



MET CS688 C1

WEB ANALYTICS AND MINING

ZLATKO VASILKOSKI

TWITTER

Social Network Analysis, Origins and Applications

- With theoretical roots in work by sociologist Georg Simmel:
- Social Network Analysis (SNA) methods emerged in the 1930s. The founder of what came to be known as the field of sociometry—the study of social network phenomena—was Jacob Moreno. One of Moreno's early insights was that social structure derived from individuals' personal preferences for affiliating with each other. Contracted by a New York girls' reformatory following a rash of runaways: he and his assistant, Helen Jennings attempted to apply Moreno's ideas about preference-based social structures to the assignment of the girls to living cottages and dining room tables with each stage of their investigation: they painstakingly constructed social network diagrams—or sociograms—depicting the relationship preferences of the 505 girls at the reformatory. Some of these hand-drawn socio grams were published in the journal Sociometry, which Moreno founded in 1937.
- Reference: “Social Analytics” by Shaila Miranda

Social Network Analysis, Origins and Applications

- A second landmark SNA study was Stanley Milgram's series of experiments in the 1960s that came to be known as the small world experiments. His question was, "Given any two people in the world, person X and person Z, how many intermediate acquaintance links are needed before X and Z are connected?" [6: 62]. So he gave a folder to individuals located in Wichita, Kansas (because it seemed so remote!), and directed them to move the folder to the wife of a divinity student in Cambridge, Massachusetts, but only through people they knew personally. Each person through whom the folder moved added their name to a "tracer" card, enabling Milgram to construct the path through which the study participants in Kansas reached the target in Massachusetts.
- Replicating the study with participants in faraway Omaha, Nebraska, targeting a stockbroker, who worked in Boston, Massachusetts, and lived in Sharon, Massachusetts, Milgram found the number of intermediaries used to connect the source to the target ranged from two to ten, and averaged six [7]. This gave rise to the notion that, on average, there are six degrees of separation between any two randomly selected individuals. This notion later was popularized in the 1993 movie, starring Will Smith, and in the Kevin Bacon game!
- Reference: "Social Analytics" by Shaila Miranda

Social Network Analysis, Origins and Applications

- A high-impact application of SNA came in the late 1970s and early 1980s. Desperate to forestall the spread of the dreaded AIDS virus, CDC researchers used SNA to identify an Air Canada flight attendant as patient zero in North America [8]. Subsequent biological evidence indicates that the AIDS epidemic in North America, in fact, preceded the flight attendants' contraction of the AIDS virus [9], highlighting the limits of SNA. However, early researchers traced the emergence and diffusion of different viral strains based on social network analyses of patients' symptoms.
- Becoming increasingly mathematical, the field of sociometry grew through the influence of graph theory and matrix algebra, from where modern SNA borrows some terminology. Well into the 1980s, though, conducting SNA was a laborious process. Networks were hand-drawn or painstakingly rendered with a typewriter and later with rudimentary computer graphics software.
- Network metrics were simplistic and computed manually. Software for visualizing and analyzing social networks began to emerge in the 1990s [10, 11], popularizing the use of SNA and refining metrics available.
- Reference: "Social Analytics" by Shaila Miranda

Social Network Analysis, Origins and Applications

- Today, organizations use social network analysis to visualize traffic on computer networks. For example, SolarWinds, a company that provides enterprise information technology infrastructure management software for IT professionals, offers software for computer network performance monitoring, and load balancing.
- Epidemiologists continue to use SNA to identify individuals at high risk for infection, and to determine the "threshold" level of virus penetration that would result in a full-blown epidemic. Financial institutions can predict fraudulent behavior by tracing networks created through organizations' resource exchanges.
- Researchers at MIT have developed a global language network, constituted by book, Wikipedia page, or Twitter post translations. They found that the position of the language one speaks in this network affects one's visibility and the recognition one is likely to enjoy for one's contributions to the arts and sciences.
- Reference: "Social Analytics" by Shaila Miranda

Social networking service

- Social networking service (SNS) is a platform to build social networks or social relations among people who share connections (interests, activities, backgrounds).
- A SNS consists of a user profile, social links, and a variety of additional services.
- Most social network services are web-based and provide means for users to interact over the Internet, such as e-mail and instant messaging.
- SNS enable small organizations without the resources to reach a broader audience with interested users.
- This is a field where trends emerge rapidly.
- Data mining is used in this field by companies to improve their sales and profitability.
 - Create customer profiles
 - Customer demographics
 - Follow online behavior
- A recent strategy of many companies has been the purchase and production of "network analysis software" some aspects of which we're covering in this class.

Mining the Social Web

- Twitter (an online social networking service)
 - Registering Your Application with Twitter
 - Twitter R API (Application Program Interface)
 - Accessing Twitter User Information
 - Searching for Tweets
 - User TweetsTrends
 - Graphing of Friends and Followers
 - Applying Text Mining to Tweets
 - Sentiment Analysis

How does OAuth work?

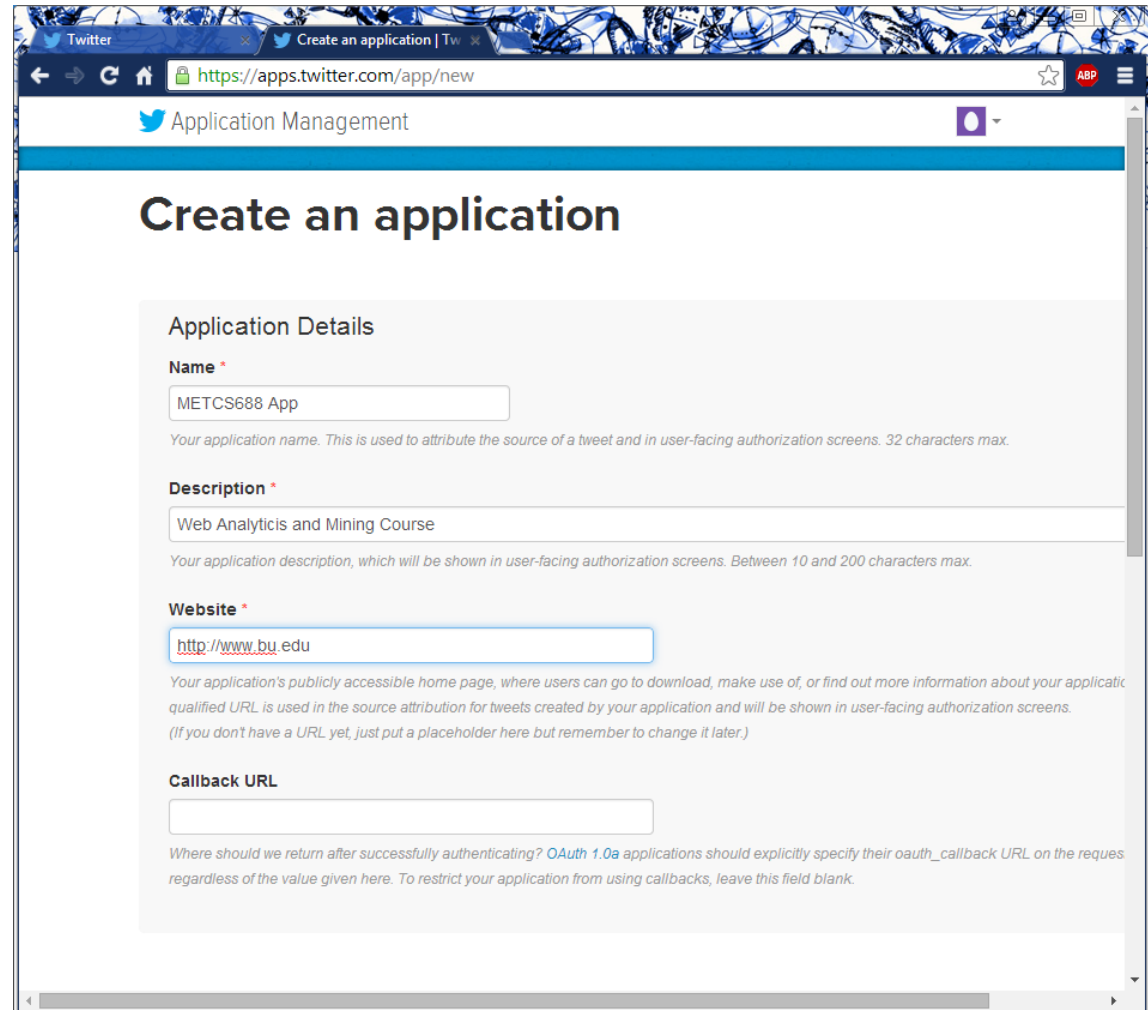
- **OAuth** allows you to approve one application interacting with another on your behalf **without giving away your password**.
- **OAuth** is an authentication protocol between **OAuth** client and provider (Facebook, Twitter etc.)
- **OAuth** allows users to approve application to act on their behalf without sharing their password.
- For this you need to create a developer account on the provider (Facebook, Twitter etc.).
- When a website wants to use the services of another—such as posting to your Twitter stream—instead of asking you to share your password, they should use OAuth instead.
- Revoke tokens for specific applications without affecting others.

