Extracting data from Online Social Networks

Cristina Pérez Solà and Jordi Herrera Joancomartí

Departament d’Enginyeria de la Informació i les Comunicacions
Universitat Autònoma de Barcelona

July 1st, 2014
Today's goal
1. Introduction
2. Online SocNs
3. Computer programming basics
4. Programming in R
5. Interacting with the provider
6. Data extraction project
1 Introduction
   - Presentation
   - What are we going to do today?

2 Online SocNs

3 Computer programming basics

4 Programming in R

5 Interacting with the provider

6 Data extraction project
Who are we?

We are from the
Department of Information and Communications Engineering
at the
Autonomous University of Barcelona.

cristina.perez@deic.uab.cat
http://deic.uab.cat/~cperez/eusnworkshop/
http://tinyurl.com/EUSNdataextraction
Among other topics, we do research about Privacy in Online Social Networks (OSN):

- **OSN Crawling**.
- Community detection algorithms.
- Classify OSN users from the social graph.
- Infer private data from public data.
Among other topics, we do research about Privacy in Online Social Networks (OSN):

- OSN Crawling.
- **Community detection algorithms.**
- Classify OSN users from the social graph.
- Infer private data from public data.
What do we do?

Among other topics, we do research about Privacy in Online Social Networks (OSN):

- OSN Crawling.
- Community detection algorithms.
- Classify OSN users from the social graph.
- Infer private data from public data.
Among other topics, we do research about Privacy in Online Social Networks (OSN):

- OSN Crawling.
- Community detection algorithms.
- Classify OSN users from the social graph.
- **Infer private data from public data.**
What are we going to do today?

## Schedule

<table>
<thead>
<tr>
<th>Topic</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intro</td>
<td>15’</td>
</tr>
<tr>
<td>About Online Social Networks (theory)</td>
<td>15’</td>
</tr>
<tr>
<td>Computer programming (activity)</td>
<td>90’</td>
</tr>
<tr>
<td>R (theory + 1 activity)</td>
<td>30’</td>
</tr>
<tr>
<td>Break</td>
<td>45’</td>
</tr>
<tr>
<td>Interaction with the OSN provider (theory + 1 activity)</td>
<td>45’</td>
</tr>
<tr>
<td>Data extraction project (activity)</td>
<td>120’</td>
</tr>
</tbody>
</table>
What are we going to do today?
1 Introduction

2 Online SocNs
   - Introduction
   - Twitter

3 Computer programming basics

4 Programming in R

5 Interacting with the provider

6 Data extraction project
Introduction
We define **social network sites** as web-based services that allow individuals to
(1) construct a public or semi-public profile within a bounded system,
(2) articulate a list of other users with whom they share a connection, and
(3) view and traverse their list of connections and those made by others within the system.
OSN Modeling

We usually model OSN with graphs.
Twitter is an **online social network** and a **microblogging** service created on 2006.

Users exchange small text messages called **tweets**.

Tweets are limited to **140** characters.
Twitter

Web interface
Information about users on Twitter:

- Unique identifier
- Username (to login)
- Screen name (showed to others, modifiable)
- Verified?
- Location
- Geolocation active?
- Language
- ...
There are 3 different kinds of Tweets:

- Status updates.
- Retweets.
- Replies.

Moreover, tweets can contain geolocation information:

- Location place identifiers.
- Geographical coordinates.
Users can **follow** other users.

**Hashtags** are used to label or classify tweets (#EUSNConference).

**Trending topics** are the most used words (or sentences) in Twitter in a given moment.
Source of data for studies

Twitter data has been used in many studies and for different purposes:

- Twitter Alerts.
- Real-time event detection by social sensors.
- Twitter mood predicts the stock market.
- Predicting Elections with Twitter.
1. Introduction

2. Online SocNs

3. Computer programming basics
   - Learning to code
   - An hour of code

4. Programming in R

5. Interacting with the provider

6. Data extraction project
But what exactly is coding? Coding is what makes it possible for us to create computer software. Coding is **writing** software! Coding can be done in many **languages** (R, Python, C, Java, Javascript,...).

- Vocabulary.
- Syntax.

**Writing a computer program**

- Think / design an algorithm.
- Code the solution (implementation / coding).
- Test / debug.
Motivation

Why is it interesting to learn to code?

- We can automatize processes!
- Computers are everywhere!
- It helps us understand how systems around us work.
- We learn how to think about problems.
- We learn how to break down problems into small pieces.
- We learn to develop systematic solutions.
An **Hour of Code** is an initiative to help introduce more than 10 million students of all ages to computer programming.

