Dynamic Topic Modeling- Code Help

Introduction

In this document, we list the code that was used for the different models being used. The sections below will cover a separate piece of code.

Data Streaming

This code will access the stream of tweets for a given set of filters and save them to a location on Google Drive. It uses python libraries for connecting to twitter(tweepy) and Google Drive(pydrive)

To run the streaming, use command 'python streaming.py'. The programs runs continuously to gather data from twitter using the twitter's firehose API.

Table 1: slistener.py

```
from tweepy import StreamListener
import json, time, sys

from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
import threading, os

def my_threaded_func(file1, file2):
        file1.Upload() # Files.insert()
        os.remove(file2)

class SListener(StreamListener):

        def __init__(self, api = None, fprefix = 'streamer'):
                self.api = api or API()
                self.counter = 0
                self.fprefix = fprefix
                self.fileString = './streaming_data/' + fprefix + '.' + time.strftime('%Y%m%d-%H%M%S') +
'.json'
                self.output  = open(self.fileString, 'w')
                self.delout  = open('./streaming_data/' + 'delete.txt', 'a')

                self.gauth = GoogleAuth()
                self.gauth.LocalWebserverAuth()
                self.drive = GoogleDrive(self.gauth)

        def on_data(self, data):

                if  'in_reply_to_status' in data:
                        self.on_status(data)
                elif 'delete' in data:
                        delete = json.loads(data)['delete']['status']
                        if self.on_delete(delete['id'], delete['user_id']) is False:
                                return False
                elif 'limit' in data:
                        if self.on_limit(json.loads(data)['limit']['track']) is False:
```

```
                return False
        elif 'warning' in data:
                warning = json.loads(data)['warnings']
                print warning['message']
                return false

    def on_status(self, status):
        self.output.write(status + "\n")


        self.counter += 1

        if self.counter >= 20000:

                file1 = self.drive.CreateFile()
                file1.SetContentFile(self.fileString)
                thread = threading.Thread(target=my_threaded_func, args=(file1, self.fileString))
                thread.start()

                self.output.close()
                self.fileString = './streaming_data/' + self.fprefix + '.' + str(time.strftime('%Y%m%d-
%H%M%S')) + '.json'
                self.output = open(self.fileString, 'w')
                self.counter = 0

        return

    def on_delete(self, status_id, user_id):
        self.delout.write( str(status_id) + "\n")
        return

    def on_limit(self, track):
        sys.stderr.write(str(track) + "\n")
        return

    def on_error(self, status_code):
        sys.stderr.write('Error: ' + status_code + "\n")
        return False

    def on_timeout(self):
        sys.stderr.write("Timeout, sleeping for 60 seconds...\n")
        time.sleep(60)
        return
```

Table 2: streaming.py

```
from slistener import SListener
import time, tweepy, sys, os

##Authentication
consumer_key = "HDJNJYmgnkecU87UvAWSwNDdK"
consumer_secret = "ADpOXeHvuh6uiG0xieweuCXEQD2a7UQqLEAuov1QTrwDhtRNkI"
```

```
access_token = "215905178-burCGOJ9LhFAIL4Um3DXB1luN4fYYMtMWPyIr5MM"
access_token_secret = "4lSni5fHILbZspY056rzhZSTWkgcCIglpuhlwuNKUQ3DH"


#Using the credentials for tweepy
auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)

#Start the streaming
api = tweepy.API(auth)

def main():
    track = ['hillary', 'clinton', 'trump', 'donald', 'election2016', 'election']
    languages = ["en"]

    listen = SListener(api, 'Twitter.Elections.Data')
    stream = tweepy.Stream(auth = api.auth, listener = listen)

    print "Streaming started..."

    while(True):
            try:
                    stream.filter(languages= languages, track = track)
            except Exception as e:
                    print e
                    pass


if __name__ == '__main__':
    main()
```

Data Processing

This code will access the tweets stored on google Drive and stores them into database. It uses python libraries for connecting to Google Drive(pydrive) and local MongoDB(pymongo) instance.

To run the streaming, use command 'python streaming.py'. The programs runs continuously to gather data from twitter using the twitter's firehose API.

Table 3: processData.py

```
from nltk.tokenize import TweetTokenizer
from nltk.corpus import stopwords
from nltk.stem.snowball import SnowballStemmer
import json

class processData(object):
    """docstring for bagOfWords"""
    tknzr = TweetTokenizer()
    stops = set(stopwords.words('english'))
    stemmer = SnowballStemmer("english")

    def __init__(self, fileName):
```

```
        self.fileName = fileName
        self.tweetText = []
        # self.train_data_features = []
    """
    "textToWords
    "tokenizes text into separate words
    "@params
    "text - Input text
    "@returns
    "words - tokenized words
    """
    def textToWords(self, text):
        return self.tknzr.tokenize(text)

    def removeStopWords(self, words):
        return [w for w in words if w not in self.stops]

    def stemWords(self, words):
        return [self.stemmer.stem(w) for w in words]

    def getTweets(self):
        return self.tweetText

    def process(self):
        with open(self.fileName) as f:
            i = 0
            tweets = []
            for line in f:
                if(line != '\n'):
                            try:
                    tweets.append(json.loads(line))
                    text = tweets[i]['text']

                    words = self.textToWords(text)

                    stemmed_words = self.stemWords(words)

                    meaningful_words = self.removeStopWords(stemmed_words)

                    self.tweetText.append({"words" : meaningful_words, "time" : tweets[i]['created_at'], "tweet_id" :
tweets[i]['id_str']})
                    i += 1
                except Exception as e:
                    print 'Exception : ' + str(e)
                    pass
```

