BIG DATA & INTELLIGENT ANALYTICS

Twitter Sentiment Analysis

Final Project Report

Prepared by

Avikal Chhetri

Table of Contents

Introduction

Social Media sites like Twitter, Facebook, etc. are like a warehouse of emotions. People tend to share their happiness, sadness and also vent out their frustrations and anger! This collection of people's sentiments in the public domain is can be of great value of utilized effectively.

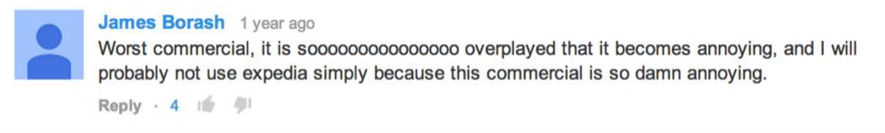
The applications of sentiment analysis are broad and powerful. The ability to extract insights from social data is a practice that is being widely adopted by organizations across the world.

Some examples include:

Shifts in sentiment on social media have been shown to correlate with shifts in the stock market.

The Obama administration used sentiment analysis to gauge public opinion to policy announcements and campaign messages ahead of 2012 presidential election.

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



James Borash 1 year ago
Worst commercial, it is sooooooooooooooo overplayed that it becomes annoying, and I will probably not use expedia simply because this commercial is so damn annoying.

Sentiment analysis is in demand because of its efficiency. Thousands of text documents can be processed for sentiment (and other features including named entities, topics, themes, etc.) in seconds, compared to the hours it would take a team of people to manually complete. Because it is so efficient (and accurate – *Semantria* has 80% accuracy for English content) many businesses are adopting text and sentiment analysis and incorporating it into their processes.

But machines still will never be able to measure sentiment as well as humans, and even humans don't agree 100% of the time. The number of sentiment types is also part of the equation. Some platforms offer three sentiments, some offer four, and some offer more than five. The more you increase the number of sentiment types, the less accurate (but more information rich) your results become. And it can be hard to figure out the sentiment from say a sarcastic tweet- which sometimes even humans have a problem demystifying.

However, if properly utilized and taking the key insights and weeding out the junk, you can generate great value.

In this report we will look at the infrastructure we built for performing sentiment analysis on twitter feeds done in Apache Spark and show the continuous visualizations of the sentiments observed.

Executive Summary

Problem Statement

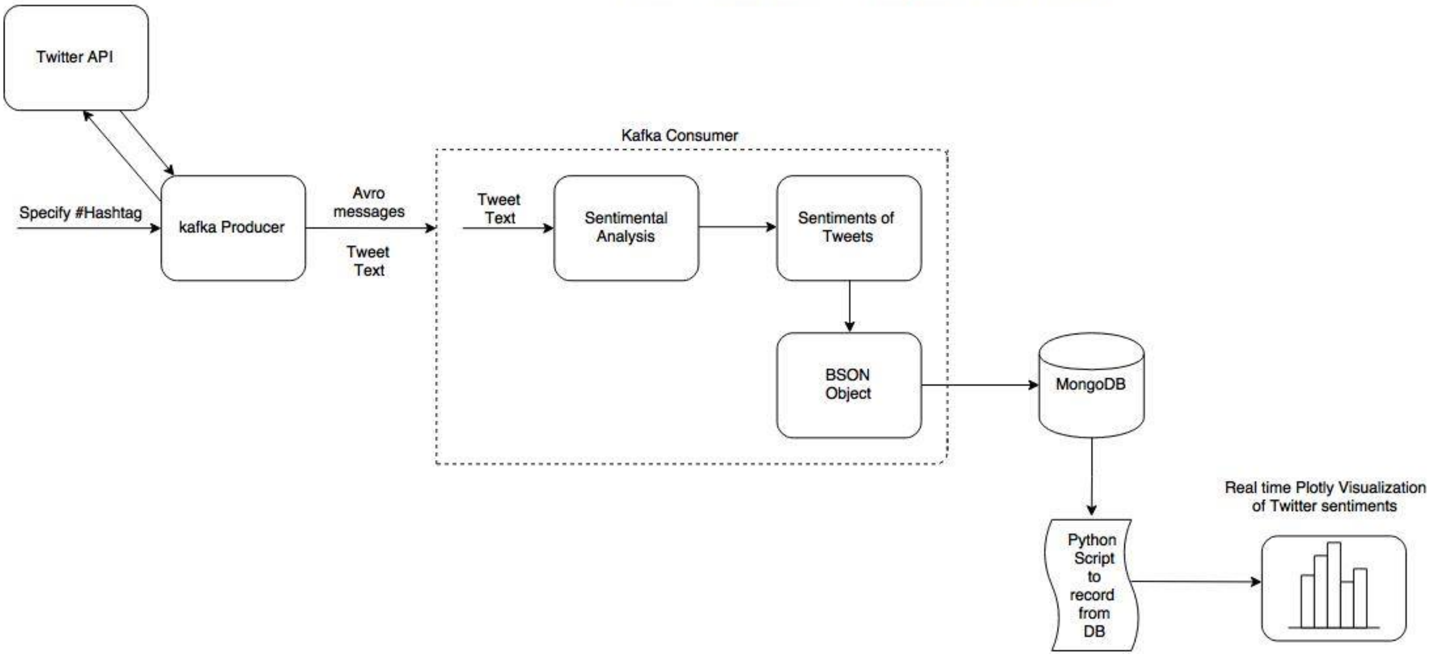
To build a model that obtains and classifies the trend of sentiments of a stream of tweets for a given #hashtag.

Approach

- First, a Kafka Producer is used to fetch the tweets from the Twitter API by applying the #hashtag input as the filter. The producer then emits the tweets in Avro format to be received by the consumer.
- We then have a Kafka consumer which receives the tweets emitted by the producer and process them by using the Spark Streaming context.
- Now the stream of tweets are processed as ‘RDDs of tweets’. Each of the tweet text is now mapped and processed for sentiment analysis. The sentiment analysis is calculated by accessing the Stanford Core NLP libraries and gives out the resultant sentiments like: ‘Positive’, ‘Very Positive’, ‘Neutral’, ‘Negative’ and ‘Very Negative’.
- The total sentiment count of each category is calculated and is stored in the database (MongoDB) by using BSON objects.
- A python script is then run to access the records in the database in a timed loop and respectively stream the data to the Plotly servers.
- Plotly, upon receiving the data displays a streaming graph of the sentiments observed.