Twitter Terminology

- It is an online social networking service
 - Registered users can read and post tweets.
 - Unregistered users can only read them.
 - Twitter is accessed through a website interface, SMS, or mobile device app.
 - Tweets are publicly visible, but some restrictions may apply.
 - Users may subscribe to other users' tweets – this is known as "**following**" and subscribers are known as "**followers**".
 - Users can group posts together by topic or type by using **hash tags** – words or phrases prefixed with a "#" sign.
 - "**Trending topic**" is a word, phrase or topic that is tagged at a greater rate than other tags.
 - A verified Twitter account is needed.
 - You can create new applications on the Twitter Developer Site at <https://dev.twitter.com/apps>. You must be logged-in to your Twitter account in order to do so.

Registering Your Application with Twitter – Step 1

- **OAuth** is an authentication protocol that allows users to approve application to act on their behalf without sharing their password.
- Check the development agreement and create the new application
- **R Packages** you need to install:
`install.packages("twitterR")`
`install.packages("ROAuth")`
`install.packages("RCurl")`
`install.packages("bitops")`
`install.packages("rjson")`



The screenshot shows the 'Create an application' page on the Twitter developer portal. The page is titled 'Create an application' and is part of the 'Application Management' section. It contains a form with the following fields:

- Name ***: A text input field containing 'METCS688 App'. Below it, a note states: 'Your application name. This is used to attribute the source of a tweet and in user-facing authorization screens. 32 characters max.'
- Description ***: A text input field containing 'Web Analytics and Mining Course'. Below it, a note states: 'Your application description, which will be shown in user-facing authorization screens. Between 10 and 200 characters max.'
- Website ***: A text input field containing 'http://www.bu.edu'. Below it, a note states: 'Your application's publicly accessible home page, where users can go to download, make use of, or find out more information about your application. A 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**: A text input field that is currently empty. Below it, a note states: 'Where should we return after successfully authenticating? OAuth 1.0a applications should explicitly specify their oauth_callback URL on the request regardless of the value given here. To restrict your application from using callbacks, leave this field blank.'

Registering Your Application with Twitter – Step 1

- Don't miss to fill out all the information necessary on the Twitter Developer Site.
- Fill out the "Settings" tab information on the App you just created.
- Make sure you enter the same "Callback URL" as the one you used on "Website".

METCS688 App

[Details](#) [Settings](#) [Keys and Access Tokens](#) [Permissions](#)

Application Details

Name *

METCS688 App

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

Description *

Web Analytics and Mining Course

Your application description, which will be shown in user-facing authorization screens. Between 10 and 120 characters.

Website *

http://www.bu.edu

Your application's publicly accessible home page, where users can go to download, make use of, or find 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

http://www.bu.edu

Where should we return after successfully authenticating? OAuth 1.0a applications should explicitly specify a callback URL. To restrict your application from using callbacks, leave this field blank.

- ☐ Enable Callback Locking (It is recommended to enable callback locking to ensure app security.)
- ☒ Allow this application to be used to [Sign in with Twitter](#)

Authorize Your App – Step 2

- Using *twitteR* version 1.1.8
- The OAuth authentication is carried out as:
 - Type this code
 - A browser opens
- Click “Sign in” and you get another browser window and message in R Studio:
 - Authentication complete.
- To test just type
- For everyday use:
You can save your twitter credentials

```
save(list=c("t.api.key", "t.api.secret"),  
file="twitter_credentials")
```

Example: Registering Your Application with Twitter

```
library("twitteR")  
library("ROAuth")  
t.api.key <- "xxxxxxxxxxxxxxxxxxxx"  
t.api.secret <- "xxxxxxxxxxxxxxxxxxxxxxxx"  
setup_twitter_oauth(t.api.key, t.api.secret, access_token=NULL,  
access_secret=NULL) # version 1.1.8
```

```
[1] "Using browser based authentication"
```

Waiting for authentication in browser...

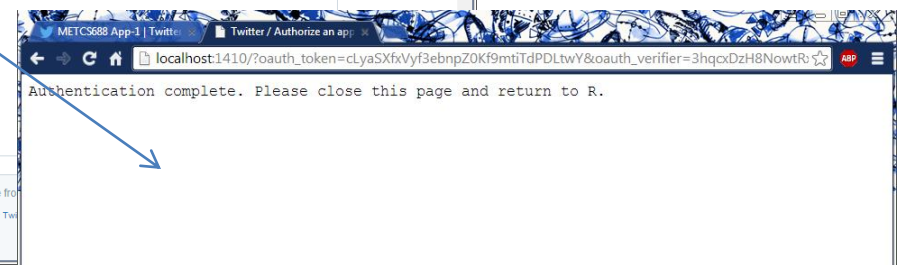
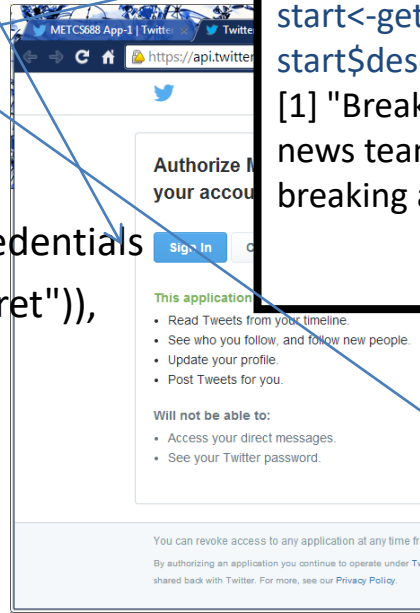
Press Esc/Ctrl + C to abort

