# COSC 2671 Social Media and Network Analytics

# Lab 6

# Evaluating Topic Model

Learning outcomes:

- Evaluate Topic models
- Compute sentiment for each topic across time

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 lab06Code.zip (available on Canvas)

Python Packages Required:

- gensim
- pandas
- sklearn
- seaborn
- adjustText

## Lab Introduction

Last week, we studied how to use Scikit learn to perform topic modelling and visualise the topics using pyLDAvis and wordclouds. This provided a visual method to evaluate the obtained topics. In this lab, we will explore additional approaches to evaluate topic models. We learn how to use likelihood scores to select the number of topics and use word2vec to evaluate our topics.

## Data & Packages

This week, we will use 20 newsgroup, a dataset that is about 20 years old but good to demonstrate another type of social media data (newsgroups are predecessors of online forums) without spending too much time on APIs. Before we start, check and/or install the following packages using Anaconda (or pip):

- gensim
- pandas
- seaborn
- adjustText

Gensim, pandas and seaborn should be installed with your default Anaconda install. Only adjustText needs installation via the terminal (similar to pyLDAvis last week):

```
conda install -c phlya adjusttext
```

# Evaluating Topic Model

Recall when we were using Scikit learn to perform topic modelling last lab, one of the parameters that needs to be specified is the number of topics. This isn't unique to Scikit learn implementation, generally we need to specify the number of topics when performing topic modelling. As discussed in tutorials, there are a number of ways we can estimate this, but we will demonstrate an approach based on using the likelihood score. We also use another embedding and visualisation to help in examining the words in each topic and their relationships with each other.

## Loading of datasets and pre-processing

In the provided notebook, topicEvaluation.ipynb, cells 1 to 7 is similar to previous weeks, loading a dataset then do some pre-processing. For this week, we will be using the 20 newsgroups dataset. The dataset is a series of posts from 20 different newsgroups (if you don't know what these are, think of them as bulletin boards that are predecessors of online discussion forums and reddit). Although dated, the dataset is available in many formats, so we don't need to focus too much about obtaining and processing the unstructured text. We are going to use the Scikit learn pre-installed version (see 3rd cell):

```
from sklearn.datasets import fetch_20newsgroups
newsGroups = fetch_20newsgroups(remove=('headers',
'footers', 'quotes'))
```

The remove=('headers', 'footers', 'quotes') will not load the headers, footers and quotes within the newsgroup posts. Run this and the next two cells (4th and 5th) to see the content that are loaded.

The 6th and 7th cell are similar to last week's lab, they do some pre-processing and for this week, we only include words that are in a loaded builtin dictionary. This is to avoid unknown words. But recall this can have a significant effect so something worth exploring a bit for your own projects.

