PROYECTO FIN DE MASTER "Distribución de prensa a través de análisis tendencias de Twitter"

Alberto Conde Moreno

https://github.com/aCondeMor/twitterSparkMagazines

1.- Planteamiento del problema

Uno de los mayores retos de la distribución (no exclusivamente de la de prensa) es el concepto básico de todo negocio, conseguir el mayor número de ventas con el menor gasto posible.

En el caso de la distribución de prensa, conseguir este objetivo se complica al tener en cuenta, que no se contabiliza las ventas como tal, sino que son un cálculo de los ejemplares que se distribuyen menos los que han sido devueltos. Esto quiere decir que, si yo hoy distribuyo una revista, en este momento mis ventas son del 100%, y hasta que no empiece a recibir devolución de los clientes no tendré ningún tipo de información al respecto. Por ello se establecen unos plazos de devolución a partir de los cuales no se permite rehusar ningún ejemplar más, en ese momento, es cuando conoceremos nuestras ventas reales. Con esto se plantea la siguiente situación, de media, no conozco mis ventas cerradas hasta pasados 2-3 meses de la salida al mercado.

Teniendo en cuenta esta información, llega el segundo problema, donde será interesante mi producto para el cliente y donde no. Basándonos en el problema anterior, tenemos en cuenta que una revista semanal que lanzo ahora, no tendré conocimiento de sus ventas hasta dentro de unos 2-3 meses, esto implica que pasaran entre 8 y 12 números desde que se sale al mercado, lo que implican miles de ejemplares repartidos durante mucho tiempo en zonas en las que no resultan interesantes.

2.- Planteamiento de una solución

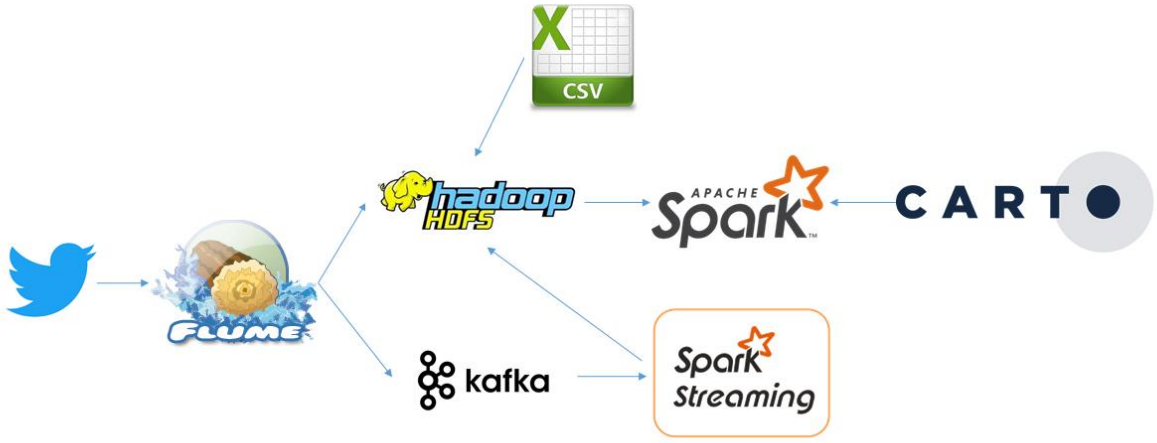
Para intentar solventar esta situación se plantea la opción de utilizar las redes sociales, en concreto la red de Twitter, cuya Api representa una gran fuente de información en Streaming.

La intención no es otra que hacer un seguimiento de las tendencias relacionadas con nuestra revista, días antes de la distribución de la misma para intentar socavar la mayor información de localizar nuestro público target y con ello abastecer los puntos de venta más cercanos a estas zonas de calor.

De esta manera, una vez extraídos los twitters, los compararemos con nuestro histórico de ventas por punto de venta, de forma que localizaremos los puntos que no venden ejemplares de la revista y que actualmente tienen servicio, y recuperaremos esos ejemplares para distribuirlos en los puntos que no tienen servicio en las zonas calientes de tendencia en Twitter, intentando que nuestro sondeo de nuevas ventas sea lo más fructífero posible.