Table 4: fetchData.py

```
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from processData import processData
from pymongo import MongoClient
```

```
#Google drive connection
gauth = GoogleAuth()
gauth.LocalWebserverAuth()
drive = GoogleDrive(gauth) # Create GoogleDrive instance with authenticated GoogleAuth instance

#Local File
fileName = './streaming_data/tweetData.json'
logFile = './logfile.txt'

#MongoDB connection
client = MongoClient()
db = client.tweet_db
collection = db.tweets_1500_End

# Auto-iterate through all files in the root folder.
file_list = drive.ListFile({'q': "'0B7ziEhBHYh1bc3hEREo1eS1fMWs' in parents and trashed=false"}).GetList()
#file_list = drive.ListFile({'q': "'root' in parents and trashed=false"}).GetList()
start = 1500 + 179
total = len(file_list[start:])
i = 0
with open(logFile, 'w') as op:
    for f in file_list[start:]:
        i += 1
        print('File %d of %d (%d, %d)' % (i, total, start, len(file_list)))
        print('Title: %s' % (f['title']))
        op.write('\r\nTitle: ' + f['title'])
        f.GetContentFile(fileName)
        processedData = processData(fileName)
        processedData.process()
        result = collection.insert_many(processedData.getTweets())
        print('Tweets %d' % (len(result.inserted_ids)))
        op.write('\r\nTweets' + str(len(result.inserted_ids)))
print 'Completed All'
```

LDA-HMM

In this model, we perform static topic modeling using LDA for each time slice and is followed by HMM.

1. We run the LDA first using the command 'python lda_static.py'. This saves the LDA models in a file.

2. We apply the Multinomial HMM after this by the command 'python hmm_apply.py'. This saves the Multinomial HMM models in files.

3. To print the results, we can use the command 'python lda_saved.py'

We use some of the standard implementations of the package sklearn.

Note: This assumes that the data is present in a MongoDB instance with database as 'tweet_db' and each timeslice data in a separate collection 'tweets_11_**', where ** is the date. Each document is a MongoDB document as a token array.

Table 5: lda_static.py

```
from pymongo import MongoClient
import pprint
from time import time

from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.decomposition import NMF, LatentDirichletAllocation


dates = []
for i in range(12,14):
        dates.append("{0:0>2}".format(i))

#MongoDB connection
client = MongoClient()
db = client.tweet_db

def print_top_words(model, feature_names, n_top_words):
    for topic_idx, topic in enumerate(model.components_):
        print("Topic #%d:" % topic_idx)
        print(" ".join([feature_names[i]
                        for i in topic.argsort()[:-n_top_words - 1:-1]]))
    print()

for i in range(len(dates)):
        collection = db['tweets_11_' + dates[i]]

        print("Extracting tf features for LDA...")
        t0 = time()
        sentences = [[word for word in doc['words'] if len(word) != 1 and word[0].isalnum()] for doc in
collection.find()]

        n_features = 1000
        n_topics = 10
        n_top_words = 20
        print("done in %0.3fs." % (time() - t0))
        t0 = time()
        # Use tf (raw term count) features for LDA.
        print("Extracting tf features for LDA...")
        tf_vectorizer = CountVectorizer(max_df=0.80, min_df=50)

        tf = tf_vectorizer.fit_transform([' '.join(s) for s in sentences])
        print("done in %0.3fs." % (time() - t0))
        t0 = time()
        print("Fitting LDA models with tf features, "
              "n_samples=%d and n_features=%d..."
              % (len(sentences), n_features))
        lda = LatentDirichletAllocation(n_topics=n_topics, max_iter=5,
                                        learning_method='online',
                                        learning_offset=50.,
                                        random_state=0)
        lda.fit(tf)
        print("done in %0.3fs." % (time() - t0))

        print("\nTopics in LDA model:")
```

```
        tf_feature_names = tf_vectorizer.get_feature_names()
        print_top_words(lda, tf_feature_names, n_top_words)

        from sklearn.externals import joblib
        joblib.dump(lda, 'lda_11_' + dates[i] + '.pkl')
        joblib.dump(tf_vectorizer, 'tf_vectorizer_11_' + dates[i] + '.pkl')
        joblib.dump(tf, 'tf_11_' + dates[i] + '.pkl')
```

Table 6: hmm_apply.py

```
from pymongo import MongoClient
import pprint
from time import time

from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.decomposition import NMF, LatentDirichletAllocation
from sklearn.externals import joblib
from hmmlearn.hmm import MultinomialHMM
import numpy as np

n_top_words = 21
start_date = 3
end_date = 10 #13

def top_words(model, feature_names, n_top_words):
        topics = []
        for topic_idx, topic in enumerate(model.components_):

                #print("Topic #%d:" % topic_idx)
                try:
                        #print([i for i in topic.argsort()[:-n_top_words - 1:-1]])
                        topics.append(" ".join([feature_names[i] for i in topic.argsort()[:-n_top_words - 1:-
1]]).replace("rt ",""))
                except:
                        pass
                print()
        return topics


lda = []
tf_vectorizer = []
tf = []

for i in range(start_date, end_date):
        lda.append(joblib.load('/freespace/local/sp1467/pickles/lda_11_{:0>2}.pkl'.format(i)))
        tf_vectorizer.append(joblib.load('/freespace/local/sp1467/pickles/tf_vectorizer_11_{:0>2}.pkl'.format(i))
)
        tf.append(joblib.load('/freespace/local/sp1467/pickles/tf_11_{:0>2}.pkl'.format(i)))

print("\nTopics in LDA model:")

day_topics = []
for i in range(len(lda)):
```

```
        tf_feature_names = tf_vectorizer[i].get_feature_names()
        day_topics.append(top_words(lda[i], tf_feature_names, n_top_words))

overall_tf_vectorizer = CountVectorizer()
overall_tf_vectorizer.fit([topic for day in day_topics for topic in day])
tf = [overall_tf_vectorizer.transform(day) for day in day_topics]

MultinomialHMMModel = []
for i in range(len(overall_tf_vectorizer.get_feature_names())):
        X = [[topic.toarray()[0][i] for topic in day] for day in tf]
        MultinomialHMMModel.append(MultinomialHMM(n_components=1))
        print overall_tf_vectorizer.get_feature_names()[i]
        MultinomialHMMModel[i].fit(np.array(X))
```

