BIA 678: Big Data Technologies

Project Report: Twitter Stream Analysis

Instructor: Professor David Belanger

From: Syed Zain Raza, Sushmith Ramesh, Hadia Hameed

Team 10 (Section B)

Dec 6th, 2019

Introduction:

When it comes to big data, Twitter is perhaps at the forefront of providing useful data for potential analysis. Data obtained from Twitter fulfills the five Vs of big data: Velocity, Volume, Value, Variety, and Veracity. It is said to generate more than 400 million messages per day which can provide some useful insights on user behavior and trending discussions around the world.

Goals:

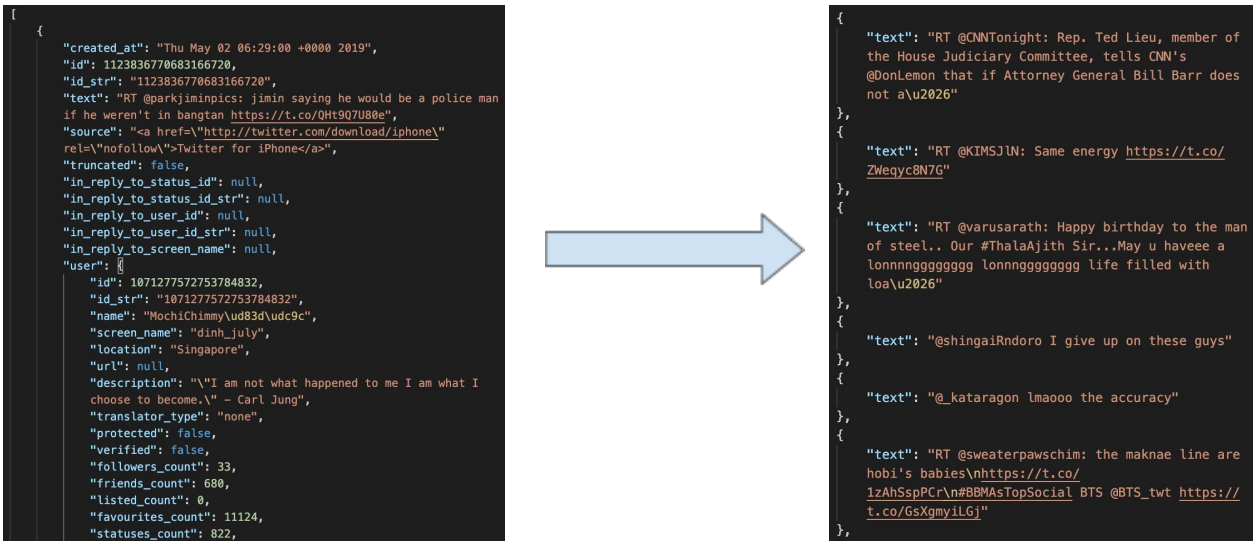
In our project, we aimed to analyze Twitter streaming data for three main applications:

1. **Word Cloud:** Getting the most frequent words used by people in their tweets.
2. **Sentiment Analysis:** understanding whether people posted more about positive or negative things by doing sentiment analysis.
3. **Clustering:** Discovering any underlying pattern in tweeting behavior by doing clustering.

Data:

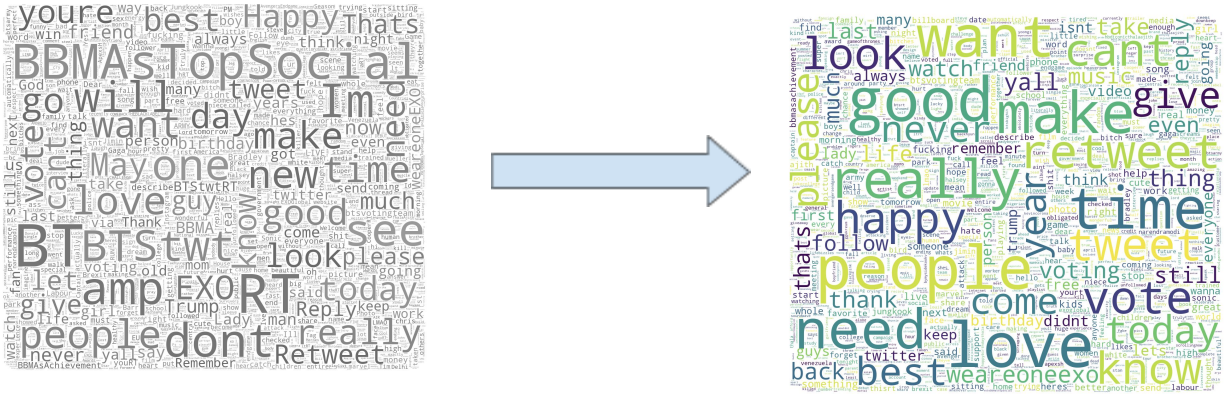
We used two datasets: the first one was collected directly using Twitter API and was published on a Kafka topic, the second was obtained from Internet Archives that has tones of open-source datasets and other resources. The data had tweets in many different languages, and apart from the actual text of tweets, each object in the dataset had more than thirty keys such as user profile details, timestamp, followers' information, and other metadata about the tweet. The figure below

shows the part of the actual json object and what we ended up using after filtering out the unnecessary keys. The final dataset we used had more than 11 million tweets.



Data Cleaning

Data cleaning steps included removing stop words, meta characters, urls, mentions, hashtags, and as an extra step in pre-processing, all words less than 4 characters long were removed.