To Test

```
start<-getUser("cnnbrk") # Users
```

```
start$description # You should get a description:
```

```
[1] "Breaking news from reporters and editors on the CNN Digital  
news team. Now 25M strong. Check @cnn for all things CNN,  
breaking and more."
```



Summary

For everyday use

```
library("twitteR"); library("ROAuth"); library("RCurl"); library("bitops"); library("rjson")  
load("twitter_credentials")  
setup_twitter_oauth(t.api.key, t.api.secret, access_token=NULL, access_secret=NULL)
```

To Test

```
start<-getUser("cnnbrk") # Users  
start$description # You should get a description
```

Twitter User Information

- In the following example, the twitter account of CNN Breaking News is used.
 - It has the handle @cnnbrk. The twitter page of the user's account is <https://twitter.com/cnnbrk>

- R Code:

```
start<-getUser("cnnbrk") # Users
```

```
# Fields
```

```
start$name
```

```
start$screenName
```

```
start$id
```

```
> start$description
```

```
[1] "Breaking news from reporters and editors on the CNN Digital news team. Now 25M strong. Check @cnn for all things CNN, breaking and more."
```

```
> start$lastStatus
```

```
[1] "Unknown: Dashcam video shows Arizona officer intentionally running into suspect.  
http://t.co/pqg6Q1S2SY."
```

- Many more examples given in the lecture notes



1.1 Twitter User Information

- The function `getUser("cnnbrk")` returns an object of the class `user` that models the information about a user on Twitter. Using the `user` object, the name, `screenName`, `id`, and `description` of the user can be obtained as follows.

```
> start$name
[1] "CNN Breaking News"
> start$screenName
[1] "cnnbrk"
> start$id
[1] "428333"
> start$description
[1] "Breaking news from reporters and editors
on the CNN Digital news team. Now 24M strong.
Check @cnn for all things CNN, breaking and
more."
```

- The same information can be accessed using the get methods or the direct methods as shown below.

```
# Using get methods
start$getName()
start$getScreenName()
start$getId()
start$getDescription()
```

```
# Using direct methods
name(start)
screenName(start)
id(start)
description(start)
```


1.2 Twitter User Information

- The *followersCount* gives the number of followers for the specified user. The *favoritesCount* gives the number of favorites for this user. The *friendsCount* shows the number of followees for this user. The location of the user can also be obtained.

```
> start$followersCount
[1] 24569350
> start$favoritesCount
[1] 13
> start$friendsCount
[1] 114
> start$location
[1] "Everywhere"
```

- The number of status updates for the specified user and the last status update of this user can be obtained as follows. The status object is the container for Twitter status messages and has the information about the text of the tweet, when it was created, how many times it has been retweeted, etc.

```
> start$statutesCount
[1] 38363
> start$lastStatus
[1] "Unknown: North Korea fires two short-
range ballistic missiles from its west
coast, South Korean defense official says.
http://t.co/cQA6zS04qf"
```

```
> status <- start$lastStatus
> status$text
[1] "North Korea fires two short-range
ballistic missiles from its west coast,
South Korean defense official says.
http://t.co/cQA6zS04qf"
> status$created
[1] "2015-03-01 23:49:38 UTC"
> status$retweetCount
[1] 659
```


1.3 Twitter User Information

- The twitter IDs of the followers of the specified user can be obtained using the *getFollowerIDs* & *getFriendIDs* method. The argument *n* is used for limiting the number of values retrieved. If the argument is omitted, all the followers are returned.

```
> start$getFollowerIDs(n=10)
[1] "3064095063" "3054320418"
[3] "3054387835" "3054391519"
[5] "2977784496" "171089762"
[7] "3054386923" "3054390175"
[9] "2856686675" "3054320004"
```

```
> start$getFriendIDs(n=10)
[1] "20522862" "166211200" "19416833"
[4] "16299754" "14601733" "17911562"
[7] "309177754" "620618811" "14515799"
[10] "22129280"
```

- The *getFollowers* & *getFriends* method returns a list of user objects. The optional argument limits the number of followers returned.

```
> start$getFollowers(n=5)
$`3064095063`
[1] "Brandon49077674"

$`3054320418`
[1] "factorymate001"

$`3054387835`
[1] "RiekoRinn"

$`3054391519`
[1] "andreschavez971"

$`2977784496`
[1] "lakho_masood"
```

```
> start$getFriends(n=5)
$`20522862`
[1] "JohnBerman"

$`166211200`
[1] "maureenpetrosky"

$`19416833`
[1] "saeed_ahmed"

$`16299754`
[1] "RamCNN"

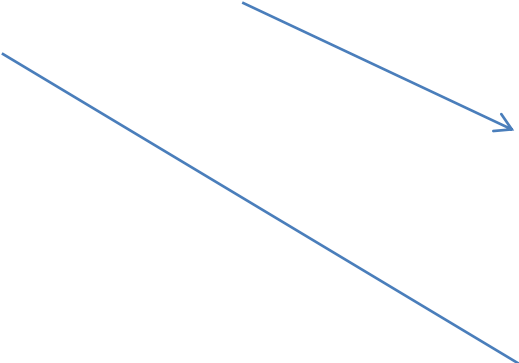
$`14601733`
[1] "FromCarl"
```

2.1 Searching for Tweets by Tag

- The `searchTwitter` method is used for searching Twitter using the specified search string.
- The method returns a list of twitter *status* objects.
- The *status* object is the container for the tweet.
- The optional argument *n* (default value 25) specifies the maximum number of tweets to be returned by the search.

