Preconference on Computational tools for text mining, processing and analysis.
May 25th 2017, 9:00-17:00 (ICA San Diego)

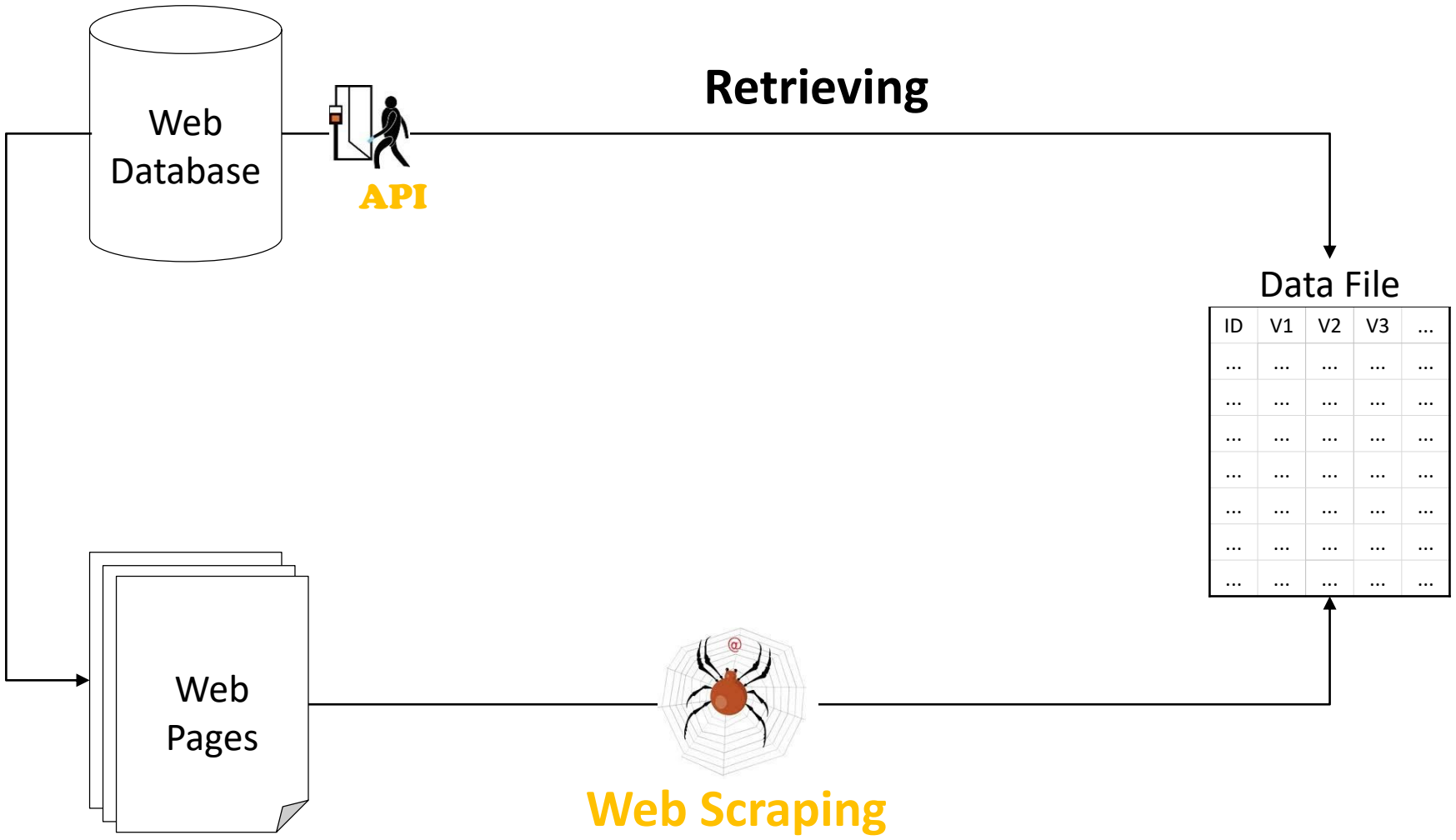
Scraping and Preprocessing of Social Media Data

HAI LIANG, ASSISTANT PROFESSOR

THE CHINESE UNIVERSITY OF HONG KONG

HAILIANG@CUHK.EDU.HK

WWW.DRHAILIANG.COM


Web Database
API
Retrieving
Data File
ID V1 V2 V3 ...
Web Pages
Web Scraping

Procedures – API

What is API:

API (Application Programming Interface) is a small script file (i.e., program) written by users, following the rules specified by the web owner, to download data from its database (rather than webpages)

An API script usually contains:

- Login information (if required by the owner)
- Name of the data source requested
- Name of the fields (i.e., variables) requested
- Range of date/time
- Other information requested
- Format of the output data
- etc.

