Appendix A: Cluster Identification and Transition Analysis in Python

This code takes messages that are on twitter, and extracts their hashtags. It then constructs a set of weighted and unweighted network structures based upon co-citation of hashtags within a tweet. The network diagrams are interpreted to have a set of clusters within them which represent 'conversations' that are happening in the pool of twitter messages. We track similarity between clusters from day to day to investigate how conversations develop.

Utilities

These scripts depend upon a number of external utilities as listed below:

```
import datetime
print 'started at %s'%datetime.datetime.now()
```

```
import json
import gzip
from collections import Counter
from itertools import combinations
import glob
import dateutil.parser
import pandas as pd
import os
import numpy as np
import datetime
import pickle
import subprocess
```

```
#load the locations of the various elements of the analysis
with open('config.json','r') as jfile:
    config = json.load(jfile)
print config
```

Data Files

We have twitter messages saved as compressed files, where each line in the file is the JSON object that the twitter sample stream returns to us. The files are created by splitting the streaming dataset according to a fixed number of lines - not necessarily by a fixed time or date range. A description of the collection process can be found in Appendix D.

All the files have the format `posts_sample_YYYYMMDD_HHMMSS_aa.txt` where the date listed is the date at which the stream was initialized. Multiple days worth of stream may be grouped under the same second, as long as the stream remains unbroken. If we have to restart the stream, then a new datetime will be added to the files.

```
# Collect a list of all the filenames that will be working with
files = glob.glob(config['data_dir']+'posts_sample*.gz')
print 'working with %i input files'%len(files)
```

Supporting Structures

Its helpful to have a list of the dates in the range that we'll be looking at, because we can't always just add one to get to the next date. Here we create a list of strings with dates in the format 'YYYYMMDD'. The resulting list looks like:

```
['20141101', '20141102', '20141103', ... '20150629', '20150630']
```

```
dt = datetime.datetime(2014, 11, 1)
end = datetime.datetime(2015, 7, 1)
step = datetime.timedelta(days=1)

dates = []
while dt < end:
    dates.append(dt.strftime('%Y%m%d'))
    dt += step

print 'investigating %i dates'%len(dates)
```

Step 1: Count hashtag pairs

The most data-intensive part of the analysis is this first piece, which parses all of the input files, and counts the various combinations of hashtags on each day.

In this demonstration we perform this counting in memory, which is sufficient for date ranges on the order of

weeks, but becomes unwieldy beyond this timescale.

```
#construct a counter object for each date
tallydict = dict([(date, Counter()) for date in dates])

#iterate through each of the input files in the date range
for i, zfile in enumerate(files):
    if i%10 == 0: #save every 10 files
        print i,
        with open(config['python_working_dir']+"tallydict.pickle", "wb" ) as picklefile:
            pickle.dump(tallydict, picklefile)
        with open(config['python_working_dir']+"progress.txt", 'a') as progressfile:
            progressfile.write(str(i)+': '+zfile+'\n')

    try:
        with gzip.open(zfile) as gzf:
            #look at each line in the file
            for line in gzf:
                try:
                    #parse the json object
                    parsed_json = json.loads(line)
                    # we only want to look at tweets that are in
                    # english, so check that this is the case.
                    if parsed_json.has_key('lang'):
                        if parsed_json['lang'] =='en':
                            #look only at messages with more than two hashtags,
                            #as these are the only ones that make connections
                            if len(parsed_json['entities']['hashtags']) >=2:
                                #extract the hashtags to a list
                                taglist = [entry['text'].lower() for entry in
                                           parsed_json['entities']['hashtags']]
                                # identify the date in the message
                                # this is important because sometimes messages
                                # come out of order.
                                date = dateutil.parser.parse(parsed_json['created_at'])
                                date = date.strftime("%Y%m%d")
                                #look at all the combinations of hashtags in the set
                                for pair in combinations(taglist, 2):
                                    #count up the number of alpha sorted tag pairs
                                    tallydict[date][' '.join(sorted(pair))] += 1
                except: #error reading the line
                    print 'd',
    except: #error reading the file
        print 'error in', zfile
```

We save the counter object periodically in case of a serious error. If we have one, we can load what we've already accomplished with the following:

```
with open(config['python_working_dir']+"tallydict.pickle", "r" ) as picklefile:
    tallydict = pickle.load(picklefile)

print 'Step 1 Complete at %s'%datetime.datetime.now()
```

Step 2: Create Weighted Edge Lists

Having created this sorted set of tag pairs, we should write these counts to files. We'll create one file for each day. The files themselves will have one pair of words followed by the number of times those hashtags were spotted in combination on each day. For Example:

```
PCMS champs 3
TeamFairyRose TeamFollowBack 3
instadaily latepost 2
LifeGoals happy 2
DanielaPadillaHoopsForHope TeamBiogesic 2
shoes shopping 5
kordon saatc 3
DID Leg 3
entrepreneur grow 11
Authors Spangaloo 2
```

We'll save these in a very specific directory structure that will simplify keeping track of our data down the road, when we want to do more complex things with it. An example:

```
twitter/
   20141116/
       weighted_edges_20141116.txt
   20141117/
       weighted_edges_20141117.txt
   20141118/
       weighted_edges_20141118.txt
   etc...
```

We create a row for every combination that has a count of at least two.