> tweets1 <- searchTwitter('#bigdata', n = 5) # R List of 5 tweets

- User defined function can be used to display the information about each tweet.
- And displayed



```
# Example : Searching for Tweets
display.tweet <- function (tweet) {
  cat("Screen name:", tweet$getScreenName(),
      "\nText:", tweet$getText(), "\n\n")
}

# Display 5 Tweets
for (t in tweets1) {
  display.tweet(t)
}
```

2.2 Searching for Tweets by User

- The tweets from a specific user's timeline can be retrieved using the *userTimeline* function.
- In the following example, the first five tweets from the twitter user “kdnuggets” are retrieved.
 - > tweets2 <- userTimeline("kdnuggets", n = 5) # Get 5 tweets from user “kdnuggets”
 - The previously user defined function can be used to display the information about each tweet.
- The tweets on the authenticated user's timeline can be retrieved using the *homeTimeline* method as shown below.
 - > tweets3 <- homeTimeline(n = 5) # Get the timeline for the 5 tweets
- Compare with https://twitter.com/kdnuggets?ref_src=twsrc%5Egoogle%7Ctwcamp%5Eserp%7Ctwgr%5Eauthor

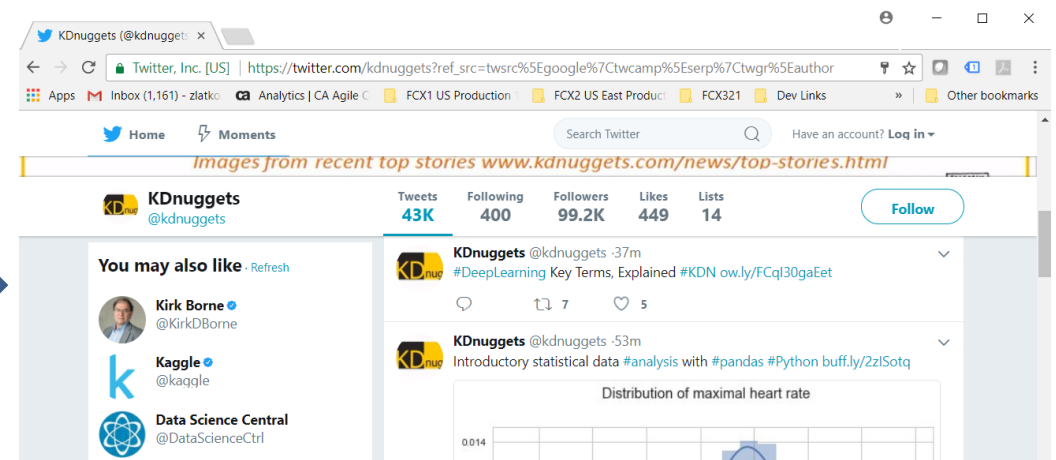
```
> tweets2
[[1]]
[1] "kdnuggets: #DeepLearning Key Terms, Explained #KDN https://t.co/SYXhrUQs0C"

[[2]]
[1] "kdnuggets: Introductory statistical data #analysis with #pandas #Python https://t.co/LfHYLnYXfs https://t.co/8TF63F7uKj"

[[3]]
[1] "kdnuggets: The 10 Statistical Techniques #DataScientists Need to Master https://t.co/rybFEsf5sz #DataScience #statistics https://t.co/jIUQ62dy05"

[[4]]
[1] "kdnuggets: The latest The KDnuggets Observer! https://t.co/n54uK7WmEq #kdn #datascience"

[[5]]
[1] "kdnuggets: Top #MachineLearning #MOOCs and Online Lectures: A Comprehensive Survey #KDN https://t.co/a8zmwqE9GU"
```



3. Trends

- “**Trending topic**” is a word, phrase or topic that is tagged at a greater rate than other tags.
- The twitter homepage shows the worldwide trends by default identifying the topics that are popular at the moment.
- The available trend locations can be obtained as shown below. The *woeid* is a numerical identification code describing a location (Where on Earth ID).

> head(trends.locations)

name	country	woeid
1	Worldwide	1
2	Winnipeg Canada	2972
3	Ottawa Canada	3369
4	Quebec Canada	3444
5	Montreal Canada	3534
6	Toronto Canada	4118

```
# Example : Trends
display.trend <- function (trend) {
  cat("Name:", trend$name,
      "\n url:", trend$url, "\n\n")
}

# AvailableTrends
trends.locations <- availableTrendLocations()
head(trends.locations)
```

- The *getTrends* function returns the current trends for the specified *woeid*.

3) Trends

The twitter homepage <http://trends24.in/> shows the worldwide trends.

```
trends.location.woeid <- trends.locations[1, "woeid"]
trends1 <- getTrends(trends.location.woeid)
for (i in 1:nrow(trends1)) {
  display.trend(trends1[i,])
}
```

Displays:

Name: #Elegidos url:

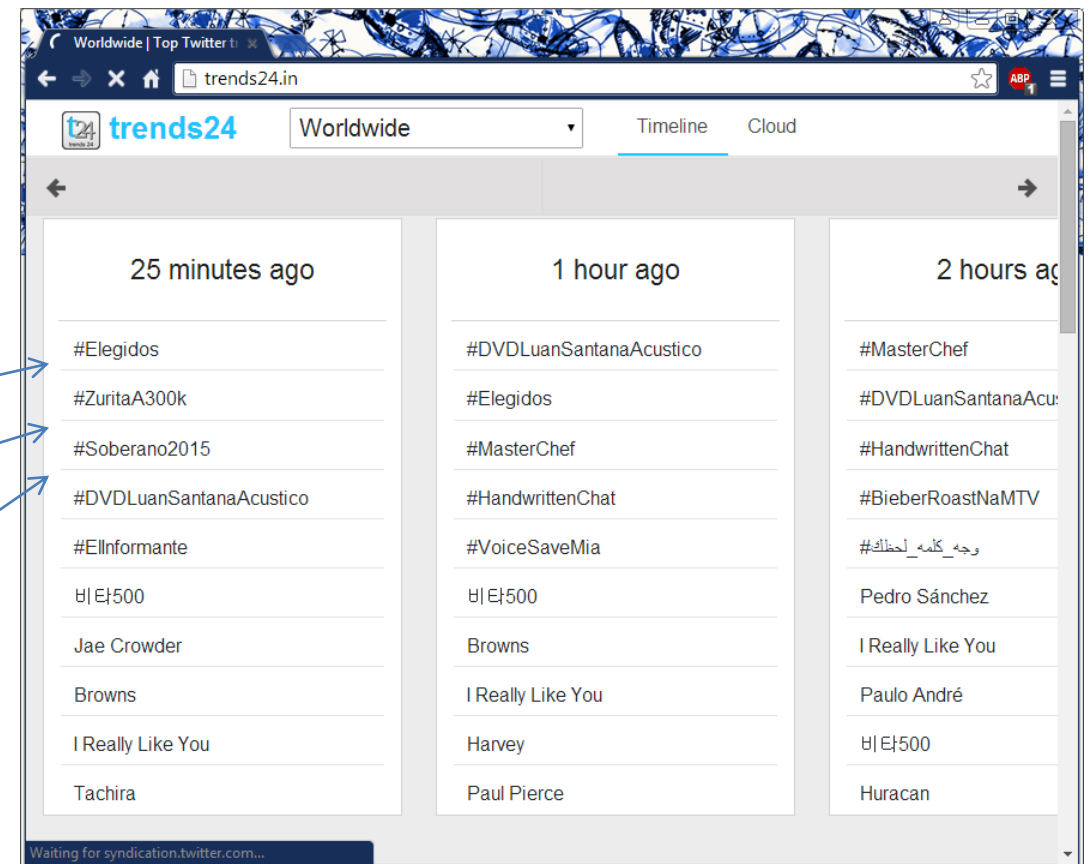
<http://twitter.com/search?q=%23Elegidos>

Name: #ZuritaA300k url:

<http://twitter.com/search?q=%23ZuritaA300k>

Name: #Soberano2015 url:

<http://twitter.com/search?q=%23Soberano2015>



4) Exercise: Graphing of Friends and Followers

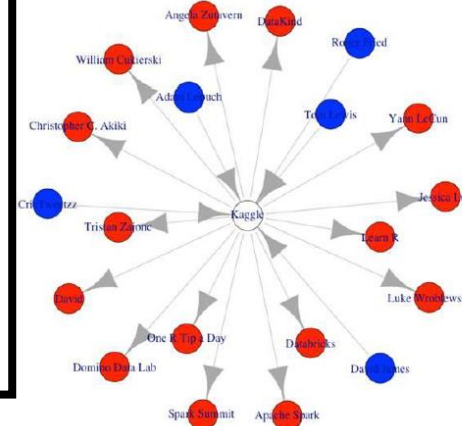
Task: Graph the connections between a specific user and the user's friends and followers.

