geoTwitter package

Andy Galdi, Robert MacNguyen, Paavni Rattan

March 21, 2013

1 Introduction

User location are required for many applications for personalization and location based services. Social Network data is an important source of user location. However for many services and social networks like Twitter the user location is an input from the user in a text box. Users often input general or nonsensical locations. A very small percentage of users use geo-tagging due to privacy concerns. This package will focus on Twitter data as the input to location field is text entered by user which is highly noisy data. It is very difficult to do any meaningful analysis with this unstructured data. The *geoTwitter* package provides methods for scraping, analyzing and plotting Twitter or similar data. This is also a big data project as the number of users on social networks is growing exponentially and the location field is a representation of the online social geographic information.

2 Objective

The package implements methods to query twitter data and obtain location information . It also includes methods to process the location text and query the cleaned location using GeoNames, a geographical database. The package provides methods to analyze and plot the location data from twitter.

3 Walkthrough (Examples)

This section provides step by step examples of how we can use the package.

Scraping

Authentication via OAuth allows the user to scrape data more effectively.

```
# ------------------------------------------------------------------------
## AUTHENTICATION WITH TWITTER
# ========================================================================
library(ROAuth)
requestURL     <- 'https://api.twitter.com/oauth/request_token'
accessURL      <- 'http://api.twitter.com/oauth/access_token'
authURL        <- 'http://api.twitter.com/oauth/authorize'
consumerKey    <- '<< YOUR KEY >>'
consumerSecret <- '<< YOUR SECRET >>'
oauth <- OAuthFactory$new(consumerKey    = consumerKey,
                          consumerSecret = consumerSecret,
                          requestURL     = requestURL,
                          accessURL      = accessURL,
                          authURL        = authURL)
cainfo <- system.file("CurlSSL", "cacert.pem", package = "RCurl")
oauth$handshake(cainfo = cainfo)
save(oauth, file = 'oauth.RData')
```

The `getFollowerIDs` and `getUsersFromIDs` functions can then be used either with or without OAuth to access the Twitter API and obtain data. The `streamR` package can also be used to access the Twitter Stream API.

```
# ------------------------------------------------------------------------
## SCRAPING FROM TWITTER
# ========================================================================
# Scrape from REST API
biebs <- getFollowerIDs('justinbieber', timeout = 10,
                        file.name = 'biebsFollowerIDs.txt', oauth = oauth)
obama <- getFollowerIDs('barackobama', timeout = 10,
                        file.name = 'obamaFollowerIDs.txt', oauth = oauth)
beliebers <- getUsersFromIDs('biebsFollowerIDs.txt', timeout = 60,
                             file.name = 'beliebers.txt', oauth = oauth)
teamobama <- getUsersFromIDs('obamaFollowerIDs.txt', timeout = 60,
                             file.name = 'teamobama.txt', oauth = oauth)
# Scrape from Stream API using streamR
filterStream(file = "streamBiebs.json",
             track = c('justin bieber', 'belieber', 'bieber', 'biebs'),
             timeout = 60, oauth = oauth)
filterStream(file = "streamOBama.json",
             track = c('barack obama', 'barack', 'obama', 'potus'),
             timeout = 60, oauth = oauth)
```

Cleaning and Querying

Once the data is scraped it is first stored on disk using `readTweetsToFile` and then location names are cleaned using `cleanLocationFromFile` which repeatedly calls `cleanLocation` on chunks of the data

```
# read into .csv file
readTweetsToFile(file = "Biebs.csv", jsonFile = "streamBiebs.json")
readTweetsToFile(file = "Obama.csv", jsonFile = "streamObama.json")

# clean locations
cleanBiebs = cleanLoationFromFile(file = "Biebs.csv",
                                  colClasses = c(id_str = "factor", created_at = "POSIXct",
                                  location = "factor", lon = "double", lat = "double"),
                                  chunkSize = 250L)
cleanObama = cleanLoationFromFile(file = "Obama.csv",
                                  colClasses = c(id\_str = "factor", created\_at = "POSIXct",
                                  location = "factor", lon = "double", lat = "double"),
                                  chunkSize = 250L)

queriedBiebs = queried <- queryLocation(cleanBiebs[,1], cleanBiebs[,3], cleanBiebs[,4],
                          username = 'usr_name')
queriedObama = queried <- queryLocation(cleanObama[,1], cleanObama[,3], cleanObama[,4],
                          username = 'usr_name')
```

