Spark Streaming

&

Twitter Sentiment Analysis

PROJECT TEAM:

DELALI AGBENYEGAH (Alliance Data Systems)

COLLINS AGYEKUM (Solvency Risk)

THEOPHILUS SIAMEH (Freelancer)

Introduction

Social Media sites like Twitter, Facebook, etc. are like a warehouse of emotions. People tend to share their happiness, sadness and also vent out their frustrations and anger. This collection of people's sentiments in the public domain can be of great value if utilized effectively. The applications of sentiment analysis are broad and powerful. The ability to extract insights from social data is a practice that is being widely adopted by organizations across the world.
Some examples include:

- Shifts in sentiment on social media have been shown to correlate with shifts in the stock market.
- The Obama administration used sentiment analysis to gauge public opinion to policy announcements and campaign messages ahead of the 2012 presidential election.
- The FBI is using sentiment analysis to track how ISIS or other individuals planning to execute a terror attack.

The ability to quickly understand consumer attitudes and react accordingly is something that Expedia Canada took advantage of when they noticed that there was a steady increase in negative feedback to the music used in one of their television adverts.

Sentiment analysis is in demand because of its efficiency. Thousands of text documents can be processed for sentiment (and other features including named entities, topics, themes, etc.) in seconds, compared to the hours it would take a team of people to manually complete. Because it is so efficient, many businesses are adopting text and sentiment analysis and incorporating it into their processes.
However, machines still will never be able to measure sentiment as well as humans, and even humans don't agree 100% of the time. The number of sentiment types is also part of the equation. Some platforms offer three types of sentiments, some offer four types, and some even offer more than five types. The more you increase the number of sentiment types, the less accurate your results become.
And it can be hard to figure out the sentiment from say a sarcastic tweet- which sometimes even humans have a problem demystifying.
Nevertheless, if properly planned and conducted, twitter sentiment analysis can provide valuable

insights that can be utilized to add great value to business decisions and processes.

In this report, we will look at the infrastructure we built for performing sentiment analysis on twitter feeds using Apache **Spark** and show Data Scientists can conduct exploratory analysis on tweeter feeds and build sentiment analysis models.

Problem Statement

To build a data product that obtains, analyzes and classifies sentiments of a stream of tweets given the police shooting in Dallas, Texas.

Methodology

- First, we pull the incident related tweets from Twitter API by applying filter such as words related to the police shootings in Dallas, Texas.
- The tweets are emitted and processed using Spark Streaming Context and written to a JSON file called twitter.json
- The streams of tweets are then processed as 'RDDs of tweets'. Each of the tweet attributes are transformed using the power of Spark SQLContext API to convert the JSON file to a structured table format for further processing.
- We then run SQL queries on the structured table to conduct exploratory analysis to gain some general insights from the tweets
- The tweets are later classified as positive, negative or neutral sentiments using a general collection of positive and negative words.
- Using Machine Learning Pipeline and leveraging its five stages(Tokenizer, StopWordsRemover, HashingTF, Inverse Document Frequency (IDF)), we cleaned and prepared the data for model building
- Finally we build a Naïve Bayes classification model on the tweets sentiments data

Technologies and Software Used

- Apache Spark 1.6.2 http://spark.apache.org/downloads.html
- Twitter API https://apps.twitter.com
- Scala IDE build of Eclipse SDK version 4.4.1 http://scala-ide.org/download/sdk.html
- Scala Programming

