CSYE 7374:
Big-Data Systems and Intelligent Analytics

Twitter based Movie Ratings

Final Project Executive Summary

Team 4

- Samir Sharan
- Harshit Shah
- Jeevan Reddy

Submitted on: 08/22/2015

Problem Statement

Twitter has become one of the most important data source in recent times. Twitter data can provide very deep insights and can be used for multiple purposes for eg, to determine user activeness and engagement, demographic analysis and sentimental analysis among others.

We came up with the idea to provide twitter based movie ratings by doing sentimental analysis of the tweets for recently released movies. This will help the moviegoers decide which movie to see based on twitter user's sentiment about the movies.

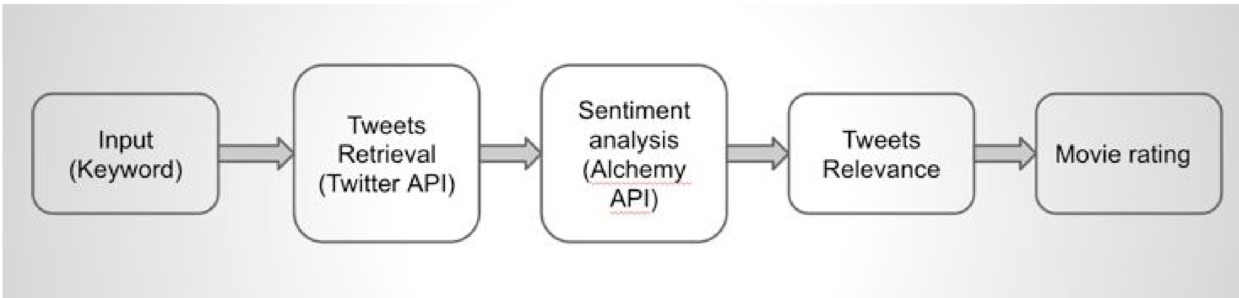
The reason behind doing twitter-based movie ratings is that every ratings related website or agency will have their own critics which will review the movie and give their personal rating. And obviously, personal choice of one critic or a couple of them should not be penultimate. Hence, we decided to take in reviews from the people who have already seen the movie. The only other way was to stand outside a movie theatre and take reviews from moviegoers, which is obviously not feasible. Hence, twitter.

There are some fixed components which we need to get ready in order to complete our deliverable.

I. Set up Twitter Credentials
II. Set up Alchemy API account
III. Set up Amazon Web Services account

WorkFlow

We follow the below workflow to reach to our final goal of finding the Rating.

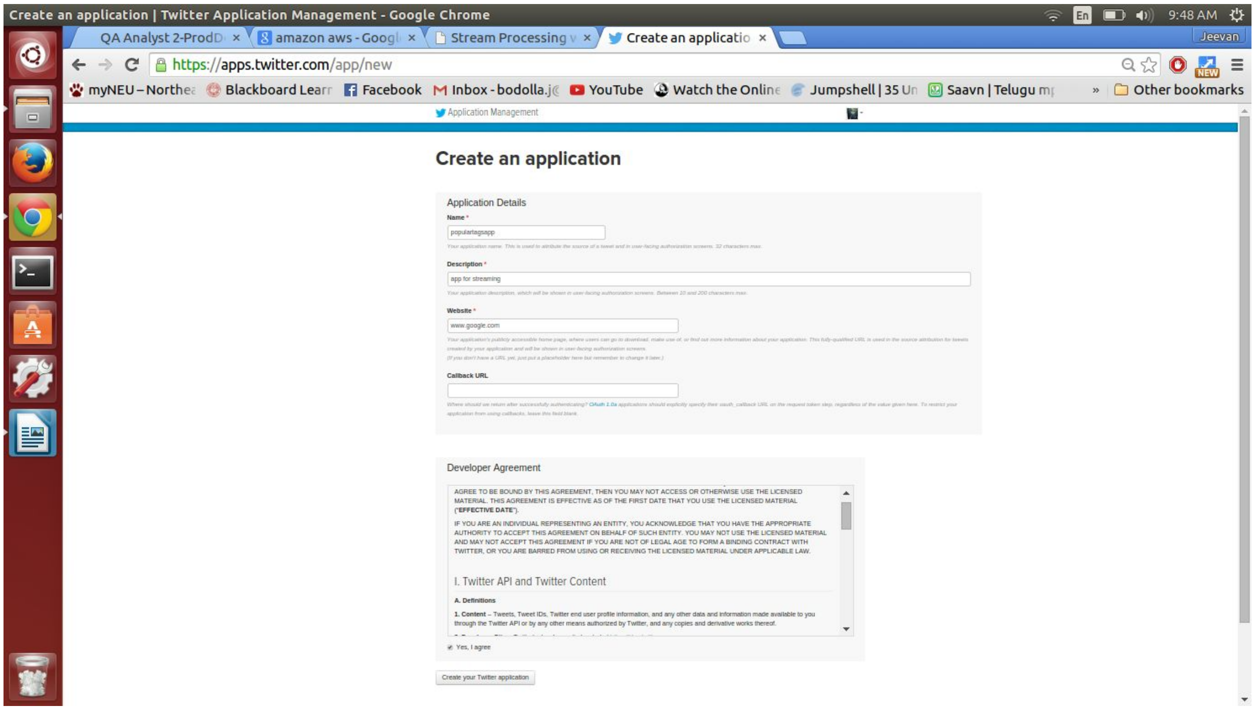


Note: AlchemyAPI has a limit of 1000 tweets per day per account.

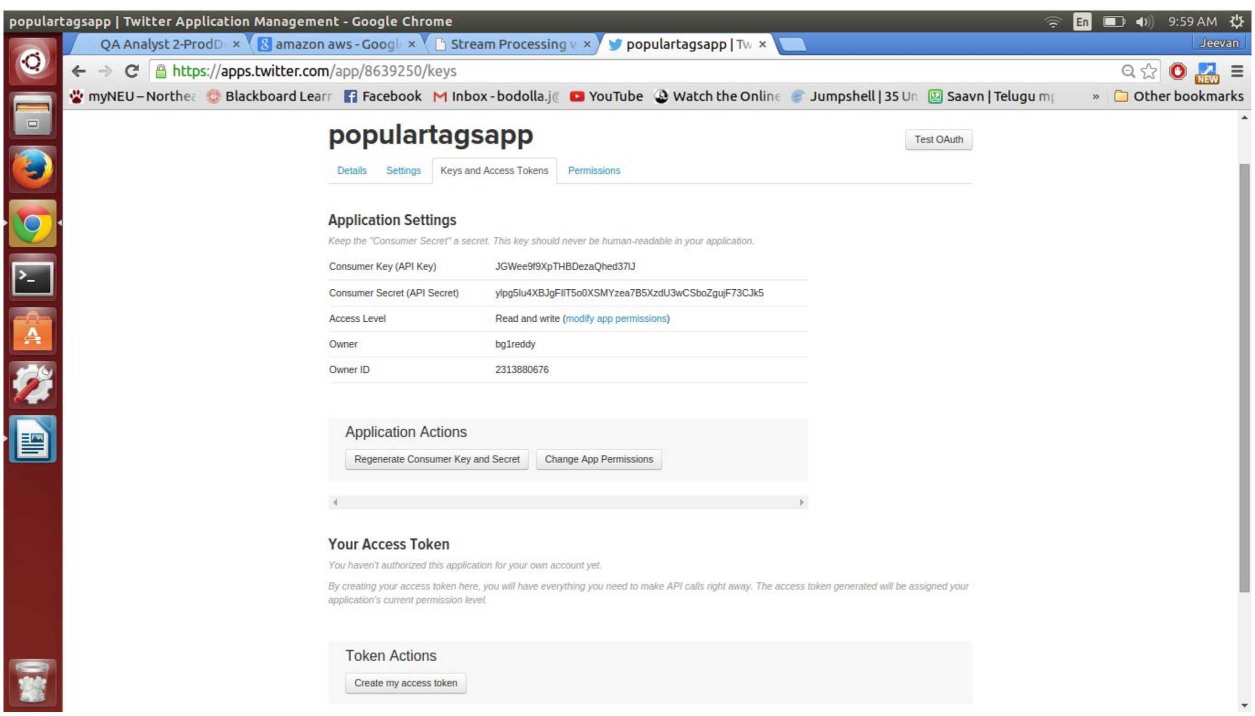
1) Twitter Credential Setup:

Since the whole exercise is based on twitter, it is necessary to configure authentication with a twitter account using a consumer key+secret pair and an access token+secret pair.

1. Open the twitter's application settings page (https://apps.twitter.com/) and create a new application by clicking on the "Create a new application" button and providing the required fields.



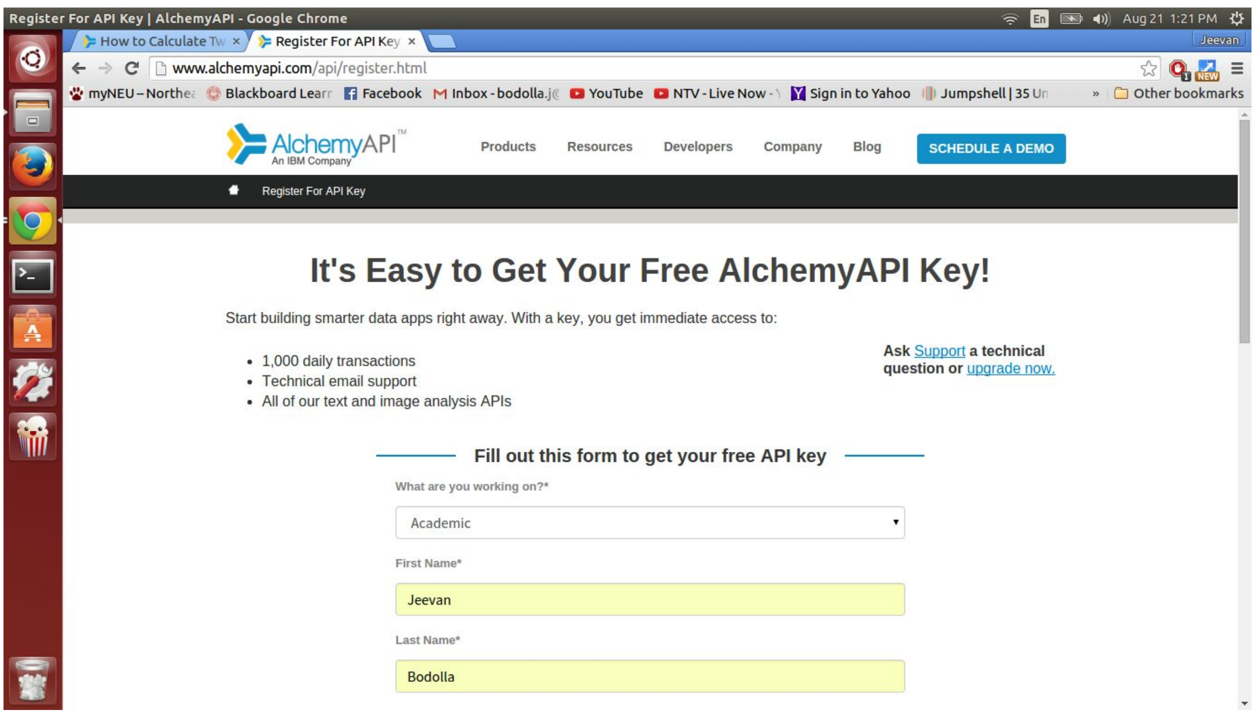
2. Once you create the application, click on the "Keys and Access Tokens", and you will come to a page that looks similar to this:



3. Copy/Paste the “Consumer Key” and “Consumer Secret” into a safe location, or just come back to this part when you need them in the later steps.