Examples

http://www.reddit.com/user/TheInvaderZim/comments/

http://www.reddit.com/user/TheInvaderZim/comments**/.json**

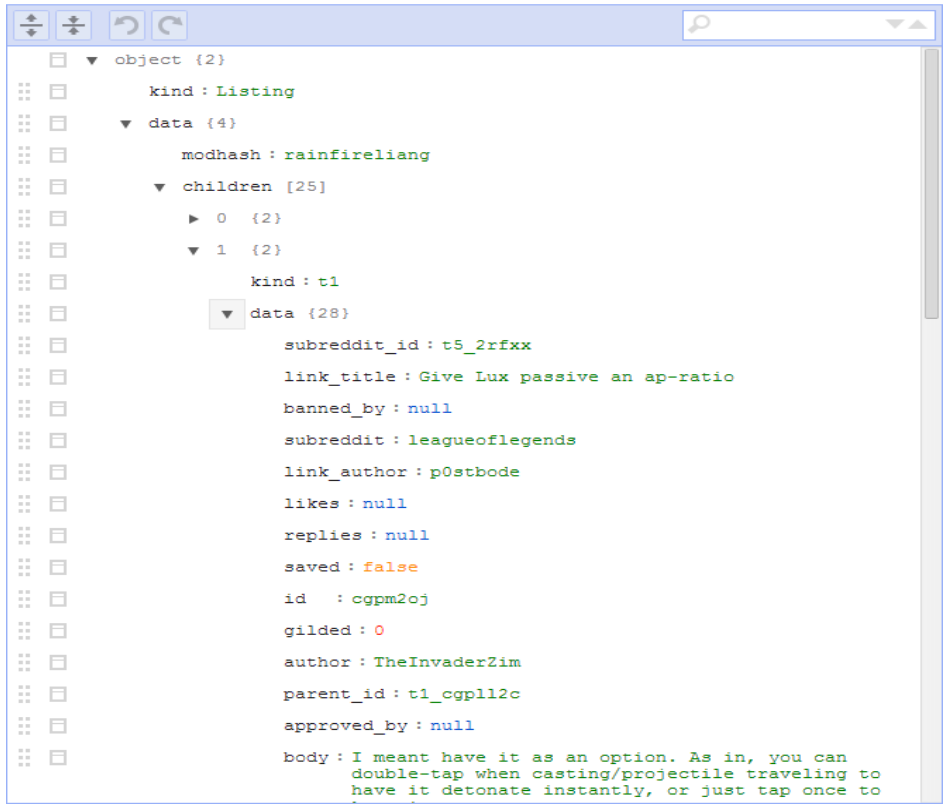
https://twitter.com/search?src=typd&q=%23tcot

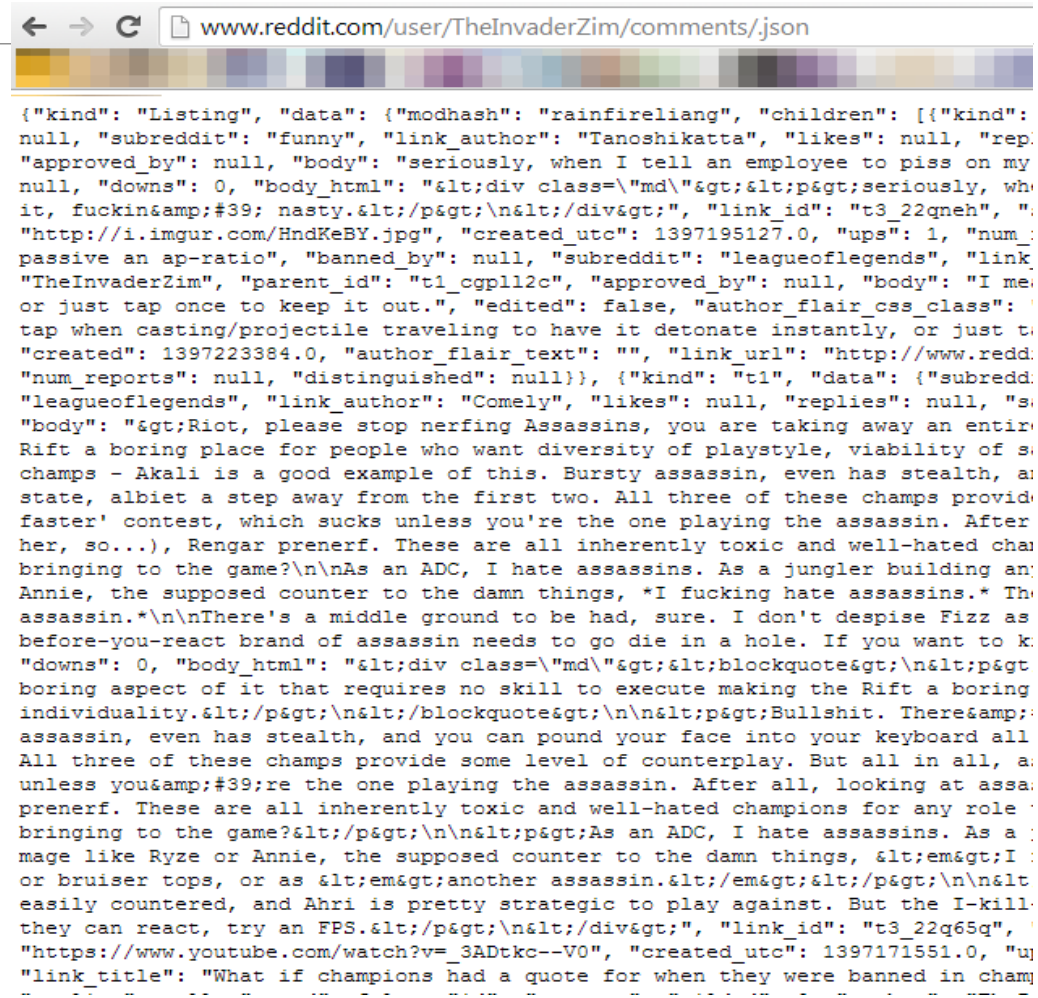
https://api.twitter.com/1.1/search/tweets.json**?src=typd&q=%23tcot**

http://api.nytimes.com/svc/search/v2/articlesearch.json?q=QUERY&**api-key=YOURS**