We are going to follow an introductory course to programming:

http://learn.code.org/hoc/reset
Video: introduction

Video 1: Introduction

http://learn.code.org/hoc/reset
Activities

Let’s try to solve puzzles 1-5!
An hour of code

Puzzle 5

turnRight();
moveForward();
turnLeft();
moveForward();
moveForward();
turnLeft();
moveForward();
turnLeft();
moveForward();
turnLeft();
moveForward();
moveForward();
turnLeft();
moveForward();
Video: for loops

Video 2: Mark Zuckerberg teaches repeat loops

http://learn.code.org/hoc/6
Activities

Let’s try to solve puzzles 6-9!
Puzzle 9

for (var count = 0; count < 3; count++) {
    moveForward();
    moveForward();
    turnRight();
}
Video 3: Chris Bosh teaches repeat until statements

http://learn.code.org/hoc/10
Activities

Let’s try to solve puzzles 10-13!
Puzzle 13

```plaintext
while (notFinished()) {
    turnRight();
    moveForward();
    turnLeft();
    moveForward();
}
```
Video: if statements

Video 4: Bill gates explains if statements

http://learn.code.org/hoc/14
Activities

Let’s try to solve puzzles 14-17!
Puzzle 17

```c
while (notFinished()) {
    moveForward();
    if (isPathRight()) {
        turnRight();
    }
}
```
An hour of code

Video: if-else statements

Video 5: Saloni on the if/else block
http://learn.code.org/hoc/18
Activities

Let’s try to solve puzzles 18-20!
Puzzle 20

```c
while (notFinished()) {
    if (isPathForward()) {
        moveForward();
    } else {
        if (isPathRight()) {
            turnRight();
        } else {
            turnLeft();
        }
    }
}
```
Video: wrap up

Video 6: Wrap up

https://www.youtube.com/watch?v=98Wft30gUQE
We use **instructions** to write computer programs:

- Basic instructions (moveForward, turnRight, turnLeft).
- Flow control instructions:
  - Repeat (n) times
  - Repeat until (something happens)
  - If
  - If - else
Wrap up (II)

Lessons learned:
- One small mistake is enough for not getting to the end!
- We always win ;) !
- As with human languages, there are also many programming languages.
- It is better to think about the solution before start coding.

Want to learn more: 20 hour course at:

http://learn.code.org/
1. Introduction

2. Online SocNs

3. Computer programming basics

4. Programming in R
   - Introduction
   - Basic instructions
   - Flow control instructions

5. Interacting with the provider

6. Data extraction project
Now, we do not want to move the (angry) bird around any more, but to extract data from OSNs!

We are going to use the programming language R:

- Data mining and graphics.
- Interpreted language.
- Uses packages / libraries to extend its functionalities.
Basic instructions

R: basic instructions (I)

Variables:

```r
x <- 5;
z <- "Hello World";
```

Arithmetic:

```r
v <- x + 5;
v <- x * 5;
```

Show result:

```r
x <- 5;
v <- x + 5;
print(v);
```

```
[1] 10
```
R: basic instructions (II)

Vectors:

```r
x <- 1:10
print(x);
```

```
[1] 1 2 3 4 5 6 7 8 9 10
```

Load a library:

```r
library(RCurl)
```
```r
x <- 0:9
y <- 0:9 * 5
lm(y ~ x)
```

Coefficients:
(Intercept) x
-4.494e-15  5.000e+00

```
help(lm)
```

```r
lm package:stats R Documentation

Fitting Linear Models

Description:

'lm' is used to fit linear models. It can be used to carry out regression, single stratum analysis of variance and analysis of covariance (although 'aov' may provide a more convenient interface for these).

[...]
```
Functions

**Functions** are self contained modules of code that accomplish a specific task.

Functions:
- **take in** data
- process it / do something
- **return** a result
R: functions (II)

We have already seen some functions in action:

- **moveForward();**
  (no input, one step forward, no output).

- **print("hello");**
  (hello as input, prints content, does not return anything).