Model Architecture

The figure below shows the model architecture

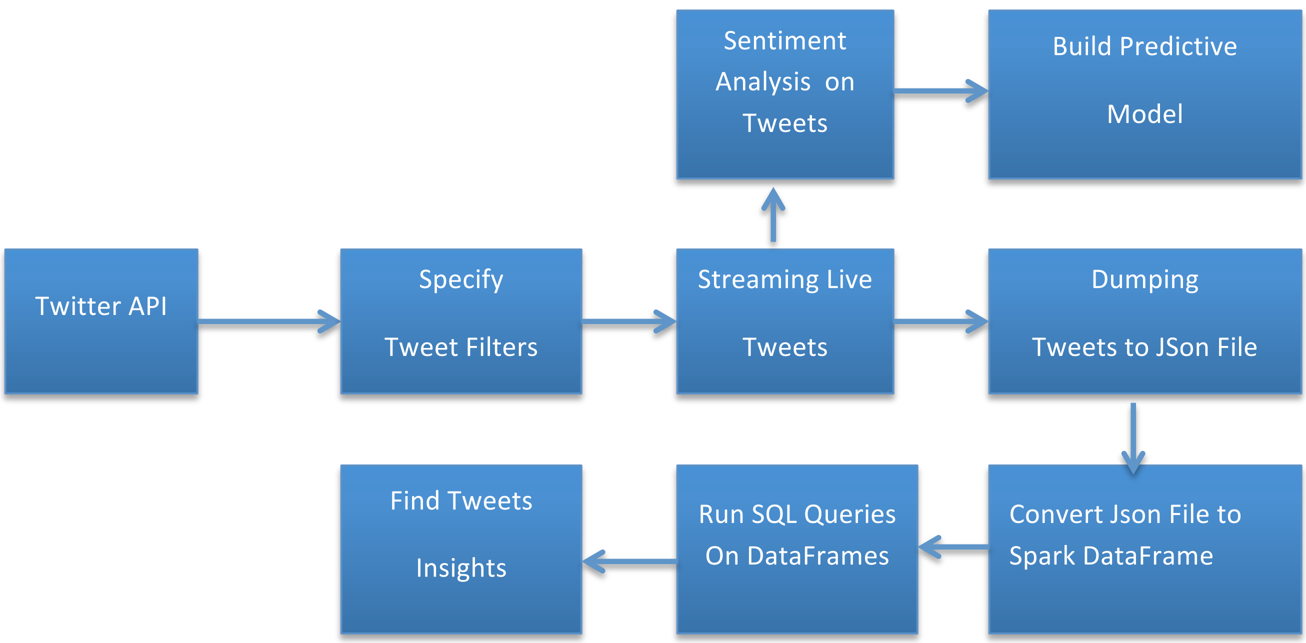

Figure 1: Model Architecture

Machine Learning Pipeline Terminologies

In machine learning, it is common to run a sequence of algorithms to process and learn from data. Spark ML represents such a workflow as a Pipeline, which consists of a sequence of Pipeline Stages (Transformers and Estimators) to be ran in a specific order.

- **Tokenizer:** Splits the raw text documents into words, adding a new column with words into the dataset.

- **StopWordsRemover:** Takes as input a sequence of strings and drops all the stop words from the input sequences. Stop words are words which should be excluded from the input, typically because the words appear frequently and don't carry as much meaning. Optionally you can provide a list of stop words. We utilized the default list within SPARK

- **HashingTF:** Takes sets of terms and converts those sets into fixed-length feature vectors.

- **Inverse Document Frequency (IDF)**: Is a numerical measure of how much information a term provides. If a term appears very often across the corpus, it means it doesn't carry special information about a particular document. IDF down-weights terms, which appear frequently in a corpus.

- **Naive Bayes:** Is a simple multiclass classification algorithm with the assumption of independence between every pair of features. Naive Bayes can be trained very efficiently. Within a single pass to the training data, it computes the conditional probability distribution of each feature given label, and then it applies Bayes' theorem to compute the conditional probability distribution of label given an observation and use it for prediction. It is typically used for document classification. Within that context, each observation is a document and each feature represents a term whose value is the frequency of the term (in multinomial naive Bayes) or a zero or one indicating whether the term was found in the document (in Bernoulli naive Bayes).

Figure 2. below shows the pipeline stages in Spark ML

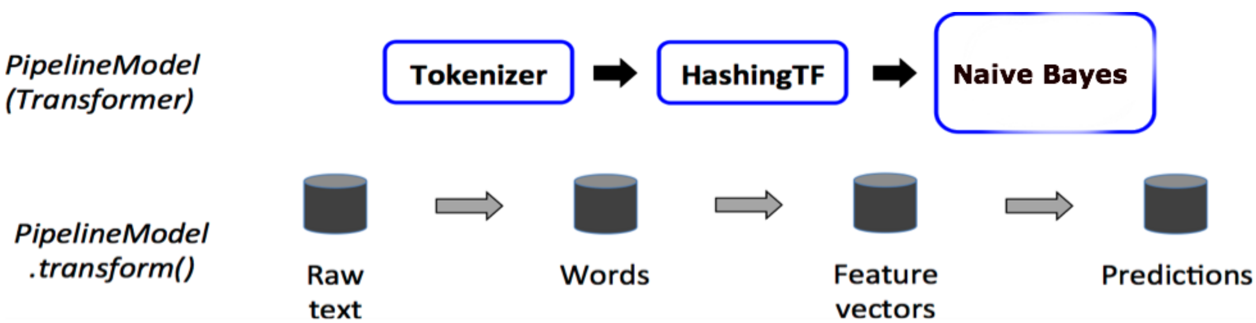

Figure 2: Pipeline Stages in Spark ML

Model Development

The Twitter API enables us to stream the tweets in real time into our Scala-IDE, where we specify the following topics: **"Dallas", "Dallas Police", "Dallas Police Shootings"** in order to filter out any tweets related to the famous Dallas Police Shootings. The idea here is determine the sentiments of twitters about the incident – whether Positive, Negative or Neutral.

These tweets are then dumped into an external JSON file in batches of one second. We then convert the JSON files to a structured format using the power of Spark SQL API in order to run SQL queries to explore any general insights such us number of tweets by location, country, retweets, tweet times, hashtags and many more.

We finally determined whether a live stream of a tweet is positive, negative or neutral and build a predictive model on the sentiments to score how well our model is performing.

In order to build such a predictive model on the tweets, the words in the tweets has to be tokenized to get a list of words, these words are now passed into a Term Frequency algorithm (TF) to get the feature vectors. After getting the feature vectors in a desired format, these vectors are then passed into our machine learning algorithm called Naïve Bayes Classifier.

Spark Performance and Insights

Spark's in memory computation makes tasks run more than a hundred times faster than Hadoop MapReduce.

Spark has ***rich support in Java, Scala, Python*** and growing libraries like ***MLlib*** and ***ML***. Spark can run in Hadoop ecosystem, EC2, Mesos or standalone cluster mode. The primary abstraction in Spark (RDD) is ***fault tolerant*** and can be operated in parallel. Spark streaming processes data in batches, which is a powerful way of doing interactive analysis. As per our project, Spark is processing thousands of live tweets in less than a second and counts their sentiments with seamless fault tolerance computation.

Our model was able to identify the following as a result of the exploratory analysis on the tweets data frame:

- The top retweeted/favorited tweets
- Identify languages with the top tweets
- Identify all verified users and non-verified users. These verified users give insights about influential people talking about the incident.
- Identify top UserFriendsCount
- Identify top UserFollowersCount (people with a lot of followers) on a particular topic/hashtag.