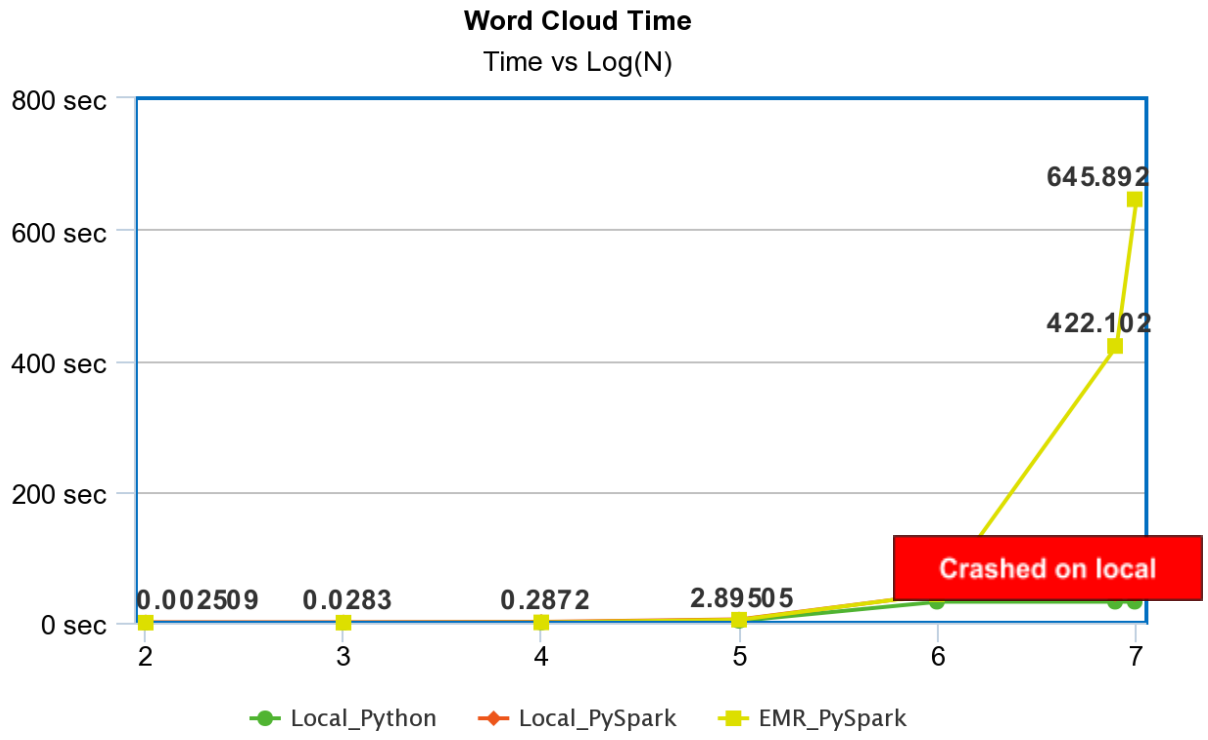
Word Cloud

Getting a Word Cloud involved a simple word count Spark job in which a mapper and reducer were used to generate word frequency pairs and the top N pairs were selected to visualize the word cloud. The figure below shows the top N = 5000 words.


```
('love', 351914)
('people', 316985)
('time', 287387)
('know', 267949)
('good', 265023)
('today', 226594)
('want', 219963)
('really', 205556)
('need', 188643)
('back', 186450)
```

The graph below shows timing for the word count job on EMR with 3 clusters versus the same job on local machine. The x-axis shows log(n) where n is the number of tweets used. While the performance was comparable for small number

of tweets, for more than 8M tweets, the code on the local machine crashed but on EMR it gave the results after almost 600s.



K-Means Clustering

Extracting Features:

While the algorithm behind K-Means clustering is relatively straightforward, it is important to note that it requires that we work with numbers rather than strings. Thus, in this particular case of Twitter data (specifically tweets), we must first convert the tweet strings to vectors of numbers

that represent those tweets in some fashion. To do this, the TF-IDF algorithm was used (TfidfVectorizer from sklearn.feature_extraction.text), where each tweet is represented by a vector of length of the dictionary of words across all the tweets (as this is huge, the max_features variable can be used to limit this number as needed), and the value in each element of the vector is calculated by multiplying the number of times that word occurs in the particular tweet (TF - this value is specific to each tweet) multiplied by the log of the number of tweets divided by the number of tweets in which the word occurs (IDF - this is specific to the word in the dictionary of words across the tweets). When computing these vectors, it is common for the TF-IDF product value of a particular word to be zero (especially when using the default dictionary of words across all the tweets for the length of each vector representing a tweet). As mentioned previously, the max_features variable can be used in this case to limit the number of words we use from the dictionary of words to what is specified and as a result produce a less sparse matrix overall.

K-Means:

As mentioned above, The K-Means algorithm is relatively straightforward, where random points are initially used as centers to cluster all of the data points together. After this, multiple iterations are run where the centers are recomputed (usually based on the average of each of the clusters of

points) and the points are again re-clustered accordingly (this process occurs, ideally, until the clusters no longer change). The difficulty, in this particular case, lies in the fact that we did not really know value of K beforehand (here, KMeans from sklearn.cluster was used). Typically, we could have used the number of unique users to cluster tweets from the same user together, however, this information was not available. Thus, we used different values of K and chose the value that produced ideal conditions in the visualizations discussed below. For future work and considerations, another option may have been to use the elbow method and compute the sum of squares for multiple different K values to find the optimal value. In this case, the clusters would more likely represent tweets that are similar in content to each other. Additionally, it may be useful to research other, potentially more optimal clustering algorithms (as well as distance metrics), as in this case the dimensionality of the data was relatively high.

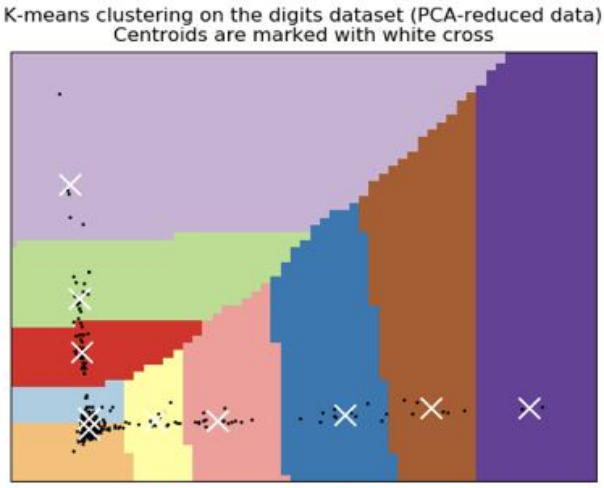
Visualization:

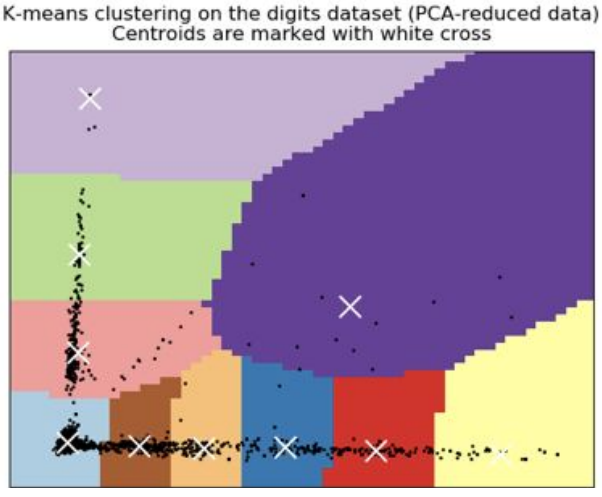
This was probably the most difficult process of clustering, mainly because the feature extracted data (after TF-IDF) was high in dimensionality. Thus, plotting and visualizing both the data and clusters would be difficult, as typically this is done with only 2 or 3 dimensions of data. As a result, dimensionality reduction techniques (PCA and TruncatedSVD from sklearn.decomposition) were used to reduce the high dimensional data to 2 dimensions to better visualize the data. One note of interest was that the PCA method from sklearn would not take a sparse matrix as input and required the matrix to be dense (as opposed to TruncatedSVD). Some future work that could be done with visualization involves linking the clustered and feature

extracted data (in the form of number vectors) back to their original tweet string data, so we can better see what exactly is being clustered (though the primary focus of this project was to evaluate the performance in various conditions).

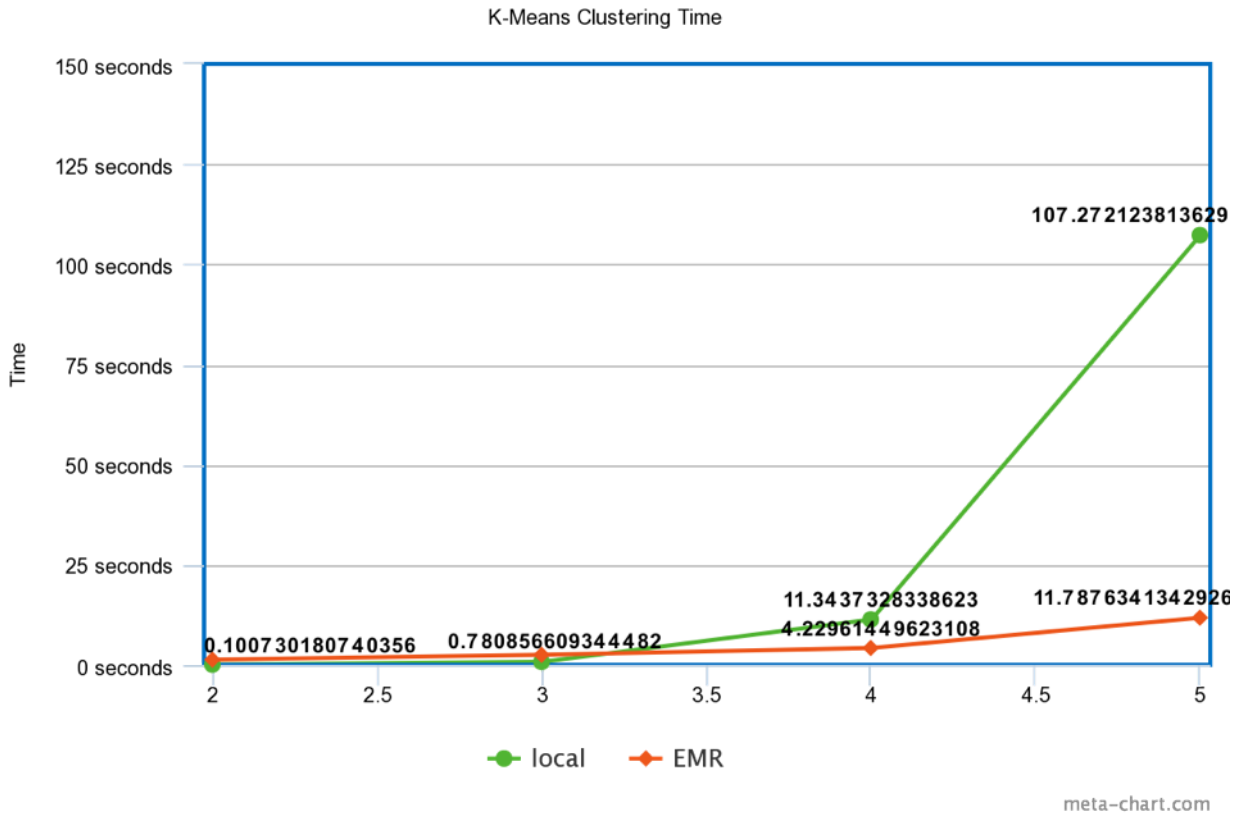
Results:

Again, as mentioned before the clustered data that was visualized could definitely be improved, but the primary goal here was to see that the data was being clustered in some fashion and focus more on the performance. The first image below represents running K-Means on 1000 data points/tweets with a K value of 10, while the second image below represents running K-Means on 10000 data points/tweets with a K value of 10.





The graph below shows the difference between running K-Means clustering on the Twitter data of increasing sizes on a local machine versus an AWS EMR cluster with 3 instances (the x-axis units are 10^(value on the axis), so 2 would represent 100, 3 would represent 1000, and so on – this was done to better space the data). As expected after 10000 tweets, we see significant performance benefits in using multiple instances with AWS EMR compared to the local machine. At the very beginning (with only 100 tweets), however, the local machine was actually a bit faster than the EMR cluster by a few seconds. This could be expected as, with such little data, directly running the code on the local machine would probably be faster than running it on the EMR clusters, where there may be some overhead involved in distributing the workload across multiple instances.



Sentimental Analysis

Labeling Tweets:

The tweets data we got after preprocessing only consisted of text. In order to train any sentiment analysis algorithm, we had to label the tweets. By doing this, we converted the task into a Supervised Learning task. As tweets consists of several jargons that are specially used in the context of social media, we decided to use a pre-trained model VADER to label the tweets.

VADER:

VADER was first introduced by Georgia Tech researchers and professors who were working on Natural Language Processing. It stands for Valence Aware Dictionary and Sentimental Reasoner. They trained the model specifically for social media text so that it will be able to assign correct polarity for a text used in any social media site.

The model is a general rule based and lexicon sentimental analysis model. Few of the advantages of this algorithm is that it can be used even for live streaming data and it does not suffer from most of the tradeoffs such as a speed-performance tradeoff. The authors in the paper of this method mentions that they used five general rules that helped it to better generalize sentiments. They also write that this method performs a lot better than other benchmark techniques such as SWN, WSD and ANEW.

Another great advantage of using this method is that performs really well on emojis and slangs used in social media settings. Following shows an example of this:

```
In [7]: sid.polarity_scores('We are good :D')
Out[7]: {'neg': 0.0, 'neu': 0.224, 'pos': 0.776, 'compound': 0.7865}
```

As it can be seen that it gives each text a positive, negative, neutral and compound percentages. Compound means that it is the sum of all ratings of lexicon. If compound score is above 0.05

then it is positive, if between 0.05 and -0.05 then it is neutral and if less than 0.05 then it is negative.

