Dynamic Topic Modelling Tutorial

Matias Hurtado
Engineering Student, Pontificia Universidad Católica de Chile
mihurtado@uc.cl

Advisor: Denis Parra
Assistant Professor, Pontificia Universidad Católica de Chile
dparra@ing.puc.cl

Social Computing and visualization group, PUC Chile
http://socialcomputing.ing.puc.cl

Introduction

The following tutorial explains how to use Dynamic Topic Modelling (Blei and Lafferty, 2006), an extension of LDA (Blei et al., 2003). It's based in the implementation by S. Gerrish and D. Blei available at https://code.google.com/p/princeton-statistical-learning/downloads/detail?name=dtm_release-0.8.tgz .

The procedure here was used during my work that led to the research article "Twitter in Academic Conferences: Usage, Networking and Participation over Time" (Wen et al, 2014). Here we analyzed tweets posted during academic conferences in the Computer Science domain, and specifically we obtain the topic evolution over five years.

Prerequisite: Following this tutorial may require some basic knowledge of python, SQL databases and basic command line instructions.

Required files

You must get Gerris & Blei DTM release files. You can download the compiled binary files from https://github.com/magsilva/dtm
You can download the python scripts from dtm_gensim repository on GitHub.
https://github.com/mihurtado/dtm_gensim
If you want the database used, you can ask for it by e-mailing dparra@ing.puc.cl

The Database

In this tutorial, we will use a small fragment of the database used for the investigation “Twitter in Academic Conferences: Usage, Networking and Participation over Time”. This fragment consist on a SQL database with ~5K tweets posted on the conference WWW . There are many dimensions of information about each tweet, such as author, date, tweetid, content, conference, year. We are going to use:

- ttID: it’s the tweet id
- content: it’s the content of the tweet
- conference: it’s the conference name
- category_index: it’s the year of the conference
- lang: we have classificate the tweets by language, because this DTM tutorial will work only on english tweets.

We’d like to share the datasets of tweets with all this information, but since Twitter API Terms of Service use does not allow us, we can share you the list of tweetIDs by e-mailing mihurtado@uc.cl.

We loaded the data into a MySQL database with this command:

```
$ mysql -u [uname] -p[pass] [twitter_conferences] < [database.sql]
```

You must load the database into a SQL server. We recommend you to install MYSQL, following this tutorial http://dev.mysql.com/doc/refman/5.1/en/windows-installation.html. Then, you must load the database, using the MYSQL command line, with your credentials.

```
$ mysql -u [uname] -p[pass] [twitter_conferences] < [database.sql]
```

The Tutorial

1. Getting and transforming the Data

We will use the generated bin file of Blei DTM, called dtm-win64.exe or dtm-win32.exe, depending on your Windows architecture. In magsilva's github account there are also binaries for Mac OSX. To execute the program, we need two essential files:

- prefix-seq.dat: contains the TimeStamps. The structure of the file must be:

```
Number_Timestamps
number_docs_time_1
...
number_docs_time_i
...
number_docs_time_NumberTimestamps
```

In our case (tweets of the conference), our timestamps are years from 2009 to 2013, so our file must look like:

```
5
number_docs_2009
number_docs_2010
number_docs_2011
number_docs_2012
number_docs_2013
```

We are missing the numer_docs_year number, so have to calculate it.

- prefix-mult.dat: contains the data vectorized. The structure of each line in this file must be like this:

```
unique_word_count index1:count1 index2:count2 ... indexn:countn
```

Where each line represent a document. In our case, each line is a tweet. The unique_word_count is the total number of unique words, each index is an identifier for a word that would be represented by a dictionary, and each count is how many times the word shows up in the document/tweet.

We also are going to construct two more files, that will help us in the interpretation of the results.

- dictionary.dict: we'll create a dictionary where each word connects with the index in prefix-mult.dat file.
- vocabuary.dat: it will store each word that appears in the tweets.
- metadata.dat: it will store the tweet simple metadata (tweet id, date and content)

2. Generating the Input Files

If you already have the files in the specified format, you can skip this section and go directly to section 3 “Back to DTM Blei’s executable”.