In this code we'll use some of the iPython 'magic' functions for file manipulation, which let us execute shell

commands as if through a terminal. Lines prepended with the exclaimation point `!` will get passed to the shell. We can include python variables in the command by prepending them with a dollar sign `$`.

```
for key in tallydict.keys(): #keys are datestamps
    #create a directory for the date in question
    date_dir = config['python_working_dir']+key
    if not os.path.exists(date_dir):
        os.makedirs(date_dir)
    #replace old file, instead of append
    with open(config['python_working_dir']+key+'/weighted_edges_'+key+'.txt', 'w') as fout:
        for item in tallydict[key].iteritems():
            if item[1] >= 2: #throw out the ones that only have one edge
                fout.write(item[0].encode('utf8')+' '+str(item[1])+'\n')
```

Now lets get a list of the wieghed edgelist files, which will be helpful later on.

```
weighted_files = glob.glob(config['python_working_dir']+'*/weight*.txt')
print 'created %i weighted edgelist files'%len(weighted_files)
print 'Step 2 Complete at %s'%datetime.datetime.now()
```

Step 3: Construct unweigheted edgelist

We make an unweighted list of edges by throwing out everything below a certain threshold. We'll do this for a range of different thresholds, so that we can compare the results later. Looks like:

```
FoxNflSunday tvtag
android free
AZCardinals Lions
usa xxx
كبلز متحره
CAORU TEAMANGELS
RT win
FarCry4 Games
```

We do this for thresholds between 2 and 15 (for now, although we may want to change later) so the directory structure looks like:

```
twitter/
20141116/
th_02/
```

```
unweighted_20141116_th_02.txt
th_03/
unweighted_20141116_th_03.txt
th_04/
unweighted_20141116_th_04.txt
etc...
20151117/
th_02/
unweighted_20141117_th_02.txt
etc...
etc...
```

Filenames include the date and the threshold, and the fact that these files are unweighted edge lists.

```
for threshold in range (2, 15):
    for infile_name in weighted_files:
        date_dir = os.path.dirname(infile_name)
        date = date_dir.split('/')[-1]
        weighted_edgefile = os.path.basename(infile_name)

        #create a subdirectory for each threshold we choose
        th_dir = date_dir+'/th_%02i'%threshold
        if not os.path.exists(th_dir):
            os.makedirs(th_dir)

        # load the weighted edgelists file and filter it to
        # only include values above the threshold
        df = pd.read_csv(infile_name, sep=' ', header=None,
                         names=['Tag1', 'Tag2', 'count'])
        filtered = df[df['count']>threshold][['Tag1','Tag2']]

        #write out an unweighted edgelist file for each threshold
        outfile_name = th_dir+'/unweighted_'+date+'_th_%02i'%threshold+'.txt'
        with open(outfile_name, 'w') as fout: #replace old file, instead of append
            for index, row in filtered.iterrows():
                try:
                    fout.write(row['Tag1']+' '+row['Tag2']+'\n')
                except:
                    print 'b',
```

Now lets get a list of all the unweighted edgelist files we created

```
unweighted_files = glob.glob(config['python_working_dir']+'*/*/unweight*.txt')
print 'created %i unweighted edgelist files'%len(unweighted_files)
print 'Step 3 Complete at %s'%datetime.datetime.now()
```

Step 4: Find the communities

We're using COS Parallel to identify k-cliques, so we feed each unweighted edge file into the `./maximal_cliques` preprocessor, and then the `./cos` algorithm.

The unweighed edgelist files should be in the correct format for `./maximal_cliques` to process at this point.

`./maximal_cliques` translates each node name into an integer to make it faster and easier to deal with, and so the output from this file is both a listing of all of the maximal cliques in the network, with an extension `.mcliques`, and a mapping of all of the integer nodenames back to the original text names, having extension `.map`.

It is a relatively simple task to feed each unweighed edgelist we generated above into the `./maximal_cliques` algorithm.

```
for infile in unweighted_files:
    th_dir = os.path.dirname(infile)
    th_file = os.path.basename(infile)
    #operate the command in the directory where we want the files created
    subprocess.call([os.getcwd()+'/'+config['maximal_cliques'], th_file], cwd=th_dir)
```

Step 5: Once this step is complete, we then feed the `.mcliques` output files into the `cosparallel` algorith.

```
maxclique_files = glob.glob(config['python_working_dir']+'*/*/*.mcliques')
print 'created %i maxcliques files'%len(maxclique_files)
print 'Step 4a Complete at %s'%datetime.datetime.now()
```

```
current_directory = os.getcwd()
for infile in maxclique_files:
    mc_dir = os.path.dirname(infile)
    mc_file = os.path.basename(infile)
    subprocess.call([os.getcwd()+'/'+config['cos-parallel'], mc_file], cwd=mc_dir)
```

```
community_files = glob.glob(config['python_working_dir']+'*/*/[0-9]*communities.txt')
print 'created %i community files'%len(community_files)
print 'Step 5 Complete at %s'%datetime.datetime.now()
```

