

Project Title:

Tweets Analysis with Spark

Team:

Code Monkeys

Team members:

Harshil Patel (8), Matthew Boerner (1), Stephanie Retzke (10) and Yong Zheng (14)

Final Report - Yong

Introduction

Twitter has a lot interesting data and tweets that are easy to collect. For increment 1, I applied some SQL queries against the data we collected. For increment 2, I created machine learning model on the Twitter data I collected. Also I applied graph algorithms to these data.

Background

Each member of the team is asked to collect 100K English tweets with various tracks. The analysis for each member is done on their own set of tweets (100K). Due to the size of the dataset and our previous conversation, I will not upload this dataset to GitHub.

For increment 1:

I am using tweepy package in Python to collect data from Twitter. I wrote a python script to collect 100K English tweets with various tracks. With data I collected, I then used Spark and SparkSQL to perform 10 interesting queries on it.

Here are my queries:

Query 1:

Find polarity percentage for tweets' text field

Query 2:

Find the created year distribution for users

Query 3:

Find most popular login sources

Query 4:

Generate word cloud for hashtags' popularity

Query 5:

Count null and not null for geo tag

Query 6:

Get country code distribution from geo tag

Query 7:

Get top 10 country codes distribution from place tag

Query 8:

Message truncated percentage

Query 9:

User verified distribution and analysis

Query 10:

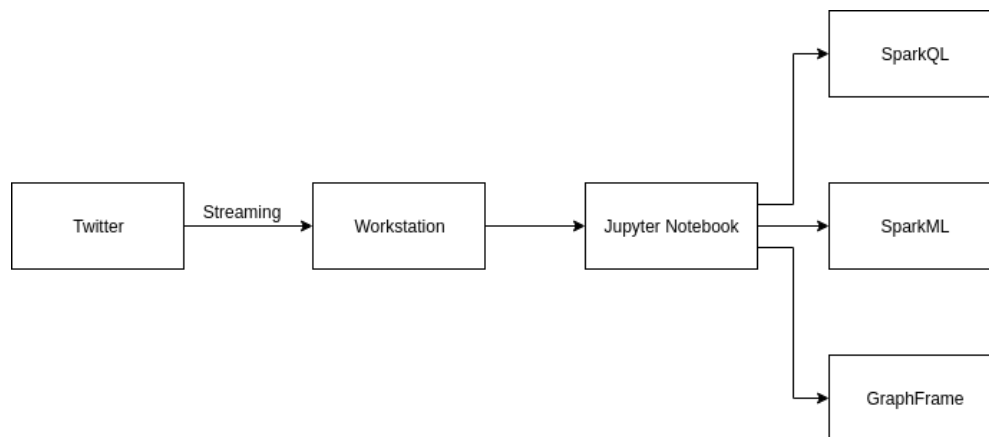
Analysis on friends_count and followers_count tags

For increment 2:

Each member of the team is asked to build a machine learning model based on the data we collected from increment 1. In order to do this, I want to know what is the most common settings people used among the a list of settings I am interested from the 100K tweets. After I selected a list of target settings (features) from the tweets, I then use frequent pattern mining to determine which combination of target settings are most common among the users in the dataset I collected. In order to do this, I will use FP-growth algorithm in Spark to perform frequent pattern mining against a list of selected features. Other than building machine learning model, I also apply graph algorithms to the datasets I collected. For the algorithms, I try to find the in-degree of vertices, out-degree of vertices, performance shortest paths with landmarks, and count the number of triangle in the graph.

Model

Architecture and workflow diagram:



I am using tweepy package in Python to stream 100k English tweets and persist to my workstation. By doing so, I can then apply data analytics with PySpark against this dataset. In order to make the code more readable and easy to understand, I decide to use Jupyter notebook as the main editor for this project (I linked Jupyter notebook as the driver for spark). By doing so, I can code spark in Jupyter notebook.

For increment 1:

I wrote 10 interesting questions with SparkQL in Jupyter notebook with the data I collected from Twitter.

For increment 2:

I build one machine learning model in the Jupyter notebook with the same dataset but with selected features. Also, I applied some graph algorithms with GraphFrame against the dataset I collected.

Dataset

For increment 1:

Each member of the team is asked to collect 100K English tweets with various tracks. The analysis for each member is done on their own set of tweets (100K). Due to the size of the dataset and our previous conversation, I will not upload this dataset to GitHub.

For increment 2:

In order to find some interesting data with a machine learning model, I decided to use features “truncated”, “user.verified”, “user.geo_enabled”, “user.profile_background_tile”, “user.profile_use_background_image”, and “user.default_profile” as the inputs for machine learning model. These features are parsed from the 100K tweets I collected from increment 1. Here is the description for each feature I selected:

- truncated: indicates whether the value of the tweet message was truncated due to 140 characters tweet limitation on Twitter
- user.verified: indicates whether a user has a verified account
- user.geo_enabled: indicates whether a user has enabled the possibility of geotagging their Tweets
- user.profile_background_tile: indicates whether a user’s profile_background_image_url should be tiled when displayed
- user.profile_use_background_image: indicates whether a user wants their uploaded background image to be used
- user.default_profile: indicates whether a user has not altered the theme or background of their user profile

For GraphFrame, I am using “user.id” and “retweeted_status.user.id” fields. See detail explanation in the “analysis of data” section.

Analysis of data

For increment 1:

Query 1:

I want to use sentential analysis on the tweet to determine the polarity of each tweet then count how many tweets are positive, negative and neutral.

Query 2:

I want to see the created year distribution for the distinct users in my data set.

Query 3:

I want to what is the most popular login sources people used for Twitter

Query 4:

I want to generate a word cloud based on hashtags' popularity

Query 5:

I want to check the count for users with and without geo info (from geo tag)

Query 6:

I want to check the country code distribution for users with not null value for geo tag. In order to get actual country code from geo code, I will need to reverse the latitude and longitude into actual address and parsed out the country code.

Query 7:

I want to check the top 10 country codes distribution from place tag. Unlike geo tag, place tag will also address info already. However, place tag will not give us the extra location of a user. It only provides the coordinates info for the approximated location.

Query 8:

I want to check how many tweets are truncated.

Query 9:

I want to get the distribution of uses whose accounts are verified. Among the verified/unverified users, I want to check how many of them are and are not using default_profile.

Query 10:

I want to know how many users have more friends count than followers count, how many users have followers count then friends count, and how many users have same number for both friends count and followers count.

For increment 2:

In order to use the data I collected, I had to perform some data pre-processing and post-processing for various of tasks. For example, for data pre-processing, I need to use multiple Spark build in functions to get the min, max, median, standard deviation for multiple fields and understand how the data are spread out among the data I collected. For the columns with multiple values, I will need to use array functions in Spark in order to access the member of the

array. Also, I generated computed columns based on the values of the certain fields in order to generate new fields for my machine learning model. I used the knowledge I learned from multiple classes that are Spark related for data pre-processing and post-processing task. For graph model I created, For GraphFrame, I am using “user.id” and “retweeted_status.user.id” as the source and destination in the graph (edges) and use “user.id” and “user.screen_name” to union “retweeted_status.user.id” and “retweeted_status.user.name” for the actual vertices and vertices’ metadata. With the vertices and edges I created here, I then create a graph with GraphFrame based on these two variables. With this info, I will then understand the relationship between the user who sent the original tweet and the user who retweet the tweet message.

Implementation & Results

For increment 1:

Query 1:

In order to perform sentential analysis on the tweets, I begin by remove special characters from each tweets then use Python’s package textblob to evaluate the popularity of each tweets.

Query 2:

In order to count the created year distribution for all users in my dataset, I will need to get the distinct users with their created year. Then I can get user count with year in the group by clause.

Query 3:

In order to find the most popular login sources, I will need to get the distinct user id and login source for each distinct user. Then I will write use substring function to parse the source naem and put the query in a subquery. With the subquery, I can then put a query outside it to get the count of login source with source in the group by clause and order by count in decreasing order. Then I use limit 10 to get the top 5 records.

Query 4:

In order generate the word cloud based on hashtags’ popularity. I first start by exploding the hashtags field then convert all of them to lower cases. Then I group the hashtag with individual hashtag and perform count on the hashtag and order by count in the decreasing order.

Query 5:

In order to count the occurance of null and not null for geo tag, I am using SUM with CASE statement to check the occurrence of nulls and use count on the geo tag to check the occurrence of not null.

Query 6:

In order to count the country code distribution from geo tag, I will need to perform reverse the latitude and longitude into actual address under the help of Python’s package geopy. I

start by filtering out the rows where geo is null. Then use geopy to perform reverse lookup. Then I parsed the output to get the country code.

Query 7:

Other than use geo tag to get the country code distribution, Twitter also has place tag which also contains the country code. I start by filtering out the rows where place is null then I group the rows with column country_code and order by the count of each distinct country code in decreasing order and show the top 10 rows.

Query 8:

In order to check the percentage of tweets that are truncated, I group the rows by column truncated and use the count to divide by the total count of all rows in the table. This will give me percentage of messages that are truncated and not truncated.

Query 9:

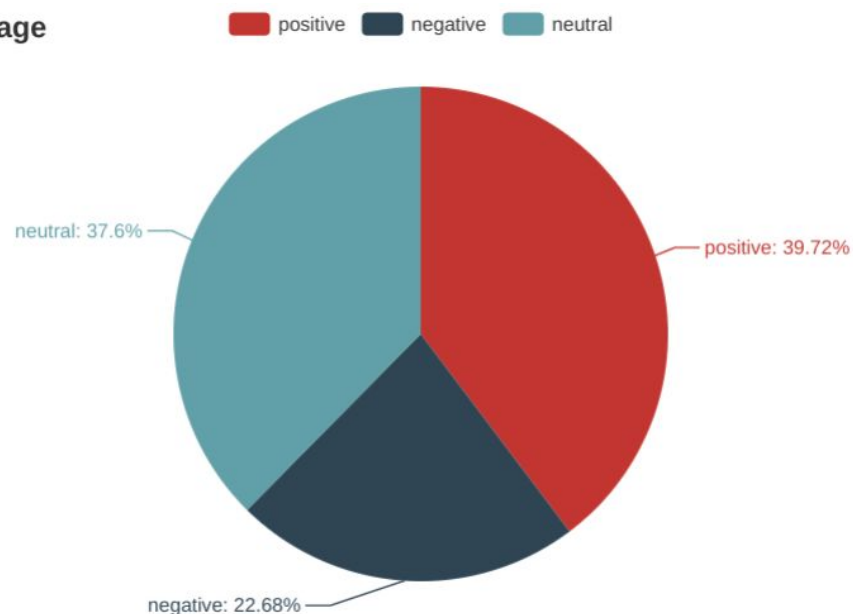
For user verified distribution and analysis, I group all rows by user.verified column to get the count of users that are using verified and unverified accounts. With this info, I then perform count on how many of each category are and are not using default profile.

Query 10:

For analysis on friends and followers count, I uses three separate queries to check how many users have friends count greater than followers count, how many users have friends count less than followers count, and how many users have same value for both friends and followers count.

Query 1:

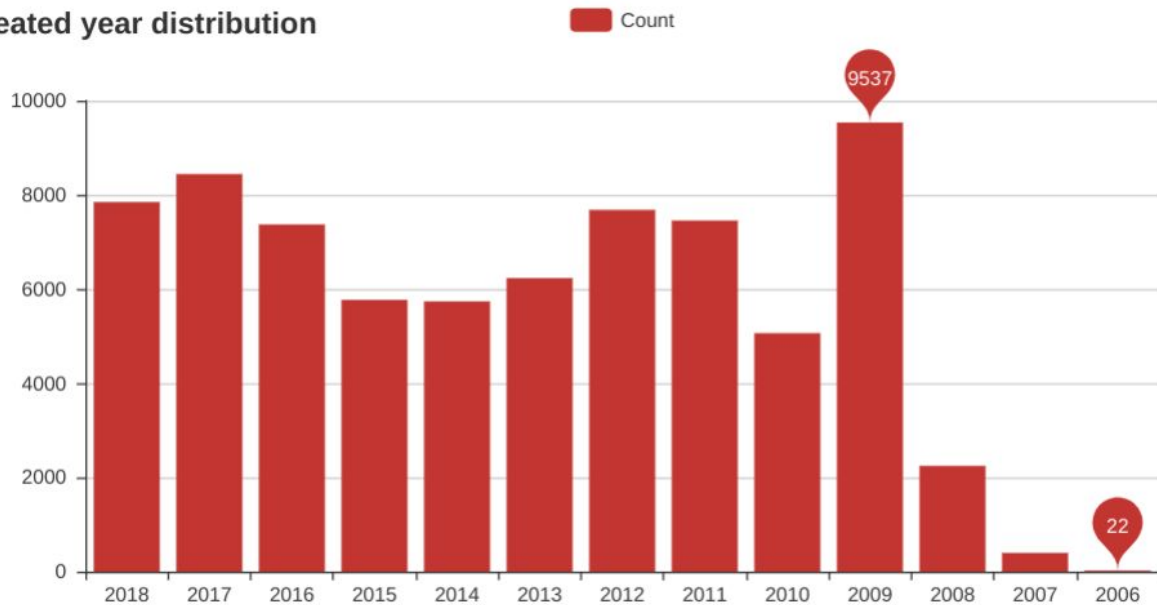
Polarity Percentage



Among the 100K tweets I collected, 39.72% of the tweets are positive, 37.6% of the tweets are neutral, and 22.68% of the tweets are negative.