2) Getting API key from AlchemyAPI:

AlchemyAPI requires an API key to be included in each API transaction. Get the API key at http://www.alchemyapi.com/api/register.html by registering for a free one under academic.



3) Clone the Application from GitHub:

Now, it’s time to get the application code. The application source code is hosted on the social coding site, GitHub, at https://github.com/AlchemyAPI/alchemyapi-recipes-twitter. Get the source code onto your local machine by cloning it.

To clone the AlchemyAPI-Twitter-Python application source code to your computer, open the terminal window and type the following commands:

```
mkdir -p ~/src/recipes
cd ~/src/recipes/
git clone https://github.com/AlchemyAPI/alchemyapi-recipes-twitter.git
cd alchemyapi-recipes-twitter/
```

If you are using windows, just replace ~/src/recipes with something like C:\src\recipes\ and it should work.

If you don't have a Git on your machine, you can download a .zip file of the application code from GitHub instead. Just go to: https://github.com/AlchemyAPI/alchemyapi-recipes-twitter and click the "Download ZIP" button on the right sidebar.

4) Configure the Application:

Now that that we have the source code on your machine, we have to configure the recipe before we can run the application.

Prepare Twitter and AlchemyAPI Credentials:

Configure your credentials for this recipe by editing the credentials.py file. After cloning from Github, the contents of this file will look like this:

```
twitter_consumer_key='YOUR_TWITTER_CONSUMER_KEY'
twitter_consumer_secret='YOUR_TWITTER_CONSUMER_SECRET'
alchemy_apikey='YOUR_ALCHEMY_API_KEY'
```

Replace YOUR_TWITTER_CONSUMER_KEY and YOUR_TWITTER_CONSUMER_SECRET with your Twitter consumer key and consumer secret from Step 1 above. Similarly, replace YOUR_ALCHEMY_API_KEY with your AlchemyAPI key as in Step 2. If you choose to not use this file, the recipe will prompt you for your credentials.

Modify the recipe.py file:

Changes should be made to the recipe.py file to configure the fileds you are interested.

```
#Fields we are interested in from the status object
tweet['movie']          = search_term
tweet['location']       = status['geo']
```

```
tweet['coordinates']            = status['coordinates']
tweet['RT_count']               = status['retweet_count']
tweet['fav_count']              = status['favorite_count']
tweet['language']        = status['lang']
tweet['user_followers_cnt']     = status['user']['followers_count']
tweet['verified']               = status['user']['verified']
```

Also, edit the store function in the recipe.py file to store the tweets from the application in json instead of mongodb as shown below. As we want to export the data to S3 bucket and access it from there:

```
def store(tweets):
        with open(JSONFILENAME.json', 'w') as fp:
                for tweet in tweets:
                        a = json.dump(tweet, fp)
                        fp.write("\n")
        return
```

5) Run the Application:

We can now run some sentimental analysis with Twitter data! To run the application, type as below:

```
python recipe.py "SEARCH_STRING" COUNT
```

where SEARCH_STRING is what you want to search Twitter for (in our case, movie name), and COUNT is how many Tweets you want to attempt to retrieve.

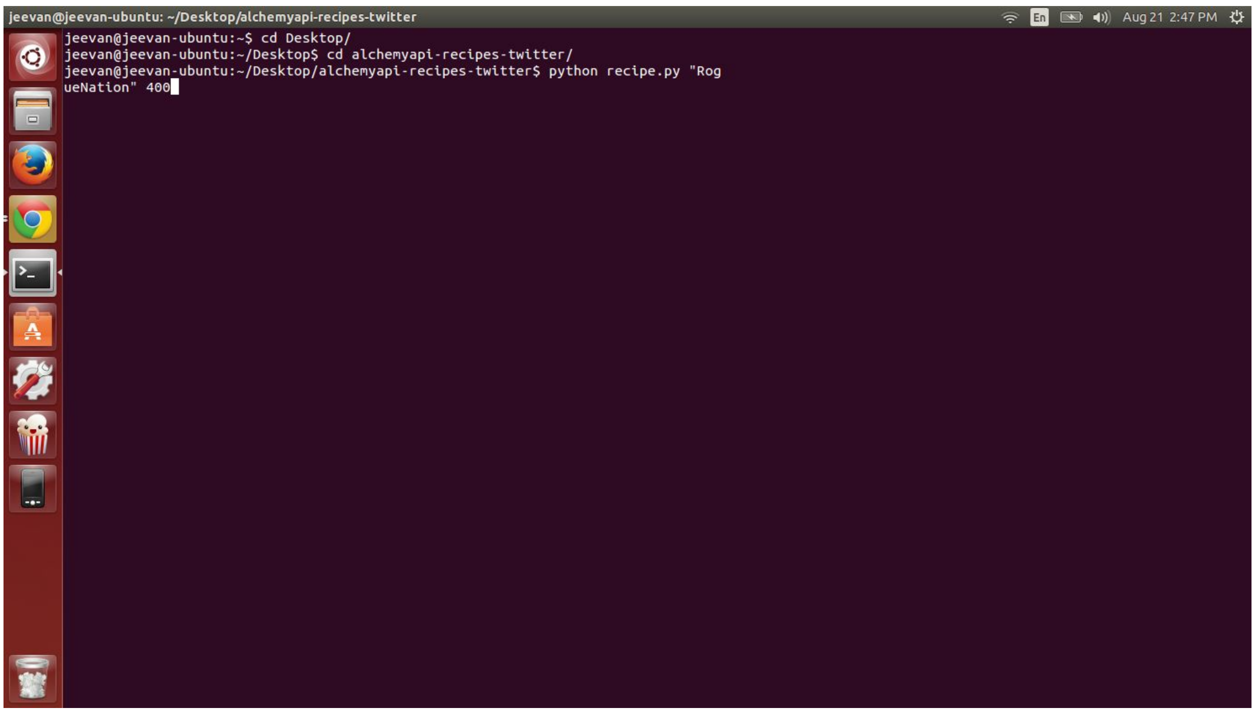
The recipe.py establishes credentials for the Twitter and AlchemyAPI and pulls Tweets from the Twitter API after de-duplicating the tweets by ID.

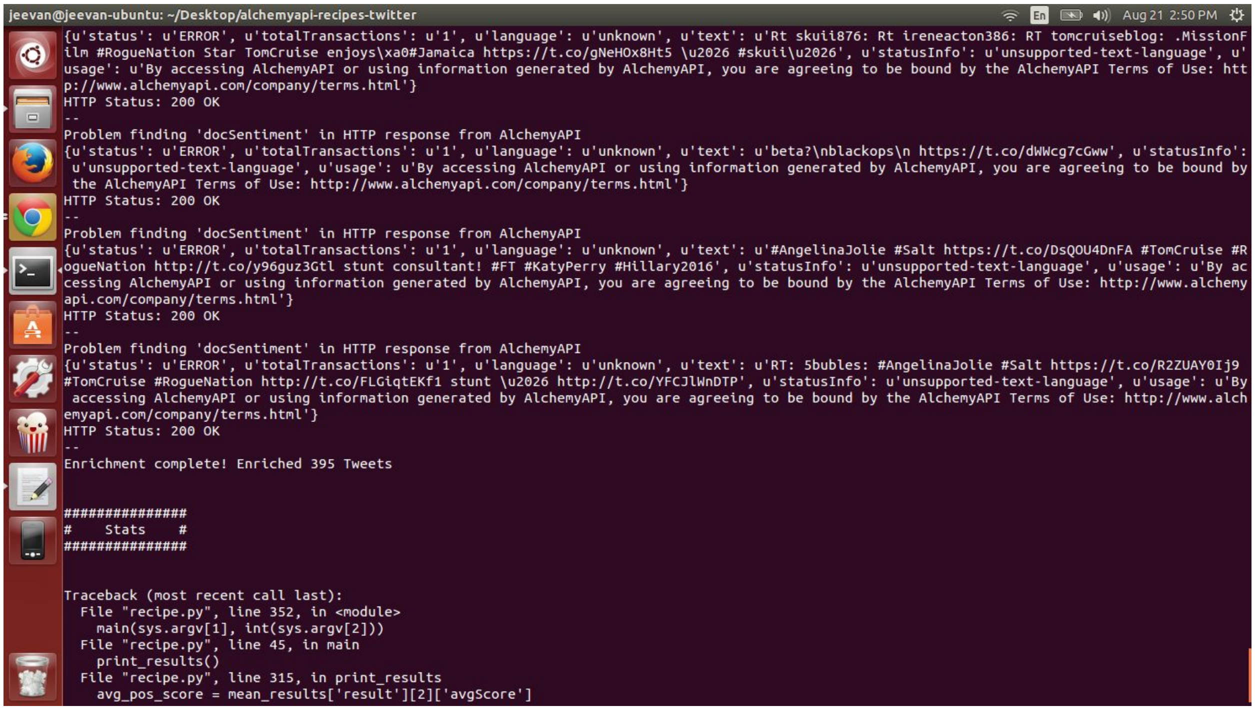
```
# Establish credentials for Twitter and AlchemyAPI
credentials = get_credentials()

# Get the Twitter bearer token
auth = oauth(credentials)

# Pull Tweets down from the Twitter API
raw_tweets = search(search_term, num_tweets, auth)

# De-duplicate Tweets by ID
```

```
unique_tweets = dedup(raw_tweets)
```