Step 6: Translate back from numbers to actual words

The algorithms we just ran abstract away from the actual text words and give us a result with integer collections and a map back to the original text. So we apply the map to recover the clusters in terms of their original words, and give each cluster a unique identifier:

```
0 Ferguson Anonymous HoodsOff OpKKK
1 Beauty Deals Skin Hair
2 Family gym sauna selfie
etc...
```

```
# we'll be reading a lot of files like this,
# so it makes sense to create a function to help with it.
def read_cluster_file(infile_name):
    """ take a file output from COS and return a dictionary
    with keys being the integer cluster name, and
    elements being a set of the keywords in that cluster"""
    clusters = dict()
    with open(infile_name, 'r') as fin:
        for i, line in enumerate(fin):
            #the name of the cluster is the bit before the colon
            name = line.split(':')[0]
            if not clusters.has_key(name):
                clusters[name] = set()
            #the elements of the cluster are after the colon, space delimited
            nodes = line.split(':')[1].split(' ')[:-1]
            for node in nodes:
                clusters[name].add(int(node))
    return clusters


current_directory = os.getcwd()
for infile in community_files:
    c_dir = os.path.dirname(infile)
    c_file = os.path.basename(infile)

    #load the map into a pandas series to make it easy to translate
    map_filename = glob.glob('%s/*.map'%c_dir)
    mapping = pd.read_csv(map_filename[0], sep=' ', header=None,
                          names=['word', 'number'], index_col='number')

    clusters = read_cluster_file(infile)
    #create a named cluster file in the same directory
    with open(c_dir+'/named'+c_file, 'w') as fout:
        for name, nodes in clusters.iteritems():
            fout.write(' '.join([str(name)]+
                                [mapping.loc[int(node)]['word'] for node in list(nodes)]+
                                ['\n']))
```

```
print 'Step 6 Complete at %s'%datetime.datetime.now()
```

While we're at it, we'll write a function to read the files we're creating

```
def read_named_cluster_file(infile_name):
    """ take a file output from COS and return a """
    clusters = dict()
    with open(infile_name, 'r') as fin:
        for i, line in enumerate(fin):
            name = line.split(' ')[0]
            if not clusters.has_key(name):
                clusters[int(name)] = set()
            nodes = line.split(' ')[1:-1]
            for node in nodes:
                clusters[int(name)].add(node)
    return clusters
```

Step 7: Compute transition likelihoods

We want to know how a cluster on one day is related to a cluster on the next day. For now, we'll use a brute-force algorithm of counting the number of nodes in a cluster that are present in each of the subsequent day's cluster. From this we can get a likelihood of sorts for subsequent clusters.

We'll define a function that, given the clusers on day 1 and day 2, creates a matrix from the two, with day1 clusters as row elements and day2 clusters as column elements. The entries to the matrix are the number of nodes shared by each cluster.

```
#brute force, without the intra-day clustering
def compute_transition_likelihood(current_clusters, next_clusters):
    transition_likelihood = np.empty([max(current_clusters.keys())+1,
                                      max(next_clusters.keys())+1])
    for current_cluster, current_elements in current_clusters.iteritems():
        for next_cluster, next_elements in next_clusters.iteritems():
            #the size of the intersection of the sets
            transition_likelihood[current_cluster, next_cluster] = (
                           len(current_elements & next_elements) )
    return transition_likelihood
```

We want to compute transition matricies for all clusters with every k and every threshold. We'll save the matrix for transitioning from Day1 to Day2 in Day1's folder. In many cases, there won't be an appropriate date/threshold/k combination, so we'll just skip that case.

```
#this should compute and store all of the transition likelihoods

for current_date in dates[:-1]:
    next_date = dates[dates.index(current_date)+1]
    for threshold in range(2,15):
        for k in range(3, 20):

            current_file_name = config['python_working_dir']+'%s/th_%02i/named%i_communities.tx
                                                                threshold, k)

            next_file_name = config['python_working_dir']+'%s/th_%02i/named%i_communities.txt'%
                                                            threshold, k)

            if os.path.isfile(current_file_name) & os.path.isfile(next_file_name):
                current_clusters = read_named_cluster_file(current_file_name)
                next_clusters = read_named_cluster_file(next_file_name)

                transition = compute_transition_likelihood(current_clusters,
                                                           next_clusters)
                transitiondf = pd.DataFrame(data=transition,
                                            index=current_clusters.keys(),
                                            columns=next_clusters.keys())

                transitiondf.to_csv(current_file_name[:-4]+'_transition.csv')
```

```
transition_files = glob.glob(config['python_working_dir']+'*/*/named*_communities_transition.cs
print 'created %i transition matrix files'%len(transition_files)
print 'Step 6 Complete at %s'%datetime.datetime.now()
```

```
```