Model Results and Outputs

After streaming the tweets from the Twitter API, we have the following screen shot of the JSON file given below:


```
{"UserID":750155544363667456,"UserDescription":"Social media advocate. Beer nerd. Thinker. Internet specialist. Devoted communic
{"UserID":532458615,"UserDescription":null,"UserScreenName":"celticgoddess77","UserFriendsCount":108,"UserFavouritesCount":786,"
{"UserID":2763418705,"UserDescription":"Someone who KNOWS our COUNTRY'S FUTURE DEPENDS on KEEPING it a CHRISTIAN NATION! Woman w
{"UserID":224243906,"UserDescription":"Transit and transportation reporter for @MetroOttawa. Send me your stories, or your choco
{"UserID":750155544363667456,"UserDescription":"Social media advocate. Beer nerd. Thinker. Internet specialist. Devoted communic
{"UserID":745549282925645824,"UserDescription":"Award-winning student. Beer scholar. Devoted organizer. Certified alcohol maven.
{"UserID":532458615,"UserDescription":null,"UserScreenName":"celticgoddess77","UserFriendsCount":108,"UserFavouritesCount":786,"
{"UserID":47389365,"UserDescription":"English comp & journalism teacher. Ping-Pong, unicycles, accordions. Joie de vivre is my m
{"UserID":190092677,"UserDescription":"DREAM BIG","UserScreenName":"misskyliejean","UserFriendsCount":271,"UserFavouritesCount":
{"UserID":38475547,"UserDescription":"Neutral #MiddleEast news from #Israel, #Egypt, #Lebanon, #Kurdistan, #Iraq, #Iran & beyond
{"UserID":246478084,"UserDescription":"Your news, weather and sports leader for the Fargo-Moorhead region and beyond!","UserScre
{"UserID":246478084,"UserDescription":"Your news, weather and sports leader for the Fargo-Moorhead region and beyond!","UserScre
{"UserID":47389365,"UserDescription":"English comp & journalism teacher. Ping-Pong, unicycles, accordions. Joie de vivre is my m
{"UserID":190092677,"UserDescription":"DREAM BIG","UserScreenName":"misskyliejean","UserFriendsCount":271,"UserFavouritesCount":
{"UserID":38475547,"UserDescription":"Neutral #MiddleEast news from #Israel, #Egypt, #Lebanon, #Kurdistan, #Iraq, #Iran & beyond
{"UserID":831864182,"UserDescription":"Constitutional Conservative | #SupportOurTroops | #HonorOurVets","UserScreenName":"Strong
{"UserID":156886659,"UserDescription":"A Rogue Political Sociologist whose frustration with postmodernity's Parade of the Bizarr
{"UserID":1058648954,"UserDescription":"be true to yourself ϯϲ SC: beemoette","UserScreenName":"beeloyal_","UserFriendsCount":16
{"UserID":453131056,"UserDescription":"I've been called the songbird of my generation.","UserScreenName":"buddha1556","UserFrien
{"UserID":119142901,"UserDescription":null,"UserScreenName":"BremenDaily","UserFriendsCount":1240,"UserFavouritesCount":0,"UserF
{"UserID":156886659,"UserDescription":"A Rogue Political Sociologist whose frustration with postmodernity's Parade of the Bizarr
```


The figure below also shows the schema of the JSON file after converting to Spark Data Frame:

```
root
 |-- HashTags: string (nullable = true)
 |-- StatusCreatedAt: string (nullable = true)
 |-- Text: string (nullable = true)
 |-- TextLength: long (nullable = true)
 |-- UserCreated: string (nullable = true)
 |-- UserDescription: string (nullable = true)
 |-- UserFavouritesCount: long (nullable = true)
 |-- UserFollowersCount: long (nullable = true)
 |-- UserFollowersRatio: double (nullable = true)
 |-- UserFriendsCount: long (nullable = true)
 |-- UserID: long (nullable = true)
 |-- UserLang: string (nullable = true)
 |-- UserLocation: string (nullable = true)
 |-- UserName: string (nullable = true)
 |-- UserScreenName: string (nullable = true)
 |-- UserStatusCount: long (nullable = true)
 |-- UserVerification: boolean (nullable = true)
```

The figure below shows the TweetsTable after converting the JSON file into a structured format (DataFrame) for easy querying with Spark SQL API;

