Data mining with Ruby and Twitter

The interesting side to a Twitter API

M. Tim Jones
Independent author

11 October 2011
(First published 04 October 2011)

Twitter is not only a fantastic real-time social networking tool, it's also a source of rich information that's ripe for data mining. On average, Twitter users generate 140 million tweets per day on a variety of topics. This article introduces you to data mining and demonstrates the concept with the object-oriented Ruby language.

In October 2008, like many others, I created a Twitter account out of curiosity. Like most people, I connected with friends and did some random searching to better understand the service. Communicating at 140 characters didn't seem like an idea that would be popular. An unrelated event helped me understand Twitter's real value.

In early July 2009, my web-hosting provider went dark. After random web searching, I found information pointing to a fire in Seattle’s Fisher Plaza as the culprit. Information from traditional web-based sources was slow and gave no indication of when the service might return. However, after searching Twitter, I found personal accounts of the incident, including real-time information on what was happening at the scene. For example, shortly before my hosting service returned, there was a tweet indicating that diesel power generators were outside the building.

This was when I realized that the true power of Twitter is open and real-time communication of information among individuals and groups. Yet, under the surface, it is a treasure trove of information about behaviors of the users, and trends at the local and global levels. I explore this realization in the context of simple scripts using the Ruby language and the Twitter gem, an API wrapper for Twitter. I also demonstrate how to build simple mashups for data visualization using other web services and applications.

Ruby knowledge

If you do not have basic knowledge of the wonderful Ruby language, find references in the Resources section. These examples demonstrate the value of Ruby and its ability to encode a significant amount of power in a limited number of source lines of code.
Twitter and APIs

Although the early web was about human-machine interaction, today's web is about machine-machine interaction, enabled using web services. These services exist for most popular websites—from various Google services to LinkedIn, Facebook, and Twitter. Web services create APIs through which external applications can query or manipulate content on websites.

Web services are implemented using a number of styles. Today, one of the most popular is Representational State Transfer, or REST. One implementation of REST is over the well-known HTTP protocol, allowing HTTP to exist as a medium for a RESTful architecture (using standard HTTP operations like GET, PUT, POST, and DELETE). The API for Twitter is developed as an abstraction over this medium. In this way, there's no knowledge of REST, HTTP, or data formats like XML or JSON, but instead an object-based interface that integrates cleanly into the Ruby language.

A quick tour of Ruby and Twitter

Let's explore how you can use the Twitter API with Ruby. First, we need to get the necessary resources. If like me you're using Ubuntu Linux®, you use the `apt` framework.

To get the latest full Ruby distribution (approximately a 13MB download), use this command line:

```
$ sudo apt-get install ruby1.9.1-full
```

Next, grab the Twitter gem using the `gem` utility:

```
$ sudo gem install twitter
```

You now have everything you need for this step, so let's continue with a test of the Twitter wrapper. For this demonstration, use a shell called the Interactive Ruby Shell (IRB). This shell allows you to execute Ruby commands and experiment with the language in real time. IRB has a large number of capabilities, but we'll use it for simple experimentation.

Listing 1 shows a session with IRB that has been broken into three sections to aid readability. The first section (lines 001 and 002) simply prepares the environment by importing the necessary run time elements (the `require` method loads and executes the named library). The next line (003) demonstrates the use of the Twitter gem to display the most recent tweet from IBM® developerWorks®. As shown, you use the `user_timeline` method of the `Client::Timeline` module to display a tweet. This first example demonstrates the "chain methods" capability of Ruby. The `user_timeline` method returns an array of 20 tweets that you chain into the `first` method. Doing so extracts the first tweet from the array (`first` is a method of the `Array` class). From this single tweet, you extract the text field emitted to output via `puts`.

The next section (line 004) uses the user-defined location field, a free-form field into which the user can provide both useful and non-useful location information. In this example, the `user` module grabs user information, constrained with the location field.
The final section (from line 005) explores the `Twitter::Search` module. The search module provides an extremely rich interface with which to search Twitter. In this example, you first create a search instance (line 005), then specify a search at line 006. You're searching for the most recent tweets containing the word `why` that are directed to the LulzSec user. The resulting list has been reduced and edited. Searches are sticky in that the search instance maintains the defined filters. You can clear these filters by executing `search.clear`.

