Project Title:

Tweets Analysis with Spark

Team:

Code Monkeys

Team members:

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

Increment 2 Report - Yong

Introduction

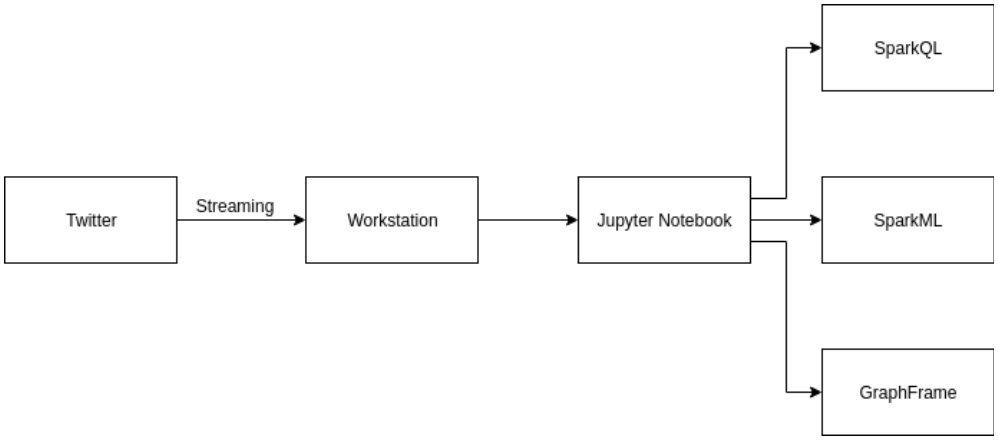
Twitter has a lot interesting data and tweets that are easy to collect. We applied some SQL queries on the data we collected for increment 1. For increment 2, we will build machine learning model on the Twitter data we collected. Also, we apply graph algorithm to the dataset we collected.

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 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

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 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

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 & Result

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 = fpGrowth.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(tdruncate=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)
```

I reasons behind the edges and vertices creation are explained 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       |
|9481710938183434 25|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 form 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
```

Increment 2 Report - Harshil

Introduction

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.

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

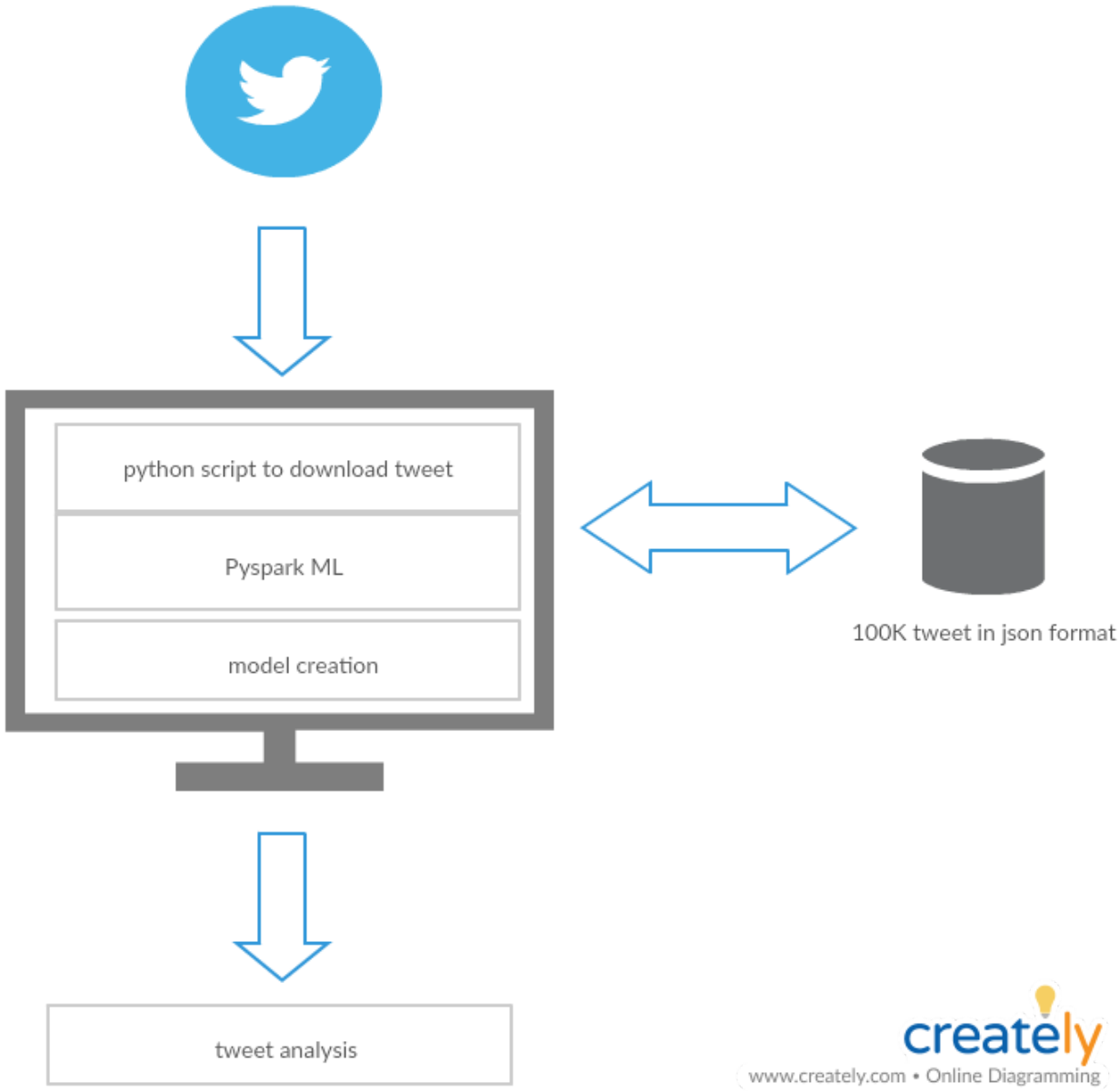
I learned that TF-IDF with Logistic Regression is quite strong combination, and showed robust performance, as high as Word2Vec + Convolutional Neural Network model. So in this post, I will try to implement TF-IDF + Logistic Regression model with Pyspark.