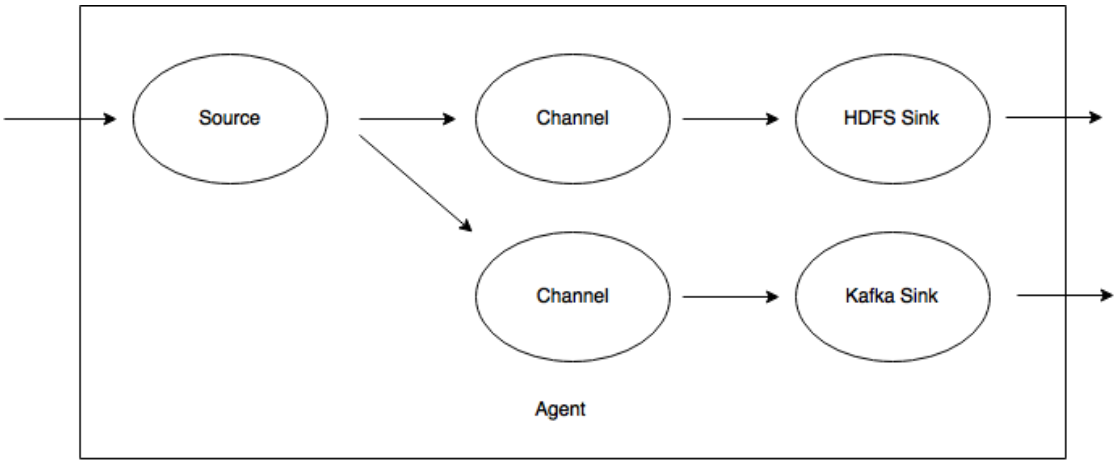
Pero y como se puede hacer esto…..

3.- Arquitectura



Para la solución utilizaremos una arquitectura Lambda en la que tendremos los siguientes componentes:

- **Flume:** lo utilizaremos como herramienta para recoger los datos de Twitter.



Recoge los datos de la fuente Twitter Api y los devuelve por dos sumideros HDFS y por otro de Kafka utilizando la siguiente configuración.

```
# Sources, channels and sinks are defined per agent,
# in this case called 'twitter'
```

```
# Name the components on this agent
twitter.sources = tweet
twitter.sinks = HDFS Kafka
twitter.channels = mem2 mem1
twitter.sources.tweet.channels = mem2 mem1
twitter.sinks.HDFS.channel = mem1
twitter.sinks.Kafka.channel = mem2

twitter.channels.mem1.type = memory
twitter.channels.mem1.capacity = 1000
twitter.channels.mem1.transactionCapacity = 1000

twitter.channels.mem2.type = memory
twitter.channels.mem2.capacity = 1000
twitter.channels.mem2.transactionCapacity = 1000

twitter.sources.tweet.type = com.cloudera.flume.source.TwitterSource
twitter.sources.tweet.consumerKey = <consumerKey>
twitter.sources.tweet.consumerSecret = <consumerSecret>
twitter.sources.tweet.accessToken = <accesToken>
twitter.sources.tweet.accessTokenSecret = <accessTokenSecret>
twitter.sources.tweet.keywords = @glamourmag,@glamour_fashion, glamourmag, fashion
twitter.sources.tweet.languages=en,En


twitter.sinks.HDFS.type = hdfs
twitter.sinks.HDFS.hdfs.useLocalTimeStamp = true
twitter.sinks.HDFS.hdfs.path = hdfs://localhost:9000/flume/raw_data/%Y/%m/%d/%H/
twitter.sinks.HDFS.hdfs.fileType = DataStream
twitter.sinks.HDFS.hdfs.writeFormat = Text
twitter.sinks.HDFS.hdfs.batchSize = 1000
twitter.sinks.HDFS.hdfs.rollSize = 0
twitter.sinks.HDFS.hdfs.rollCount = 10000

twitter.sinks.Kafka.type = org.apache.flume.sink.kafka.KafkaSink
twitter.sinks.Kafka.kafka.topic = twitter
twitter.sinks.Kafka.kafka.bootstrap.servers = localhost:9092
```

- **HDFS (Hadoop):** En HDFS se almacenan los datos puros tal cual se obtienen de Flume.

```
hdfs://localhost:9000/flume/raw_data/%Y/%m/%d/%H/
```

- **Kafka y Zookeeper:** Se utiliza kafka y zookeeper como cola de mensajes para pasar los mensajes desde Flume hasta Spark Streaming.
  bootstrap.servers = localhost:9092

- **Spark Streaming:** Procesa los datos streaming desde Twitter, comprueba si tienen coordenadas de geoposición, y si no las tiene, le añade unas generadas aleatoriamente en un radio aproximado que comprende la Comunidad de Madrid para nuestro ejemplo.
  Estos datos una vez recogidos todos los datos los almacena en una carpeta de HDFS.

