# Stat 222 Project 3: Twitter
# Getting started with Python and Twitter

## 1 Webservices

Many data sources are available on the Internet. You've probably used a web browser interface to search through some of this data and even download it to your computer. As you may have noticed, this manual process is labor intensive, error prone, and hard to document.

To provide programmatic and automatic interaction with these data stores, many websites serve this information through a documented application programmer interface (API).[1] These webservice APIs provide a mechanism to create new functionality on top of existing webcontent.

While there are several ways to implement webservices, Representational State Transfer (REST) has gained widespread popularity. REST is more of a style than a standard. A system designed in the REST style is called RESTful. RESTful systems typically use HTTP requests to query and post data using the standard HTTP verbs (GET, POST, PUT, DELETE, etc.). Often some form of authentication is necessary to communicate with a webservice. It is common to use OAuth for this purpose.

## 2 Data serialization

When you use a web browser to view a webpage, your browser handles the HTTP communication for you. You specify what you wish to view in the form of a URI such as `http://example.org/absolute/URI/with/absolute/path/to/resource.txt`. At this point your browser communicates with the webserver and requests the resource. The resource is typically provided to your web browser as an HTML document, which browsers are specifically designed to render.

Similarly, you will use HTTP to communicate with the Twitter webservice. However rather than passively consuming webpages, you will be interested in retrieving data in a form that is amenable to further processing.

Data serialization refers to the process of encoding data structures and objects in a format that can be used to store this information on disk or transmit it

[1] For several examples see the Appendix.

over the web. For example, in R you can use the Rdata format to save R objects to disk and reload them later. For webservices, JSON and XML are standard formats. To better understand this, let's briefly look at data serialization more generally.

## Python object

First let's create a Python object.

```
>>> mydict = {
...     "parents": [{"name": "mom",
...                  "number": "555-123-4567"},
...                 {"name": "dad",
...                  "number": "555-123-4567"}],
...     "colleagues": {"name": "advisor",
...                    "number": "555-123-4567"}
... }
```

And let's print the results:

```
>>> mydict
{'colleagues': {'name': 'advisor', 'number': '555-123-4567'}, 'parents': [{'name': 'mom', 'n
>>> import pprint
>>> pprint.pprint(mydict)
{'colleagues': {'name': 'advisor', 'number': '555-123-4567'},
 'parents': [{'name': 'mom', 'number': '555-123-4567'},
             {'name': 'dad', 'number': '555-123-4567'}]}
```

## XML

How does this object look if we convert it to XML?[2]

```
>>> from dict2xml import dict2xml
>>> print(dict2xml(mydict))
<colleagues>
  <name>advisor</name>
  <number>555-123-4567</number>
</colleagues>
<parents>
  <name>mom</name>
  <number>555-123-4567</number>
</parents>
<parents>
  <name>dad</name>
  <number>555-123-4567</number>
</parents>
```

[2]This functionality is not part of the standard library. And should not be used in practice.

## JSON

What if we convert it to JSON?

```
>>> import json
>>> print(json.dumps(mydict, indent=4, sort_keys=True))
{
    "colleagues": {
        "name": "advisor",
        "number": "555-123-4567"
    },
    "parents": [
        {
            "name": "mom",
            "number": "555-123-4567"
        },
        {
            "name": "dad",
            "number": "555-123-4567"
        }
    ]
}
```

## YAML

What if we convert it to YAML?

```
>>> import yaml
>>> print(yaml.dump(mydict))
colleagues: {name: advisor, number: 555-123-4567}
parents:
- {name: mom, number: 555-123-4567}
- {name: dad, number: 555-123-4567}
```

## Questions

Looking over the output of the above formats you should notice several things.

- Which of the formats uses the largest number of characters?
- Which uses the fewest?
- Which looks most like Python?

## Saving JSON and CSV

For this project you will be querying the Twitter webservice and will be getting responses in the JSON format. After you get your response, you will want to

save it to disk. I recommend that you use the JSON format as your main data storage format for this project.

```
>>> import json
>>> with open("data.json", "w") as outfile:
...     json.dump(mydict, outfile, indent=4, sort_keys=True)
...
```

If you decide that you would like to use R for part of your analysis or for creating figures, I recommend saving the information you want to work with in R as a CSV file. Your JSON file will have nested and non-homogeneous structure, which is not possible to directly store using CSV. So you will need to first decide what data you want to save as CSV and then transform the JSON data into the necessary form. Here is an example of how you might transform `mydict` above into a list of equal length tuples.

```
mydict["colleagues"] = [mydict["colleagues"]]
mylist = [(e["name"], e{"number"], k) for k, v in mydict.items() for e in v]
```