- Check out the related `twitter_code.R` on Blackboard.
- You can get friends and followers' names and map them into a vector and data frame.
- Merge the 2 data frames and plot them

Example : Friends/Followers Tweets

```
friends <- lookupUsers(user$getFriendIDs(n=15))
followers <- lookupUsers(user$getFollowerIDs(n=5))
friends.names <- sapply(friends, name)
followers.names <- sapply(followers, name)
friends.frame <- data.frame(User=user.name, Follower=friends.names)
followers.frame <- data.frame(User=followers.names, Follower=user.name)
relations <- merge(friends.frame, followers.frame, all = T)
```

```
library(igraph)
g <- graph.data.frame(relations, directed = T)
V(g)[2:6]$color <- 'blue'
V(g)[7:21]$color <- 'red'
plot(g)
```



```
> relations
      User      Follower
1    Kaggle  Angela Zutavern
2    Kaggle   Apache Spark
3    Kaggle Christopher C. Akiki
4    Kaggle   Databricks
5    Kaggle   DataKind
6    Kaggle    David
7    Kaggle  Domino Data Lab
8    Kaggle   Jessica Luo
9    Kaggle   Learn R
10   Kaggle  Luke Wroblewski
11   Kaggle  One R Tip a Day
12   Kaggle   Spark Summit
13   Kaggle  Tristan Zajonc
14   Kaggle  William Cukierski
15   Kaggle   Yann LeCun
16 Adam Lopuch      Kaggle
17 CricTweetzz      Kaggle
18 David James      Kaggle
19 Roger Fried      Kaggle
20   Tom Lewis      Kaggle
```

5) Applying Text Mining to Tweets

- For text mining on the twitter data use the following packages
`library(tm)`
`library(SnowballC)`
- The tweets from the user “kdnuggets” timeline are retrieved and the associated text for each tweet is extracted into a corpus as shown.
- Transformations can be applied to this corpus in a standard way using `tm_map()`.

As before:

- The term document matrix is created
- Word frequency is calculated
- Wordcloud is displayed

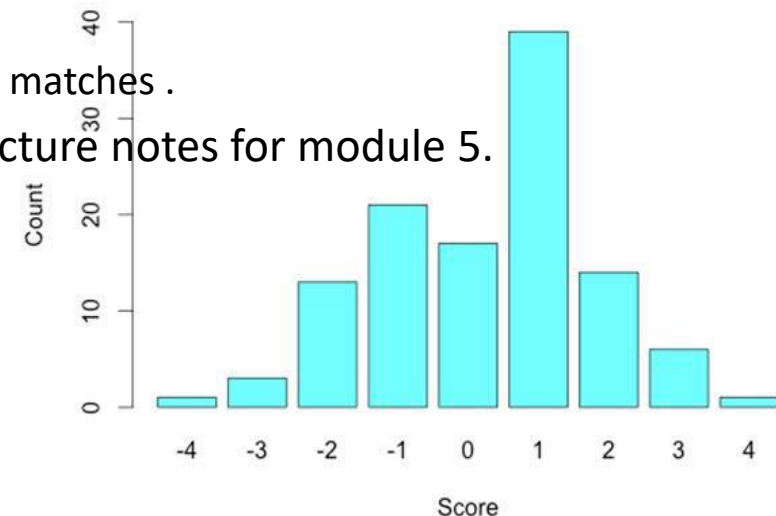
```
# Example : Text Mining Tweets
tweets.text <- lapply(tweets, function(t) {t$getText()})

# Standard Text Mining
data.source <- VectorSource(tweets.text)
data.corpus <- Corpus(data.source)

tdm <- TermDocumentMatrix(data.corpus)
```

6) Sentiment Analysis

- Sentiment analysis is used for analyzing the given text and determining whether it conveys a positive, negative or a neutral sentiment.
- In a simplistic model, the total number of positive words is compared against the total number of negative words.
- A vector of positive words, and a vector of negative words is considered.
- Look at the function “sentiment()” from module example code:
 - The function returns the difference between the number of positive matches and the number of negative matches.
 - Preprocessing removes punctuation, control characters, and digits .
 - Then lowercase text is turned into vector of words.
 - Positive matches and negative matches are then computed.
 - The function returns the difference between the number of positive and negative matches .
- Note the link to the lists of positive and negative words that is given in the lecture notes for module 5.
- The results are typically displayed graphically.



The Sentiment function

- Have a look at the lexicon to better understand what it is.



- Note how the lexicon is imported.



```
# Lexicons
pos.words = scan('positive-words.txt',
                 what='character',
                 comment.char=';')

neg.words = scan('negative-words.txt',
                 what='character',
                 comment.char=';')
```

- Note how the “sentiment” function works by pre-processing and matching.



```
366 sentiment <- function(text, pos.words, neg.words) {
367   text <- gsub('[:punct:]', '', text)
368   text <- gsub('[:cntrl:]', '', text)
369   text <- gsub('\\d+', '', text)
370   text <- tolower(text)
371   # split the text into a vector of words
372   words <- strsplit(text, '\\s+')
373   words <- unlist(words)
374   # find which words are positive
375   pos.matches <- match(words, pos.words)
376   pos.matches <- !is.na(pos.matches)
377   # find which words are negative
378   neg.matches <- match(words, neg.words)
379   neg.matches <- !is.na(neg.matches)
380   # calculate the sentiment score
381   score <- sum(pos.matches) - sum(neg.matches)
382   cat (" Positive: ", words[pos.matches], "\n")
383   cat (" Negative: ", words[neg.matches], "\n")
384   return (score)
385 }
386
```

```
34 ~~~~~
35
36 2-faced
37 2-faces
38 abnormal
39 abolish
40 abominable
41 abominably
42 abominate
43 abomination
44 abort
45 aborted
46 aborts
47 abrade
48 abrasive
49 abrupt
50 abruptly
51 abscond
52 absence
53 absent-minded
54 absentee
55 absurd
56 absurdity
57 absurdly
58 absurdness
59 abuse
60 abused
61 abuses
62 abusive
63 abysmal
64 abysmally
65 abyss
66 accidental
67
```