- **Spark:** Recoge tanto el fichero csv con el histórico de los puntos de venta como los ficheros Part generados por Spark Streaming con los datos de la geoposición de los tweets escritos de HDFS. Crea un listado con los puntos que no tienen servicio de la revista y los compara con los datos de geoposicion agrupados por zonas, y recoge los puntos de venta más cercanos a estas zonas, a los cuales se les pondrá servicio de la revista.

  Finalmente se generan dos ficheros csv, uno con los datos de coordenadas de los tweets, y otro con los datos de coordenadas de los puntos de venta seleccionados.

- **Carto:** se cargan en Carto los ficheros CSV con las coordenadas donde se pueden visualizar en el Mapa de Madrid.

4 .- Codigo fuente Spark
-Spark Streaming

```
from __future__ import print_function
from pyspark import SparkContext
from pyspark.streaming import StreamingContext
from pyspark.streaming.kafka import KafkaUtils
import json
import random
import time


# Random generator of Madrid long. coordinates
def coord_long():


   longdec = random.uniform(-0.353348, -0.955310)

   longitud = -3 + longdec
   longitud = float("{0:.6f}".format(longitud))

   return longitud

# Random generator of Madrid lat. coordinates
def coord_latitud():
   latdec = random.uniform(0.184333, 0.602099)

   latitud = 40 + latdec
   latitud = float("{0:.6f}".format(latitud))
   return latitud


# Create Spark context
sc = SparkContext(appName="PythonStreamingDirectKafka")

# Create Streaming context
ssc = StreamingContext(sc, 10)

# Getting data from Kafka and topic "twitter"
kafkaStream = KafkaUtils.createStream(ssc, 'localhost:2181', 'spark-streaming', {'twitter':1})


# Parse the inbound message as json
parsed = kafkaStream.map(lambda v: json.loads(v[1]))

#Getting from Json the following data (user, geo coordinates, text)
# if geo coordinates is null, we generate it with the random coordinates generator
dataStream = parsed.map(lambda tweet: (tweet['geo']['coordinates'] if tweet['geo'] else coord_latitud(),tweet['geo']['coordinates'] if tweet['geo'] else coord_long()))
```

```
#Saving in HDFS
path = "hdfs://localhost:9000/spark/streaming/test/"
dataStream.repartition(1).saveAsTextFiles(path + str(time.time()))

ssc.start()
ssc.awaitTermination()
```

