Welcome to PrairieML!

Wifi: AirUC-Guest

Materials of Interest: bit.ly/PrairieML0629

github.com/nbryans/TwitterProfileClassifier

Events we’ve done

- Machine Learning and Computer Vision Workshops
- Sessions on Data Analytics with SQL, Outlier Detection and Data Imputation
- An exploration into Neural Networks (and tensorflow)
- Kaggle Competititon Open Nights
- Pub night Socials

Events we’ll do

- Always open to session, workshop and social ideas
- Individual Contributors – This is the opportunity for experts to share what they’ve worked on and hobbyists to share what they’ve learned.
  - Good excuse to get your hands dirty, inspire others, and beef up your portfolio
- Researchers & Labs – Promote your work in a community forum.
- Businesses & Corporations – We are interested in hosting nights for you to discuss what you do, and you can use it to recruit if you’d like.

Email me at nbryans1@gmail.com, or message on the Meetup.com group

Also looking for venue ideas

Learning from Twitter Data

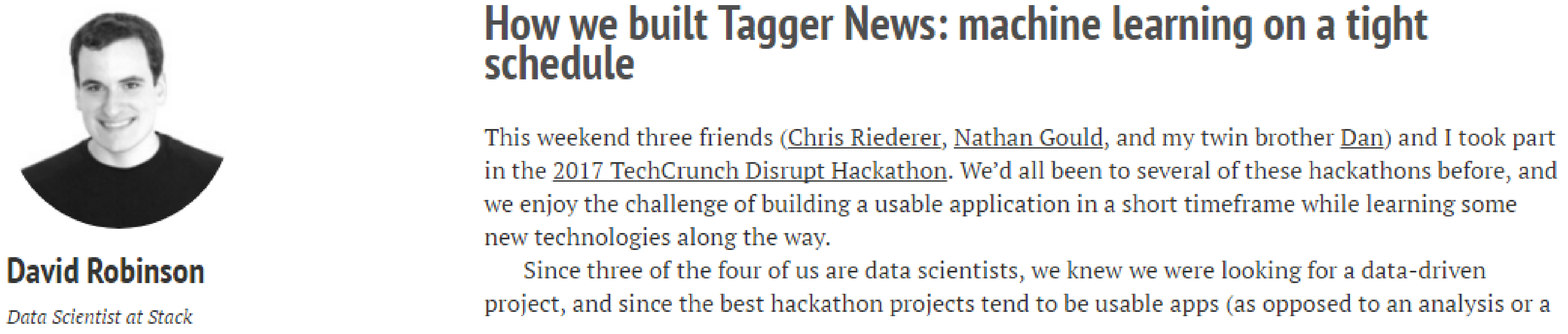
Nathan Bryans – PrairieML

nbryans1@gmail.com

Outline of Today's Talk

- Accessing Twitter Data
- Working with Twitter Data
- Building a Classifier (and what can be done better)

Inspiration: http://varianceexplained.org/programming/tagger-news



David Robinson

Data Scientist at Stack

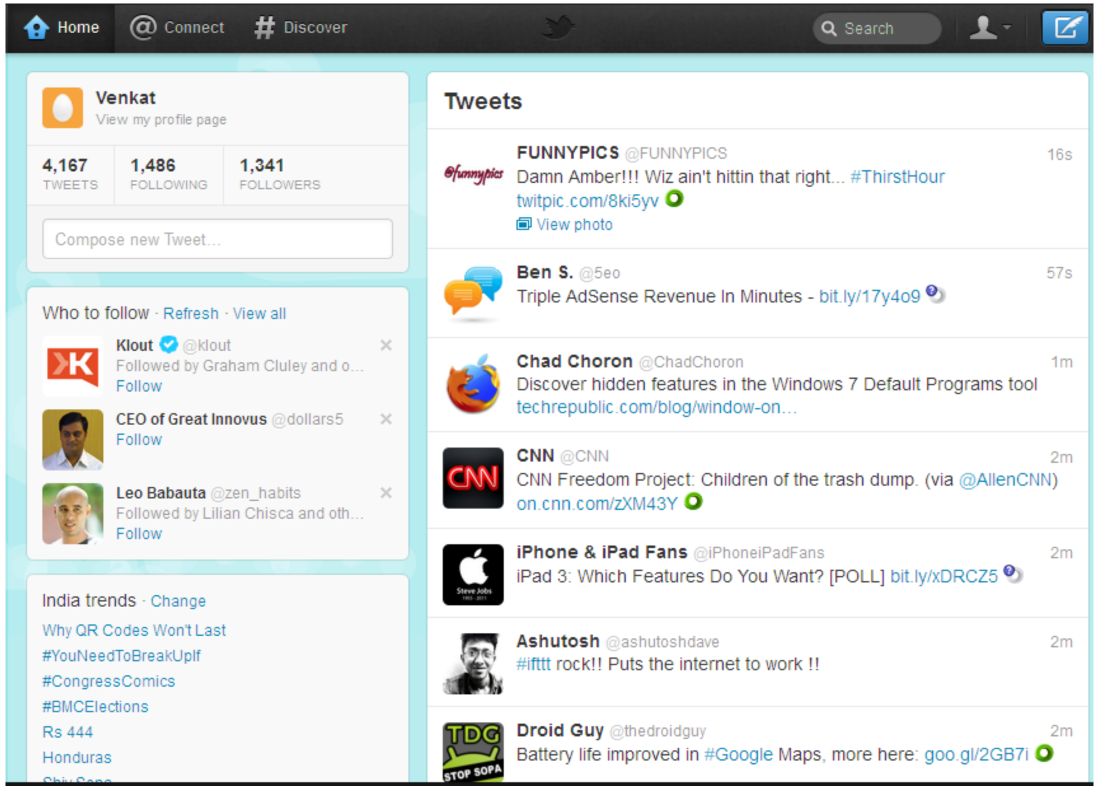
How we built Tagger News: machine learning on a tight schedule

This weekend three friends (Chris Riederer, Nathan Gould, and my twin brother Dan) and I took part in the 2017 TechCrunch Disrupt Hackathon. We'd all been to several of these hackathons before, and we enjoy the challenge of building a usable application in a short timeframe while learning some new technologies along the way.

Since three of the four of us are data scientists, we knew we were looking for a data-driven project, and since the best hackathon projects tend to be usable apps (as opposed to an analysis or a

Twitter

- 328 million monthly active users across the globe
- Tweets in 140 characters spanning all demographics and topics imaginable
- Data has been used in a wide variety of studies and learning algorithms



Venkat
View my profile page

4,167 TWEETS | 1,486 FOLLOWING | 1,341 FOLLOWERS

Compose new Tweet...

Who to follow · Refresh · View all

Klout @klout
Followed by Graham Cluley and o...
Follow

CEO of Great Innovus @dollars5
Follow

Leo Babauta @zen_habits
Followed by Lilian Chisca and oth...
Follow

India trends · Change
Why QR Codes Won't Last
#YouNeedToBreakUpIf
#CongressComics
#BMCElections
Rs 444
Honduras

Tweets

FUNNYPICS @FUNNYPICS 16s
Damn Amber!!! Wiz ain't hittin that right... #ThirstHour
twitpic.com/8ki5yv
View photo

Ben S. @5eo 57s
Triple AdSense Revenue In Minutes - bit.ly/17y4o9

Chad Choron @ChadChoron 1m
Discover hidden features in the Windows 7 Default Programs tool
techrepublic.com/blog/window-on...

CNN @CNN 2m
CNN Freedom Project: Children of the trash dump. (via @AllenCNN)
on.cnn.com/zXM43Y

iPhone & iPad Fans @iPhoneiPadFans 2m
iPad 3: Which Features Do You Want? [POLL] bit.ly/xDRCZ5

Ashutosh @ashutoshdave 2m
#ifttt rock!! Puts the internet to work !!

Droid Guy @thedroidguy 2m
Battery life improved in #Google Maps, more here: goo.gl/2GB7i

DATA SOURCE: TWITTER'S QUARTERLY PRESS RELEASES

A Model for Mining Public Health Topics from Twitter

Michael J. Paul and Mark Dredze;

‘[P]resent the Ailment Topic Aspect Model (ATAM), a new topic model for Twitter that associates symptoms, treatments and general words with diseases (ailments).

… ATAM isolates more coherent ailments, such as influenza, infections, obesity, as compared to standard topic models. …'

Twitter mood predicts the stock market

Johan Bollen, Huina Mao, and Xiaojun Zeng; Journal of Computational Science, 2011.

‘[I]nvestigate whether measurements of collective mood states derived from large-scale Twitter feeds are correlated to the values of the Dow Jones Industrial Average over time.'