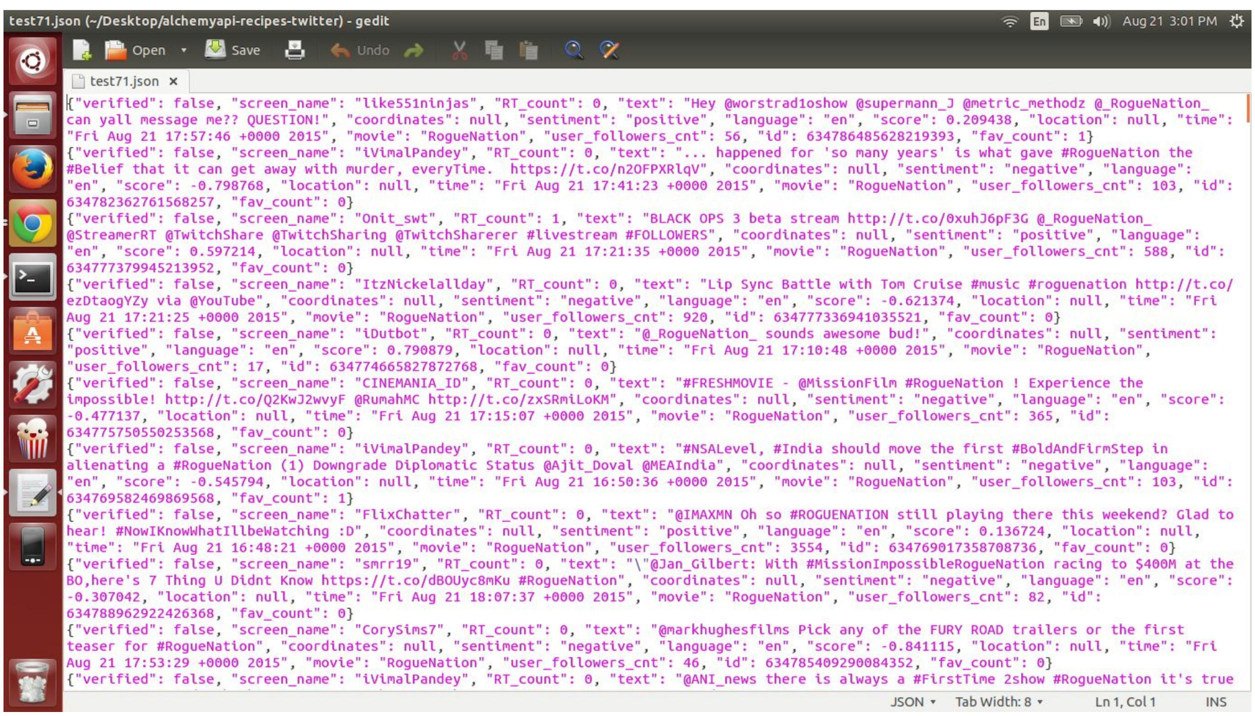
The search function looks for the given SEARCH_STRING and returns a collection of tweets with all the fields you are interested in from the Twitter by filtering out retweets. Once the tweets are enriched with the sentiments by using AlchemyAPI, the store function saves the tweets with the sentiments in a .json file.

Running the application will take a little while depending on how many Tweets you are analyzing, and it will produce lots of output to the terminal window. The output would be somewhat similar as below:





Once the application is run, you can see a .json file created in the "**alchemyapi-recipes-twitter"** folder containing all the tweets with the fields we are interested as shown below :



Collect Tweets using tweet_script.sh that runs the recipe.py script for all the movies and collects 100 tweets/movie and concatenates them to data.json(train data) file.
This script is triggered using a cronjob that runs every 15 mins. We scheduled another cronjob that transfers the data to Amazon S3 at Friday 8pm after we collected all the tweets.

CronJob:

Cronjob can be scheduled by using by typing crontab -e command on the terminal where u want to run the script:

```
!/Users/insignia/anaconda/bin/python
*/15 * * * * /Users/insignia/anaconda/bin/python /Users/insignia/Desktop/Big_Data_Analytics/alchemy/tweet_script.sh
* 20 * * 5 s3cmd sync /Users/insignia/Desktop/Big_Data_Analytics/alchemy/a/ s3://bigdataanalyticscourse/Json_input_train/
~
~
```

The first scheduler:
*/15 * * * * /Users/insignia/anaconda/bin/python
/Users/insignia/Desktop/Big_Data_Analytics/alchemy/tweet_script.sh

This command schedules the tweet_script.sh to run every 15 mins.
The second scheduler:
* 20 * * 5 s3cmd sync
/Users/insignia/Desktop/Big_Data_Analytics/alchemy/a/
s3://bigdataanalyticscourse/Json_input_train/

This command triggers the script to pick up the input file from the "/Users/insignia/Desktop/Big_Data_Analytics/alchemy/a/ " folder and put it to the S3 bucket location "s3://bigdataanalyticscourse/Json_input_train/" and it runs every Friday at 8pm.

Tweet_script.sh:

```
  bin/bash
python recipe.py 'SouthPaw' 100
cat test.json >> data.json
python recipe.py 'Minions' 100
cat test.json >> data.json
python recipe.py 'FantasticFour' 100
cat test.json >> data.json
python recipe.py 'RogueNation' 100
cat test.json >> data.json
python recipe.py 'InsideOut' 100
cat test.json >> data.json
```

After transferring the data.json(train data) to Amazon S3 bucket (before storing the data we need to create a bucket in S3), the detail steps regarding creating the bucket has been provided later in the document, this is how it appears in the amazon S3 bucket (bigdataanalyticscourse/Json_input_train):

Name	Storage Class	Size
data.json	Standard	3.1 MB
final3.json	Standard	6.1 MB

The same steps are repeated for the final3.json(test data) to store it in the S3 bucket.

After we have both the data.json(train data) and the final3.json(test data) in the S3 bucket, We then run the python script to find the relevance score of the tweet depending upon retweet count, favorite count, number of followers of the users, whether the user is verified or not, and relevance score will assign the weightage to the tweets.

The relevancescore.py and the relevancescore_test.py is used to collect the relevance score of the train and the test data and generates csv files for the train(train.csv) and the test(test.csv) data. We will use these csv files for prediction.

relevancescore.py description:

```
def relevanceScore_rt(frame):
    rel_rt = 0.3
    if frame <= 50:
        rel_rt = (rel_rt/3)
    elif (frame > 50) and (frame <= 500):
        rel_rt = (((rel_rt)*2)/3)
    else:
        rel_rt
    return rel_rt

def relevanceScore_fol(frame):
    rel_fol = 0.3
    if frame <= 500:
        rel_fol = (rel_fol/3)
    if (frame > 500) and (frame <= 1000):
        rel_fol = (((rel_fol)*2)/3)
    else:
        rel_fol
    return rel_fol

def relevanceScore_fav(frame):
    rel_fav = 0.3
    if frame <= 500:
        rel_fav = (rel_fav/3)
    if (frame > 500) and (frame <= 5000):
        rel_fav = (((rel_fav)*2)/3)
    else:
        rel_fav
    return rel_fav
def relevanceScore_ver(frame):
    rel_ver = 0.1
    if frame == False:
        rel_ver = 0
    else:
        rel_ver
    return rel_ver

def get_dummy(x):
    if x == False:
        x = 0
    elif x == True:
        x = 1
    elif x == 'positive':
        x = 1
    elif x == 'negative':
        x = -1
    elif x == 'neutral':
        x = 0
    return x
```

```python
for x in df_test['verified']:
    df_test['verified_numerical'][m] = get_dummy(x)
    m = m+1

for x in df_test['sentiment']:
    df_test['sentiment_numerical'][n] = get_dummy(x)
    n = n+1

for x in df_test['RT_count']:
    df_test['Rel_score_rt'][i] = relevanceScore_rt(x)
    i=i+1

for x in df_test['user_followers_cnt']:
    df_test['Rel_score_fol'][j] = relevanceScore_fol(x)
    j=j+1

for x in df_test['fav_count']:
    df_test['Rel_score_fav'][k] = relevanceScore_fav(x)
    k=k+1

for x in df_test['verified']:
    df_test['Rel_score_ver'][l] = relevanceScore_ver(x)
    l=l+1

df_test['Rel_Score'] = df_test['Rel_score_rt'] + df_test['Rel_score_fol'] + df_test['Rel_score_fav'] + df_test['Rel_score_ver']
df_test.drop(df_test.columns[[1, 3, 4, 5, 6, 8, 9, 10, 11, 13]], axis=1, inplace=True)

#df.columns
df_test.to_csv("/train.csv", encoding='utf-8', header=False)
```

Steps in relevancescore.py:

1) Read the data.json file
2) Calulate the relevanceScore_rt(Relevance based on Re-tweet), relevanceScore_fol (Relevance based on follower count), relevanceScore_fav (Relevance based on favorite count), relevanceScore_ver (Relevance based on verified or not)
3) Data conversions of the senitment field to dummies
4) Calculate the Rel_Score based on the below formula:

```python
df_test['Rel_Score'] = df_test['Rel_score_rt'] + df_test['Rel_score_fol'] + df_test['Rel_score_fav'] + df_test['Rel_score_ver']
```

5) Generate the train.csv file

relevancescore_test.py script is run on the test data to generate the test.csv file, this script uses the same logic to calculate the rel_score and and generate the csv file.

The relevancescore.py and the relevancescore_test.py are triggered using script.sh and script_test.sh which first uses the wget command to get the json data from the S3 bucket and then uses the respective json data as an input for the relevancescore.py and the relevancescore_test.py respectively to generate the csv files.

script.sh:

```
# bin/bash

sudo wget https://s3.amazonaws.com/bigdataanalyticscourse/Json_input_train/data.json

python /relevancescore.py
~
~
```

script_test.sh:

```
# bin/bash

sudo wget https://s3.amazonaws.com/bigdataanalyticscourse/Json_input_train/final3.json

sudo python /relevancescore_test.py
~
~
~
```

After running the script.sh and the script_test.sh, the train.csv and the test.csv files are generated. We use csv files to predict how accurate the our relevance score is using the predict.py script. Predict.py script uses LinearRegressionWithSGD algorithm to train the data and then calculate the Mean Squared Error.

