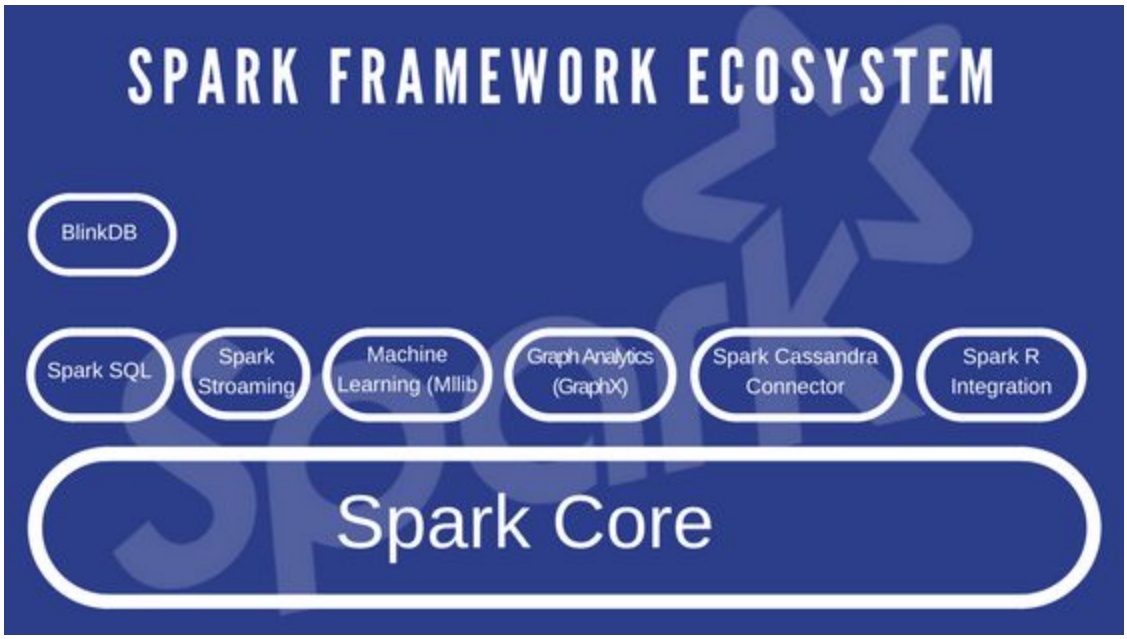
# Twitter Sentiment Analysis Using Apache Spark with Cluster Mode

By: Kartik Deshpande, Mohit Joshi, Sagar Rikame

## Inspiration:

Hadoop and Apache Spark both are today's booming open source Big data frameworks. Though Hadoop and Spark don't do the same thing, however, they are interrelated. The need of Hadoop is everywhere for Big data processing. However, Hadoop has a major drawback despite its many important features and benefits for data processing. MapReduce which is the native batch processing engine of Hadoop is not as fast as Spark. And that's where Spark takes an edge over Hadoop.




We implemented the code for performing sentiment analysis, both on Pyspark and Scala API model of the Apache Pyspark running with Spark Standalone mode. We used two models Logistic regression and Naive Bayes for python code and only the Naive Bayes model when implementing the application in Scala.

## Machine Learning Library (MLlib)

We have used Spark's machine learning library to implement the core functionality of sentiment analysis of the already extracted twitter data. Machine learning Library provides a lot of tools such as ML Algorithms, Featurization, Pipelines Persistence and Utilities.

Our focus here is ML Pipelines

**ML Pipelines** provide a uniform set of high-level APIs built on top of DataFrames that help us create and tune practical ML Pipelines

Some of the key concepts in the Pipeline API are as follows:

- DataFrame: This ML API uses DataFrame from Spark SQL as an ML dataset, which can hold a variety of data types.

- Transformer: A Transformer transforms one DataFrame into another DataFrame. It implements a method transform() which does the conversion. We used Hashing transformer for our code which maintains a map for term to their frequencies.

- Estimator: An Estimator is an algorithm which can be fit on a DataFrame to produce a Transformer. It implements a method fir() which accepts a DataFrame and produces a model which is a Transformer.