- Spark Batch

```
import os.path

from pyspark import SparkContext
from pyspark import sql
from pyspark.sql import Row
from pyspark.sql.types import *
import random



inputPathCSV = ('hdfs://localhost:9000/spark')
inputFileName = ('Glamour.csv')
inputStreamFile = ("hdfs://localhost:9000/spark/streaming/*/")

#create Spark context
sc = SparkContext(appName="distribution")

# getting from the file in HDFS the selling points for Madrid and Barcelona with sales for Glamour
# points with -1 copies received are not being served
sellingPoints = os.path.join(inputPathCSV, inputFileName)


#getting all the twitter coordinates from HDFS
twitter = sc.textFile(inputStreamFile)

# Random generator of Barcelona long. coordinates
def barc_long():


    longdec = random.uniform(0.0, 0.7)

    longitud = 1.73485 + longdec
    longitud = float("{0:.6f}".format(longitud))

    return longitud

# Random generator of Barcelona lat. coordinates
def Barc_latitud():

    latdec = random.uniform(0.216943, 0.770186)

    latitud = 41 + latdec
    latitud = float("{0:.6f}".format(latitud))
    return latitud
```

```
# Random generator of Madrid long. coordinates
def Mad_long():


    longdec = random.uniform(-0.353348, -0.955310)

    longitud = -3 + longdec
    longitud = float("{0:.6f}".format(longitud))

    return longitud

# Random generator of Madrid lat. coordinates
def Mad_latitud():
    latdec = random.uniform(0.184333, 0.602099)

    latitud = 40 + latdec
    latitud = float("{0:.6f}".format(latitud))
    return latitud



# Load a text file and convert each line to a Row.
# if the province is "Madrid" we generate coordinates of Madrid else the Barcelona ones
def toRow(entry):
    lines = sc.textFile(entry)
    parts = lines.map(lambda l: l.split(";"))
    rdd = parts.map(lambda p: Row(sellPoint=int (p[0]), province=p[1], \
                        zipcode=int (p[2]), copiesDel=int(p[3]), unsolds=int(p[4]), \
                        sales=int(p[5]),latitud = float (Mad_latitud() if p[1]=='MADRID' else
Barc_latitud()),longitud = float (Mad_long() if p[1]=='MADRID' else barc_long())))
    return rdd

# Load the twitter data with the coordinates, taking care of the format of the string coordinates
def toRowTwitter(lines):
    parts = lines.map(lambda l: l.split(","))
    rdd = parts.map(lambda p: Row(latitud= float (p[0][1:len(p[0])].strip('[')), longitud = float
(p[1][0:len(p[1])-1].strip(']'))))
    print rdd.collect()
    return rdd


#count the number of copies in selling points with zero sales
def nonSalesCount(rdd):
  return  rdd.map(lambda x: (x.sales, x.copiesDel)).filter(lambda s: (s[0] == 0))\
  .reduceByKey(lambda a,b:a+b).map(lambda f: (f[1])).take(1)


#list of selling points with zero sales
def nonSalesPoints(rdd):
  return  rdd.filter(lambda s: (s.sales == 0))

# list of available selling points in (Key,Value) format and key rounded to 3 decimals; the value is a
{dictionary}
def availablePoints(rdd):
  return rdd.filter(lambda s: (s.sales == -1)).map(lambda x:
(("{0:.3f}".format(x.latitud))+("{0:.3f}".format(x.longitud)),{ 'sellPoint': x.sellPoint,
'longitud':x.longitud,'latitud':x.latitud}))
```

```
#list of the selling points with the magazine that sales more than zero
def salesPoints(rdd):
  return  rdd.filter(lambda s: (s.sales >0))


#join of the twitter points and the available points rounded to 3 decimals
def searchSellingPoints(twitterPoints, availablePoints):
   roundTweet = twitterPoints.map(lambda x:
("{0:.3f}".format(x.latitud),"{0:.3f}".format(x.longitud))).map(lambda x: (x[0]+x[1],1))\
   .reduceByKey(lambda a,b:a+b).sortBy(lambda(k,v): -v)

   roundedAvailable = availablePoints.map (lambda x: (x[0],x[1]))
   sellPointsDistributed = roundedAvailable.join(roundTweet)
   return sellPointsDistributed


#list of the selling points with the magazine that sales more than zero
actualMagPoints =toRow(sellingPoints)

#getting an RDD with all the selling points
allPoints = toRow(sellingPoints)

#getting an RDD witl all the twitter points
twitterPoints = toRowTwitter(twitter)

#getting an RDD with all the selling points that actually are not selling the magazine
newPoints = availablePoints(allPoints)


# comparing the selling points with the twitter points for getting the better points
finalList = searchSellingPoints(twitterPoints,newPoints)

# parsing from Row to list
final = finalList.map(lambda x: (x[1])).map(lambda y: (y[0]['longitud'],y[0]['latitud'], y[0]['sellPoint']))
print ('Listado total de puntos a distribuir:')
print final.collect()
print final.count()

# and the twitter points to a list too
outputtweets = twitterPoints.map(lambda x: (x.longitud, x.latitud))


# finally create s SQL dataframe for save as csv file for use the data in Carto
customSchema = StructType([ \
   StructField("long", FloatType(), True), \
   StructField("lat", FloatType(), True)])


sqlContext = sql.SQLContext(sc)
df = sqlContext.createDataFrame(outputtweets, customSchema)
df.coalesce(1).write.option("header", "true").csv("hdfs://localhost:9000/spark/localtwitter.csv")


finalPoints = final.map(lambda x: (x[0],x[1]))
df = sqlContext.createDataFrame(finalPoints, customSchema)
df.coalesce(1).write.option("header", "true").csv("hdfs://localhost:9000/spark/New_Selling_Points.csv")
```