To run the predict.py script we followed the below mentioned steps:

1. cd /usr/lib/spark-1.4.0-bin-hadoop2.6/bin/
2. ./pyspark /predict.py

predict.py:

```python
from pyspark.sql import *
import os
import requests
from pyspark.mllib.classification import SVMWithSGD, SVMModel, LogisticRegressionWithLBFGS
from pyspark.mllib.regression import LabeledPoint, LassoWithSGD
from pyspark.mllib.regression import LinearRegressionWithSGD, RidgeRegressionWithSGD
from pyspark import SparkContext

sparkHome = os.environ.get('SPARK_HOME')
sc = SparkContext(appName="Classification")

#defining a function to split values
def parsePoint(line):
    values = [float(x) for x in line.split(',')]
    return LabeledPoint(values[11], values[1:10])


#reading csv file into rdd
rdd_train = sc.textFile("/train.csv")
rdd_test = sc.textFile("/test.csv")

parsedData_train = rdd_train.map(parsePoint)
parsedData_test = rdd_test.map(parsePoint)

#run LinearRegression
model = LinearRegressionWithSGD.train(parsedData_train,100,0.00000001)

#Evaluate the model on training data
valuesAndPreds_train = parsedData_train.map(lambda p: (p.label, model.predict(p.features)))
MSE = valuesAndPreds_train.map(lambda (v, p): (v - p)**2).reduce(lambda x, y: x + y) / valuesAndPreds_train.count()
print("Training Mean Squared Error for Linear Regression = " + str(MSE))

#Evaluate the model on testing data
valuesAndPreds_test = parsedData_test.map(lambda p: (p.label, model.predict(p.features)))
MSE = valuesAndPreds_test.map(lambda (v, p): (v - p)**2).reduce(lambda x, y: x + y) / valuesAndPreds_test.count()
print("Testing Mean Squared Error for Linear Regression = " + str(MSE))
```

Output:

Training Mean Squared Error for Linear Regression = 1.62518759338e+64
Testing Mean Squared Error for Linear Regression = 3.23531007137e+64

Train Error:

```
15/08/22 06:10:02 INFO DAGScheduler: ResultStage 105 (count at /predict.py:30) finished in 0.734 s
15/08/22 06:10:02 INFO DAGScheduler: Job 105 finished: count at /predict.py:30, took 0.740799 s
Training Mean Squared Error for Linear Regression = 1.62518759338e+64
15/08/22 06:10:02 INFO FileInputFormat: Total input paths to process : 1
15/08/22 06:10:02 INFO SparkContext: Starting job: reduce at /predict.py:35
```

Test Error:

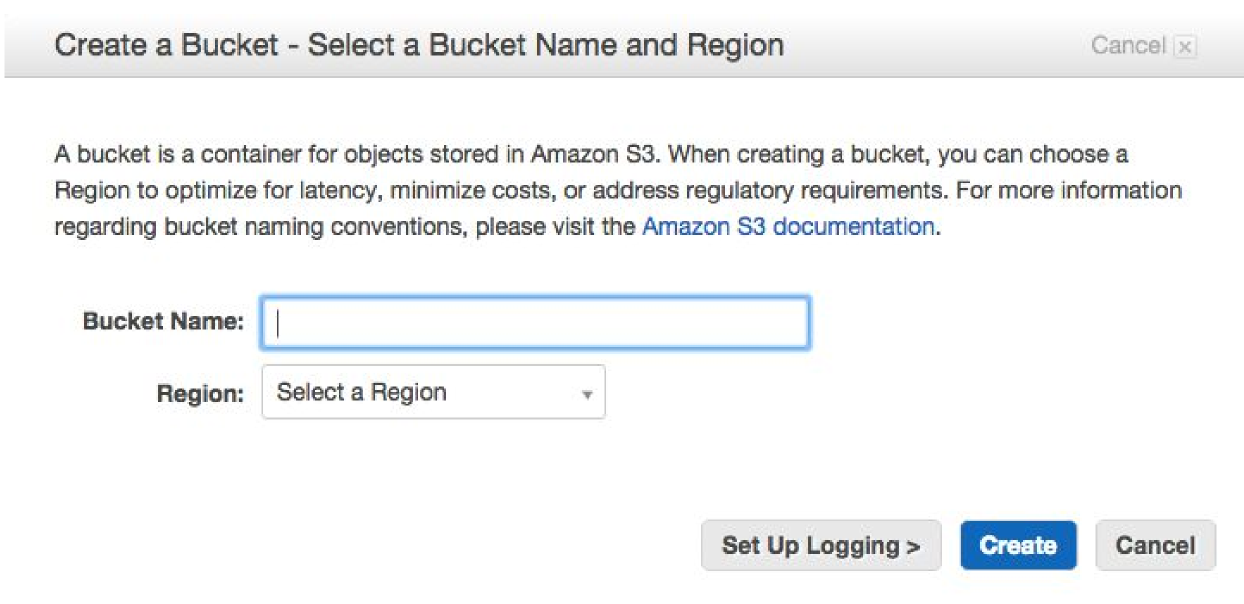
```
15/08/22 06:10:03 INFO DAGScheduler: ResultStage 107 (count at /predict.py:35) finished in 0.258 s
15/08/22 06:10:03 INFO DAGScheduler: Job 107 finished: count at /predict.py:35, took 0.263900 s
Testing Mean Squared Error for Linear Regression = 3.23531007137e+64
15/08/22 06:10:03 INFO SparkContext: Invoking stop() from shutdown hook
```

Twitter Sentiment Analysis: Using AWS

1. Create a Twitter account(the steps have been explained at the top of the document)
2. Create a bucket in S3:

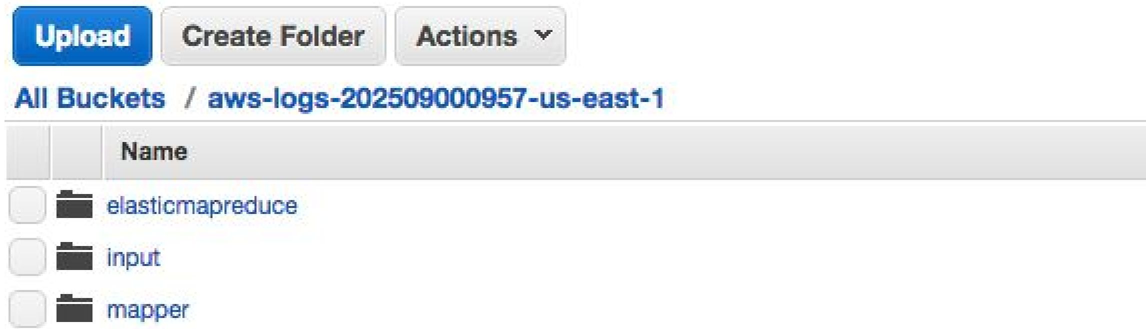
   i) Click on S3

   ii) Click on create bucket:



Enter a bucket name and select region as US Standard

iii) Create input and the mapper folder



The cronjob that picks up the twitter data json file places them in the input folder.

iv) Open your vi editor and save the content , replace the “term1” in red with the name of the movie(SouthPaw,Minions,FantasticFour, InsideOut,RogueNation)

```
#!/usr/bin/python

import cPickle as pickle
import nltk.classify.util
from nltk.classify import NaiveBayesClassifier
from nltk.tokenize import word_tokenize
import sys

sys.stderr.write("Started mapper.\n");


def word_feats(words):
    return dict([(word, True) for word in words])


def subj(subjLine):
    subjgen = subjLine.lower()
    # Replace term1 with your subject term
    subj1 = "term1"
    if subjgen.find(subj1) != -1:
        subject = subj1
        return subject
```

```
    else:
        subject = "No match"
        return subject


def main(argv):
    classifier = pickle.load(open("classifier.p", "rb"))
    for line in sys.stdin:
        tolk_posset = word_tokenize(line.rstrip())
        d = word_feats(tolk_posset)
        subjectFull = subj(line)
        if subjectFull == "No match":
            print "LongValueSum:" + " " + subjectFull + ": " + "\t" + "1"
        else:
            print "LongValueSum:" + " " + subjectFull + ": " +
classifier.classify(d) + "\t" + "1"


if __name__ == "__main__":
    main(sys.argv)
```

Note: For sentimental Analysis using aws we store input files for 1 movie at a time , this movie name is replaced by the term1 above and upload it to the mapper folder in the bucket.

The movie data is collected in the following format:

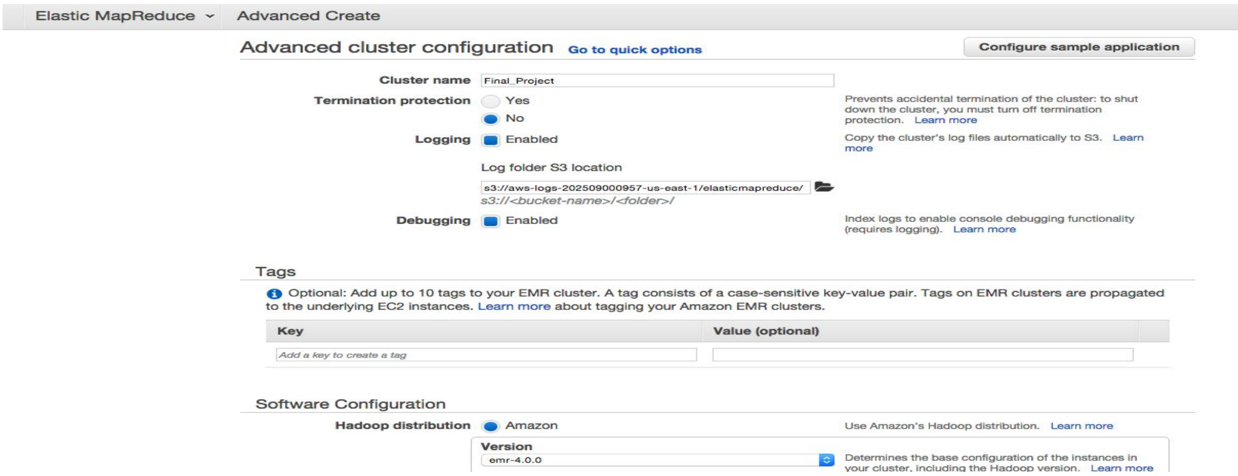
```
id: 633054969629962240, text: Why didn't you see straight outta Compton man im disappointed https://t.co/blSScbLkoG
id: 633054980249939968, text: RT @DepressedDarth: May the minions be with you http://t.co/dquSrAUft2
id: 633054980564455425, text: RT @DepressedDarth: May the minions be with you http://t.co/dquSrAUft2
id: 633054984792227840, text: RT @zaynmalik: Didn't realise the hate for minions was this real fuck these aliens got it bad
id: 633054994321698816, text: Eu odeio os Minions
id: 633055000571310080, text: Arab minions ?\n https://t.co/V0kUBvlDGT via @YouTube
id: 633055010318852097, text: RT @zaynmalik: Didn't realise the hate for minions was this real fuck these aliens got it bad
id: 633055010394361856, text: RT @DepressedDarth: May the minions be with you http://t.co/dquSrAUft2
id: 633055021131689984, text: Do you know Misha? Clev:Misha Collins is a trickster god He is the Great Confuser He is the Overlord of the Minions, and we do his bidding
id: 633055026571689984, text: RT @zaynmalik: Didn't realise the hate for minions was this real fuck these aliens got it bad
id: 633055027289047040, text: Head Back-to-School with the Minions and @MyAvonCanada! WIN a backpack &amp; stationary set on #mmblog #giveaway CAN http://t.co/xmmVQx5XQs
id: 633055029038071808, text: Head Back-to-School with the Minions and @MyAvonCanada! WIN a backpack &amp; stationary set on #mmblog #giveaway CAN http://t.co/0Y0XLLojqv
id: 633055031944720384, text: i just got cut off by someone with a minions bumper sticker
id: 633055040933109760, text: RT @candydrawings: - Los minions son adorables *~*\n- http://t.co/JeD0Myojkd
id: 633055044905005057, text: RT @DepressedDarth: May the minions be with you http://t.co/dquSrAUft2
id: 633055048369610752, text: RT @DepressedDarth: May the minions be with you http://t.co/dquSrAUft2
id: 633055059052494848, text: Fui a ver los minions, si quieren los dejo decirme \"feliz d\u00eda\" me lo merezco
id: 633055060138852352, text: RT @camilacabello97: @alexanderdeleon minions movie soon please
id: 633055062630248448, text: RT @jacksfilms: Really hope you guys have a Minions\u2122 kind of day today
id: 633055064123379712, text: RT @sorrygru: combine the minions movie with paul blart. but instead of animation all the minions are played by Adam sandler except he's a \u2026
id: 633055069114646528, text: I'm at Cin\u00e9polis - @cinepolis for Minions in Monterrey, NLE, Nuevo Le\u00f3n https://t.co/bo7fFKkrzV
id: 633055069949313025, text: Por fin ganamos en el Nacional!!! Cagaron los minions. #VamosLaU
id: 633055070494437376, text: RT @DepressedDarth: May the minions be with you http://t.co/dquSrAUft2
id: 633055088228073476, text: I don't think blurryface likes minions https://t.co/LtsRnrwWrP
id: 633055099842109441, text: Op snapchat merken dat @Annabelll__ en ik allebei in een #minions t shirt slapen \ud83d\ude4a\ud83d\ude4a.\n#DollePret
id: 633055100903096321, text: @ctrlkimani ur bringing up minions ?? .. i Gtg real quick
id: 633055106490048512, text: RT @L4RRYV0DKAS: estaba viendo los minions y no entiendo una chota de lo que dicen pero igual son una ternura
id: 633055111833513988, text: RT @zaynmalik: Didn't realise the hate for minions was this real fuck these aliens got it bad
id: 633055117571264512, text: @BILLYTHEBORG @Elisa_Lo_  #minions
id: 633055127566483456, text: Those minions are awesome! Check out our Daily Recipes for new ideas for meals here --&gt;... http://t.co/xK0Z5UEmBq
id: 633055127805521921, text: RT @hugoisaacmonte: Estamos todos!!!!! Hasta el Minions Mestrista!!!@ramonjmestre @JBeeSellares Mestre Sigue!!! #avancemos http://t.co/q2LL\u2026
id: 633055130795937792, text: Visit @ToughCookieMom to win a $50 @Fandango Gift Card to plan a movie outing! #FandangoFamily #Giveaway #Ad http://t.co/bjjKslVIrq
id: 633055131903234049, text: RT @fascinatingtip: In Despicable Me, the gibberish the minions speak is an actual language called minion-ese.
id: 633055151918608384, text: RT @zaynmalik: I have minions on my jumper and there sick .. #Don'tmesswithminions http://t.co/Dd5gJbbHoo
id: 633055151847309312, text: RT @zaynmalik: Didn't realise the hate for minions was this real fuck these aliens got it bad
id: 633055153055264768, text: RT @candydrawings: - Los minions son adorables *~*\n- http://t.co/JeD0Myojkd
id: 633055156368711680, text: RT @versagermitstiI: gibt es eigentlich auch Minions-Kondome? so mit Bananengeschmack?
id: 633055158805463040, text: RT @zaynmalik: I have minions on my jumper and there sick .. #Don'tmesswithminions http://t.co/Dd5gJbbHoo
id: 633055160458084352, text: Please help support I HATE MINIONS, add a #Twibbon now! http://t.co/E8a9jp5SZ0
id: 633055160588066816, text: RT @zaynmalik: Didn't realise the hate for minions was this real fuck these aliens got it bad
id: 633055161359970304, text: RT @2sam200: Only minions I accept.  https://t.co/NOOdn8Cpye
id: 633055164233068545, text: RT @camilacabello97: @alexanderdeleon minions movie soon please
```

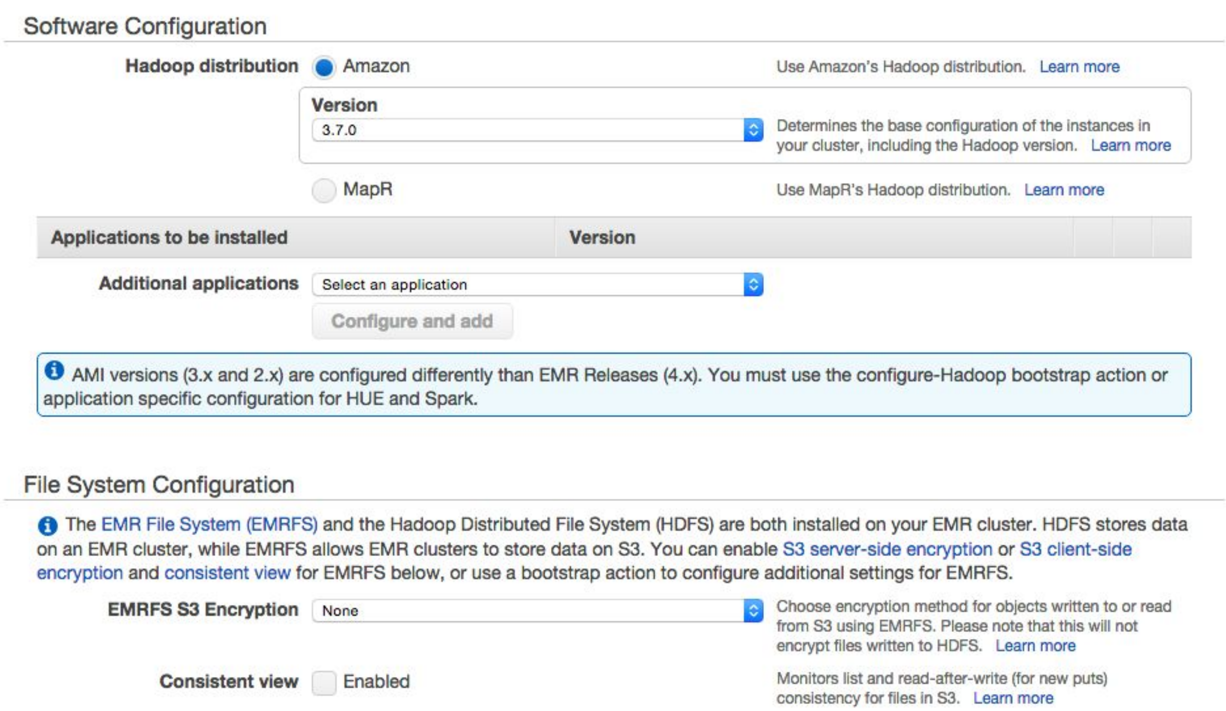
Using the amazon.py script:

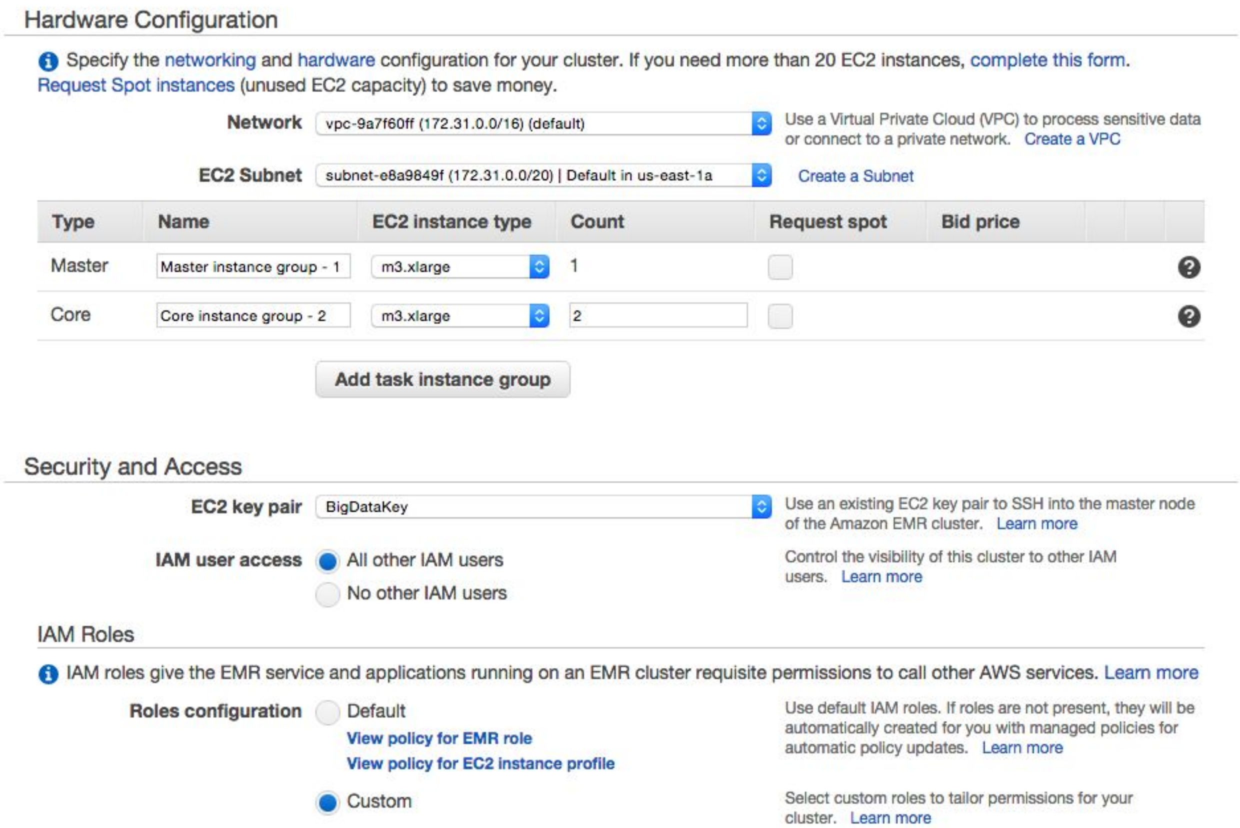
```
import twitter
import csv
import HTMLParser
import pymongo
from numpy import *
import re

api = twitter.Api(
consumer_key='JeCdbwdDuVpPzwRh4aLvJYKj2',
consumer_secret='6cEXsnpvcaonemZz9KJG059E2yOXWLFCnQax0t1rdq9FRcPwIq',
access_token_key='2313880676-QhO7bT3tojJfurFZJjjpJqAOXtj7fdB6gJPLzgk',
access_token_secret='XVTtZu0sYtd7WqBemaDf8bOXSEhUkqfUEwJDXv1MoRFJf'
)

html_parser = HTMLParser.HTMLParser()
search4 = api.GetSearch(term=('Minions'),lang='en', count=200, result_type='recent')

file = open('/Users/insignia/Desktop/Big_Data_Analytics/alchemy/out.txt', 'w')

for tweet4 in search4:
    t4 = html_parser.unescape(tweet4.text)
    text4 = re.sub(r"http\S+", "", t4)
    encode4 = text4.encode('ascii','ignore').decode('ascii')
    twe4 = " ".join(re.findall('[A-Z][^A-Z]*', encode4))
    cleaned4 = ' '.join(re.sub("(@[A-Za-z0-9]+)|([^0-9A-Za-z \t])","",twe4).split())
    file.write('id: '+str(tweet4.id)+', '+'text: '+cleaned4+'\n')
file.close()
~
~
~
```