- Pipeline: A Pipeline chains multiple Transformers and Estimators together to specify an ML workflow. A simple text document processing workflow will include several stages:

    - Split each document's text into words
    - Convert each document's words into numerical feature vector
    - Learn a prediction model using the feature vectors and labels

This workflow is a pipeline which consists of sequence of stages to be run in a specific order.

- Parameter: All Transformers and Estimators now share a common API for specifying parameters.

The stages in the pipeline are either a Transformer or an Estimator
Above is a text document workflow, which is training time usage of a pipeline

The top rows represent the Pipeline with three stages
First two (Tokenizer and HashingTF) are Transformers and the third (Logistic Regression) is an Estimator.

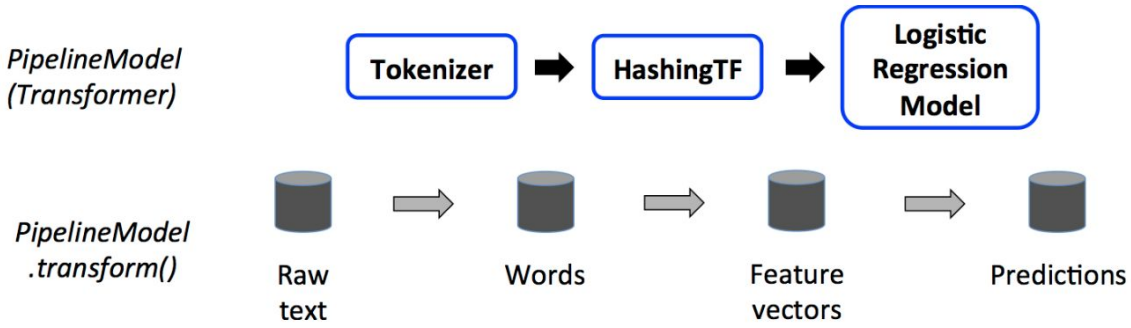
The bottom row represents the data flowing through the Pipeline where the cylinders indicate the data frame. The Pipeline.fit() is called on the original DataFrame which consists of raw text documents and labels.

The Tokenizer.transform() method splits the raw text documents into words adding a new column with those vectors to the DataFrame

Logistic Regression is an Estimator therefore the pipeline first calls LogisticRegression.fit() to produce a Logistic Regression Model.

If the Pipeline had more Estimators it would call the LogisticRegressionModel's transform() method on the DataFrame before passing the DataFrame to the next stage.

A Pipeline is an Estimator. Thus, after a Pipeline's fit() method runs, it produces a PipelineModel, which is a Transformer. This PipelineModel is used at test time; the figure below illustrates this usage

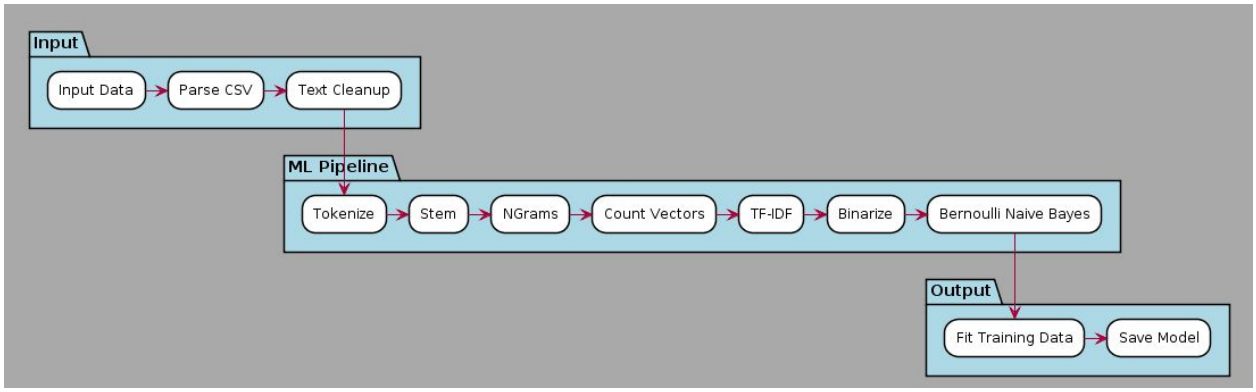



The PipelineModel has the same number of stages as the original Pipeline, but all Estimators in the original Pipeline have become Transformers.
When the PipelineModel's transform() method is called on a test dataset, the data are passed through the fitted pipeline in order. Each stage's transform() method updates the dataset and passes it to the next stage.