Query 2:

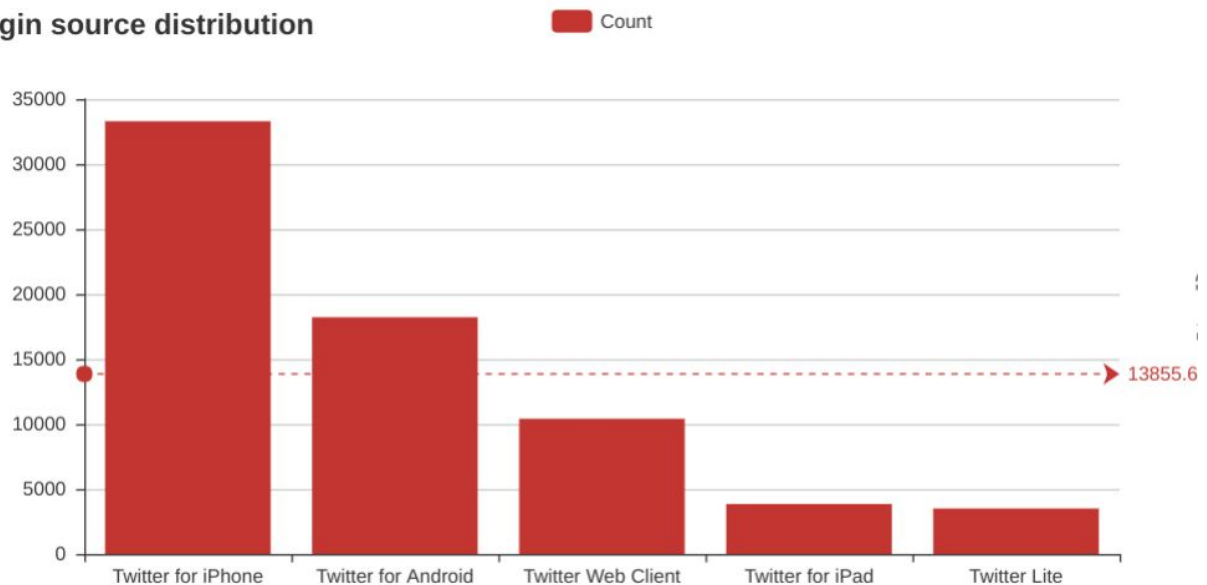
Created year distribution



Among the 100K tweets, there are 9537 accounts were created at 2009 (highest) and only 22 of them were created at 2006 (lowest).

Query 3:

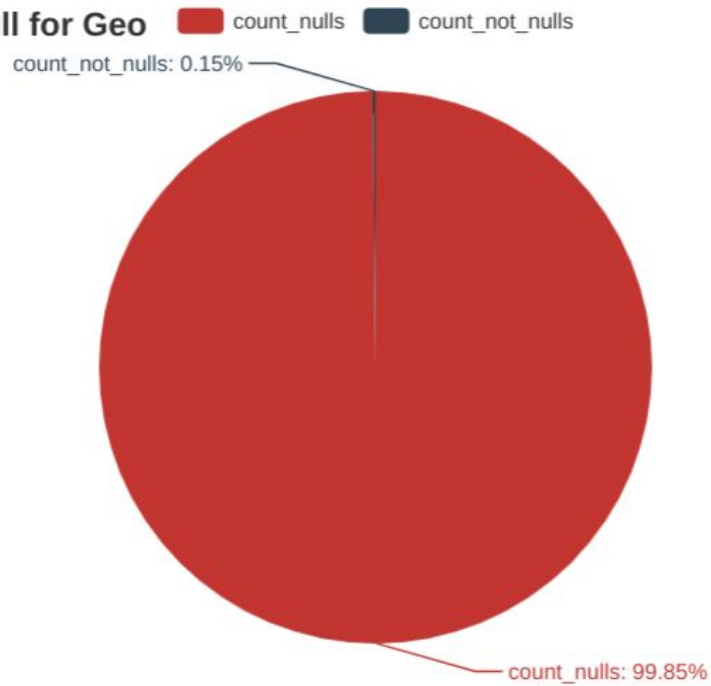
Login source distribution



Query 4:



Count Null and Not Null for Geo



When check the proportion of users who has geo info enable (with geo enabled, we can get the extra location of a user), there are only 0.15% of the users have geo enable. The remaining 99.85% of the users have that setting turned off.

Query 6:

| | Count |
|----------------------|-------|
| USA | 119 |
| Canada | 11 |
| New Zealand/Aotearoa | 3 |
| UK | 3 |
| Brasil | 2 |
| Philippines | 2 |
| ประเทศไทย | 1 |
| Chile | 1 |
| Ireland | 1 |
| Australia | 1 |
| 대한민국 | 1 |
| India | 1 |

Among the users with geo enabled, most of them are from USA (119 users).

Query 7:

| country_code | count |
|--------------|-------|
| US | 1147 |
| CA | 49 |
| GB | 47 |
| AU | 17 |
| PH | 10 |
| BR | 6 |
| IN | 6 |
| JP | 5 |
| NZ | 4 |
| MY | 4 |

Among the users with place enabled (with place enabled, we can only get the approximated location of a user), most of them are from USA (1147 users).

Query 8:

Message truncated percentage



There are only 9% of tweets that are truncated (when the message of a tweet is larger than the max size of a single tweet, Twitter will automatically truncate that tweet and send it with multiple smaller tweets instead of one large tweet).

Query 9:

```
+-----+-----+
|verified|count|
+-----+-----+
|    true| 1221|
|   false|98779|
+-----+-----+
```

There are 1221 users with account verified and the remaining 98779 users have account unverified.

```
+-----+-----+
|default_profile|count|
+-----+-----+
|             true|  218|
|             false| 1003|
+-----+-----+
```

Among the 1221 users with accounts verified, there are 218 of them are using default profile and 1003 of them are using custom profile.

```
+-----+-----+
|default_profile|count|
+-----+-----+
|             true|55598|
|             false|43181|
+-----+-----+
```

Among the 98779 unverified users, there are 55598 users with default profile and 43181 users with custom profile. This suggests a verified user is more likely to use custom profile.

Query 10:

```
: # Users count where the number of friends_count is greater than followers_count
data_df.where('user.friends_count > user.followers_count').count()
: 61730

: # Users count where the number of friends_count is less than followers_count
data_df.where('user.friends_count < user.followers_count').count()
: 37952

: # Users count where the number of friends_count is the same as followers_count
data_df.where('user.friends_count = user.followers_count').count()
: 318
```

There are 61730 users whose friends count is larger than the follower count. There are 37952 users whose friends count is less than followers count. There are 318 users with the same value for both friends and users counts.

For increment 2:

For machine learning, in order to use FPGrowth, I need to import the needed library:

```
from pyspark.ml.fpm import FPGrowth
```

I then generate computed columns based on the data I collected:

```
# Generate computed columns for geo_enabled, profile_background_tile, profile_use_background_image,
# and default profile and save the result to a list
data_list = data_df.select(
    when(data_df.truncated == "true", 1).otherwise(0).alias("truncated"),
    when(data_df.user_verified == "true", 1).otherwise(0).alias("verified"),
    when(data_df.user_geo_enabled == "true", 1).otherwise(0).alias("geo_enabled"),
    when(data_df.user_profile_background_tile == "true", 1).otherwise(0).alias("profile_background_tile"),
    when(data_df.user_profile_use_background_image == "true", 1).otherwise(0).alias("profile_use_background_image"),
    when(data_df.user_default_profile == "true", 1).otherwise(0).alias("default_profile")
).collect()
```

The reason for why I called collect() at the end of this function is to collect of of the computed columns into a python list object. Because I am want to FPGrowth, so I want data to be in same column instead of multiple columns:

```
# Generate new data list for ML
ml_data_list = []
index = 0

for row in data_list:
    temp_data_list = []
    if row[0] == 1:
        temp_data_list.append('truncated')
    if row[1] == 1:
        temp_data_list.append('verified')
    if row[2] == 1:
        temp_data_list.append('geo_enabled')
    if row[3] == 1:
        temp_data_list.append('profile_background_tile')
    if row[4] == 1:
        temp_data_list.append('profile_use_background_image')
    if row[5] == 1:
        temp_data_list.append('default_profile')
    if temp_data_list:
        ml_data_list.append((index, temp_data_list))
        index += 1
```

Then I converted this nested list back to a Spark dataframe:


```
# Generate analysis dataframe
analysis_df = spark.createDataFrame(ml_data_list, ['id', 'items'])
```

As the data pre-processing and post-processing are completed, I then generate the machine learning model with FPGrowth:

```
# Create the model and fit the dataframe into the model
fp_growth = FPGrowth(itemsCol="items", minSupport=0.50, minConfidence=0.6)
model = fp_growth.fit(analysis_df)
```

As what I explained above, FPGrowth will show me what is the most common combination of items/settings users used based on the dataset I collect, here is the result for most common settings people use:

```
# Display frequent itemsets
model.freqItemsets.show(truncate=False)
```

| items | freq |
|-------------------------------------------------|-------|
| [profile_use_background_image] | 80317 |
| [default_profile] | 55816 |
| [default_profile, profile_use_background_image] | 55816 |

Here is the association rules generated by my model:

```
# Display generated association rules
model.associationRules.show(truncate=False)
```

| antecedent | consequent | confidence |
|--------------------------------|--------------------------------|--------------------|
| [default_profile] | [profile_use_background_image] | 1.0 |
| [profile_use_background_image] | [default_profile] | 0.6949462753837917 |

Here is the distinct prediction count that generated by my model:

```
# Show distinct prediction count
model.transform(analysis_df).groupBy("prediction").count().show()
```

| prediction | count |
|-------------------|-------|
| [] | 64601 |
| [default_profile] | 24501 |

Here is a way to show the actual prediction based on the input:

```
# Display prediction
model.transform(analysis_df).show(truncate=False)
```

| id | items | prediction |
|----|-------------------------------------------------------------------------|-------------------|
| 0 | [profile_use_background_image, default_profile] | [] |
| 1 | [geo_enabled, profile_use_background_image, default_profile] | [] |
| 2 | [profile_use_background_image] | [default_profile] |
| 3 | [truncated, profile_use_background_image, default_profile] | [] |
| 4 | [profile_use_background_image, default_profile] | [] |
| 5 | [profile_use_background_image, default_profile] | [] |
| 6 | [profile_use_background_image, default_profile] | [] |
| 7 | [geo_enabled, profile_use_background_image] | [default_profile] |
| 8 | [geo_enabled, profile_use_background_image, default_profile] | [] |
| 9 | [profile_use_background_image, default_profile] | [] |
| 10 | [profile_use_background_image, default_profile] | [] |
| 11 | [truncated, geo_enabled, profile_use_background_image, default_profile] | [] |
| 12 | [truncated, profile_use_background_image] | [default_profile] |
| 13 | [geo_enabled, profile_background_tile, profile_use_background_image] | [default_profile] |
| 14 | [profile_use_background_image, default_profile] | [] |
| 15 | [profile_background_tile, profile_use_background_image] | [default_profile] |
| 16 | [profile_use_background_image, default_profile] | [] |
| 17 | [profile_use_background_image, default_profile] | [] |
| 18 | [profile_use_background_image, default_profile] | [] |
| 19 | [profile_use_background_image, default_profile] | [] |

only showing top 20 rows

For graph algorithm, I also need to import some needed packages:

```
from graphframes import *
```

With GraphFrames, I can then create edges and vertices and create a graph based on the edges and vertices:

```
# Create edges
e = data_df.select(
    col("user.id").alias("src"),
    col("retweeted_status.user.id").alias("dst"),
    lit("retweet").alias("relationship")
).where(
    col("retweeted_status.user.id").isNotNull()
).distinct()
```

```
# Create vertices
v = data_df.select(
    col("user.id"),
    col("user.screen_name")
).union(
    data_df.select(
        col("retweeted_status.user.id"),
        col("retweeted_status.user.name"),
    ).where(
        col("retweeted_status.user.id").isNotNull()
    )
).distinct()
```

```
# Create graph
g = GraphFrame(v, e)
```

The reasons behind the edges and vertices creation are explained in the section 'analysis of data'.