We will use Python to collect all the tweets from the SQL database, then we’ll create a dictionary and finally we’ll transform the text of the tweets to a Vector Space Model (VSM) representation.

2.1 Read From database

Assuming that your tweets are in a MySQL database and you want to write them into a file to process them with DTM, go through this part of the tutorial; if not, jump directly to section **3** “**Generating the Corpus with gensim**”.
So first we’ll get the tweets from the SQL database. We will use simple SQL commands to extract the information we need. We will apply some filters over the content, such as removing the hashtags and mentions, removing stopwords, stemming, and lemmatization, and keeping only nouns and adjectives.

Initial configuration:

```
#Import some modules for reading and getting data.
#If you don't have this modules, you must install them.
import csv
import MySQLdb
import re
import nltk
from nltk.stem.wordnet import WordNetLemmatizer
from nltk.corpus import stopwords
import os
from gensim import corpora, models, similarities #to create a dictionary


#Set years, this would be the timestamps
time_stamps = ['2009', '2010', '2011', '2012', '2013']
#Set the conference name to be analyzed
conference = ''

#DB MYSQL Connect. Put your credentials here.
db_host = 'localhost' #Host
db_user = 'user' #User
db_pass = 'password' #Password
db_database = 'twitter_conferences' #Database

##Connect...
db = MySQLdb.connect(host=db_host, user=db_user, passwd=db_pass,db=db_database)
```

Getting the data. Here we will get the data from the database, keep it in memory (tweets python list) and save it for future reference in metadata.dat file.

```
#Set metadata output file
dat_outfile = open(os.path.join('data', conference, 'metadata.dat'), 'w')
dat_outfile.write('id\tdate\tcontent\n') #write header

tweets = list()
#Set total_tweets list per year, starting at 0
total_tweets_list = [0 for year in time_stamps]

#Analyze each year..
time_stamps_count = 0

for year in time_stamps: #For each year
    print('Analyzing year ' + str(year))

    #Set total_tweets to 0
    total_tweets = 0

    #Get tweets with mysql
    cursor = db.cursor()

    #Query
     query = "SELECT ttID, content, category_index FROM con_tweets_filtered WHERE conference
= '" + conference + "' and category_index = " + year + " and relevant=1 and lang='en'"

    #Execute query
    cursor.execute(query)
    result = cursor.fetchall() #store results
    cursor.close()

    #For each result (tweet), get content
    for line in result:

        #Remove @xxxx and #xxxxx
        content = [unicode(word.lower(), errors='ignore') for word in line[1].split() if
                  word.find('@') == -1 and word.find('#') == -1 and word.find('http') == -1]

        #join words list to one string
        content = ' '.join(content)
```

```
        #remove symbols
        content = re.sub(r'[^\w]', ' ', content)

        #remove stop words, this could also be done in the next step with gensim
       content = [word for word in content.split() if word not in stopwords.words('english')
                 and len(word) > 3 and not any(c.isdigit() for c in word)]

        #join words list to one string
        content = ' '.join(content)

        #Stemming and lemmatization
        lmtzr = WordNetLemmatizer()

        content = lmtzr.lemmatize(content)

        #Filter only nouns and adjectives
        tokenized = nltk.word_tokenize(content)
        classified = nltk.pos_tag(tokenized)

        #join words list to one string
        content = ' '.join(content)

        tweets.append([line[0], content, line[2]])
        total_tweets += 1
        dat_outfile.write(str(line[0]) + '\t' + str(line[2]) + '\t' + content)
        dat_outfile.write('\n')

    #Add the total tweets to the total tweets per year list
    total_tweets_list[time_stamps_count] += total_tweets

    time_stamps_count+=1

dat_outfile.close() #Close the metadata file

print('Done collecting tweets')
```

Now that you have all the tweets and you have already counted how many are there, you can write the prefix-seq.dat file with the specificated format.

```
#Write seq file
seq_outfile = open(os.path.join('data', conference, 'foo-seq.dat'), 'w')
seq_outfile.write(str(len(total_tweets_list)) + '\n') #number of TimeStamps

for count in total_tweets_list:
    seq_outfile.write(str(count) + '\n') #write the total tweets per year (timestamp)
```

```
seq_outfile.close()

print('Done writing seq')
```

Next step is to write the mult.dat file. To do this, we must have our documents vectorized and our dictionary.