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|
+---+--------------------------------------------------------------------------------------------------------------------------------------------------------+------+
|0  |RT Trump�s press secretary who frequently calls accurate but critical reporting �fake  is sharing doctored video produced b�                          |1     |
|1  |RT What is tweeting "Horseface from the residence about a porn star you allegedly slept with https://t.co/aAWcqaBwMw                                    |0     |
|2  |RT Go to a hand recount in each county Supervise every vote count every vote &amp;amp accept the results. Any other process wi�                        |0     |
|3  |RT I�m moving to Budapest later on this month and I know exactly what kind of country�                                                                  |1     |
|4  |RT Little-known fact There are currently more former car salesmen on the House Science Committee than Ph.D scientists. Littl�                           |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

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 than identified feature and than 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 frequecy for IDF (Inverse Document Freqeuncy) 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
```

Increment 2 Report - Shaun

Introduction

I realised 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

I used Rstudio and the twitteR 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

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

Dataset

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

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.
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.

Increment 2 Report - Stephanie

Introduction

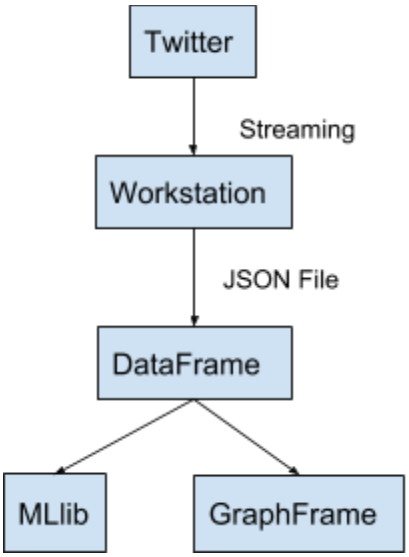
The tweets dataset now will have machine learning and graphs applied to it. A model is created and trained using the tweets dataset. Then it is tested to see how accurate it is when predicting.

Background

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, performed breadth first search, and counted the number of triangles in the graph.

Model



I streamed the tweets from twitter. These tweets were saved as a JSON file. Spark and Scala have something called DataFrames. I could create a DataFrame containing the columns and tweets I needed for the machine learning. Then I performed the machine learning on it. 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. 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 and Results

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 @tay...|  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\\1k.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

I did not include the Breadth First Search Algorithm because I want to correct the graph before I start traversing it.

Project Management - All Team Members

Worked completed

Description:

- Machine learning model creation with Spark
- GraphFrames with Spark

Responsibility (Task, Person):

- Yong
    - Wrote a machine learning model with Spark
    - Wrote couple graph algorithm in GraphFrames with the graph I created
- Stephanie
    - Created a model with machine learning in Spark using MLlibs

    - Created a graphical representation of the Tweets but it requires adjustment
- Shaun
    - Got the data into spark graphs
    - Created edges and vertices
- Harshil
    - Wrote python script to clean data
    - Wrote a machine learning model with Spark

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):

- Harshil
    - Apply graph algorithms with Spark GraphFrames
- Shaun
    - Create a model with SparkML
    - Apply graph algorithms with Spark GraphFrames

Issues/Concerns:

None

References/Bibliography

Stephanie:

https://blog.scalac.io/scala-spark-ml.html

https://spark.apache.org/docs/2.3.0/ml-pipeline.html