4.- Datos de prueba utilizados

Los datos que se utilizan en la arquitectura son básicamente una serie de Json con esquema de Twitter y un csv con los datos del histórico de ventas:

- CSV con histórico de ventas

  Cuyo esquema sería el siguiente:

  Cod. Punto Venta; Provincia; C. Postal; Ej. Distribuidos; Ej. Devueltos; Venta

39438;MADRID;28045;6;6;0
39484;MADRID;28033;2;2;0
39502;MADRID;28023;2;2;0
39515;MADRID;28046;2;2;0
39525;MADRID;28220;2;2;0
39539;MADRID;28691;2;2;0
39550;MADRID;28450;2;2;0
39563;MADRID;28804;1;1;0
39570;MADRID;28230;2;2;0
39574;MADRID;28922;2;2;0
39586;MADRID;28023;2;2;0
39592;MADRID;28700;2;2;0
39593;MADRID;28411;2;2;0
39609;MADRID;28230;2;2;0
39673;MADRID;28914;2;2;0
39779;MADRID;28028;2;2;0

No se ha introducido también en el fichero las coordenadas de los puntos de venta por no disponer de ellas, por lo que al igual que con los datos de Twitter, se generan aleatoriamente en la zona de Madrid.

- Schema Json Twitter


```
{
        "text": "RT @PostGradProblem: In preparation for the NFL lockout, I
    will be spending twice as much time analyzing my fantasy baseball team
    during ...",
        "truncated": true,
        "in_reply_to_user_id": null,
        "in_reply_to_status_id": null,
        "favorited": false,
        "source": "<a href=\"http://twitter.com/\" rel=\"nofollow\">Twitter
    for iPhone</a>",
        "in_reply_to_screen_name": null,
        "in_reply_to_status_id_str": null,
```

```
      "id_str": "54691802283900928",
      "entities": {
            "user_mentions": [
                  {
                        "indices": [
                              3,
                              19
                        ],
                        "screen_name": "PostGradProblem",
                        "id_str": "271572434",
                        "name": "PostGradProblems",
                        "id": 271572434
                  }
            ],
            "urls": [ ],
            "hashtags": [ ]
      },
      "contributors": null,
      "retweeted": false,
      "in_reply_to_user_id_str": null,
      "place": null,
      "retweet_count": 4,
      "created_at": "Sun Apr 03 23:48:36 +0000 2011",
      "retweeted_status": {
            "text": "In preparation for the NFL lockout, I will be spending
twice as much time analyzing my fantasy baseball team during company time.
#PGP",
            "truncated": false,
            "in_reply_to_user_id": null,
            "in_reply_to_status_id": null,
            "favorited": false,
            "source": "<a href=\"http://www.hootsuite.com\"
rel=\"nofollow\">HootSuite</a>",
            "in_reply_to_screen_name": null,
            "in_reply_to_status_id_str": null,
            "id_str": "54640519019642881",
            "entities": {
                  "user_mentions": [ ],
                  "urls": [ ],
                  "hashtags": [
                        {
                              "text": "PGP",
                              "indices": [
                                    130,
                                    134
                              ]
```

```
                        }
                    ]
                },
                "contributors": null,
                "retweeted": false,
                "in_reply_to_user_id_str": null,
                "place": null,
                "retweet_count": 4,
                "created_at": "Sun Apr 03 20:24:49 +0000 2011",
                "user": {
                    "notifications": null,
                    "profile_use_background_image": true,
                    "statuses_count": 31,
                    "profile_background_color": "C0DEED",
                    "followers_count": 3066,
                    "profile_image_url":
"http://a2.twimg.com/profile_images/1285770264/PGP_normal.jpg",
                    "listed_count": 6,
                    "profile_background_image_url":
"http://a3.twimg.com/a/1301071706/images/themes/theme1/bg.png",
                    "description": "",
                    "screen_name": "PostGradProblem",
                    "default_profile": true,
                    "verified": false,
                    "time_zone": null,
                    "profile_text_color": "333333",
                    "is_translator": false,
                    "profile_sidebar_fill_color": "DDEEF6",
                    "location": "",
                    "id_str": "271572434",
                    "default_profile_image": false,
                    "profile_background_tile": false,
                    "lang": "en",
                    "friends_count": 21,
                    "protected": false,
                    "favourites_count": 0,
                    "created_at": "Thu Mar 24 19:45:44 +0000 2011",
                    "profile_link_color": "0084B4",
                    "name": "PostGradProblems",
                    "show_all_inline_media": false,
                    "follow_request_sent": null,
                    "geo_enabled": false,
                    "profile_sidebar_border_color": "C0DEED",
                    "url": null,
                    "id": 271572434,
                    "contributors_enabled": false,
```

```
                "following": null,
                "utc_offset": null
            },
            "id": 54640519019642880,
            "coordinates": null,
            "geo": null
        },
        "user": {
            "notifications": null,
            "profile_use_background_image": true,
            "statuses_count": 351,
            "profile_background_color": "C0DEED",
            "followers_count": 48,
            "profile_image_url":
"http://a1.twimg.com/profile_images/455128973/gCsVUnofNqqyd6tdOGevROvko1_500_normal.jpg",
            "listed_count": 0,
            "profile_background_image_url":
"http://a3.twimg.com/a/1300479984/images/themes/theme1/bg.png",
            "description": "watcha doin in my waters?",
            "screen_name": "OldGREG85",
            "default_profile": true,
            "verified": false,
            "time_zone": "Hawaii",
            "profile_text_color": "333333",
            "is_translator": false,
            "profile_sidebar_fill_color": "DDEEF6",
            "location": "Texas",
            "id_str": "80177619",
            "default_profile_image": false,
            "profile_background_tile": false,
            "lang": "en",
            "friends_count": 81,
            "protected": false,
            "favourites_count": 0,
            "created_at": "Tue Oct 06 01:13:17 +0000 2009",
            "profile_link_color": "0084B4",
            "name": "GG",
            "show_all_inline_media": false,
            "follow_request_sent": null,
            "geo_enabled": false,
            "profile_sidebar_border_color": "C0DEED",
            "url": null,
            "id": 80177619,
            "contributors_enabled": false,
            "following": null,
```

```
            "utc_offset": -36000
        },
        "id": 54691802283900930,
        "coordinates": null,
        "geo": null
}
```