**Listing 1. Experimenting with the Twitter API through IRB**

```ruby
$ irb
irb(main):001:0> require "rubygems"
=> true
irb(main):002:0> require "twitter"
=> true

irb(main):003:0> puts Twitter.user_timeline("developerworks").first.text
 dow Twitter is saving $600K per month: will #Google+ add to that? >
http://t.co/HiRwir7 #Tech #webdesign #Socialmedia #webapp #app
=> nil

irb(main):004:0> puts Twitter.user("MTimJones").location
Colorado, USA
=> nil

irb(main):005:0> search = Twitter::Search.new
=> #<Twitter::Search::Twitter::Search:0xb7437e04 @oauth_token_secret=nil,
@endpoint="https://api.twitter.com/1/",
@user_agent="Twitter Ruby Gem 1.6.0",
@oauth_token=nil, @consumer_secret=nil,
@search_endpoint="https://search.twitter.com/",
@query={:tude=>[], :q=>[]}>, @cache=nil, @gateway=nil, @consumer_key=nil,
@proxy=nil, @format=:json, @adapter=:net_http>

irb(main):006:0> search.containing("why").to("LulzSec").result_type("recent").each do |r| puts r.text end
@LulzSec why not stop posting <bleep> and get a full time job! MYSQLi isn't hacking you <bleep>.
...

irb(main):007:0>
```

Next, let's look at the schema for a user in Twitter. You can also do this through IRB, but I'll reformat the result to illustrate more simply the anatomy of a Twitter user. **Listing 2** shows the result of printing the user structure, which in Ruby is a `Hashie::Mash`. This structure is useful, because it permits an object to have method-like accessors for hash keys (an open object). As you can see from **Listing 2**, this object contains a wealth of information (user-specific and rendering information), including current user status (with geocode information). A tweet also contains a large amount of information, and you can easily visualize generating this information using the `user_timeline` class.

**Listing 2. Anatomy of a Twitter user (Ruby perspective)**

```ruby
irb(main):007:0> puts Twitter.user("MTimJones")
<#Hashie::Mash
  contributors_enabled=false
  created_at="Wed Oct 08 20:40:53 +0000 2008"
  default_profile=false default_profile_image=false
  description="Platform Architect and author (Linux, Embedded, Networking, AI)."
  favourites_count=1
  follow_request_sent=nil
  followers_count=148
```

Data mining with Ruby and Twitter
That's it for the quick tour. Now, let's explore some simple scripts that you can use to collect and visualize data using Ruby and the Twitter API. Along the way, you'll get to know some of the concepts of Twitter, such as authentication and rate limiting.
Mining Twitter data

The following sections present several scripts for collecting and presenting data available through the Twitter API. These scripts focus on simplicity, but you can extend and combine them to create new capabilities. Further, this section touches the surface of the Twitter gem API, where many more capabilities are available.

It's important to note that the Twitter API only allows clients to make a limited number of calls in a given hour, that is, Twitter rate-limits requests (currently no more than 150 per hour), which means that after some amount of use, you'll get an error message and be required to wait before submitting new requests.

User information

Recall from Listing 2 that a large amount of information is available about each Twitter user. This information is only accessible if the user isn't protected. Let's look at how you can extract a user's data and present it in a more convenient way.

Listing 3 presents a simple Ruby script to retrieve a user's information (based on his or her screen name), and then emit some of the more useful elements. You use the `to_s` Ruby method to convert the value to a string as needed. Note that you first ensure that the user isn't protected; otherwise, this data wouldn't be accessible.

Listing 3. Simple script to extract Twitter user data (user.rb)

```ruby
#!/usr/bin/env ruby
require "rubygems"
require "twitter"

screen_name = String.new ARGV[0]
a_user = Twitter.user(screen_name)

if a_user.protected != true
    puts "Username   : " + a_user.screen_name.to_s
    puts "Name       : " + a_user.name
    puts "Id         : " + a_user.id_str
    puts "Location   : " + a_user.location
    puts "User since : " + a_user.created_at.to_s
    puts "Bio        : " + a_user.description.to_s
    puts "Followers  : " + a_user.followers_count.to_s
    puts "Friends    : " + a_user.friends_count.to_s
    puts "Listed Cnt : " + a_user.listed_count.to_s
    puts "Tweet Cnt  : " + a_user.statuses_count.to_s
    puts "Geocoded   : " + a_user.geo_enabled.to_s
    puts "Language   : " + a_user.lang
    if (a_user.url != nil)
        puts "URL        : " + a_user.url.to_s
    end
    if (a_user.time_zone != nil)
        puts "Time Zone  : " + a_user.time_zone
    end
    puts "Verified   : " + a_user.verified.to_s

    tweet = Twitter.user_timeline(screen_name).first
    puts "Tweet time : " + tweet.created_at
```
puts "Tweet ID : " + tweet.id.to_s
puts "Tweet text : " + tweet.text
end