Use the same script replacing the movie name with all the movie data you want to collect. Upload the generated file to S3 bucket input folder.

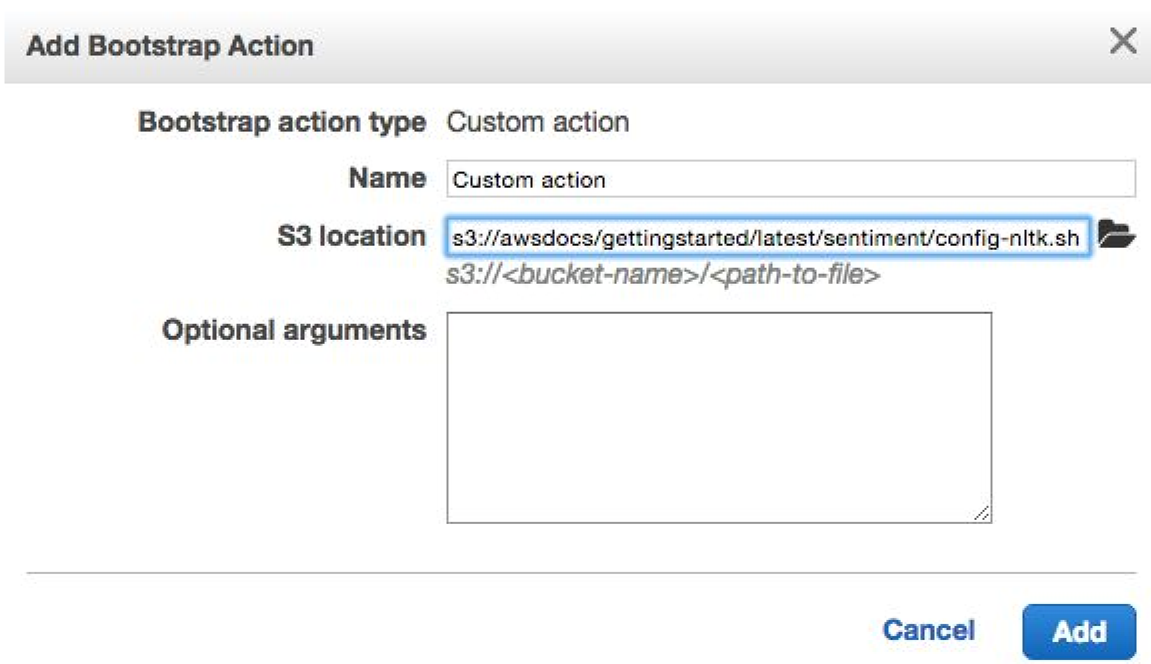
Create the amazon clusterin EMR using the following configuration:



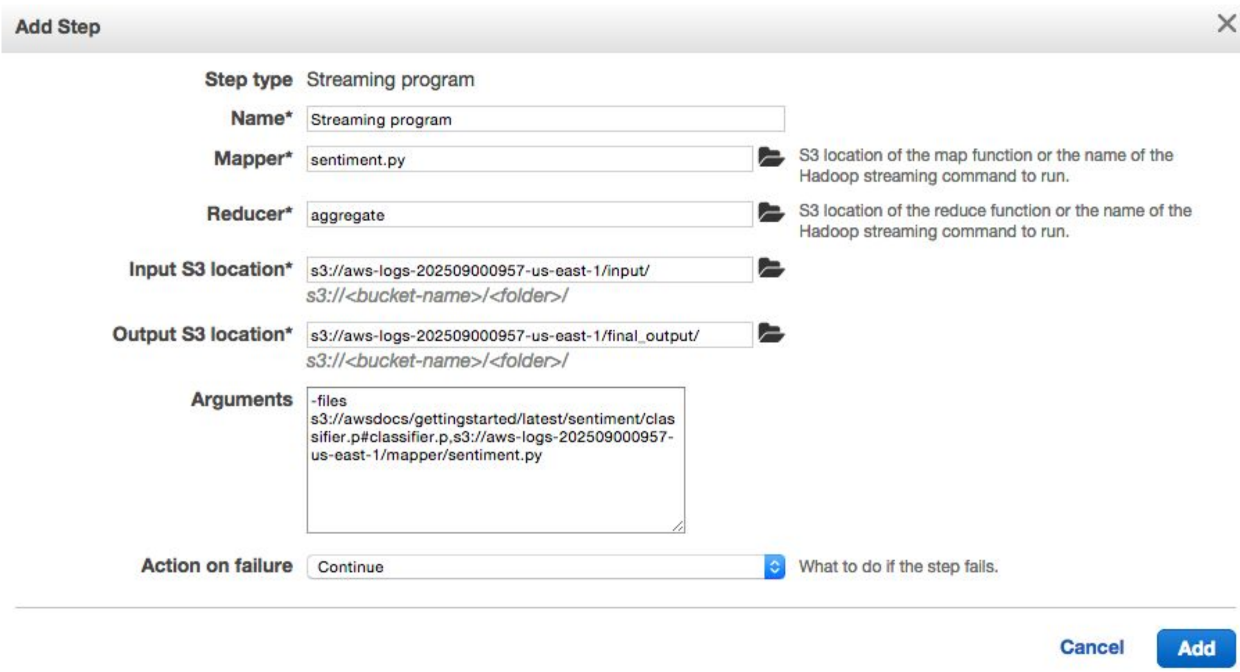




Hardware Configuration
Specify the networking and hardware configuration for your cluster. If you need more than 20 EC2 instances, complete this form.
Request Spot instances (unused EC2 capacity) to save money.
Network
vpc-9a7f60ff (172.31.0.0/16) (default)
Use a Virtual Private Cloud (VPC) to process sensitive data or connect to a private network. Create a VPC
EC2 Subnet
subnet-e8a9849f (172.31.0.0/20) | Default in us-east-1a
Create a Subnet
Type
Name
EC2 instance type
Count
Request spot
Bid price
Master
Master instance group - 1
m3.xlarge
1
Core
Core instance group - 2
m3.xlarge
2
Add task instance group
Security and Access
EC2 key pair
BigDataKey
Use an existing EC2 key pair to SSH into the master node of the Amazon EMR cluster. Learn more
IAM user access
All other IAM users
No other IAM users
Control the visibility of this cluster to other IAM users. Learn more
IAM Roles
IAM roles give the EMR service and applications running on an EMR cluster requisite permissions to call other AWS services. Learn more
Roles configuration
Default
View policy for EMR role
View policy for EC2 instance profile
Use default IAM roles. If roles are not present, they will be automatically created for you with managed policies for automatic policy updates. Learn more
Custom
Select custom roles to tailor permissions for your cluster. Learn more



Add Bootstrap Action
Bootstrap action type
Custom action
Name
Custom action
S3 location
s3://awsdocs/gettingstarted/latest/sentiment/config-nltk.sh
s3://<bucket-name>/<path-to-file>
Optional arguments
Cancel
Add



Click on add and then start the cluster. The job takes sometime to run. After the job completes it will generate the output in the output location specified above(Note: the output folder should not be created beforehand, EMR creates it on the go and generates the output data.)

It created separate files for the number of tweets catagorized as positive, no match and rest by default are catogarized as negatives:

The above output is for the minions movie that had 500 tweets out of which 364 where categorized as positives and 12 where categorized as no match and rest where negatives.

Analysis

Our main aim here is to determine the rating of a movie based purely upon user tweets about the movie. Doing this is not a very straight forward task but it takes some preliminary analysis into account to see a general trend in the tweets about a particular movie.