- **lm(y ~ x);**
  (x, y as inputs, computes linear regression, returns the coefficients of the regression as output).
Basic instructions

R: data frames

```r
name <- "Ford"
surname <- "Perfect"
df = data.frame(name, surname)
df

  name surname
1   Ford  Perfect

df$name
df$surname

[1] Ford
[1] Perfect
```
for(i in 1:5)
{
print(paste("i =", i));
}

[1] "i = 1"
[1] "i = 2"
[1] "i = 3"
[1] "i = 4"
[1] "i = 5"

x<-1:5
for (i in 1:length(x))
{
print(x[i])
}

[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
R: flow control instructions (loops II)

```r
x <- 1:10
i <- 1
while (i <= 5) {
  print(x[i])
  i <- i + 1
}

[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

```r
i <- 0
repeat {
  i <- i + 1;
  if(i > 5) break;
  print("Hi");
}

[1] "Hi"
[1] "Hi"
[1] "Hi"
[1] "Hi"
[1] "Hi"
```
R: flow control instructions (conditionals)

day <- 1;
if (day == 1) {
  x <- "WORKSHOP";
} else {
  x <- "CONFERENCE";
}
print(x)

[1] "WORKSHOP"

day <- 2;
if (day == 1) {
  x <- "1st day WORKSHOP";
} else if (day == 2) {
  x <- "2nd day WORKSHOP";
} else {
  x <- "CONFERENCE";
}
print(x)

[1] "2nd day WORKSHOP"
R: activity

Write a program to print all odd numbers from 0 to 100 in R.

Write a program to print the sequence of Fibonacci numbers (1,1,2,3,5,8,13,...)

http://www.r-fiddle.org/#/
Flow control instructions

R: activity (solutions to 1st problem)

```r
num <- 0:100;
flag <- 0
for (i in 1:length(num))
{
  if (flag == 0){
    flag <- 1;
  } else {
    print(num[i]);
    flag <- 0;
  }
}
```