OpinionFinder – measures positive vs. Negative mood

Google-Profile of Mood States – measures mood in terms of 6 dimensions (calm, alert, sure, vital, kind, and happy)

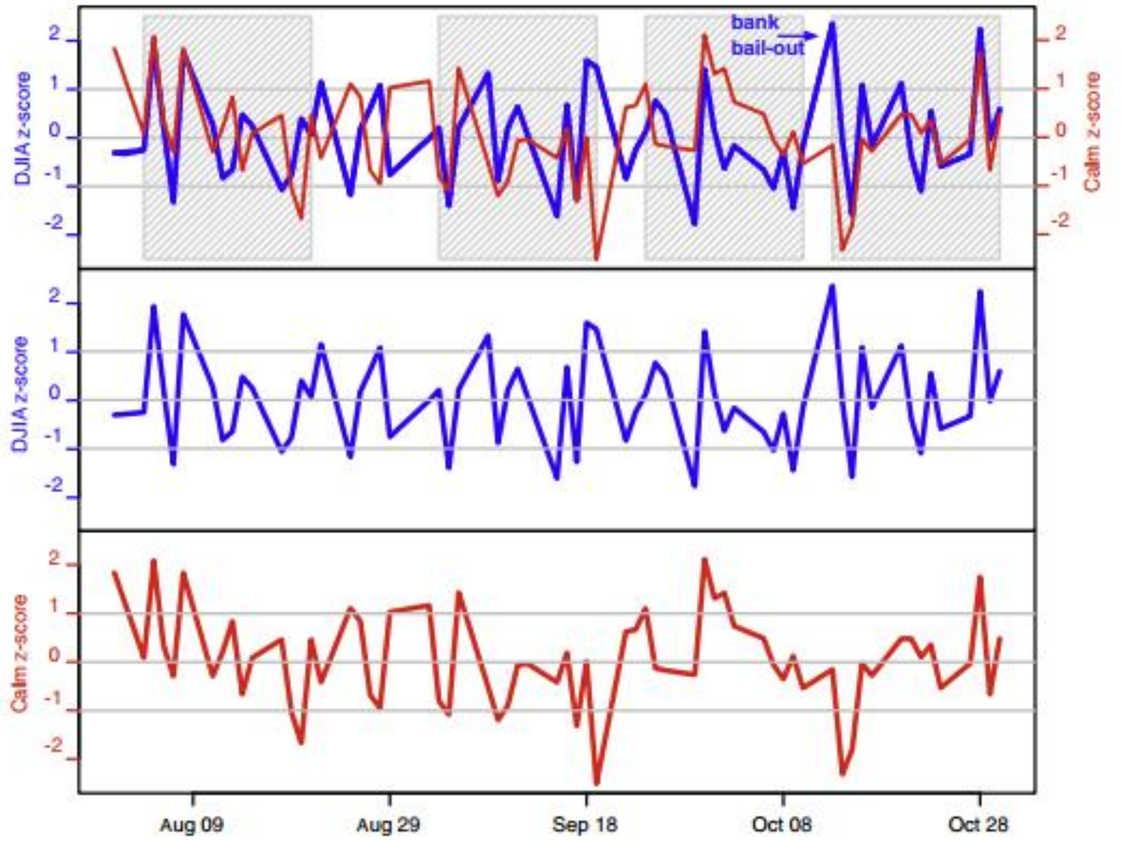


Fig. 3. A panel of three graphs. The top graph shows the overlap of the day-to-day difference of DJIA values (blue : $\mathbb{Z}_{D_t}$) with the GPOMS' Calm time series (red : $\mathbb{Z}_{X_t}$) that has been lagged by 3 days. Where the two graphs overlap the Calm time series predict changes in the DJIA closing values that occur 3 days later. Areas of significant congruence are marked by gray areas. The middle and bottom graphs show the separate DJIA and GPOMS' Calm time series. (For interpretation of the references to color in text, the reader is referred to the web version of the article.)

Ways to Get (Mine) Twitter Data

- Web Crawling
    - What you can see/click in the browser
- REST APIs
    - Programmatic access to read and write Twitter data
- Streaming APIs
    - Continuously deliver new responses to REST API queries over a long-lived HTTP connection
- Gnip/DataSift (Twitter Firehose)
    - Enterprise quality streaming data (at enterprise level pricing)

Twitter Data by Page Crawling

- Downloading page source (whether this be profile page, friend page, search results) to scrape data and possibly follow links.

- Conceivably faster than APIs (which have limits), but messy and requires complex wrappers.

- *May* be against Twitter's terms of use.

Twitter via REST APIs - Registering

Apps querying twitter data through the REST APIs require a API Key, API Secret, Access Token and Access Token Secret

There is User Authentication and Application-only Authentication

Instructions:

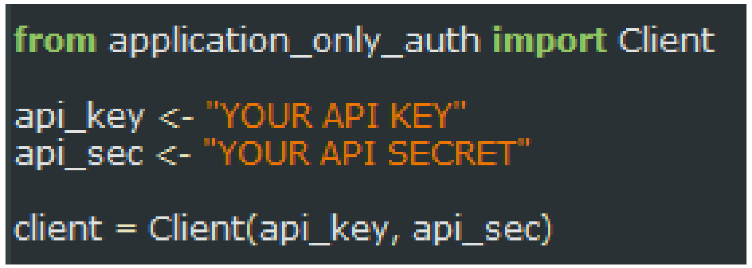
- Create a Twitter account
- Create a new App at apps.twitter.com, and get API Keys

Twitter via REST APIs - Connecting

The REST APIs identify Twitter applications and users using OAuth, and give responses in JSON.

Python

twitter-application-only-auth



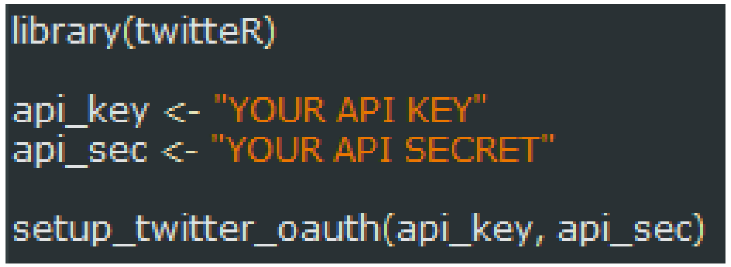
```
from application_only_auth import Client

api_key <- "YOUR API KEY"
api_sec <- "YOUR API SECRET"

client = Client(api_key, api_sec)
```

R

twitteR



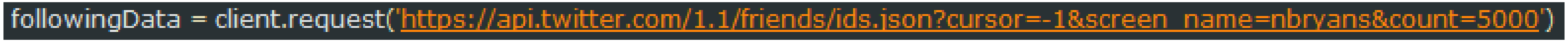
```
library(twitteR)

api_key <- "YOUR API KEY"
api_sec <- "YOUR API SECRET"

setup_twitter_oauth(api_key, api_sec)
```

Twitter REST APIs - Downloading

- Once authenticated, queries are made as calls to https://api.twitter.com/1.1 , and are thus language agnostic.
  - Many libraries exist to abstract these calls

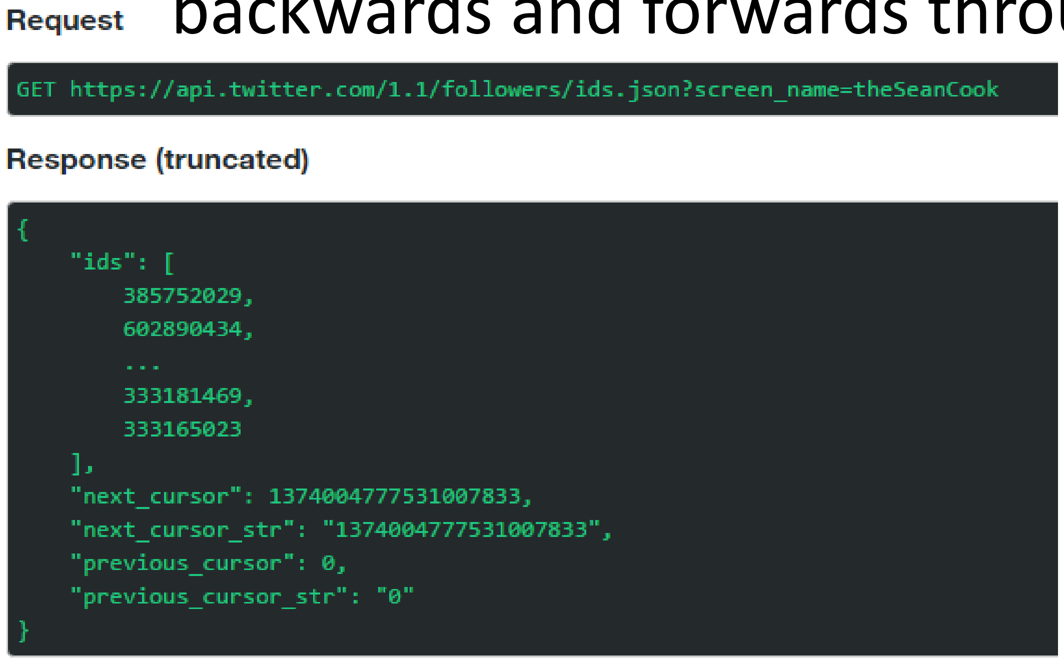


```
followingData = client.request('https://api.twitter.com/1.1/friends/ids.json?cursor=-1&screen_name=nbryans&count=5000')
```

- The types of data that can be requested: https://dev.twitter.com/rest/reference

Twitter REST APIs - Cursors

- The twitter REST API utilizes a technique called 'cursoring' to paginate large result sets.
- Cursoring separates results into pages (the size of which are defined by the 'count' request parameter) and provides a means to move backwards and forwards through these pages.