Architecture Diagram

The following is a visual representation of our approach explained above:



Technologies And Software Used

- Kafka 2.11-0.8.2.1
- Twitter API 1.1
- Apache Spark 1.4.0
- IPython Notebook
- Plotly 1.8.3
- Gradle (for building)

Code

Kafka Producer: KafkaProducerApp.scala

```
import java.util.Properties

import com.twitter.bijection.avro.SpecificAvroCodecs.{toJson, toBinary}

import com.typesafe.config.ConfigFactory

import kafka.javaapi.producer.Producer

import kafka.producer.{KeyedMessage, ProducerConfig}

import net.mkrcah.TwitterStream.OnTweetPosted

import net.mkrcah.avro.Tweet

import twitter4j.{Status, FilterQuery}

object KafkaProducerApp {

  private val conf = ConfigFactory.load()

  val KafkaTopic = "tweets"

  val kafkaProducer = {

    val props = new Properties()

    props.put("metadata.broker.list", conf.getString("kafka.brokers"))

    props.put("request.required.acks", "1")

    val config = new ProducerConfig(props)

    new Producer[String, Array[Byte]](config)

  }

  def main (args: Array[String]) {

    val twitterStream = TwitterStream.getStream

      TwitterStream.keyword_=("Obama")

      var keyword = TwitterStream._keyword

    twitterStream.addListener(new OnTweetPosted(s => sendToKafka(toTweet(s))))

    twitterStream.filter(new FilterQuery().track(keyword.split(",")))

  }

private def toTweet(s: Status): Tweet = {
    new Tweet(s.getUser.getName, s.getText)
  }
  private def sendToKafka(t:Tweet) {
    println(toJson(t.getSchema).apply(t))
    val tweetEnc = toBinary[Tweet].apply(t)
    val msg = new KeyedMessage[String, Array[Byte]](KafkaTopic, tweetEnc)
    kafkaProducer.send(msg)
  }
}
```

Kafka Consumer : KafkaConsumerApp.scala

```
import com.typesafe.config.ConfigFactory
import kafka.serializer.{DefaultDecoder, StringDecoder}
import com.twitter.bijection.avro.SpecificAvroCodecs
import net.mkrcah.avro.Tweet
import org.apache.spark._
import org.apache.spark.storage.StorageLevel
import org.apache.spark.streaming.StreamingContext._
import org.apache.spark.streaming._
import org.apache.spark.streaming.kafka._
import org.apache.spark.rdd.RDD
import org.apache.spark.{SparkConf, SparkContext}
import org.apache.spark.sql.SQLContext
import edu.stanford.nlp.io
import edu.stanford.nlp.ling
import edu.stanford.nlp.trees
import edu.stanford.nlp.util
import edu.stanford.nlp.ling.CoreAnnotations
import edu.stanford.nlp.ling.CoreLabel
import edu.stanford.nlp.pipeline
import edu.stanford.nlp.time
import edu.stanford.nlp.util.CoreMap
import edu.stanford.nlp.pipeline.StanfordCoreNLP
import edu.stanford.nlp.sentiment.SentimentCoreAnnotations
import edu.stanford.nlp.ling.CoreAnnotations.SentencesAnnotation
import java.util.{List => JList}
import scala.collection.JavaConversions._
import java.lang.Object
import java.util.Properties
import java.util.List
import java.util.Calendar
import org.apache.spark.sql.SQLContext
import org.apache.spark.Partitioner.defaultPartitioner
import org.apache.spark.annotation.Experimental
import org.apache.spark.deploy.SparkHadoopUtil
import org.apache.spark.executor.{DataWriteMethod, OutputMetrics}
import org.apache.spark.mapreduce.SparkHadoopMapReduceUtil
import org.apache.spark.partial.{BoundedDouble, PartialResult}
import org.apache.spark.serializer.Serializer
import org.apache.spark.util.collection.CompactBuffer
import org.apache.spark.util.random.StratifiedSamplingUtils
import org.apache.spark.Logging
import org.apache.spark.rdd.PairRDDFunctions
import scala.Serializable
import org.bson.BasicBSONObject
import org.apache.hadoop.conf.Configuration
import org.bson.BSONObject
import com.mongodb.hadoop.{
  MongoInputFormat, MongoOutputFormat,
  BSONFileInputFormat, BSONFileOutputFormat}
object KafkaConsumerApp extends App{
  private val conf = ConfigFactory.load()
```

```
val sparkConf = new SparkConf().setAppName("kafka-twitter-spark-example").setMaster("local[*]")
val ssc = new SparkContext(sparkConf)
val sc = new StreamingContext(ssc, Seconds(10))
val sqlContext = new SQLContext(ssc)
val config = new Configuration()
config.set("mongo.input.uri", "mongodb://localhost:27017/twitter.twitter")
config.set("mongo.output.uri", "mongodb://localhost:27017/twitter.twitter")
 val encTweets = {
 val topics = Map(KafkaProducerApp.KafkaTopic -> 1)
 val kafkaParams = Map(
 "zookeeper.connect" -> conf.getString("kafka.zookeeper.quorum"),
 "group.id" -> "1")
 KafkaUtils.createStream[String, Array[Byte], StringDecoder, DefaultDecoder](
 sc, kafkaParams, topics, StorageLevel.MEMORY_ONLY)
 }
 val tweets = encTweets.flatMap(x => SpecificAvroCodecs.toBinary[Tweet].invert(x._2).toOption)
 def sentiment(text:String) : String = {
 val props = new Properties()
 props.setProperty("annotators", "tokenize, ssplit, pos, lemma, parse, sentiment")
 val pipeline = new  StanfordCoreNLP(props)
 val annotation = pipeline.process(text)
 val sentences : java.util.List[CoreMap] = annotation.get(classOf[SentencesAnnotation])
 var senti = ""
 for (sentence <- sentences){
 val sentiment = sentence.get(classOf[SentimentCoreAnnotations.SentimentClass])
 senti = sentiment
 }
 senti
 }
val sentiments = tweets.map(twt => sentiment(twt.getText)).map((_,1)).reduceByKey(_ + _)
 val sentiments2 = sentiments.reduceByKey(_ + _)
```

```
    val countsSorted = sentiments2.transform(_.sortBy(_._2, ascending = false))

    val countSorted2 = countsSorted.reduceByKey(_ + _)



    val saveRDD = countSorted2.map((tuple) => {

    var bson = new BasicBSONObject()

    var twt_time = Calendar.getInstance().getTime()

    bson.put("timestamp", twt_time.toString)

    bson.put("sentiment", tuple._1)

    bson.put("count", tuple._2.toString)

    bson.put("flag", "0")

      (null, bson)

    })

    saveRDD.foreachRDD(rdd => {

    val pair_rdd = new PairRDDFunctions[Null, org.bson.BasicBSONObject](rdd)

     pair_rdd.saveAsNewAPIHadoopFile("file:///bogus", classOf[Any], classOf[Any],
classOf[com.mongodb.hadoop.MongoOutputFormat[Any, Any]], config)

    })

    countSorted2.print(

  sc.start()

  sc.awaitTermination()

}
```

