Advanced Data Science And Architecture

**LIVE BLOG ANALYSIS**

horizontal line

# 

# PROJECT BY

TEAM 10

Harshad Pardeshi

Prithvi Kamath

Tejesh Rathi

**About WebHose**



WebHose provides on-demand access to web data feeds anyone can consume. Webhose.io empowers its customers to build, launch, and scale data-driven operations as they grow -- whether they are an entrepreneur, a researcher, or a senior executive at a Fortune 500 company. Developers get free access to the same web data feeds that power their growing customer base of global media analytics and monitoring leaders. Every web data feed is optimized to deliver up-to-the-minute coverage of a specific content domain, such as news, blogs, online discussions, and more. Just define your filters so you can focus on what you do best.

Webhose.io is the brainchild of Ran Geva and Guy Mor, two entrepreneurs with extensive experience in technology, data mining, and product development who set up to build a simple solution for a complicated problem for anyone who wants to consume data from the web.

**Dataset Summary:**

We picked up datasets on various different type of message boards, discussions and reviews from https://webhose.io/datsets and combined them into one dataframe, so that we have access too all different types of data provided by webhoseio on which we can work on.

Each JSON files has the below structure



**Project Flow**

We have completed the project in below phases:

* Download historical data and structure them
* Perform Data Wrangling and generate processed data
* Store Processed Data
* Use Processed Data to perform exploratory data analysis
* Implement TF-IDF algorithm to search the entered keyword in the blogs
* Generate bag of words using the top 50 words for each keyword
* Pipeline the above steps using Luigi
* Dockerize the Luigi pipeline and push docker image on dockerhub
* Implement classification algorithms to distinguish relevant and n on-relevant blogs
* Group relevant blogs into clusters using LDA (Latent Dirichlet Allocation) Algorithm
* Download live data
* Test the above trained models on live data

**Download Historical Data**

* Manually download historical data from webhose.io in json format and stored them. We have not opted for the paid API service to programmatically download this data
* Collaborated 500 files from each category to create sample data
* Tagged 250 files of each category as relevant and added the category name in the Keyword column and tagged the rest 250 as irrelevant and added a random category name in Keyword column

**Perform Data Wrangling and generate processed data**

**Data Concatenation:**

* Read all the JSON files from ‘/Final/Data/Input Data’ into a DataFrame
* As we know that these are nested JSON files, merge the keys from inner JSON with the keys from the outer JSON using lambda function
* Drop redundant columns from this DataFrame

**Handling Missing Data**

* Check if the dataset has missing values and get a count and percentage of missing values for each attribute
* From this we understand that ‘rating’ has 99% missing values. We therefore ignore this variable in our analysis.
* ‘domain\_rank’ has 55% missing values. To fill these missing values we have to understand what ‘domain\_rank’ represents and what are the factors it is based on.
* ‘main\_image’ is the image URL and its missing values are filled with ‘NA’
* Remove ‘new line’ characters

Next we are converting 'published' variable to datetime format and deriving a new column 'published\_day' which only extracts the day from the published variable.

This dataframe is stored as 'ProcessedData.csv' in '/Final/Data/Processed Data'

We shall henceforth be using this file to perform analysis.

**Exploratory Data Analysis**

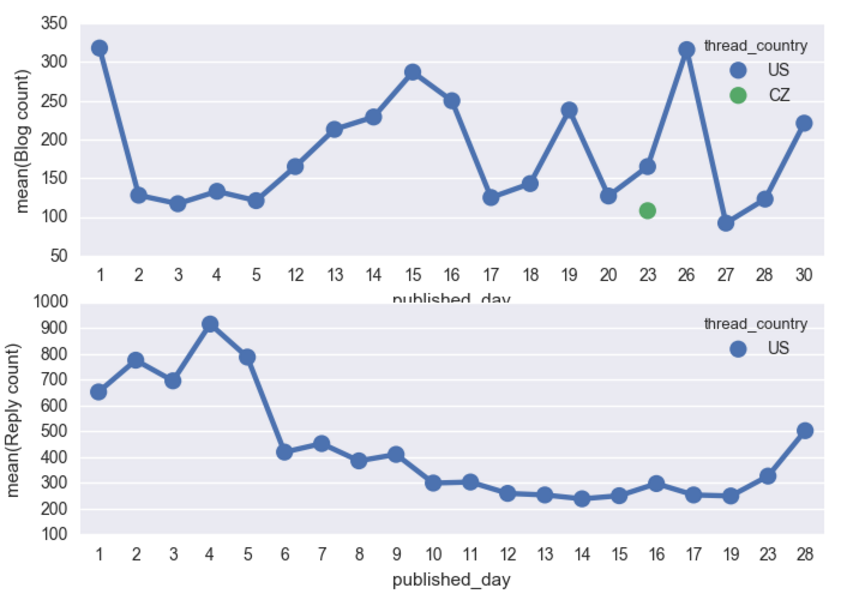
**Analysis 1:** Get country wise distribution of number of people contributing to blog writing and replying to various blog over the past month.

**Approach:**

* Read the processed data
* Get top 20 countries which contribute to blog writing using groupby function on ‘published\_day’ and ‘country’ variable
* Get top 20 countries which contribute to replying to blog using groupby function on ‘published\_day’ and ‘country’ and getting a count of ‘replies\_count’
* Use matplotlib to generate subplots
* Generate graphs using seaborn for the above generated DataFrames

**Conclusion:**

* On observing the trend, we understand that there is a large contribution of people from US to writing blogs and replying to blogs
* As this trend is continuous, we can also conclude that they contribute on an everyday basis



**Analysis 2:** Find the most preferred language to write a blog

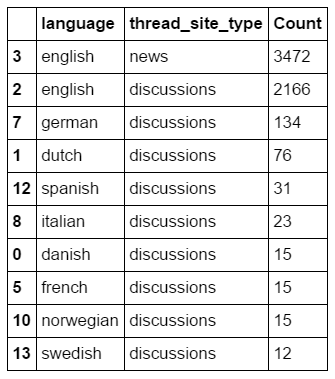
**Approach:**

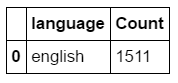
* Read the processed data
* Group the entire dataset based on languages that blogs are published in
* Get the various ‘site\_type’ supported by various languages
* Get the count of replies for blogs in each language

**Conclusion:**

English is the most preferred language to write a blog because of the following:

* It is the only language that supports all 3 types of articles that is blog, news and discussions
* It also has the highest number of replies compared to other languages





We move ahead with finding the relevancy of ‘Keyword’ in each document. To do that we use TF-IDF algorithm to get the tf-idf score for the keyword in each document. Comparing the score

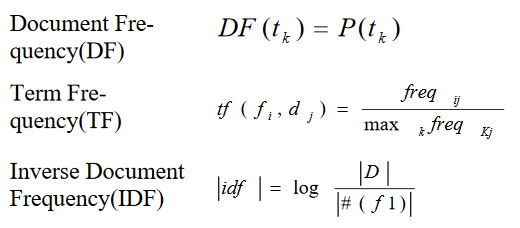
**TF-IDF (Text frequency- Inverse Document Frequency)**

This algorithm calculates the weight for each word based on the its occurrence in the file and compares it with words in all other files. It reduces the weight of words which appears in multiple files thereby increasing the weight of unique words of each file.