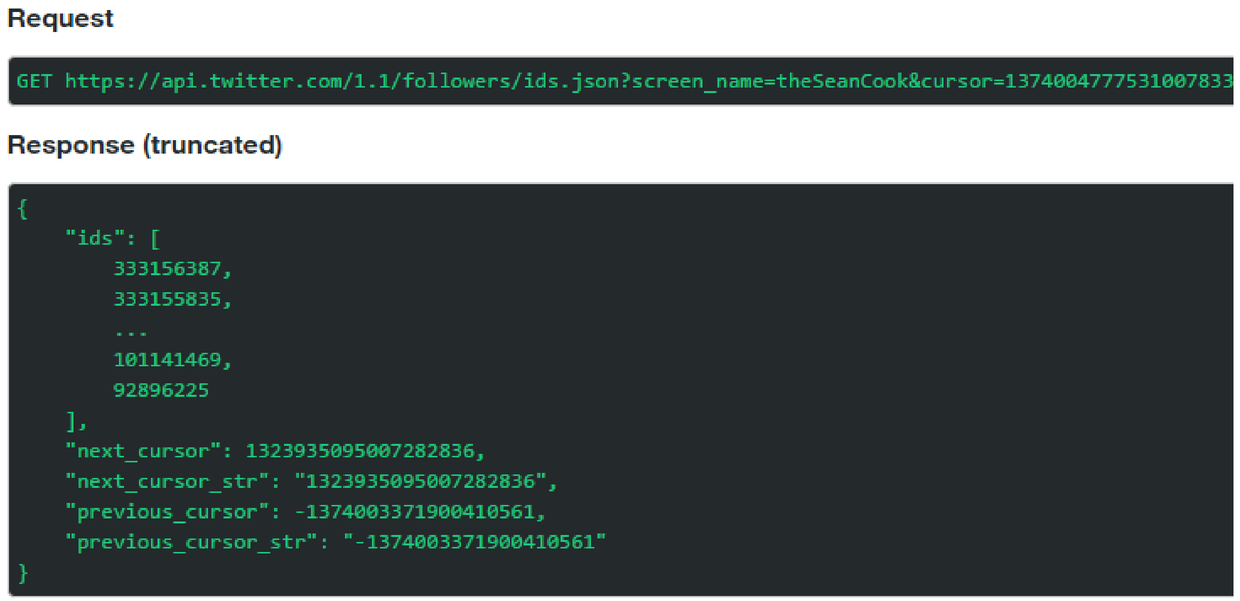


Request

```
GET https://api.twitter.com/1.1/followers/ids.json?screen_name=theSeanCook
```

Response (truncated)

```
{
    "ids": [
        385752029,
        602890434,
        ...
        333181469,
        333165023
    ],
    "next_cursor": 1374004777531007833,
    "next_cursor_str": "1374004777531007833",
    "previous_cursor": 0,
    "previous_cursor_str": "0"
}
```



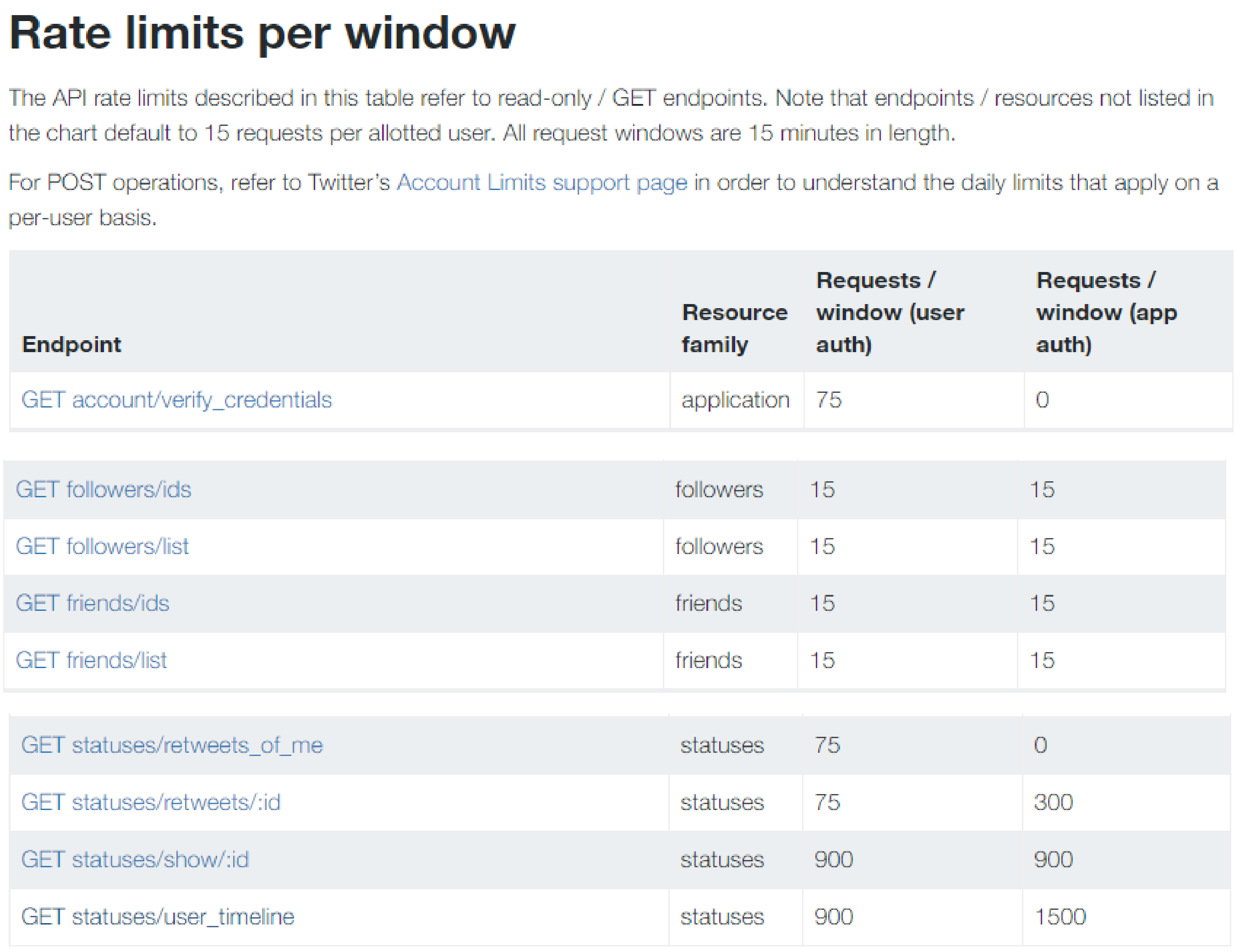
Request

```
GET https://api.twitter.com/1.1/followers/ids.json?screen_name=theSeanCook&cursor=1374004777531007833
```

Response (truncated)

```
{
    "ids": [
        333156387,
        333155835,
        ...
        101141469,
        92896225
    ],
    "next_cursor": 1323935095007282836,
    "next_cursor_str": "1323935095007282836",
    "previous_cursor": -1374003371900410561,
    "previous_cursor_str": "-1374003371900410561"
}
```

Twitter REST APIs - Limits



Rate limits per window

The API rate limits described in this table refer to read-only / GET endpoints. Note that endpoints / resources not listed in the chart default to 15 requests per allotted user. All request windows are 15 minutes in length.

For POST operations, refer to Twitter's Account Limits support page in order to understand the daily limits that apply on a per-user basis.

Endpoint	Resource family	Requests / window (user auth)	Requests / window (app auth)
GET account/verify_credentials	application	75	0
GET followers/ids	followers	15	15
GET followers/list	followers	15	15
GET friends/ids	friends	15	15
GET friends/list	friends	15	15
GET statuses/retweets_of_me	statuses	75	0
GET statuses/retweets/:id	statuses	75	300
GET statuses/show/:id	statuses	900	900
GET statuses/user_timeline	statuses	900	1500

Twitter Streaming APIs

- Low latency access to Twitter’s global stream of tweet data.
- Streaming client is pushed messages (without the need to poll) via a persistent HTTP connection.
- Can specify topics to receive data on.
- No guarantee you’ll get everything you want (need Firehose)

Working with Twitter Data

Saving the Data

- The API calls are slow (limited responses per window) and bandwidth intensive.
  - Not the sort of data we want to hold exclusively in memory until needed.