Truncated form …

HashTags	StatusCreatedAt	Text	TextLength	UserCreated	UserDescription	UserFavouritesCount	UserFollowersCount
	2016-07-27T20:20:...	Police officers a...	138	2016-07-04T22:33:...	Social media advo...	0	5
	2016-07-27T20:20:...	RT @TucsonNewsNow...	121	2012-03-21T12:49:...	null	786	39
	2016-07-27T20:20:...	RT @realDonaldTru...	139	2014-08-24T14:04:...	Someone who KNOWS...	1203	761
	2016-07-27T20:20:...	RT @metroottawa: ...	135	2010-12-08T09:26:...	Transit and trans...	527	1292
	2016-07-27T20:20:...	Police officers a...	138	2016-07-04T22:33:...	Social media advo...	0	5
	2016-07-27T20:20:...	Police officers a...	138	2016-06-22T05:29:...	Award-winning stu...	0	2
	2016-07-27T20:20:...	RT @TucsonNewsNow...	121	2012-03-21T12:49:...	null	786	39
	2016-07-27T20:20:...	Police Use Finger...	93	2009-06-15T13:28:...	English comp & jo...	2653	1402
	2016-07-27T20:20:...	i ate all my ice ...	95	2010-09-12T22:01:...	DREAM BIG	1533	373
	2016-07-27T20:20:...	Youth pulls handg...	86	2009-05-07T13:34:...	Neutral #MiddleEa...	91	45346
	2016-07-27T20:20:...	VIDEO: Police rel...	115	2011-02-02T16:26:...	Your news, weathe...	1124	10902
	2016-07-27T20:20:...	VIDEO: Police rel...	115	2011-02-02T16:26:...	Your news, weathe...	1124	10902
	2016-07-27T20:20:...	Police Use Finger...	93	2009-06-15T13:28:...	English comp & jo...	2653	1402
	2016-07-27T20:20:...	i ate all my ice ...	95	2010-09-12T22:01:...	DREAM BIG	1533	373
	2016-07-27T20:20:...	Youth pulls handg...	86	2009-05-07T13:34:...	Neutral #MiddleEa...	91	45346
	2016-07-27T20:20:...	RT @V_of_Europe: ...	140	2012-09-18T16:53:...	Constitutional Co...	532	5069
#ImWithHer…	2016-07-27T20:20:...	RT @ChrisJZullo: ...	140	2010-06-18T02:05:...	A Rogue Political...	1928	65
	2016-07-27T20:20:...	RT @janae_angeliq...	133	2013-01-03T14:54:...	be true to yourse...	3341	1381
	2016-07-27T20:20:...	RT @hale_razor: W...	140	2012-01-02T11:24:...	I've been called ...	749	54
	2016-07-27T20:20:...	Police evacuate B...	69	2010-03-02T15:04:...	null	0	2070

only showing top 20 rows

truncation continue …

UserFollowersRatio	UserFriendsCount	UserID	UserLang	UserLocation	UserName	UserScreenName	UserStatusCount	UserVerification
0.1111111119389534	45	750155544363667456	en	Kansas City, MO	Megan Mueller	isaiyayakovlev2	6326	false
0.3611111044883728	108	532458615	en	null	Kelli Hartley	celticgoddess77	293	false
0.48533162474632263	1568	2763418705	en	null	Savannah Lee	LeeSavannahlee1	2035	false
1.4884792566299438	868	224243906	en	Ottawa, Canada	Emma Jackson	EmmaEJackson	3981	false
0.1111111119389534	45	750155544363667456	en	Kansas City, MO	Megan Mueller	isaiyayakovlev2	6326	false
0.20000000298023224	10	745549282925645824	en	null	robinmaidner	robinmaidnerob1	19141	false
0.3611111044883728	108	532458615	en	null	Kelli Hartley	celticgoddess77	293	false
0.7453482151031494	1881	47389365	en	Atlantic IA	Allison Berryhill	allisonberryhil	12736	false
1.3763837814331055	271	190092677	en	Las Vegas, NV	kylie mogard	misskyliejean	29431	false
18.38102912902832	2467	38475547	en	null	Middle East News	MidEastNews	183364	false
44.13765335083008	247	246478084	en	Fargo, ND	WDAY TV News	WDAYnews	59740	false
44.13765335083008	247	246478084	en	Fargo, ND	WDAY TV News	WDAYnews	59740	false
0.7453482151031494	1881	47389365	en	Atlantic IA	Allison Berryhill	allisonberryhil	12736	false
1.3763837814331055	271	190092677	en	Las Vegas, NV	kylie mogard	misskyliejean	29431	false
18.38102912902832	2467	38475547	en	null	Middle East News	MidEastNews	183364	false
1.3176500797271729	3847	831864182	en	null	Stronger America	StrongerAmerca	27510	false
0.5327869057655334	122	156886659	en	Knoxville, TN	Jonathan Vedamuthu	jjvedamuthu	4251	false
0.840535581111908	1643	1058648954	en	Loreauville, LA	brianca.	beeloyal_	52430	false
0.5346534848213196	101	453131056	en	null	Diarmaid O'Hanvirra	buddha1556	994	false
1.669354796409607	1240	119142901	en	Bremen, Germany	Bremen Daily	BremenDaily	17017	false

The figure below shows the query results using:

```
//get tweets lang, name and text
sqlContext.sql("SELECT UserLang, UserName, Text FROM TweetTable WHERE UserLang='en' LIMIT 10").show()
```

UserLang	UserName	Text
en	Megan Mueller	Police officers a...
en	Kelli Hartley	RT @TucsonNewsNow...
en	Savannah Lee	RT @realDonaldTru...
en	Emma Jackson	RT @metroottawa: ...
en	Megan Mueller	Police officers a...
en	robinmaidner	Police officers a...
en	Kelli Hartley	RT @TucsonNewsNow...
en	Allison Berryhill	Police Use Finger...
en	kylie mogard	i ate all my ice ...
en	Middle East News	Youth pulls handg...

The figure below shows the query results using:

```
//count languages with the highest tweets
sqlContext.sql("SELECT UserLang, COUNT(*) AS Counts FROM TweetTable " +
               "GROUP BY UserLang ORDER BY Counts DESC LIMIT 1000").show()
```

```
+--------+------+
|UserLang|Counts|
+--------+------+
|      en|   709|
|      es|    16|
|      nl|    10|
|      pt|     8|
|      fr|     6|
|      ja|     6|
|      it|     2|
|      de|     2|
|      ko|     2|
|      ru|     2|
+--------+------+
```

It appears from the table that English tweets was the highest followed by Spanish and so forth.