```r
i<-1;
while (i < 100) {
  print(i);
  i <- i + 2;
} 
```
1 Introduction

2 Online SocNs

3 Computer programming basics

4 Programming in R

5 Interacting with the provider
   - APIs
   - The Twitter APIs
     - The REST API
     - The Streaming API
   - Authentication

6 Data extraction project
An **Application Programming Interface (API)** specifies how some software components should interact with each other.

Sometimes an API comes as a specification of remote calls exposed to the API consumers.
Popular APIs

APIs are everywhere:

- **Twitter**
- Flickr
- Facebook
- Google Maps
- YouTube

We can use them to obtain data, but also to send data.
Twitter has two main APIs:

- **REST**: ask for past information (6-9 days).
- **Streaming**: filter real-time information.
**Twitter REST API : search tweets (I)**

**GET search/tweets**: Returns a collection of relevant Tweets matching a specified **query**.

<table>
<thead>
<tr>
<th>Query</th>
<th>Returns tweets...</th>
</tr>
</thead>
<tbody>
<tr>
<td>EUSN</td>
<td>containing the word ”EUSN”.</td>
</tr>
<tr>
<td>EUSN Barcelona</td>
<td>containing both ”EUSN” and ”Barcelona”.</td>
</tr>
<tr>
<td>&quot;online social networks&quot;</td>
<td>containing the exact sentence.</td>
</tr>
<tr>
<td>european OR conference</td>
<td>containing either ”european” or ”conference” (or both).</td>
</tr>
<tr>
<td>”social network” -online</td>
<td>containing ”social network” but not ”online”.</td>
</tr>
<tr>
<td>#EUSNConference</td>
<td>containing the hashtag ”EUSNConference”.</td>
</tr>
<tr>
<td>from:isidromj</td>
<td>sent from user ”isidromj”.</td>
</tr>
</tbody>
</table>
### Twitter REST API: search tweets (II)

<table>
<thead>
<tr>
<th>Query</th>
<th>Returns tweets...</th>
</tr>
</thead>
<tbody>
<tr>
<td>to:isidromj</td>
<td>sent to person &quot;techcrunch&quot;.</td>
</tr>
<tr>
<td>@isidromj</td>
<td>referencing person &quot;mashable&quot;.</td>
</tr>
<tr>
<td>EUSN since:2010-12-27</td>
<td>containing &quot;EUSN&quot; and sent since date &quot;2010-12-27&quot;.</td>
</tr>
<tr>
<td>EUSN until:2014-07-01</td>
<td>containing &quot;EUSN&quot; and sent before the date &quot;2014-07-01&quot;.</td>
</tr>
<tr>
<td>barcelona :)</td>
<td>containing &quot;barcelona&quot; and with a positive attitude.</td>
</tr>
<tr>
<td>barcelona :(</td>
<td>containing &quot;barcelona&quot; and with a negative attitude.</td>
</tr>
<tr>
<td>EUSN ?</td>
<td>containing &quot;EUSN&quot; and asking a question.</td>
</tr>
<tr>
<td>EUSN filter:links</td>
<td>containing &quot;EUSN&quot; and linking to URL.</td>
</tr>
<tr>
<td>EUSN source:twitterfeed</td>
<td>containing &quot;EUSN&quot; and entered via TwitterFeed.</td>
</tr>
</tbody>
</table>
The Twitter APIs

Twitter REST API: search tweets (III)

Try it yourself:

https://twitter.com/search-home

Can you write queries to obtain...?

- Tweets containing the word Barcelona but not the word football.
- Tweets containing the word Barcelona, Paris or London.
- Tweets containing the words Barcelona, Paris and London.
- Tweets containing the exact sentence "tomorrow i will".
- Tweets sent yesterday with the word holidays.
- Tweets containing questions about paper submissions.
The Twitter APIs

Twitter REST API : search tweets (IV)

Note that you have used keywords like **since**, **until**, **from** or **to** to build your queries.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>q</strong></td>
<td>A UTF-8, URL-encoded search query of 500 characters maximum, including operators. Queries may additionally be limited by complexity.</td>
</tr>
<tr>
<td><strong>geocode</strong></td>
<td>Returns tweets by users located within a given radius of the given latitude/longitude. The location is preferentially taken from the Geotagging API, but will fall back to their Twitter profile. The parameter value is specified by &quot;latitude,longitude,radius&quot;, where radius units must be specified as either &quot;mi&quot; (miles) or &quot;km&quot; (kilometers). Note that you cannot use the near operator via the API to geocode arbitrary locations; however you can use this geocode parameter to search near geocodes directly. A maximum of 1,000 distinct &quot;sub-regions&quot; will be considered when using the radius modifier.</td>
</tr>
<tr>
<td><strong>lang</strong></td>
<td>Restricts tweets to the given language, given by an ISO 639-1 code. Language detection is best-effort.</td>
</tr>
<tr>
<td><strong>locale</strong></td>
<td>Specify the language of the query you are sending (only ja is currently effective). This is intended for language-specific consumers and the default should work in the majority of cases.</td>
</tr>
</tbody>
</table>
Twitter REST API: search tweets (V)

**result_type**

Optional. Specifies what type of search results you would prefer to receive. The current default is "mixed." Valid values include:

* mixed: Include both popular and real-time results in the response.
* recent: Return only the most recent results in the response.
* popular: Return only the most popular results in the response.

**Example Values:** mixed, recent, popular

**count**

Optional. The number of tweets to return per page, up to a maximum of 100. Defaults to 15. This was formerly the "rpp" parameter in the old Search API.

**Example Values:** 100

**until**

Optional. Returns tweets generated before the given date. Date should be formatted as YYYY-MM-DD. Keep in mind that the search index may not go back as far as the date you specify here.

**Example Values:** 2012-09-01

**since_id**

Optional. Returns results with an ID greater than (that is, more recent than) the specified ID. There are limits to the number of Tweets which can be accessed through the API. If the limit of Tweets has occurred since the since_id, the since_id will be forced to the oldest ID available.

**Example Values:** 12345
GET statuses/show/:id : Returns a single Tweet, specified by the id parameter. The Tweet’s author will also be embedded within the tweet.

GET statuses/retweets/:id : Returns a collection of the 100 most recent retweets of the tweet specified by the id parameter.