Table 7: lda_saved.py

```
from pymongo import MongoClient
import pprint
from time import time

from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.decomposition import NMF, LatentDirichletAllocation
from sklearn.externals import joblib
from hmmlearn.hmm import MultinomialHMM

n_top_words = 20
start_date = 3
end_date = 13

def top_words(model, feature_names, n_top_words):
        topics = []
        for topic_idx, topic in enumerate(model.components_):

                print("Topic #%d:" % topic_idx)
                try:
                        print([feature_names[i] for i in topic.argsort()[:-n_top_words - 1:-1]])
                        topics.append([feature_names[i] for i in topic.argsort()[:-n_top_words - 1:-1]])
                except:
                        pass
                print()
        return topics


lda = []
tf_vectorizer = []
tf = []

for i in range(start_date, end_date):
        lda.append(joblib.load('/freespace/local/sp1467/pickles/lda_11_{:0>2}.pkl'.format(i)))
        tf_vectorizer.append(joblib.load('/freespace/local/sp1467/pickles/tf_vectorizer_11_{:0>2}.pkl'.format(i))
)
        tf.append(joblib.load('/freespace/local/sp1467/pickles/tf_11_{:0>2}.pkl'.format(i)))
```

```
print("\nTopics in LDA model:")

for i in range(len(lda)):
        tf_feature_names = tf_vectorizer[i].get_feature_names()
        print(top_words(lda[i], tf_feature_names, n_top_words))
        print('Day : ' + str(i))
```

Dynamic LDA

In this model, we use one of the standard models provided by genism. It is a dynamic extension to the LDA model and gives topics for each time slice.

To run the code, use the command 'python lda_seq.py'

Note: This assumes that the data is present in a MongoDB instance with database as 'tweet_db' and each timeslice data in a separate collection 'tweets_11_**', where ** is the date. Each document is a MongoDB document as a token array.

Table 8: lda_seq_py

```
from gensim import corpora
from gensim.models.ldaseqmodel import LdaSeqModel
from gensim.models.ldamodel import LdaModel
from pymongo import MongoClient
from sklearn.externals import joblib
from time import time
import itertools

dates = []
for i in range(3,13):
        dates.append("{0:0>2}".format(i))

#MongoDB connection
client = MongoClient()
db = client.tweet_db

sentences = []
sentences_len = []

vocab_words = joblib.load('/freespace/local/sp1467/pickles/vocab_stc.pkl')

K = 10

for w in vocab_words:
    if w[:5] != 'https' and w != '': # Removing links
        sentences.append([w])
dictionary = corpora.Dictionary(sentences)
print len(sentences)

print("Dumping Dictionary for LDA...")
t0 = time()
dictionary.save('/freespace/local/sp1467/NO_URLgensimDumps/vocab_lda.dict')  # store the dictionary, for future reference
print("done in %0.3fs." % (time() - t0))
```

```
corpus = []
for i in range(len(dates)):
        collection = db['tweets_11_' + dates[i]]

        print("Converting Day {0} to gensims BleiCorpus and saving for LDA...".format(dates[i]))
        t0 = time()
        sentences_time = [[word for word in doc['words'] if len(word) != 1 and word[0].isalnum()] for doc in
collection.find()]
        corpus_time = [dictionary.doc2bow(s) for s in sentences_time]
        corpora.BleiCorpus.save_corpus('/freespace/local/sp1467/NO_URLgensimDumps/corpora_lda_{:0>2}.l
da-c'.format(dates[i]), corpus_time)

        print("done in %0.3fs." % (time() - t0))


print("Appending corpora and calculaing lengths for each corpus...")
t0 = time()
corpus = corpora.BleiCorpus('/freespace/local/sp1467/NO_URLgensimDumps/corpora_lda_{:0>2}.lda-
c'.format(3))
sentences_len.append(len(list(corpus)))

print("done in %0.3fs." % (time() - t0))
for i in range(len(dates[1:])):
   t0 = time()
   corpus_time = corpora.BleiCorpus('/freespace/local/sp1467/NO_URLgensimDumps/corpora_lda_{:0>2}.lda-
c'.format(dates[1:][i]))
   sentences_len.append(len(list(corpus_time)))
   corpus = itertools.chain(corpus, corpus_time)
   print("done in %0.3fs." % (time() - t0))

t0 = time()
print 'Fitting LDA'
ldaseq = LdaSeqModel(corpus=corpus, time_slice= sentences_len, num_topics=K, initialize='gensim',
sstats=None, lda_model=None, obs_variance=0.5, chain_variance=0.005, passes=10, random_state=None,
lda_inference_max_iter=25, em_min_iter=6, em_max_iter=20, chunksize=100)
ldaseq.save("/freespace/local/sp1467/NO_URLgensimDumps/ldaseq")
print("done in %0.3fs." % (time() - t0))

dictionary = corpora.Dictionary.load('/freespace/local/sp1467/NO_URLgensimDumps/vocab_lda.dict')
ldaseq = LdaSeqModel.load('/freespace/local/sp1467/NO_URLgensimDumps/ldaseq')
for t in range(len(dates)):
   print [[dictionary[int(word)] for word, freq in topic] for topic in ldaseq.print_topics(time=t, top_terms=20)]
```

Dynamic STC

In this model, we apply the Sparse Topical Coding, with dynamic extension and a competition extension.