## Evaluating number of topics

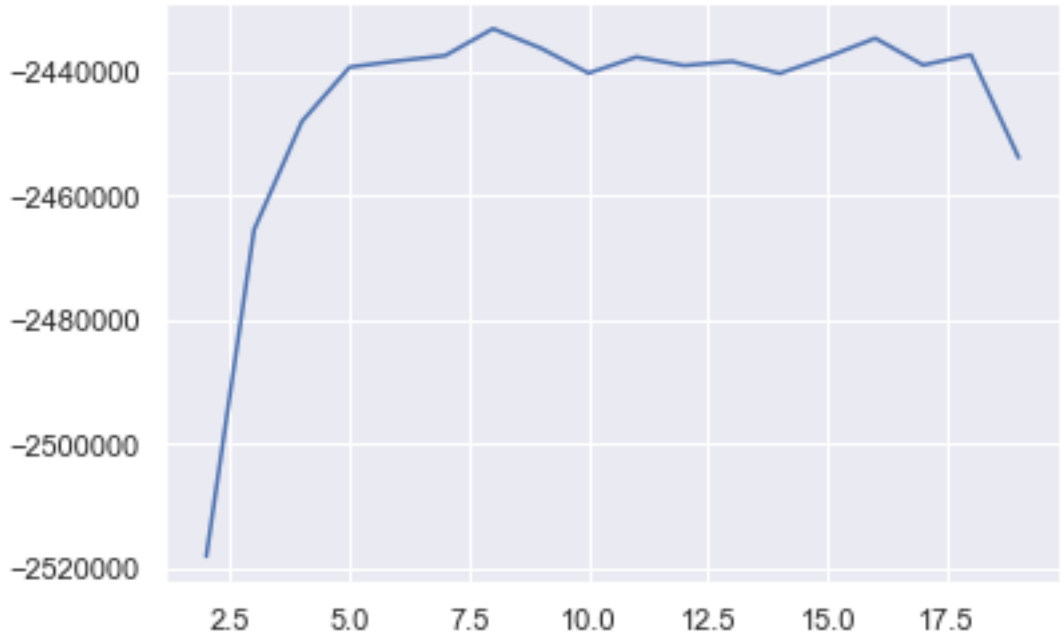
The following code (the 8th cell of notebook), constructs a topic model for a range of topic number. There are several possibilities to measure the "appropriate" number of topics – for this lab, we use the (negative) likelihood score. Think of this as measuring the quality of fit of model to the data, and the smaller in absolute terms, the better the fit is.

```
# try from topicNumMin to topicNumMax
topicNumMin = 2
topicNumMax = 18
```

```
lTopicLikelihood = []
for tn in range(topicNumMin, topicNumMax):
     ldaModelTry = LatentDirichletAllocation(n_components
     =tn, max_iter=10, learning_method='online').fit(tf)
      lTopicLikelihood.append(ldaModelTry.score(tf))

# plot as line chart
plt.plot(range(topicNumMin, topicNumMax),
lTopicLikelihood)
plt.show()
```

Run the code/cell, and you should obtain a plot similar to the following:




The x-axes is the number of topics and y-axis is the likelihood.

We generally want a larger likelihood with the minimal number of topics. Hence for this plot, it appears 8 topics is roughly a good selection and used for the rest of the lab (you might find it is slightly different for you, that's okay, as topic modelling is an unsupervised, stochastic method so not guaranteed to get the same result, particularly if random initialisation is used).

**Exercise 1:**
Update the above script to use perplexity score instead of the likelihood score (see https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.LatentDirichletAllocation.html#sklearn.decomposition.LatentDirichletAllocation.perplexity )
Does it say the same thing as the likelihood?

# Using Word Embeddings to Visualise Topics

Last week we used pyLDAvis and wordclouds to visualise the topics. Although it gave us an indication about the topics, it doesn't really say whether the words are semantically related to each other, which can be helpful in evaluating the topics. This week, we will use a recent development that allows words to be visualised as points in a 2D space – words that are similar will be plotted close to each other, while words that are dissimilar will be far.

Similar to last week, cells 9-11 constructs the topic model, then extracts out the top probability words for each topic. We use the top probability words for each topic to avoid visualising too many words, which makes it difficult to read the subsequent plots:

```
lTopTopics = []
# print out the topic distributions
for topicId, lTopicDist in
enumerate(ldaModel.components_):
     lTopTopics.append([tfFeatureNames[i] for i in
     lTopicDist.argsort()[:-wordNumToDisplay - 1:-1]])
```

Start from the 12th cell, we begin to see code relating to word embeddings. A word embedding is essentially a mapping of words to a vector of numbers. These vectors are multi-dimensional, so we are essentially mapping words to a multi-dimensional vector (recall from your geometry, a multi-dimensional vector can be considered as part of a (vector) space). The vectors are meant to be close to each other in this space if they are semantically similar to each other. We will use this idea to visualise the words in each topic to see if they are close to each other in the embedding space.