**Explanation**

* Tf(word,blob): Calculates term frequency for every word normalized by dividing by the total number of words in the file.
* N\_containing(word,bloblist): Returns the number of documents containing the word passed in the argument.
* Idf(word,bloblist): Computes Inverse Document Frequency by computing how the occurrence of the word. We take the ratio of total number of documents to the number of documents in which the word occurs and take a log of this. We also add 1 to the divisor to prevent division by 0.
* Tfidf(word,blob,bloblist): Computes tfidf score by multiplying tf and idf values calculated above.
* We then use NLTK package to remove the stop words in English and generate a ‘tfanalysis’ DataFrame containing only the text

**Mathematical equation:**



**Approach:**

* Read the processed data
* Remove punctuations from the text
* Remove stop words using NLTK package
* Convert the entire text to BlobText object
* Find the score for each word using the above defined functions
* Sort the words in descending order
* Find the ‘Keyword’ in this list of words and save its respective ‘score’ and ‘position’ into a same DataFrame
* Create a ‘Bag-Of-Words’ for each keyword selecting only the top 50 sorted words in each blog
* If ‘Keyword’ is not found save ‘score’ and ‘position’ as -1

**Output:**

* After implementing TF\_IDF on each blog, we get the tf-idf score of the keyword in each blog.
* Bag of words for each keyword

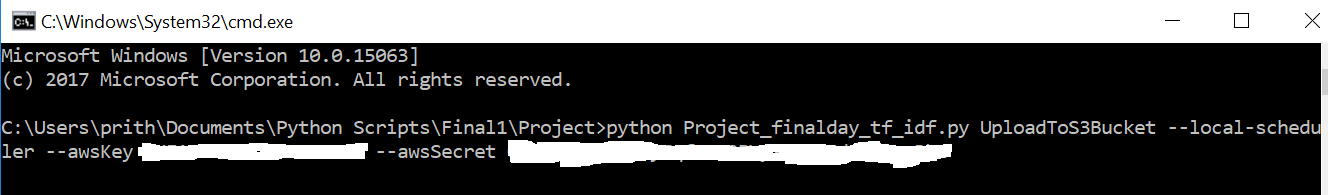


**Pipelining using Luigi:**

**Approach:**

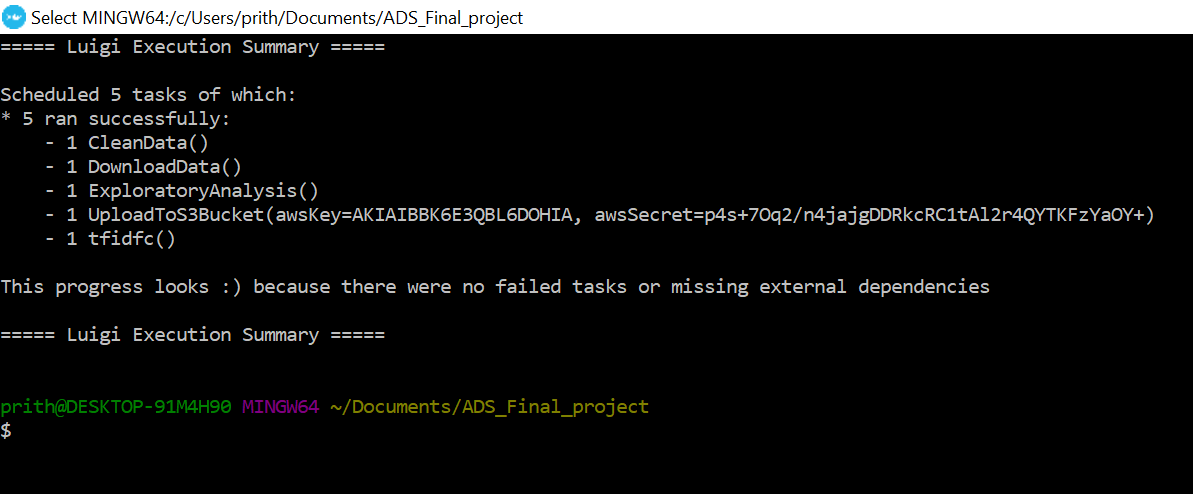
* Create function for the below tasks:
* Read downloaded data
* Clean data
* Exploratory Data Analysis on historical data
* Calculate tf-idf
* Upload files to Amazon S3
* Link the last task to the previous and so on till the first
* Run the luigi pipeline using the below code snippet:

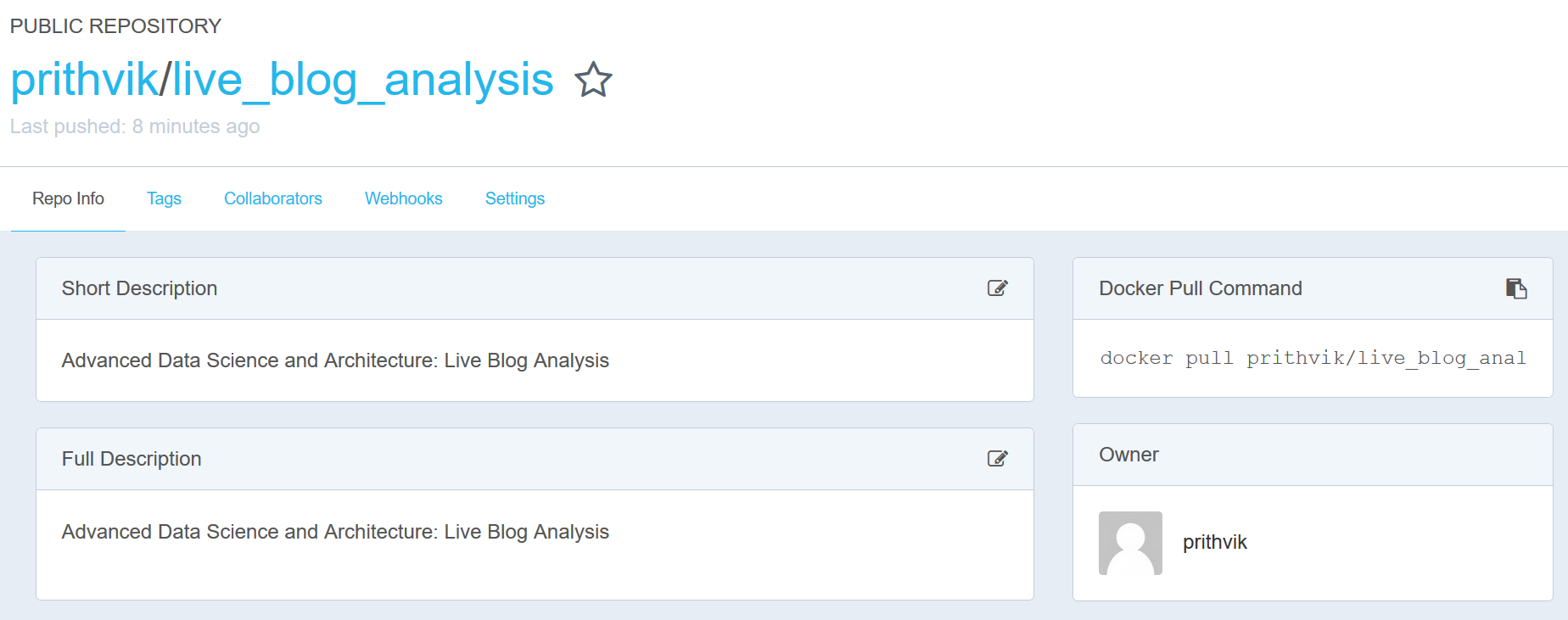
Python Project\_finalday\_tf\_idf.py UploadToS3Busket --local-scheduler --awsKey <AWS\_KEY> --awsSecret <AWS\_SECRET KEY>



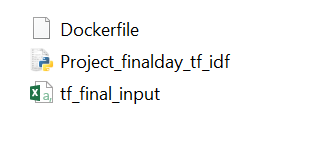
**Dockerize the entire pipeline:**

We have dockerized the entire luigi pipeline.



**Docker Image and Repository on DockerHub:**

**Input files for Docker are as follows:**



**Steps to run Docker image:**

* docker pull prithvik/live\_blog\_analysis
* docker run prithvik/live\_blog\_analysis

**Uploading files on Amazon S3:**

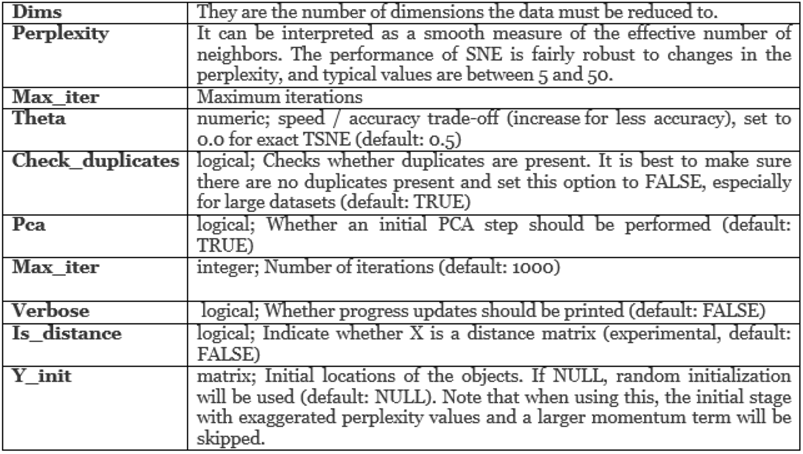
The following files are uploaded on Amazon S3 bucket in ‘team10ads\_final\_project’ repository:

* Analysis2.csv
* Bag\_of\_Words.csv
* clean\_data.csv
* tf-idf\_data.csv

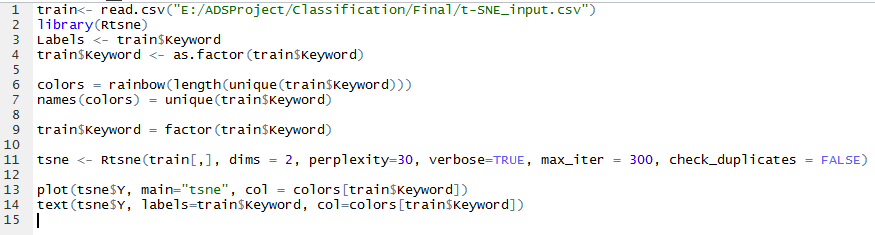
**Implementation of t-SNE - Dimensionality Reduction Algorithm**

t-SNE a non-linear dimensionality reduction algorithm finds patterns in the data by identifying observed clusters based on similarity of data points with multiple features. It is not a clustering algorithm, mainly a data exploration and visualization technique.

