Implementing DataWeave's Social API for Social Data Analysis

4th Aug, 2015

BY DATAWEAVE

In today's world, the analysis of any social media stream can reap invaluable information about, well, pretty much everything. If you are a business catering to a large number of consumers, it is a very important tool for understanding and analyzing the market's perception about you, and how your audience reacts to whatever you present before them.

At DataWeave, we sat down to create a setup that would do this for some e-commerce stores and retail brands. And the first social network we decided to track was the micro-blogging giant, Twitter. Twitter is a great medium for engaging with your audience. It's also a very efficient marketing channel to reach out to a large number of people.

Data Collection

The very first issue that needs to be tackled is collecting the data itself. Now quite understandably, Twitter protects its data vigorously. However, it does have a pretty **solid REST API** for data distribution purposes too. The API is simple, nothing too complex, and returns data in the easy to use JSON format. Take a look at the **timeline API**, for example. That's quite straightforward and has a lot of detailed information.

The issue with the Twitter API however, is that it is seriously rate limited. Every function can be called in a range of 15–180 times in a 15-minute window. While this is good enough for small projects not needing much data, for any real-world application however, these rate limits can be really frustrating. To

avoid this, we used the **Streaming API**, which creates a long-lived HTTP GET request that continuously streams tweets from the public timeline.

Also, Twitter seems to suddenly return null values in the middle of the stream, which can make the streamer crash if we don't take care. As for us, we simply threw away all null data before it reached the analysis phase, and as an added precaution, designed a simple e-mail alert for when the streamer crashed.

Data Storage

Next is data storage. Data is traditionally stored in tables, using RDBMS. But for this, we decided to use MongoDB, as a document store seemed quite suitable for our needs. While I didn't have much clue about MongoDB or what purpose it's going to serve at first, I realized that is a seriously good alternative to MySQL, PostgreSQL and other relational schema-based data stores for a lot of applications.

Some of its advantages that I very soon found out were: documents-based data model that are very easy to handle analogous to Python dictionaries, and support for expressive queries. I recommend using this for some of your DB projects. You can play about with it **here.**

Data Processing

Next comes data processing. While data processing in MongoDB is simple, it can also be a hard thing to learn, especially for someone like me, who had no experience anywhere outside SQL. But MongoDB queries are simple to learn once the basics are clear.

For example, in a DB DWSocial with a collection tweets, the syntax for getting all tweets would be something like this in a Python environment:

```
rt = list(db.tweets.find())
```

The list type-cast here is necessary, because without it, the output is simply a MongoDB reference, with no value. Now, to find all tweets where user_id is 1234, we have

```
rt = list(db.retweets.find({ 'user_id': 1234 })
```

Apart from this, we used regexes to detect specific types of tweets, if they were, for example, "offers", "discounts", and "deals". For this, we used the Python re library, that deals with regexes. Suffice is to say, my reaction to regexes for the first two days was much like

Once again, its just initial stumbles. After some (okay, quite some) help from Thothadri, Murthy and Jyotiska, I finally managed a basic parser that could detect which tweets were offers, discounts and deals. A small code snippet is here for this purpose.

```
def deal(id):

re_offers = re.compile(r'''

\b

(?:

deals?

|

offers?

|

discount

|

promotion

|

sale

|

rs?

|

rs\?

|

inr\s*([\d\.,])+

|

([\d\.,])+\s*inr

)

\b

|

\b\d+%
```

```
|

\$\d+\b

''',

re.I|re.X)

x = list(tweets.find({'user_id' : id,'created_at': { '$gte': fourteen_days_ago }}))

mylist = []

newlist = []

for a in x:

b = re_offers.findall(a.get('text'))

if b:

print a.get('id')

mylist.append(a.get('id'))

w = list(db.retweets.find( { 'id' : a.get('id') } ))

if w:

mydict = {'id' : a.get('id'), 'rt_count' : w[0].get('rt_count'), 'text' : a.get('tex
t'), 'terms' : b}

else:

mydict = {'id' : a.get('id'), 'rt_count' : 0, 'text' : a.get('text'), 'terms' : b}

track.insert(mydict)
```

This is much less complicated than it seems. And it also brings us to our final step–integrating all our queries into a REST-ful API.

Data Serving

For this, mulitple web-frameworks are available. The ones we did consider were **Flask**, **Django** and **Bottle**.

Weighing the pros and cons of every framework can be tedious. I did find this awesome presentation on slideshare though, that succinctly summarizes each framework. You can go through it here.

We finally settled on Bottle as our choice of framework. The reasons are simple. Bottle is monolithic, i.e., it uses the one-file approach. For small applications, this makes for code that is easier to read and maintainable.

Some sample web address routes are shown here:

#show all tracked accounts

```
id_legend = {57947109 : 'Flipkart', 183093247: 'HomeShop18', 89443197: 'Myntra', 43133
6956: 'Jabong'}

@route('/ids')

  def get_ids():

    result = json.dumps(id_legend)

    return result
```

#show all user mentions for a particular account @route('/user_mentions')

```
def user_mention():

  m = request.query.id

  ac_id = int(m)

  t = list(tweets.find({'created_at': { '$gte': fourteen_days_ago }, 'retweeted': 'n
o', 'user_id': { '$ne': ac_id} }))

  a = len(t)

  mylist = []

  for i in t:

    mylist.append({i.get('user_id'): i.get('id')})

  x = { 'num_of_mentions': a, 'mentions_details': mylist }

  result = json.dumps(x)

  return result
```

This is how the DataWeave Social API came into being. I had a great time doing this, with special credits to Sanket, Mandar and Murthy for all the help that they gave me for this. That's all for now, folks!

Originally published at **blog.dataweave.in.**

- DataWeave Marketing

4th Aug, 2015

API