To invoke this script, ensuring that it's executable (chmod +x user.rb), you invoke it with a user. The result is shown in Listing 4 for the developerworks user, showing the user information and current status (last tweet information). Note here that Twitter defines followers as people who follow you; but people that you follow are called friends.

Listing 4. Sample output from user.rb

```
$ ./user.rb developerworks
Username : developerworks
Name : developerworks
Id : 16362921
Location :
User since : Fri Sep 19 13:10:39 +0000 2008
Bio : IBM's premier Web site for Java, Android, Linux, Open Source, PHP, Social, Cloud Computing, Google, jQuery, and Web developer educational resources
Followers : 48439
Friends : 46299
Listed Cnt : 3801
Tweet Cnt : 9831
Geocoded : false
Language : en
URL : http://bit.ly/EQ7te
Time Zone : Pacific Time (US & Canada)
Verified : false
Tweet time : Sun Jul 17 01:04:46 +0000 2011
Tweet ID : 92399309022167040
Tweet text : dW Twitter is saving #IBM over $600K per month: will #Google+ add to that? > http://t.co/HiRwir7 #Tech #webdesign #Socialmedia #webapp #app
```

Friends popularity

Look at your friends (people you follow), and gather data to understand their popularity. In this case, you gather your friends and sort them in the order of their followers count. This simple script is shown in Listing 5.

In this script, after you understand the user you want to analyze (based on their screen name), you create a user hash. A Ruby hash (or associative array) is a data structure that allows you to define the key for storage (instead of a simple numerical index). Your hash is then indexed by Twitter screen name, and the associated value is the user's follower count. The process is simply to iterate your friends and hash their followers count. Sort your hash (in descending order), and emit it as output.

Listing 5. Friend's popularity script (friends.rb)

```
#!/usr/bin/env ruby
require "rubygems"
require "twitter"
require 'google_chart'
name = String.new ARGV[0]
user = Hash.new
```
Data mining with Ruby and Twitter

Sample output from the friends script in Listing 5 is shown in Listing 6. I've clipped the output to conserve space, but as you can see, ReadWriteWeb (RWW) and Playstation are popular Twitter users in my direct network.

**Listing 6. Screen output from the friends script in Listing 5**

```
$ ./friends.rb MTimJones
RWW, 1096862
PlayStation, 1026634
HarvardBiz, 541139
tedtalks, 526886
lifehacker, 146162
wandfc, 121683
AnonymousIRC, 117896
iTunesPodcasts, 82581
adultswim, 76188
forrester, 72945
googleresearch, 66318
Gartner_inc, 57468
developerworks, 48518
```

**Where are my followers?**

Recall from Listing 2 that Twitter provides a wealth of location information. There's a location field that is free form, user defined, and optional geocoding data. However, a user-defined time zone can also provide a hint as to the follower's actual location.

In this example, you build a mash-up that extracts time zone data from your Twitter followers, and then visualize this data using Google Charts. Google Charts is an interesting project that allows you to build a variety of different chart types over the web; defining the chart type and data as an HTTP request, where the result is rendered directly in the browser as the response. To install the Ruby gem for Google Charts, use the following command line:

```
$ gem install gchartrb
```

Listing 7 provides the script for extracting time zone data, then building the Google Charts request. First, unlike previous scripts, this script requires that you be authenticated with Twitter. To do this, you need to register an application with Twitter, which provides you with a set of keys and tokens. Those tokens can be applied to the script in Listing 7 to successfully extract the data. See Resources for details on this easy process.
Following a similar pattern, this script accepts a screen name, and then iterates the followers of that user. The time zone is extracted for the current follower and stored in the `tweetlocation` hash. Note, you first test whether this key is in the hash and, if so, increment the counter for that key. You also keep a tab on the number of total time zones for the later construction of percentages.