The upgraded Sentiment function

- Documents with equal number of positive and negative words are useless.
- Eliminating them will improve the sentiment analysis.
- You can come up with your own lexicon for a practical problem and improvements to the sentiment function.

```
443 sentiment.na <- function(text, pos.words, neg.words) {  
444   text <- gsub('[:punct:]', '', text)  
445   text <- gsub('[:cntrl:]', '', text)  
446   text <- gsub('\\d+', '', text)  
447   text <- tolower(text)  
448   # split the text into a vector of words  
449   words <- strsplit(text, '\\s+')  
450   words <- unlist(words)  
451   # find which words are positive  
452   pos.matches <- match(words, pos.words)  
453   pos.matches <- !is.na(pos.matches)  
454   # find which words are negative  
455   neg.matches <- match(words, neg.words)  
456   neg.matches <- !is.na(neg.matches)  
457   # calculate the sentiment score  
458   p <- sum(pos.matches)  
459   n <- sum(neg.matches)  
460   if (p == 0 & n == 0)  
461     return (NA)  
462   else  
463     return (p - n)  
464 }  
465
```

Accessing Twitter without an account

Twitter homepage <https://twitter.com/search-home> provides a quick look at what's the latest on this social network for those who aren't registered users. Note operators & advanced search

See what's happening right now

Tip: use **operators** for advanced search.

Search

Trends for you

#ImaDoMe

 Promoted by TikTok

#GivingTuesday patriots #WednesdayWisdom #HallmarkMoviesIn5Words Powell
#NationalFrenchToastDay Trey Songz Margaret Atwood Jeffrey Epstein

Search Operators

Operator	Finds tweets...
twitter search	containing both "twitter" and "search". This is the default operator.
"happy hour"	containing the exact phrase "happy hour".
love OR hate	containing either "love" or "hate" (or both).
beer -root	containing "beer" but not "root".
#haiku	containing the hashtag "haiku".
from:alexiskold	sent from person "alexiskold".
to:techcrunch	sent to person "techcrunch".
@mashable	referencing person "mashable".
"happy hour" near:"san francisco"	containing the exact phrase "happy hour" and sent near "san francisco".
near:NYC within:15mi	sent within 15 miles of "NYC".
superhero since:2010-12-27	containing "superhero" and sent since date "2010-12-27" (year-month-day).
ftw until:2010-12-27	containing "ftw" and sent up to date "2010-12-27".
movie -scary :)	containing "movie", but not "scary", and with a positive attitude.
flight :(containing "flight" and with a negative attitude.
traffic ?	containing "traffic" and asking a question.
hilarious filter:links	containing "hilarious" and linking to URLs.
news source:twitterfeed	containing "news" and entered via TwitterFeed

Accessing Twitter without an account

- For example searching for the tweets on Apple's stock "AAPL"

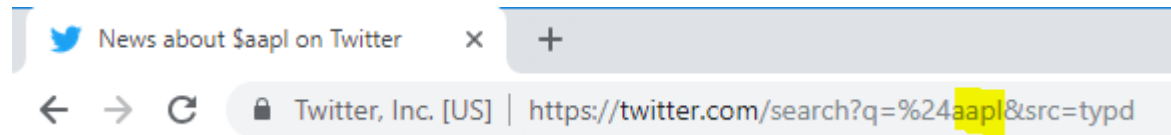
See what's happening right now

\$aapl

Tip: use [operators](#) for [advanced search](#).

Search

- Results in the following URL



Accessing Twitter without an account

- Using the package “rvest” and the following R code we can get the latest 20 tweets.
- Note how on line 21 the URL is composed to access the dynamic web page for “\$AAPL”
- The function Get.Tweets() scrapes the dynamic web page and returns the tweet text content

```
1 # Alternative Exercise without the Developer's Account
2 rm(list=ls()); cat("\014") # clear all
3 library(rvest)
4 library(tm)
5 library(SnowballC)
6
7 # ==== || ====
8 # Get the Tweets from https://twitter.com/search-home
9 Twitter.URL.Home <- "https://twitter.com/"
10
11 Hashtag <- "%23" # "#"
12 Dollar <- "%24" # "$"
13 Emoticon.happy <- "%20%3A)" # ":)"
14 Emoticon.unhappy <- "%20%3A(" # ":("
15
16 Get.Tweets <- function(URL){ # Gets tweet text from HTML content
17   tweets <- read_html(URL) %>% html_nodes(".js-tweet-text-container") %>% html_text(trim = TRUE)
18 }
19
20 Topic <- "AAPL"
21 Tweeter.URL <- paste0(Twitter.URL.Home, "search?q=", Dollar, Topic, "&src=typd")
22 AAPL.tweets <- Get.Tweets(Tweeter.URL)
23
```

Twitter Sentiment Analysis Exercise & Assignment

Task: Using the R Twitter API with a Developer's Account, OR the alternate without a Developer's Account, do the following:

1. Pick two topics, “#RedSox” and “#Yankees”, and search for the tweets associated with these two hashtags.
2. Create separate (two) data corpora for the above two sets of tweets.
3. Use the pre-processing transformations described in the lecture.
4. Create the term-document matrix for each hashtag.
5. Compare the frequent terms from each hashtag.
6. Show word cloud for each hashtag.
7. Using the positive and negative word lists, compute the sentiment score (as described in the lecture) for all the tweets for each hashtag.
8. Create a bar chart and draw your inference.
9. Submit your R code as a zipped project folder named “Your First Name Your Last Name”

Twitter Exercise - Solution

Using package “twitterR” with a Developer's Account

Exercise 2:

a) Pick two words, USA and China, and search for the 50 tweets
associated with these two terms.

```
tweets1 <- searchTwitter('usa', n = 50)
tweets2 <- searchTwitter('china', n = 50)
```

Alternative without the Developer's Account

Exercise 2:

a) Pick two words, USA and China, and search for
the 20 tweets associated with these two terms.

```
Topic1 <- "china"
Topic2 <- "usa"
```

```
Twitter.URL_1 <-
paste0(Twitter.URL.Home,"search?q=",Hashtag,Topic1
,"&src=typd")
Twitter.URL_2 <-
paste0(Twitter.URL.Home,"search?q=",Hashtag,Topic2
,"&src=typd")
```

```
tweets1 <- Get.Tweets(Twitter.URL_1)
tweets2 <- Get.Tweets(Twitter.URL_2)
```

Twitter Exercise – Alternate Solution

Using package “twitterR” with a Developer's Account

```
library(tm); library(SnowballC)
# b) Create two separate data corpora for the above two sets of
tweets.
getCorpus <- function (tweets) {
  tweets.text <- lapply(tweets, function(t) {t$getText()})
  data.source <- VectorSource(tweets.text)
  data.corpus <- Corpus(data.source)
  return (data.corpus)
}
data.corpus1 <- getCorpus(tweets1)
data.corpus2 <- getCorpus(tweets2)
```

Alternative without the Developer's Account

```
library(tm); library(SnowballC)
# b) Create two separate data corpora for the above
two sets of tweets.
getCorpus <- function (tweets) {
  # tweets.text <- lapply(tweets, function(t) {t$getText()})
  data.source <- VectorSource(tweets.text)
  data.corpus <- Corpus(data.source)
  return (data.corpus)
}
data.corpus1 <- getCorpus(tweets1)
data.corpus2 <- getCorpus(tweets2)
```