3 Generating the Corpus with gensim

We will use the gensim python package to generate our corpus. You can see the full tutorial and other options in this page:
http://radimrehurek.com/gensim/tut1.html#from-strings-to-vectors.

Following the tutorial of gensim, we'll first create a dictionary and a vocabulary using the fields created in the previous step (tweet content filtered with stemming, lemmatization, lowercase, etc.), metadata.dat, which columns are: tweet_id, tweet_date, tweet_content.

Considering that documents is a list containing the tweets, e.g.
['I'm at WWW conference 2013, hope to have a great time!', 'This is such an interesting project, #HT2010',...]
We also will remove stop words and words than appear only once (you must define a stopwords list or use, for instance, the one available in NLTK library).

```
stoplist = set('for a of the and to in'.split())

#Construct the dictionary
dictionary = corpora.Dictionary(line[1].lower().split() for line in tweets)

# remove stop words and words that appear only once
stop_ids = [dictionary.token2id[stopword] for stopword in stoplist
            if stopword in dictionary.token2id]
once_ids = [tokenid for tokenid, docfreq in dictionary.dfs.iteritems() if docfreq == 1]
dictionary.filter_tokens(stop_ids + once_ids) # remove stop words and words that appear only
once
dictionary.compactify() # remove gaps in id sequence after words that were removed

dictionary.save(os.path.join('data', conference, 'dictionary.dict')) # store the dictionary

#Save vocabulary
vocFile = open(os.path.join('data', conference, 'vocabulary.dat'),'w')
for word in dictionary.values():
```

```
    vocFile.write(word+'\n')

vocFile.close()
print('Dictionary and vocabulary saved')
```

So, our dictionary is finished and ready to be implemented in our vectorization. We will introduce a class to prevent storing the words of each document in RAM. Instead, we will analyze each document words separately.

```
class MyCorpus(object):
        def __iter__(self):
        for line in tweets:
                # assume there's one document per line, tokens separated by whitespace
                yield dictionary.doc2bow(line.lower().split())
```

And we will create an instance of the class, containing the corpus

```
corpus_memory_friendly = MyCorpus()
```

Now that our corpus is ready and each document will be vectorized when we call each line, we can start writing our mult.dat file to use it in **DTM**, but we have a little problem: the corpus is represented by a list of lists of tuples, like this:

```
[[(0, 1), (1, 1), (2, 1)], [(0, 1), (3, 1), (4, 1), (5, 1), (6, 1), (7, 1)], … ]
```

And we have to write in the specified format:

```
unique_word_count index1:count1 index2:count2 ... indexn:counnt
```

Now, we can write to the mult.dat file

```
multFile = open(os.path.join('data','WWW','mult.dat'),'w')

for vector in corpus_memory_friendly: # load one vector into memory at a time
    multFile.write(str(len(vector)) + ' ')
    for (wordID, weigth) in vector:
        multFile.write(str(wordID) + ':' + str(weigth) + ' ')
    multFile.write('\n')

multFile.close()
print('mult file saved')
```

3. Back to DTM Blei's executable

Once we have the corpus ready (`mult.dat` created in the previous step), we can load it into Blei's executable to perform DTM topic modelling. To do this, we can call this from comand like:

```
$dtm-win64.exe    ./main    --ntopics=3    --mode=fit    --rng_seed=0    --initialize_lda=true
--corpus_prefix=data/WWW/    --outname=data/WWW/output    --top_chain_var=0.9    --alpha=0.01
--lda_sequence_min_iter=6 --lda_sequence_max_iter=20 --lda_max_em_iter=20
```

In the example, data/WWW/ is just the name of a folder but you can change it to something else; but don't miss having the file mult.dat in the folder specificed in `--corpus_prefix`.