The last portion of the script is the construction of the Google Pie Chart URL. You create a new `PieChart` and specify some options (size, title, and whether it's 3D). Then, you iterate your time zone hash, emitting data for the chart for the time zone string (removing the `&` symbol) and the percentage of the time zone from the total.

**Listing 7. Building a pie chart from Twitter followers' time zones (followers-location.rb)**

```ruby
#!/usr/bin/env ruby
require "rubygems"
require "twitter"
require 'google_chart'

screen_name = String.new ARGV[0]
tweetlocation = Hash.new
timezones = 0.0

# Authenticate
Twitter.configure do |config|
  config.consumer_key = '<consumer_key>'
  config.consumer_secret = '<consumer_secret>'
  config.oauth_token = '<oauth_token>'
  config.oauth_token_secret = '<oauth_token_secret>'
end

cursor = "-1"

# Loop through all pages
while cursor != 0 do
  # Iterate followers, hash their location
  followers = Twitter.follower_ids(screen_name, :cursor=>cursor)
  followers.ids.each do |fid|
    f = Twitter.user(fid)
    loc = f.time_zone.to_s
    if (loc.length > 0)
      if tweetlocation.has_key?(loc)
        tweetlocation[loc] = tweetlocation[loc] + 1
      else
        tweetlocation[loc] = 1
      end
      timezones = timezones + 1.0
    end
  end
  cursor = followers.next_cursor
end
```
To execute the script from Listing 7, provide it with a Twitter screen name, and then copy and paste the resulting URL into a browser. **Listing 8** shows this process with the resulting generated URL.

**Listing 8. Invoking the followers-location script (result is a single line)**

```bash
$ ./followers-location.rb MTimJones
&chs=650x350&cht=Time+Zones&chd=s:KDDyKcKDOcKDKDDDDOKDDDDOKDDDDO&cht=p
```

When you paste the URL from Listing 8 into a browser, you get the result shown in **Figure 1**.

**Figure 1. Pie chart of Twitter followers' locations**

![Pie chart of Twitter followers' locations](image)

**Twitter user behavior**

Twitter contains a large amount of data that you can mine to understand some elements of user behavior. Two simple examples are to analyze when a Twitter user tweets and from what application the user tweets. You can use the following two simple scripts to extract and visualize this information.

**Listing 9** presents a script that iterates the tweets from a particular user (using the `user_timeline` method), and then for each tweet, extracts the particular day on which the tweet originated. You use a simple hash again to accumulate your weekday counts, then generate a bar chart using...
Google Charts in a similar fashion to the previous time zone example. Note also the use of default for the hash, which specifies the value to return for undefined hashes.

**Listing 9. Building a bar chart of tweet days (tweet-days.rb)**

```ruby
#!/usr/bin/env ruby
require "rubygems"
require "twitter"
require "google_chart"

screen_name = String.new ARGV[0]
dayhash = Hash.new

# Initialize to avoid a nil error with GoogleCharts (undefined is zero)
dayhash.default = 0
timeline = Twitter.user_timeline(screen_name, :count => 200 )
timeline.each do |t|
  tweetday = t.created_at.to_s[0..2]
  if dayhash.has_key?(tweetday)
    dayhash[tweetday] = dayhash[tweetday] + 1
  else
    dayhash[tweetday] = 1
  end
end

GoogleChart::BarChart.new('300x200', screen_name, :vertical, false) do |bc|
  bc.data "Sunday", [dayhash["Sun"]], '00000f'
  bc.data "Monday", [dayhash["Mon"]], '0000ff'
  bc.data "Tuesday", [dayhash["Tue"]], '00ff00'
  bc.data "Wednesday", [dayhash["Wed"]], '00ffff'
  bc.data "Thursday", [dayhash["Thu"]], 'ff0000'
  bc.data "Friday", [dayhash["Fri"]], 'ff00ff'
  bc.data "Saturday", [dayhash["Sat"]], 'ffff00'
  puts bc.to_url
end
```

**Figure 2** provides the result of the execution of the tweet-days script in Listing 9 for the developerWorks account. As shown, Wednesday tends to be the most active tweet day, with Saturday and Sunday the least active.

**Figure 2. Relative bar chart of per-day tweet activity**

The next script determines from which source a particular user tweets. There are several ways you can tweet, and this script doesn't encode them all. As shown in **Listing 10**, you use a similar
pattern to extract the user timeline for a given user, and then attempt to decode the source of the tweet in a hash. You use the hash later to create a simple pie chart using Google Charts to visualize the data.

