Fall 2016

COMP-SCI 5540 - Principles of Big Data

Group 4

Professor Anas Katib

Phase 2 Report

Mining Twitter data from the first presidential debate between Hillary Clinton and Donald Trump collected between 7 – 9:45pm on September 26, 2016

Team Members:

Todd Brees

Bill Capps

Luke McDuff

Submission Information

All queries can be found at the following location in submission folder: “Big_Data_Fall_2016\Source\Word_Count_Tweet_Collector\src\main\scala”. All query output files can be found at the following location: “Big_Data_Fall_2016\Source\Word_Count_Tweet_Collector\Phase_2_output”. All output files are labeled according to their query and are either CSV or TXT. All input files used can be found at: https://drive.google.com/drive/folders/0B0o8Y-H909qFNS1fdjVnNEZEVjg?usp=sharing

Theme

Politics is a subject that can cause a stirring of the masses. It can awaken the inner voice and inspire those who ordinarily would not be moved, to enlist in a cause. The most current presidential election in the United States is no exception to this rule, and in fact more people than ever are taking part and speaking their voice, fighting for their causes. This election in particular is causing such a stir on social media it has been termed the “Facebook Election” because so much discussion surrounding the politics of the election is taking place on social media platforms.

For our project we felt that it would be relevant to attempt to capture some of the excitement on social media towards the election by capturing Twitter tweets and running analysis on the data using techniques learned from the class curriculum. In particular, we captured tweets during the entire two hours of the first debate between presidential candidates Donald Trump and Hillary Clinton on September 26, 2016. During this time, we were able to collect 200,000 tweets. From these tweets we hoped to be able to answer questions from basic to complex like which candidate was discussed the most and what was the overall sentiment towards that candidate. Further explanation of the queries will take place in latter sections.

Architecture

The current architecture of the system is fairly straightforward. All the queries are run completely from the IntelliJ IDE using Scala as the predominant programming language. Java is used to supplement Scala with user defined functions (UDF) that include requests to the Stanford Core NLP API library, which we use particularly to access the sentiment analysis library provided. Within Scala, we are accessing data structures and methods from the Spark framework which include both RDDs and Dataframes. Spark, if more nodes were available, would divide the job across different nodes, however since this is run on a single machine only a single worker is enabled. The data input source consists of text file containing all 200,000 tweets as JSON strings. All of these technologies are open source. This architectural flow allows us to perform queries over the data fast and efficiently. The following is an architecture diagram of the current phase:

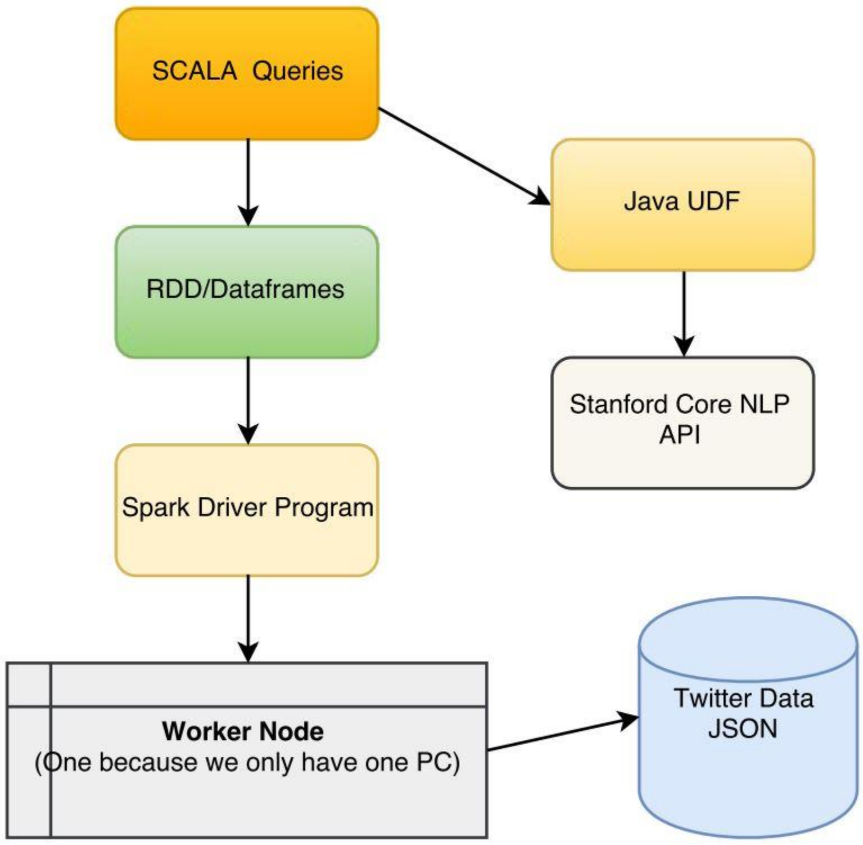

Image 1

Resources Used

- Libraries
  - Spark
  - com.databricks - CSV library for Spark
- API
  - Stanford Core NLP
- Programming Languages
  - Scala - queries
  - Java – UDF
  - Python – processing and cleaning unstructured tweet data
- Testing Machine Specifications
  - Intel® NUC NUC5i3MYHE
  - Intel® Core™ i3-5010U Processor (3M Cache, 2.10 GHz)
  - 16 GB RAM
  - 250 GB SSD
  - Windows 10
  - IntelliJ IDE

Queries

1) Query 1 asks the question: Out of all the tweets collected, which candidate is being discussed the most? How popular are the vice-president nominees versus the presidential nominees? This query performs a number of selects and filters out only tweets that mention Trump, Hillary, and the vice president nominees and then performs a count on them. This query uses a **Dataframe**.

```
    val dataFrame = sqlContext.read.json("COMBINED_Twitter_Debate_Data.json")
    dataFrame.registerTempTable("TweetText")
    //count of Trump themed tweets
    val resultTrump = sqlContext
      .sql("SELECT COUNT(*) as ct FROM TweetText " +
        "WHERE UPPER(text) LIKE '%TRUMP%' " +
        "OR UPPER(text) LIKE '%DONALD TRUMP%' " +
        "OR UPPER(text) LIKE '%TRUMP2016%' " +
        "OR UPPER(text) LIKE '%DONALD%'")

    //count of Pence themed tweets
    val resultPence = sqlContext
      .sql("SELECT COUNT(*) as ct FROM TweetText " +
        "WHERE UPPER(text) LIKE '%MIKE PENCE%' " +
        "OR UPPER(text) LIKE '%PENCE%' ")

    //count of Kaine themed tweets
    val resultKaine = sqlContext
      .sql("SELECT COUNT(*) as ct FROM TweetText " +
        "WHERE UPPER(text) LIKE '%TIM KAINE%' " +
        "OR UPPER(text) LIKE '%KAINE%' ")

    //count of hillary themed tweets
    val resultHillary = sqlContext
      .sql("SELECT COUNT(*) as ct FROM TweetText " +
        "WHERE UPPER(text) LIKE '%HILLARY%' " +
        "OR UPPER(text) LIKE '%HILLARY CLINTON%' " +
        "OR UPPER(text) LIKE '%CLINTON%' " +
        "OR UPPER(text) LIKE '%HILLARY2016%' ")

    //register results of above queries as temp tables
    resultHillary.registerTempTable("hillary")
    resultKaine.registerTempTable("kaine")
    resultTrump.registerTempTable("trump")
    resultPence.registerTempTable("pence")

    //query new temp tables to format for nice output
    val totalMentions = sqlContext
      .sql("SELECT h.ct AS Hillary, k.ct AS Kaine, t.ct AS Trump, p.ct AS Pence " +
        "FROM hillary AS h " +
        "JOIN kaine AS k " +
        "JOIN trump AS t " +
        "JOIN pence AS p ")

    // output total mentions of presidents versus their vice president
    totalMentions.show()
    totalMentions.repartition(1).write.format("com.databricks.spark.csv").option("header",
"true").save("Q1_Total_Mentions.csv")
```