We implemented t-SNE in order to understand the dimensions which are required to classify them into various models. We implemented t-SNE in R, using the Rtsne package and involving and adjusting various Hyper Parameter tuning as given below:



Below is the code we wrote in R, in order to cluster our keywords in various clusters and determine the dimensions required for us to perform classification.

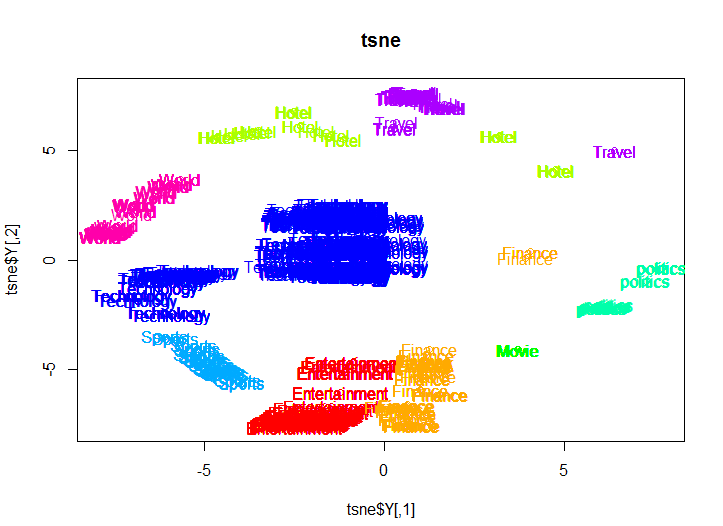


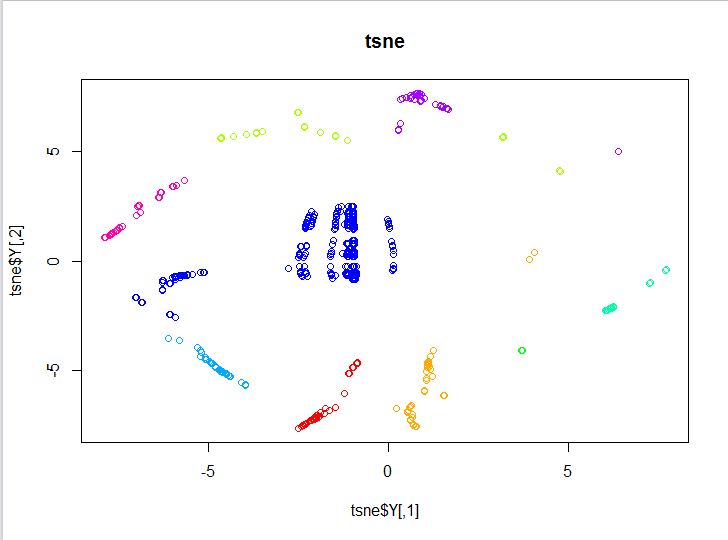
The code above when run on the console, by calling

source(“tsne.R”)

Gives the visually appealing clusters (except for a few anomalies)

We labelled the clusters as the Keywords so that we understand how the clustering has been and how is the dimensions we included help us form a better clusters.





**Classification**

There are different types of message board or blogs on Reviews on Hotel, movies, etc and different section of news like Technology, Sports, Politics, Finance, Entertainment or Travel related news available on the internet. Being a Marketing Manager at a firm, one would like to see very specific section of news available on internet. The person wouldn’t be spending whole day and night just to figure out all the relevant news related to the domain they work in. So, this is the classification model which gives a user the relevance of all the news on the internet based on its relevance and irrelevance.

The model and the algorithm we implemented is based on a keyword entered by the user. We have a distinguished data, divided into various categories and so we trained the model based on the section it belongs to using the Keyword.

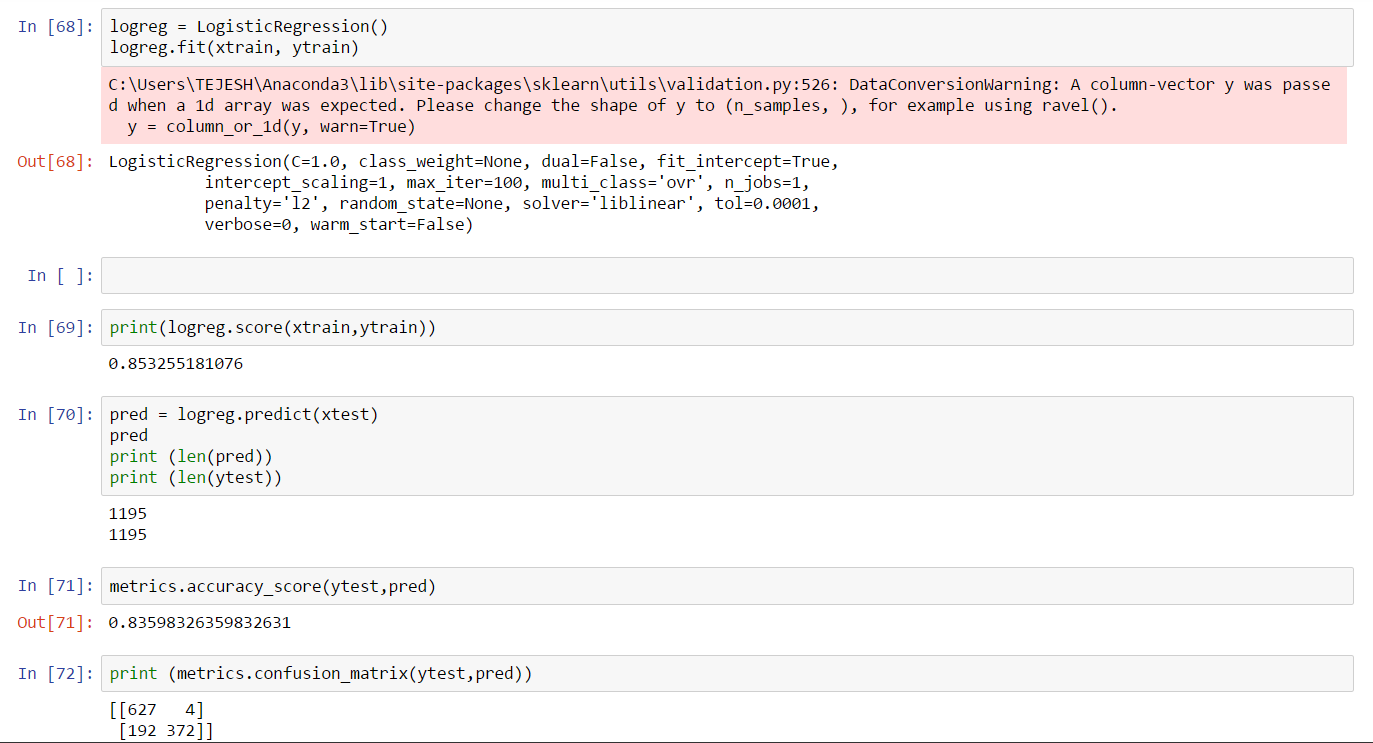
To run the Classification models, we first generated the TF-IDF score based on the Keyword and the text column in the data. A relevant score and a position is generated as a part of the TF-IDF algorithm. This score and position helps us classify the data as relevant or irrelevant.