**Listing 10. Building a pie chart of a user’s tweet sources (tweet-source.rb)**

```ruby
#!/usr/bin/env ruby
require "rubygems"
require "twitter"
require 'google_chart'

screen_name = String.new ARGV[0]
tweetsource = Hash.new

timeline = Twitter.user_timeline(screen_name, :count => 200 )
timeline.each do |t|
  if (t.source.rindex('blackberry')) then
    src = 'Blackberry'
  elsif (t.source.rindex('snaptu')) then
    src = 'Snaptu'
  elsif (t.source.rindex('tweetmeme')) then
    src = 'Tweetmeme'
  elsif (t.source.rindex('android')) then
    src = 'Android'
  elsif (t.source.rindex('LinkedIn')) then
    src = 'LinkedIn'
  elsif (t.source.rindex('twitterfeed')) then
    src = 'Twitterfeed'
  elsif (t.source.rindex('twitter.com')) then
    src = 'Twitter.com'
  else
    src = t.source
  end

  if tweetsource.has_key?(src)
    tweetsource[src] = tweetsource[src] + 1
  else
    tweetsource[src] = 1
  end
end

GoogleChart::PieChart.new('320x200', "Tweet Source", false) do |pc|
  tweetsource.each do|source,count|
    pc.data source.to_s, count
  end
end

puts "\nPie Chart"
puts pc.to_url
```

**Figure 3** provides a visualization of a user on Twitter who has an interesting set of tweet sources. The traditional Twitter website is used most often, along with a mobile phone application next.
Followers graph
Twitter is a massive network of users that forms a graph. As you've seen from the scripts, it's easy to iterate your contacts, and then iterate their contacts. Doing so forms the basis for a large graph, even at this level.

To visualize a graph, I've chosen to use the graph visualization software GraphViz. On Ubuntu, you can easily install this tool using the following command line:

```
$ sudo apt-get install graphviz
```

The script shown in **Listing 11** iterates a user's followers, and then iterates their followers. The only real difference in this pattern is the construction of a GraphViz dot-formatted file. GraphViz uses a simple script format to define graphs, which you'll emit as part of your enumeration of the Twitter users. As shown, you define a graph simply by specifying the relationships of the nodes.

**Listing 11. Visualizing a Twitter followers graph (followers-graph.rb)**
```ruby
#!/usr/bin/env ruby
require "rubygems"
require "twitter"
require 'google_chart'
screen_name = String.new ARGV[0]
tweetlocation = Hash.new

# Authenticate
Twitter.configure do |config|
  config.consumer_key = '<consumer_key>'
  config.consumer_secret = '<consumer_secret>'
  config.oauth_token = '<oauth_token>'
  config.oauth_token_secret = '<oath_token_secret>'
end

my_file = File.new("graph.dot", "w")
my_file.puts "graph followers {
  node [ fontname=Arial, fontsize=6, penwidth=4 ];
"

# Get the first page of followers
followers = Twitter.follower_ids(screen_name, :cursor=> -1 )

# Iterate the followers returned in the Array (max 10).
followers.ids[0..[5,followers.ids.length].min].each do |fid|
```

```
f = Twitter.user(fid)

# Only iterate if we can see their followers
if (f.protected.to_s != "true")
    my_file.puts "  " + screen_name + " -- " + f.screen_name.to_s + ""

    # Get the first page of their followers
    followers2 = Twitter.follower_ids(f.screen_name, :cursor => -1 )

    # Iterate the followers returned in the Array (max 10).
    followers2.ids[0..[5,followers2.ids.length].min].each do |fid2|
        f2 = Twitter.user(fid2)
        my_file.puts "    " + f.screen_name.to_s + " -- " + f2.screen_name.to_s + ""
    end
end

my_file.puts "}"

Execute the script from Listing 11 on a user results in a dot file that you then generate an image from using GraphViz. First, invoke the Ruby script to gather the graph data (stored as graph.dot); then, use GraphViz to generate the graph image (here, using circo, which specifies a circular layout). The process of generating this image is defined as follows:

$ ./followers-graph.rb MTimJones
$ circo graph.dot -Tpng -o graph.png

The resulting image is shown in Figure 4. Note that the Twitter graphs tend to be large, so I've constrained the graph by minimizing the number of users and their followers to enumerate (per the :count option in Listing 11).
Figure 4. Sample Twitter follower graph (extreme subset)

**Location information**

When enabled, Twitter collects geolocation data about you and your tweets. This data consists of latitude and longitude information that can be used to pinpoint a user or from where a tweet originates. Further, searches can incorporate this information so that you can identify places or people based on a defined location or your location.

Not all users or tweets are geo-enabled (for privacy reasons), but this information serves as an interesting dimension to the overall Twitter experience. Let's look at a script that allows you to visualize with geolocation data as well as another that allows you to search with this information.

In the first script (shown in Listing 12), you grab latitude and longitude data from a user (recall the bounding box from Listing 2). Although the bounding box is a polygon defining the area represented for the user, I simplify and use one point of this data. With this data, I generate a simple JavaScript function in a simple HTML file. This JavaScript code interfaces with Google Maps to present an overhead map of this location (given the latitude and longitude data extracted from the Twitter user).
Listing 12. Ruby script to construct a map of a user (where-am-i.rb)