Here is the top 10 in-degree for vertices in decreasing order:

```
# Get top 10 id with highest in degree
g.inDegrees.sort("inDegree", ascending=False).show(10, False)
```

| id | inDegree |
|---------------------|----------|
| 1604444052 | 2201 |
| 487297085 | 1979 |
| 130496027 | 1579 |
| 4196983835 | 1359 |
| 1008440487915720708 | 1319 |
| 16989178 | 1110 |
| 19697415 | 1070 |
| 150078976 | 1056 |
| 2828212668 | 984 |
| 3267456386 | 930 |

only showing top 10 rows

Here is the top 10 in-degree for vertices in increasing order:

```
# Get top 10 id with lowest in degree
g.inDegrees.sort("inDegree").show(10, False)
```

| id | inDegree |
|--------------------|----------|
| 3358687222 | 1 |
| 18611344 | 1 |
| 2859622263 | 1 |
| 20813564 | 1 |
| 179176923 | 1 |
| 948171093818343425 | 1 |
| 540321025 | 1 |
| 72954856 | 1 |
| 95755482 | 1 |
| 135138678 | 1 |

only showing top 10 rows

Here is the top 10 out-degree for vertices in increasing order:

```
# Get top 10 id with highest out degree
g.outDegrees.sort("outDegree", ascending=False).show(10, False)
```

| id | outDegree |
|---------------------|-----------|
| 988374538491777025 | 48 |
| 1053011875007483905 | 24 |
| 824216698551209984 | 18 |
| 822210115784806400 | 18 |
| 80602426 | 17 |
| 2905278738 | 16 |
| 62147616 | 16 |
| 4053477192 | 16 |
| 954454874665771013 | 16 |
| 2874358611 | 15 |

only showing top 10 rows

Here I show some random 20 edges in my graph:

```
g.edges.show(10, False)
```

| src | dst | relationship |
|---------------------|------------|--------------|
| 828822710478331904 | 4196983835 | retweet |
| 798687870240182272 | 487297085 | retweet |
| 2199552860 | 112047805 | retweet |
| 903177352259260419 | 460226159 | retweet |
| 498933814 | 1398759560 | retweet |
| 231156352 | 111490230 | retweet |
| 1045729609793208320 | 1604444052 | retweet |
| 921562006955724800 | 49698134 | retweet |
| 572011574 | 4429003533 | retweet |
| 2776139595 | 788110982 | retweet |
| 2448240715 | 435207636 | retweet |
| 437169409 | 11856032 | retweet |
| 774786524 | 37824038 | retweet |
| 933613870756892673 | 2696243754 | retweet |
| 3374292849 | 4196983835 | retweet |
| 1045631048187617280 | 593948174 | retweet |
| 795963986650939392 | 14580438 | retweet |
| 992038732114034690 | 20878297 | retweet |
| 57198714 | 42447494 | retweet |
| 47761597 | 1339835893 | retweet |

only showing top 20 rows

I also used shortest path algorithm from the graph I created. I compute the shortest paths from each vertex to the given set of landmark vertex for one if with low number of in-degree:

```
# Compute shortest paths from each vertex to the given set of landmark vertices
# For one id with low number of in degree
g.shortestPaths(
  landmarks=["3358687222"]
).select("id", "distances").where(
  size(col("distances")) > 0
).show(10, False)
```

| id | distances |
|--------------------|-------------------|
| 885245073813798912 | [3358687222 -> 1] |
| 3358687222 | [3358687222 -> 0] |

Then I did a similar shortest path algorithm but with a vertex with high number of in-degree:

```
# Compute shortest paths from each vertex to the given set of landmark vertices
# For one id with high number of in degree
g.shortestPaths(
  landmarks=["150078976"]
).select("id", "distances").where(
  size(col("distances")) > 0
).show(10, False)
```

| id | distances |
|--------------------|------------------|
| 740005760801726465 | [150078976 -> 1] |
| 334776400 | [150078976 -> 1] |
| 849374799868637185 | [150078976 -> 1] |
| 935707527580397569 | [150078976 -> 1] |
| 3424129696 | [150078976 -> 1] |
| 827544166624325632 | [150078976 -> 1] |
| 303950400 | [150078976 -> 1] |
| 498077600 | [150078976 -> 2] |
| 241941600 | [150078976 -> 1] |
| 1566650000 | [150078976 -> 1] |

only showing top 10 rows

At the end, I also count the number of triangles created in my graph for each id whose count is not 0:

```
# Computes the number of triangles passing through each vertex
g.triangleCount().select("id", "count").where(col("count") > 0).show(10, False)
```

```
+-----+-----+
|id          |count|
+-----+-----+
|908149255654854656|104|
|883855148136763392|4|
|883855148136763392|4|
|1003631763133001729|4|
|719662077934243840|12|
|197111814          |4|
|862106769862078464|2|
|909519301299892225|40|
|909519301299892225|40|
|3100613023         |4|
+-----+-----+
only showing top 10 rows
```

Conclusion

From this project, I am using Jupyter Notebook as my main development tool for this project. I applied what I learned from class (e.g. Spark SQL, Spark ML, and Spark GraphFrames) to our project. I performed data analysis to the Twitter data I collected and generate interesting insights based on these data. I finished all of the work that we purposed in our project proposal. I think this project is pretty interesting.

Future Work

For the future work, I think it will be more interesting to find detail profile info for all of the users we collected. However, this will require a lot time due to the limitation from the Twitter API. With the detail info from each user, we can do more prediction and classification based on the each user.

Final Report - Harshil

Introduction

Each member of the team is asked to collect 100K English tweets with various tracks. The analysis for each member is done on their own set of tweets (100K). Due to the size of the dataset and our previous conversation, I will not upload this dataset to GitHub.

I have downloaded 100k tweets related to [trump,senate,house,voting]

Twitter has a lot interesting data and tweets that are easy to collect. We applied some SQL queries on the data we collected for increment 1. For increment 2, we will build machine learning model on the Twitter data we collected.

Detail Design of Features

For collecting data from Twitter, I am using tweepy package in Python. I wrote a python script to collect 100K English tweets with various tracks[trump,senate,house,voting]. With data I collected, I then used Spark and SparkSQL to perform 10 interesting queries on it.

Here are my queries:

Query 1:

Get top 10 user which have most favourites count

Query 2:

Find the tracks in the tweet and shows its distribution

Query 3:

Show tweets generated from united state and other than united states

Query 4:

Get the count of the user which have account verified

Query 5:

Get top 10 tweet that are retweeted the most

Query 6:

Get top 10 user which have most followers count

Query 7:

Get top 10 user which have most friends count

Query 8:

Get percentage of tweet based on negative, positive, neutral

Query 9:

Find gender of the user based on their name

Query 10:

Get the count of tweets

Analysis

Query 1:

I want to get the list of top 10 user which have the most favourites count.

Query 2:

I want to see how i received the tweet based upon my search criteria and want to show the distribution of the tweet in different title

Query 3:

I want to see how many people in united state tweet about trump,election,senate in us and rest of the world and shows it's comparison.

Query 4:

I want see how many people have the account verified(celebrities) on twitter.

Query 5:

I want to get the list of top 10 user tweet which are retweeted the most.

Query 6:

I want to get the list of top 10 user which have the most followers count.

Query 7:

I want to get the list of top 10 user which have the most friends count.

Query 8:

I want to use sentential analysis on the tweet to determine the polarity of each tweet then count how many tweets are positive, negative and neutral.

Query 9:

I want to use analysis on the tweet to determine the gender of the user who wrote the tweet

Query 10:

I want to get the count of the tweets

Background

For increment 2, each member of the team is asked to build a machine learning model based on the data we collected from increment 1. In order to do this, I want to perform sentiment analysis on 100k tweets. In increment 1, I have done this using already available python library [TextBlob]. For second increment, I created a new model using spark ml libraries to get the polarity of the tweets. To achieve this

- 1) retrieve data from twitter
- 2) cleaning up data
- 3) creating dataset with input (tweet) and label ([positive, negative])
- 4) used two different technique to build the model using sparkML

- HashingTF + IDF + Logistic Regression
- CountVectorizer + IDF + Logistic Regression

HashingTF + IDF + Logistic Regression

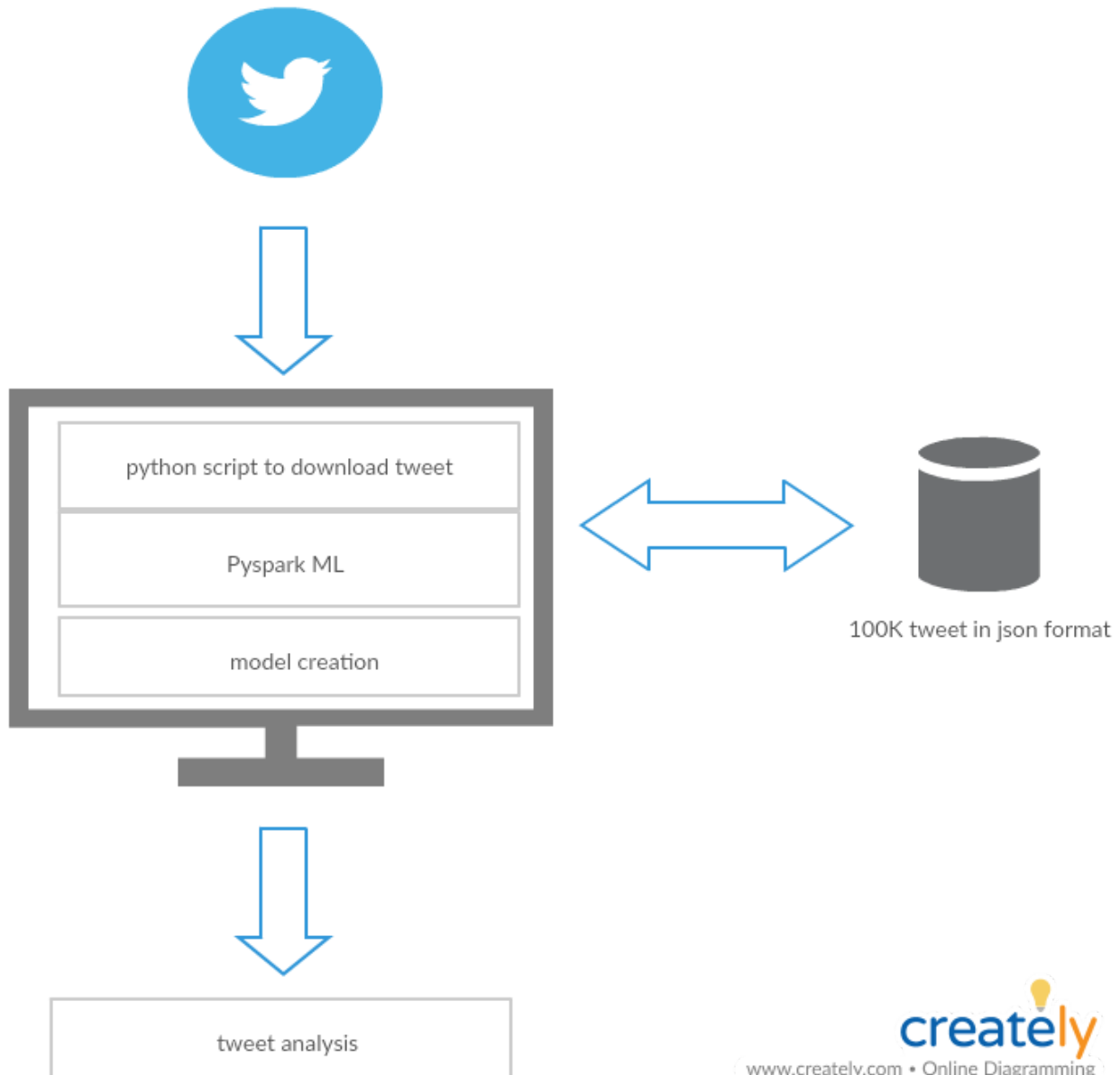
I learned that TF-IDF with Logistic Regression is quite strong combination, and showed robust performance, as high as Word2Vec + Convolutional Neural Network model. So in this post, I will try to implement TF-IDF + Logistic Regression model with Pyspark.