For running Dynamic STC.

1. Convert the documents to TF-IDF vectors using the command 'python stc_tf_idf.py'

2. Run 'python stc_brands.py' to create the brand vectors for the documents.

3. Run 'python stc_init.py' to initialize the parameteres for STC.

4. Now to run the STC use command 'python stc.py'

5. To print the topics run 'python stc_print.py'

We use the sklearn package for some data structures.

Note: This assumes that the data is present in a MongoDB instance with database as 'tweet_db' and each timeslice data in a separate collection 'tweets_11_**', where ** is the date. Each document is a MongoDB document as a token array.

Table 9: stc_tf_idf.py

```
from pymongo import MongoClient
import pprint
from time import time

from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.decomposition import NMF, LatentDirichletAllocation
from sklearn.externals import joblib
import random

dates = []
for i in range(3,13):
        dates.append("{0:0>2}".format(i))

#MongoDB connection
client = MongoClient()
db = client.tweet_db

'''
#Fetch corpora from DB
sentences_day = []
for i in range(len(dates)):
        collection = db['tweets_11_' + dates[i]]

        print("Reading data from DB...")
        t0 = time()
        sentences = [[word for word in doc['words'] if len(word) != 1 and word[0].isalnum() and word[:5] !=
'https'] for doc in collection.find()]

        print("done in %0.3fs." % (time() - t0))

        sentences_day.append([' '.join(s) for s in sentences])

#Create Vocabulary
vocab = {}
for sentences in sentences_day:
   for s in sentences:
      for w in s.split(' '):
         if w in vocab:
            vocab[w] += 1
         else:
            vocab[w] = 1
```

```
#Refine Vocabulary(Frequency < 50 removed)
i = 0
vocab_words = {}
for w in vocab:
    if vocab[w] > 50:
        vocab_words[w] = i
        i += 1

# Save Vocabulary
joblib.dump(vocab_words, '/freespace/local/sp1467/NO_URLpickles/vocab_stc.pkl')
'''
#Load Vocabulary
vocab_words = joblib.load('/freespace/local/sp1467/NO_URLpickles/vocab_stc.pkl')

print("Extracting tf-idf features for STC...")
tf_idf_vectorizer = TfidfVectorizer(vocabulary=vocab_words)


for i in range(len(dates)):
    print("Extracting from DB...")
    t0 = time()
    collection = db['tweets_11_' + dates[i]]

    # Removing tweets without the mention of either brands(Hillary or Trump)
    sentences = [[word for word in doc['words'] if len(word) != 1 and word[0].isalnum()] for doc in
collection.find() if 'donald' in doc['words'] or 'trump' in doc['words'] or 'hillary' in doc['words'] or 'clinton' in
doc['words']]

    # Random sample
    numDocs = len(sentences)
    randSample = random.sample(range(numDocs), 10000)
    sentences = [sentences[r] for r in randSample]
    # Random sample

    tf_idf = tf_idf_vectorizer.fit_transform([' '.join(s) for s in sentences])
    print("done in %0.3fs." % (time() - t0))
    print("Saving model to file...")
    t0 = time()

    joblib.dump(tf_idf_vectorizer, '/freespace/local/sp1467/NO_URLpickles/tf_idf_vectorizer_11_' + dates[i] +
'.pkl')
    joblib.dump(tf_idf, '/freespace/local/sp1467/NO_URLpickles/tf_idf_11_' + dates[i] + '.pkl')
    print("done in %0.3fs." % (time() - t0))
```

Table 10: stc_brands.py

```
import pprint
from time import time

from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.decomposition import NMF, LatentDirichletAllocation
from sklearn.externals import joblib
from scipy.sparse import csr_matrix
import numpy as np
```

```
dates = []
for i in range(7,13):
            dates.append("{0:0>2}".format(i))

def print_top_words(model, feature_names, n_top_words):
    for topic_idx, topic in enumerate(model.components_):
        print("Topic #%d:" % topic_idx)
        print(" ".join([feature_names[i]
                        for i in topic.argsort()[:-n_top_words - 1:-1]]))
    print()

def save_sparse_csr(filename,array):
    np.savez(filename,data = array.data ,indices=array.indices,
             indptr =array.indptr, shape=array.shape )

def load_sparse_csr(filename):
    loader = np.load(filename)
    return csr_matrix((  loader['data'], loader['indices'], loader['indptr']),
                         shape = loader['shape'])

tf_idf_vectorizer = []
tf_idf = []
brands = [('donald','trump'),('hillary','clinton')]

start_date = 3
end_date = 13

# Create Brand tf-idf vectors
for i in range(start_date, end_date):
            t = time()
            print 'Converting to Brands...'
            brand_vectors = []
            tf_idf_vectorizer.append(joblib.load('/freespace/local/sp1467/NO_URLpickles/tf_idf_vectorizer_11_{:0
>2}.pkl'.format(i)))
            tf_idf.append(joblib.load('/freespace/local/sp1467/NO_URLpickles/tf_idf_11_{:0>2}.pkl'.format(i)))
            t0, t1 = brands[0]
            c0, c1 = brands[1]

            for d in tf_idf[i-start_date]:
                brand_vector = [0, 0]
                if(d.toarray()[0][tf_idf_vectorizer[i-start_date].vocabulary_[t0]] > 0 or
d.toarray()[0][tf_idf_vectorizer[i-start_date].vocabulary_[t1]] > 0):
                    brand_vector[0] = 1
                if(d.toarray()[0][tf_idf_vectorizer[i-start_date].vocabulary_[c0]] > 0 or
d.toarray()[0][tf_idf_vectorizer[i-start_date].vocabulary_[c1]] > 0):
                    brand_vector[1] = 1
                brand_vectors.append([float(b)/sum(brand_vector) for b in brand_vector])

            save_sparse_csr('/freespace/local/sp1467/NO_URLpickles/brands_11_{:0>2}.pkl'.format(i),
csr_matrix(brand_vectors))
            print("done in %0.3f s." % (time() - t))
```