Below illustrates how the functions called within `cleanLocationFromFile` work.

```
# load stop words
loadStopWords()
# update stop words with a list of new words
updateStopwords(c("sky", "lol"))
# or a single word
updateStopwords("lol")

# match US Postal Code pattern
matchPostalCode("94305")
matchPostalCode("94305-1234")
matchPostalCode("943")
matchPostalCode("943051234")

# trim unwanted spaces from strings
trim("     I     live    in Stanford      ")

# remove stop words from the string
loadStopWords()
result <- removeStopWords("I am living a dream in LA")

# clean a vector of user locations
user <- c("1111a", "2222b", "3333c", "4444d")
location <- c("College Park, MD", "The SKY is the  LIMIT",
"21093", "iPhone: 34.479984,-93.004616")
result <- cleanLocation(user,location)
# or a single location
location <- c("Living in C.A.")
result <- cleanString(location)
```

```
location <- c("iPhone: 34.479984,-93.004616")
result <- cleanString(location)

# query a vector of user locations
user <- c("1111a", "2222b", "3333c", "4444d", "5555d")
location <- c("College Park, MD", "21093",
"34.479984,-93.004616", "The sky is the LIMIT", "")
type <- c("Name", "Postal", "Coordinates", "Name", "")
result <- queryLocation(user,location,type,username = "geotwitter")
# or a single location
location <- c("College Park, MD")
type <- c("Name")
result <- queryGeoNames(location,type,username = "geotwitter")
location <- c("")
type <- c("")
result <- queryGeoNames(location,type,username = "geotwitter")
```

Final Processing and Plotting

Various functions in the package process and plot the data.

Dynamic time plots

Interactive plots that show changes in tweet distribution over regions over time.

```
# find state numbers for each lat lon for plotting by state
queriedBiebs$lat = as.numeric(queriedBiebs$lat)
queriedBiebs$lng = as.numeric(queriedBiebs$lng)
queriedObama$lat = as.numeric(queriedObama$lat)
queriedObama$lng = as.numeric(queriedObama$lng)

stateIds = latLong2RegionNum(lon=queried$lng, lat=queried$lat, map='state')

# create data.frame for plotting
plotDFBiebs <- mergeTimesRegionId(file = "Biebs.csv", regionIds=stateIds,
                                  colClasses=c(id_str = "factor",
                                               created_at = "POSIXct",
                                               location = "factor",
                                               lon = "double",
                                               lat = "double"))

plotDFObama <- mergeTimesRegionId(file = "Obama.csv", regionIds=stateIds,
                                  colClasses=c(id_str = "factor",
                                               created_at = "POSIXct",
                                               location = "factor",
                                               lon = "double",
                                               lat = "double"))

createDynamicTimePlot(times=plotDFBiebs$times, regionIds=plotDF$regionIds,
                      map='state', units='mins', interval=2)

createDynamicTimePlot(times=plotDFObama$times, regionIds=plotDF$regionIds,
                      map='state', units='mins', interval=2)
```

Mapping Users

The following code plots users onto maps with various different settings.

```
# need to clean <NA> from the data and make sure coordinates are numeric
beliebers <- queryBeliebers[!is.na(queryBeliebers$lat), ]
teamobama <- queryTeamobama[!is.na(queryTeamobama$lat), ]
beliebers$lat <- as.numeric(beliebers$lat)
beliebers$lng <- as.numeric(beliebers$lng)
teamobama$lat <- as.numeric(teamobama$lat)
teamobama$lng <- as.numeric(teamobama$lng)

# can plot two data sets on the same map by manipulating ggplot layers
# future implementations could enable this functionality in one function
region <- map_data('state')
b <- plotUsersInRegion(lng, lat, beliebers, long, lat, region, group,
                       colors = 'red')
o <- plotUsersInRegion(lng, lat, teamobama, long, lat, region, group,
                       colors = 'blue')
p <- b + o$layers[2]

# plot only users in a given state with a density cloud and contours
```

```
# the functionality provided by plotUsersInCountry() is the same
state <- 'texas'
b <- plotUsersInState(lng, lat, beliebers, state,
                      colors = c('red', 'blue'),
                      types = c('cloud', 'contour'))
b + coord_cartesian(xlim = c(-106, -94), ylim = c(26, 36))

country <- 'Mexico'
b <- plotUsersInCountry(lng, lat, beliebers, country,
                     colors = c('red', 'blue'),
                     types = c('cloud', 'points'))
b + coord_cartesian(xlim = c(-106, -94), ylim = c(26, 36))
```

Plotting Over Regions

The following code uses aggregated data (counts and densities of users in a region) to color regions.

```
# color regions based on count relative to other regions
region <- map_data('state')
p <- fillRegionsByUsers(lng, lat, teamobama, long, lat, region, group, FALSE)
p

# color regions based on density (users/sq mile) relative to other regions
region <- map_data('state')
p <- fillRegionsByUsers(lng, lat, teamobama, long, lat, region, group, TRUE)
p

# compare the counts from each dataset in a given region and return the winner
region <- map_data('state')
xlist <- c('lng', 'lng')
ylist <- c('lat', 'lat')
datalist <- list(TheBiebs = beliebers, BarryO = teamobama)
p <- compareOverRegions(xlist, ylist, datalist, long, lat, region, group)
# add points from each dataset (note that there are stacked points)
barryO   <- plotUsersInRegion(lng, lat, teamobama, long, lat, region,
                              group, colors = 'red')
theBiebs <- plotUsersInRegion(lng, lat, beliebers, long, lat, region,
                              group, colors = 'dark green')
p + barryO$layers[2] + theBiebs$layers[2]
```