CountVectorizer + IDF + Logistic Regression

There's another way that you can get term frequency for IDF (Inverse Document Frequency) calculation. It is CountVectorizer in SparkML. Apart from the reversibility of the features (vocabularies), there is an important difference in how each of them filters top features. In case of HashingTF it is dimensionality reduction with possible collisions. CountVectorizer discards infrequent tokens.

Model

Architecture and workflow diagram:



I am using tweepy package in Python to stream 100k English tweets and persist to my workstation. By doing so, I can then apply data analytics with PySpark against this dataset. For increment 1, I wrote 10 interesting questions with SparkQL in Jupyter notebook with the data I collected from Twitter. For increment 2, I build two machine learning model in the Pycharm with the same dataset.

Dataset

From increment 1:

Each member of the team is asked to collect 100K English tweets with various tracks. The analysis for each member is done on their own set of tweets (100K). Due to the size of the dataset and our previous conversation, I will not upload this dataset to GitHub.

For increment 2:

In order to find some interesting data with a machine learning model, I created a new model using spark ml libraries to get the polarity of the tweets for this i use “text” field of the data. First i need to clean the text as it has special character which ml model can not understand and then create label with value 0,1[positive,negative].

```
C:\Users\harsh\AppData\Local\Programs\Python\Python36-32\python.exe C:/Users/harsh/Desktop/sem2/untitled1/sparkml.py
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Just created a SparkContext
```

| | _c0 text | target |
|----|------------------------------------------------------------------------------------------------------------------------------------------------------------|--------|
| 10 | RT Trump's press secretary who frequently calls accurate but critical reporting fake is sharing doctored video produced by | 1 |
| 11 | RT What is tweeting "Horseface from the residence about a porn star you allegedly slept with https://t.co/aAWcqaBwMw | 0 |
| 12 | RT Go to a hand recount in each county Supervise every vote count every vote & accept the results. Any other process will | 0 |
| 13 | RT I'm moving to Budapest later on this month and I know exactly what kind of country | 1 |
| 14 | RT Little-known fact There are currently more former car salesmen on the House Science Committee than Ph.D scientists. Little | 1 |

only showing top 5 rows

Analysis of data

I created a new model using spark ml libraries to get the polarity of the tweets for this i use “text” field of the data. First i need to clean the text as it has special character which ml model can not understand and then create label with value 0,1[positive,negative]. To achieve this

- 1) retrieve data from twitter
- 2) cleaning up data
- 3) creating dataset with input (tweet) and label ([positive,negative])
- 4) used two different technique to build the model using sparkML

- HashingTF + IDF + Logistic Regression
- CountVectorizer + IDF + Logistic Regression

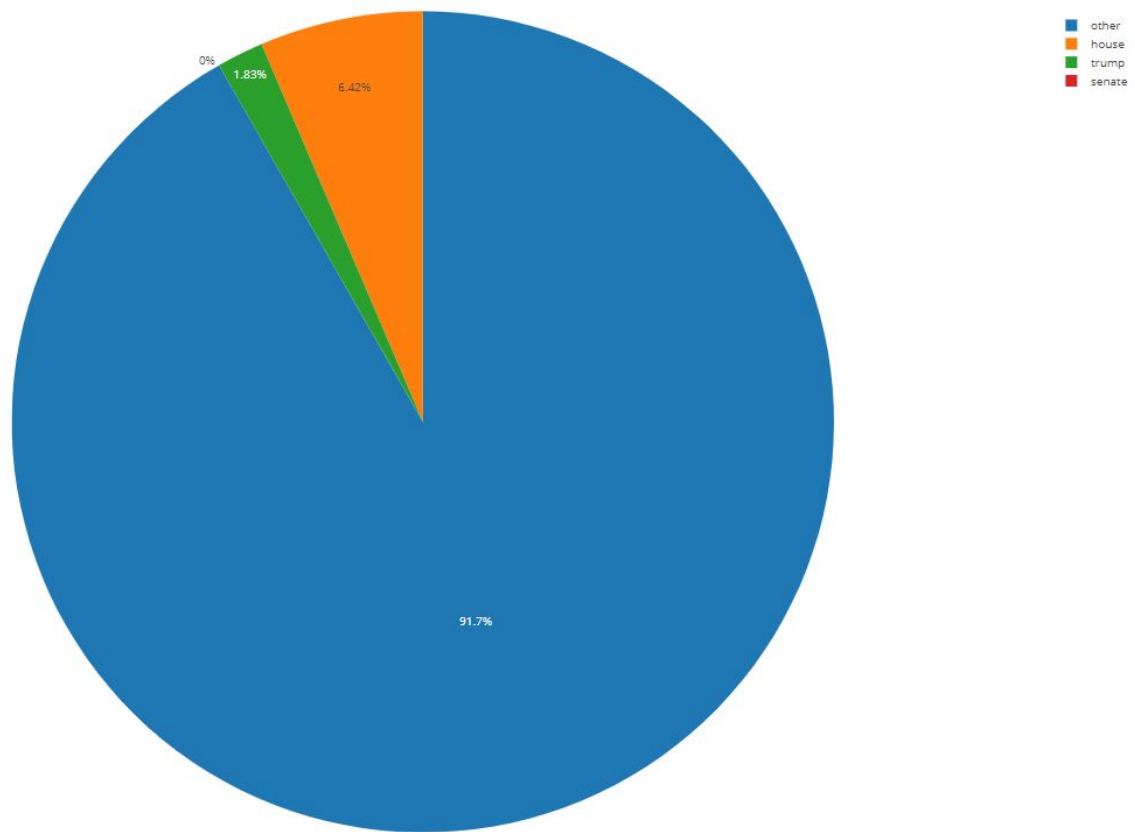
Implementation & Result

Query 1:

```
C:\Users\harsh\AppData\Local\Programs\Python\Python36-32\pyth
-----favourites_count-----
meh
1073419
Brian D.
965279
EnigMAA
851456
Madana Bhat-Khandige
823206
Monica Cates
821897
DerekPlatt
815322
rebecca lauren
812977
Brennen Burleson
800649
Jeanette Baratta
799905
Grand Moff Snarkin, Surefire Intelligence CFO
772169
```

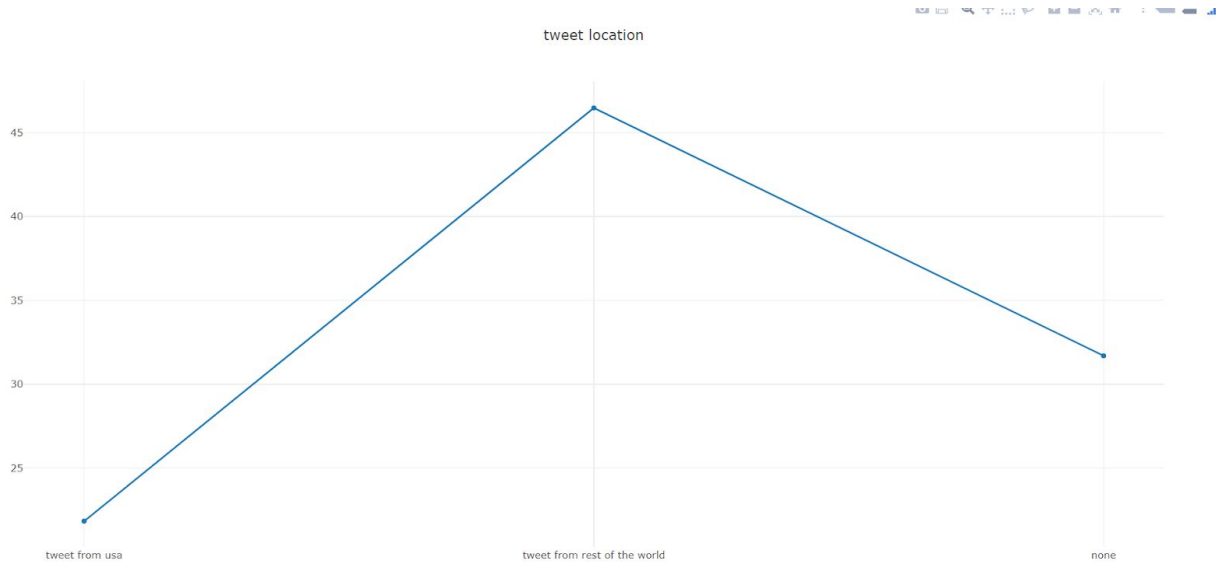
Among 100k tweet this are user names of top 10 user which have the most favourites count.the output show the username and the cout.

Query 2:



Among 100k tweet actual text containing keyword like trump,senate,house are where few most of the tweet does not contain that keywords in there tweets.

Query 3:



Among 100k tweet 22% of tweet where originated from united states and 46% of tweets were generated from rest of the world and remaining were having location unknown.

Query 4:

```
-----account verified-----  
1319
```

Among 100k tweet there were 1319 user which have the account verified.

Query 5:

```
-----tweet that are retweeted the most-----  
Rylie Geraci  
633089  
c h i e f  
178413  
BAYU ARISANDY  
125011  
monika bielskyte  
125010  
Rihanna  
125009  
Nasir Shakur  
125007  
shay  
125007  
RichFanAcc  
118884  
SaltSaltSalt  
110527  
rory miller  
107464
```

This show the list of top 10 users tweets that were retweeted.the output show user name and number of the time the tweet was retweeted.

Query 6:

```
-----followers_count-----  
Donald J. Trump  
55525198  
President Trump  
24414561  
The Economist  
23428140  
Reuters Top News  
19957413  
The White House  
17502570  
The Washington Post  
13010477  
The Washington Post  
13010416  
China Xinhua News  
11550188  
Jimmy Kimmel  
11427575  
HuffPost  
11389165  
-----
```

Among 100k tweet this are user names of top 10 user which have the most followers count.the output show the username and the cout.

Query 7:

```
-----friends_count-----  
Ed Krassenstein  
641614  
Travel  
572779  
Jeffrey Levin  
428351  
Jeffrey Levin  
428351  
Music Lovers Fans🎵  
306347  
🍷 MonsterFunder  
305853  
Tina Stull  
236552  
Reg Saddler  
229134  
Reg Saddler  
229134  
Dorian Sage 🍷  
227725  
-----
```

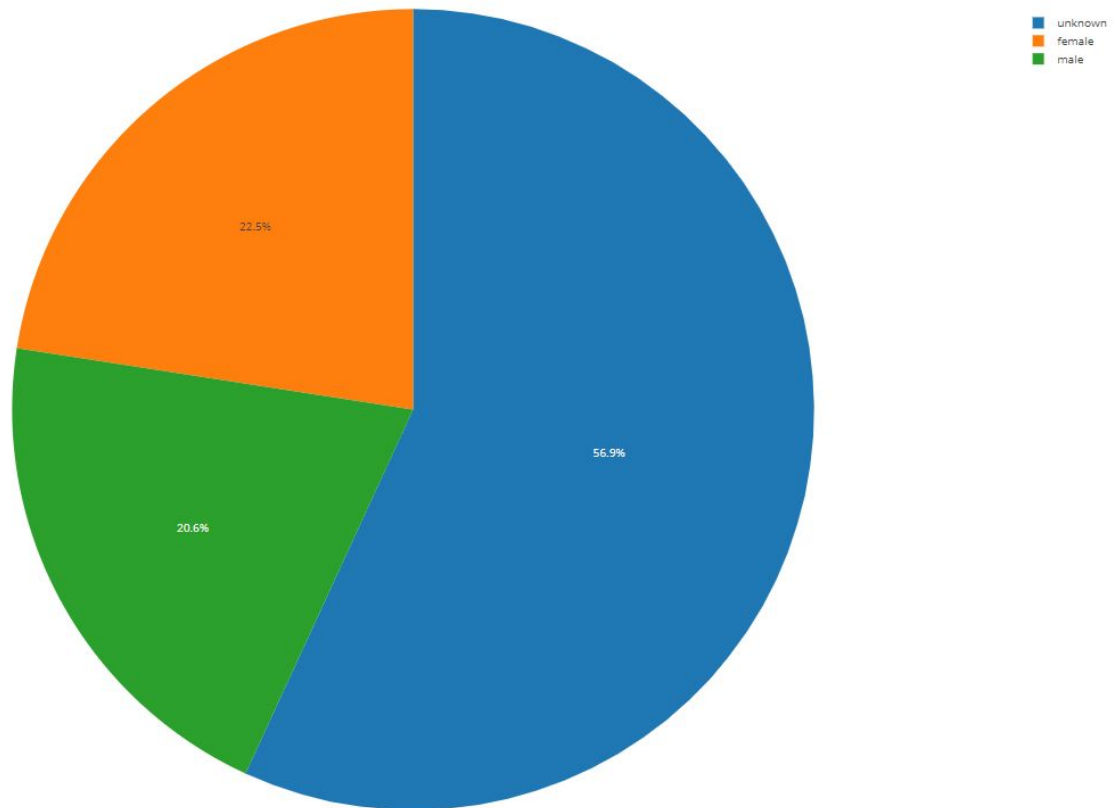
Among 100k tweet this are user names of top 10 user which have the most friends count.the output show the username and the cout.