2) Query 2 asks the question: Are many people tweeting a lot about the debate, or are just a few people dominating the discussion with lots of tweets? What is the average tweet per user during the debate? Groups all the tweets by screenname and then orders them by the count of the grouped by names. Also computes the average tweet per user based on the result of previous table. This query uses a **Dataframe**.

```
   //make dataframe with 200,000 tweets - 800MB
    val dataFrame = sqlContext.read.json("COMBINED_Twitter_Debate_Data.json")

    dataFrame.registerTempTable("TweetText")

    //top tweeters for data collected
    val uniqueUsers = sqlContext
      .sql("SELECT user.screen_name as screenName, COUNT(user.screen_name) as total " +
        "FROM TweetText " +
        "GROUP BY user.screen_name " +
        "ORDER BY COUNT(user.screen_name) DESC")

    uniqueUsers.registerTempTable("topUsers")

    //get AVG number of tweets per user
    val avgTweetPerUser = sqlContext
      .sql("SELECT AVG(total) as AverageTweetPerUser FROM topUsers")

    // output top tweeters
    uniqueUsers.show()
    uniqueUsers.repartition(1).write.format("com.databricks.spark.csv").option("header",
"true").save("Q2_Unique_Users.csv")

    // output avg tweet per user
    avgTweetPerUser.show()
    avgTweetPerUser.repartition(1).write.format("com.databricks.spark.csv").option("header",
"true").save("Q2_AVG_Twt_Per_User.csv")
```

3) Query 3 asks the question: What is the overall sentiment of all the tweets about Hillary Clinton versus the overall sentiment of all the tweets about Donald Trump? Performs select to filter all tweets to find those that reference Trump into a table, likewise with Clinton. Performs sentiment analysis on each table with the aid of a UDF. This query uses a **Dataframe**. This query uses an **API**; Stanford Core NLP for sentiment analysis.

```
    val dataFrame = sqlContext.read.json("COMBINED_Twitter_Debate_Data.json")
    dataFrame.registerTempTable("TweetText")

    val resultHillaryTweets = sqlContext
      .sql("SELECT text AS Text FROM TweetText " +
        "WHERE UPPER(text) LIKE '%HILLARY%' " +
        "OR UPPER(text) LIKE '%HILLARY CLINTON%' " +
        "OR UPPER(text) LIKE '%CLINTON%' " +
        "OR UPPER(text) LIKE '%HILLARY2016%' ")

    val resultTrumpTweets = sqlContext
      .sql("SELECT text AS Text FROM TweetText " +
        "WHERE UPPER(text) LIKE '%TRUMP%' " +
        "OR UPPER(text) LIKE '%DONALD TRUMP%' " +
        "OR UPPER(text) LIKE '%TRUMP2016%' " +
        "OR UPPER(text) LIKE '%DONALD%'")

    // Create User Defined Function for processing sentiment
    val coder: (String => Int) = (arg: String) => {

      TweetSentiment.TweetSentimentFinder(arg)}
      val sqlfunc = udf(coder)


    // process new column with sentiment value for Trump
    val trumpSentiment = resultTrumpTweets.withColumn("sentiment", sqlfunc(col("Text")))
    trumpSentiment.registerTempTable("trumpSentiment")

    // process new column with sentiment value for Hillary
    val hillarySentiment = resultHillaryTweets.withColumn("sentiment", sqlfunc(col("Text")))
    hillarySentiment.registerTempTable("hillarySentiment")

    //Count totals
    val totalTrumpRows = resultTrumpTweets.select("Text").count()
    val totalHillaryRows = resultHillaryTweets.select("Text").count()

    val sentimentResultTrump = sqlContext
      .sql("SELECT sentiment, COUNT(sentiment) AS sentIndividualTotal, " +
        "(COUNT(sentiment) * 100.0)/(" + totalTrumpRows * 100.0 + ") AS percentage, " +
        totalTrumpRows + " AS totalCount " +
        "FROM trumpSentiment " +
        "GROUP BY sentiment " )
    sentimentResultTrump.show()

sentimentResultTrump.repartition(1).write.format("com.databricks.spark.csv").option("header"
,"true").save("Q3_Sentiment_Trump.csv")

    val sentimentResultHillary = sqlContext
      .sql("SELECT sentiment, COUNT(sentiment) AS sentIndividualTotal, " +
        "(COUNT(sentiment) * 100.0)/(" + totalHillaryRows * 100.0 + ") AS percentage, " +
        totalHillaryRows + " AS totalCount " +
        "FROM hillarySentiment " +
        "GROUP BY sentiment " )

    sentimentResultHillary.show()

sentimentResultHillary.repartition(1).write.format("com.databricks.spark.csv").option("heade
r","true").save("Q3_Sentiment_Hillary.csv")
```