```ruby
#!/usr/bin/env ruby
require "rubygems"
require "twitter"
require 'google_chart'

Twitter.configure do |config|
  config.consumer_key = '<consumer_key>'
  config.consumer_secret = '<consumer_secret>'
  config.oauth_token = '<oauth_token>'
  config.oauth_token_secret = '<oauth_token_secret>'
end

screen_name = String.new ARGV[0]
a_user = Twitter.user(screen_name)

if a_user.geo_enabled == true
  long = a_user.status.place.bounding_box.coordinates[0][0][0];
  lat  = a_user.status.place.bounding_box.coordinates[0][0][1];

  my_file = File.new("test.html", "w")
  my_file.puts "<!DOCTYPE html">
  my_file.puts "<html><head>
  my_file.puts "<meta name="viewport" content="initial-scale=1.0, "
  my_file.puts "user-scalable=no" />
  my_file.puts "<style type="text/css">
  my_file.puts "html { height: 100% }"
  my_file.puts "body { height: 100%; margin: 0px; padding: 0px }
  my_file.puts "#map_canvas { height: 100% }
  my_file.puts "</style>"
  my_file.puts "<script type="text/javascript">
  my_file.puts "<script src="http://maps.google.com/maps/api/js?sensor=false">
  my_file.puts "<script type="text/javascript">"
  my_file.puts "function initialize() {
  my_file.puts "var latlng = new google.maps.LatLng(" + lat.to_s + ", " + long.to_s + ");"
  my_file.puts "var myOptions = {
  my_file.puts "zoom: 12,"
  my_file.puts "center: latlng,"
  my_file.puts "mapTypeId: google.maps.MapTypeId.HYBRID"
  my_file.puts "};"
  my_file.puts "var map = new google.maps.Map(document.getElementById("map_canvas"),";
  my_file.puts "myOptions);"
  my_file.puts "}
  my_file.puts "</script>
  my_file.puts "<head>"
  my_file.puts "</head>"

  my_file.puts "</div>
  my_file.puts "<body onload="initialize()">"
  my_file.puts "<div id="map_canvas" style="width:100%; height:100%"></div>"
  my_file.puts "<body>
  my_file.puts "</html>"
else
  puts "no geolocation data available."
end
```

The script in Listing 12 is executed simply as:

```
$ ./where-am-i.rb MTimJones
```
The resulting HTML file is rendered through a browser, such as:

```
$ firefox test.html
```

This script can fail if no location information is available; but if it succeeds, an HTML file is generated that a browser can read to render the map. Figure 5 presents the resulting map image, which shows a portion of the Front Range of northern Colorado, USA.

**Figure 5. Sample image rendered from the script in Listing 12**

![Map Image](image)

With the geolocation, you can also search Twitter to identify Twitter users and tweets related to a particular location. The Twitter Search API allows geocoding information to restrict its results. The following example shown in **Listing 13** extracts latitude and longitude data for a user, then uses this data to fetch tweets within a radius of 5 miles of that location.

**Listing 13. Search for local tweets with latitude and longitude data (tweets-local.rb)**

```ruby
#!/usr/bin/env ruby
require "rubygems"
require "twitter"