Pipelines and PipelineModels help to ensure that training and test data go through identical feature processing steps.

## Overall Structure of the pipeline :

Here is the overall steps occurred during the processing of pipeline

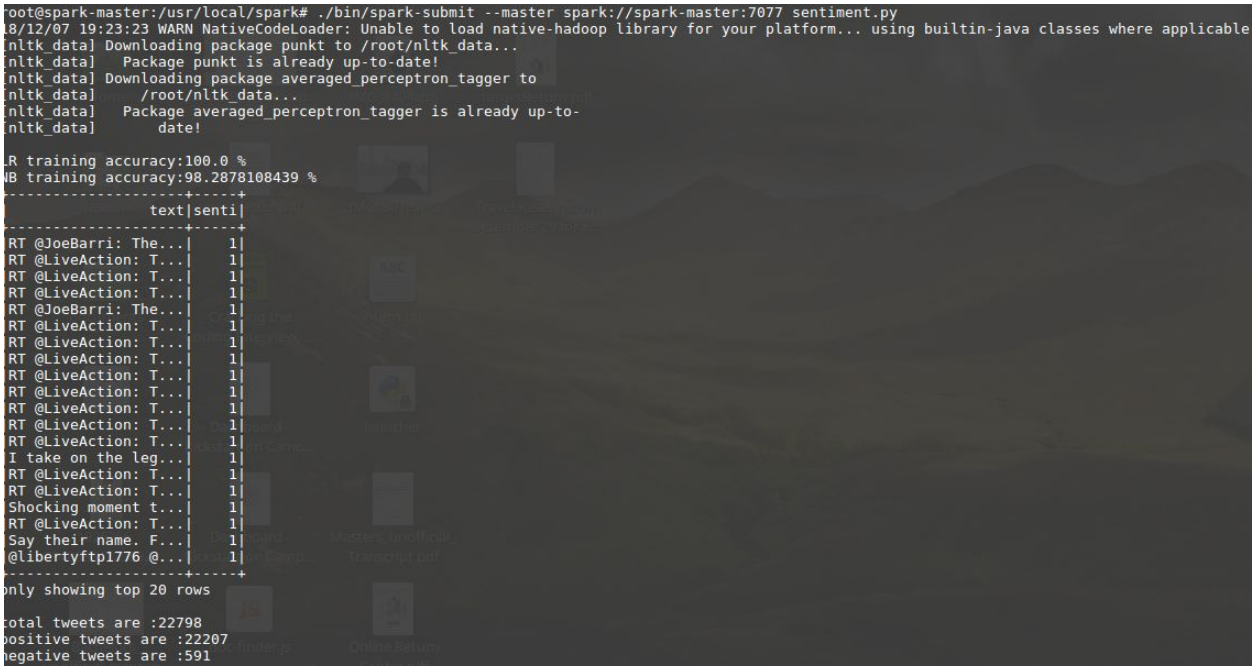



Following is the explanation of the working of pipeline:

Input data: is the dataset read and forms the base for training
Parse Data: parsing stage picks the data line by line
Text Cleanup: Text is cleaned up removing any html/xml encodings, removing twitter handles, converting to lowercase, extra white symbols removed and so on
Tokenize: The cleaned up data is used by tokenizer to create a bag of words.
Stem: The individual words than can be stemmed further to reduce word to its root.
Count Vectors: makes a counts of the stemmed words
TF-IDF: Reduce the importance of very frequently occuring words in training data to impact the results.
Naive Bayes: is a probabilistic model that uses Bayes's Theorem with the assumption of independence between features. This section of code provides conditional probability to the each feature.
Fit Training Data: Our ML pipeline is fitted with this training data.
Save Model : This training model can now be saved to file system so that can be used later to analyse the sentiment data.