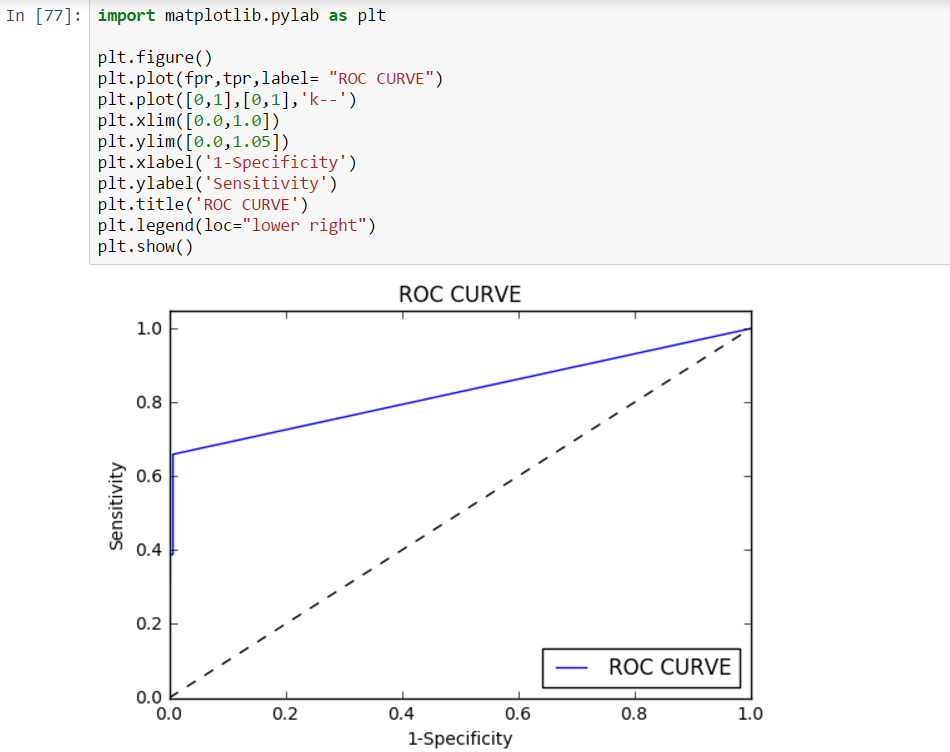
We implemented the model using Logistic Regression, Neural Networks and Random Forest Algorithm on Python. The accuracy did not really change much for any of it.

Model Accuracy:

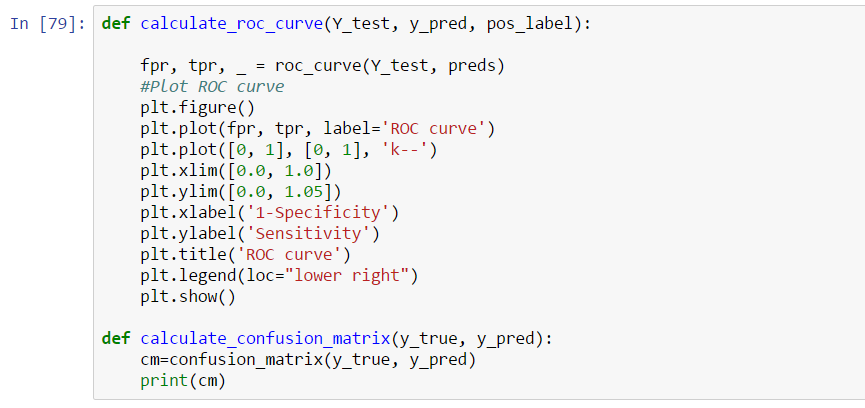
|  |  |  |  |
| --- | --- | --- | --- |
| Model | Logistic Regression | Neural Networks | Random Forest |
| Accuracy | 83.59 | 83.52 | 83.68 |

