COSC 2671 Social Media and Network Analytics

Lab 5

Topic Model

Learning outcomes:

- Learn how to implement topic modelling (LDA) for real social media data (Twitter)
- Visualise topic models
- Learn how to use word clouds

Requirements:

- A PC with Internet connection and Python installed (preference is version 3.X, but 2.7 is also okay, but unfortunately, we don't have resources to provide support if there are version incompatibility issues)

Resources:

- This lab worksheet (available on Canvas)
- Associated code in lab05Code.zip (available on Canvas)

Python Packages Required:

- tweepy
- nltk
- numpy
- sklearn
- pyLDAvis
- wordcloud

Lab Introduction

In this lab, we will apply topic modelling on Tweets. We will also learn how to visualise the discovered topics via the packages pyLDAvis and wordcloud.

Data & Packages

This week, we will use your timeline (or rmit_csit if you don't have any tweets) and see what topics are discussed. The code is available in the usual accompanying zip file, lab05Code.zip. In addition, before we start install the following packages using Anaconda (or pip):

- pyLDAvis
- wordcloud

These aren't available on the default environment screen in the Anaconda navigator. Similar to week 3 lab, we will need to use the terminal from the navigator to install. After opening the terminal, type the following to install the packages.

```
$ conda install -c conda-forge pyldavis
$ conda install -c conda-forge wordcloud
```

You'll also need your twitterClient.py from week 3, with the appropriate access keys etc. Copy that script file to your working directory for this lab.

Topic Model

Topic models allow us, among many things, to explore and summarise the topics and ideas being discussed within a set of documents, whether these are tweets or question and answer posts. There are a few great packages available in Python for topic modelling, including gensim, but to avoid learning about yet another package, we will use Scikit Learn, which also has good topic modelling implementations. In fact, it only takes a few lines of code to perform topic modelling in Scikit Learn.

Open the twitterTopicModel.pynb in the zip file of this week's lab. Most of it is like the code for processing twitter, but there are two parts worth highlighting.

The first is the 8th cell, and located after the cell with the LDA parameter settings:

```
tfVectorizer = CountVectorizer(max_df=0.95, min_df=2,
      max_features=no_features, stop_words='english')
tf = tfVectorizer.fit_transform(lTweets)
```

which uses the CountVectorizer from Scikit learn to construct a document-term/word from the set of tweets. We have seen the TfidfVectorizer() for doing something similar, but with an addition TF-IDF step. This one just counts without TF-IDF reweighting.

The second is to construct and run an LDA model (note this is one line in Python, broken up for better readability):

```
ldaModel =
LatentDirichletAllocation(n_components=args.topicNum,
      max_iter=10, learning_method='online').fit(tf)
```

This is it – all the code needed to run LDA in Scikit Learn! Essentially a few lines of code. There are many parameters (see http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.LatentDirichletAllocation.html), but the important ones are:

- n_components: specifies the number of topics

The other ones are less important, and related to how we train the LDA:

- max_iter: remember LDA is an iterative training process, this specifies how many iterations to use before stopping.
- learning_method: this is the training method to use, online means we feed the training method with instance by instance, as opposed to a batch

Run the notebook.

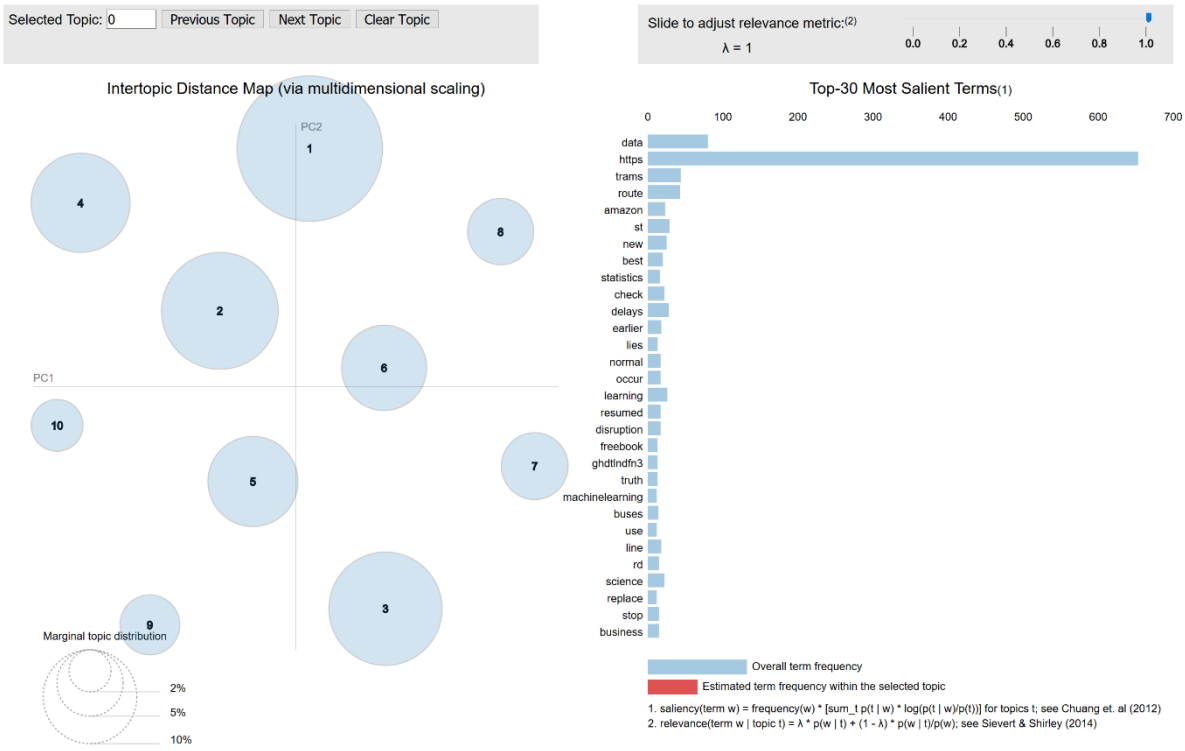
Exercise 1:
The notebook will print out the topics discovered, and the number of top k/most probable words associated with each topic. Examine these, do they make sense?