Python Script to access records from MongoDB and deploy to Plotly for the streaming visualization

```
from pymongo import MongoClient

connection = MongoClient("mongodb://localhost:27017/db.twitter")
db = connection.twitter

import collections
import time
from bson import json_util
import json
from bson.son import SON

positive_sentiment_list = []
negative_sentiment_list= []
```

```
vnegative_sentiment_list = []
vpositive_sentiment_list = []
neutral_sentiment_list = []
positive_count_list = []
negative_count_list = []
vpositive_count_list = []
vnegative_count_list = []
neutral_count_list = []
myresults = []

def extractFromMongo():

    global myresults
    db.eval('db.twitter.find().forEach(function(x){ x.count = parseInt(x.count); db.twitter.save(x);})')
    pipeline = [
        {"$group": {"_id": {"sentiment" :"$sentiment","timestamp":"$timestamp", "flag":"$flag"}, "count": {"$sum":
"$count"}}}
    ]
    myresults = list(db.twitter.aggregate(pipeline))

    for i , v in enumerate(myresults):

        global positive_sentiment_list
        global negative_sentiment_list
        global vnegative_sentiment_list
        global vpositive_sentiment_list
        global neutral_sentiment_list
        global positive_count_list
        global negative_count_list
        global vpositive_count_list
        global vnegative_count_list
        global neutral_count_list

        if (v['_id']['sentiment'] == 'Positive' and v['_id']['flag'] == '0'):
            positive_sentiment_list.append(json.dumps(v['_id']['timestamp'], json_util.default))
            positive_count_list.append(v['count'])

        if (v['_id']['sentiment'] == 'Negative' and v['_id']['flag'] == '0'):
            negative_sentiment_list.append(json.dumps(v['_id']['timestamp'], json_util.default))
            negative_count_list.append(v['count'])

        if (v['_id']['sentiment'] == 'Very positive' and v['_id']['flag'] == '0'):
            vpositive_sentiment_list.append(json.dumps(v['_id']['timestamp'], json_util.default))
            vpositive_count_list.append(v['count'])

        if (v['_id']['sentiment'] == 'Very negative' and v['_id']['flag'] == '0'):
            vnegative_sentiment_list.append(json.dumps(v['_id']['timestamp'], json_util.default))
            vnegative_count_list.append(v['count'])

        if (v['_id']['sentiment'] == 'Neutral' and v['_id']['flag'] == '0'):
            neutral_sentiment_list.append(json.dumps(v['_id']['timestamp'], json_util.default))
            neutral_count_list.append(v['count'])


def executeSomething():
    extractFromMongo()
    db.twitter.update_many({'flag': '0'}, {'$set': {'flag': '1'}})

import plotly.plotly as py
import plotly.tools as tls
from plotly.graph_objs import *

streamid_0='jzfwahcw81'
streamid_1='0dvrtxfmia'
streamid_2='sp6smkrptf'
streamid_3='yb45098xg4'
streamid_4='dbw2orc57h'

py.sign_in('avikalchhetri', 'gxu9cteniu')

trace0 = Bar(
    x = vpositive_sentiment_list,
    y = vpositive_count_list,
    name='Very Positive Count',
    stream=Stream(token=streamid_0, maxpoints=100),
    marker=Marker(
        color='rgb(0, 102, 0)',
        opacity=0.7,
    ),
)
trace1 = Bar(
    x = positive_sentiment_list,
    y= positive_count_list,
    name='Positive Count',
    stream=Stream(token=streamid_1, maxpoints=100),
    marker=Marker(
        color='rgb(51, 204, 51)',
        opacity=0.5,
    ),
)
trace2 = Bar(
    x = neutral_sentiment_list,
    y= neutral_count_list,
    name='Neutral Count',
    stream=Stream(token=streamid_2, maxpoints=100),
    marker=Marker(
        color='rgb(0, 102, 255)',
        opacity=0.5,
    ),
)
trace3 = Bar(
    x = negative_sentiment_list,
    y= negative_count_list,
    name='Negative Count',
    stream=Stream(token=streamid_3, maxpoints=100),
    marker=Marker(
        color='rgb(255, 0, 0)',
        opacity=0.5,
    ),
)
trace4 = Bar(
    x = vnegative_sentiment_list,
    y= vnegative_count_list,
    name='Very Negative Count',
    stream=Stream(token=streamid_4, maxpoints=100),
    marker=Marker(
        color='rgb(126, 40, 40)',
        opacity=0.5,
    )
)
data = Data([trace0, trace1, trace2, trace3, trace4])
layout = Layout(
    xaxis=XAxis(
        # set x-axis' labels direction at 45 degree angle
        tickangle=-15,
    ),
    barmode='group',
)
fig = Figure(data=data, layout=layout)
plot_url = py.plot(fig, filename='Twitter Sentimental Analysis')
#tls.embed(plot_url)

s0 = py.Stream(streamid_0)
s1 = py.Stream(streamid_1)
s2 = py.Stream(streamid_2)
s3 = py.Stream(streamid_3)
s4 = py.Stream(streamid_4)

s0.open()
s1.open()
s2.open()
s3.open()
s4.open()

while True:
    executeSomething()

    s0.heartbeat()
    s1.heartbeat()
    s2.heartbeat()
    s3.heartbeat()
    s4.heartbeat()


   s0.write(dict(x= vpositive_sentiment_list, y = vpositive_count_list))
    s1.write(dict(x= positive_sentiment_list, y=positive_count_list) )
    s2.write(dict(x= neutral_sentiment_list, y=neutral_count_list))
    s3.write(dict(x= negative_sentiment_list, y=negative_count_list))
    s4.write(dict(x= vnegative_sentiment_list, y=vnegative_count_list))

    s0.heartbeat()
    s1.heartbeat()
    s2.heartbeat()
    s3.heartbeat()
```

```
s4.heartbeat()

time.sleep(4)

s0.heartbeat()
s1.heartbeat()
s2.heartbeat()
s3.heartbeat()
s4.heartbeat()
```

Deployment Instructions

1. Start Zookeeper server

```
bin/zookeeper-server-start.sh /Users/avikalchhetri/kafka_2.11-0.8.2.1/config/zookeeper.properties
```

2. Start Kafka server

```
bin/kafka-server-start.sh /Users/avikalchhetri/kafka_2.11-0.8.2.1/config/server.properties
```

3. Mention the #hastag in 'TwitterStream.keyword' in the Kafka Producer program

4. Start Kafka producer

```
./gradlew produce
```

This will start to read recent tweets, encode them to Avro and send to the Kafka cluster.