Twitter REST API: calls about users

**GET users/search**: Provides a simple, relevance-based search interface to public user accounts on Twitter. Try querying by topical interest, full name, company name, location, or other criteria.

**GET account/settings**: Returns settings (including current trend, geo and sleep time information) for the authenticating user.

**GET users/show**: Returns a variety of information about the user specified by the required user_id or screen_name parameter. The author’s most recent Tweet will be returned inline when possible.
Twitter REST API: calls about user relationships

**GET friends/ids**: Returns a cursored collection of user IDs for every user the specified user is following (otherwise known as their "friends").

**GET followers/ids**: Returns a cursored collection of user IDs for every user following the specified user.

**GET friendships/incoming**: Returns a collection of numeric IDs for every user who has a pending request to follow the authenticating user.

**GET friendships/outgoing**: Returns a collection of numeric IDs for every protected user for whom the authenticating user has a pending follow request.
Have a look at the full specification:

https://dev.twitter.com/docs/api/1.1
The Twitter APIs

Twitter streaming API

We are going to use just one of the Twitter streaming APIs (public stream) and one endpoint (statuses/filter). There are many other alternatives!

The statuses/filter endpoint returns public statuses that match one or more filter predicates though a single connection.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>follow</strong></td>
<td>A comma separated list of user IDs, indicating the users to return statuses for in the stream. See the follow parameter documentation for more information.</td>
</tr>
<tr>
<td><strong>track</strong></td>
<td>Keywords to track. Phrases of keywords are specified by a comma-separated list. See the track parameter documentation for more information.</td>
</tr>
<tr>
<td><strong>locations</strong></td>
<td>Specifies a set of bounding boxes to track. See the locations parameter documentation for more information.</td>
</tr>
<tr>
<td><strong>delimited</strong></td>
<td>Specifies whether messages should be length-delimited. See the delimited parameter documentation for more information.</td>
</tr>
<tr>
<td><strong>stall_warnings</strong></td>
<td>Specifies whether stall warnings should be delivered. See the stall_warnings parameter documentation for more information.</td>
</tr>
</tbody>
</table>
Streaming API: Follow

**Follow** allows us to specify data collection depending on user information:

- Tweets created by the user.
- Tweets which are retweeted by the user.
- Replies to any Tweet created by the user.
- Retweets of any Tweet created by the user.
- Manual replies, created without pressing a reply button.

The stream will not contain: Tweets mentioning the user, Manual Retweets created without pressing a Retweet button and Tweets by protected users.
Streaming API: Track

**Track** allows us to specify data collection depending on content:

<table>
<thead>
<tr>
<th>Parameter value</th>
<th>Will match...</th>
<th>Will not match...</th>
</tr>
</thead>
</table>
| Twitter                  | TWITTER
  twitter
  "Twitter"
  twitter.
  #twitter
  @twitter
  http://twitter.com     | TwitterTracker
                        #newtwitter                          |
| Twitter's                | I like Twitter's new design                        | Someday I'd like to visit @Twitter's office |
| twitter api, twitter streaming | The Twitter API is awesome
  The twitter streaming service is fast
  Twitter has a streaming API | I'm new to Twitter                       |
**Location** allows us to specify data collection depending on geographical position. Location uses the coordinates of the bounding box to specify positions.
About authentication

**Authentication** is the process of determining whether someone or something is, in fact, who or what it is declared to be.

We use usernames and passwords to authenticate ourselves everywhere:

- Email account.
- Online banking services.
- University virtual campus (moodle?).
- Facebook account.
OSN authentication

We also need to authenticate ourselves to use APIs:
- Identify the application that is accessing the API.
- Do things **on behalf of other users**.

What happens when Candy Crush wants to send a message to my Facebook friends?

We do not want to give our username and password to others!

Solution: tickets (or tokens)!
OAuth

OAuth is an open standard for authorization that provides client applications a ‘secure delegated access’ to server resources on behalf of a resource owner.

In other terms, OAuth allows users to give specific rights to applications to act on their behalf:

- Let Candy Crush send messages to my Facebook friends.
- Let TwitterDeck send tweets from my account.
- Let Facebook fetch my Gmail account contacts.
1 Introduction

2 Online SocNs

3 Computer programming basics

4 Programming in R

5 Interacting with the provider

6 Data extraction project
   - Preparing the environment
   - Twitter Authentication
   - Let’s do it!
     - REST API
     - STREAMING API
Preparation of the environment

**Download and Install R (I)**

Go to [http://cran.r-project.org/](http://cran.r-project.org/)

---

**Precompiled binary distributions of the base system and contributed packages,** *Windows and Mac* users most likely want one of these versions of R:

- Download R for Linux
- Download R for (Mac) OS X
- Download R for Windows

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.
Preparing the environment

Download and Install R (II)

Windows users:
- Download R for Windows
- base
- Download R 3.1.0 for Windows

Ubuntu / debian users:
- sudo add-apt-repository ppa:marutter/rrutter
- sudo apt-get update
- sudo apt-get install r-base r-base-dev

Mac users:
- Download R for (Mac) OS X
- R-3.1.0-snowleopard.pkg
Preparing the environment

R console (Windows)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
Preparing the environment

R console (Linux)