Using your newfound knowledge of the arguments of this script (7th cell), add an extra argument that changes the number of words to print per topic to 20.

Visualisation

pyLDAvis

We are using two types of visualisation. The first is based on a topic model visualisation package/library, called pyLDAvis (if you are familiar with R and topic modelling, you may have heard of the R package LDAvis – this is essentially the Python port of that). What pyLDAvis allows us to do is to visualise the topics, and word distributions. Below is an output from pyLDAvis.



Exercise 2:
To achieve this, again only takes 3 lines of code. Add the following to the 11th cell, with comment "pyLDAvis"

```
panel = pyLDAvis.sklearn.prepare(lda, tf, tfVectorizer,
                mds='tsne')
pyLDAvis.show(panel)
```

Examine pyLDAvis documentation for details about arguments, http://pyldavis.readthedocs.io/en/latest/modules/API.html .

Add the import at the top of the notebook (3rd cell), in the area that has the other package imports.

```
import pyLDAvis.sklearn
```

Rerun the script and see the topic models in their full glory (as an example see the figure above). Play with the display, with the left panel representing topics and their distances to other topics. Size of circle measures how probable that topic is, relative to the others. The right panel displays the corpus wide and topic specific word distributions. In the top right, is the λ parameter, which controls which words are displayed in the right hand figure:

- If λ = 0, then ranked by P(word | topic)
- If λ = 1, then ranked by P(word | topic) / P(word)
- If between 0 and 1, then ranked by (1- λ) * P(word|topic) + λ * P(word|topic) / P(word)

Wordcloud

The second visualisation is using word clouds to visualise the most important words for each topic. Instead of using a parameter to control the number of words/terms to display (see the current code), word clouds display words and their size according to their frequency or probability in that topic, with more frequent/more probable words will appear larger.

To implement word clouds, we will be using the wordcloud package. This is package makes it relatively simple to construct nice looking work clouds. An example of what it can do, with pyplot, is the following for 10 topics:

Import the package by (add to 3rd cell):

```
from wordcloud import WordCloud
```

The wordcloud package has several ways we can pass the words to it, the way we used is the one closest to the output we have from our Scikit Learn LDA model.

Recall the following:

```
ldaModel =
LatentDirichletAllocation(n_components=args.topicNum,
      max_iter=10, learning_method='online').fit(tf)
```

The returned model (ldaModel) has a few attributes, but the most important one is 'component_'. It is a list of topics, and for each topic the weighting of each word/term in the topic. As an example of a 2 topic decomposition of documents that are represented by 4 words:

```
[
      [0.117, 0.115, 0.117, 2.342],
      [1.834, 0.113, 0.118, 1.517]
]
```

Generally speaking, the values are relative within each topic. Word clouds require probabilities or frequencies. To make the topics and words comparable across topics, we normalise each topic (each row) such that we obtain probabilities. We do that by dividing each row/topic by its sum:

```
normalisedComponents = model.components_ /
      model.components_.sum(axis=1)[:, numpy.newaxis]
```

After normalising the example above, we will obtain the following:

```
[
      [
            [0.160, 0.157, 0.160, 0.523],
            [0.512, 0.032, 0.032, 0.424]
      ]
]
```

See how each row sums to 1 now?

Now we are ready to display the word clouds for each topic.