The figure below shows the query results using:

```
//get highest location counts
sqlContext.sql("SELECT UserLocation, COUNT(*) AS LoCount FROM TweetTable "+
            "GROUP BY UserLocation ORDER BY LoCount DESC LIMIT 10").show()
```

As we can see from the table below, unknown location had the highest tweets followed by Dallas with about 26 tweets and so forth.

```
+--------------------+-------+
|        UserLocation|LoCount|
+--------------------+-------+
|                null|    264|
|              Dallas|     26|
|              Global|     10|
|       Baltimore, MD|      8|
|         Houston, TX|      8|
|       United States|      8|
|                 USA|      7|
|               Texas|      4|
|     California, USA|      4|
|Republic of the P...|      4|
+--------------------+-------+
```

The figure below shows the query results using:

```
// get UserFriends Count
sqlContext.sql("SELECT UserName,UserFriendsCount,UserLang FROM TweetTable "+
              "GROUP BY UserFriendsCount,UserName,UserLang ORDER BY UserFriendsCount DESC  LIMIT 10").show()
```

We can see from the figure below that, the UserName "The Really Cars " had the highest UserFriendsCount of **108158** located somewhere in Netherland, followed by the UserName "Collecting Drugs" with UserFriendsCount of about **65387** also from Netherlands and the third with UserName of "Gucci" with about **41356** UserFriendsCount and speaks English.

```
+----------------+----------------+--------+
|        UserName|UserFriendsCount|UserLang|
+----------------+----------------+--------+
| The Really Cars|          108158|      nl|
|Collecting Drugs|           65387|      nl|
|           Gucci|           41356|      en|
|    Kylie Jenner|           38600|      en|
|  Music and More|           31976|      en|
|   Isaac Brinker|           27890|      en|
|    Natural Love|           27778|      en|
|      ShaaySquad|           26903|      en|
|        EyeCandy|           25627|      en|
|      Brittany ❤|           24042|      en|
+----------------+----------------+--------+
```

The figure below shows the query results using:

```
// get all verified users UserVerification
sqlContext.sql("SELECT DISTINCT(UserName),UserVerification FROM TweetTable ").show()
```

The results shows that most of the people who tweeted were unverified users as compared to those who were verified by Twitter, Inc. As indicated in the table Sara Small is a verified user.

```
+--------------------+----------------+
|            UserName|UserVerification|
+--------------------+----------------+
|              Chanel|           false|
|           wenpezasw|           false|
|        TeamTripleD®|           false|
|               Katie|           false|
|                 dyl|           false|
|       Jessica Brook|           false|
|        Bremen Daily|           false|
|           Gooman436|           false|
|        gfx designer|           false|
|       Debi Muhammad|           false|
|       Hood Pope ⁶övö|           false|
|          Sara Small|            true|
|       Melissa Plank|           false|
|   joyce porter-dunn|           false|
|       carolfairiarc|           false|
|           Icey Life|           false|
|                 Leo|           false|
|     Jean M. O'Brien|           false|
|         Viral Today|           false|
|danielmingsedanielmi|           false|
+--------------------+----------------+
only showing top 20 rows
```

The figure below shows the query results using:

```
// get top UserFollowersCount
sqlContext.sql("SELECT DISTINCT(UserName),UserFollowersCount FROM TweetTable " +
            "ORDER BY UserFollowersCount DESC LIMIT 10 ").show()
```

```
+--------------------+------------------+
|            UserName|UserFollowersCount|
+--------------------+------------------+
|KHOU 11 News Houston|            331773|
|   The Baltimore Sun|            188827|
|        Las Vegas RJ|            139407|
|  Nine News Brisbane|            101999|
|     The Really Cars|            101760|
|                WBEZ|             98314|
|               Gucci|             69031|
|    Nine News Sydney|             67459|
|       Isaac Brinker|             66280|
|    Collecting Drugs|             58942|
+--------------------+------------------+
```

It appears from the table above that, most of the news agencies reported the Dallas police shootings so most people wanted to know that they reported so they had to follow them.

The figure below shows the query results using:

```
 // get top UserFavouritesCount
sqlContext.sql("SELECT DISTINCT(UserName),UserFavouritesCount FROM TweetTable  " +
                "ORDER BY UserFavouritesCount DESC LIMIT 10").show()
```

```
+--------------------+-------------------+
|            UserName|UserFavouritesCount|
+--------------------+-------------------+
|           Truthglow|             253471|
|        Janis Sexton|             156438|
|     Jean M. O'Brien|             125961|
|मौलाना बरखा हाफिज दत|             105485|
|     TrumpkinsBakery|              99933|
|               Bruce|              90004|
|     Bothic Brawford|              61274|
|NonCompliant lobster|              61044|
|              Simone|              60542|
|        Gary Collins|              58697|
+--------------------+-------------------+
```

It appears that **Truthglow, Janis Sexton; Jean M. O'Brian** made some comments about the Dallas police shooting incident so a lot of people made them their favorites.

Sentiment Analysis of Tweets

Here incoming stream of tweets are classified as either positive, negative or neutral based on a corpus of words in our text files: **pos-words.txt, neg-words.txt and stop-words.txt**

Positive words are label as 1, Negative words as 0 and Neutral words as 2. But for simplicity of this project we considered Naïve Bayes with a multi-class response (Positive-1, Negative-0 and Neutral-2).