## Below is the screenshot for the python version of sentiment analysis:

This case was run on one master and two worker nodes



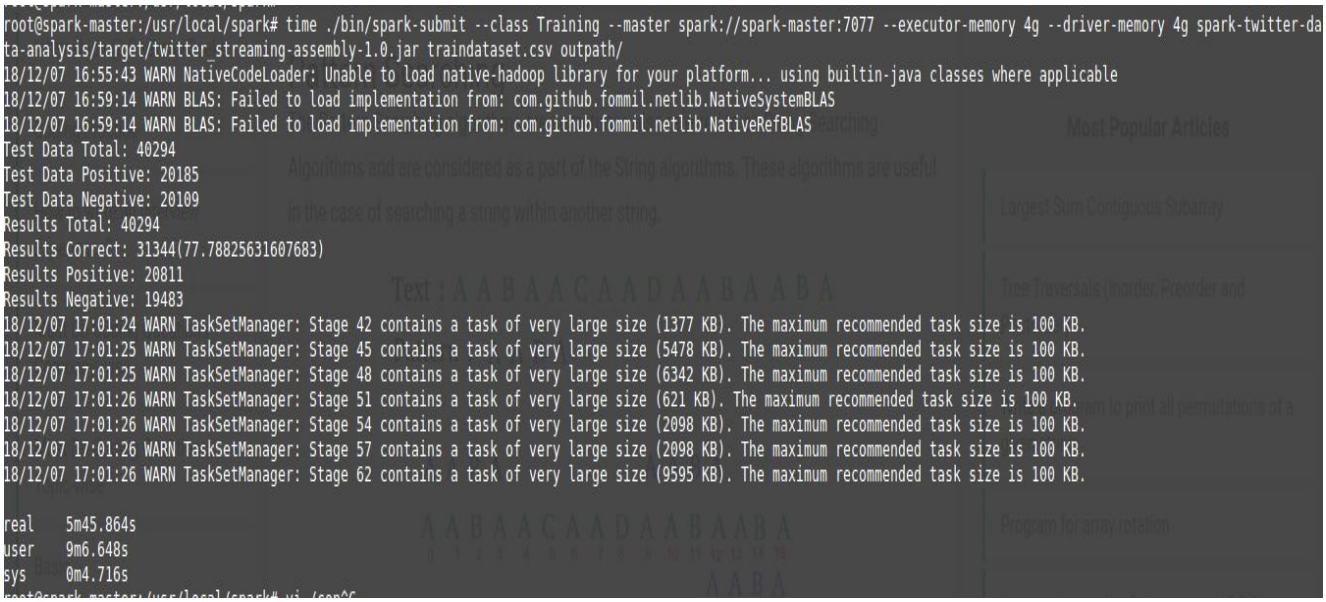
```
root@spark-master:/usr/local/spark# ./bin/spark-submit --master spark://spark-master:7077 sentiment.py
18/12/07 19:23:23 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!

LR training accuracy:100.0 %
NB training accuracy:98.2878108439 %
+--------------------+-----+
|                text|senti|
+--------------------+-----+
|RT @JoeBarri: The...|    1|
|RT @LiveAction: T...|    1|
|RT @LiveAction: T...|    1|
|RT @LiveAction: T...|    1|
|RT @JoeBarri: The...|    1|
|RT @LiveAction: T...|    1|
|RT @LiveAction: T...|    1|
|RT @LiveAction: T...|    1|
|RT @LiveAction: T...|    1|
|RT @LiveAction: T...|    1|
|RT @LiveAction: T...|    1|
|RT @LiveAction: T...|    1|
|RT @LiveAction: T...|    1|
|I take on the leg...|    1|
|RT @LiveAction: T...|    1|
|RT @LiveAction: T...|    1|
|Shocking moment t...|    1|
|RT @LiveAction: T...|    1|
|Say their name. F...|    1|
|@libertyftp1776 @...|    1|
+--------------------+-----+
only showing top 20 rows

total tweets are :22798
positive tweets are :22207
negative tweets are :591
```

Log file on one of the slaves.