Before I can use list comprehension to form the list of tuples I ensure that the nested structure that I iterate over has equal depth in each substructure. Then I save the list of tuples as a CSV file.

```
import csv
with open("data.csv", "w") as outfile:
    csv_out = csv.writer(outfile)
    csv_out.writerow(["name", "number", "relation"])
    for row in mylist:
        csv_out.writerow(row)
```

# 3 Example: US Senate tweets

If we are interested in Tweets from members of the U.S. Senate,[3] we could start by getting a list of their Twitter accounts.[4] Then we could retrieve each Senator's user profile,[5] which contains their most recent tweets. This information could then be used to create a document-term matrix where a document is defined as a senator's recent tweets. To try to visualize this data we might project the document-term matrix onto its first and second principal components. At this point, it would be useful to have additional data about each senator (e.g., party affiliation, years in office).[6]

In the appendix, you will find a script to do the above data gathering and exploration. This should give you a sense of how you might go about collecting your data. For additional examples see [1].

[3] https://twitter.com/gov/lists/us-senate/members
[4] https://dev.twitter.com/rest/reference/get/lists/members
[5] https://dev.twitter.com/rest/reference/get/statuses/user_timeline
[6] https://sunlightlabs.github.io/congress/

# Appendix

## Code

```
import json
import re
from operator import itemgetter
import numpy as np
import matplotlib.pyplot as plt
import twitter

CONSUMER_KEY       = ""
CONSUMER_SECRET    = ""
OAUTH_TOKEN        = ""
OAUTH_TOKEN_SECRET = ""

auth = twitter.oauth.OAuth(OAUTH_TOKEN, OAUTH_TOKEN_SECRET,
                           CONSUMER_KEY, CONSUMER_SECRET)
api = twitter.Twitter(auth=auth)

# get the list of senators
senators = api.lists.members(owner_screen_name="gov", slug="us-senate", count=100)
with open("senators-list.json", "w") as f:
    json.dump(senators, f, indent=4, sort_keys=True)

# get all the senators' timelines
names = [d["screen_name"] for d in senators["users"]]
timelines = [api.statuses.user_timeline(screen_name=name) for name in names]
with open("timelines.json", "w") as f:
    json.dump(timelines, f, indent=4, sort_keys=True)

# could check who has the most followers
followers = [t[0]["user"]["followers_count"] for t in timelines]
zipped = zip(names, followers)
zipped.sort(key=itemgetter(1))

# get all the tweets and see what words are used
tweets = [" ".join([tweet["text"] for tweet in tweets]) for tweets in timelines]
words = [w for text in tweets for w in re.split('\W', text) if w]
vocab = sorted(set(words))

# construct document-term matrix
M = np.asmatrix(np.zeros([len(tweets), len(vocab)]))
for n, tweet in enumerate(tweets):
    for m, term in enumerate(vocab):
        M[n, m] = tweet.count(term)
```

```
# pca using scikit-learn
from sklearn import decomposition
pca = decomposition.PCA(n_components=2)
pca.fit(M)
pc = pca.transform(M)
plt.scatter(pc[:, 0], pc[:, 1])
plt.show()
```

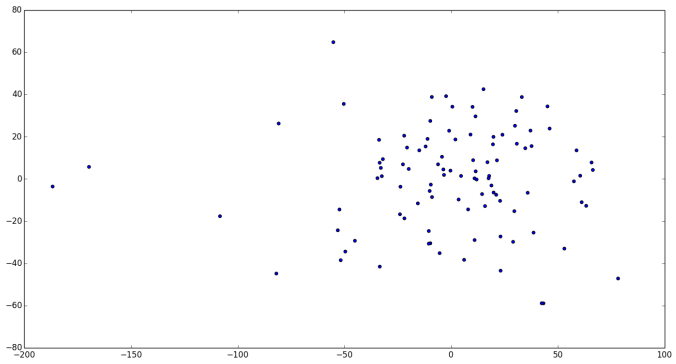


Figure 1: Document-term matrix projected on first and second principal axes.

## Links

### Webservices

- https://dev.twitter.com/overview/documentation
- https://developers.facebook.com/docs/graph-api
- https://developers.google.com/youtube/getting_started
- http://en.wikipedia.org/w/api.php
- http://www.mediawiki.org/wiki/API:Main_page
- https://developer.github.com/v3/

### Serialization

- http://en.wikipedia.org/wiki/Serialization
- http://en.wikipedia.org/wiki/Comparison_of_data_serialization_formats

- http://www.json.org/xml.html
- http://yaml.org/
- http://www.drdobbs.com/web-development/after-xml-json-then-what/240151851
- http://www.cowtowncoder.com/blog/archives/2012/04/entry_473.html

## References

[1] Matthew A Russell. *Mining the Social Web: Data Mining Facebook, Twitter, LinkedIn, Google+, GitHub, and More*. O'Reilly Media, Inc., 2013.