Naïve Bayes Outputs

```
-------------------------------------------
Time: 1470349025000 ms
-------------------------------------------
(RT Full 5yo eyewitness account of Baltimore County Police shooting that killed his,0.0)
(RT It is hard to but the gist is did not FIRE at police and nor did she use him as a,0.0)
(RT The police say a social media app is possibly responsible for her death and not the actual police who killed,0.0)
(RT Trans folks worse attacked by the Uganda fondled abused in raid of a Pride Beauty Contest,2.0)
(RT is proof that when it comes to black women girls that are victims of police brutality always,0.0)
(From Major Arrest Schenectady Police report dollars in cash along,2.0)

+--------------------+-----+
|                text|label|
+--------------------+-----+
|RT Full 5yo eyewi...|  0.0|
|RT It is hard to ...|  0.0|
|RT The police say...|  0.0|
|RT Trans folks wo...|  2.0|
|RT is proof that ...|  0.0|
|From Major Arrest...|  2.0|
+--------------------+-----+

Total Document Count = 6
Training Count = 4, 66.66666666666667%
Test Count = 2, 33.333333333333336%
+--------------------+--------------------+-----+----------+
|                text|         probability|label|prediction|
+--------------------+--------------------+-----+----------+
|RT It is hard to ...|[0.51893191803476...|  0.0|       0.0|
|RT The police say...|[0.67035507688230...|  0.0|       0.0|
+--------------------+--------------------+-----+----------+

Accuracy: 1.0
-------------------------------------------
```

```
-------------------------------------------
Time: 1470349026000 ms
-------------------------------------------
(RT The police say a social media app is possibly responsible for her death and not the actual police who killed,0.0)
(City police raid Chinese tour company,2.0)
(RT Mothers of Plaza de Mayo harassed by the Police under the Macri,0.0)
(RT By calling this a health the police have volunteered their last scrap of Taking the mick,0.0)
(RT Full 5yo eyewitness account of Baltimore County Police shooting that killed his,0.0)

+--------------------+-----+
|                text|label|
+--------------------+-----+
|RT The police say...|  0.0|
|City police raid ...|  2.0|
|RT Mothers of Pla...|  0.0|
|RT By calling thi...|  0.0|
|RT Full 5yo eyewi...|  0.0|
+--------------------+-----+

Total Document Count = 5
Training Count = 1, 20.0%
Test Count = 4, 80.0%
+--------------------+-----------+-----+----------+
|                text|probability|label|prediction|
+--------------------+-----------+-----+----------+
|City police raid ...|      [1.0]|  2.0|       0.0|
|RT By calling thi...|      [1.0]|  0.0|       0.0|
|RT Full 5yo eyewi...|      [1.0]|  0.0|       0.0|
|RT Mothers of Pla...|      [1.0]|  0.0|       0.0|
+--------------------+-----------+-----+----------+

Accuracy: 0.6428571428571428
-------------------------------------------
```

```
-------------------------------------------
Time: 1470349032000 ms
-------------------------------------------
(media and police the attack after We can now undead that poor,0.0)
(RT The real gag is social workers have their lives just as much as the police and never kill anyone,0.0)
(RT Police shoots and kills homeless man in Los,0.0)

+--------------------+-----+
|                text|label|
+--------------------+-----+
|media and police ...|  0.0|
|RT The real gag i...|  0.0|
|RT Police shoots ...|  0.0|
+--------------------+-----+

Total Document Count = 3
Training Count = 2, 66.66666666666667%
Test Count = 1, 33.333333333333336%
+--------------------+-----------+-----+----------+
|                text|probability|label|prediction|
+--------------------+-----------+-----+----------+
|RT Police shoots ...|      [1.0]|  0.0|       0.0|
+--------------------+-----------+-----+----------+

Accuracy: 1.0
-------------------------------------------
```

Future work on tweeter analysis within spark

Given our time constraint, not all special features for tweeter analysis and predictive modeling within SPARK were utilized.

Below are some other interesting dimensions we will continue to add to this project in the future

- Geo-spatial maps could be plotted, signifying the location (and sentiment) of the tweets.
- Creating dashboards to monitor all incoming tweets, user interaction and present results in a visually appealing manner.
- Creating a pipeline to dump all processed tweets into HDFS, Databases like Cassandra, Mongo DB, PostgreSQL or MySQL.
- This project can be incorporated into **marketing campaigns** in order to make decisions.

Lessons Learned and Challenges Faced

- In Spark Streaming, one has to be really careful while using data taken from the streaming context. Operations performed on the RDDs should be done using *transform* and *forEachRDD* functions.

- In Spark, transforming the DStreams into Data Frames can be quite a pain and one has to be cautious in order to maintain the schema properly.

- While streaming into your local disk, one has to limit the number of tweets that are coming in to prevent disk from getting fully loaded with streams of tweets.

- One has to also make sure all spam related tweets are filtered out to avoid writing meaningless data into database files or text files.

- There is no spark API currently available for graphical visualization of results but there are other external tools like **plotly** than could be used.

Conclusion

Spark provides a friendly and efficient platform cluster computing. Leveraging this platform, we successfully build scala and spark application infrastructure that analyzes the sentiments of tweets streamed about the Dallas Police Shootings, except that we did not include any form of visualization in our reports since we restricted our domain on spark and scala. Most of the tweets that came in at each run of the application, were mostly negative tweets as compared to the positive and neutral tweets.