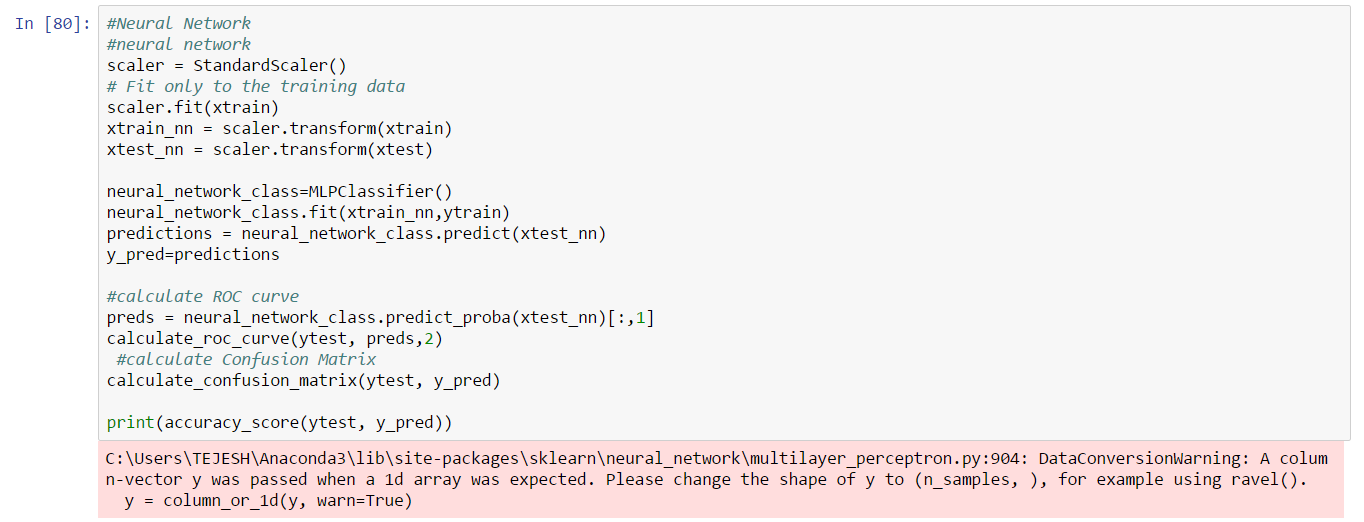
**Logistic Regression:**

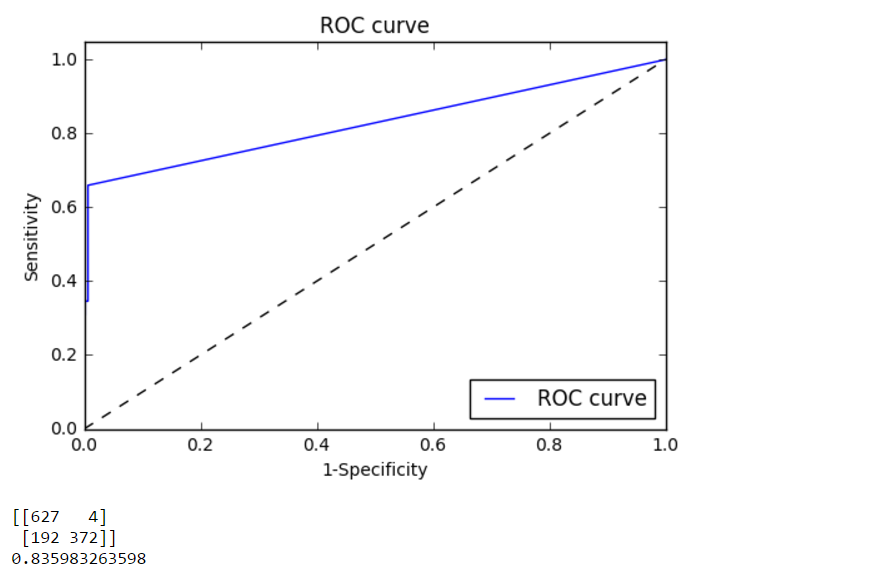


**ROC Curve:**

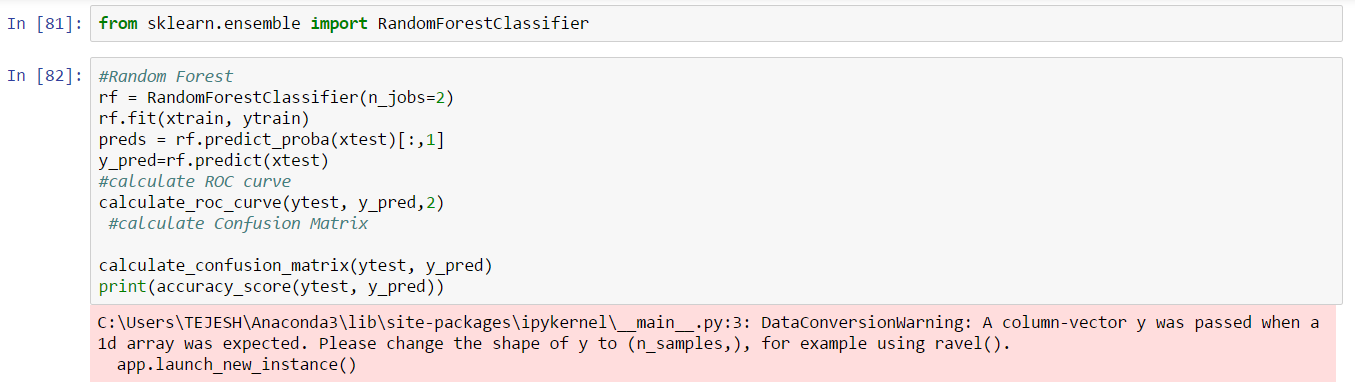
**Neural Networks:**

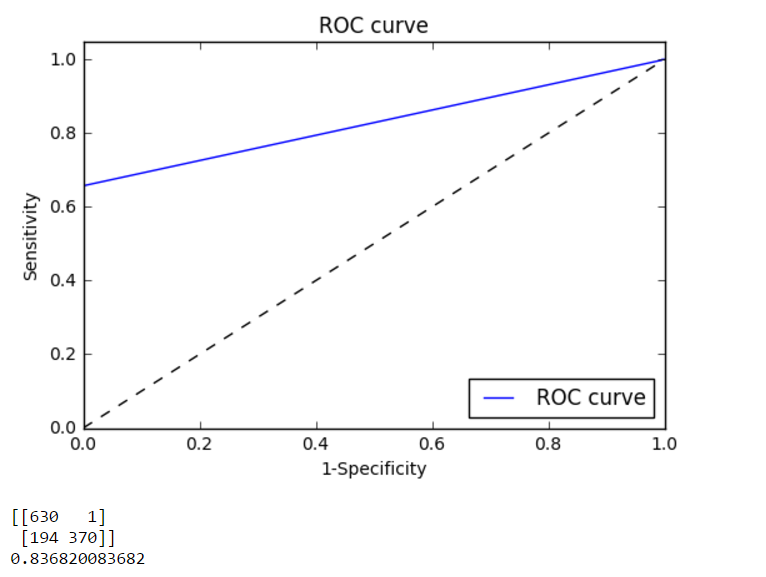






**Random Forest:**





Based on the accuracy of the three models, which was very close to approx. 84%, Random forest was the best with the accuracy of 83.68%. So, we implemented this model on Microsoft Azure Machine Learning Studio. Since, all the three did not differ too much between the accuracy, so we tried and implemented all the three on Azure Machine Learning Studio and there as well, we found Random Forest to be the best by a small margin.

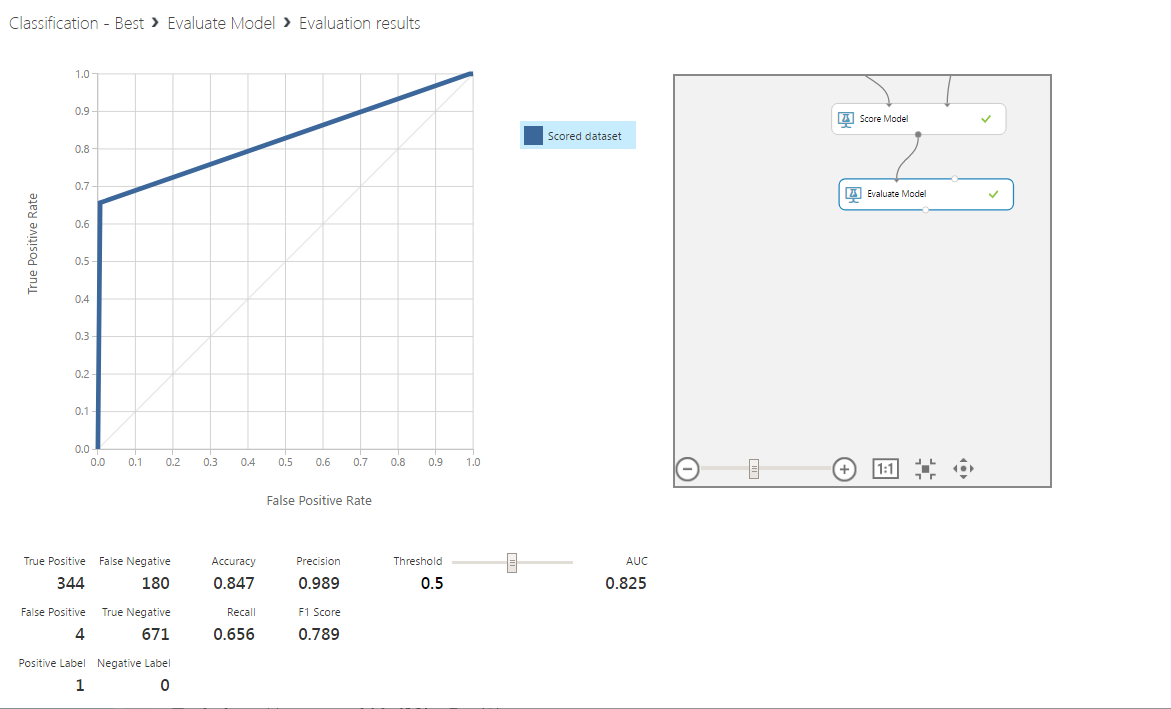
**Best Model- Random Forest**

Since, the accuracy was pretty close though our models, so just to confirm we implemented all the models on Microsoft Azure Machine Learning Algorithms and found out the below results:

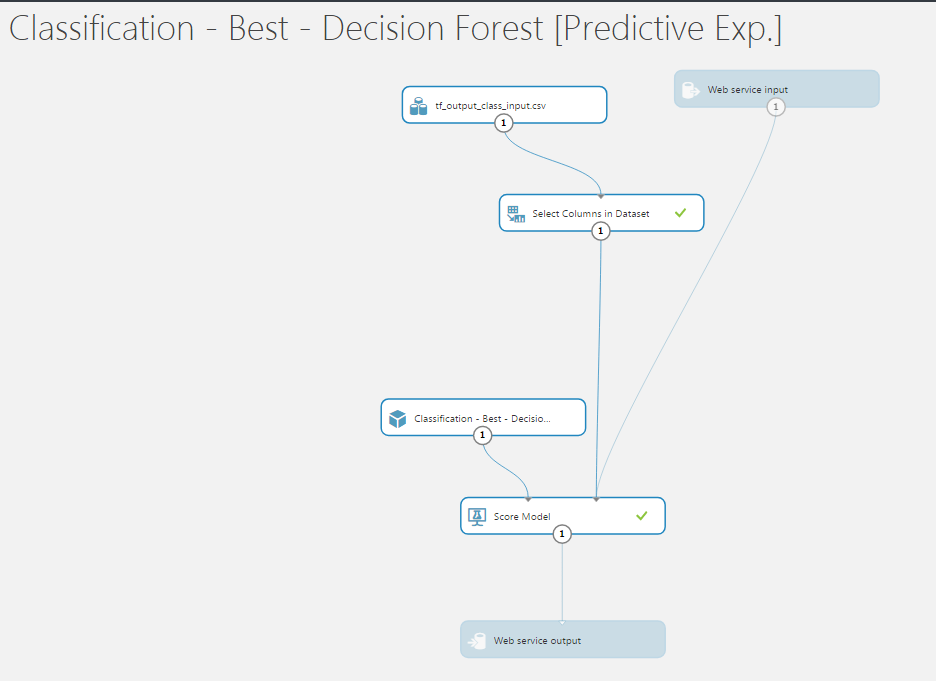
Azure Machine Learning Accuracy:

|  |  |  |  |
| --- | --- | --- | --- |
| Model | Logistic Regression | Neural Networks | Random Forest |
| Accuracy | 83.8% | 84.1% | 84.7% |

With a slight variation, even on Azure Machine Learning Studio Random Forest proved to be the best algorithm for our data in classifying the blogs as relevant or not relevant.



**Classification Web Service - Decision Forest**



We have implemented our analysis on live data as well as historical data.

We fetched the live data from the webhose API, but Azure Machine Learning Studio does not take textblob data and hence creating web service was not possible. So, as a workaround we still have downloaded the lived data the previous day, performed TF-IDF on it and then classifying it.

The data on which we are presenting is the one generated a day before and performing all the mentioned tasks on it.