First, let's examine how to display the word cloud of one topic. We need to do a few things:

1. Word cloud requires associating the actual word with the output of ldaModel.components_, which only has the weights associated with each word. We will use featureNames.
2. Construct the word cloud, using the package wordcloud.

3. Display the word cloud, using pyplot.

You'll implement all of these within **displayWordcloud**(model, featureNames) function, which is called from the last cell with a number of arguments.

- The model is the ldaModel constructed by Scikit Learn and contains the important component_.
- featureNames is a list of strings, where each string is a word in the document and ldaModel. The index in featureNames has a one to one corresponds with the word distributions of ldaModel.components_, i.e., featureNames[i] and ldaModel.components_[0][i] refer to the same word. We use featureNames to recover what was the word each element in ldaModel.components_ refer to. featureNames is obtained from the CountVectorizer.

Now we go through each of the wordcloud steps.
Step 1: To construct the word cloud, we first need to normalise the weights (see above) then construct a dictionary of tuples (word, word probability). To do this, we can use the following code (one line in python, but I split into two to make it easier to read):

```
lWordProb = {featureNames[i] : wordProb
     for i,wordProb in enumerate(lTopicDist)}
```

where lTopicDist is one of the normalised rows of "normalisedComponents", or the probabilities each word appears in a topic. lWordProb is generated from dictionary comprehension, same as list comprehension but for dictionaries. For those not familiar with enumerate, have a look at the link http://book.pythontips.com/en/latest/enumerate.html . For list and dictionary comprehension, have a look at http://www.pythonforbeginners.com/basics/list-comprehensions-in-python .
Please understand this line fully before moving on, as in Python, you will see list and dictionary comprehension a lot and something very useful to know – ask your labmates or friendly demonstrator if need help.

Step 2: To construct the word cloud, use the following commands:

```
wordcloud = WordCloud(background_color='black')
wordcloud.fit_words(frequencies=lWordProb)
```

WordCloud is a class in the wordcloud package, that does the heavy lifting for us. First line constructs the object, with background colour been black. You can change this and other parameters, see https://amueller.github.io/word_cloud/generated/wordcloud.WordCloud.html#wordcloud.WordCloud for more information.
Second line uses the dictionary of (words, probabilities) for the topic to construct the actual word cloud. The argument in fit_words() is that dictionary, which we constructed in step 1. Almost there, now to display the actual word cloud.

Step 3: Final step is to display the cloud. We use matplotlib.pyplot to do so (recall we encountered this when displaying the term frequencies histogram in week 2 tute-lab), consider the following lines:

```
plt.title('Topic %d:' % (topicId+1))
```

```
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
```

plt.title() sets the title of the wordcloud plot. plt.imshow() displays the wordcloud, and we use 'bilinear' interpolation to layout the words (see https://matplotlib.org/gallery/images_contours_and_fields/interpolation_methods.html for what the different interpolation approaches do). plt.axis() is used to turn off the axis (typically a plot or histogram will have axes, but for a wordcloud they have no meaning hence we don't show them). plt.show() displays the actual plot.

Together, all these steps are as follows:

```
lWordProb = {featureNames[i] : wordProb for i,wordProb in
        enumerate(lTopicDist)}
wordcloud = WordCloud(background_color='black')
wordcloud.fit_words(frequencies=lWordProb)
plt.subplot(plotRowNum, plotColNum, topicId+1)
plt.title('Topic %d:' % (topicId+1))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show(block=True)
```

Exercise 3:
Using the above code and the normalised component, we display the first topic, i.e., the one stored in normalisedComponent[0]. Hint: lTopicDist above represents the distribution of a topic, so does normalisedComponent[0], can you use this information to help you?

We want to display multiple word clouds, one per topic. One of pyplot's functionality is to display multiple plots within the same figure, similar to par() or layout() in R (for those that are familiar with R, otherwise ignore). In pyplot, we can do this via "subplot()". The following code displays 2 plots, in a 2 by 2 grid:

```
import matplotlib.pyplot as plt

plt.subplot(2,2,1)
plt.plot(…) # plot subplot 1
plt.subplot(2,2,2)
plt.plot(…)# plot subplot 2
plt.subplot(2,2,3)
plt.plot(…)# plot subplot 3
plt.subplot(2,2,4)
plt.plot(…)# plot subplot 4
```

Exercise 4 :
Modify your code for displaying the wordcloud of one topic to displaying word clouds for each topic extracted by the lda model as its own subplot. Use pyplot.subplot to help you. To loop through the topics, consider the following code:

```
for topicId, lTopicDist in enumerate(normalisedComponents):
```

```
# TODO: You add code from plt.subplot() and the code to display one word cloud to
#       display one word cloud per topic.  Hint: Remember to read up what enumerate
#       does!
```

Extension:
Looking at the word clouds, the tweet quite a few random noisy words. To improve the word cloud, perform data-processing to clean the data.