Sample Scala Code

```scala
 */
object TwitterTransmitter {

  //private var gson = new Gson()


  //only words function
  def onlyWords(text: String) : String = {
      text.split(" ").filter(_.matches("^[a-zA-Z0-9 ]+$")).fold("")((a,b) => a + " " + b).trim
  }


  def main(args: Array[String]) {

    // Define Logging Level
   setStreamingLogLevels()

    // Accept Twitter Stream filters from arguments passed while running the job
    //val filters = if (args.length == 0) List() else args.toList
    // create filter Strings
    val filters : Array[String] = Array("Dallas Shooter","Dallas Killing","Dallas Shootings","Dallas Police")

   // Print the filters being used
   // println("Filters being used: " + filters.mkString(","))

    // can use them to generate OAuth credentials
    System.setProperty("twitter4j.oauth.consumerKey", "kitWON6h8pqfC7z2Kai03U72y")
    System.setProperty("twitter4j.oauth.consumerSecret", "FhdO5UAdTFrPyKvHkGl8wLqxVmhrfFhojljNWegf9QKmmQ8p0l")
    System.setProperty("twitter4j.oauth.accessToken", "338883393-xxcZ1Jy3h6vm3gPCTWXNl50095cqJq4OCI4t6K2B")
    System.setProperty("twitter4j.oauth.accessTokenSecret", "gkQrToUEGfv3QmCypqGeYvXFfDEzYI89CNqIJGShwklpR")

    // Initialize a SparkConf with all available cores
    val sparkConf = new SparkConf().setAppName("Streaming").setMaster("local[*]")
```

```scala
    // Create a StreamingContext with a batch interval of 1 seconds.
    val ssc = new StreamingContext(sparkConf, Seconds(1))

   // Create a DStream the gets streaming data from Twitter with the filters provided
    val stream = TwitterUtils.createStream(ssc, None, filters,StorageLevels.MEMORY_AND_DISK)

    // Process each tweet in a batch
    val tweetMap = stream.map(status => {

      // Defined a DateFormat to convert date Time provided by Twitter to a format understandable

    val formatter = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss.SSSZZ")

    def checkObj (x : Any) : String  = {
        x match {
          case i:String => i
          case _ => ""
        }
      }


      // Creating a JSON using json4s.JSONDSL with fields from the tweet and calculated fields
        val tweetMap =
        ("UserID" -> status.getUser.getId) ~
          ("UserDescription" -> status.getUser.getDescription) ~
          ("UserScreenName" -> status.getUser.getScreenName) ~
          ("UserFriendsCount" -> status.getUser.getFriendsCount) ~
          ("UserFavouritesCount" -> status.getUser.getFavouritesCount) ~
          ("UserFollowersCount" -> status.getUser.getFollowersCount) ~
          // Ratio is calculated as the number of followers divided by number of people followed
          ("UserFollowersRatio" -> status.getUser.getFollowersCount.toFloat / status.getUser.getFriendsCount.to
          ("GeoLatitude" -> status.getGeoLocation.getLatitude.toDouble) ~
          ("GeoLongitude" -> status.getGeoLocation.getLongitude.toDouble) ~
```

```
            //("PlaceCountry" -> (status.getPlace.getCountry.toString()))~
            //Tokenized the tweet message and then filtered only words starting with #
            ("HashTags" -> status.getText.split(" ").filter(_.startsWith("#")).mkString(" ")) ~
            ("StatusCreatedAt" -> formatter.format(status.getCreatedAt.getTime)) // ~


        // This function takes Map of tweet data and returns true if the message is not a spam
        def spamDetector(tweet: Map[String, Any]): Boolean = {
          {
            // Remove recently created users = Remove Twitter users who's profile was created less than a day ago
            Days.daysBetween(new DateTime(formatter.parse(tweet.get("UserCreated").mkString).getTime),
              DateTime.now).getDays > 1
          } & {
            // Users That Create Little Content =  Remove users who have only ever created less than 50 tweets
            tweet.get("UserStatusCount").mkString.toInt > 50
          } & {
            // Remove Users With Few Followers
            tweet.get("UserFollowersRatio").mkString.toFloat > 0.01
          } & {
            // Remove Users With Short Descriptions
            tweet.get("UserDescription").mkString.length > 20
          } & {
            // Remove messages with a Large Numbers Of HashTags
            tweet.get("Text").mkString.split(" ").filter(_.startsWith("#")).length < 5
          } & {
            // Remove Messages with Short Content Length
            tweet.get("TextLength").mkString.toInt > 20
          } & {
            // Remove Messages Requesting Retweets & Follows
            val filters = List("rt and follow", "rt & follow", "rt+follow", "follow and rt", "follow & rt", "foll
            !filters.exists(tweet.get("Text").mkString.toLowerCase.contains)
          }
        }
```

NAÏVE BAYES CODE

```
/** Simple application to listen to a stream of Tweets and print them out sentiments */
object NaiveBayes {

  // Read corpus of words from text files
  val posWords = Source.fromFile("/Users/theophilus/Desktop/TwitterStream/TwitterStream/src/com/spark/streaming
  val negWords = Source.fromFile("/Users/theophilus/Desktop/TwitterStream/TwitterStream/src/com/spark/streaming
  val stopWords = Source.fromFile("/Users/theophilus/Desktop/TwitterStream/TwitterStream/src/com/spark/streamin

  val posWordsArr = mutable.MutableList("")
  val negWordsArr = mutable.MutableList("")

  for(posword <-posWords){
    posWordsArr +=(posword)
  }
  for(negWord <-negWords){
    negWordsArr +=(negWord)
  }

  // custom function to find whether the tweet is
  // positive or negative or neutral
  def findTweetSentiment(tweet:String):String = {
    var count = 0
    for(w <- tweet.split(" ")){
      //positive words
      for(positiveW <- posWordsArr){
        if(w != " " && positiveW.toString.toLowerCase()==w){
          count = count+1
        }
      }
      //negative words
      for(negativeW <- negWordsArr){
        if(w != " " && negativeW.toString.toLowerCase()==w){
```

```scala
val sqlContext = new SQLContext(sc)
import sqlContext.implicits._

import org.apache.spark.ml.feature.{HashingTF, StopWordsRemover, IDF, Tokenizer}
import org.apache.spark.ml.PipelineStage
import org.apache.spark.sql.Row
import org.apache.spark.ml.classification.LogisticRegression
import org.apache.spark.ml.{Pipeline, PipelineModel}
import org.apache.spark.ml.evaluation.BinaryClassificationEvaluator
import org.apache.spark.ml.tuning.{ParamGridBuilder, CrossValidator}
import org.apache.spark.mllib.linalg.Vector


data.foreachRDD { rdd =>

 // convert RDD into DataFrame
 val df = rdd.toDF("text","label")

 // register as tempTable
 df.registerTempTable("STable")

 // run some queries
 sqlContext.sql("SELECT text,label FROM STable").show()

 // Split data into training (60%) and test (40%).
 val splits = df.randomSplit(Array(0.6, 0.4), seed = 12345L)

 val training = splits(0).cache()
 val testing = splits(1)