JSON Editor

http://www.jsoneditoronline.org/





Workflow of API

1. Identify API URLs
2. Get the JSON Webpage
3. Parse JSON Data
4. Save the content

1. URL = "http://api.nytimes.com/svc/search/v2/articlesearch.json?q=QUERY&api-key=YOURS"
2. webpage = RCurl::getURL(URL)
   - ✓ URLencode(URL)
3. data = rjson::fromJSON(webpage)
   - ✓ data$response$docs
4. write.table(xxx, file='xxx.txt', sep='\t\t', append=T)

Procedures – Web Scraping

1. Get the website – using HTTP library (e.g., RCurl/httr)
2. Parse the html document – using any parsing library (e.g., XML)
3. Store the results – either a db, csv, text file, etc.

- HTML Tags & Attributes

```
<html>
 <body>
  <p>
    <a class="title may-blank" href="http://www.nyu.com"> Li Ka-shing, Asia's richest man </a>
    <a class="title may-blank" href="http://www.cnn.com"> URA Director quits </a> (element node)
  </p>
 </body>
</html>
```

- XPath

CSS selector

Search node or attribute (e.g., python BeautifulSoup)

Regular expression

…

XPath:

- //div/*/a[@class='title']
- //div/*/a[0]/@href
- //*/tbody[contains(@id,'normalthread')]/*/td[@class="author"]/em

- XPath

Reading and examples: http://www.w3schools.com/xml/xml_xpath.asp

/ Select from the root

// Select anywhere in document

@ Select attributes. Use in square brackets

In this example, we select all elements of ‘a' …Which have an **attribute** “class” of the value “title may-blank” …then we select all links …and return all attributes labeled “href” (the urls)

"//a[@class='title may-blank']/@href"

* selects any node or tag

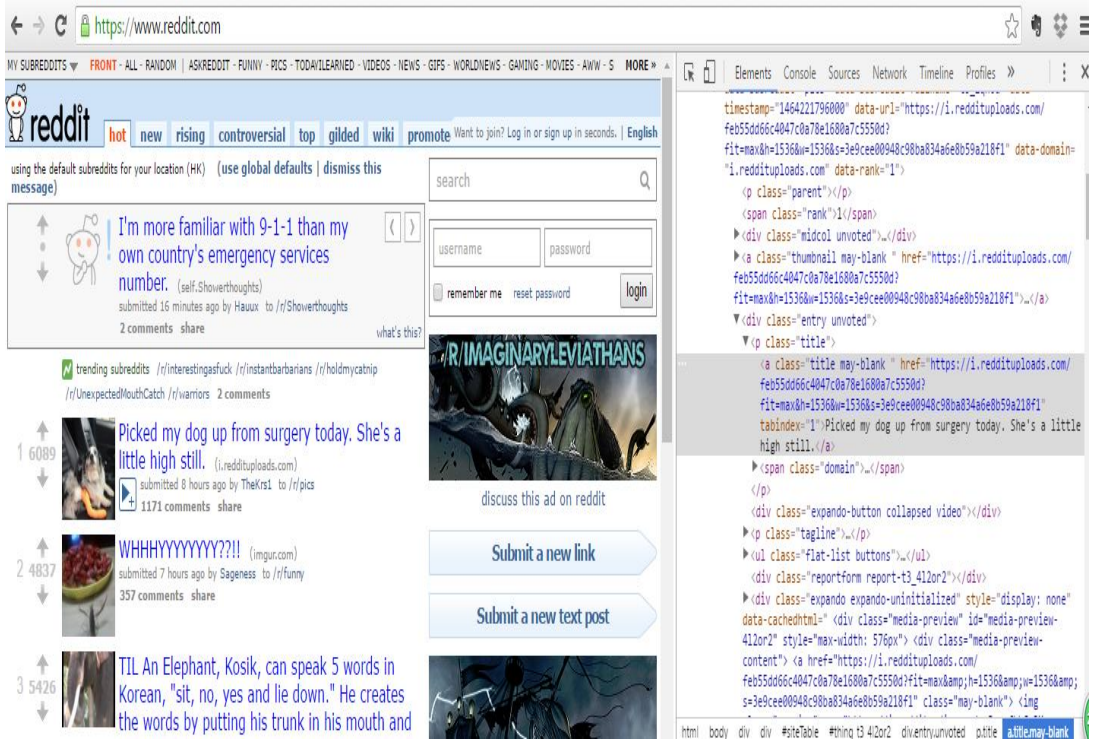
@* selects any attribute (used to define nodes)