It is important to notice that there is a parameter named "*alpha*". This parameter allows to force to obtain similar (when is close to 0) or not-similar (close to 1) topics over the timespan. You have to decide how similar do you want the topics over the years. You can try with different values to observe what happen. In our experience, setting this parameter too high (such as 0.9) returns topics completely unrelated to each other.
In the parameter *ntopics*, you should put the number of topics that you want to export to the output. In this case, we will get five topics.
After executing this command, we are ready to proceed interpreting the output.

4. Interpreting Blei's DTM Output data

After executing DTM, you'll have a list of files. Some of this files are:

- topic-xxx-var-e-log-prob.dat: this file presents the distribution of words for the topic *xxx* for each period analyzed. This values are in logarithm.
- gam.dat : this file stores the parameters of the Dirichlet variational (gammas) for each document.

For interpreting the data, you might generate a csv file with the probabilities of each topic per year by getting the values from the topic-xxx-var-e-log-prob.dat file. Remember to apply exp() to the values. You also might want to generate a csv with the topic mixtures for each document. To do this, you have to analyze the gam.dat file. Each set of *n lines* followed represent the gammas for each topic in each document. By dividing the *i line* (i topic) by the sum of the *n lines* will give you the topic proportion.
To make this task easier, we will use a python package to interpret and visualize the output. You are free to do this with your own method.

4.1 Using tethne to Interpret and Visualize DTM output

To interpret this output data, we will use a python package to visualize corpus data named Tethne [2]. You might use another package if you want. With this tool, we will be able to

export the topics' evolution over the timespan and the topics' most common words into simple txt files.

To do this, we use the tethne from_gerrish tool to load the corpus, and then we can generate the outputs that tethne allows us to do.
To import the corpus into tethne, just do:

```
#Import to tethne
dtm = tethne.model.corpus.dtmmodel.from_gerrish('data/' + conference + '/output/','data/' +
conference + '/metadata.dat','data/' + conference + '/vocabulary.dat')
```

Then, we can generate the exports showed in tethne documentation. As an example, here we will print the topics' most 10 common words for each topic (in our case 5 topics) and for each year (in our case 5 years) with the probability of appearing in a document, so then we can plot a topic evolution.

```
for topic_i in range(5):
  arr = dtm.topic_evolution(topic_i,10)
  for key in arr[1].keys():
    for year_i in range(5):
      print([conference, topic_i , key, (year_i + 2009), arr[1][key][year_i]])
```

Using the output from the code above, we can generate using R some interesting plots, like the one below, that shows three topic evolution over the timespan in one conference. We can observe that the conference was very stable in its topics over the five years.