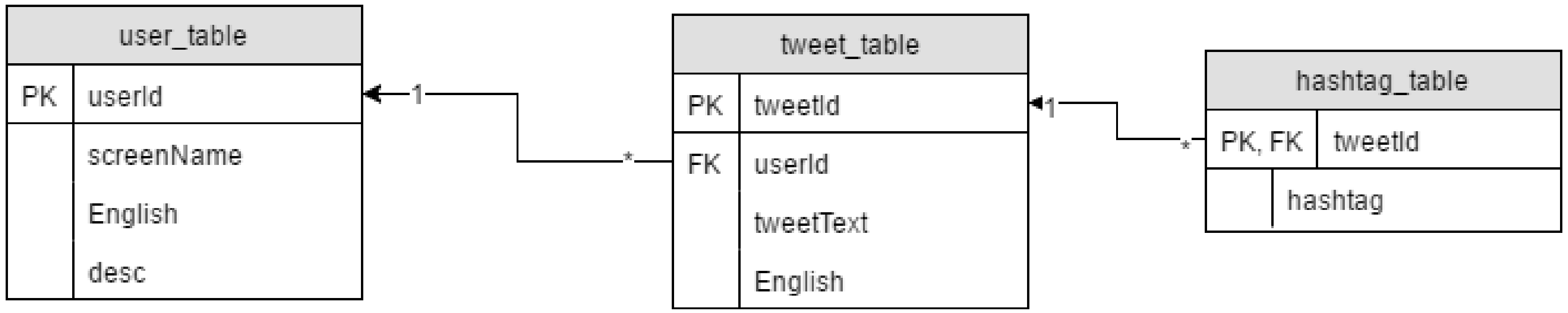
Options:

Save raw JSON to file

Databases (SQL or other)

My SQL Database

- SQLite3 – serverless with a single DB file

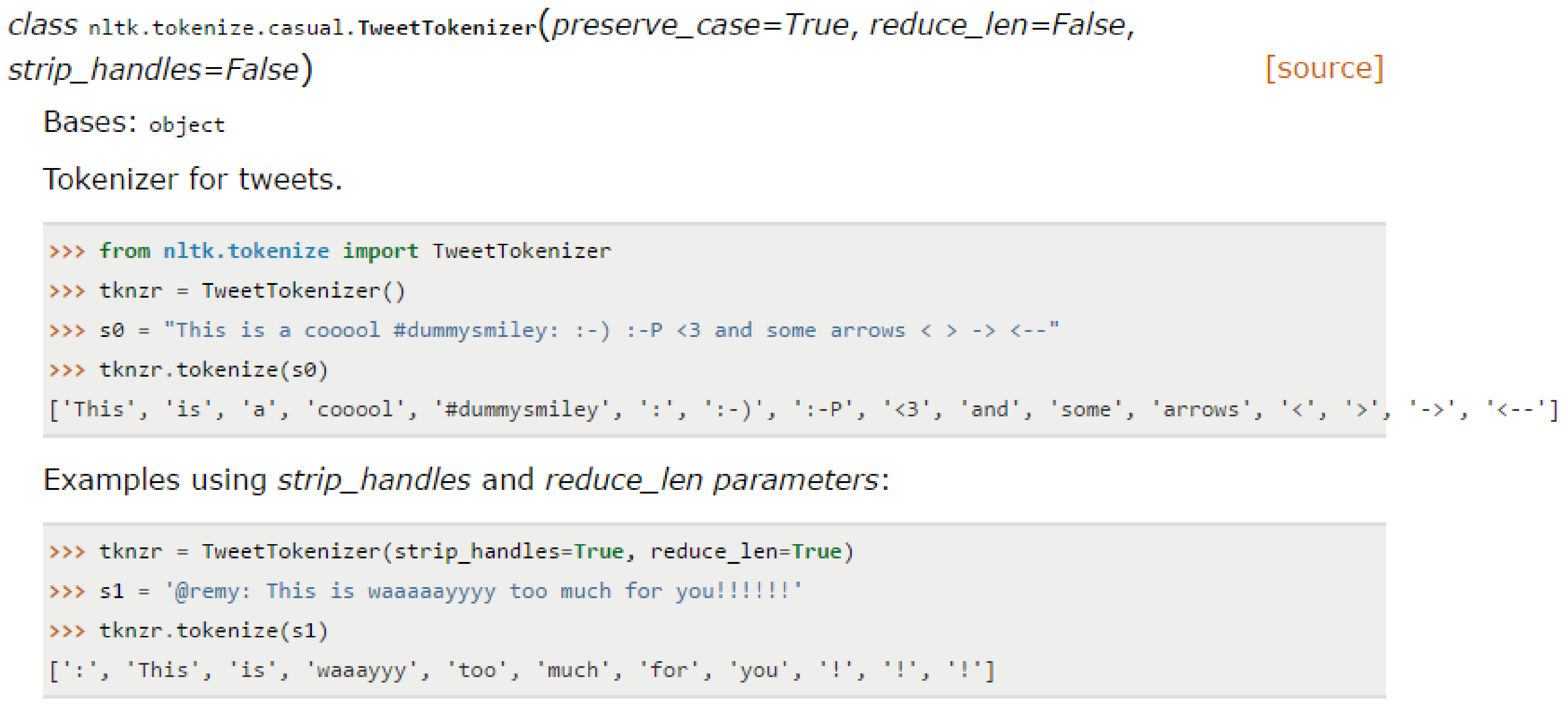


Twitter Specific Considerations

- Unicode (UTF-8)
  - Emojis
- Hashtags
- URLs and handles (@nbryans)
- Impressions & Retweets

nltk TweetTokenizer

- http://www.nltk.org/api/nltk.tokenize.html
- A special, twitter-aware tokenizer



class `nltk.tokenize.casual.TweetTokenizer`(*preserve_case=True*, *reduce_len=False*, *strip_handles=False*) [source]

Bases: `object`

Tokenizer for tweets.

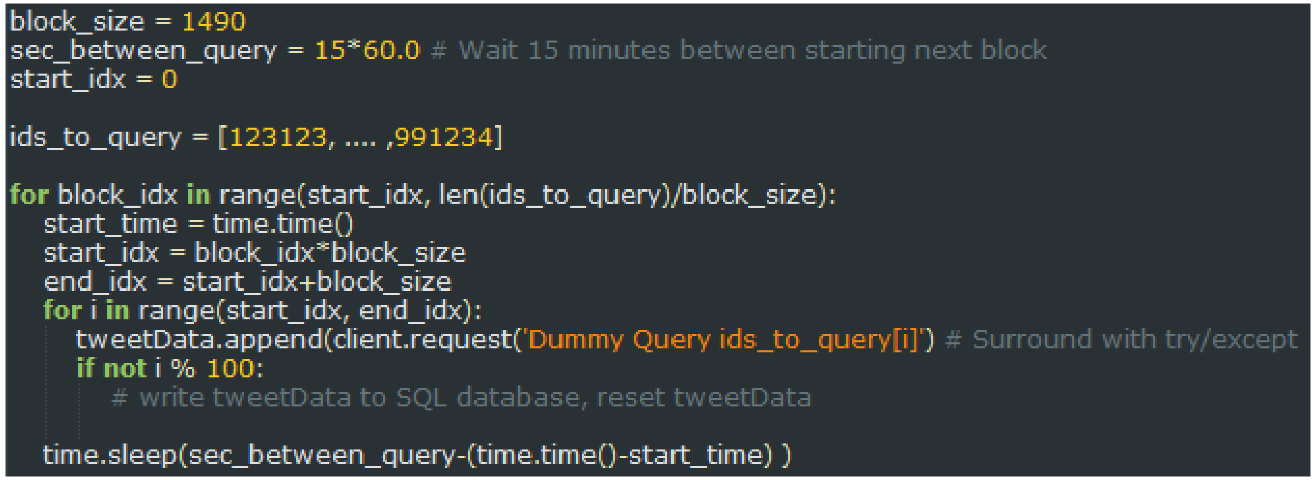
```
>>> from nltk.tokenize import TweetTokenizer
>>> tknzr = TweetTokenizer()
>>> s0 = "This is a cooool #dummysmiley: :-) :-P <3 and some arrows < > -> <--"
>>> tknzr.tokenize(s0)
['This', 'is', 'a', 'cooool', '#dummysmiley', ':', ':-)', ':-P', '<3', 'and', 'some', 'arrows', '<', '>', '->', '<--']
```

Examples using *strip_handles* and *reduce_len parameters*:

```
>>> tknzr = TweetTokenizer(strip_handles=True, reduce_len=True)
>>> s1 = '@remy: This is waaaaayyyy too much for you!!!!!!'
>>> tknzr.tokenize(s1)
[':', 'This', 'is', 'waaayyy', 'too', 'much', 'for', 'you', '!', '!', '!']
```

Delaying Queries

- The Twitter API limits work in windows of 15 minutes
- Can keep track of time as you go along, or make calls to 'application/rate_limit_status' every 5 seconds to see if reset

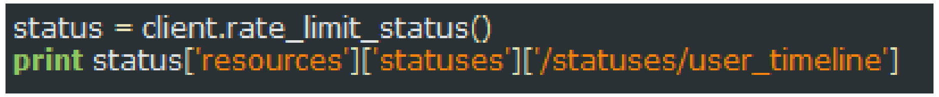


```
block_size = 1490
sec_between_query = 15*60.0 # Wait 15 minutes between starting next block
start_idx = 0

ids_to_query = [123123, .... ,991234]

for block_idx in range(start_idx, len(ids_to_query)/block_size):
    start_time = time.time()
    start_idx = block_idx*block_size
    end_idx = start_idx+block_size
    for i in range(start_idx, end_idx):
        tweetData.append(client.request('Dummy Query ids_to_query[i]') # Surround with try/except
        if not i % 100:
            # write tweetData to SQL database, reset tweetData

    time.sleep(sec_between_query-(time.time()-start_time) )
```