```
cris@enigma:~$ R

R version 3.1.0 (2014-04-10) -- "Spring Dance"
Copyright (C) 2014 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
```

>`
Hello world!

```r
Copyright (C) 2014 The R Foundation for Statistical Computing
Platform: i386-w64-mingw32/i386 (32-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> print("Hello World!!")
[1] "Hello World!!"
>
```
Preparing the environment

Download source code

Go to:

http://deic.uab.cat/~cperez/eusnworkshop/
http://tinyurl.com/EUSNdataextraction

Download and extract source-code.zip.
Preparing the environment

Download source code

Content of source-code.zip:

- credentials.R
- hello-world.R
- install-libraries.R
- load-credentials.R
- map.png
- StreamR-example1.R
- StreamR-example2.R
- test-credentials.R
- TwitterR-example1.R
- TwitterR-example2.R
Prepare the environment

Configure working directory

 RGhi (32-bit)

File Edit View Misc Packages Windows Help

Source R code...
New script
Open script...
Display file(s)...

Load Workspace...
Save Workspace...
Load History...
Save History...
Change dir...
Print...
Save to File...
Exit

>
Preparing the environment

Hello world (again)

(a)

R is free software and comes with ABSOLUTELY NO WARRANTY. You are welcome to redistribute it under certain conditions. Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors. Type 'contributors()' for more information and 'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or 'help.start()' for an HTML browser interface to help. Type 'q()' to quit R.

> source("hello-world.R")
[1] "Hello World!!"
>

(b)
# install required packages
install.packages("ROAuth");
install.packages("wordcloud");
install.packages("tm");
install.packages("twitteR");
install.packages("streamR");
install.packages("ggplot2");
install.packages("maps");
install.packages("ggmap");

source("install-libraries.R")

The downloaded binary packages are in
    C: Documents and Settings packages
trying URL 'http://cran.es.r-project.org/bin/windows/contrib/3.0/maps_2.3-7.zip'
Content type 'application/zip' length 2072327 bytes (2.0 Mb)
opened URL
downloaded 2.0 Mb

package 'maps' successfully unpacked and MD5 sums checked
Creating a new App (I)

1. Create a Twitter account:
   - Go to http://twitter.com
   - Click Sign Up for Twitter
   - Fill the form

2. Register a Twitter App:
   - Go to https://dev.twitter.com/apps
   - Create New App
   - Fill the form
Creating a new App (II)

Create an application

Application details

**Name**

An app for the Workshop

Your application name. This is used to attribute the source of a tweet and in user-facing authorization screens. 32 characters max.

**Description**

An app for the Workshop

Your application description, which will be shown in user-facing authorization screens. Between 10 and 200 characters max.

**Website**

http://localhost.com

Your application’s publicly accessible home page, where users can go to download, make use of, or find out more information about your application. This fully-qualified URL is used in the source attribution for tweets created by your application and will be shown in user-facing authorization screens. (If you don’t have a URL yet, just put a placeholder here but remember to change it later.)

**Callback URL**

Where should we return after successfully authenticating? OAuth 1.0a applications should explicitly specify their oauth_callback URL on the request token step, regardless of the value given here. To restrict your application from using callbacks, leave this field blank.
### An app for the Workshop

**Application settings**

*Keep the "API secret" a secret. This key should never be human-readable in your application.*

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>API key</td>
<td>s5AtMN5ThGlwTkRMyczREPnx</td>
</tr>
<tr>
<td>API secret</td>
<td>UD0Pnlc7C6jymG0HvNX29LdrySd47iSZ4Puf7P0jeZtu98DJgn</td>
</tr>
<tr>
<td>Access level</td>
<td>Read-only <em>(modify app permissions)</em></td>
</tr>
<tr>
<td>Owner</td>
<td>cisckjet</td>
</tr>
<tr>
<td>Owner ID</td>
<td>169012380</td>
</tr>
</tbody>
</table>
library(RCurl)
options(RCurlOptions = list(cainfo = system.file("CurlSSL", "cacert.pem", package = "RCurl")))
require(twitteR)

apiKey <- ""
apiSecret <- ""

reqURL <- "https://api.twitter.com/oauth/request_token"
accessURL <- "https://api.twitter.com/oauth/access_token"