Table 11: stc_init.py

```
import pprint
from time import time

from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.decomposition import NMF, LatentDirichletAllocation
from sklearn.externals import joblib
from scipy.sparse import csr_matrix
import numpy as np
import random

def save_sparse_csr(filename,array):
    np.savez(filename,data = array.data ,indices=array.indices,
             indptr =array.indptr, shape=array.shape )

def load_sparse_csr(filename):
    loader = np.load(filename)
    return csr_matrix((  loader['data'], loader['indices'], loader['indptr']),
                         shape = loader['shape'])

start_date = 3
end_date = 13
tf_idf_vectorizer = joblib.load('/freespace/local/sp1467/NO_URLpickles/tf_idf_vectorizer_11_03.pkl')
brands = [('donald','trump'),('hillary','clinton')]

k = 10                #Number of Topics
L = 2                 #Number of brands
g = len(tf_idf_vectorizer.vocabulary_)
min_num = 0.0
max_num = 1.0

# Loop to initialize beta, phi, theta, z, r
for t in range(start_date, end_date):
    '''
    For each time slice set the initial params and load the known entities
    '''

    t0 = time()

    tf_idf = joblib.load('/freespace/local/sp1467/NO_URLpickles/tf_idf_11_{:0>2}.pkl'.format(t))

    print 'TF-IDF : ', tf_idf.shape

    brand_vectors =
load_sparse_csr('/freespace/local/sp1467/NO_URLpickles/brands_11_{:0>2}.pkl.npz'.format(t))

    print 'Brand Vectors : ', brand_vectors.shape

    beta = np.random.uniform(min_num, max_num, (k, g-1))
    print 'Beta : ', beta.shape

    beta = [np.append(np.append([min_num],np.sort(beta_k)),[max_num]) for beta_k in beta]
    beta = [[beta_k[i+1] - beta_k[i] for i in range(g)] for beta_k in beta]

    print 'Beta : ', np.array(beta).shape
```

```
    phi = np.random.uniform(min_num, max_num, (k, L-1))
    print 'Phi : ', phi.shape

    phi = [np.append(np.append([min_num],np.sort(phi_k)),[max_num]) for phi_k in phi]
    phi = [[phi_k[l+1] - phi_k[l] for l in range(L)] for phi_k in phi]

    print 'Phi : ', np.array(phi).shape

    D = tf_idf.shape[0]

    theta_documents = []
    z_words_documents = []
    r_words_documents = []
    for d in range(D):
        theta = np.random.uniform(min_num, max_num, k)
        theta_documents.append(np.array(theta))
        z_words = []
        N = len(tf_idf[d].nonzero()[0])
        for n in range(N):
            z = np.random.uniform(min_num, max_num, k)
            z_words.append(np.array(z))
        z_words_documents.append(np.array(z_words))

        r_words = []
        for l in range(L):
            r = np.random.uniform(min_num, max_num, k)
            r_words.append(np.array(r))
        r_words_documents.append(np.array(r_words))

    print 'Theta :', np.array(theta_documents).shape

    print 'Z : ', np.array(z_words_documents).shape

    print 'R : ', np.array(r_words_documents).shape

    print 'Day {0} complete, time taken '.format(t), time() - t0

    joblib.dump(beta, '/freespace/local/sp1467/NO_URLpickles/beta_11_{:0>2}.pkl'.format(t))
    joblib.dump(phi, '/freespace/local/sp1467/NO_URLpickles/phi_11_{:0>2}.pkl'.format(t))
    joblib.dump(np.array(theta_documents),
'/freespace/local/sp1467/NO_URLpickles/theta_documents_11_{:0>2}.pkl'.format(t))
    joblib.dump(np.array(z_words_documents),
'/freespace/local/sp1467/NO_URLpickles/z_words_documents_11_{:0>2}.pkl'.format(t))
    joblib.dump(np.array(r_words_documents),
'/freespace/local/sp1467/NO_URLpickles/r_words_documents_11_{:0>2}.pkl'.format(t))
```