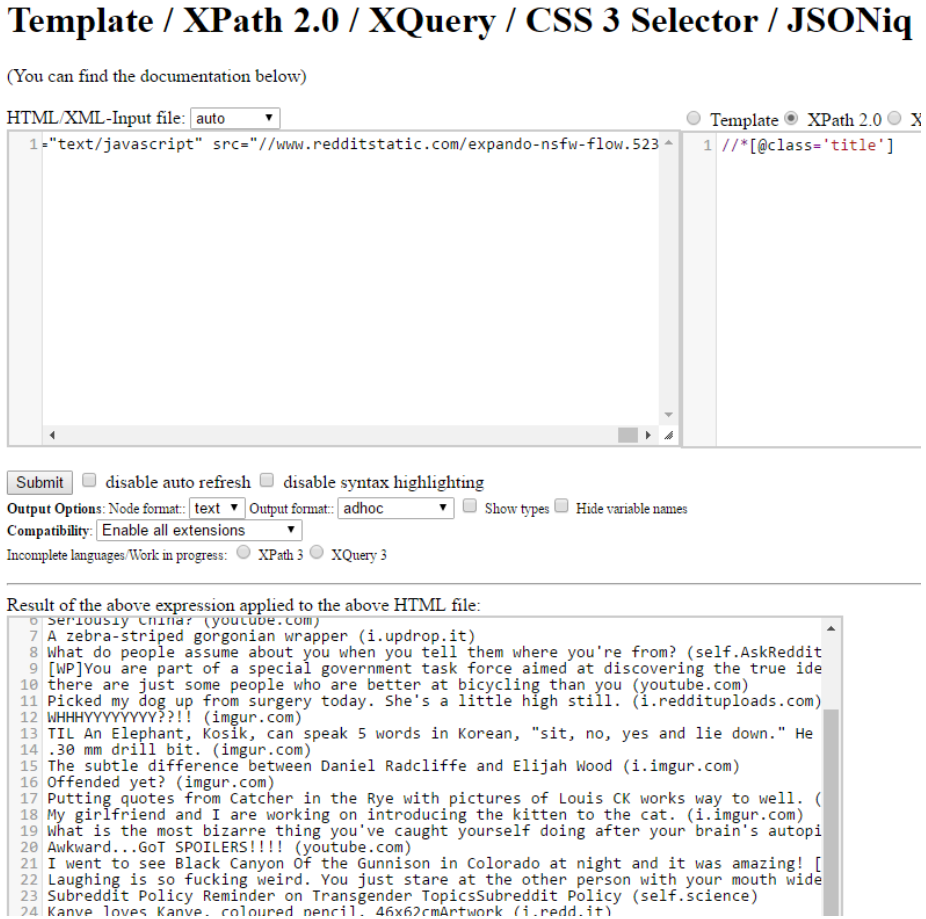
"//[@class='citation news'][17]/a/@href"*

"//span[@class='citation news'][17]/a/@"*

- XPath Online

http://videlibri.sourceforge.net/cgi-bin/xidelcgi





- Workflow of Scraping

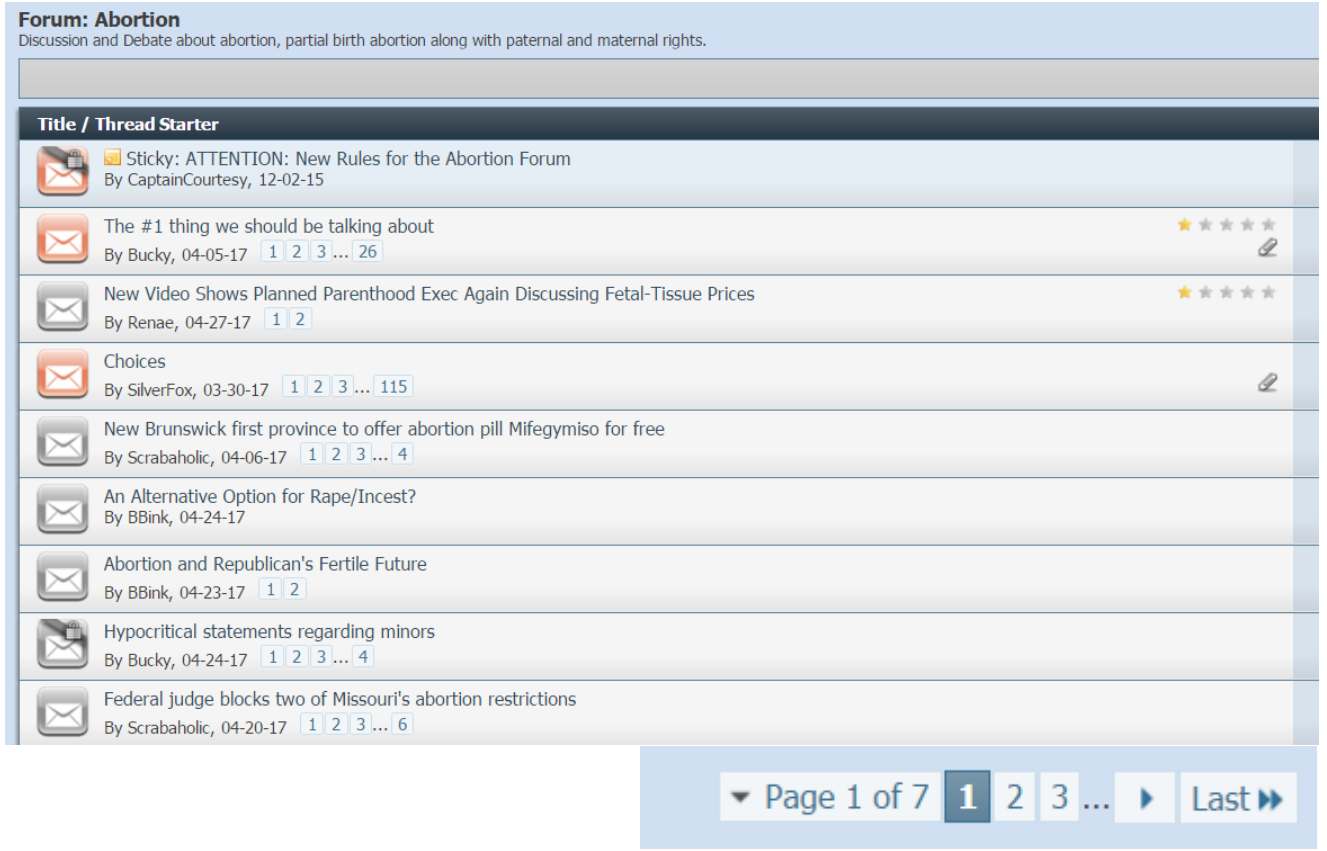
1. Get the website – using HTTP library
2. Parse the HTML document – using any parsing library
3. Store the results – either a DB, csv, text file, etc.