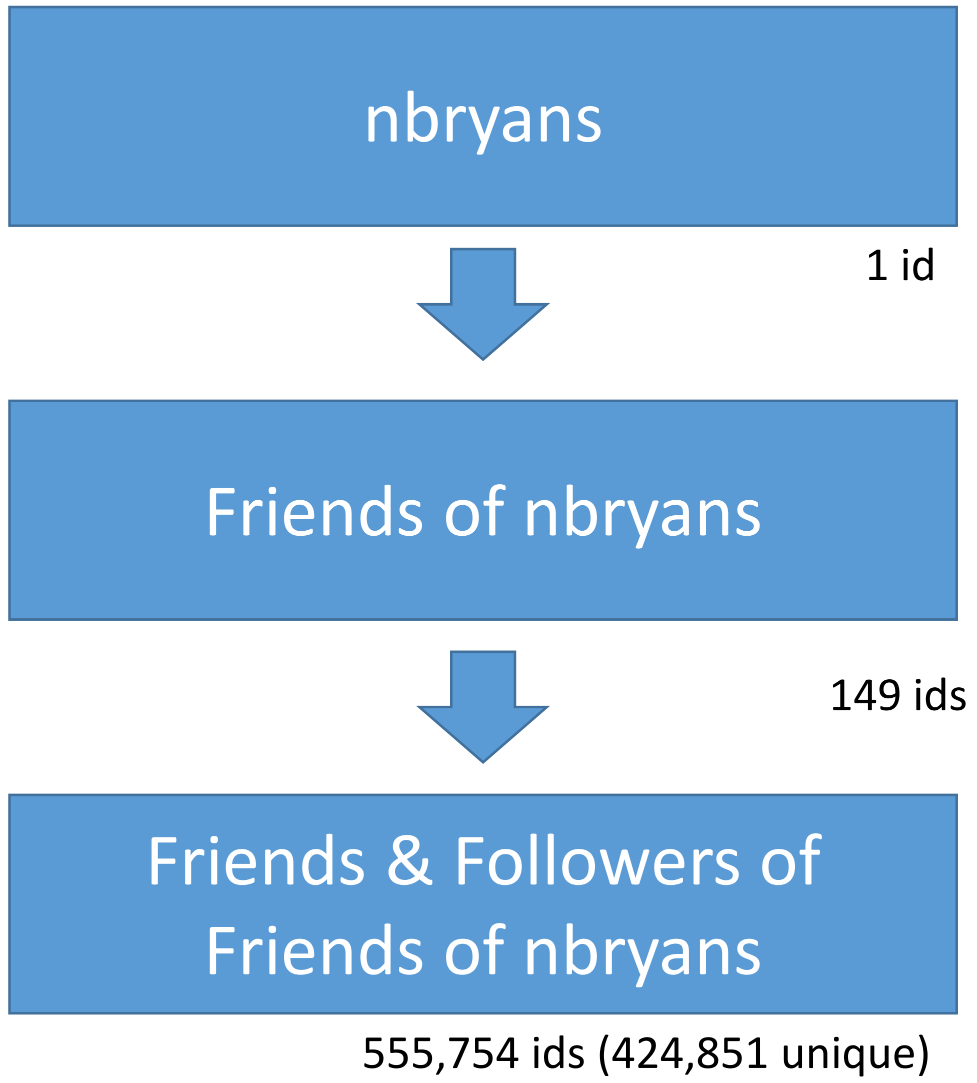


```
status = client.rate_limit_status()
print status['resources']['statuses']['/statuses/user_timeline']
```

A Classifier from Twitter Data

Identifying what subjects a user tweets about from the content of their tweets

Data Collection

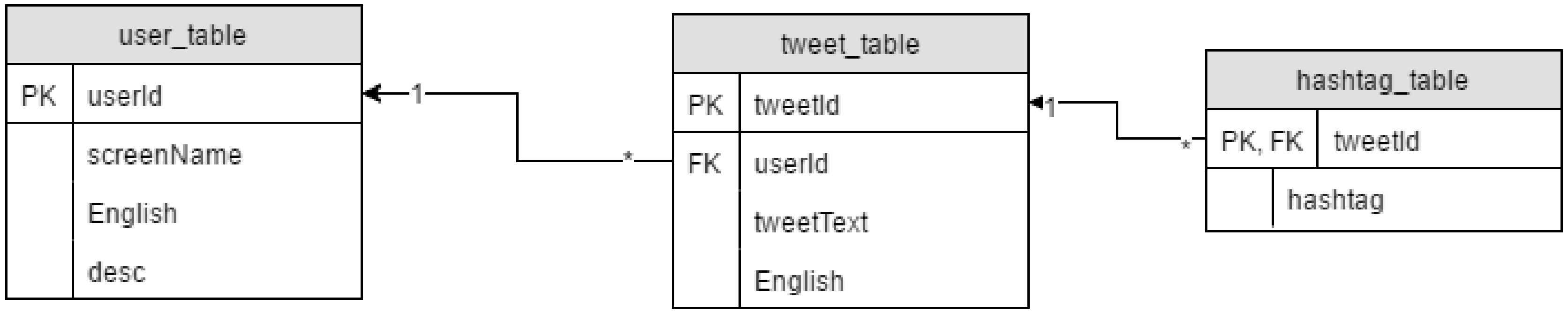

nbryans
1 id
Friends of nbryans
149 ids
Friends & Followers of Friends of nbryans
555,754 ids (424,851 unique)

- Max 5000 friends/followers returned at a time
- 15 ids per window

- To go one level deeper would take ~300 days

Data Collection

- API calls to download ‘statuses’ (tweets) return a LOT of data (including everything I need to fill the below table)
- Can request up to 3200 tweets per user (I requested only 100 tweets/user to speed up the calls and save bandwidth)
- Can query 1500 users every 15 minute window

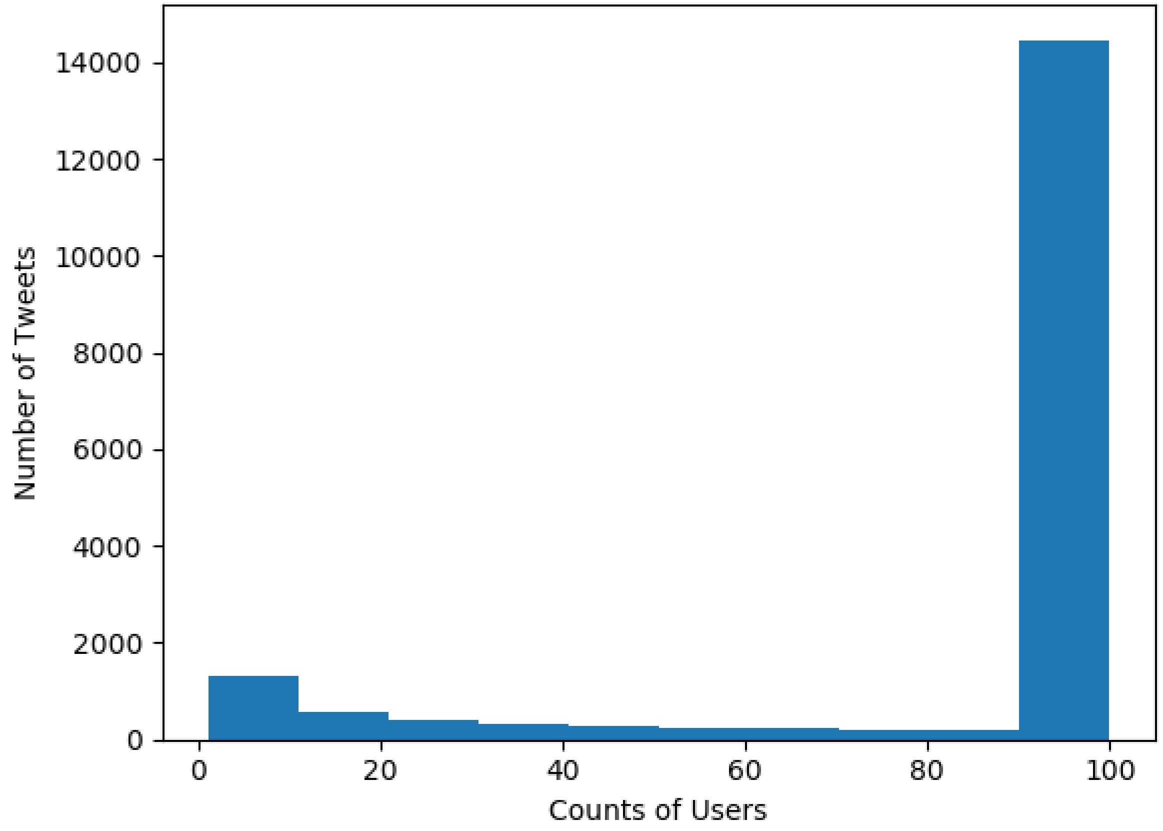


I have 1,550,153 tweets for 18,185 users.