Table 12: stc.py

```
from time import time
t0 = time()
import pprint
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
```

```
from sklearn.decomposition import NMF, LatentDirichletAllocation
from sklearn.externals import joblib
from scipy.sparse import csr_matrix
import numpy as np
from scipy import optimize

print 'Time taken to load libraries : ', time()-t0

def save_sparse_csr(filename,array):
    np.savez(filename,data = array.data ,indices=array.indices,
             indptr =array.indptr, shape=array.shape )

def load_sparse_csr(filename):
    loader = np.load(filename)
    return csr_matrix((  loader['data'], loader['indices'], loader['indptr']),
                         shape = loader['shape'])

def proj_unit_simplex(y):
    u = np.sort(y)
    u = u[::-1]
    rho_vec = [j for j in range(len(u)) if ((u[j] + float(1 - np.sum(u[0:j+1]))/(j+1)) > 0)]
    rho = rho_vec[len(rho_vec)-1]
    lam = float(1-np.sum(u[0:rho+1]))/(rho+1)
    return [y_i + lam if y_i + lam > 0 else 0 for y_i in y]

start_date = 3
end_date = 13
tf_idf_vectorizer = joblib.load('/freespace/local/sp1467/NO_URLpickles/tf_idf_vectorizer_11_03.pkl')
brands = [('donald','trump'),('hillary','clinton')]

K = 10                #Number of Topics
L = 2                 #Number of brands
G = len(tf_idf_vectorizer.vocabulary_)
min_num = 0.0
max_num = 10.0 ** 6
iterations_inner = 2 #Coordinate descent iterations
iterations_outer = 2 #E-M Algorithms

#Hyperparameters
rho_1 = 1.0
sigma_1 = 1.0
rho_2 = 1.0
sigma_2 = 1.0
delta_1 = 1.0
pi_1 = 1.0
delta_2 = 1.0
pi_2 = 1.0
tau_0 = 5.0
tau = 10.0
epsilon = 1.0
M = 1000

for t in range(start_date, end_date):
    '''
    For each time slice
    '''
```

```
    t0 = time()
    w = joblib.load('/freespace/local/sp1467/NO_URLpickles/tf_idf_11_{:0>2}.pkl'.format(t))
    g = load_sparse_csr('/freespace/local/sp1467/NO_URLpickles/brands_11_{:0>2}.pkl.npz'.format(t))
    beta = np.array(joblib.load('/freespace/local/sp1467/NO_URLpickles/beta_11_{0:0>2}.pkl'.format(t)))
    phi = np.array(joblib.load('/freespace/local/sp1467/NO_URLpickles/phi_11_{0:0>2}.pkl'.format(t)))
    if t != start_date:
        beta_prev =
joblib.load('/freespace/local/sp1467/NO_URLpickles_5_iter/beta_trained_11_{0:0>2}.pkl'.format(t-1))
        phi_prev =
joblib.load('/freespace/local/sp1467/NO_URLpickles_5_iter/phi_trained_11_{0:0>2}.pkl'.format(t-1))
    theta = joblib.load('/freespace/local/sp1467/NO_URLpickles/theta_documents_11_{0:0>2}.pkl'.format(t))
    z = joblib.load('/freespace/local/sp1467/NO_URLpickles/z_words_documents_11_{0:0>2}.pkl'.format(t))
    r = joblib.load('/freespace/local/sp1467/NO_URLpickles/r_words_documents_11_{0:0>2}.pkl'.format(t))

    print 'Time taken to load parameters : ', time()-t0
    t0 = time()
    D = w.shape[0]

    print 'Coordinate Descent : '

    for i_outer in range(iterations_outer):

        t2 = time()

        # Step 1
        # Fix beta & phi and optimize theta, z and r
        for i_inner in range(iterations_inner):

            # Step 1.a
            # Fix theta and optimize z and r
            print 'Day :', t, ' OI : ', i_outer ,' of ', iterations_outer, ' II :', i_inner, ' of ', iterations_inner, ' Step 1.a'
            for d in range(D):
                N = len(w[d].nonzero()[0])
                for n in range(N):
                    nz_n = np.sort(w[d].nonzero()[1])[n]
                    for k in range(K):
                        z[d][n][k] = max(0, (w[d].toarray()[0][nz_n]*beta[k][nz_n] + theta[d][k] - beta[k][nz_n] *
sum([z[d][n][j]*beta[j][nz_n] for j in range(K) if j != k]) - (rho_1/2))/((beta[k][nz_n] ** 2) + sigma_1))
                for l in range(L):
                    for k in range(K):
                        r[d][l][k] = max(0, (g[d].toarray()[0][l]*phi[k][l] + theta[d][k] - phi[k][l] * sum([r[d][l][j]*phi[j][l]
for j in range(K) if j != k]) - (rho_2/2))/((phi[k][l] ** 2) + sigma_2))


            # Step 1.b
            # Fix z & r and optimize theta
            # lambda = gamma
            print 'Day :', t, ' OI : ', i_outer ,' of ', iterations_outer, ' II :', i_inner, ' of ', iterations_inner, ' Step 1.b'
            for d in range(D):
                #print 'Day :', t, ' OI : ', i_outer ,' of ', iterations_outer, ' II :', i_inner, ' of ', iterations_inner, ' :
Document ', d, ' of ', D, ' Step 1.b'
                N = len(w[d].nonzero()[0])
                for k in range(K):
                    theta[d][k] = 1.0/(1+N) * sum([z[d][n][k] for n in range(N)])

        # Step 2
```

```
        # Fix theta, z & r and optimize beta and phi

        '''
        #def log_poisson_online(optimizer, delta, pi):
        #    optimizer = optimizer.reshape(K, G)
        #    logPoisson = 0
        #    for d in range(D):
        #        N = len(w[d].nonzero()[0])
        #        for n in range(N):
        #            nz_n = np.sort(w[d].nonzero()[1])[n]
        #            logPoisson += delta * np.linalg.norm(w[d].toarray()[0][nz_n] -
(z[d][n].T.dot(optimizer[:,nz_n])))
        #
        #    if t == start_date:
        #        return logPoisson
        #    else:
        #        return logPoisson + pi * ((beta - beta_prev) ** 2).sum()

        #flat_beta = beta.flatten()
        #flat_beta = optimize.fmin_cg(log_poisson_online, flat_beta, args=(delta_1, pi_1), epsilon=0.5, maxiter=
1)
        #beta = flat_beta.reshape(K, G)
        '''

        alpha = tau_0/(i_outer + tau)

        for d_batch in range(D/M):
            beta_gradient = np.array([[0.0 for g_i in range(G)] for k in range(K)])
            phi_gradient = np.array([[0.0 for l in range(L)] for k in range(K)])

            print 'Day :', t, ' OI : ', i_outer ,' of ', iterations_outer, ' : Document batch ', d_batch, ' of ', D/M, ' Step 2.a'
            for d  in range(M):
                N = len(w[d_batch + d].nonzero()[0])
                for n in range(N):
                    nz_n = np.sort(w[d_batch + d].nonzero()[1])[n]
                    beta_gradient[:,nz_n] = (1 - (w[d_batch + d].toarray()[0][nz_n]/(z[d_batch +
d][n].T.dot(beta[:,nz_n]) + epsilon))) * z[d_batch + d][n]
                for l in range(L):
                    phi_gradient[:,l] = (1 - (g[d_batch + d].toarray()[0][l]/(r[d_batch + d][l].T.dot(phi[:, l]) + epsilon))) *
r[d_batch + d][l]


            beta -= alpha/M * beta_gradient
            #print phi
            phi -= alpha/M * phi_gradient
            beta = np.array([proj_unit_simplex(beta[k]) for k in range(K)])
            #print phi
            phi = np.array([proj_unit_simplex(phi[k]) for k in range(K)])
            #print phi

        if t != start_date:
            print 'Day :', t, ' OI : ', i_outer ,' of ', iterations_outer, ' Step 2.b'
            beta_gradient = np.array([[0.0 for g_i in range(G)] for k in range(K)])
            phi_gradient = np.array([[0.0 for l in range(L)] for k in range(K)])

            beta_gradient = alpha * (beta - beta_prev)
```

```
            phi_gradient = alpha * (phi - phi_prev)

            beta -= beta_gradient
            phi -= phi_gradient

            beta = np.array([proj_unit_simplex(beta[k]) for k in range(K)])
            phi = np.array([proj_unit_simplex(phi[k]) for k in range(K)])
        print 'Time taken to for this iteration : ', time()-t2
    t1 = time()
    print 'Dumping...'
    joblib.dump(beta, '/freespace/local/sp1467/NO_URLpickles_5_iter/beta_trained_11_{0:0>2}.pkl'.format(t))
    joblib.dump(phi, '/freespace/local/sp1467/NO_URLpickles_5_iter/phi_trained_11_{0:0>2}.pkl'.format(t))
    joblib.dump(theta,
'/freespace/local/sp1467/NO_URLpickles_5_iter/theta_documents_trained_11_{0:0>2}.pkl'.format(t))
    joblib.dump(z,
'/freespace/local/sp1467/NO_URLpickles_5_iter/z_words_documents_trained_11_{0:0>2}.pkl'.format(t))
    joblib.dump(r,
'/freespace/local/sp1467/NO_URLpickles_5_iter/r_words_documents__trained_11_{0:0>2}.pkl'.format(t))
    print 'Time taken to dump parameters : ', time()-t1

    print 'Time taken to for this time slice : ', time()-t0
```