1. SOURCE = RCurl:: getURL(URL, .encoding="UTF-8")
   - ✓ SOURCE = iconv(SOURCE, "gb2312", "UTF-8")
2. PARSED = XML::htmlParse(SOURCE,encoding="UTF-8")
   - ✓ titles <- xpathSApply(PARSED, "//span[@class='c_tit']/a",xmlValue)
   - ✓ links <- xpathSApply(PARSED, "//span[@class='c_tit']/a/@href")
3. write.table(xxx, file='xxx.txt', sep='\t\t', append=T)

Crawling Rules

- URL pattern
- Next Page
- Random sampling
- Search
- Breadth first/depth first
- …

Start from a list



Crawling Rules

- URL pattern
- Next Page
- Random sampling
- Search
- Breadth first/depth first
- …

https://www.debatepolitics.com/abortion/index1.html

https://www.debatepolitics.com/abortion/index2.html

https://www.debatepolitics.com/abortion/index3.html

https://www.debatepolitics.com/abortion/index4.html

https://www.debatepolitics.com/abortion/index5.html

https://www.debatepolitics.com/abortion/index6.html

https://www.debatepolitics.com/abortion/index7.html

…

```
for (URL in URLs) {
    webpage = RCurl::getURL(URL)
    …
    Sys.sleep(sample(1:10, 1))
  }
```

Crawling Rules

- URL pattern
- Next Page
- Random sampling
- Search
- Breadth first/depth first
- …

```
library(doParallel)

cl = makeCluster(4)

registerDoParallel(cl)

df = foreach(i,…) %dopar% {

 res = f(i)

 return (res)
 }

stopCluster(cl)

RCurl::getURL(URL,
                useragent = “…”,
                referrer=“…”,
                proxy = "109.205.54.112:8080",
                followlocation = TRUE)
```

Crawling Rules

- URL pattern
- Next Page
- Random sampling
- Search
- Breadth first/depth first
- …

```
handle <- getCurlHandle(useragent
=str_c(R.version$platform,R.version$version.st
ring, sep=", "), httpheader = c(from
="ed@datacollection.com"), followlocation =
TRUE, cookiefile = "/tmp/cookies.txt")

postForm("http://example.com/login",
login="mylogin", passwd="mypasswd",
curl=handle)

getURL("http://example.com/anotherpage",
curl=handle)
```

Crawling Rules

- URL pattern
- Next Page
- Random sampling
- Search
- Breadth first/depth first
- …

//*[@class="next-button”]/a/@*href*

//*[contains(@class,”next”)]/a/@*href*

```
"paging": {
  "previous": "https://graph.facebook.com/v2.8/355665009819/posts?format=json&since=1484618455&access
  "next": "https://graph.facebook.com/v2.8/355665009819/posts?format=json&access_token=EAACEdEose0cBA
```

data$paging$next

```
while (!is.null(next)) {
    webpage = RCurl::getURL(URL)
    …
    next = xpathSApply(PARSED,'//*[contains(@class,”next”)]/a/@href')
    Sys.sleep(sample(1:10, 1))
  }
```

Crawling Rules

- URL pattern
- Next Page
- Random sampling
- Search
- Breadth first/depth first
- …

AJAX Websites:

- More stories
- Scroll down
- Mouse over

Selenium (Automated Browser)

- findElement() -> clickElement()
- Scrolling down

```
window.scrollTo(0, document.body.scrollHeight);
```

- Mouse over

```
((JavascriptExecutor)driver).executeScript("$('element_selector').hover();");
```

Crawling Rules

- URL pattern
- Next Page
- Random sampling
- Search
- Breadth first/depth first
- …