Q3 (continued) - UDF - API Call to Stanford Core NLP

```
   public static int TweetSentimentFinder(String line) {
        RedwoodConfiguration.empty().capture(System.err).apply();
        Properties properties = new Properties();
        properties.setProperty("annotators", "tokenize, split, parse, sentiment");
        StanfordCoreNLP pipeline = new StanfordCoreNLP(properties);
        int mainSentiment = 0;
        if (line != null && line.length() > 0) {
            int longest = 0;
            Annotation annotation = pipeline.process(line);
            for (CoreMap sentence :
annotation.get(CoreAnnotations.SentencesAnnotation.class)) {
                Tree tree = sentence.get(SentimentCoreAnnotations.AnnotatedTree.class);
                int sentiment = RNNCoreAnnotations.getPredictedClass(tree);
                String partText = sentence.toString();
                if (partText.length() > longest) {
                    mainSentiment = sentiment;
                    longest = partText.length();
                }
            }
        }
        return mainSentiment;
    }
```

4) Query 4 asks the question: What are the most popular hashtags that are in the tweets collected? Uses a python script to grab all the tweets out of the original data set and forms a list similar to that given by professor. Performs a map, then a reduceByKey on our hashtags, then performs a join on the professor provided list of hashtags. Sorts by the hashtags with the highest prevalence descending. This query uses an **RDD**.

```
   // create RDD from hashtags collected from debate tweets
    val tweetTextFile = context.textFile("CLEAN_HASHTAGS_ONLY.txt")
      //do word count on top hash tags
      .map(word => (word, 1))
      .reduceByKey(_ + _)
      //sort the tuples in descending by the value
      .map( item => item.swap)
      .sortByKey(false, 1)
      .map(item => item.swap)


    //load popular hashtags given by professor into RDD, do a basic map to create RDD pair
    val popularHashTags = context.textFile("HashTagTopics.txt").map(word => (word, 0))

    //Perform JOIN on the two RDDs
    val popularDebateHashTags = tweetTextFile
      .join(popularHashTags)
      //re-map to get rid of ther original value for popularHashTags tuple and keep original
word count tuple value
      //for example this would change (#hash, (352, 0))  into -> (#hash, 352)
      .map(item => (item._1, item._2._1))
      //Sort by descending value
      .map(item => item.swap)
      .sortByKey(false, 1)
      .map(item => item.swap)

    println(popularDebateHashTags.first())
    popularDebateHashTags.saveAsTextFile("Q4 _popular_debate_hashtags_output");
```

5) Query 5 asks the question: At what times were the most tweets create during the debate? The text file that is loaded that contains all the timestamps from the “created_at” value from the debate tweets collected. To create this file a python script was created a ran on the original collected data, all python scripts used during phase are in the directory “Data_Helper_Py_Scripts”. Query performs a map reduce on the time stamps for each tweet. This query uses an **RDD**.

```
   val timeStamps = context.textFile("CLEAN_CREATED_AT_ONLY_1.txt")

    //map reduce the timestamps and sort by the the key which is the timestamp (ascending)
    val timeCounts = timeStamps
      .map(word => (word, 1))
      .reduceByKey(_ + _)
      .sortByKey()

    //reduce to one partition so  output file is easy to parse
    // NOTE: if this directory currently exists in output, the program will fail, it cannot
overwrite a directory
    timeCounts.coalesce(1).saveAsTextFile("Q5_Largest_Density_of_Tweets_TimeStamp")
```

Performance Measurements

Query	Data size	Time (unit = seconds)
1	200,000 Tweets ~ 843 MB	48.036160948
2	200,000 Tweets ~ 843 MB	73.914213603
3	200,000 Tweets ~ 843 MB	18371.864213774
4	154,856 Hashtags Joined with 10,652 Hashtags ~ 2 MB	3.658943807
5	200,000 Timestamps ~ 1.5 MB	2.613706821