Table 13: stc_print.py

```
from sklearn.externals import joblib

start_date = 3
end_date = 13
K = 10

tf_idf_vectorizer = joblib.load('/freespace/local/sp1467/NO_URLpickles/tf_idf_vectorizer_11_03.pkl')
inv_map = {v: k for k, v in tf_idf_vectorizer.vocabulary_.iteritems()}

for t in range(start_date, end_date):
    print 'Day {:0>2}'.format(t)
    beta = joblib.load('/freespace/local/sp1467/NO_URLpickles_5_iter/beta_trained_11_{0:0>2}.pkl'.format(t))
    phi = joblib.load('/freespace/local/sp1467/NO_URLpickles_5_iter/phi_trained_11_{0:0>2}.pkl'.format(t))
    for k in range(K):
        print 'Topic #{0}'.format(k)
        print 'Trump : {0}, Clinton : {1}'.format(phi[k][0],phi[k][1])
        print ' '.join([inv_map[i] for i in beta[k].argsort()[-20:][::-1]])
```

W2V-Fisher Vectors-Affinity Propagation

w2v.py - input from mongodb database for a specific day

- output tweets_<month>_<day>_vocab.txt and tweets_<month>_<day>.txt

fisher_vector_help.py - input tweets_<month>_<day>.txt

- output fishers_<month>_<day>.txt

affinity_help.py - input fishers_<month>_<day>.txt and tweets_<month>_<day>_vocab.txt

- output clustering results in these files located in https://drive.google.com/drive/folders/0B2OTdTSHCBp2ZjVpemhnSklONDQ

RESULTS_<month>_<day>_cluster_<K>_total_#.txt - Cluster K with a total of # samples for that particular day.

RESULTS_<month>_<day>_cluster_labels.txt - The cluster labels for all samples

RESULTS_<month>_<day>_cluster_exemplars.txt - The exemplars as readable words.