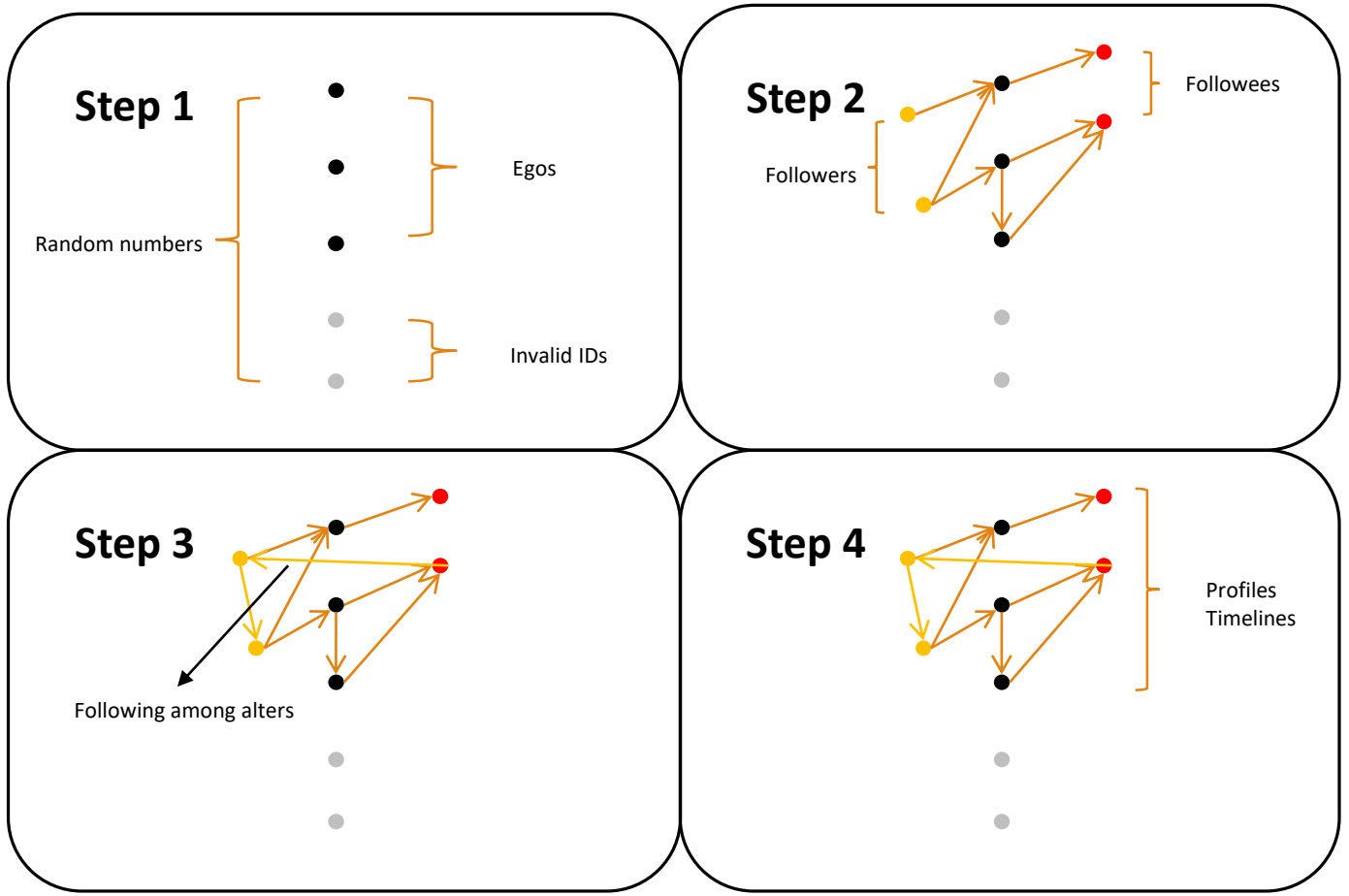
- Twitter ID is a unique (numeric) value that every account on Twitter has.

- An account can change its user name, it can never change its Twitter ID.

- Twitter ID ranges from 0 to 3,000,000,000 [Nov. 2014]—18 digits (since 2016)

Crawling Rules

- URL pattern
- Next Page
- Random sampling
- Search
- Breadth first/depth first
- …