authURL <- "https://api.twitter.com/oauth/authorize"
cred <- OAuthFactory$new(consumerKey=apiKey, consumerSecret=apiSecret, requestURL=reqURL, accessURL=accessURL, authURL=authURL)
cred$handshake(cainfo = system.file("CurlSSL", "cacert.pem", package = "RCurl"))
registerTwitterOAuth(cred)

# save credentials to file
save(cred, file="credentials.RData");

source("credentials.R")

Loading required package: bitops
Loading required package: twitteR
Loading required package: ROAuth
Loading required package: digest
Loading required package: rjson
To enable the connection, please direct your web browser to:
https://api.twitter.com/oauth/authorize?oauth_token=TJICeiWs7wLVc4ujQy6UTj25F4U1sq0JoYiZm0fUU
When complete, record the PIN given to you and provide it here: 7983356
>
Autoritzes un app per al Workshop a fer servir el teu compte?

Aquesta aplicació **podrà:**
- Llegeix els tuits de la teva cronologia
- Mira a qui segueixes

Autoritza l'aplicació

Aquesta aplicació **no podrà:**
- Segueix gent nova.
- Actualitza el teu perfil.
- Publica els tuits que t'han enviat.
- Accedeix als teus missatges directes.
- Consulta la teva contrasenya de Twitter.

An app for the Workshop
localhost.com
An app for the Workshop
# load libraries
library(twitteR);

# load credentials
load("credentials.RData");
# OAuth register
registerTwitterOAuth(cred);

# get three tweets about hashtag #Obama and print some information
tweets <- searchTwitter("#Obama", n=2, lang="en");
for(tweet in tweets) {
  # get one tweet and print some information
  show(paste("User ", tweet$getScreenName(), ", says: ", tweet$getText(), ", sep=""));
}

File: test-credentials.R

source("test-credentials.R")

[1] "User 'claudianpliego' says: 'RT @Matt_VanDyke: #Bush’s recklessness, #Obama’s
fecklessness leave U.S. looking weak as #Iraq crumbles
[1] "User 'roziedb' says: 'RT @Politicule: #ISIS militants conduct public beheadings
in #Iraq. #Obama asks if he can play through while golfing in PalmSprings’
library(twitteR);
source('load-credentials.R');
# search. Examples: #hashtag, @user, etc
tweets <- searchTwitter("#Obama", n=10, lang="es");
# get only one tweet and analyse it
tweet <- tweets[[1]];
# show the low level structure of 'status'
show("STRUCTURE OF 'STATUS' OBJECT:");
str(tweet);
# get some basic information
show(paste("TWEET ID: ",tweet$getId()));
show(paste("TEXT: ",tweet$text()));
show(paste("USER NAME: ",tweet$screenName()));
show(paste("IS RETWEET?: ",tweet$isRetweet()));
show(paste("RETWEETED: ",tweet$retweeted()));
# get information about the user
user <- getUser(tweet$screenName());
# print structure of 'user'
show("STRUCTURE OF 'USER' OBJECT:");
str(user);
# get some information about the user
show(paste("USER ID: ",user$id()));
show(paste("USER NAME: ",user$name()));
show(paste("SCREEN NAME: ",user$screenName()));
show(paste("LOCATION: ",user$location()));
show(paste("TWEETS NUMBER: ",user$statusesCount()));
show(paste("FOLLOWERS: ",user$followersCount()));
show(paste("DESCRIPTION: ",user$description()));
Example 1: REST API (output)

source("TwitterR-example-1.R")

[...] text, favorited, favoriteCount, replyToSN, created, truncated, getCreated, getFavoriteCount, getFavorited, getId, getIsRetweet, getLatitude, getLongitude, getReplyToSID, getReplyToSN, getReplyToUID, getRetweetCount, getRetweeted, getRetweets, getScreenName, getStatusSource, getText, getTruncated, getUrls, inidfialize, setCreated, setFavoriteCount, setFavorited, setId, setIsRetweet, setLatitude, setLongitude, setReplyToSID, setReplyToSN, setReplyToUID, setRetweetCount, setRetweeted, setScreenName, setStatusSource, setText, setTruncated, setUrls, toDataFrame, toDataFrame#twitterObj

[...] [1] "TWEET ID: 483133728210694144"
[1] "TEXT: @jacklyn_ballard @_woahitsemily Nuestra lucha es por nuestra PATRIA SOBERANA, por eso le hemos dicho #GoHomeGringosAsesinos #Obama"
[1] "USER NAME: rinconero43"
[1] "IS RETWEET?: FALSE"
[1] "RETWEETED: FALSE"
[1] "USER ID: 139597099"
[1] "USER NAME: @Rinconero #TROPA"
[1] "SCREEN NAME: rinconero43"
[1] "LOCATION: Algun lugar de Venezuela"
[1] "TWEETS NUMBER: 27858"
[1] "FOLLOWERS: 7791"
[1] "DESCRIPTION: Del libro rojo,el poder debe radicar en el PUEBLO,con participacion y protagonismo... Sin miedo a denunciar a funcionarios INEPTOS y CORRUPTOS."
Example 2: REST API (source code)