To run them, use the commands in sequence

1. 'python w2v.py'

2. 'python fisher_vector_help.py'

3. 'python affinity_help.py'

Table 14: w2v.py

```
import gensim, logging
#logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', level=logging.INFO)
import pandas as pd
import nltk
import json
from nltk.tokenize import TweetTokenizer
from nltk.corpus import stopwords
from nltk.stem.snowball import SnowballStemmer
import os.path
import pickle
import pymongo
from pymongo import MongoClient

# Variable Initialization
tweets = []
sentences=[]
words=[]
sentence=[]
i=0
vocab=[]
#text;

stemmer = SnowballStemmer("english")
tknzr = TweetTokenizer()
stops = set(stopwords.words('english'))

class word2vec():

        def __init__(self, dirname,modelFileName):
                self.dirname = dirname
                self.fname=[]
                self.modelFileName=os.path.join(self.dirname, modelFileName)
```

```
        def textToWords(self, text):
                return tknzr.tokenize(text)

        def removeStopWords(self, words):
                return [w for w in words if not w in stops]

        def stemWords(self, words):
                return [stemmer.stem(w) for w in words]

        # read the entire file into a python array

        def preprocess(self):


                client = MongoClient()
                db = client.local
                collection = db.tweets_11_12
                sentences=([[word for word in doc['words'] if len(word) != 1 and word[0].isalnum()] for doc
in collection.find()])
                print(sentences)

                self.model = gensim.models.Word2Vec(sentences, min_count=50,size=200,sg=1)

                vocab_dict={}
                w2v_doc_matrix= [[0 for col in range(200)] for row in range(0,len(self.model.vocab))]
                count=0
                for i in self.model.vocab:
                        print(i,self.model[i])
                        vocab_dict[count]=i
                        w2v_doc_matrix[count]=self.model[i]
                        count=count+1


                pickle.dump(vocab_dict, open("tweets_11_12_vocab.txt", "wb" ))
                pickle.dump(w2v_doc_matrix, open("tweets_11_12.txt", "wb" ))



        def process(self):
                #Checks if a word vector file is created
                # yes: loadsthe existing model and trains with more sentence.
                # no: fits the model,and saves the model.
                #self.getFile(self.dirname)
                #print self.fname

                self.preprocess()



w2v = word2vec("","")
w2v.process()
```

Table 15: fisher_vector_help.py

```
import numpy as np
import pickle
from sklearn.mixture import GMM


data = pickle.load(open("tweets_11_12.txt",'rb'))
K = 20
gmm = GMM(n_components=K, covariance_type='diag')
gmm.fit(data)#xx_tr
fv = {}
for i in range(0,len(data)):
    #fv[i] = fisher_vector(data(i), gmm)

    xx = data[i]

    xx = np.atleast_2d(xx)
    N = xx.shape[0]

    # Compute posterior probabilities.
    Q = gmm.predict_proba(xx)  # NxK

    # Compute the sufficient statistics of descriptors.
    Q_sum = np.sum(Q, 0)[:, np.newaxis] / N
    Q_xx = np.dot(Q.T, xx) / N
    Q_xx_2 = np.dot(Q.T, xx ** 2) / N

    # Compute derivatives with respect to mixing weights, means and variances.
    d_pi = Q_sum.squeeze() - gmm.weights_
    d_mu = Q_xx - Q_sum * gmm.means_
    d_sigma = (
        - Q_xx_2
        - Q_sum * gmm.means_ ** 2
        + Q_sum * gmm.covars_
        + 2 * Q_xx * gmm.means_)

        # Merge derivatives into a vector.
        #return np.hstack((d_pi, d_mu.flatten(), d_sigma.flatten()))
    fv[i] = np.hstack((d_pi, d_mu.flatten(), d_sigma.flatten()))
pickle.dump(fv, open("fishers_11_12.txt", "wb" ))
```

Table 16: affinity_help.py

```
import pickle
import collections
from sklearn.cluster import AffinityPropagation
from sklearn.decomposition import PCA

# load fishers
X = pickle.load(open("fishers_11_12.txt",'rb'))

#PCA
pca = PCA(n_components=8000)
```

```
pca.fit(list(X.values()))
result = pca.explained_variance_ratio_
curr_sum = 0
index = 0
while curr_sum < 0.9:
    curr_sum+=result[index]
    index = index + 1

pca = PCA(n_components=index)
reduced_X = PCA(n_components=index).fit_transform(list(X.values()))

# affinity
af = AffinityPropagation(preference=-300).fit(reduced_X)
exemplars = af.cluster_centers_indices_
labels = af.labels_

# load dictionary
vocab = pickle.load(open("tweets_11_12_vocab.txt",'rb'))

# making sure things look good
exemplar_words = {}
for i in range(0,len(exemplars)):
    exemplar_words[i] = vocab[exemplars[i]]
    print(labels[exemplars[i]])
    exemplar_words[i]


# PRINTING TO FILES

# affinity labels for the clusters
pickle.dump(labels, open("RESULTS_11_12_cluster_labels.txt", "wb" ))


# printing exemplars
all_exemplars = ''
for i in range(0,len(exemplars)):
    all_exemplars = all_exemplars + '\n' + vocab[exemplars[i]]

text_file = open('RESULTS_11_12_cluster_exemplars.txt', "w")
text_file.write(all_exemplars[1:len(all_exemplars)])
text_file.close()


# print exemplars and all the words in the cluster
for ex in range(0,len(exemplars)):
    total = 0
    for lbl in range(0,len(labels)):
        if labels[lbl] == ex:
            total = total + 1;
    # Write the cluster exemplar
    text_file = open('RESULTS_11_12_cluster_'+str(ex)+'_total_'+str(total)+'.txt', "w")
    text_file.write(vocab[exemplars[ex]]+'\n')
    text_file.write("-----------------------------------\n")
    # Write the cluster words
    for lbl in range(0,len(labels)):
        if labels[lbl] == ex:
```

```
        text_file.write(vocab[lbl]+'\n')
    text_file.close()
```

The full code along with the data can be found at the google drive location
https://drive.google.com/drive/folders/0B7ziEhBHYh1bX1BQVHVEUFFZSU0?usp=sharing