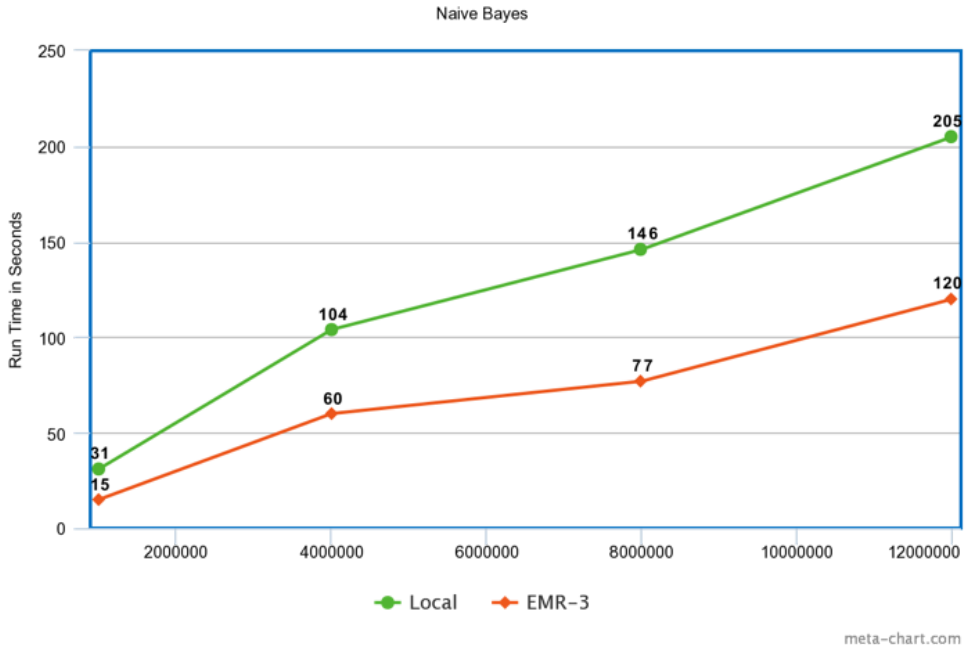
Training and Testing Data:

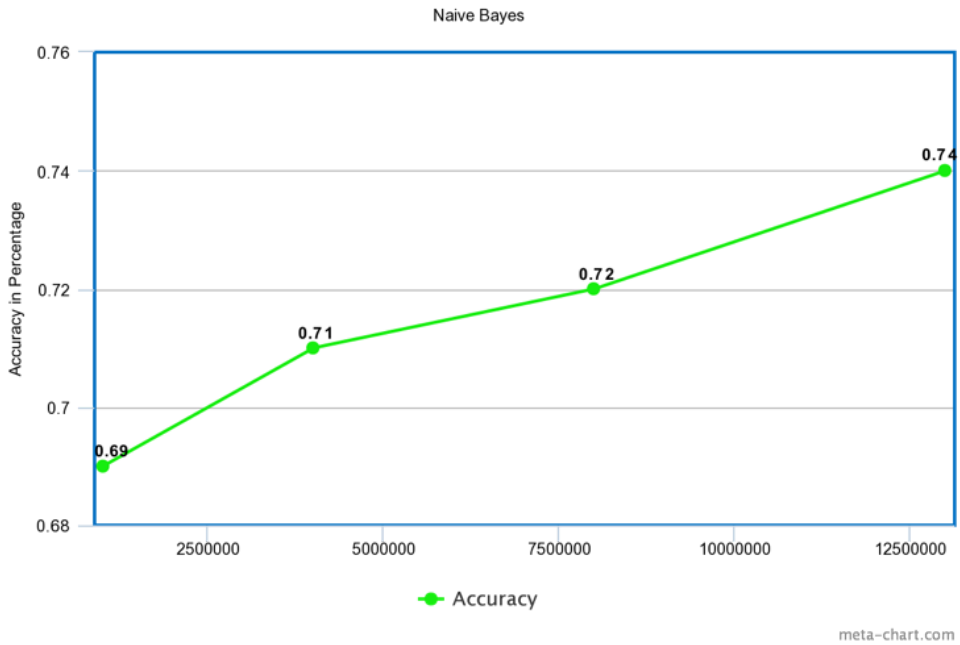
We used these percentages of positive, negative and compound ratings to give label 0 for negative tweets and label 1 for positive tweets. We ignored neutral percentages for this project. It turned out that there were 5739412 positive and 5513141 negative tweets which combines to be a total of 11252553 tweets. We concatenated them together and then randomly mixed them so that all the positive and negative tweets are not together. For training, we used three different machine learning algorithms which are Naïve Bayes, Logistic Regression and Support Vector Machines. We tried different training and testing sizes but the setting which gave the best performance was 75% of the training data and 25% of testing data.