```r
library(twitteR);
source('load-credentials.R');

# show favorites
show("FAVORITE TWEETS FOR USER wpmayor")
tweets <- favorites(user='wpmayor', n=3);
show(tweets);

# find trending topics by location
show("TRENDING TOPICS ASSOCIATED TO A LOCATION");
atl <- availableTrendLocations();
tweets <- getTrends(753692); # Barcelona
show(tweets);

# timelines
show("TIMELINE OF USER BaraackObama")
tweets <- userTimeline(user='BarackObama', n=3);
show(tweets);
```
Example 2: REST API (output)

source("TwitteR-example-2.R")

[1] "FAVORITE TWEETS FOR USER wpmayor"
[[1]]
[1] "eddwp: Oh hey, checkout what just showed up: http://t.co/V8mFbeCWJN"
[[2]]
[1] "LisaSabinWilson: Love #WordPress? Good with front-end and theme dev? Want to work from home? We’re hiring at @webdevstudios - http://t.co/024viilgid"
[[3]]
[1] "jchristopher: Wow @irontoiuron is turning two _tomorrow_! Trying to grasp"

[1] "TRENDING TOPICS ASSOCIATED TO A LOCATION"
name url query woeid
1 Pinilla http://twitter.com/search?q=Pinilla Pinilla 753692
2 Vamos Colombia http://twitter.com/search?q=%22Vamos+Colombia%22 %22Vamos+Colombia%22 753692
3 Scolari http://twitter.com/search?q=Scolari Scolari 753692
4 Hulk http://twitter.com/search?q=Hulk Hulk 753692
5 Julio Cesar http://twitter.com/search?q=%22Julio+Cesar%22 %22Julio+Cesar%22 753692
[[...]]

[1] "TIMELINE OF USER BaraackObama"
[[1]]
[1] "BarackObama: Support @OFA today and keep fighting for change: http://t.co/GRYYEpJzdD http://t.co/7xw6aQIusx"
[[2]]
[1] "BarackObama: Skills. cc #USMNT http://t.co/hzUHPsz3t9"
Try it yourself! Play around with...

tweets <- searchTwitter("#Obama", n=10, lang="en", since="2014-06-01", until="2014-07-03");

user <- getUser(tweet$getScreenName());

tweets <- favorites(user='wpmayor', n=3);

atl <- availableTrendLocations();
tweets <- getTrends(753692); # Barcelona

tweets <- userTimeline(user='BarackObama', n=3);
Example 1: STREAMING API

```r
# load packages
library(streamR);

# load credentials
source('load-credentials.R');

# connect to Twitter stream a get messages
filterStream("tweets.json", track = c("Obama", "Putin"), timeout = 60, oauth = cred);

# parse tweets
tweets.df <- parseTweets("tweets.json", simplify = TRUE);

# compute some measures
show(paste("Number of tweets with #Obama: ", length(grep("Obama", tweets.df$text, ignore.case = TRUE))));
show(paste("Number of tweets with #Putin: ", length(grep("Putin", tweets.df$text, ignore.case = TRUE))));

source("StreamR-example-1.R")
```

Capturing tweets...
Connection to Twitter stream was closed after 61 seconds with up to 101 tweets downloaded.
51 tweets have been parsed.
[1] "Number of tweets with #Obama: 34"
[1] "Number of tweets with #Putin: 5"
Example 2: STREAMING API

```r
# load packages
library(streamR);
library(ggmap);

# load credentials
source('load-credentials.R');

# get tweets from specified location
filterStream("tweetsSpain.json", locations = c(-9, 35, 4, 44), timeout = 60, oauth = cred);
# parse tweets
tweets.df <- parseTweets("tweetsSpain.json", verbose = FALSE);
# get points set (lon/lat)
points <- data.frame(x = as.numeric(tweets.df$lon), y = as.numeric(tweets.df$lat));

# get map
spain <- get_map('Spain', zoom=6);
spainMap <- ggmap(spain, extent='device', legend='topleft');
# save map to file
png(file='map.png', width=640, height=640, units='px', pointsize=12);
# map + points of lon/lat from tweets
print(spainMap + geom_point(aes(x = x, y = y), data = points, colour = 'red', size = 1));
dev.off();

source("StreamR-example-2.R")
```
Let's do it!

Example 2: STREAMING API (output)
Example 2: REST API (activity)

Try it yourself! Play around with...

```r
filterStream("tweets.json", track = c("Obama", "Putin"), timeout = 60, oauth = cred);
filterStream("tweetsSpain.json", locations = c(-9, 35, 4, 44), timeout = 60, oauth = cred);
```
Thanks for attending!

cristina.perez@deic.uab.cat