SQL Database is 325MB

Data Collection

- 1,550,153 tweets for 18,185 users (fewer than expected)
- Average description is 10.7 words
- Average tweets per user is 85

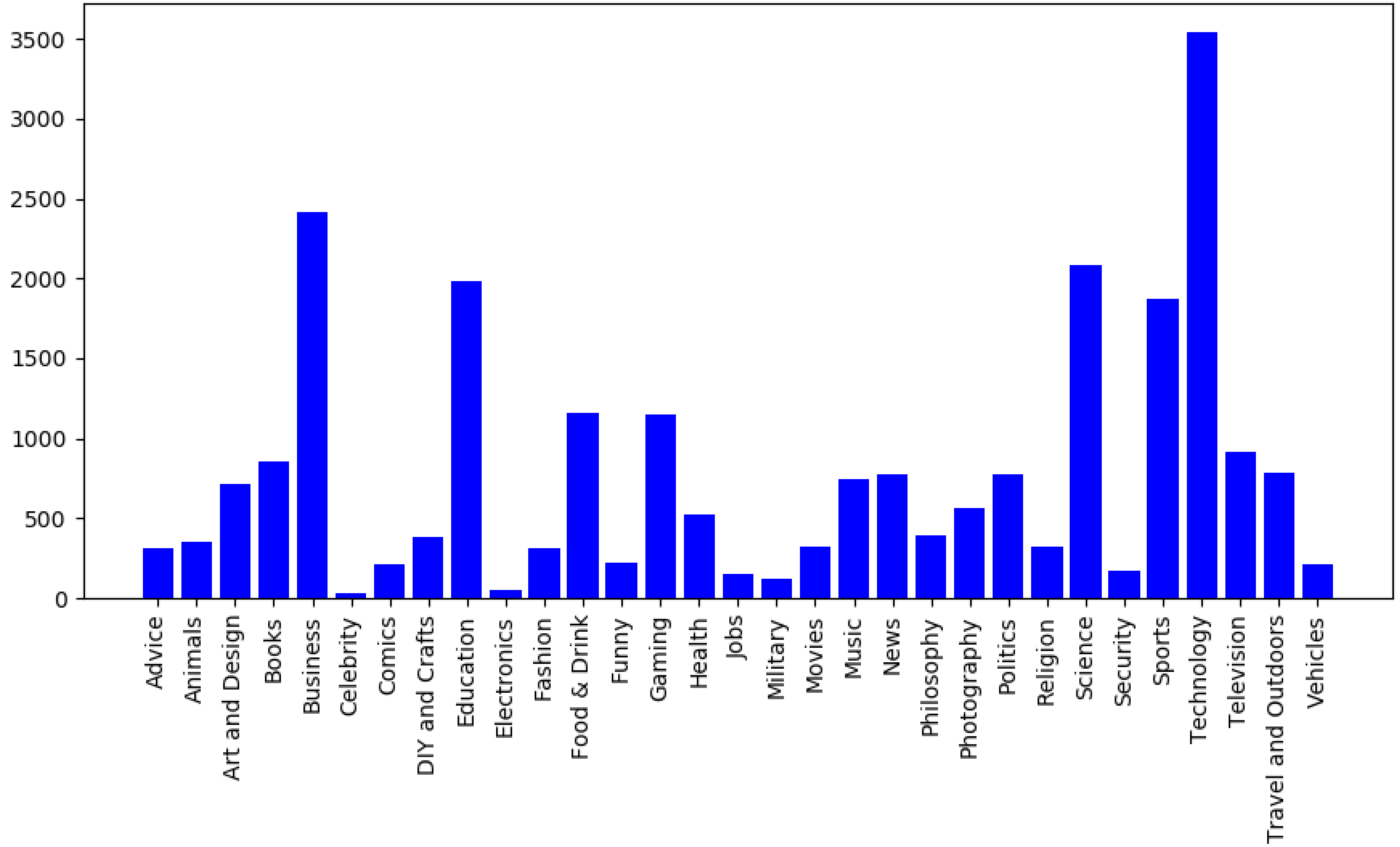


Building a Training/Testing Set

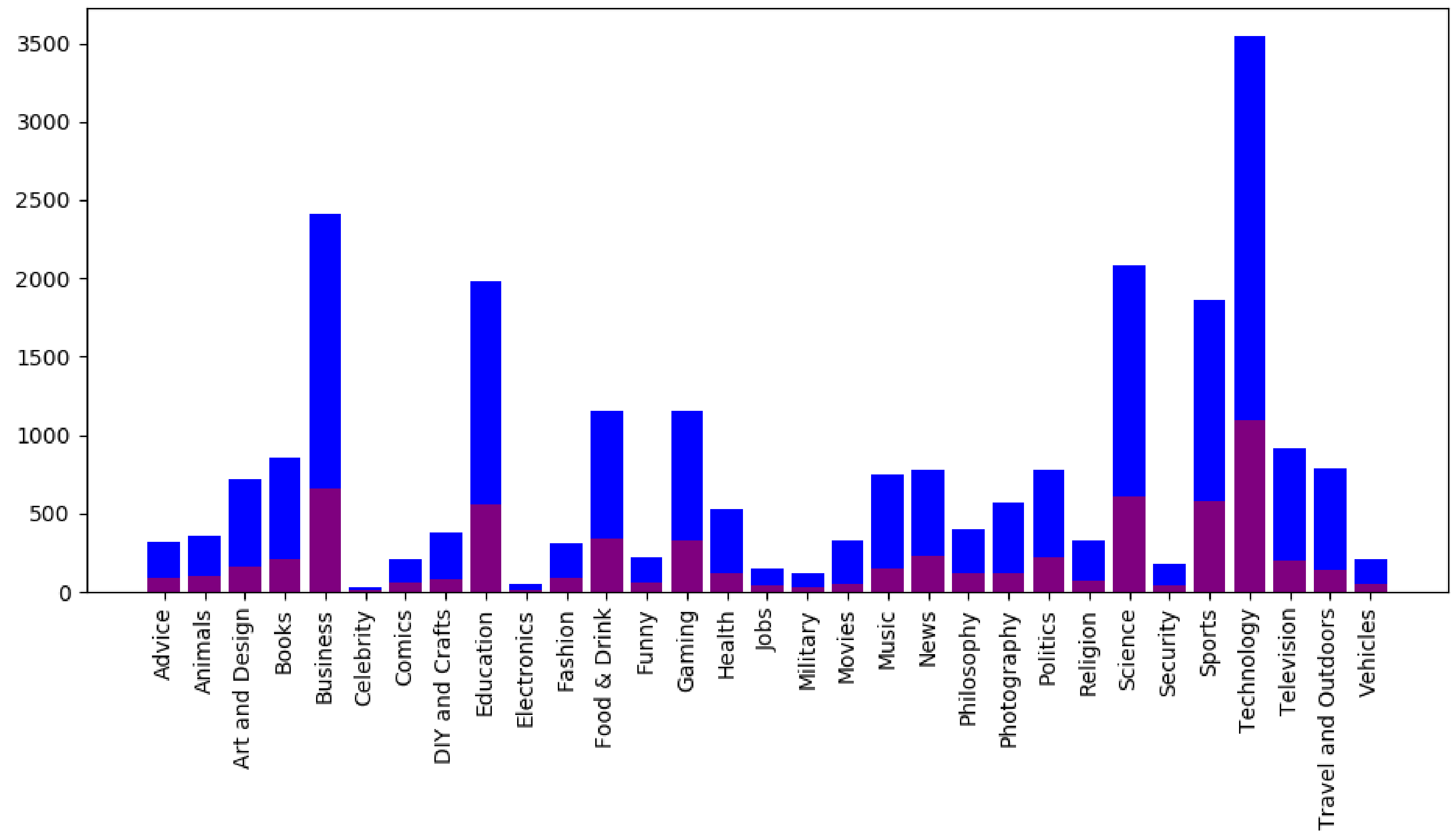
Advice-advice-help-wisdom**Animals**-animal-animals-mammal-mammals-reptile-reptiles-bird-birds-fish-nature**Art and Design**-artist-design-designing-painting-paint-sculpting-sculpt-drawing-draw**Books**-books-book-novel-novella-short-story-poem-poetry-sonnet-scifi-fantasy-fiction-nonfiction-romance-western-biography**Business**-business-company-finance-businesses-ceo-cfo-cto-executive-manager-money-capital-capitalist-venture-entrepreneur-invest-investor**Celebrity**-celebrity-celebrities-gossip-listers-paparazzi-famous**Comics**-comics-comic-strips-cartoon-batman-superman-wonder-xmen-fantastic-spiderman-hulk-ironman-joker-villian-hero**DIY and Crafts**-DIY-craft-knit-knitting-quilting-quilt-handy-construction-construct-build-builder-glue-tinkerer**Education**-professor-education-adjunct-teach-teacher-school-university-college-student**Electronics**-electronic-electronics-electricity-arduinoFashion-fashion-dress-fashionable-wear-clothes-clothing-shirt-shirts-pant-pants-shoes-style**Food & Drink**-food-foods-taste-taster-tasty-dine-dines-foodie-cook-chef-cooks-kitchen-drink-wine-winery-drinks-beer-chocolate-treat**Funny**-funny-joke-jokes-comedian-comedians-humor-humour-humors-humours-satire**Gaming**-gaming-gamer-game-video-play-xbox-playstation-wii-nintendo-sony-mario-pc-steam-console-controller**Health**-health-wellness-healthy-medicine-self-care-holistic**Jobs**-job-career-jobs-careers-resume**Military**-military-army-navy-airforce-marine-marines-conflict-combat-force-forces**Movies**-movie-movies-film-blockbuster**Music**-music-musician-musicians-instrument-instrumental-singer-singing-melody-album**News**-news-current-event-events-breaking**Philosophy**-philosophy-think-thinker-thinking-critical-reason-logic-contemplate**Photography**-photography-photo-camera-film-shooting**Politics**-politic-political-politics-democrat-democratic-democracy-republican-republic-GOP-dems-liberal-conservative-president-NDP-libs-senate-congress-governor-senator-laws-govern-government**Religion**-gospel-religion-religions-religious-church-christian-jesus-cross-cathedral-catholic-catholicism-protestant-protestantism-evangelical-evangelicalism-baptist-baptism-othodox-anglican-muslim-islam-sunni-shia-jewish-judaism-jews-hindu-hinduism-taoism-atheist-atheism-agnostic-pagan-paganism-wicca**Science**-chemisty-biochemistry-physics-biophysics-biology-science-bioinformatics-experiment-experiments-evolution-laboratory-medicine-geology-engineering-math-mathematics-research-researcher-genomic-spacex-nasa-space**Security**-security-worm-ransomware-attack-virus-patch-infosec**Sports**-sports-sport-bicycle-bicyclist-football-nfl-cfl-baseball-mlb-basketball-nba-hockey-nhl-chl-rugby-cricket-soccer-tennis-badminton-volleyball-ball-curling-skiing-boarding-swimming-polo-gymnastics-boxing-wrestling-compete**Technology**-gadget-device-mobile-techno-technology-programming-program-software-develop-developer-microsoft-apple-google-facebook-amazon-startup-startup-database-linux-windows-ios-machine-learning**Television**-television-tv-web-network-show-season-cable**Travel and Outdoors**-travel-adventure-trip-outdoor-outdoors-camp-hike-hiker-climb-climber-trek-trekker**Vehicles**-vehicle-vehicles-plane-planes-airplane-airplanes-boat-boats-speed-vintage-transport-transportation-auto