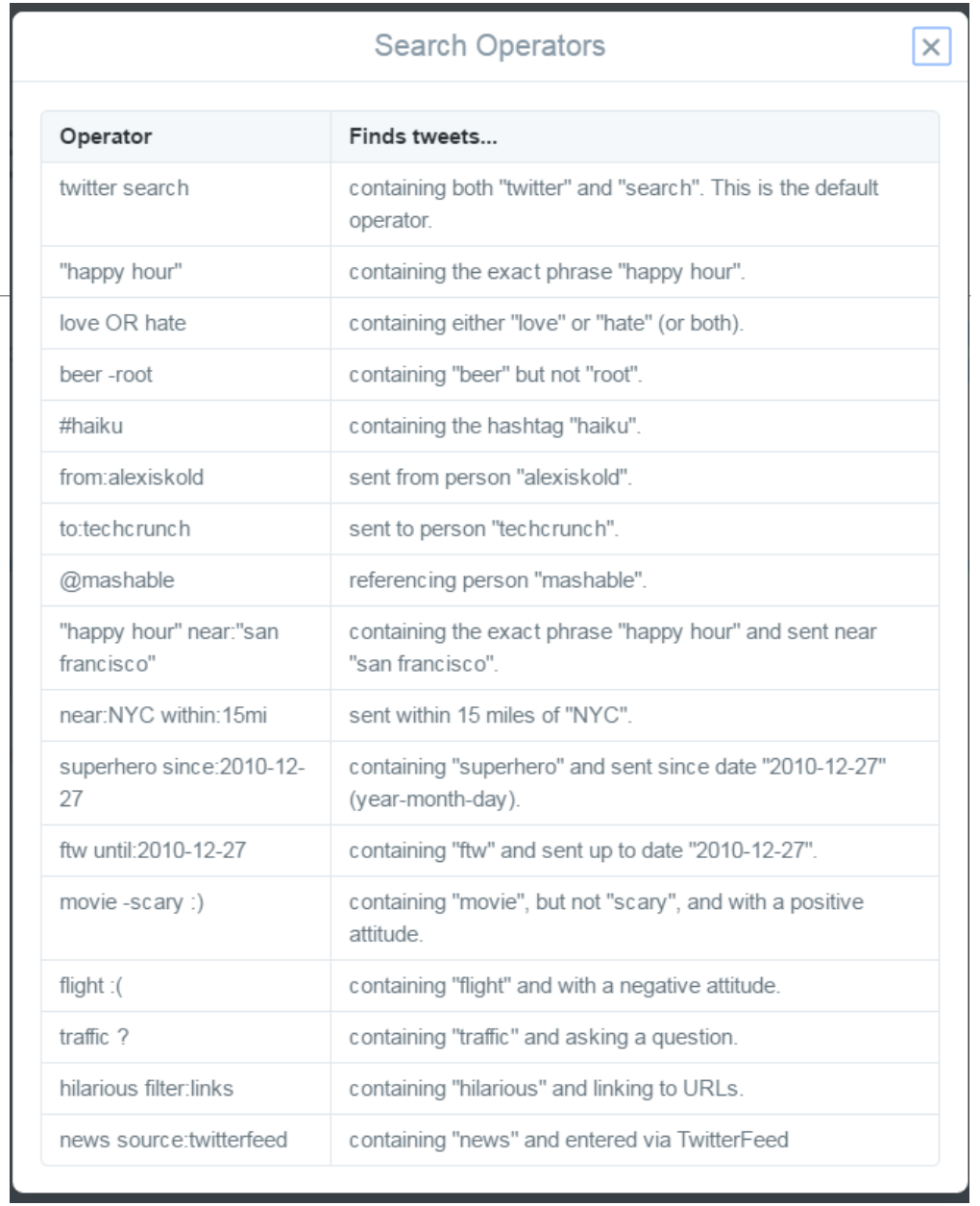


Source: Liang & Fu (2015) PLOS ONE

Crawling Rules

- URL pattern
- Next Page
- Random sampling
- Search
- Breadth first/depth first
- …

- By keywords
- By hashtags
- By authors
- By URLs
- …



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

Crawling Rules

- URL pattern
- Next Page
- Random sampling
- Search
- Breadth first/depth first
- …

library("twitteR")

setup_twitter_oauth("xxx", "xxx", "xxx","xxx")

Twitter search

searchTwitter("#Hillary2016")

Streaming API

streamR::filterStream("tweets.json", track = c(“Trump", “Clinton"), timeout = 120, oauth = my_oauth)

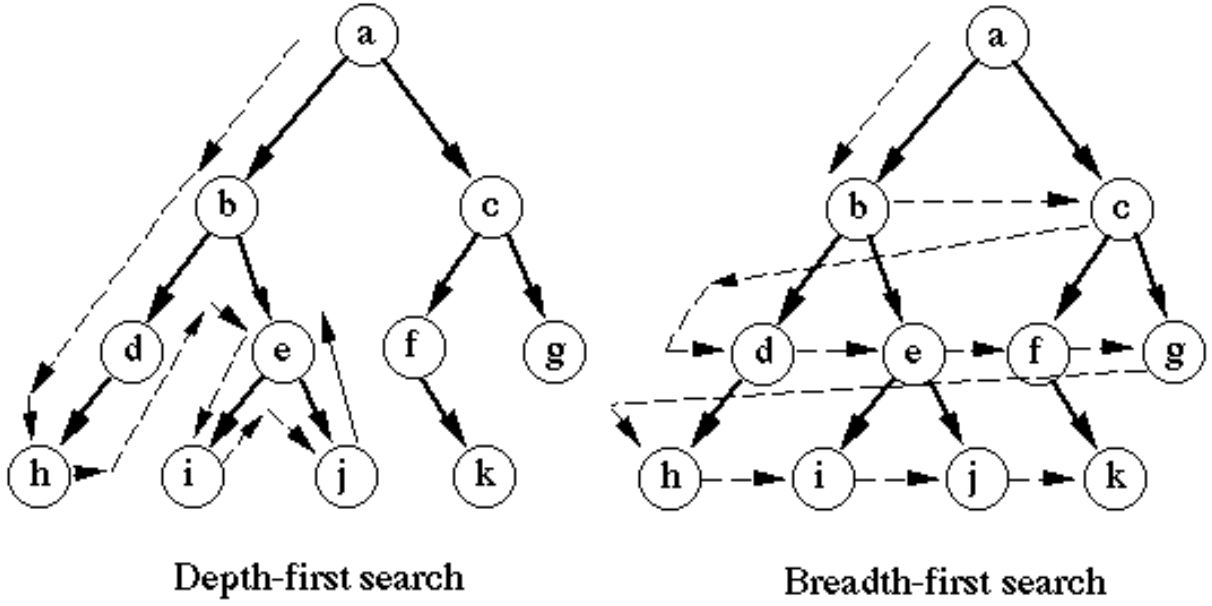
Tweets.df = parseTweets("tweets.json", simplify = TRUE)

httr package for Twitter OAuth

Parallel the process with multiple tokens

Crawling Rules

- URL pattern
- Next Page
- Random sampling
- Search
- Breadth first/depth first
- …



SOURCE

Twitter

1. User Profiles

TWEETS	FOLLOWING	FOLLOWERS	LIKES	MOMENTS
717	103	76	41	0

2. User Timelines (tweets)
3. Following (User-User) Networks

From Words to Numbers

1. Preprocess raw text:

“A mouse lives in my house.”