Twitter Exercise - Solution

c) Use the pre-processing transformations described in the lecture.

```
removeURL <- function(x) {  
  gsub("(http[^ ]*)", "", x) # Remove the URLs from the tweets  
}
```

```
removeNumberWords <- function(x) {  
  gsub("([[:digit:]]+)([[:alnum:]]*)", "", x)  
}
```

```
getTransCorpus <- function (data.corpus) {  
  data.corpus <- tm_map(data.corpus, content_transformer(removeURL))  
  data.corpus <- tm_map(data.corpus, content_transformer(removePunctuation))  
  data.corpus <- tm_map(data.corpus, content_transformer(tolower))  
  english.stopwords <- stopwords("en")  
  data.corpus <- tm_map(data.corpus, content_transformer(removeWords), english.stopwords)  
  data.corpus <- tm_map(data.corpus, content_transformer(removeNumberWords))  
  data.corpus <- tm_map(data.corpus, content_transformer(stemDocument))  
  data.corpus <- tm_map(data.corpus, content_transformer(stripWhitespace))  
  return (data.corpus)  
}
```

```
data.Trans.corpus1 <- getTransCorpus(data.corpus1)  
data.Trans.corpus2 <- getTransCorpus(data.corpus2)
```

Twitter Exercise - Solution

d) Create the term-document matrix for each state/country.

```
tdm1 <- TermDocumentMatrix(data.Trans.corpus1) # Build the term document matrix
```

```
tdm2 <- TermDocumentMatrix(data.Trans.corpus2) # Build the term document matrix
```

e) Compare the frequent terms from each state/country.

```
FFT1 <- findFreqTerms(tdm1, lowfreq=3) # frequent terms
```

```
FFT2 <- findFreqTerms(tdm2, lowfreq=3) # frequent terms
```

```
wordFreq1 <- rowSums(as.matrix(tdm1))
```

```
wordFreq2 <- rowSums(as.matrix(tdm2))
```

f) Show word cloud for each state/country.

```
library(wordcloud)
```

```
palette <- brewer.pal(8,"Dark2")
```

```
set.seed(137)
```

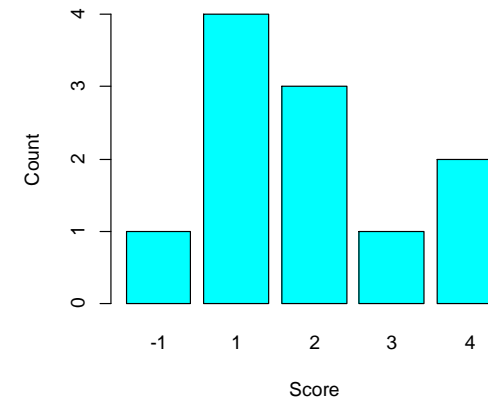
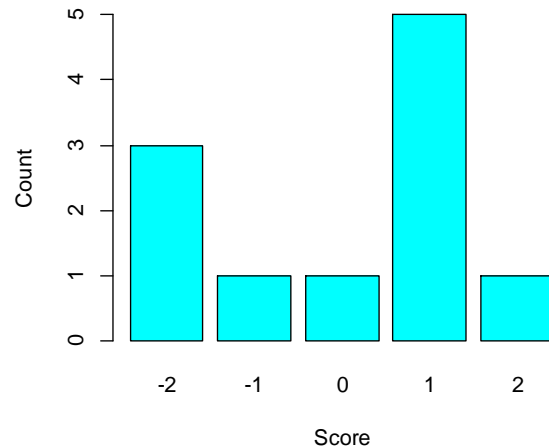
```
FFT1 # Display Words in Console
```

```
wordcloud(words=names(wordFreq1), freq=wordFreq1, min.freq=3, random.order=F, colors=palette)
```

```
FFT2 # Display Words in Console
```

```
wordcloud(words=names(wordFreq2), freq=wordFreq2, min.freq=3, random.order=F, colors=palette)
```

Twitter Exercise - Solution



Twitter Sentiment Analysis Exercise & Assignment

- Note that you can get the tweets for a given hashtag with happy and un-happy sentiment by adding a happy “:)” and un-happy “:(” emoticons.

```
# Sentiment Analysis using happy and un-happy emoticons
```

```
Hashtag <- "%23" # "#"
```

```
Dollar <- "%24" # "$"
```

```
Emoticon.happy <- "%20%3A)" # ":)"
```

```
Emoticon.unhappy <- "%20%3A(" # ":("
```

```
Topic <- "USA"
```

```
tweets1 <- Get.Tweets( paste0(Twitter.URL.Home,"search?q=",Hashtag,Topic,"%20%3A(", "&src=typd") )
```

```
tweets2 <- Get.Tweets( paste0(Twitter.URL.Home,"search?q=",Hashtag,Topic,"%20%3A(", "&src=typd") )
```

Facebook – Mining the Social Web

- Recently many companies have been utilizing the "network analysis software" for tasks such the ones covered in this class.
 - For example Europe's largest travel agency Booking.com uses very simple neural network software called ART developed few decades ago by Gail Carpenter here at BU's CNS (Cognitive Neural Systems).
- Facebook
 - Registering Your Application with Facebook
 - Facebook R API
 - Facebook User Information
 - Posts from Facebook Pages
 - Details from Facebook Posts
 - Search Facebook

Facebook

- Another (largest) online social networking service.
- Sells online advertising on their site.
- Apparently Apple accuses Facebook model in which users are defined as product and a commodity.
- Facebook and other social networking tools are increasingly the object of scholarly research.
- Facebook has been especially important to marketing strategists.
 - companies gain access to the millions of profiles in order to tailor their ads to a Facebook user's own interests and hobbies.
- Facebook sells tracked "social actions" that track the websites a user uses outside of Facebook through a program called Facebook Beacon. Related with privacy issues.

Finding package relevant information

<https://cran.r-project.org/web/packages/Rfacebook/Rfacebook.pdf>

Package 'Rfacebook'

March 3, 2016

Title Access to Facebook API via R

Description Provides an interface to the Facebook API.

Version 0.6.3

Date 2016-03-03

Author Pablo Barbera <pablo.barbera@nyu.edu>, Michael Piccirilli
<mrp2181@columbia.edu>, Andrew Geisler

Maintainer Pablo Barbera <pablo.barbera@nyu.edu>

URL <https://github.com/pablobarbera/Rfacebook>

BugReports <https://github.com/pablobarbera/Rfacebook/issues>

Depends R (>= 2.12.0), httr, rjson, httpuv

License GPL-2

NeedsCompilation no

Repository CRAN

Date/Publication 2016-03-03 23:30:55

R topics documented:

Rfacebook-package	2
callAPI	2
fbOAuth	3
getCheckins	4
getFQL	5
getFriends	6
getGroup	7
getInsights	8
getLikes	9
getNetwork	10
getNewsfeed	11
getPage	12
getPost	13
getUsers	14



getPage

Extract list of posts from a public Facebook page

Description

getPage retrieves information from a public Facebook page. Note that information about users that have turned on the "follow" option on their profile can also be retrieved with this function.