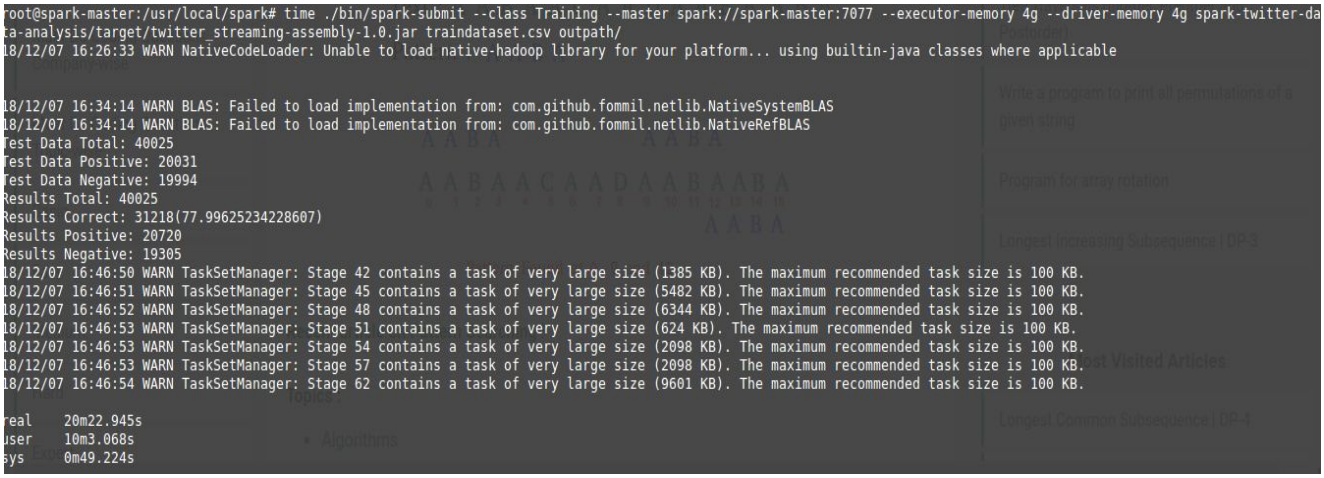
```
2018-12-07 19:24:36 INFO  ExecutorRunner:54 - Launch command: "/usr//bin/java" "-cp" "/usr/local/spark/conf/:/usr/local/spark/jars/*" "-Xmx1024M" "-Dspark.driver.port=4
6508" "org.apache.spark.executor.CoarseGrainedExecutorBackend" "--driver-url" "spark://CoarseGrainedScheduler@localhost:46508" "--executor-id" "49" "--hostname" "127.0.
0.1" "--cores" "2" "--app-id" "app-20181207192325-0000" "--worker-url" "spark://Worker@127.0.0.1:37259"
2018-12-07 19:24:39 INFO  Worker:54 - Executor app-20181207192325-0000/49 finished with state EXITED message Command exited with code 1 exitStatus 1
2018-12-07 19:24:39 INFO  Worker:54 - Asked to launch executor app-20181207192325-0000/51 for SentimentAnalysis
2018-12-07 19:24:39 INFO  SecurityManager:54 - Changing view acls to: root
2018-12-07 19:24:39 INFO  SecurityManager:54 - Changing modify acls to: root
2018-12-07 19:24:39 INFO  SecurityManager:54 - Changing view acls groups to:
2018-12-07 19:24:39 INFO  SecurityManager:54 - Changing modify acls groups to:
2018-12-07 19:24:39 INFO  SecurityManager:54 - SecurityManager: authentication disabled; ui acls disabled; users  with view permissions: Set(root); groups with view per
missions: Set(); users  with modify permissions: Set(root); groups with modify permissions: Set()
2018-12-07 19:24:39 INFO  ExecutorRunner:54 - Launch command: "/usr//bin/java" "-cp" "/usr/local/spark/conf/:/usr/local/spark/jars/*" "-Xmx1024M" "-Dspark.driver.port=4
6508" "org.apache.spark.executor.CoarseGrainedExecutorBackend" "--driver-url" "spark://CoarseGrainedScheduler@localhost:46508" "--executor-id" "51" "--hostname" "127.0.
0.1" "--cores" "2" "--app-id" "app-20181207192325-0000" "--worker-url" "spark://Worker@127.0.0.1:37259"
2018-12-07 19:24:42 INFO  Worker:54 - Executor app-20181207192325-0000/51 finished with state EXITED message Command exited with code 1 exitStatus 1
2018-12-07 19:24:42 INFO  Worker:54 - Asked to launch executor app-20181207192325-0000/53 for SentimentAnalysis
2018-12-07 19:24:42 INFO  SecurityManager:54 - Changing view acls to: root
2018-12-07 19:24:42 INFO  SecurityManager:54 - Changing modify acls to: root
2018-12-07 19:24:42 INFO  SecurityManager:54 - Changing view acls groups to:
2018-12-07 19:24:42 INFO  SecurityManager:54 - Changing modify acls groups to:
2018-12-07 19:24:42 INFO  SecurityManager:54 - SecurityManager: authentication disabled; ui acls disabled; users  with view permissions: Set(root); groups with view per
missions: Set(); users  with modify permissions: Set(root); groups with modify permissions: Set()
2018-12-07 19:24:42 INFO  ExecutorRunner:54 - Launch command: "/usr//bin/java" "-cp" "/usr/local/spark/conf/:/usr/local/spark/jars/*" "-Xmx1024M" "-Dspark.driver.port=4
6508" "org.apache.spark.executor.CoarseGrainedExecutorBackend" "--driver-url" "spark://CoarseGrainedScheduler@localhost:46508" "--executor-id" "53" "--hostname" "127.0.
0.1" "--cores" "2" "--app-id" "app-20181207192325-0000" "--worker-url" "spark://Worker@127.0.0.1:37259"