In testing, we used classification accuracy as a measure of performance which is calculated on the number of correct predictions divided by the total number of predictions.

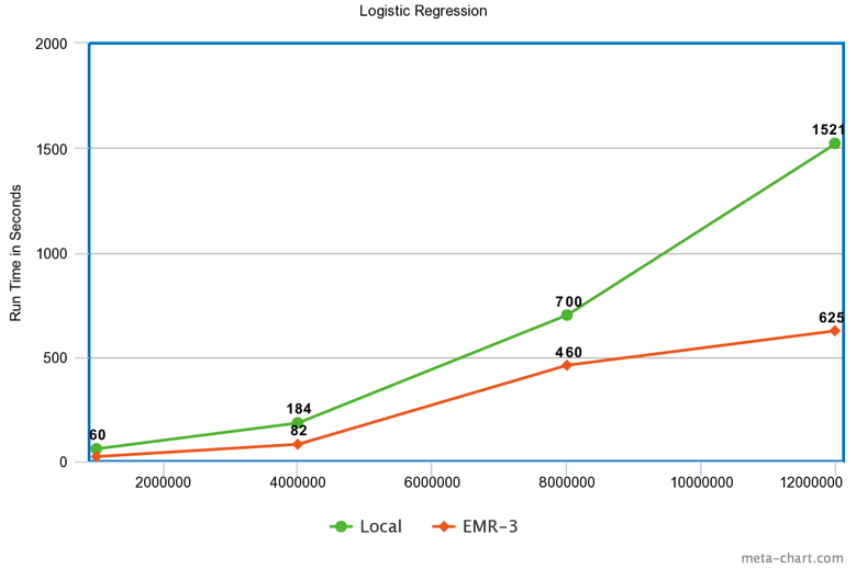
Following are the results of the different algorithms that we performed on Local (EC2 one instance) using Python only and EMR(Elastic Map Reduce) 3 instances using PySpark.

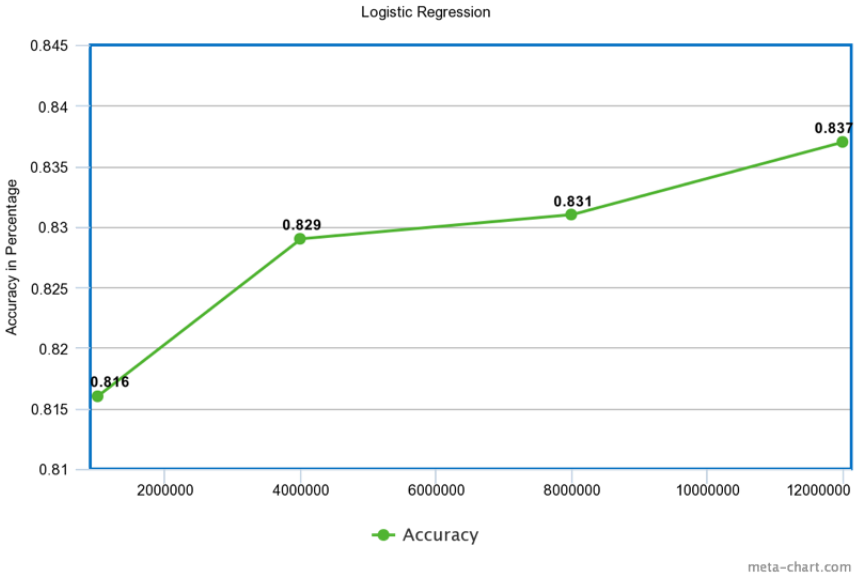
Naïve Bayes:



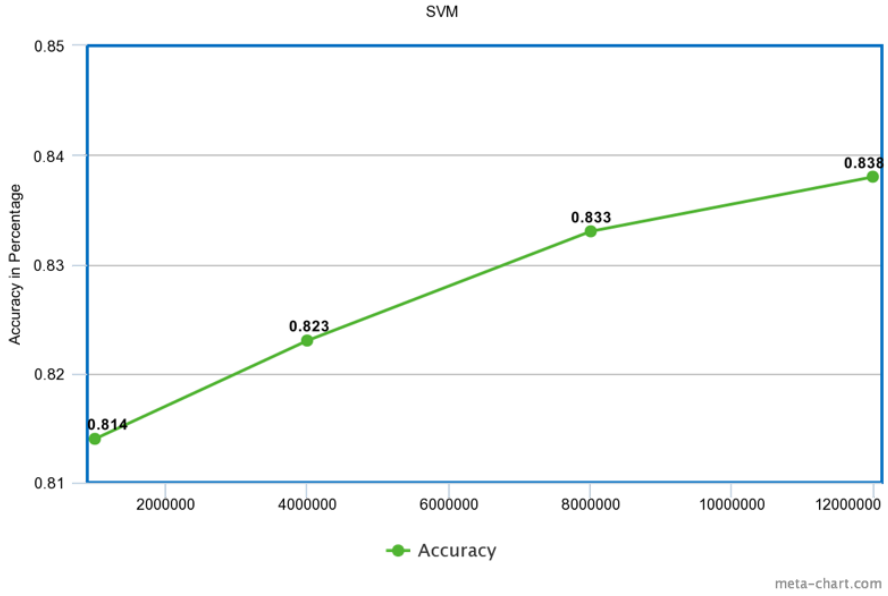


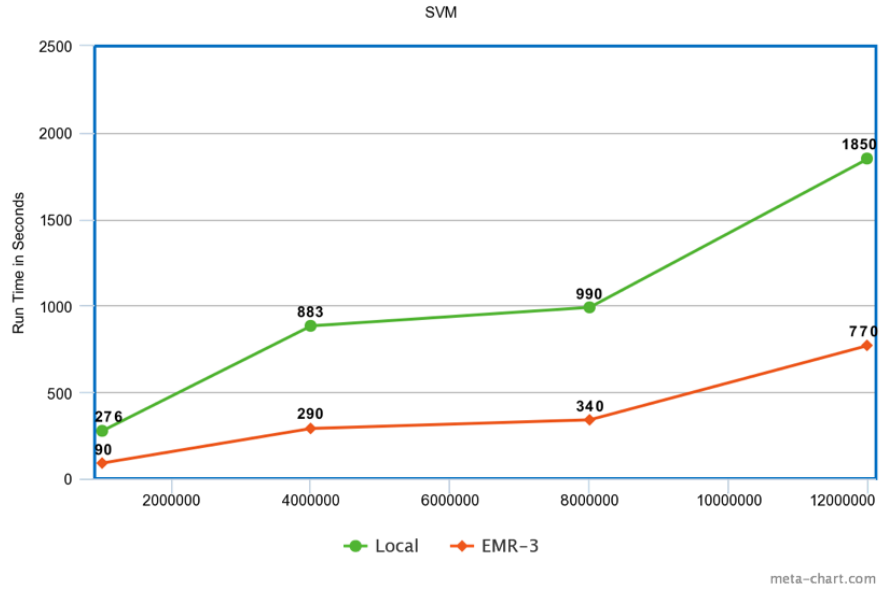
Logistic Regression:





Support Vector Machines:


SVM
0.85
0.84
0.83
0.82
0.81
Accuracy in Percentage
0.838
0.833
0.823
0.814
2000000
4000000
6000000
8000000
10000000
12000000
Accuracy
meta-chart.com


SVM
2500
2000
1500
1000
500
0
Run Time in Seconds
1850
990
883
770
340
290
276
90
2000000
4000000
6000000
8000000
10000000
12000000
Local
EMR-3
meta-chart.com

References:

1. https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_digits.html
2. https://www.freecodecamp.org/news/how-to-process-textual-data-using-tf-idf-in-python-cd2bbc0a94a3/
3. https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html
4. https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html#sklearn.decomposition.TruncatedSVD
5. https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html
6. https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html
7. http://comp.social.gatech.edu/papers/icwsm14.vader.hutto.pdf
8. https://medium.com/analytics-vidhya/simplifying-social-media-sentiment-analysis-using-vader-in-python-f9e6ec6fc52f