```

```scala
 import org.apache.spark.ml.classification.NaiveBayes
 import org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator

 println("Total Document Count = " + df.count())
 println("Training Count = " + training.count() + ", " + training.count*100/(df.count()).toDouble + "%")
 println("Test Count = " + testing.count() + ", " + testing.count*100/(df.count().toDouble) + "%")

 // Configure an ML pipeline, which consists of five stages: tokenizer, remover, hashingTF,IDF and lr.
 val tokenizer = new Tokenizer().setInputCol("text").setOutputCol("words")
 val remover = new StopWordsRemover().setInputCol("words").setOutputCol("filtered").setCaseSensitive(false)
 val hashingTF = new HashingTF().setNumFeatures(20000).setInputCol("filtered").setOutputCol("rawFeatures")
 val idf = new IDF().setInputCol("rawFeatures").setOutputCol("features").setMinDocFreq(0)
 val nb = new NaiveBayes()

 val pipeline = new Pipeline().setStages(Array(tokenizer, remover, hashingTF, idf, nb))

 // Fit the pipeline to training documents.
 val model = pipeline.fit(training)

 // Select example rows to display.
 val predictions = model.transform(testing)
 //predictions.show()

 predictions.select("text", "probability", "label","prediction").show(5)

 // Select (prediction, true label) and compute test error
 val evaluator = new MulticlassClassificationEvaluator()
                   .setLabelCol("label")
                   .setPredictionCol("prediction")

```

```scala

    // evaluate for accuracy
    val accuracy = evaluator.evaluate(predictions)
    println("Accuracy: " + accuracy)
    println(evaluator.isLargerBetter)


    // Tune hyperparameters
    val paramGrid = new ParamGridBuilder().addGrid(hashingTF.numFeatures, Array(1000, 10000, 100000))
                    .addGrid(idf.minDocFreq, Array(0,10, 100))
                    .build()


    // Cross validation
    val cv = new CrossValidator().setEstimator(pipeline)
                 .setEvaluator(evaluator)
                 .setEstimatorParamMaps(paramGrid).setNumFolds(2)

   // cross-evaluate
    val cvModel = cv.fit(training)
    println("Area under the ROC curve for best fitted model = " + evaluator.evaluate(cvModel.transform(testing)

    println("Area under the ROC curve for non-tuned model = " + evaluator.evaluate(predictions))
    println("Area under the ROC curve for best fitted model = " + evaluator.evaluate(cvModel.transform(testing)
    println("Improvement = " + "%.2f".format((evaluator.evaluate(cvModel.transform(testing)) - evaluator.evalua


   }
```

SPARK DATA FRAME CODE

```scala
  def main(args: Array[String]) {

    //logger
    setupLogging()

    // Set up a SparkContext named WordCount that runs locally using
    // all available cores.
    val conf = new SparkConf().setMaster("local[*]").setAppName("TweetSQL")
    //initialized sparkcontext
    val sc = new SparkContext(conf)
    sc.setLogLevel("WARN")

    // sc is an existing SparkContext.
    val sqlContext = new SQLContext(sc)
    // this is used to implicitly convert an RDD to a DataFrame.
    import sqlContext.implicits._

    // Create an RDD of Person objects and register it as a table.
    val tweets = sqlContext.read.json("/Users/theophilus/Desktop/TwitterStream/TwitterStream/src/com/spark/streaming/twitter.json")

    //print schema
    tweets.printSchema()

    //show table
    tweets.show()

    //register temp table
    tweets.registerTempTable("TweetTable")

    // SQL statements can be run by using the sql methods provided by sqlContext.
    sqlContext.sql("SELECT Text FROM TweetTable WHERE UserLang='en' LIMIT 10").show()

    //get tweets lang, name and text
    sqlContext.sql("SELECT UserLang, UserName, Text FROM TweetTable WHERE UserLang='en' LIMIT 10").show()

     //count languages with the highest tweets
    sqlContext.sql("SELECT UserLang, COUNT(*) AS Counts FROM TweetTable " +
                    "GROUP BY UserLang ORDER BY Counts DESC LIMIT 1000").show()

```

References

- Learning Spark –[Lightening-Fast Data Analysis]
  By Holden Karau, Andy Konwinski, Patrick Wendell & Matei Zaharia
- Advanced Analytics with Spark – [Patterns for learning from data at scale]
  By Sandy Ryza, Uri Laserson, Sean Owen & Josh Wills.
- Learning Scala –[Practical Functional Programming for the JVM]
  By Jason Swartz
- Apache Spark Official Page: http://spark.apache.org