2018-12-07 19:24:45 INFO  Worker:54 - Executor app-20181207192325-0000/53 finished with state EXITED message Command exited with code 1 exitStatus 1
2018-12-07 19:24:45 INFO  Worker:54 - Asked to launch executor app-20181207192325-0000/55 for SentimentAnalysis
2018-12-07 19:24:45 INFO  SecurityManager:54 - Changing view acls to: root
2018-12-07 19:24:45 INFO  SecurityManager:54 - Changing modify acls to: root
2018-12-07 19:24:45 INFO  SecurityManager:54 - Changing view acls groups to:
2018-12-07 19:24:45 INFO  SecurityManager:54 - Changing modify acls groups to:
2018-12-07 19:24:45 INFO  SecurityManager:54 - SecurityManager: authentication disabled; ui acls disabled; users  with view permissions: Set(root); groups with view per
missions: Set(); users  with modify permissions: Set(root); groups with modify permissions: Set()
2018-12-07 19:24:45 INFO  ExecutorRunner:54 - Launch command: "/usr//bin/java" "-cp" "/usr/local/spark/conf/:/usr/local/spark/jars/*" "-Xmx1024M" "-Dspark.driver.port=4
6508" "org.apache.spark.executor.CoarseGrainedExecutorBackend" "--driver-url" "spark://CoarseGrainedScheduler@localhost:46508" "--executor-id" "55" "--hostname" "127.0.
0.1" "--cores" "2" "--app-id" "app-20181207192325-0000" "--worker-url" "spark://Worker@127.0.0.1:37259"
2018-12-07 19:24:46 INFO  Worker:54 - Asked to kill executor app-20181207192325-0000/55
2018-12-07 19:24:46 INFO  ExecutorRunner:54 - Runner thread for executor app-20181207192325-0000/55 interrupted
2018-12-07 19:24:46 INFO  ExecutorRunner:54 - Killing process!
2018-12-07 19:24:46 INFO  Worker:54 - Executor app-20181207192325-0000/55 finished with state KILLED exitStatus 143
2018-12-07 19:24:46 INFO  ExternalShuffleBlockResolver:186 - Application app-20181207192325-0000 removed, cleanupLocalDirs = true
2018-12-07 19:24:46 INFO  Worker:54 - Cleaning up local directories for application app-20181207192325-0000
```

## Below are the results when the application was run as Scala application:

Application running with two slave workers




The application running with no slave workers



## Learning:

- Learned the Pyspark API how to code for the parallelization of the resources on the spark framework.
- Learned how to code in scala for running application on spark.
- Learned Sbt(simple build tool) to build jars for the scala packages.
- Learned the tweepy API search, update features as well as the rate limitations.
- Learned Twarc: In order to visualize, save, and explore the vast number of tweets that have appeared under various accounts, we decided to turn to Twarc. The dataset generated by us  is generated by twarc, using the tweet ids archived dataset.
- Learned the MLlib library of pyspark.
- Learned to build our own docker image.

## Future Work:

The idea is to assess different avenues to explore the other areas where Spark can be applied to the already known data intensive application.

We also want to explore the true power of spark by using different distributed file system such as Apache Cassandra, Hive etc.