Building a Training/Testing Set

8257 users classified with this set of keywords



Absolute Categorization of Data

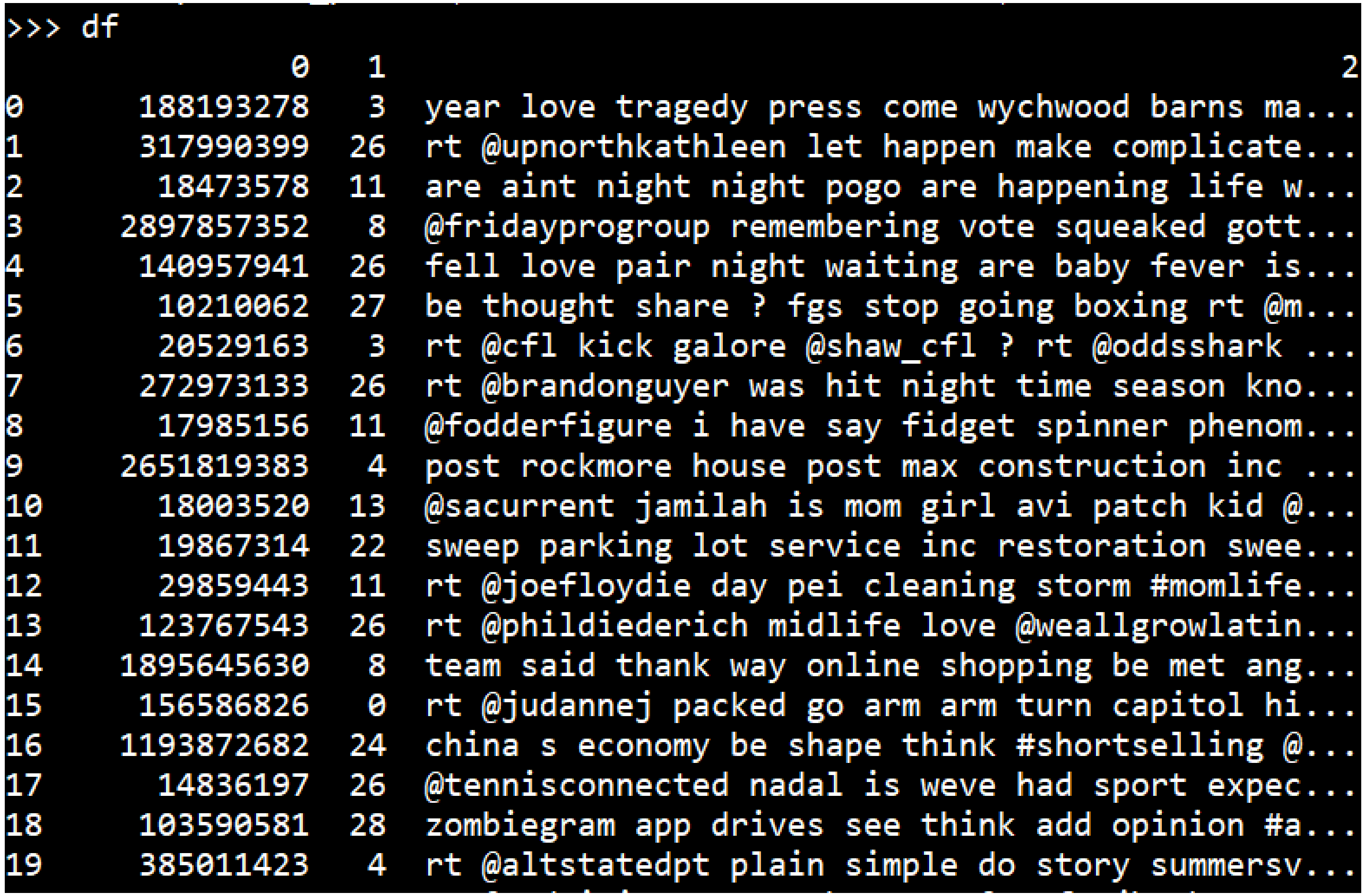


Data Cleaning

- Appended all tweets together (for each user) to form a ‘document’

```
tweet_str= ... # Appended tweets forming document
no_url =  re.sub(r'http\S+', '', tweet_str).lower()
tknzr = nltk.tokenize.TweetTokenizer
tokens = tknzr.tokenize(no_url)
pos_tags = nltk.pos_tag(tokens)
' '.join([i[0] for i in pos_tags if i[1]=='NN'
                                    or i[1].startswith('V')])
```

Result of Data Cleaning



```
>>> df
               0   1                                                 2
0      188193278   3  year love tragedy press come wychwood barns ma...
1      317990399  26  rt @upnorthkathleen let happen make complicate...
2       18473578  11  are aint night night pogo are happening life w...
3     2897857352   8  @fridayprogroup remembering vote squeaked gott...
4      140957941  26  fell love pair night waiting are baby fever is...
5       10210062  27  be thought share ? fgs stop going boxing rt @m...
6       20529163   3  rt @cfl kick galore @shaw_cfl ? rt @oddsshark ...
7      272973133  26  rt @brandonguyer was hit night time season kno...
8       17985156  11  @fodderfigure i have say fidget spinner phenom...
9     2651819383   4  post rockmore house post max construction inc ...
10      18003520  13  @sacurrent jamilah is mom girl avi patch kid @...
11      19867314  22  sweep parking lot service inc restoration swee...
12      29859443  11  rt @joefloydie day pei cleaning storm #momlife...
13     123767543  26  rt @phildiederich midlife love @weallgrowlatin...
14    1895645630   8  team said thank way online shopping be met ang...
15     156586826   0  rt @judannej packed go arm arm turn capitol hi...
16    1193872682  24  china s economy be shape think #shortselling @...
17      14836197  26  @tennisconnected nadal is weve had sport expec...
18     103590581  28  zombiegram app drives see think add opinion #a...
19     385011423   4  rt @altstatedpt plain simple do story summersv...
```

LDA w/ Random Forest Classifier

- David Robinson and co. used Latent Dirichlet Allocation as a means of dimensionality reduction.
  - This turned each document into a vector of ‘topics’ that were fit from the data.

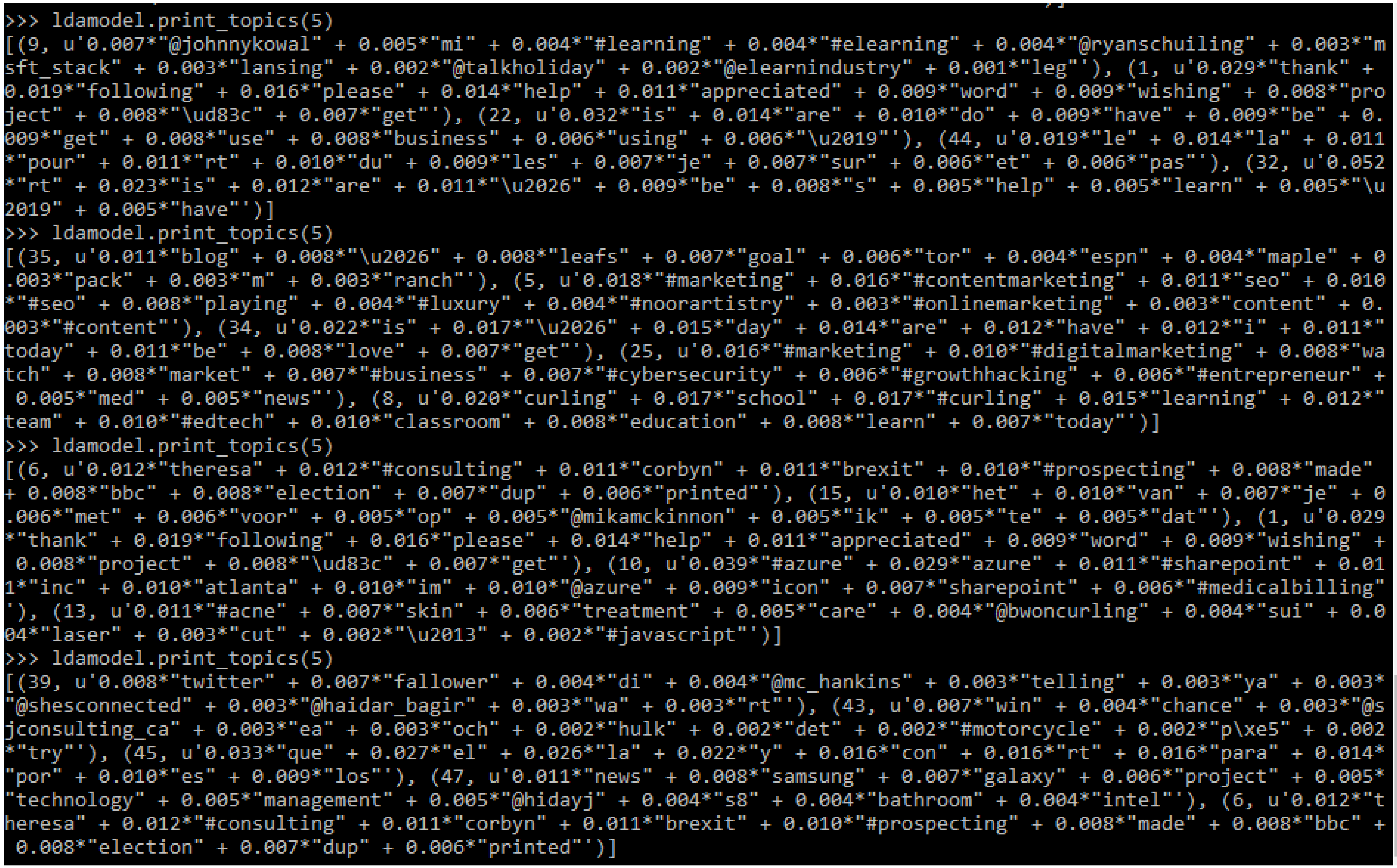
- Each document is viewed as a mixture of topics
- A topic has probabilities of generating various words
- Words without special relevance (such as ‘the’) have roughly equal probabilities between classes.
- Topics are not strongly defined, but is fit from labelled data.