- **Problemas localizados**

    - Inicialmente los primeros problemas fueron en el planteamiento, al no recibir los datos de Twitter con geoposición, por lo que se opta por generarlos aleatoriamente.

    - Posteriormente aparecen los problemas para escoger la distribución. Como punto de partida intento realizar el proyecto bajo la distribución de Cloudera en una máquina virtual, pero los escasos 4 GB de memoria ram de mi PC me lo impiden, por lo que trato de utilizar Dockers para simular la distribución de cloudera en ellos, pero nuevamente la memoria Ram me hace descartar esta opción, por lo que finalmente opto por realizar el proyecto en Local.

    - Intento generar los datos de coordenadas con Json-Generator, pero no consigo generar los datos por este medio, por lo que finalmente realizo una función para generar las coordenadas de Madrid.

    - Una vez generados los datos finales con Spark me planteo la mejor manera de trasladarlos a visualización. Me planteo inicialmente el uso de Tableau pero corre en Windows por lo que finalmente opto por utilizar Carto. Al no poder conectar los datos a Carto en su versión gratuita a través de una BBDD, descarto conectar los mismo a una BBDD. En caso de tener interés en hacerlo en un futuro se podría realizar sencillamente a través de Hive por medio del HDFS.

- **Conclusiones**

Las conclusiones que obtengo del proyecto es que obviamente por los datos de entrada ficticios, ya que los datos generados por twitter con geoposición no son los suficientes como para poder extraer la información necesaria y poder realizar una comparativa real y fehaciente , actualmente la solución no es funcional.

Sin embargo, si se me plantea la posibilidad de que podrían utilizarse las aplicaciones móviles de los editores de las revistas para extraer los datos de geoposición de las personas que leen sus artículos a través de ellas, y así realizar el proyecto con esa fuente de información en vez de a través de Twitter.