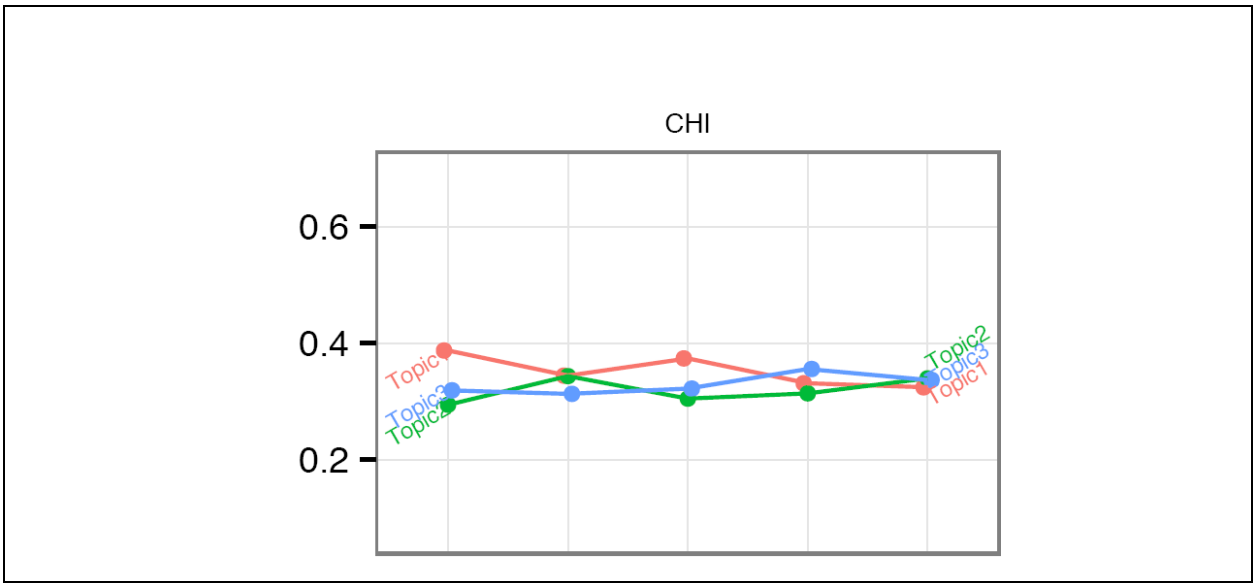
4.2 Plotting with R

Having the data into this format, you can plot this into R:

```
library(ggplot2)
library(gridExtra)
library(directlabels)

> str(dfc)
'data.frame':  240 obs. of  8 variables:
 $ year  : Factor w/ 5 levels "2009","2010",..: 1 1 1 1 1 1 1 1 1 1 ...
 $ confID: Factor w/ 16 levels "CHI","CIKM","ECTEL",..: 1 1 1 2 2 2 3 3 3 4 ...
 $ topic : Factor w/ 3 levels "Topic1","Topic2",..: 1 2 3 1 2 3 1 2 3 1 ...
 $ N     : num  2083 2083 2083 37 37 ...
 $ value : num  0.387 0.294 0.319 0.28 0.391 ...
 $ sd    : num  0.44 0.412 0.42 0.425 0.447 ...
```

```
 $ se    : num  0.00964 0.00903 0.0092 0.06985 0.07347 ...
 $ ci    : num  0.0189 0.0177 0.018 0.1417 0.149 ..

pd <- position_dodge(.1)

ggplot(dfc, aes(x=year, y=value, colour=topic, group=topic)) + geom_point(position=pd) +
geom_dl(aes(label=topic),size=2.5,list("first.qp",cex = 0.5, rot = 30))  +
geom_dl(aes(label=topic),size=2.5,list("last.qp",cex = 0.5, rot = 30)) +
geom_smooth(aes(group=topic, ymin = value-se, ymax = value+se)) + facet_wrap(~confID,ncol=4)
+ theme_bw()  +
  theme( legend.position="bottom",  panel.grid.minor = element_blank()) + theme(axis.title.x
= element_blank(), axis.title.y = element_blank(), axis.text.x = element_text(angle = 30,
vjust = 0.1, hjust=0.1,size=5), strip.background=element_blank(),
strip.text=element_text(size=7), legend.text=element_text(size=4),
legend.title=element_text(size=4),panel.margin = unit(0,"null"), plot.margin =
rep(unit(0.1,"cm"),4) ) + labs(x=NULL)
```

will look like this:

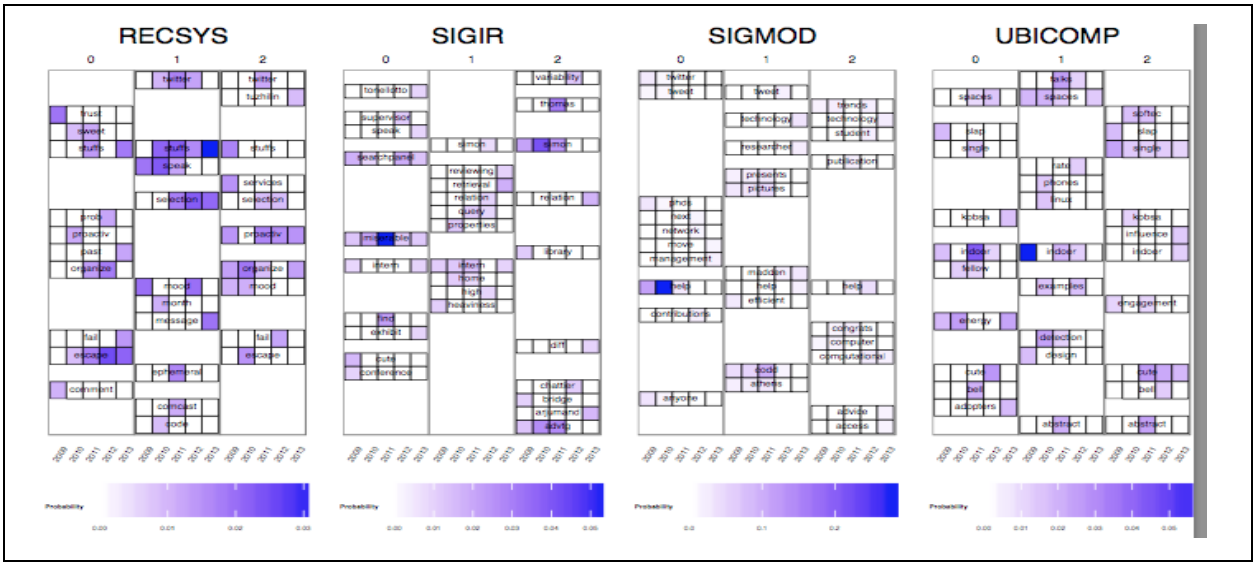


or even this:

```
dtm.df <- read.csv(file="OutputDTM-HT4.csv",sep=",");

dtm.df$TopicID = as.factor(dtm.df$TopicID)
dtm.df$Year = as.factor(dtm.df$Year)

library(ggplot2)
library(gridExtra)

# 'data.frame':  125 obs. of  5 variables:
#  $ Conference : Factor w/ 1 level "ACMMM": 1 1 1 1 1 1 1 1 1 1 ...
# $ TopicID    : int  0 0 0 0 0 0 0 0 0 0 ...
# $ Word       : Factor w/ 23 levels "ahead","award",..: 4 4 4 4 4 7 7 7 7 7 ...
```

```
# $ Year       : int  2009 2010 2011 2012 2013 2009 2010 2011 2012 2013 ...
# $ Probability: num  0 0 0.0142 0 0 ...

# ===
conf.dtm.df <- dtm.df[dtm.df$Conference == "CHI", ];

gchi <- ggplot(data=conf.dtm.df,aes(x=Year,y=Word))
gchi <- gchi + geom_tile(aes(fill = Probability), colour="black",stat = "identity") +
scale_fill_gradient(low="white", high="blue") +
ggtitle(paste("",as.character(conf.dtm.df$Conference)," ") )  +  facet_wrap(~ TopicID,
scales="free_y", ncol=5) + geom_text(data=conf.dtm.df[conf.dtm.df$Year == 2011,],
aes(label=Word), size=4, vjust=0.25) + theme_bw() + theme(panel.grid.major =
element_blank(), legend.position="bottom",  panel.grid.minor = element_blank()) +
theme(axis.ticks = element_blank(), axis.text.y = element_blank(), axis.title.x =
element_blank(), axis.title.y = element_blank(), axis.text.x = element_text(angle = 60,
vjust = 0.1, hjust=0.1,size=5), strip.background=element_blank(),
strip.text=element_text(size=7), legend.text=element_text(size=4),
legend.title=element_text(size=4),panel.margin = unit(0,"null"), plot.margin =
rep(unit(0.1,"cm"),4) , legend.margin=unit(-0.6,"cm"), legend.key.height = unit(0.4, "cm"))
+ labs(x=NULL)

#visualze
gchi
```

.. that will look like this



References

[1] Wen, X., Lin, Y., Trattner, C. and Parra, D.: Twitter in Academic Conferences: Usage, Networking and Participation over Time, In Proceedings of the ACM 2014 International

Conference in Hypertext and Social Media (Hypertext 2014), ACM, New York, USA, 2014. PDF

[2] Tethne package for python - http://diging.github.io/tethne/

[3] Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent dirichlet allocation. *the Journal of machine Learning research*, *3*, 993-1022.

[4] Blei, D. M., & Lafferty, J. D. (2006, June). Dynamic topic models. In*Proceedings of the 23rd international conference on Machine learning* (pp. 113-120). ACM.