Twitter.configure do |config|
  config.consumer_key = '<consumer_key>'
  config.consumer_secret = '<consumer_secret>'
  config.oauth_token = '<oauth_token>'
  config.oauth_token_secret = '<oauth_token_secret>'
end

screen_name = String.new ARGV[0]
a_user = Twitter.user(screen_name)
```
if a_user.geo_enabled == true
    long = a_user.status.place.bounding_box.coordinates[0][0][0]
    lat  = a_user.status.place.bounding_box.coordinates[0][0][1]
    Array tweets = Twitter::Search.new.geocode(lat, long, "5mi").fetch
    tweets.each do |t|
        puts t.from_user + " | " + t.text
    end
end

The result of the script in Listing 13 is shown in Listing 14. This is a subset of the tweets given the frequency of tweeters out there.

Listing 14. Viewing local tweets within 5 miles of my location

$ ./tweets-local.rb MTimJones
Breese summer | @DaltonOls did he answer u
LongmontRadMon | 60 CPM, 0.4872 uSv/h, 0.6368 uSv/h, 2 time(s) over natural radiation
grae slon | on every street there is a memory; a time and place we can never be again.
Breese summer | #I'm in a fight with @DaltonOls to see who will marry @TheCodySimpson I will marry him! ! ! : /
_JennieJune_ | ok I'm done, goodnight everyone!
Breese summer | @DaltonOls same
_JennieJune_ | @sylquejr sleep well!
Breese summer | @DaltonOls ok let's see what he says
LongmontRadMon | 90 CPM, 0.7308 uSv/h, 0.7864 uSv/h, 2 time(s) over natural radiation
Breese summer | @TheCodySimpson would u marry me or @DaltonOls
$

Going further

This article presented a number of simple scripts for extracting data from Twitter using the Ruby language. The emphasis was on the development and presentation of simple scripts to illustrate the fundamental ideas, but much more is possible. For example, you can also use the API to explore your friends networks and identify the most popular Twitter users of interest to you. Another interesting area is the mining of tweets themselves, using geolocation data to understand location-based behaviors or events (such as flu outbreaks). This article only scratched the surface, but feel free to comment below with your own mash-ups. Ruby and the Twitter gem make it simple to develop useful mash-ups or dashboards for your data-mining needs.
Resources

Learn

- **Ruby's official language website** is the single source for Ruby news, information, releases, documentation, and community support for the Ruby language. Given Ruby's growing use in web frameworks (such as Ruby on Rails), you can also learn the most recent security vulnerabilities and their solutions.
- **The Github social coding site** provides the official source for the Twitter gem. At this site, you can get access to the source, documentation, and mailing list for the Ruby Twitter gem.
- **Registering a Twitter application** is necessary to use certain elements of the Twitter API. The process is free and allows you to access some of the more useful elements of the API.
- **Google Maps JavaScript API tutorial** shows how to use Google Maps to render maps of various types using user-provided geolocation data. The JavaScript used in this article was based on the "Hello World" example code provided within.
- **developerWorks Open source zone** provides a wealth of information on open source tools and using open source technologies.
- **developerWorks on Twitter**: Follow us and follow this author at M. Tim Jones.
- **developerWorks on-demand demos**: Watch and learn demos ranging from product installation and setup demos for beginners to advanced functionality for experienced developers.

Get products and technologies

- **The Twitter Ruby gem**, developed by John Nunemaker, provides a useful interface to the Twitter service that cleanly integrates into the Ruby language.
- **The Google Chart API** is a useful service that provides the ability to construct complex and rich graphics using a variety of styles and options. This service provides an API through which a URL results that is rendered at the Google site.
- **The Google Chart API Ruby wrapper** provides a Ruby interface to the Google Charts API for the construction of useful charts within Ruby.
- **Evaluate IBM products** in the way that suits you best: Download a product trial, try a product online, use a product in a cloud environment, or spend a few hours in the **SOA Sandbox** learning how to implement service-oriented architecture efficiently.

Discuss

- **developerWorks community**: Connect with other developerWorks users while exploring the developer-driven blogs, forums, groups, and wikis.
About the author

M. Tim Jones

M. Tim Jones is an embedded firmware architect and the author of *Artificial Intelligence: A Systems Approach, GNU/Linux Application Programming* (now in its second edition), *AI Application Programming* (in its second edition), and *BSD Sockets Programming from a Multilanguage Perspective*. His engineering background ranges from the development of kernels for geosynchronous spacecraft to embedded systems architecture and networking protocols development. Tim is a platform architect with Intel and author in Longmont, Colorado.

© Copyright IBM Corporation 2011
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)