“There are mice living in my houses.”

“我覺得沒有比這更糟糕的課了！”

From Words to Numbers

1. Preprocess raw text: lowercase,

“a mouse lives in my house.”

“there are mice living in my houses.”

“我覺得沒有比這更糟糕的課了！”

```
cps = tm::VCorpus(VectorSource(docs))
cps = tm::tm_map(cps,tolower)
```

From Words to Numbers

1. Preprocess raw text: lowercase, remove stop words and punctuations,

"~~a~~ mouse lives ~~in~~ ~~my~~ house~~.~~"

"~~there~~ ~~are~~ mice living ~~in~~ ~~my~~ houses~~.~~"

"~~我覺得沒有比這更糟糕的課子！~~"

```
cps = tm_map(cps,removePunctuation)
cps = tm_map(cps,removeWords, stopwords("english"))
```

From Words to Numbers

1. Preprocess raw text: lowercase, remove stop words and punctuations, stem/lemmatization,

“mouse lives house”	“mouse live house”	“mous live hous”
“mice living houses”	“mouse live house”	“mice live hous”
“覺得糟糕課”	“覺得糟糕課”	“覺得糟糕課”
	lemmatization	stemming

```
cps = tm_map(cps,stemDocument) #stemming
coreNLP::getToken(annotateString(docs)) #lemmatization
```

From Words to Numbers

1. Preprocess raw text: lowercase, remove stop words and punctuations, stem/lemmatization, tokenize

“mouse/live/house”

“mouse/live/house”

“覺得/糟糕/課”

```
cps = tm_map(cps,scan_tokenizer)
cps = sapply(cps,jiebaR::segment,jiebaR::worker())
```

From Words to Numbers

1.Preprocess raw text: lowercase, remove stop words and punctuations, stem/lemmatization, tokenize, tag (part-of-speech)

“mouse[NN]/live[VBZ]/house[NN]”

“mouse[NNS]/live[VBG]/house[NNS]”

“覺得[V]/糟糕[JJ]/課[N]”

coreNLP::getToken(annotateString(docs)) #lemmatization+POS

cps = sapply(cps,jiebaR::segment,jiebaR::worker(“tag”))

From Words to Numbers

1.Preprocess raw text: lowercase, remove stop words and punctuations, stem/lemmatization, tokenize, tag (part-of-speech)

- Doc1: [mouse, live, house]
- Doc2: [mouse, live, house]
- Doc3: [覺得,糟糕,課]

```
cps = tm_map(cps,PlainTextDocument)
dtm = DocumentTermMatrix(cps,control=list(weighting=weightTfIdf))
```

2. Document-term matrix:

	mouse	live	house	覺得	糟糕	課
Doc1	1	1	1	0	0	0
Doc2	1	1	1	0	0	0
Doc3	0	0	0	1	1	1

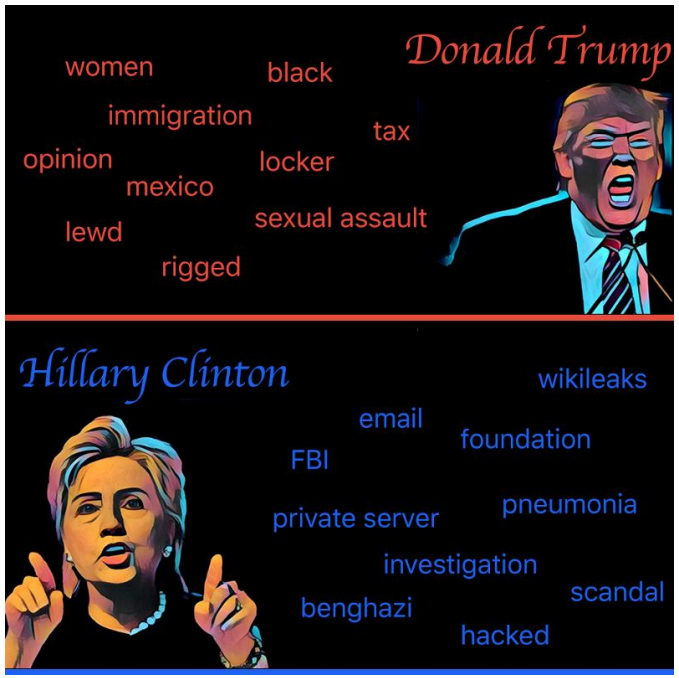
More on Common Words

(removing) Common words -> (selecting) Discriminant words

- RT @drhailiang The Chinese University of Hong Kong https://www.cuhk.edu.hk
- https, com, RT@, …
- post, reply, comment,…
- US, election, trump, Clinton, …

Selecting a reference dataset -> feature selection (MI, Chi)

- Category A vs. Category B (e.g., US Election vs. political texts)
- Log odds ratio for word *j* is given by
- $log\ odds\ ratio_j = \log\left(\frac{\pi_{A,j}}{1-\pi_{A,j}}\right) - \log\left(\frac{\pi_{B,j}}{1-\pi_{B,j}}\right)$
- $\pi_{A,j}$ is the proportion that category A documents contain word *j*.



Word Clouds in News

Thank You!

hailiang@cuhk.edu.hk