We chose the following 5 movies to base our analysis:

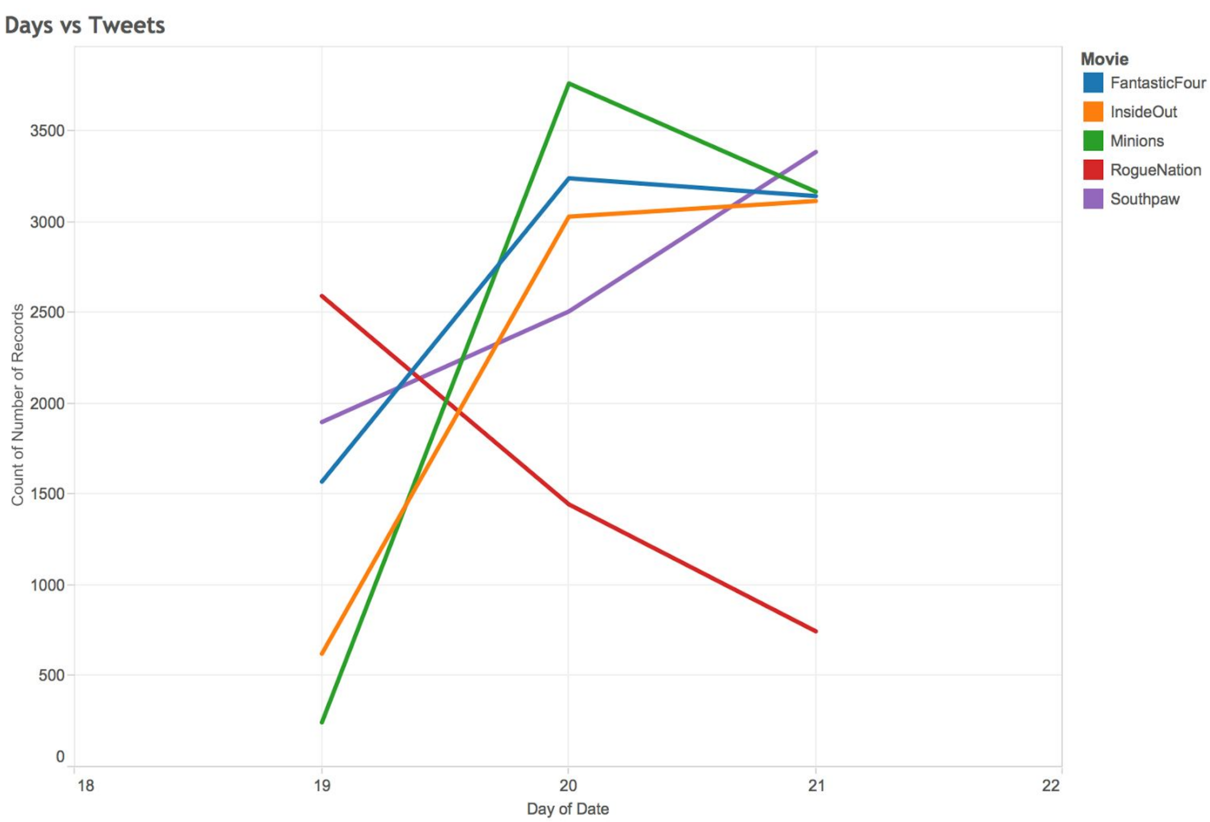
- Fantastic Four
- Inside Out
- Minions
- Rogue Nation
- Southpaw

At the end of our analysis, we will try to suggest the users which movie should be a must watch, which should be a No-Go and which should be Maybe-Watch.

1. Days vs Tweets: The first thing we try to analyse is the number of tweets each of these movies are getting on a daily basis. We took data for 3 days, 19th, 20th and 21st August 2015 and then plotted the number of tweets versus the movie.

   Findings
   - Number of tweets increased continuously for the movie **Southpaw**
   - Number of tweets decreased continuously for the movie **Rogue Nation**
   - For the rest of the movies, number of tweets increased on 20th August, but decrease slightly on 21st August.

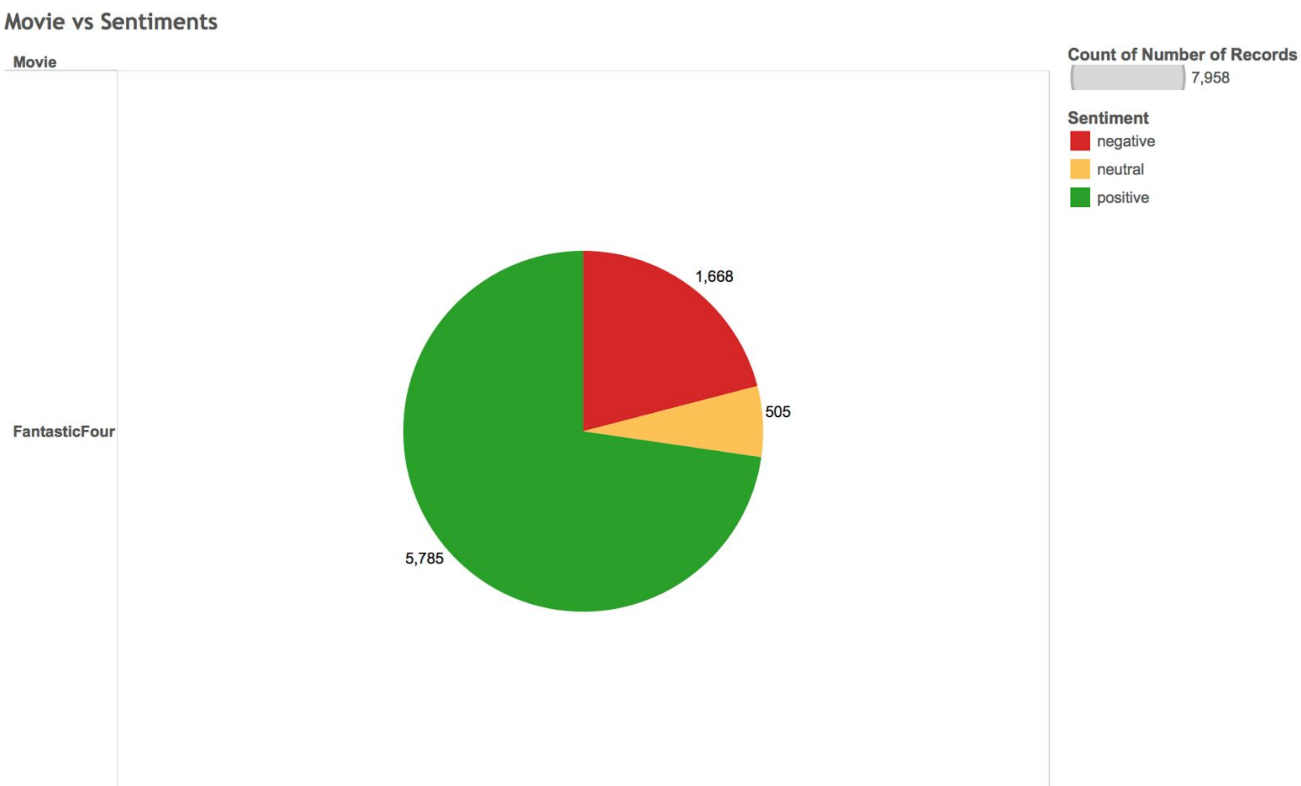
This can be inferred as Popularity of the movie among twitter users.



The trend of count of Number of Records for Date Day. Color shows details about Movie. The view is filtered on Movie, which keeps FantasticFour, InsideOut, Minions, RogueNation and Southpaw.

2. <u>Movie</u> <u>vs</u> <u>Tweet</u> <u>Sentiments</u>: Each tweet about a movie has a sentiment assigned to it. The tweet is either Positive, Neutral or Negative. So, over a period of 3 days, we analyse, for each movie, how many tweets are positive, neutral or negative.

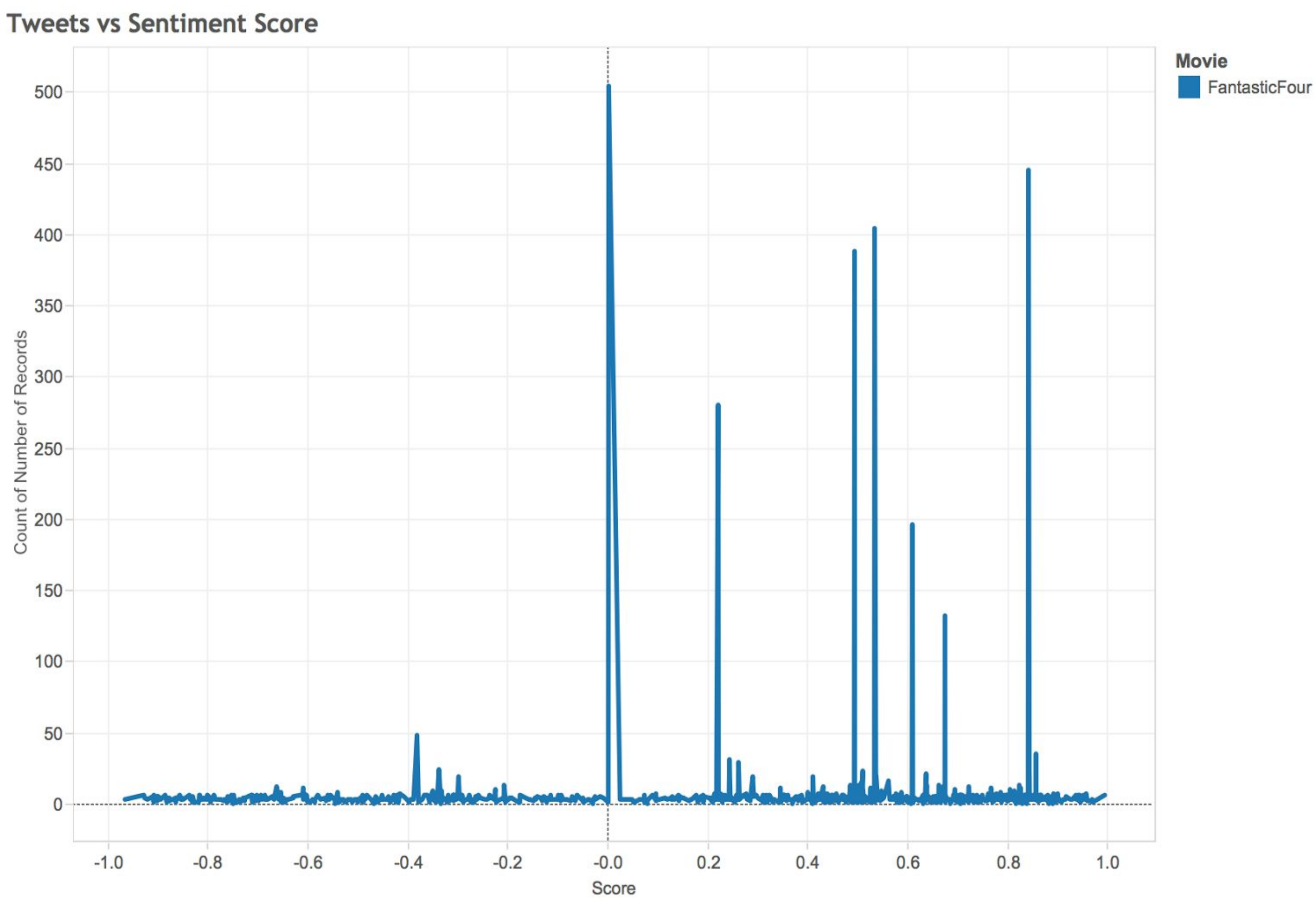
Example: Here, for the movie Fantastic Four, we gathered 7958 tweets for 3 days and among those tweets, 5,785 are positive, 505 are neutral whereas 1668 tweets are marked as negative.