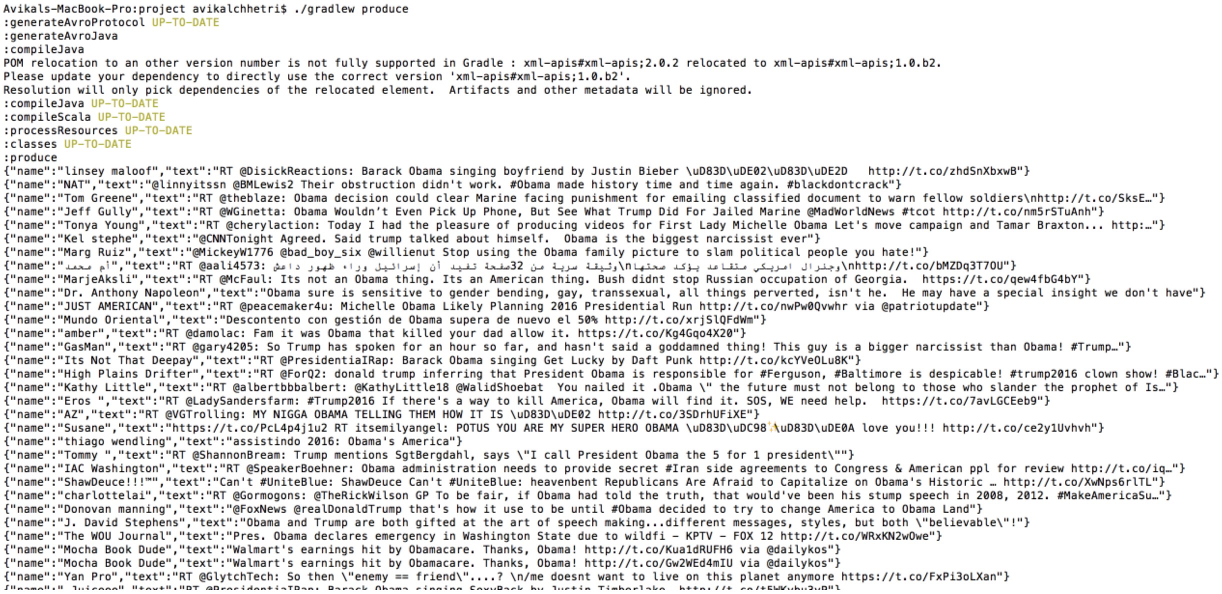
5. Start Kafka consumer

```
./gradlew consume
```

6. Run python script for visualization in plotly.

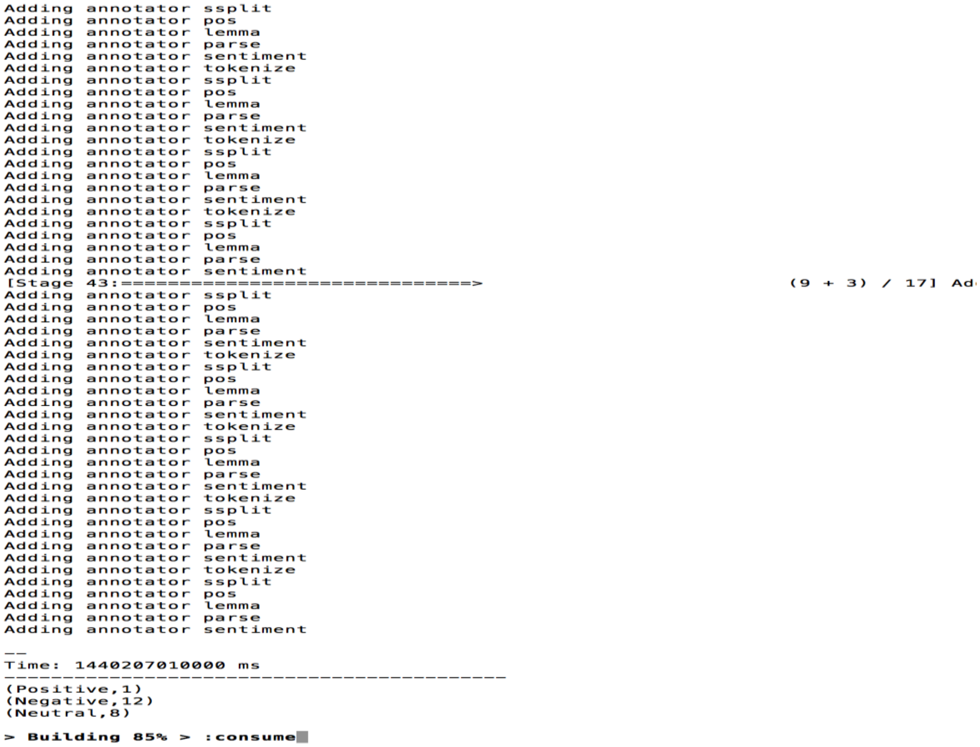
Output & Screenshots

After running the zookeeper server, running the Kafka producer:



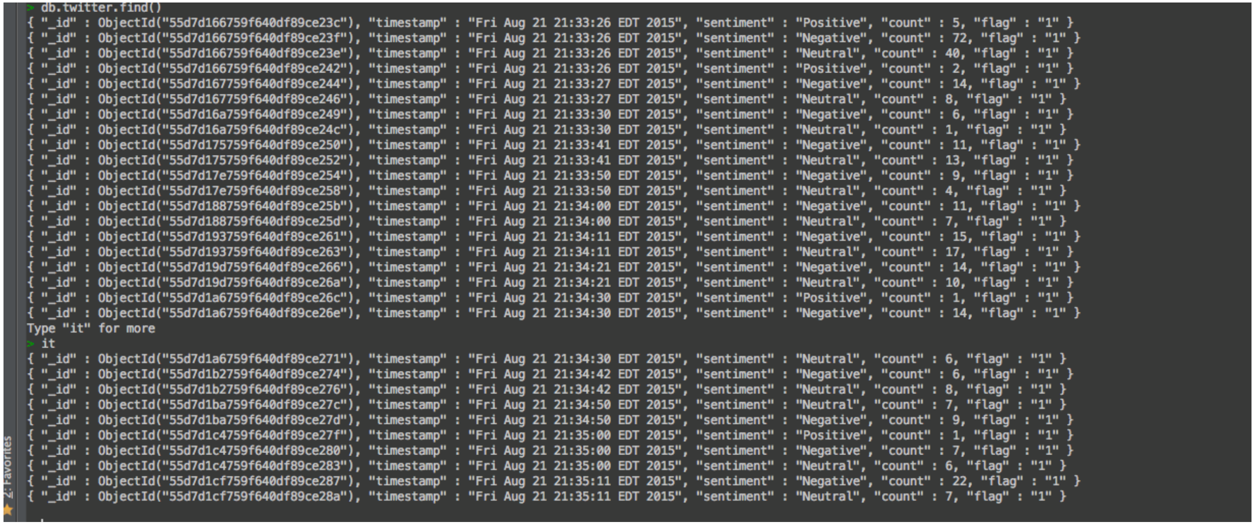
```
Avikals-MacBook-Pro:project avikalchhetri$ ./gradlew produce
:generateAvroProtocol UP-TO-DATE
:generateAvroJava
:compileJava
POM relocation to an other version number is not fully supported in Gradle : xml-apis#xml-apis;2.0.2 relocated to xml-apis#xml-apis;1.0.b2.
Please update your dependency to directly use the correct version 'xml-apis#xml-apis;1.0.b2'.
Resolution will only pick dependencies of the relocated element.  Artifacts and other metadata will be ignored.
:compileJava UP-TO-DATE
:compileScala UP-TO-DATE
:processResources UP-TO-DATE
:classes UP-TO-DATE
:produce
{"name":"linsey maloof","text":"RT @DisickReactions: Barack Obama singing boyfriend by Justin Bieber \uD83D\uDE02\uD83D\uDE2D   http://t.co/zhdSnXbxwB"}
{"name":"NAT","text":"@linnyitssn @BMLewis2 Their obstruction didn't work. #Obama made history time and time again. #blackdontcrack"}
{"name":"Tom Greene","text":"RT @theblaze: Obama decision could clear Marine facing punishment for emailing classified document to warn fellow soldiers\nhttp://t.co/SksE…"}
{"name":"Jeff Gully","text":"RT @WGinetta: Obama Wouldn't Even Pick Up Phone, But See What Trump Did For Jailed Marine @MadWorldNews #tcot http://t.co/nm5rSTuAnh"}
{"name":"Tonya Young","text":"RT @cherylaction: Today I had the pleasure of producing videos for First Lady Michelle Obama Let's move campaign and Tamar Braxton... http:…"}
{"name":"Kel stephe","text":"@CNNTonight Agreed. Said trump talked about himself.  Obama is the biggest narcissist ever"}
{"name":"Marg Ruiz","text":"@MickeyW1776 @bad_boy_six @willienut Stop using the Obama family picture to slam political people you hate!"}
{"name":"أم محمد","text":"RT @aali4573: وثيقة سرية من 32صفحة تفيد أن إسرائيل وراء ظهور داعش\nجنرال امريكي متقاعد يؤكد صحتها\nhttp://t.co/bMZDq3T7OU"}
{"name":"MarjeAksli","text":"RT @McFaul: Its not an Obama thing. Its an American thing. Bush didnt stop Russian occupation of Georgia.  https://t.co/qew4fbG4bY"}
{"name":"Dr. Anthony Napoleon","text":"Obama sure is sensitive to gender bending, gay, transsexual, all things perverted, isn't he.  He may have a special insight we don't have"}
{"name":"JUST AMERICAN","text":"RT @peacemaker4u: Michelle Obama Likely Planning 2016 Presidential Run http://t.co/nwPw0Qvwhr via @patriotupdate"}
{"name":"Mundo Oriental","text":"Descontento con gestión de Obama supera de nuevo el 50% http://t.co/xrjSlQFdWm"}
{"name":"amber","text":"RT @damolac: Fam it was Obama that killed your dad allow it. https://t.co/Kg4Gqo4X20"}
{"name":"GasMan","text":"RT @gary4205: So Trump has spoken for an hour so far, and hasn't said a goddamned thing! This guy is a bigger narcissist than Obama! #Trump…"}
{"name":"Its Not That Deepay","text":"RT @PresidentialRap: Barack Obama singing Get Lucky by Daft Punk http://t.co/kcYVeOLu8K"}
{"name":"High Plains Drifter","text":"RT @ForQ2: donald trump inferring that President Obama is responsible for #Ferguson, #Baltimore is despicable! #trump2016 clown show! #Blac…"}
{"name":"Kathy Little","text":"RT @albertbbbalbert: @KathyLittle18 @WalidShoebat  You nailed it .Obama \" the future must not belong to those who slander the prophet of Is…"}
{"name":"Eros ","text":"RT @LadySandersfarm: #Trump2016 If there's a way to kill America, Obama will find it. SOS, WE need help.  https://t.co/7avLGCEeb9"}
{"name":"AZ","text":"RT @VGTrolling: MY NIGGA OBAMA TELLING THEM HOW IT IS \uD83D\uDE02 http://t.co/3SDrhUFiXE"}
{"name":"Susane","text":"https://t.co/PcL4p4j1u2 RT itsemilyangel: POTUS YOU ARE MY SUPER HERO OBAMA \uD83D\uDC98✨\uD83D\uDE0A love you!!! http://t.co/ce2y1Uvhvh"}
{"name":"thiago wendling","text":"assistindo 2016: Obama's America"}
{"name":"Tommy ","text":"RT @ShannonBream: Trump mentions SgtBergdahl, says \"I call President Obama the 5 for 1 president\""}
{"name":"IAC Washington","text":"RT @SpeakerBoehner: Obama administration needs to provide secret #Iran side agreements to Congress & American ppl for review http://t.co/iq…"}
{"name":"ShawDeuce!!!™","text":"Can't #UniteBlue: ShawDeuce Can't #UniteBlue: heavenbent Republicans Are Afraid to Capitalize on Obama's Historic … http://t.co/XwNps6rlTL"}
{"name":"charlottelai","text":"RT @Gormogons: @TheRickWilson GP To be fair, if Obama had told the truth, that would've been his stump speech in 2008, 2012. #MakeAmericaSu…"}
{"name":"Donovan manning","text":"@FoxNews @realDonaldTrump that's how it use to be until #Obama decided to try to change America to Obama Land"}
{"name":"J. David Stephens","text":"Obama and Trump are both gifted at the art of speech making...different messages, styles, but both \"believable\"!"}
{"name":"The WOU Journal","text":"Pres. Obama declares emergency in Washington State due to wildfi - KPTV - FOX 12 http://t.co/WRxKN2wOwe"}
{"name":"Mocha Book Dude","text":"Walmart's earnings hit by Obamacare. Thanks, Obama! http://t.co/Kua1dRUFH6 via @dailykos"}
{"name":"Mocha Book Dude","text":"Walmart's earnings hit by Obamacare. Thanks, Obama! http://t.co/Gw2WEd4mIU via @dailykos"}
{"name":"Yan Pro","text":"RT @GlytchTech: So then \"enemy == friend\"....? \n/me doesnt want to live on this planet anymore https://t.co/FxPi3oLXan"}
{"name":" Juiceee","text":"RT @PresidentialRap: Barack Obama singing SexyBack by Justin Timberlake  http://t.co/t5WKvhu3vB"}
```

Running the consumer:



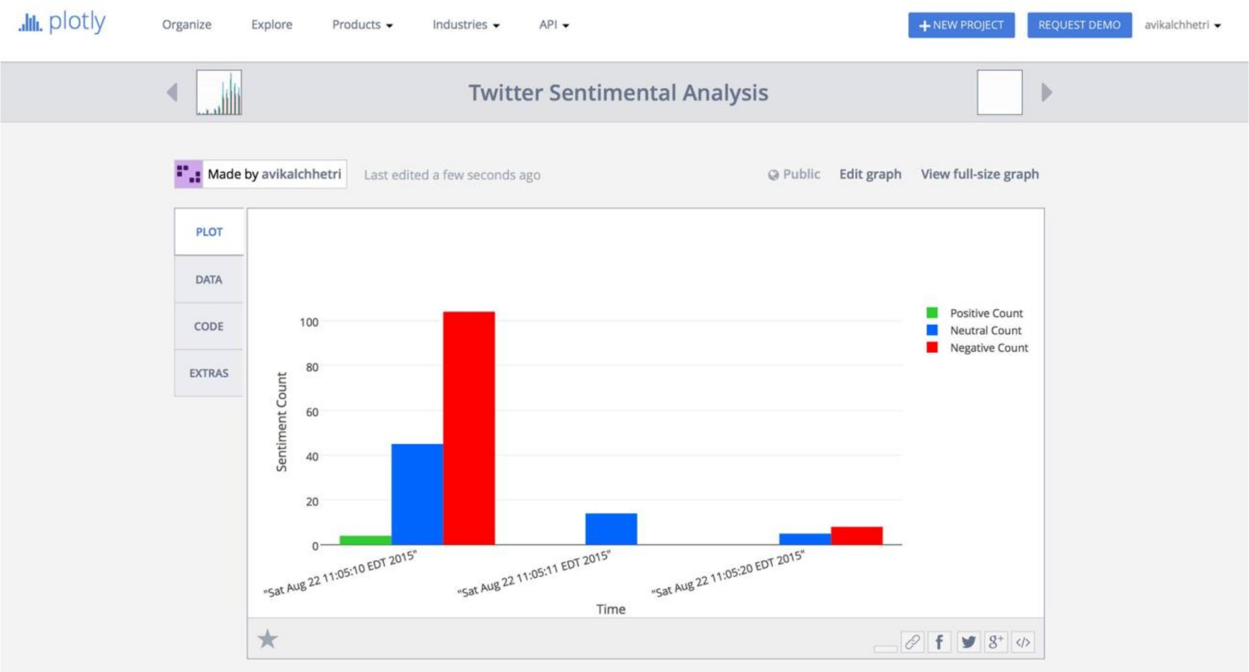
```
Adding annotator ssplit
Adding annotator pos
Adding annotator lemma
Adding annotator parse
Adding annotator sentiment
Adding annotator tokenize
Adding annotator ssplit
Adding annotator pos
Adding annotator lemma
Adding annotator parse
Adding annotator sentiment
Adding annotator tokenize
Adding annotator ssplit
Adding annotator pos
Adding annotator lemma
Adding annotator parse
Adding annotator sentiment
Adding annotator tokenize
Adding annotator ssplit
Adding annotator pos
Adding annotator lemma
Adding annotator parse
Adding annotator sentiment
[Stage 43:===============================>                                   (9 + 3) / 17] Ad
Adding annotator ssplit
Adding annotator pos
Adding annotator lemma
Adding annotator parse
Adding annotator sentiment
Adding annotator tokenize
Adding annotator ssplit
Adding annotator pos
Adding annotator lemma
Adding annotator parse
Adding annotator sentiment
Adding annotator tokenize
Adding annotator ssplit
Adding annotator pos
Adding annotator lemma
Adding annotator parse
Adding annotator sentiment
Adding annotator tokenize
Adding annotator ssplit
Adding annotator pos
Adding annotator lemma
Adding annotator parse
Adding annotator sentiment
Adding annotator tokenize
Adding annotator ssplit
Adding annotator pos
Adding annotator lemma
Adding annotator parse
Adding annotator sentiment

--
Time: 1440207010000 ms
-------------------------------------------
(Positive,1)
(Negative,12)
(Neutral,8)

> Building 85% > :consume
```

The data being pumped into MongoDB:

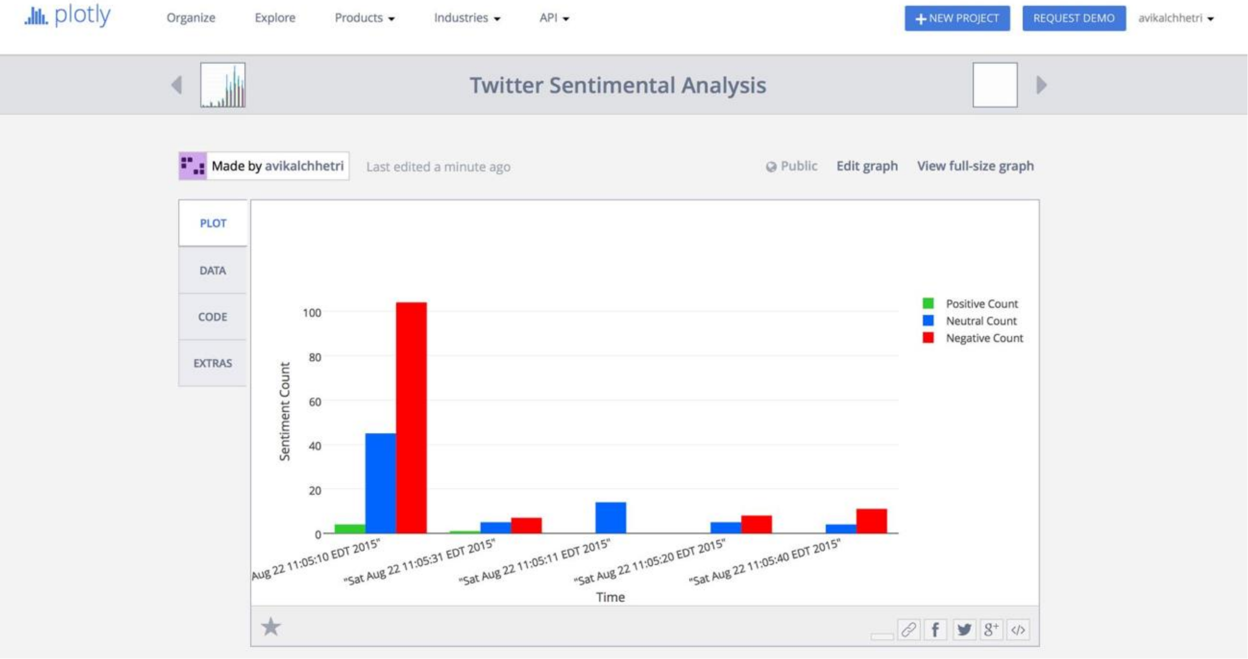


```
> db.twitter.find()
{ "_id" : ObjectId("55d7d166759f640df89ce23c"), "timestamp" : "Fri Aug 21 21:33:26 EDT 2015", "sentiment" : "Positive", "count" : 5, "flag" : "1" }
{ "_id" : ObjectId("55d7d166759f640df89ce23f"), "timestamp" : "Fri Aug 21 21:33:26 EDT 2015", "sentiment" : "Negative", "count" : 72, "flag" : "1" }
{ "_id" : ObjectId("55d7d166759f640df89ce23e"), "timestamp" : "Fri Aug 21 21:33:26 EDT 2015", "sentiment" : "Neutral", "count" : 40, "flag" : "1" }
{ "_id" : ObjectId("55d7d166759f640df89ce242"), "timestamp" : "Fri Aug 21 21:33:26 EDT 2015", "sentiment" : "Positive", "count" : 2, "flag" : "1" }
{ "_id" : ObjectId("55d7d167759f640df89ce244"), "timestamp" : "Fri Aug 21 21:33:27 EDT 2015", "sentiment" : "Negative", "count" : 14, "flag" : "1" }
{ "_id" : ObjectId("55d7d167759f640df89ce246"), "timestamp" : "Fri Aug 21 21:33:27 EDT 2015", "sentiment" : "Neutral", "count" : 8, "flag" : "1" }
{ "_id" : ObjectId("55d7d16a759f640df89ce249"), "timestamp" : "Fri Aug 21 21:33:30 EDT 2015", "sentiment" : "Negative", "count" : 6, "flag" : "1" }
{ "_id" : ObjectId("55d7d16a759f640df89ce24c"), "timestamp" : "Fri Aug 21 21:33:30 EDT 2015", "sentiment" : "Neutral", "count" : 1, "flag" : "1" }
{ "_id" : ObjectId("55d7d175759f640df89ce250"), "timestamp" : "Fri Aug 21 21:33:41 EDT 2015", "sentiment" : "Negative", "count" : 11, "flag" : "1" }
{ "_id" : ObjectId("55d7d175759f640df89ce252"), "timestamp" : "Fri Aug 21 21:33:41 EDT 2015", "sentiment" : "Neutral", "count" : 13, "flag" : "1" }
{ "_id" : ObjectId("55d7d17e759f640df89ce254"), "timestamp" : "Fri Aug 21 21:33:50 EDT 2015", "sentiment" : "Negative", "count" : 9, "flag" : "1" }
{ "_id" : ObjectId("55d7d17e759f640df89ce258"), "timestamp" : "Fri Aug 21 21:33:50 EDT 2015", "sentiment" : "Neutral", "count" : 4, "flag" : "1" }
{ "_id" : ObjectId("55d7d188759f640df89ce25b"), "timestamp" : "Fri Aug 21 21:34:00 EDT 2015", "sentiment" : "Negative", "count" : 11, "flag" : "1" }
{ "_id" : ObjectId("55d7d188759f640df89ce25d"), "timestamp" : "Fri Aug 21 21:34:00 EDT 2015", "sentiment" : "Neutral", "count" : 7, "flag" : "1" }
{ "_id" : ObjectId("55d7d193759f640df89ce261"), "timestamp" : "Fri Aug 21 21:34:11 EDT 2015", "sentiment" : "Negative", "count" : 15, "flag" : "1" }
{ "_id" : ObjectId("55d7d193759f640df89ce263"), "timestamp" : "Fri Aug 21 21:34:11 EDT 2015", "sentiment" : "Neutral", "count" : 17, "flag" : "1" }
{ "_id" : ObjectId("55d7d19d759f640df89ce266"), "timestamp" : "Fri Aug 21 21:34:21 EDT 2015", "sentiment" : "Negative", "count" : 14, "flag" : "1" }
{ "_id" : ObjectId("55d7d19d759f640df89ce26a"), "timestamp" : "Fri Aug 21 21:34:21 EDT 2015", "sentiment" : "Neutral", "count" : 10, "flag" : "1" }
{ "_id" : ObjectId("55d7d1a6759f640df89ce26c"), "timestamp" : "Fri Aug 21 21:34:30 EDT 2015", "sentiment" : "Positive", "count" : 1, "flag" : "1" }
{ "_id" : ObjectId("55d7d1a6759f640df89ce26e"), "timestamp" : "Fri Aug 21 21:34:30 EDT 2015", "sentiment" : "Negative", "count" : 14, "flag" : "1" }
Type "it" for more
> it
{ "_id" : ObjectId("55d7d1a6759f640df89ce271"), "timestamp" : "Fri Aug 21 21:34:30 EDT 2015", "sentiment" : "Neutral", "count" : 6, "flag" : "1" }
{ "_id" : ObjectId("55d7d1b2759f640df89ce274"), "timestamp" : "Fri Aug 21 21:34:42 EDT 2015", "sentiment" : "Negative", "count" : 6, "flag" : "1" }
{ "_id" : ObjectId("55d7d1b2759f640df89ce276"), "timestamp" : "Fri Aug 21 21:34:42 EDT 2015", "sentiment" : "Neutral", "count" : 8, "flag" : "1" }
{ "_id" : ObjectId("55d7d1ba759f640df89ce27c"), "timestamp" : "Fri Aug 21 21:34:50 EDT 2015", "sentiment" : "Neutral", "count" : 7, "flag" : "1" }
{ "_id" : ObjectId("55d7d1ba759f640df89ce27d"), "timestamp" : "Fri Aug 21 21:34:50 EDT 2015", "sentiment" : "Negative", "count" : 9, "flag" : "1" }
{ "_id" : ObjectId("55d7d1c4759f640df89ce27f"), "timestamp" : "Fri Aug 21 21:35:00 EDT 2015", "sentiment" : "Positive", "count" : 1, "flag" : "1" }
{ "_id" : ObjectId("55d7d1c4759f640df89ce280"), "timestamp" : "Fri Aug 21 21:35:00 EDT 2015", "sentiment" : "Negative", "count" : 7, "flag" : "1" }
{ "_id" : ObjectId("55d7d1c4759f640df89ce283"), "timestamp" : "Fri Aug 21 21:35:00 EDT 2015", "sentiment" : "Neutral", "count" : 6, "flag" : "1" }
{ "_id" : ObjectId("55d7d1cf759f640df89ce287"), "timestamp" : "Fri Aug 21 21:35:11 EDT 2015", "sentiment" : "Negative", "count" : 22, "flag" : "1" }
{ "_id" : ObjectId("55d7d1cf759f640df89ce28a"), "timestamp" : "Fri Aug 21 21:35:11 EDT 2015", "sentiment" : "Neutral", "count" : 7, "flag" : "1" }
```

