM, RF to train our models. We used 50-25-25 to split our data for better accuracy. We used the model on the tests to acquire a 95% prediction and have attached all our results

as part of our observations to this assignment. We have also broken our steps individually with pseudo codes to help people understand our flow better.

How to improve our performance in future:

We can learn to clean more data and try to implement the classifiers and gain the accuracy around 99-100%. We can learn more ways to implement and visualize the data in future using some interactive tools.

References:

- **http://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html**
- https://pandas.pydata.org/
- https://www.prowlapp.com/api.php
- https://www.tensorflow.org/api_docs/python

Hope we helped understand you better!

Adios,

Karan Pankaj Makhija, Jeet Bhavesh Thakur