Sum of Number of Records broken down by Movie. Color shows details about Sentiment. Size shows count of Number of Records. The marks are labeled by sum of Number of Records. The view is filtered on Movie, which keeps FantasticFour.

3. Tweets vs Sentiment Score: Not only are the tweets categorized as positive, neutral or negative, but each tweet has a sentiment score attached with them. Sentiment score measures the positiveness, negativeness and neutrality of the tweet.

Example: Here, for the movie Fantastic Four, we can see that majority of the tweets have a sentiment score greater than 0, which in turn suggests that majority of tweets for the movie Fantastic Four are positive and that the general sentiment for this movie is positive.



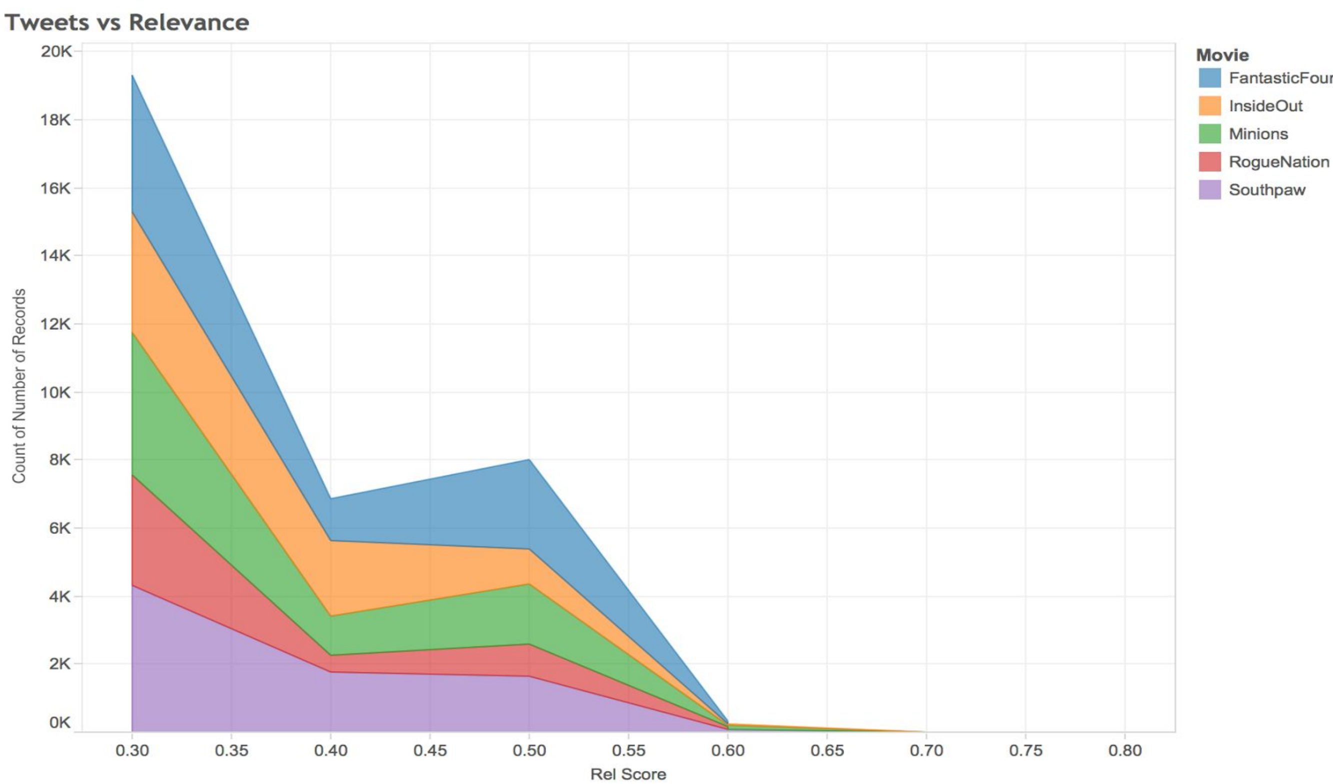
The trend of count of Number of Records for Score. Color shows details about Movie. The view is filtered on Movie, which keeps Fantastic-Four.

4. <u>Tweets</u> <u>vs</u> <u>Relevance:</u> Though each tweet has a sentiment score attached to it, we cannot directly use it to determine the final rating a movie should get. We devised a formula to calculate the relevance of the tweet. When we say relevance, we mean to say how much that tweet is going to affect the people who read it.

   We take 4 features into account to determine the relevance score:

- the count of retweets that tweet has got
- the count of favorites that tweet has got
- the number of followers of the user who tweeted
- whether the user is verified or not

We trained our model by manually giving relevance score for a portion of tweets and then tried to predict the relevance score on the rest of the tweets. The results are plotted in the graph below.



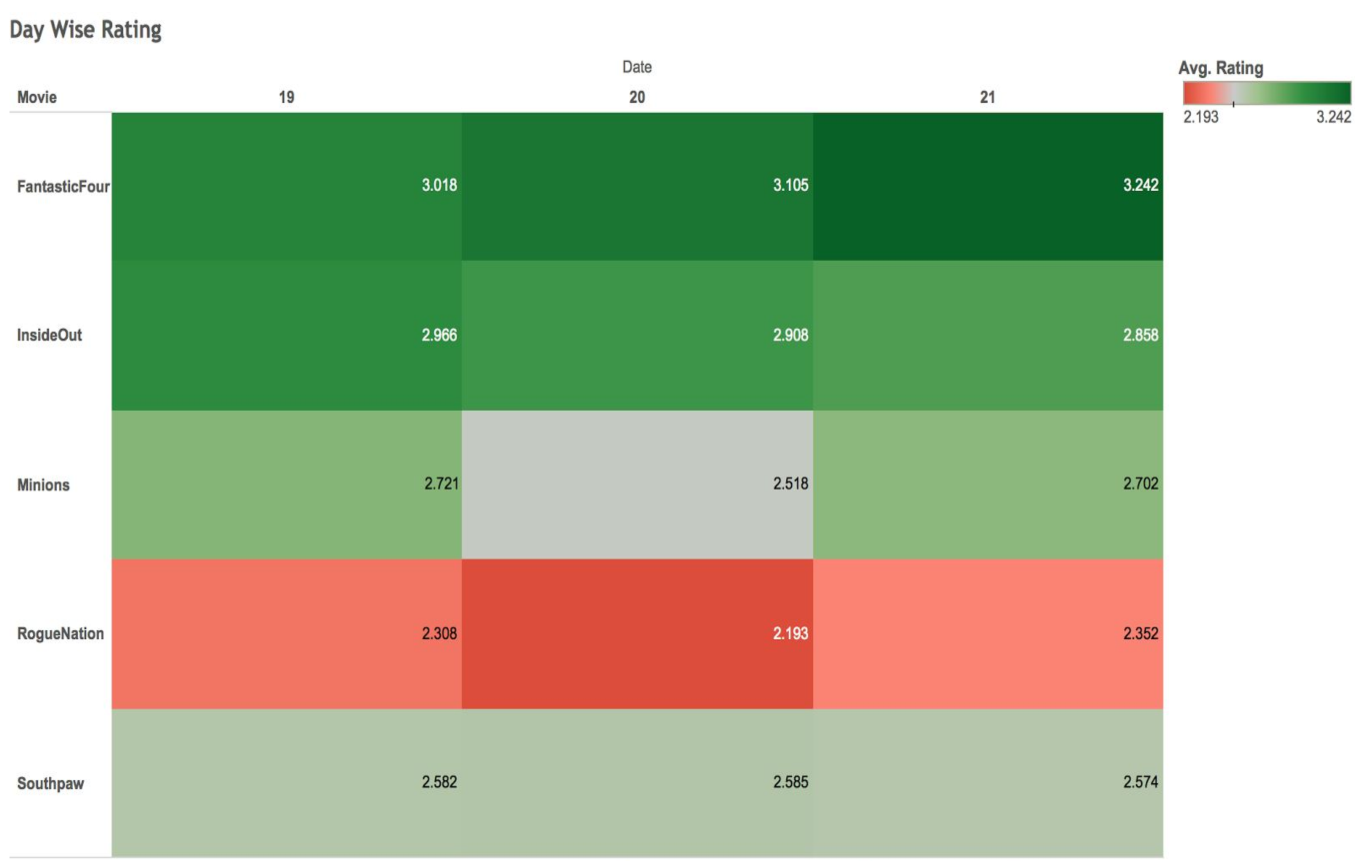
The plot of count of Number of Records for Rel Score. Color shows details about Movie. The view is filtered on Movie, which keeps FantasticFour, InsideOut, Minions, RogueNation and Southpaw.

Final Rating

Since we now have Sentiment Score and Relevance score, we find the final rating of the movie.

As per our model, the maximum product of sentiment score and relevance score can be One (1). So we scaled these measures on a scale of 5, since we are finding a rating out of 5.

After calculating the final ratings for each movie, day-wise, we see that Fantastic Four has increasing rating day-by-day, which means that users on twitter are liking this movie. Similarly, for the movie Rogue Nation, the ratings for each day are below average, which is a flag and a bad indicator of the liking of the movie.



Average of Rating broken down by Date Day vs. Movie. Color shows average of Rating. The marks are labeled by average of Rating. The view is filtered on Movie, which keeps FantasticFour, InsideOut, Minions, RogueNation and Southpaw.

Conclusion

We also take the average rating over a period of three days, and after 3 days, we can say that **Fantastic Four** is a must watch movie, **Rogue Nation** is a No-Go and the rest of the movies are a Maybe-Watch.

Average of Rating broken down by Movie. Color shows average of Rating. The marks are labeled by average of Rating. The view is filtered on Movie, which keeps FantasticFour, InsideOut, Minions, RogueNation and Southpaw.

Dashboard

Finally, we present a dashboard for overall analysis of the tweets for all movies. This dashboard helps in understanding how a movie has been faring in the twitter world.