Query 8:

```
-----negative postive tweet-----  
negative:27.449725502744972  
positive:34.229657703422966  
neutral:38.31961680383196
```

Among the 100K tweets, 34.23% of the tweets are positive, 38.32% of the tweets are neutral, and 27.44% of the tweets are negative.

Query 9:



Among the 100K tweets, 20.6% of the tweets are of male, 22.5% of the tweets are of female, and 56.9% of the tweets where unknown due the poor accuracy of the library.

Query 10:

```
-----count-----  
100000
```

As i have downloaded 100k tweet.it show 100k count of the tweet.

Increment 2

Cleaning data

I created new dataset(csv file) with three field sequence number, text, target(label) from json data

```
ss=str(c)+","+clean_tweet(x["text"]).replace(","," ").strip("\r\n")+","+str(result)+"\n"
```

c= sequence number

X[text] =tweet text

result= label[0, 1]

HashingTF + IDF + Logistic Regression

I first use hashing TF to index each word and then identified feature and then applied logistic regression on it.

```
from pyspark.ml.feature import HashingTF, IDF, Tokenizer
from pyspark.ml.feature import StringIndexer
from pyspark.ml import Pipeline

tokenizer = Tokenizer(inputCol="text", outputCol="words")
hashtf = HashingTF(numFeatures=2**16, inputCol="words", outputCol='tf')
idf = IDF(inputCol='tf', outputCol="features", minDocFreq=5) #minDocFreq: remove sparse terms
label_stringIdx = StringIndexer(inputCol = "target", outputCol = "label")
pipeline = Pipeline(stages=[tokenizer, hashtf, idf, label_stringIdx])

pipelineFit = pipeline.fit(train_set)
train_df = pipelineFit.transform(train_set)
val_df = pipelineFit.transform(val_set)
train_df.show(5,False)

from pyspark.ml.classification import LogisticRegression
lr = LogisticRegression(maxIter=100)
lrModel = lr.fit(train_df)
predictions = lrModel.transform(val_df)

from pyspark.ml.evaluation import BinaryClassificationEvaluator
evaluator = BinaryClassificationEvaluator(rawPredictionCol="rawPrediction")
evaluator.evaluate(predictions)
|
evaluator.getMetricName()

accuracy = predictions.filter(predictions.label == predictions.prediction).count() / float(val_set.count())
print(accuracy)
```

result:

| _c0 | text target | words | tf | features | label |
|-----|----------------------|------------------------|--------------------------|--------------------------|-------|
| 0 | RT Trump's press ... | 1 rt, trump's, pre... | (65536, [312, 7752, ...] | (65536, [312, 7752, ...] | 0.0 |
| 1 | RT What is tweeti... | 0 rt, what, is, tw... | (65536, [12716, 158... | (65536, [12716, 158... | 1.0 |
| 2 | RT Go to a hand r... | 0 rt, go, to, a, h... | (65536, [1038, 1354... | (65536, [1038, 1354... | 1.0 |
| 3 | RT I'm moving to ... | 1 rt, i'm, moving, ... | (65536, [1197, 8436... | (65536, [1197, 8436... | 0.0 |
| 4 | RT Little-known f... | 1 rt, little-known... | (65536, [19996, 290... | (65536, [19996, 290... | 0.0 |
| 5 | RT FACT funded th... | 0 rt, fact, funded... | (65536, [20464, 217... | (65536, [20464, 217... | 1.0 |
| 6 | RT Trump's firing... | 1 rt, trump's, fir... | (65536, [9639, 1553... | (65536, [9639, 1553... | 0.0 |
| 7 | RT _hooty_regula... | 1 rt, _hooty, _reg... | (65536, [3811, 7612... | (65536, [3811, 7612... | 0.0 |

only showing top 8 rows

```
18/11/28 19:51:04 WARN BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeSystemBLAS
18/11/28 19:51:04 WARN BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeRefBLAS
0.9415204678362573
```

Accuracy: 0.94

CountVectorizer + IDF + Logistic Regression

There's another way that you can get term frequency for IDF (Inverse Document Frequency) calculation. It is CountVectorizer in SparkML. Apart from the reversibility of the features (vocabularies), there is an important difference in how each of them filters top features. In case of HashingTF it is dimensionality reduction with possible collisions. CountVectorizer discards infrequent tokens.

```
from pyspark.ml.feature import CountVectorizer

tokenizer = Tokenizer(inputCol="text", outputCol="words")
cv = CountVectorizer(vocabSize=2**16, inputCol="words", outputCol='cv')
idf = IDF(inputCol='cv', outputCol="features", minDocFreq=5) #minDocFreq: remove sparse terms
label_stringIdx = StringIndexer(inputCol = "target", outputCol = "label")
lr = LogisticRegression(maxIter=100)
pipeline = Pipeline(stages=[tokenizer, cv, idf, label_stringIdx, lr])

pipelineFit = pipeline.fit(train_set)
predictions = pipelineFit.transform(val_set)
accuracy = predictions.filter(predictions.label == predictions.prediction).count() / float(val_set.count())
roc_auc = evaluator.evaluate(predictions)

print("Accuracy Score: {0:.4f}".format(accuracy))
print("ROC-AUC: {0:.4f}".format(roc_auc))
```

Result:

```
Accuracy Score: 0.9181
ROC-AUC: 0.9544
```

```
Process finished with exit code 0
```

Graph:

For implementing graphs i have used python GraphFrames library

I am creating graph with vertices as used_id and edges as retweet in a particular location for this graph i am using location as united states.

```
g.degrees.show(5)
```

```
+-----+-----+
|          id|degree|
+-----+-----+
|    133938408|    13|
|    2896675593|    1|
|823651149823688705|    4|
|883855148136763392|    2|
|818632400641097733|    1|
+-----+-----+
```

only showing top 5 rows

vertices

```
+-----+-----+
|          id| screen_name|
+-----+-----+
|    210185175| emmaburnsapp|
|    822926826|    kjtc1979|
|943892654160531457|    aesopgalt|
|767072603278159872|m8nkey2chm00n|
|    215441260|    LPWhitt|
+-----+-----+
```

only showing top 5 rows

Edges:

```
+-----+-----+-----+
|          src|          dst|relationship|
+-----+-----+-----+
|963967475875500033|          395674143|    retweet|
|          320908938|1034665733005959168|    retweet|
|789588266198700036|          232901331|    retweet|
|          385689121|          4731701624|    retweet|
|879531117811965953| 988960980548931585|    retweet|
+-----+-----+-----+
only showing top 5 rows
```

Get top 5 id with highest in degree:

```
g.inDegrees.sort("inDegree", ascending=False).show(5, False)
```

```
+-----+-----+
|id          |inDegree|
+-----+-----+
|25073877    |105     |
|21728303    |44      |
|822215673812119553|34      |
|18643437    |32      |
|91386979    |31      |
+-----+-----+
only showing top 5 rows
```

Get top 5 id with lowest in degree:

```
g.inDegrees.sort("inDegree").show(5, False)
```

```

+-----+-----+
|id          |inDegree|
+-----+-----+
|15469000    |1       |
|577432615   |1       |
|864068108   |1       |
|1038512907120779264|1       |
|53190110    |1       |
+-----+-----+
only showing top 5 rows

```

Compute shortest paths from each vertex to the given set of “25073877” vertices:

```

g.shortestPaths(landmarks=["25073877"]).select("id", "distances").where(size(col("distances")) > 0).show(10, False)

```

```

+-----+-----+
|id          |distances|
+-----+-----+
|954609953087553536|Map(25073877 -> 1)|
|733279356492079106|Map(25073877 -> 1)|
|877879587493085184|Map(25073877 -> 1)|
|43870406         |Map(25073877 -> 1)|
|977695071096262658|Map(25073877 -> 1)|
|2880381305       |Map(25073877 -> 1)|
|2370953706       |Map(25073877 -> 1)|
|976262602421448705|Map(25073877 -> 1)|
|227835612        |Map(25073877 -> 1)|
|2149493108       |Map(25073877 -> 1)|
+-----+-----+
only showing top 10 rows

```

Final Report - Shaun

Introduction

Increment 1:

I pulled 100k Tweets and performed some basic queries on those Tweets to develop some insights. I used HQL and Hive to perform the analysis, so limited options were available in terms of insights that could be drawn.

Increment 2:

I realized that the Tweets that I collected in the last increment were strictly marketing Tweets. For this increment I switched to Tweets about food, so there would be some interesting connections that I could make. I managed to get 100,000 unique Tweets together into a single csv file. This took quite some time, as Twitter limits the rate that you can take tweets from their server.

Background

Increment 1:

The main purpose of this increment was to get the data together and perform some introductory queries on the data to get some background. The next increment is where the data is then loaded into graph frames to enable further analysis to be done in the future.

Increment 2:

I used Rstudio and the twitterR package to get the tweets and then loaded them into a csv file. I imported the csv into a spark dataframe and then make the data into a graphframe. This give the opportunity for further analysis to be done in the final increment. This increment can be seen as setting the foundation for further analysis that we will be made in the future.

Model

Increment 1:

I get the tweets from Twitter and then they go into a csv file.
From the csv file they get loaded into a spark data frame.

Increment 2:

I get the tweets from Twitter and then they go into a csv file.
From the csv file they get loaded into a spark data frame.
From the data frame they get constructed into graphframes

Dataset

Increment 1 & 2:

The fields are as follows. There are 100,000 total rows.

- text
- favorited
- favoriteCount
- replyToSN
- created truncated
- replyToSID
- id
- replyToUID
- statusSource
- screenName
- retweetCount
- isRetweet
- retweeted
- longitude
- latitude

Analysis of data, Implementation & Results

Increment 1:

To collect my data from Twitter I used R and RStudio in conjunction with twitterR to pull the tweets in and then exported to csv and put it into a Hive table.

Query 1:

Find total number of favorites for all the Tweets.

Select sum(favoritecount) from tweets;


```
cloudera@quickstart:~/Downloads
File Edit View Search Terminal Help
set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
set mapreduce.job.reduces=<number>
Starting Job = job_1541805655545_0005, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1541805655545_0005/
Kill Command = /usr/lib/hadoop/bin/hadoop job -kill job_1541805655545_0005
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2018-11-09 19:06:34,989 Stage-1 map = 0%, reduce = 0%
2018-11-09 19:06:44,909 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 2.39 sec
2018-11-09 19:06:54,648 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 4.01 sec
MapReduce Total cumulative CPU time: 4 seconds 10 msec
Ended Job = job_1541805655545_0005
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 4.01 sec HDFS Read: 3225976
5 HDFS Write: 6 SUCCESS
Total MapReduce CPU Time Spent: 4 seconds 10 msec
OK
36852
Time taken: 30.182 seconds, Fetched: 1 row(s)
hive>
```

Query 2:

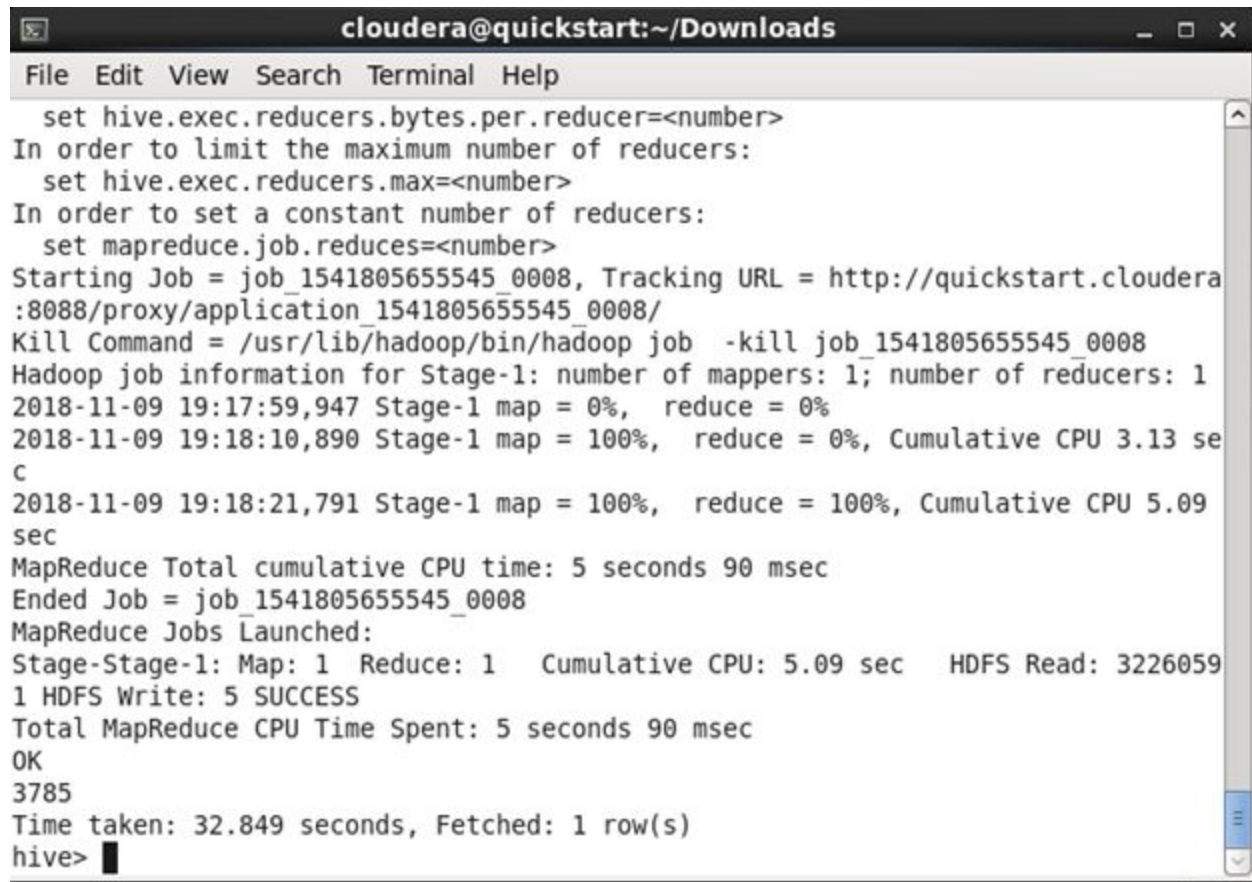
Find how many tweets contain the word Trump.

```
cloudera@quickstart:~/Downloads
File Edit View Search Terminal Help
set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
set mapreduce.job.reduces=<number>
Starting Job = job_1541805655545_0007, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1541805655545_0007/
Kill Command = /usr/lib/hadoop/bin/hadoop job -kill job_1541805655545_0007
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2018-11-09 19:11:34,363 Stage-1 map = 0%, reduce = 0%
2018-11-09 19:11:44,401 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 2.85 sec
2018-11-09 19:11:56,258 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 4.5 sec
MapReduce Total cumulative CPU time: 4 seconds 500 msec
Ended Job = job_1541805655545_0007
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 4.5 sec HDFS Read: 32259979
HDFS Write: 3 SUCCESS
Total MapReduce CPU Time Spent: 4 seconds 500 msec
OK
29
Time taken: 32.686 seconds, Fetched: 1 row(s)
hive>
```

Select count(text) from tweets where text like “%Trump%” or text like “%trump%”;

Query 3

Find how many tweets are from the Northern hemisphere.

A screenshot of a terminal window titled "cloudera@quickstart:~/Downloads". The terminal shows the execution of a Hive job. It starts with setting Hive configuration properties: "set hive.exec.reducers.bytes.per.reducer=<number>" and "set hive.exec.reducers.max=<number>". It then shows the starting job information: "Starting Job = job_1541805655545_0008, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1541805655545_0008/". The job is then killed with the command "Kill Command = /usr/lib/hadoop/bin/hadoop job -kill job_1541805655545_0008". The terminal displays the Hadoop job information for Stage-1: "Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1". It shows the progress of the job: "2018-11-09 19:17:59,947 Stage-1 map = 0%, reduce = 0%", "2018-11-09 19:18:10,890 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 3.13 sec", and "2018-11-09 19:18:21,791 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 5.09 sec". The final output shows the job completion: "MapReduce Total cumulative CPU time: 5 seconds 90 msec", "Ended Job = job_1541805655545_0008", "MapReduce Jobs Launched:", "Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 5.09 sec HDFS Read: 3226059", "1 HDFS Write: 5 SUCCESS", "Total MapReduce CPU Time Spent: 5 seconds 90 msec", "OK", "3785", "Time taken: 32.849 seconds, Fetched: 1 row(s)", and "hive>".

```
cloudera@quickstart:~/Downloads
File Edit View Search Terminal Help
set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
set mapreduce.job.reduces=<number>
Starting Job = job_1541805655545_0008, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1541805655545_0008/
Kill Command = /usr/lib/hadoop/bin/hadoop job -kill job_1541805655545_0008
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2018-11-09 19:17:59,947 Stage-1 map = 0%, reduce = 0%
2018-11-09 19:18:10,890 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 3.13 sec
2018-11-09 19:18:21,791 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 5.09 sec
MapReduce Total cumulative CPU time: 5 seconds 90 msec
Ended Job = job_1541805655545_0008
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 5.09 sec HDFS Read: 3226059
1 HDFS Write: 5 SUCCESS
Total MapReduce CPU Time Spent: 5 seconds 90 msec
OK
3785
Time taken: 32.849 seconds, Fetched: 1 row(s)
hive>
```

Select count(text) from tweets where latitude > 0;

Query 4

See how many tweets are from the southern hemisphere.

Select count(text) from tweets where latitude < 0;


```
cloudera@quickstart:~/Downloads
File Edit View Search Terminal Help
set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
set mapreduce.job.reduces=<number>
Starting Job = job_1541805655545_0009, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1541805655545_0009/
Kill Command = /usr/lib/hadoop/bin/hadoop job -kill job_1541805655545_0009
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2018-11-09 19:20:39,587 Stage-1 map = 0%, reduce = 0%
2018-11-09 19:20:50,903 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 3.13 sec
2018-11-09 19:21:00,722 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 4.82 sec
MapReduce Total cumulative CPU time: 4 seconds 820 msec
Ended Job = job_1541805655545_0009
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 4.82 sec HDFS Read: 32260579 HDFS Write: 2 SUCCESS
Total MapReduce CPU Time Spent: 4 seconds 820 msec
OK
0
Time taken: 32.725 seconds, Fetched: 1 row(s)
hive>
```

I should have gotten a total of 100000 between this query and the last. The problem turns out that a vast majority of the tweets don't share their location.

Query 5

Find out how many Tweets contain the word the.

Select count(text) from tweets where text like "%the%";

```
cloudera@quickstart:~/Downloads
File Edit View Search Terminal Help
set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
set mapreduce.job.reduces=<number>
Starting Job = job_1541805655545_0010, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1541805655545_0010/
Kill Command = /usr/lib/hadoop/bin/hadoop job -kill job_1541805655545_0010
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2018-11-09 19:26:42,716 Stage-1 map = 0%, reduce = 0%
2018-11-09 19:26:55,052 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 2.78 sec
2018-11-09 19:27:05,778 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 4.42 sec
MapReduce Total cumulative CPU time: 4 seconds 420 msec
Ended Job = job_1541805655545_0010
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 4.42 sec HDFS Read: 3225967
6 HDFS Write: 6 SUCCESS
Total MapReduce CPU Time Spent: 4 seconds 420 msec
OK
19053
Time taken: 33.608 seconds, Fetched: 1 row(s)
hive>
```

Query 6

Find out the most common latitude and longitude for the tweets;

Select avg(latitude) from tweets;

```
cloudera@quickstart:~/Downloads
File Edit View Search Terminal Help
set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
set mapreduce.job.reduces=<number>
Starting Job = job_1541805655545_0010, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1541805655545_0010/
Kill Command = /usr/lib/hadoop/bin/hadoop job -kill job_1541805655545_0010
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2018-11-09 19:26:42,716 Stage-1 map = 0%, reduce = 0%
2018-11-09 19:26:55,052 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 2.78 sec
2018-11-09 19:27:05,778 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 4.42 sec
MapReduce Total cumulative CPU time: 4 seconds 420 msec
Ended Job = job_1541805655545_0010
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 4.42 sec HDFS Read: 3225967
6 HDFS Write: 6 SUCCESS
Total MapReduce CPU Time Spent: 4 seconds 420 msec
OK
19053
Time taken: 33.608 seconds, Fetched: 1 row(s)
hive>
```

Query 7

Find the most common longitude in the Tweets.

Select avg(longitude) from tweets;

```
cloudera@quickstart:~/Downloads
File Edit View Search Terminal Help
set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
set mapreduce.job.reduces=<number>
Starting Job = job_1541805655545_0012, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1541805655545_0012/
Kill Command = /usr/lib/hadoop/bin/hadoop job -kill job_1541805655545_0012
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2018-11-09 19:33:15,972 Stage-1 map = 0%, reduce = 0%
2018-11-09 19:33:25,797 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 2.51 sec
2018-11-09 19:33:37,654 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 4.21 sec
MapReduce Total cumulative CPU time: 4 seconds 210 msec
Ended Job = job_1541805655545_0012
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 4.21 sec HDFS Read: 3226004
7 HDFS Write: 19 SUCCESS
Total MapReduce CPU Time Spent: 4 seconds 210 msec
OK
3.9695155902004453
Time taken: 32.373 seconds, Fetched: 1 row(s)
hive>
```

Query 8

Find out what how many Tweets contain user generated content.

Select count(text) from tweets where text like "%User%" and text like "%Generated%";

```
cloudera@quickstart:~/Downloads
File Edit View Search Terminal Help
set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
set mapreduce.job.reduces=<number>
Starting Job = job_1541805655545_0012, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1541805655545_0012/
Kill Command = /usr/lib/hadoop/bin/hadoop job -kill job_1541805655545_0012
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2018-11-09 19:33:15,972 Stage-1 map = 0%, reduce = 0%
2018-11-09 19:33:25,797 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 2.51 sec
2018-11-09 19:33:37,654 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 4.21 sec
MapReduce Total cumulative CPU time: 4 seconds 210 msec
Ended Job = job_1541805655545_0012
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 4.21 sec HDFS Read: 3226004
7 HDFS Write: 19 SUCCESS
Total MapReduce CPU Time Spent: 4 seconds 210 msec
OK
3.9695155902004453
Time taken: 32.373 seconds, Fetched: 1 row(s)
hive>
```

Query 9

Find out how many total retweets are in the dataset.

Select sum(retweetcount) from tweets;


```
cloudera@quickstart:~/Downloads
File Edit View Search Terminal Help
set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
set mapreduce.job.reduces=<number>
Starting Job = job_1541805655545_0014, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1541805655545_0014/
Kill Command = /usr/lib/hadoop/bin/hadoop job -kill job_1541805655545_0014
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2018-11-09 19:39:56,156 Stage-1 map = 0%, reduce = 0%
2018-11-09 19:40:05,960 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 2.64 sec
2018-11-09 19:40:16,739 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 4.32 sec
MapReduce Total cumulative CPU time: 4 seconds 320 msec
Ended Job = job_1541805655545_0014
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 4.32 sec HDFS Read: 3225976
2 HDFS Write: 7 SUCCESS
Total MapReduce CPU Time Spent: 4 seconds 320 msec
OK
426624
Time taken: 31.288 seconds, Fetched: 1 row(s)
hive>
```

Query 10

Find out how long the longest tweet is.

Select max(length(text)) from tweets;

```
cloudera@quickstart:~/Downloads
File Edit View Search Terminal Help
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1541805655545_0015, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1541805655545_0015/
Kill Command = /usr/lib/hadoop/bin/hadoop job -kill job_1541805655545_0015
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2018-11-09 19:45:06,715 Stage-1 map = 0%, reduce = 0%
2018-11-09 19:45:18,573 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 2.9 sec
2018-11-09 19:45:29,326 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 4.52 sec
MapReduce Total cumulative CPU time: 4 seconds 520 msec
Ended Job = job_1541805655545_0015
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 4.52 sec HDFS Read: 3226015
2 HDFS Write: 4 SUCCESS
Total MapReduce CPU Time Spent: 4 seconds 520 msec
OK
820
Time taken: 33.2 seconds, Fetched: 1 row(s)
hive>
```

Increment 2:

As you can see here I have the code that I used to load the data into Spark and then had to use SQL to filter the data down the way that I wanted to. This included filtering out all of the tweets that weren't replies so that I could use that as the source for the vertices.

```
val tweets = spark.read.format("csv").option("header", "true").load(path = "C:\\Users\\calcalocalo\\Documents\\Food
tweets.createOrReplaceTempView(viewName = "tweet")
val v = spark.sql(sqlText = "select id, text, retweetCount from tweet where replyToSID <> 'NA'")
val e = spark.sql(sqlText = "select id as src, replyToSID as dst from tweet where replyToSID <> 'NA'")

val g = GraphFrame(v, e)
g.vertices.show()
g.edges.show()
```

I then show the vertices and edges. I will show some screenshots in the results phase of the report.