https://en.wikipedia.org/wiki/Latent_Dirichlet_allocation

LDA w/ Random Forest Classifier

To build the classifier:

1. Split data in training (80%) and testing (20%) sets
2. Split tweet ‘documents’ into a corpus (b.o.w.)
3. Create an LdaModel by finding 50 topics in this data
4. Find the vectors for each document in the train (and test) data
5. Train a Random Forest Classifier on these vectors with the training data
6. Test the Random Forest Classifier on the test set using the vectors



```
>>> ldamodel.print_topics(5)
[(9, u'0.007*"@johnnykowal" + 0.005*"mi" + 0.004*"#learning" + 0.004*"#elearning" + 0.004*"@ryanschuiling" + 0.003*"m
sft_stack" + 0.003*"lansing" + 0.002*"@talkholiday" + 0.002*"@elearnindustry" + 0.001*"leg"'), (1, u'0.029*"thank" +
0.019*"following" + 0.016*"please" + 0.014*"help" + 0.011*"appreciated" + 0.009*"word" + 0.009*"wishing" + 0.008*"pro
ject" + 0.008*"\ud83c" + 0.007*"get"'), (22, u'0.032*"is" + 0.014*"are" + 0.010*"do" + 0.009*"have" + 0.009*"be" + 0.
009*"get" + 0.008*"use" + 0.008*"business" + 0.006*"using" + 0.006*"\u2019"'), (44, u'0.019*"le" + 0.014*"la" + 0.011
*"pour" + 0.011*"rt" + 0.010*"du" + 0.009*"les" + 0.007*"je" + 0.007*"sur" + 0.006*"et" + 0.006*"pas"'), (32, u'0.052
*"rt" + 0.023*"is" + 0.012*"are" + 0.011*"\u2026" + 0.009*"be" + 0.008*"s" + 0.005*"help" + 0.005*"learn" + 0.005*"\u
2019" + 0.005*"have"')]
>>> ldamodel.print_topics(5)
[(35, u'0.011*"blog" + 0.008*"\u2026" + 0.008*"leafs" + 0.007*"goal" + 0.006*"tor" + 0.004*"espn" + 0.004*"maple" + 0
.003*"pack" + 0.003*"m" + 0.003*"ranch"'), (5, u'0.018*"#marketing" + 0.016*"#contentmarketing" + 0.011*"seo" + 0.010
*"#seo" + 0.008*"playing" + 0.004*"#luxury" + 0.004*"#noorartistry" + 0.003*"#onlinemarketing" + 0.003*"content" + 0.
003*"#content"'), (34, u'0.022*"is" + 0.017*"\u2026" + 0.015*"day" + 0.014*"are" + 0.012*"have" + 0.012*"i" + 0.011*"
today" + 0.011*"be" + 0.008*"love" + 0.007*"get"'), (25, u'0.016*"#marketing" + 0.010*"#digitalmarketing" + 0.008*"wa
tch" + 0.008*"market" + 0.007*"#business" + 0.007*"#cybersecurity" + 0.006*"#growthhacking" + 0.006*"#entrepreneur" +
 0.005*"med" + 0.005*"news"'), (8, u'0.020*"curling" + 0.017*"school" + 0.017*"#curling" + 0.015*"learning" + 0.012*"
team" + 0.010*"#edtech" + 0.010*"classroom" + 0.008*"education" + 0.008*"learn" + 0.007*"today"')]
>>> ldamodel.print_topics(5)
[(6, u'0.012*"theresa" + 0.012*"#consulting" + 0.011*"corbyn" + 0.011*"brexit" + 0.010*"#prospecting" + 0.008*"made"
+ 0.008*"bbc" + 0.008*"election" + 0.007*"dup" + 0.006*"printed"'), (15, u'0.010*"het" + 0.010*"van" + 0.007*"je" + 0
.006*"met" + 0.006*"voor" + 0.005*"op" + 0.005*"@mikamckinnon" + 0.005*"ik" + 0.005*"te" + 0.005*"dat"'), (1, u'0.029
*"thank" + 0.019*"following" + 0.016*"please" + 0.014*"help" + 0.011*"appreciated" + 0.009*"word" + 0.009*"wishing" +
 0.008*"project" + 0.008*"\ud83c" + 0.007*"get"'), (10, u'0.039*"#azure" + 0.029*"azure" + 0.011*"#sharepoint" + 0.01
1*"inc" + 0.010*"atlanta" + 0.010*"im" + 0.010*"@azure" + 0.009*"icon" + 0.007*"sharepoint" + 0.006*"#medicalbilling"
'), (13, u'0.011*"#acne" + 0.007*"skin" + 0.006*"treatment" + 0.005*"care" + 0.004*"@bwoncurling" + 0.004*"sui" + 0.0
04*"laser" + 0.003*"cut" + 0.002*"\u2013" + 0.002*"#javascript"')]
>>> ldamodel.print_topics(5)
[(39, u'0.008*"twitter" + 0.007*"fallower" + 0.004*"di" + 0.004*"@mc_hankins" + 0.003*"telling" + 0.003*"ya" + 0.003*
"@shesconnected" + 0.003*"@haidar_bagir" + 0.003*"wa" + 0.003*"rt"'), (43, u'0.007*"win" + 0.004*"chance" + 0.003*"@s
jconsulting_ca" + 0.003*"ea" + 0.003*"och" + 0.002*"hulk" + 0.002*"det" + 0.002*"#motorcycle" + 0.002*"p\xe5" + 0.002
*"try"'), (45, u'0.033*"que" + 0.027*"el" + 0.026*"la" + 0.022*"y" + 0.016*"con" + 0.016*"rt" + 0.016*"para" + 0.014*
"por" + 0.010*"es" + 0.009*"los"'), (47, u'0.011*"news" + 0.008*"samsung" + 0.007*"galaxy" + 0.006*"project" + 0.005*
"technology" + 0.005*"management" + 0.005*"@hidayj" + 0.004*"s8" + 0.004*"bathroom" + 0.004*"intel"'), (6, u'0.012*"t
heresa" + 0.012*"#consulting" + 0.011*"corbyn" + 0.011*"brexit" + 0.010*"#prospecting" + 0.008*"made" + 0.008*"bbc" +
 0.008*"election" + 0.007*"dup" + 0.006*"printed"')]
```

Feature Vectors w/ Classifer

- Inspired by the scikit-learn Working with Text Data tutorial

1. Preprocess documents and generate a CountVector
2. Convert Counts to Frequencies, and downscales words appearing in more documents (they are less informative)
   tf-idf = Term Frequency times Inverse Document Frequency
3. Train a classifier (I tried a multinomial naïve bayes and a random forest) using this tf-idf data

Results

Classifier	Classified Correctly
LDA + Random Forest	Approx 30%
Tfidf + Multinomial Naïve Bayes	24 %
Tfidf + Random Forest (30 est.)	34.5 %
Tfidf + Random Forest (50 est.)	34.7 %
Tfidf + Random Forest (100 est.)	37.1 %

Limitations & Possible Improvements

- Better selection of topics and keywords
- More tweets per user
- More users that could be assigned a classification
- Limited words to nouns and verbs, is this ok?

- Concurrent downloads of twitter data using multiple ‘apps’
- Better support for multiple classification

- Would an SVM outperform the Random Forest Classifier?

Thank You!

Please send comments, suggestions and any interest in hosting a future session to: nbryans1@gmail.com (or write them on the Meetup.com page)