Usage

```
getPage(page, token, n = 80, since = NULL, until = NULL, feed = FALSE)
```

Arguments

page	A page ID or page name.
token	Either a temporary access token created at https://developers.facebook.com/tools/explorer or the OAuth token created with fbOAuth.
n	Number of posts of page to return. Note that number can be sometimes higher or lower, depending on status of API.
since	A UNIX timestamp or strtotime data value that points to the start of the time range to be searched. For more information on the accepted values, see: http://php.net/manual/en/function.strptime.php

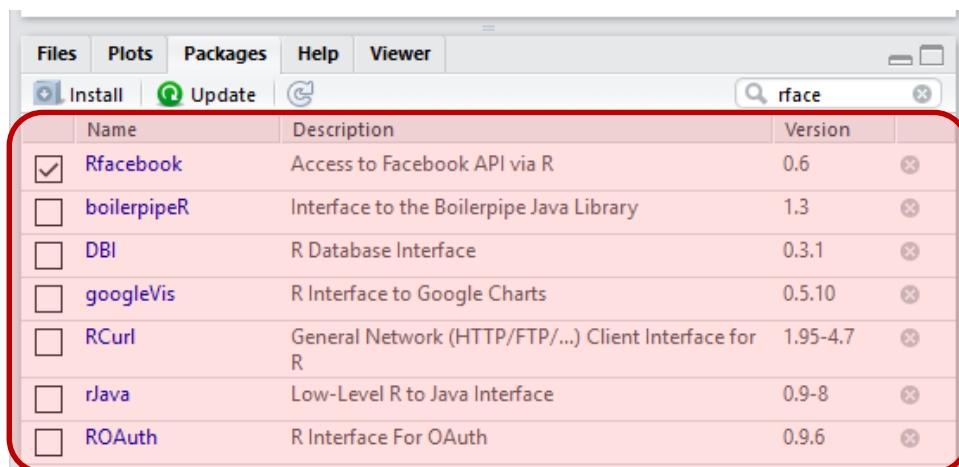
Facebook Platform Changelog

Note: R Packages often get updated, be aware of the changes. Providers keep Changelogs about them.

- <https://developers.facebook.com/docs/graph-api/using-graph-api>
- <https://developers.facebook.com/docs/apps/changelog>
- <https://cran.r-project.org/web/packages/Rfacebook/Rfacebook.pdf>

Introducing Graph API v2.6

The Graph API is the backbone of our developer platform, powering Facebook-integrated experiences around the world and serving an average of 890 billion API requests per day. In the latest version of the Graph API (v2.6), we're introducing new and updated features that enable you to build even richer app experiences across Messenger, Facebook Live, Reactions, Video, Rights Manager, Facebook Ads, and Sharing for devices.



Versions

Version	Path	Date Introduced	Available Until
v2.6	/v2.6/{object}	April 12, 2016	At least until April 2018
v2.5	/v2.5/{object}	October 7, 2015	April 12, 2018
v2.4	/v2.4/{object}	July 8, 2015	October 7, 2017
v2.3	/v2.3/{object}	March 25, 2015	July 8, 2017
v2.2	/v2.2/{object}	October 30, 2014	March 25, 2017
v2.1	/v2.1/{object}	August 7, 2014	October 30, 2016
v2.0	/v2.0/{object}	April 30, 2014	August 7, 2016
v1.0	/v1.0/{object}	April 21, 2010	Unavailable as of April 30, 2015

Facebook User Information – old way

- The public information about one or more Facebook users is retrieved using the `getUsers()` function.
`> user <- getUsers("me", token = fb.oauth)`
- You can find people's Facebook ID at <http://findmyfacebookid.com/>
`> user1 <- getUsers("60844829835", token = fb.oauth)`
`> user1$name`
`[1] "David Ortiz"`
`> user2 <- getUsers("216311481960", token = fb.oauth)`
`> user2$name`
`[1] "Bill Gates"`
- Given the Facebook username, the information about the Facebook ID of the user can be obtained through explicit Facebook Graph API call:
`http://graph.facebook.com/ David.Ortiz`
- The above call returns the data in JSON format. The *RJSONIO* and *RCurl* libraries can be used to convert the JSON data into an *R* data structure.
- Have a look at the function `getFacebookId()` in the lecture notes.

Registering Your Application with Facebook

- The first step is to register an application (App) on Facebook.

<https://developers.facebook.com/>

Click the “My Apps” menu and then click “Add a New App”.

Choose the Website option for the platform. Type a name for the new app, say METCS688.

Click the button to create the new Facebook App ID.

Select an appropriate category in the next page and click “Create App ID”.

Click “Skip Quick Start”.

In the Dashboard, click on the “Show” button to show the “App Secret”.

- Next time you want to use the Facebook R API just load your credentials:

```
> load(file="facebook_credentials") # (Use YOUR credentials)
```

```
> user <- getUsers("me", token = fb.oauth)
```

```
> fb.oauth <- fbOAuth(app_id = fb.app.id, app_secret = fb.app.secret, extended_permissions = TRUE)
Copy and paste into Site URL on Facebook App Settings: http://localhost:1410/
When done, press any key to continue...
Waiting for authentication in browser...
Press Esc/Ctrl + C to abort
Authentication complete.
>
```

Note from sample code on Blackboard which other libraries are used (install them if you haven't).

```
1 # Facebook
2
3 library("Rfacebook")
4 library("rjson")
5 library("httr")
6 library("httpuv")
7
```

Facebook User Information

- The public information about one or more Facebook users is retrieved using the *getUsers()* function.

```
> user <- getUsers("me", token = fb.oauth)
```

- You can find people's Facebook ID at <http://findmyfacebookid.com/>

```
> user1 <- getUsers("60844829835", token = fb.oauth)
```

```
> user1$name
```

```
[1] "David Ortiz"
```

```
> user2 <- getUsers("216311481960", token = fb.oauth)
```

```
> user2$name
```

```
[1] "Bill Gates"
```

- Given the Facebook username, the information about the Facebook ID of the user can be obtained through explicit Facebook Graph API call:

```
http://graph.facebook.com/ David.Ortiz
```

- The above call returns the data in JSON format. The *RJSONIO* and *RCurl* libraries can be used to convert the JSON data into an *R* data structure.
- Have a look at the function `getFacebookId()` in the "Social Media.pdf" notes.

Posts from Facebook Pages

- The public posts from the pages of Facebook users can be obtained and analyzed with the *Rfacebook* library.
- The *getPage* function retrieves the posts from the public Facebook page of the specified user.

```
> page <- getPage("CNN", n=100, token=fb.oauth)
```
- By default, the maximum number posts to retrieve are 100 (the parameter *n*). The most recent posts are the ones returned.
- Once obtained from these posts you can select
 - The categories of the retrieved posts
 - The most liked post
 - The most commented post
 - Their *id*