**LDA Clustering:**

We have performed clustering on blogs that are classified as relevant after classification using LDA (Latent Dirichlet Allocation) algorithm

**Approach:**

* For each blog, take the text and convert it to lowercase
* Find tokens in this lowercase text using nltk tokenizer
* Stem each word to get its basic form
* Remove stop words from these stemmed words
* Append these tokens into a list on which lda model shall be implemented
* Convert these tokenized documents into an id term dictionary
* Get a document -term matrix using doc2bow (document to bag of words) function
* Implement gensim LDA model for the above generated dictionary and document-term matrix
* This gensim LDA model is trained to display 3 clusters after 100 passes in each blog
* Next, we consider top 5 words from these 3 clusters and append it to 3 separate lists called word\_list1[], word\_list2[] and word\_list3[] respectively.
* We check each word of these word\_list[] in the bag\_of\_words generated after implementing tf-idf algorithm
* If the word in a word\_list[] belongs to a particular type of bag\_of\_words, we consider this blog to belong to that particular blog type.

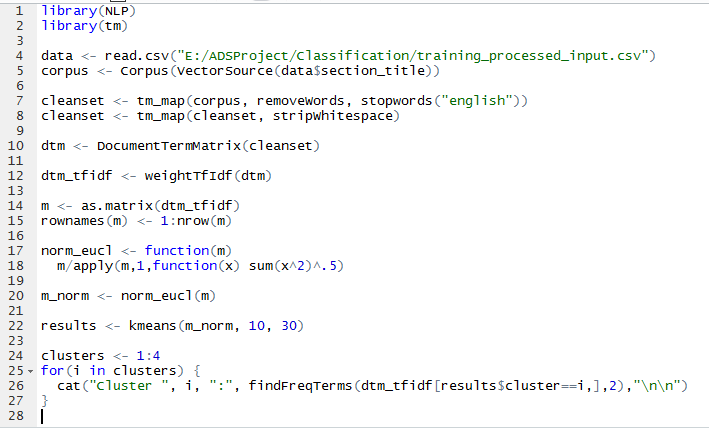
**Output:**

LDA algorithm would finally help us get the count of blogs for each topic.

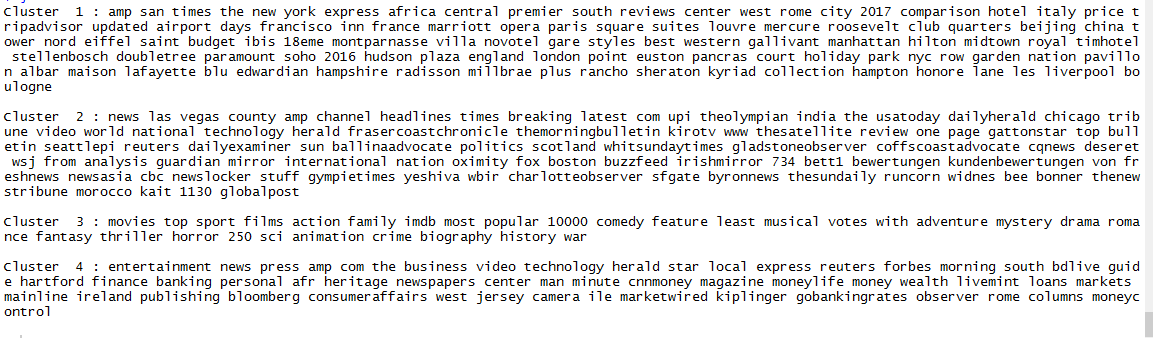


**K-Means Clustering**

We also tried clustering on the basis of TF-IDF on R, as to get the most listed words in a blog or message boards or discussions. We obtained the TF-IDF score based on the weight of the words used, created a document matrix for the cleaned input file. We then calculated the euclidean distance based on the words in the Term Documentation Matrix to get the words in a particular cluster. Then, we provide the number of clusters we want to create and loop across to get that many clusters and data divided among those clusters based on the TF-IDF score.



**Cluster Output:**

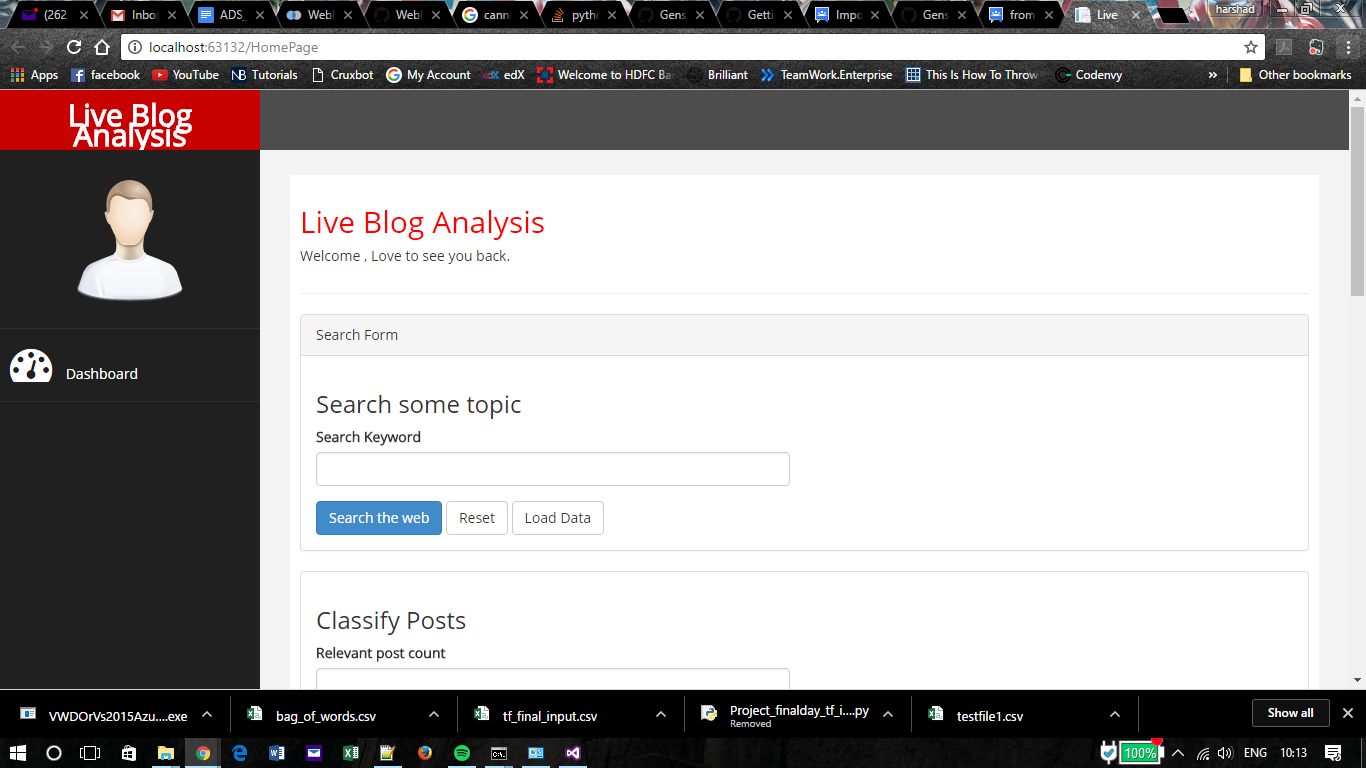


**User Interface:**

A C# application is created to build an user interface which can connect to the web services deployed on azure machine learning studio. Web forms are created in asp.net which uses server side scripts to run the C# code. A simple user interface is created which takes input as a search keyword from the user. This search keyword is then passed to the Webhose api which returns 100 blogs which contains this search keyword.

User can input the search keyword in a text box and click ‘***search the web button’*** . This will initiate a function which will form a webhose query and fire that query as a http request. Response of this will be 100 blogs and post containing searchkeyword in a separate json format. As we are getting all the data in one response, we have implemented this as a synchronous call. Further these json objects are parsed one by one to display on web page.

We have also implemented a functionality which can read the live data stored over past few days from a csv file. We perform TDIDF on this data and calculate score of the search keyword, which was used while downloading the data. Further, this data is classified as relevant data or irrelevant blogs. A web service is called to classify as a post as relevant data or irrelevant blogs by using scores and weight of the words.



Once the data is classified as relevant and non relevant, we can move forward with the relevant data. This data is given to LDA one by one which will cluster the words in the document in 3 clusters according to the topic and weights. We compare words in these clusters from the Bag of words and the matching word is moved into the particular section among the 9 mentioned.