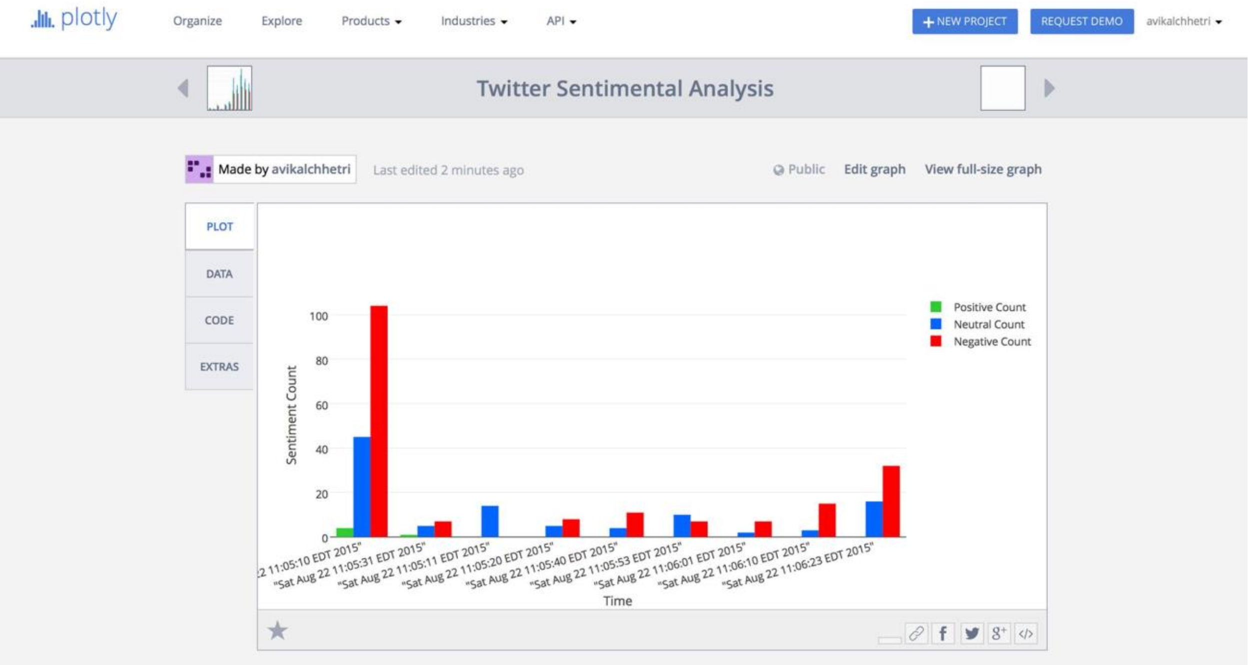
**After executing the python script to access the database and open up Plotly ,
Here you shall see a Streaming Bar graph which shows the sentiment of tweets happening every min/hour (depending on the time you set):**



After a few seconds…..



After a few MORE seconds…


plotly
Organize
Explore
Products ▾
Industries ▾
API ▾
+ NEW PROJECT
REQUEST DEMO
avikalchhetri ▾
Twitter Sentimental Analysis
Made by avikalchhetri
Last edited 2 minutes ago
Public
Edit graph
View full-size graph
PLOT
DATA
CODE
EXTRAS
Sentiment Count
0
20
40
60
80
100
🟩 Positive Count
🟦 Neutral Count
🟥 Negative Count
2 11:05:10 EDT 2015"
"Sat Aug 22 11:05:31 EDT 2015"
"Sat Aug 22 11:05:11 EDT 2015"
"Sat Aug 22 11:05:20 EDT 2015"
"Sat Aug 22 11:05:40 EDT 2015"
"Sat Aug 22 11:05:53 EDT 2015"
"Sat Aug 22 11:06:01 EDT 2015"
"Sat Aug 22 11:06:10 EDT 2015"
"Sat Aug 22 11:06:23 EDT 2015"
Time

Analysis and Justification for tools

Why this approach?

The standout observation from our approach is that we have streamed data from the database. We chose the idea to stream the data from the database (and not directly from Spark) as a foresight to have the need to do complex aggregate functions to show for special features during the visualizations phase and of course, keeping the history of the data. It is of course, possible to stream the data as soon as it loads to the database, by running the python script immediately after the consumer starts.

Why these tools?

KAFKA

Kafka is known one of the best ingestion tools used when you have ***a firehose of events*** (around 100k+/sec), which can easily be the case in our scenario, where number of tweets can reach that benchmark of 100k+/sec during a major event say like the Super Bowl finale. Also the 'atleast once' message guarantee helps. Last but not the least, it provides a seamless integration with Apache Spark.

Apache Spark

Spark's in memory computation makes tasks run 100x ***faste*r** than Hadoop MapReduce. Spark has ***rich support in Java, Scala, Python*** and growing libraries like ***MLlib*** and ***ML***. Spark can run in Hadoop ecosystem, EC2, Mesos or standalone cluster mode. The primary abstraction in Spark (RDD) are ***fault tolerant*** and can be operated in parallel. Spark streaming processes data in batches which is a powerful way of doing interactive analysis. As per our project, Spark is processing 1000s of live tweets in less than a second and counts their sentiments with seamless fault tolerance computation.

MongoDB

To be frank, we used MongoDB because we already ***know how it works beforehand***. Since we tend to store the sentiment values as BSON objects from Spark, a ***NoSQL database*** had to be chosen. Further, because of unstructured data, ***the data structure can be evolved overtime*** with no hassles. Also, MongoDB ***understands geo-spatial coordinates*** and natively supports geo-spatial indexing, which can be a further application in this project.

Plotly

The main takeaway from Plotly is that it is really ***user friendly to develop*** and mostly importantly: to ***SHARE***! It is free. It is easy to develop charts in ***python*** and can also be embedded in IPython Notebooks. There is no installation process required except for signing in with an id. Being the ***one of the mostly widely used tool*** among data scientists for sharing visualizations in recent times, it was an easy pick.

Why Spark Streaming and not Storm?

In terms of ***fault tolerance*** and ***data guarantees***, spark streaming provides better support as stateful computation is fault tolerant. Whereas in storm, each individual record has to be tracked through the system, so storm only guarantees that each record will be processed at least once, but allows duplicates to appear during recovery from a fault.

Storm is not capable of stateful operations, which are essential in making real-time decisions. Also with its dependency on additional components such as ZooKeeper and Cassandra, Storm is unable to look up dimension data, update an aggregate, or act directly on an event (that is, make real-time decisions).

Though Storm calculates data in real-time and Spark in near-real time, for the scope of our project ***near-real-time was good enough*** given the context, as absolute precision in timing is not necessary.

Other possible and viable solutions

PostgreSQL works well for storing millions of records instead of MongoDB. Kibana is a sophisticated analytics tool for visualizing and exploring the data but becomes a bit complicated with Elasticsearch feature. Apache Storm is a popular tool for developing streaming application as it processes data in real time and it's integration with Kafka for data ingestion. The Storm framework is designed to move data from sources to user processing in a horizontally scalable and failure-tolerant way. It provides at least once or at most once ingestion semantics and it has the power to restart work if processes fail.

Further applications possible with this project

Given our time constraint not all special features were possible to be added.

But, here are anyways the further possible applications:

- Geo-spatial maps could be plotted, signifying the location (and sentiment) of the tweets.
- The top retweeted/favorited tweets from a specific sentiment category
- Find the opinions/sentiment of the top influential people (people with a lot of followers) on a particular topic/hashtag.

Lessons Learned And Challenges Faced

- In Spark Streaming, one has to be really careful while using data taken from the streaming context. Operations performed on the RDDs should be done using *transform* and *forEachRDD* functions.
- In Spark, transforming the DStreams into Data Frames can be quite a pain and one has to take care to maintain the schema properly.
- In Spark, while inserting to the database using BSON objects only string values can be stored. This can cause problems while aggregating on those values from the database and hence further unnecessary cast functions would be needed to be applied on the database.

Conclusion

Hence we were successful in building an infrastructure to analyze the sentiments of tweets streamed based on a given hashtag.