Implementation & Results

First you can see some of the vertices of the graph.

| id | text | retweetCount |
|---------------------|----------------------------------|--------------|
| NA | FALSE | TRUE |
| NA | FALSE | FALSE |
| NA | FALSE | FALSE |
| NA | FALSE | FALSE |
| NA | FALSE | TRUE |
| NA | FALSE | FALSE |
| NA | FALSE | TRUE |
| NA | FALSE | FALSE |
| 1067919125257818112 | @nova_meat @Moeda... | 0 |
| NA | FALSE | TRUE |
| NA | FALSE | FALSE |
| NA | FALSE | FALSE |
| NA | FALSE | FALSE |
| NA | FALSE | TRUE |
| NA | ""But #Daniel #r... firstseekHim | |
| NA | FALSE | FALSE |
| NA | FALSE | TRUE |
| NA | FALSE | FALSE |
| NA | FALSE | TRUE |
| NA | FALSE | TRUE |

only showing top 20 rows

Here are some of the edges of the graph.


```

18/11/28 19:38:57 INFO DAGScheduler: Job 2 finished: show at SparkGraphFrame.scala:39, took 0.039184 s
+-----+
|          src|          dst|
+-----+
|      NA|1067920754518437888|
|      NA|1067920694263070720|
|      NA|1067920419930480640|
|      NA|1067920021735817216|
|      NA|1067919970221346818|
|      NA|1067919789648162816|
|      NA|1067919734211923968|
|      NA|1067919495262597126|
|1067919125257818112|1067887026190655491|
|      NA|1067919029522841600|
|      NA|1067918989727154176|
|      NA|1067918692120444930|
|      NA|1067918586482737156|
|      NA|1067918461869809665|
|      NA|          TRUE|
|      NA|1067918239747911680|
|      NA|1067917706199007233|
|      NA|1067917679300960257|
|      NA|1067917632446320640|
|      NA|1067917507472822273|
+-----+
only showing top 20 rows

```

Having the data in this form will make way to make the final increment where I will be able to perform machine learning analysis on the dataset.

Conclusion

I would say that there is a lot more that could be done with this project in order to make it more robust. I found that it was cool be able to go from a datasource like Twitter and get the data into a form like a graphframe. All in all I learned a lot through the process and found this very meaningful.

Future Work

In the future some more advanced machine learning algorithms could applied to the data in order to figure out trends. See what kind of things are being said about food. Would be cool!

Final Report - Stephanie

Introduction

I used tweepy and python to create a json file. Then I read this json file and performed queries in scala using spark. The tweets dataset also had machine learning and graphs applied to it. A model is created and trained using the tweets dataset. Then it is tested to see how accurate it is when predicting. Graphframes were used to create a graph to illustrate and understand how the data is connected.

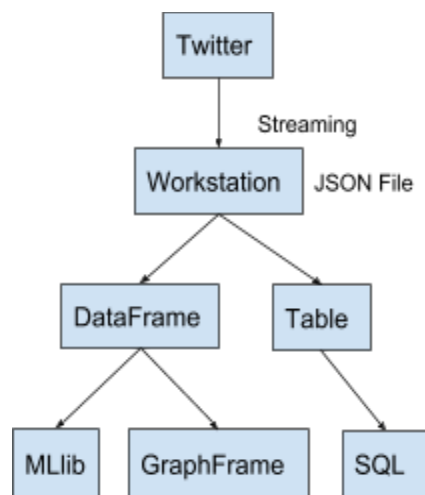
Background

First I used tweepy with python to stream tweets from twitter. These tweets were collected and saved as a json file. This file was then used multiple times. First it was used to create a table where spark queries were run to analyze the data. Then it was used to create dataFrames and graphFrames.

I used Spark's Dataframes and MLlib in scala to create the model. I wanted to see how much people were communicating with one another on twitter. So I selected features I believed were related to communication. These were the columns "favorite_count", "id", "in_reply_to_screen_name", "is_quote_status", "reply_count", "retweeted", "text", and "lang". Then I used Pipeline in MLlib to generate my model and RegressionEvaluator to evaluate how accurate it is.

I used graphFrames to evaluate the tweets as well. Once I had my graph I found the in-degree and out-degree of the vertices, and counted the number of triangles in the graph.

Model



I streamed the tweets from twitter. These tweets were saved as a JSON file. Then a table could be created from this JSON file and queried using spark. Spark has dataFrames where the data can have a table like structure. Then I could select the columns I needed for the machine learning. A model was created using a training subset of the tweets dataset. Then the model was tested and evaluated using a testing subset of the tweets dataset.

The GraphFrame was created by first creating two dataFrames. A vertice dataFrame from the tweets dataset, and an edges dataFrame. These two dataFrames were then used to create the graphFrame. The graph then had algorithms applied to it.

Dataset

The dataset was obviously the collection of tweets. It had many properties but this section would be far too long if I described every column. But for the machine learning I used a subset of that larger dataset. I only selected the columns "favorite_count", "id", "in_reply_to_screen_name", "is_quote_status", "reply_count", "retweeted", "text", and "lang" This means that these were my

features for the machine learning. I wanted to look at communication so here is the description for each feature I selected:

- favorite_count: this is how many times the tweet was favorited by other users
- id: the id of the user
- in_reply_to_screen_name: this shows if the tweet was a reply to another user and which user
- Is_quote_status: this shows if the tweet was quoting another tweet
- reply_count: this is how many replies a tweet has
- retweeted: if a user has retweeted the tweet
- text: the text of the tweet it sometimes mentions other users but this would take more time to analyze
- lang: the language the user is tweeting in

Analysis of Data

When using the data, I did not do it as well as I could have. The only real data “pre-processing” I did for the model creation was select the columns I needed and make sure there were no null values. I could have done more to clean it up. If I had, perhaps my model would have been more accurate.

Implementation & Results

Increment 1:

Queries

Query 1 Count how many tweets were favorited and those that were not

```
//1
val favTweets = sqlContext
  .sql( sqlText = "SELECT favorited, count(*) as count FROM EntertainmentTable where favorited is not null group by favorited order by count desc limit 10")
favTweets.show
```

Query 2 Count how many tweets were retweeted and those that were not

```
//2
val retweets = sqlContext
  .sql( sqlText = "SELECT retweeted, count(*) as count FROM EntertainmentTable where retweeted is not null group by retweeted order by count desc limit 10")
retweets.show
```

Query 3 Count how many tweets were quotes and those that were not

```
//3
val quoteTweets = sqlContext
  .sql( sqlText = "SELECT is_quote_status, count(*) as count FROM EntertainmentTable where is_quote_status is not null group by is_quote_status order by count")
quoteTweets.show
```

| is_quote_status | count |
|-----------------|-------|
| false | 802 |
| true | 198 |

Query 4 See what filter levels people use and how many

```
//4
val filterTweets = sqlContext
  .sql( sqlText = "SELECT filter_level, count(*) as count FROM EntertainmentTable where filter_level is not null group by filter_level order by count desc limit 10")
filterTweets.show
```

| filter_level | count |
|--------------|-------|
| low | 1000 |

Query 5 Count how many tweets were truncated and those that were not

```
//5
val truncatedTweets = sqlContext
  .sql( sqlText = "SELECT truncated, count(*) as count FROM EntertainmentTable where truncated is not null group by truncated order by count desc limit 10")
truncatedTweets.show
```

| truncated | count |
|-----------|-------|
| false | 905 |
| true | 95 |

Query 6 Count how many tweets were quotes and truncated

```
//6
val tqTweets = sqlContext
  .sql( sqlText = "SELECT id, truncated, is_quote_status, count(*) as count FROM EntertainmentTable where truncated = true AND is_quote_status = true group by id")
tqTweets.show
```

| id | truncated | is_quote_status | count |
|---------------------|-----------|-----------------|-------|
| 1057777525441679361 | true | true | 1 |
| 1057777569389436928 | true | true | 1 |
| 1057777296055119872 | true | true | 1 |
| 1057777564482240516 | true | true | 1 |
| 1057777322135191553 | true | true | 1 |
| 1057777304359686144 | true | true | 1 |
| 1057777353340928000 | true | true | 1 |
| 1057777303285989376 | true | true | 1 |

Query 7 Count how many tweets were created at the same time

```
//7
val createdTweets = sqlContext
  .sql( sqlText = "SELECT created_at, count(*) as count FROM EntertainmentTable where created_at is not null group by created_at order by count desc limit 15")
createdTweets.show
```

| created_at | count |
|----------------------|-------|
| Wed Oct 31 23:32:... | 24 |
| Wed Oct 31 23:33:... | 23 |
| Wed Oct 31 23:33:... | 21 |
| Wed Oct 31 23:32:... | 21 |
| Wed Oct 31 23:33:... | 20 |
| Wed Oct 31 23:32:... | 20 |
| Wed Oct 31 23:32:... | 19 |
| Wed Oct 31 23:33:... | 19 |
| Wed Oct 31 23:32:... | 19 |
| Wed Oct 31 23:32:... | 18 |
| Wed Oct 31 23:32:... | 18 |
| Wed Oct 31 23:32:... | 18 |
| Wed Oct 31 23:32:... | 18 |
| Wed Oct 31 23:32:... | 17 |
| Wed Oct 31 23:33:... | 17 |

Query 8 Find the max replies a tweet had at a certain time

```
//8
val replyCountTweets = sqlContext
  .sql( sqlText = "SELECT created_at, max(reply_count) as max_reply FROM EntertainmentTable where created_at is not null group by created_at order by max_reply desc")
replyCountTweets.show
```

| created_at | max_reply |
|----------------------|-----------|
| Wed Oct 31 23:32:... | 0 |
| Wed Oct 31 23:32:... | 0 |
| Wed Oct 31 23:32:... | 0 |
| Wed Oct 31 23:33:... | 0 |
| Wed Oct 31 23:32:... | 0 |
| Wed Oct 31 23:32:... | 0 |
| Wed Oct 31 23:32:... | 0 |
| Wed Oct 31 23:32:... | 0 |
| Wed Oct 31 23:32:... | 0 |
| Wed Oct 31 23:32:... | 0 |
| Wed Oct 31 23:33:... | 0 |
| Wed Oct 31 23:33:... | 0 |
| Wed Oct 31 23:32:... | 0 |
| Wed Oct 31 23:32:... | 0 |
| Wed Oct 31 23:33:... | 0 |
| Wed Oct 31 23:32:... | 0 |

Query 9 Find the max quotes a tweet had at a certain time

```
//9
val quoteCountTweets = sqlContext
  .sql( sqlText = "SELECT created_at, max(quote_count) as max_quote FROM EntertainmentTable where created_at is not null group by created_at order by max_quote desc")
quoteCountTweets.show
```

| created_at | max_quote |
|----------------------|-----------|
| Wed Oct 31 23:32:... | 0 |
| Wed Oct 31 23:32:... | 0 |
| Wed Oct 31 23:32:... | 0 |
| Wed Oct 31 23:32:... | 0 |
| Wed Oct 31 23:33:... | 0 |
| Wed Oct 31 23:32:... | 0 |
| Wed Oct 31 23:32:... | 0 |
| Wed Oct 31 23:32:... | 0 |
| Wed Oct 31 23:32:... | 0 |
| Wed Oct 31 23:33:... | 0 |
| Wed Oct 31 23:33:... | 0 |
| Wed Oct 31 23:32:... | 0 |
| Wed Oct 31 23:32:... | 0 |
| Wed Oct 31 23:33:... | 0 |
| Wed Oct 31 23:32:... | 0 |
| Wed Oct 31 23:32:... | 0 |
| Wed Oct 31 23:33:... | 0 |
| Wed Oct 31 23:32:... | 0 |

Query 10 Count how many users are protected and those that are not

```
//10
val protectedTweets = sqlContext
  .sql( sqlText = "SELECT user.protected, count(*) as count FROM EntertainmentTable where user.protected = false group by user.protected order by count desc limit 10")
protectedTweets.show
```

| protected | count |
|-----------|-------|
| false | 1000 |

Increment 2:

I needed the proper libraries in the build.sbt and the proper versions to go along with them.

```
"org.apache.spark" %% "spark-core" % "2.3.1",
"org.apache.spark" %% "spark-sql" % "2.3.1",
"org.apache.spark" %% "spark-mllib" % "2.3.1"
```

Then I had to import the MLlib libraries I was going to be using.

```
import org.apache.spark.ml.Pipeline
import org.apache.spark.ml.evaluation.RegressionEvaluator
import org.apache.spark.ml.feature.{StringIndexer, VectorAssembler}
import org.apache.spark.ml.regression.GBTRegressor
```

The JSON file containing the tweets was read and stored as a dataframe. This is not the one used to create the model.

```
val df = spark.read.format( source = "json").option("header", "true")
  .load( path = "C:\\Users\\steph\\Downloads\\SparkDataframe1\\SparkDataframe\\src\\main\\scala\\1k.json")
```

A subset of the columns are selected and only the rows that did not have null values are included. This is then stored as a dataframe and will be the dataframe used to create the model.


```

val child = df
  .select( col = "favorite_count", cols = "id", "in_reply_to_screen_name", "is_quote_status", "reply_count", "retweeted", "text", "lang")
  .where( conditionExpr = "in_reply_to_screen_name is not null " +
    "and text is not null " +
    "and favorite_count is not null " +
    "and id is not null " +
    "and is_quote_status is not null " +
    "and reply_count is not null " +
    "and retweeted is not null " +
    "and text is not null" )

```

The DataFrame is randomly split into the training and test data. The training data will be used to train the model through machine learning. The test data will be used to test to see how accurate the model is when predicting.

```

//We'll split the set into training and test data
val Array(trainingData, testData) = child.randomSplit(Array(0.8, 0.2))

val labelColumn = "id"

//We define two StringIndexers for the categorical variables

val countryIndexer = new StringIndexer()
  .setInputCol("lang")
  .setOutputCol("replyIndex")

//We define the assembler to collect the columns into a new column with a single vector - "features"
val assembler = new VectorAssembler()
  .setInputCols(Array("reply_count", "replyIndex"))
  .setOutputCol("features")

//For the regression we'll use the Gradient-boosted tree estimator
val gbt = new GBTRegressor()
  .setLabelCol(labelColumn)
  .setFeaturesCol("features")
  .setPredictionCol("Predicted " + labelColumn)
  .setMaxIter(50).setMaxBins(100)

```

The indexer, assembler, and tree for the pipeline are created and placed in an array.

```

//We define the Array with the stages of the pipeline
val stages = Array(
  countryIndexer,
  assembler,
  gbt
)

```

The array is then used to construct the Pipeline. The Pipeline will mainly do the machine learning. The training data from before is then fitted into the Pipeline. When it is being fitted, this is the machine learning part because the Pipeline uses this data and tries to fit what it has to it. In the end, the result is the model. This model is then tested. The test data from earlier is applied to the model to get predictions as to where it fits in the model.

```
//Construct the pipeline
val pipeline = new Pipeline().setStages(stages)

//We fit our DataFrame into the pipeline to generate a model
val model = pipeline.fit(trainingData)

//We'll make predictions using the model and the test data
val predictions = model.transform(testData)
predictions.show()
```

Below is the output of the predictions.

| t | id in_reply_to_screen_name is_quote_status reply_count retweeted | text lang replyindex | features | Predicted id |
|-----------------------|------------------------------------------------------------------|----------------------|--------------------------------------|--------------|
| 0 1057777287033171975 | VonnieCalland false 0 false @VonnieCalland Th... | en | 0.0 (2, [], []) 1.057777426920863... | |
| 0 1057777351533228032 | molly_knight false 0 false @molly_knight My ... | en | 0.0 (2, [], []) 1.057777426920863... | |
| 0 1057777359221374982 | WhenWeAllVote false 0 false @WhenWeAllVote @M... | en | 0.0 (2, [], []) 1.057777426920863... | |
| 0 1057777379047690240 | realDonaldTrump false 0 false @realDonaldTrump ... | en | 0.0 (2, [], []) 1.057777426920863... | |
| 0 1057777436409163776 | Twitter true 0 false @Twitter u r not ... | en | 0.0 (2, [], []) 1.057777426920863... | |
| 0 1057777441501052928 | littlefonty false 0 false @littlefonty @stay... | en | 0.0 (2, [], []) 1.057777426920863... | |
| 0 1057777461021290497 | JulieAnnLily false 0 false @JulieAnnLily @Le... | en | 0.0 (2, [], []) 1.057777426920863... | |
| 0 1057777534924918786 | Need2Impeach false 0 false @Need2Impeach @Da... | en | 0.0 (2, [], []) 1.057777426920863... | |
| 0 1057777544827551744 | Attractivepup false 0 false @Attractivepup He... | en | 0.0 (2, [], []) 1.057777426920863... | |
| 0 1057777553312632832 | tomezine false 0 false @tomezine @Verita... | en | 0.0 (2, [], []) 1.057777426920863... | |
| 0 1057777572908433408 | KevinMKruse false 0 false @KevinMKruse My f... | en | 0.0 (2, [], []) 1.057777426920863... | |

Now that predictions have been made, the testing isn't done. The Regression Evaluator is used to test how accurate the predictions are. This is essentially testing to see how accurate the model itself is.

```
//This will evaluate the error/deviation of the regression using the Root Mean Squared deviation
val evaluator = new RegressionEvaluator()
    .setLabelCol(labelColumn)
    .setPredictionCol("Predicted " + labelColumn)
    .setMetricName("rmse")

//We compute the error using the evaluator
val error = evaluator.evaluate(predictions)

println("The Root Mean Square Deviation error: " + error + "\n")
```

The below value is the output and how accurate the model is. As you can see, it is not really accurate.

```
The Root Mean Square Deviation error: 9.363686403376337E10
```

Next I began to apply graphFrames. In the end, my graph was not correct but it did was technically a graph. It was just not a graph where any meaningful data can be gathered from.

First step was editing the build.sbt to include graphx and the graphframe libraries.

```
"org.apache.spark" %% "spark-graphx" % "2.1.0",
"graphframes" % "graphframes" % "0.5.0-spark2.1-s_2.11"
```

Then I imported graphFrames.


```
}import org.graphframes.GraphFrame
```

First step was reading the tweets from the json file.

```
val df = spark.read.format( source = "json").option("header","true")  
  .load( path = "C:\\Users\\steph\\Downloads\\SparkDataframe1\\SparkDataframe\\src\\main\\scala\\lk.json")
```

Then I created the vertices dataframe. This dataframe is the id, the user's name, and I should have used the user's id and not the tweet id. That was a mistake.

```
val verticesTweets = df  
  .select( col = "id", cols = "user.screen_name" )  
  .where( conditionExpr = "user.screen_name is not null " +  
    "and id is not null " )
```

Next was the edges dataframe. I had difficulty renaming the columns so I renamed each individually. It is less efficient and produces more lines of code, but it works.

```
val edgesPrototype = df  
  .select( col = ("id"), cols = "in_reply_to_screen_name", "in_reply_to_user_id" )  
  .where( conditionExpr = "in_reply_to_screen_name is not null " +  
    "and in_reply_to_user_id is not null " +  
    "and id is not null " )  
  
val e = edgesPrototype.withColumnRenamed( existingName = "id", newName = "src")  
val ed = e.withColumnRenamed( existingName = "in_reply_to_screen_name", newName = "dst")  
val edgesTweets = ed.withColumnRenamed( existingName = "in_reply_to_user_id", newName = "relationship")
```

I chose columns related to replies. I wanted to continue to look at communication between users. The vertices should have been the user's id, name, and possibly later the text of the tweet to see if a user is mentioned in a tweet. The edges should have been the user's id as src, the screen name of the user they were replying to as dst, and the tweet id might be relationship. I am still thinking about relationship. Above it is the user who was being replied to's id.

Next I created the graph from these two dataframes.

```
val tweetGraph = GraphFrame(verticesTweets,edgesTweets)
```

The Triangle Count algorithm was applied to the graph.

```
val triCount = tweetGraph.triangleCount.run()  
triCount.select( col = "id", cols = "count").show()
```

Below is the output.

```

+-----+-----+
|          id|count|
+-----+-----+
|1057777280309706752|    0|
|1057777336354041856|    0|
|1057777348135829504|    0|
|1057777441551409152|    0|
|1057777569389436928|    0|
|1057777361406455809|    0|
|1057777400145174529|    0|
|1057777419015208960|    0|
|1057777480717819904|    0|
|1057777283572805632|    0|
|1057777324941369344|    0|
|1057777335825506305|    0|
|1057777386945691648|    0|
|1057777468797583360|    0|
|1057777536250400768|    0|
|1057777562359906304|    0|
|1057777567019778048|    0|
|1057777280079065088|    0|
|1057777357656899584|    0|
|1057777409607589888|    0|
+-----+-----+

```

Then the sorted Out Degrees of the nodes.

```

println("Out Degrees: ")
tweetGraph.outDegrees.sort ( sortCol = "outDegree" ).show()

```

```

+-----+-----+
|          id|outDegree|
+-----+-----+
|1057777441551409152|      1|
|1057777419015208960|      1|
|1057777283572805632|      1|
|1057777338564440064|      1|
|1057777395900526592|      1|
|1057777384663977984|      1|
|1057777482651201536|      1|
|1057777436409163776|      1|
|1057777401244106753|      1|
|1057777461021290497|      1|
|1057777467455234048|      1|
|1057777286970142720|      1|
|1057777288433958912|      1|
|1057777372068483074|      1|
|1057777557553250306|      1|
|1057777304359686144|      1|
|1057777546421587968|      1|
|1057777543674322946|      1|
|1057777379865755648|      1|
|1057777351533228032|      1|
+-----+-----+

```

And lastly the sorted In Degrees of the nodes.

```

println("In Degrees: ")
tweetGraph.inDegrees.sort ( sortCol = "inDegree" ).show()

```

| id | inDegree |
|-----------------|----------|
| Annaleen | 1 |
| adnilxa | 1 |
| GraceOM1967 | 1 |
| kzannarbor | 1 |
| FlatEarthGang | 1 |
| GROGParty | 1 |
| MakedaIsRight | 1 |
| TrollTerrific | 1 |
| sallyacb275 | 1 |
| schneiderleonid | 1 |
| LouDobbs | 1 |
| jjbittenbinders | 1 |
| natvanlis | 1 |
| AyyBates | 1 |
| WhenWeAllVote | 1 |
| _BenMonroe_ | 1 |
| grantwarkentin | 1 |
| SoulSolaris23 | 1 |
| molly_knight | 1 |
| HawaiianTrash_ | 1 |

Conclusion

The project results could be better, but there is always room for improvement. That's why technology is always advancing. This was my first time programming in a language like scala and doing this type of work. I think what was produced is a good start.

Future Work

If I were to continue this project, I would nearly rewrite my entire graph. I do not feel confident enough with it. The results were not meaningful so I would try to find a better way to define the relationship and create the edges. If I can manage to get it to have meaningful connections, then it will hopefully produce meaningful data. Gaining better data to work with would also be on my list of things to do if I were to continue this. My dataset wasn't diverse enough, that can be clearly seen by looking at the results of my queries. If I had better data I may have gained more insight into twitter and had a more accurate analysis. The improved data may assist in making my machine learning model more accurate with it's predictions. It probably needed more data to train with than I gave it.

Project Management - All Team Members

Worked completed

Description:

- Twitter streaming script
- Collect 100K for each member of the team
- Perform 10 interesting queries with the data we collect
- Machine learning model creation with Spark
- GraphFrames with Spark

Responsibility (Task, Person):

- Yong
 - Wrote twitter streaming script
 - Collected 100K tweets
 - Performed 10 interesting queries
 - Wrote a machine learning model with Spark
 - Wrote couple graph algorithm in GraphFrames with the graph I created
- Stephanie
 - Collected tweets
 - Performed 10 queries
 - Wrote a machine learning model with Spark
 - Created a graph with GraphFrames and applied a few algorithms
- Shaun
 - Collected 100k Tweets
 - Loaded into Hive and perform HQL Analysis
 - Load into Spark and get into Graphframe
- Harshil
 - Collected tweets
 - Performed 10 queries
 - Wrote a machine learning models with Spark
 - Created a graph with GraphFrames and applied a few algorithms

Contributions (members/percentages):

- Each member of the team is assigned with the extra same task. The contribution for each member of the team will be 25% if they finished their assign tasks.

Work to be completed

Description:

None

Responsibility (Task, Person):

None

Issues/Concerns:

None

References/Bibliography

None