**Visualization - Tableau:**

A picture is worth a million rows. Expressive visualization enables you to get beyond static charts to create multi-faceted views of data and explore every dimension. We extracted the data, cleaned and pre-processed it, analyzed the data and then final task of making the data insights visually appealing using Tableau.

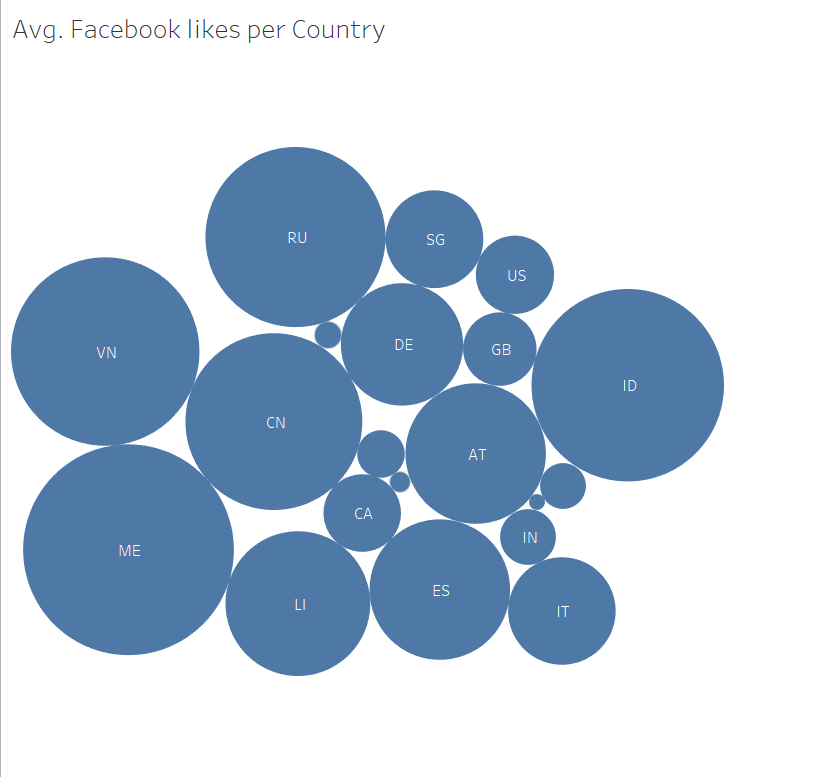
Analysis is done for two reasons:

i) to get favorable insights

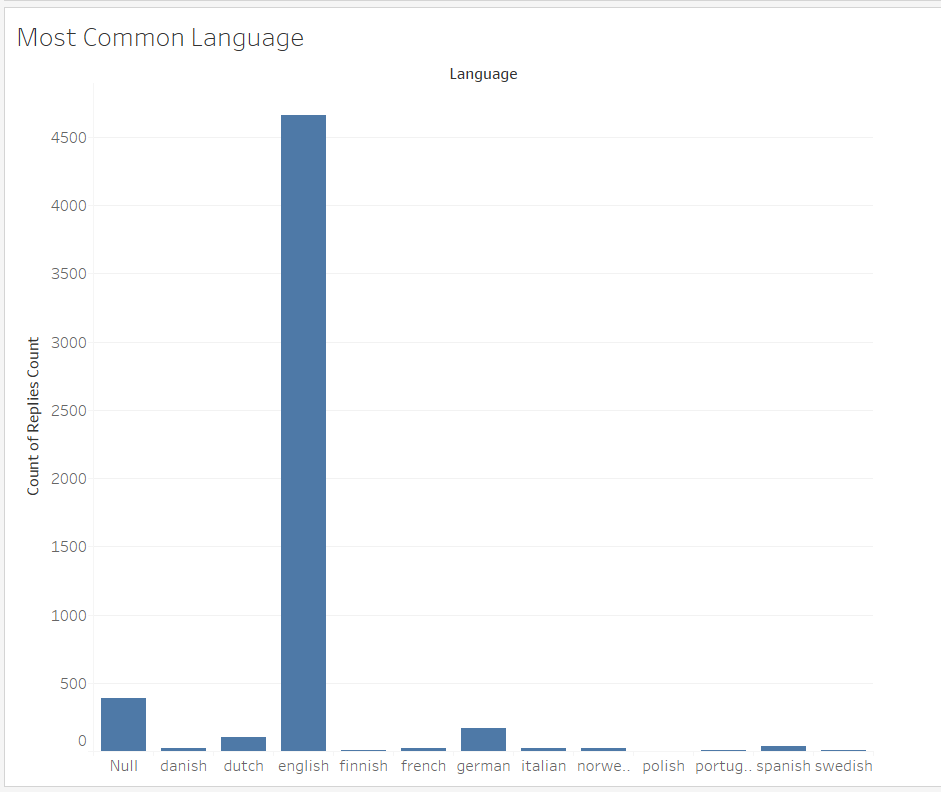
ii) for statistical purpose

Based on the data we had, we analyzed and visualized a list of few on both types:

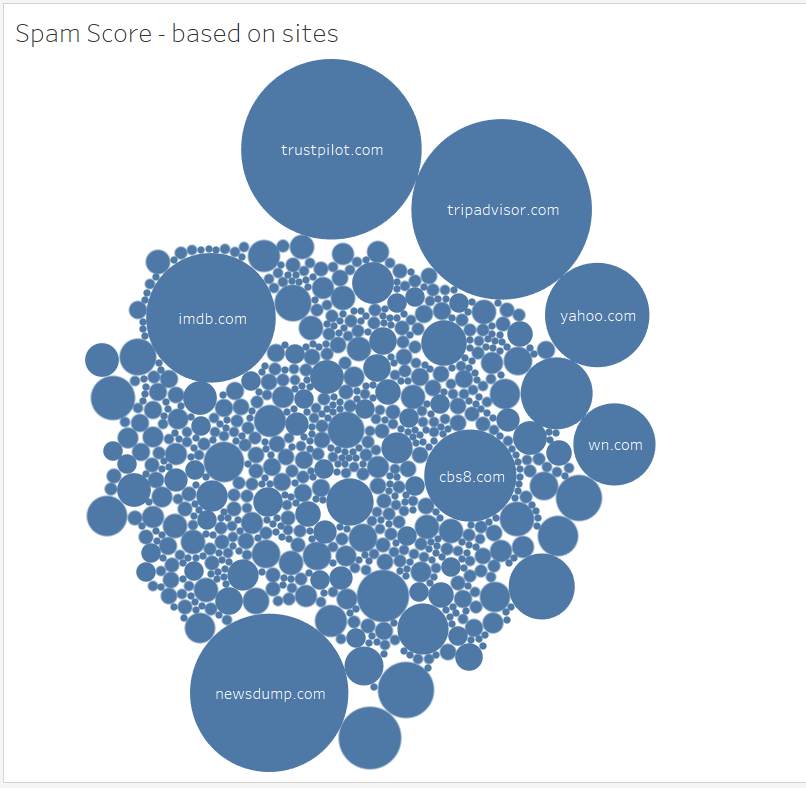
1. Average facebook likes per Country - This gives us the Country wise Facebook likes, which will help the advertisers understand the Country to target and the Country where people are socially more active.



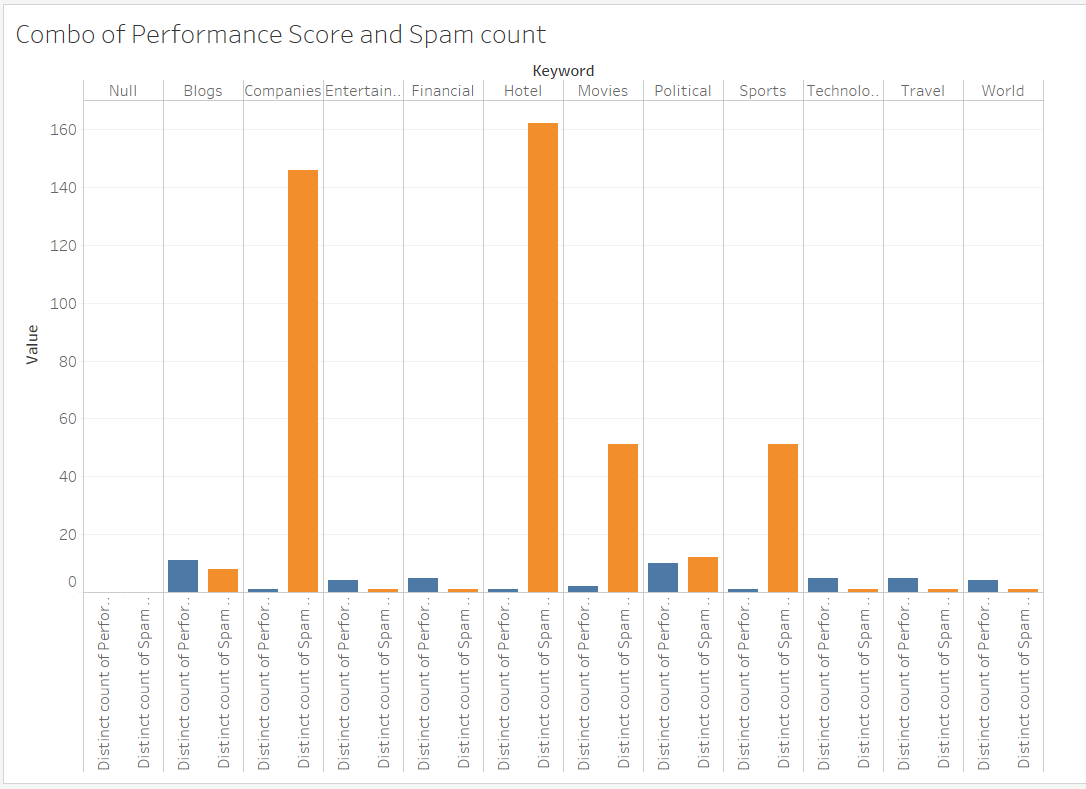
2) This is a statistical analysis which helps us know the most common language currently in the world or the most common language in which people are socially active.



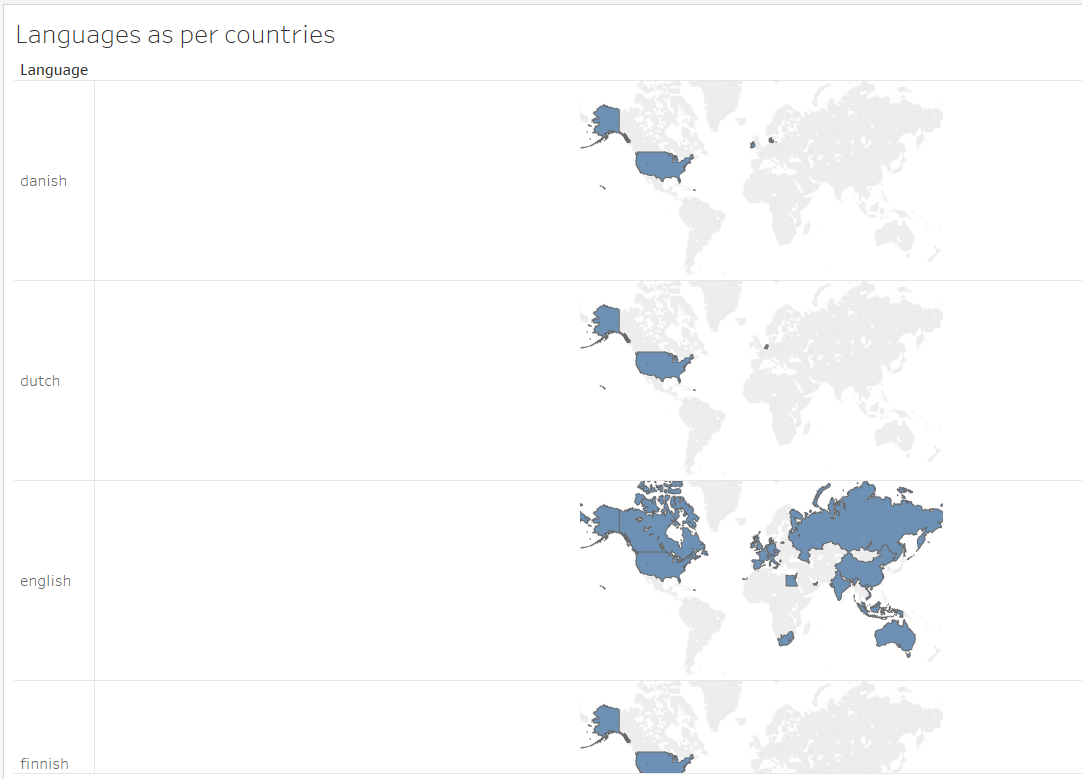
3) Internet blogs sites are more prone to spams and abuses. The below analysis gives us the most number of spams reported for any given website. The count of spams reported on each site over the month of March 2017 and the spam score given to each site. This helps the user understand the amount of spams a particular website might have, as compared to others.



4) This is a really interesting visualization with multi-measure analysis. The below graph makes us understand the performance score and the spam score are inversely proportional (except for a few exceptions like Political news). Whenever the spam score of a site is more, it’s performance score is less and vice-versa.



5) This visualization takes Geographical input in the form of Longitudes and Latitudes and the blogs or reviews or message boards arrived from that country based on Languages. This helps us get the most common languages spoken in any given Country. From the below graph, we could conclude that US is a Country where every language is spoken, which shows the mixed diversity existing in the Country.



-- Finally the one below is the dashboard which has the combination of all different visualizations and it has been published on the Tableau server, which can be accessed using the below link:

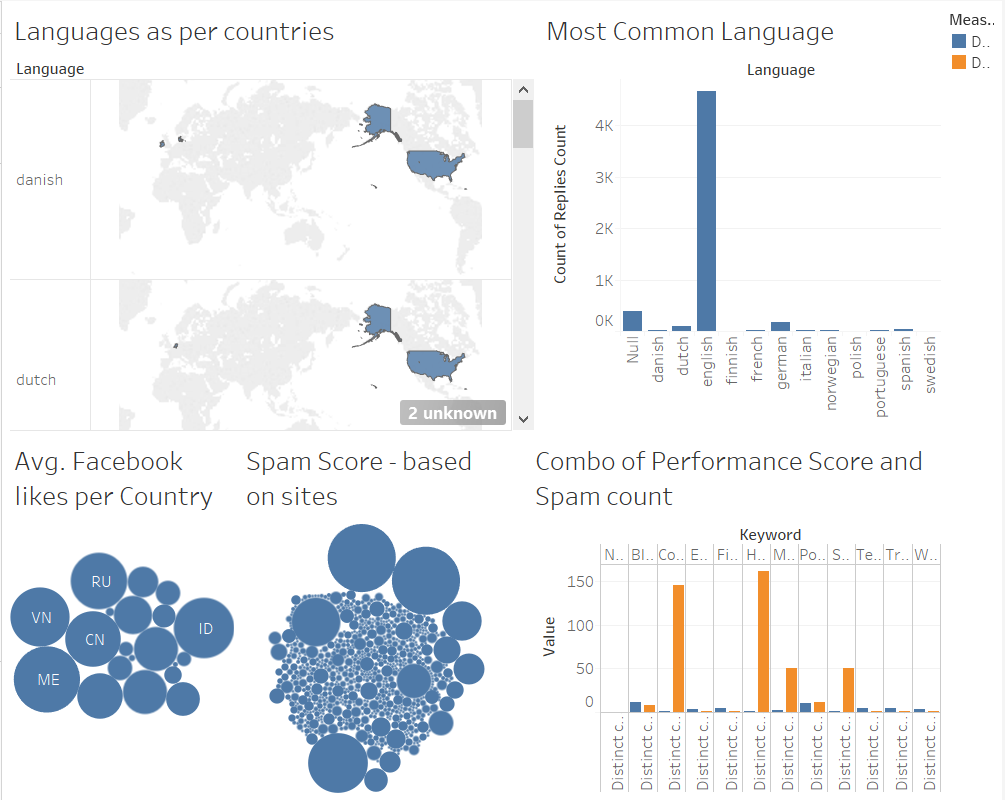


Tableau visualization are published online and can be accessed from below link:

<https://public.tableau.com/profile/publish/Project_Visualization_0/Dashboard#!/publish-confirm>

**References**

<https://webhose.io/datasets>

<https://www.analyticsvidhya.com/blog/2017/01/t-sne-implementation-r-python/>

<http://stevenloria.com/finding-important-words-in-a-document-using-tf-idf/>

**Github Link:**

<https://github.com/Tejesh0711/Blog-Analysis>

**Demo Link:**

<http://adsfinalprojectteam10.azurewebsites.net/HomePage>