First, we need an embedding. Luckily for us there are some embeddings built in already. We will use a small one (to avoid excessive downloading), that is available for genism (recall this is another popular package for topic modelling):

```
import gensim.downloader as api

# show some info about word embedding
info = api.info('glove-wiki-gigaword-50')
print(info)

# load it
word2vecModel = api.load("glove-wiki-gigaword-50")
```

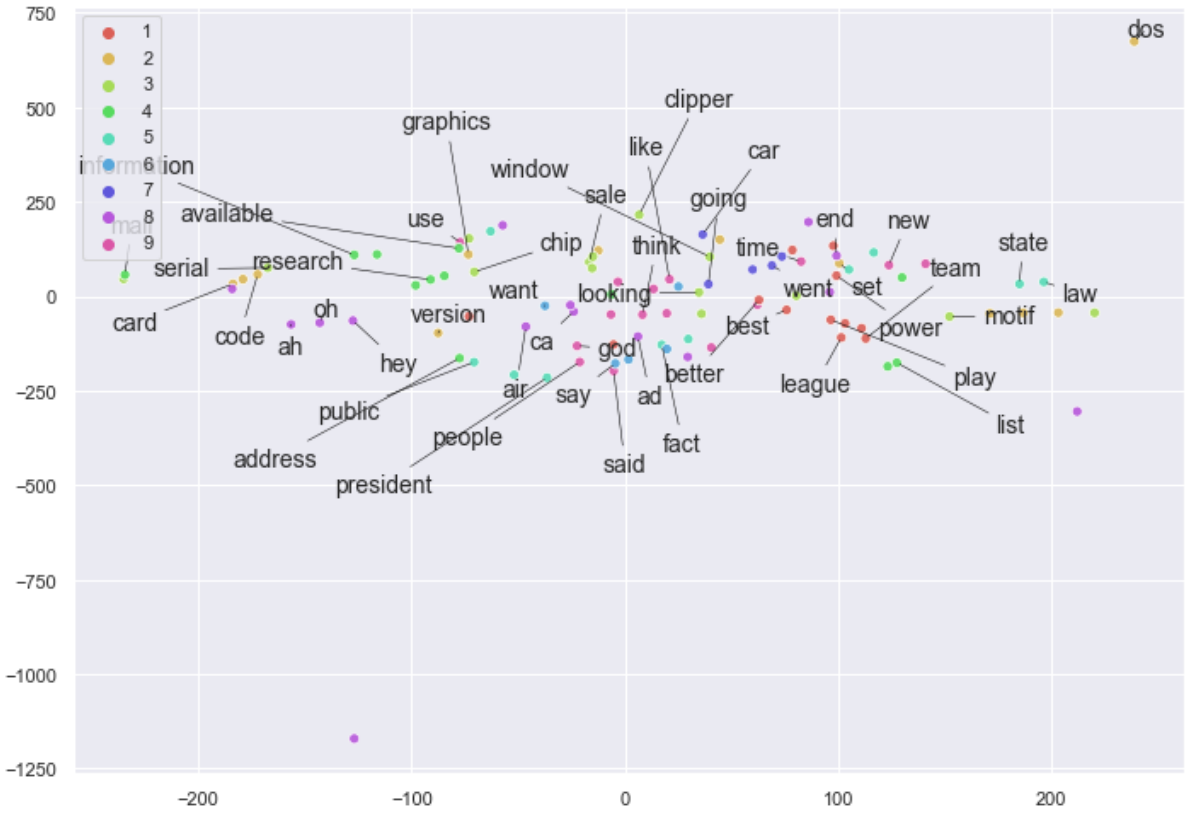
The one we load is called glove and represent words by a 50 dimension vector. Note we likely get better results if we use a more appropriate embedding, this is something we encourage you to explore later.

We could visualise a 50 dimension vector, but that would be difficult to read – we are not equipped to visualise more than 3 dimensions.

Hence the next 2 cells first constructs data frames in pandas (if not familiar with pandas and data frames, think of these as arrays), then use a projection algorithm called t-SNE that projects/reduces higher dimension data to a lower number of dimensions (2 in our case to make it easier to visualise). t-SNE needs data frames as input hence we have to introduce

this, but in general many packages uses pandas data frames, so something worth familiarising, not necessarily for this lab, but for the future.

One more package before visualisation. The last cell uses seaborn, a great visualisation package, to visualise the words in a 2D space. It performs a scatterplot, but uses the adjustText package to add text to the scatterplot. The adjustText package adds the words to the scatterplot. Run the cell – you should obtain a visualisation like the following:




**Exercise 2**

Run the code. You may see something slightly different as t-SNE is a stochastic method. Notice there are words like ‘dos’ in the corner of the space. Why do you think this is so? What other interpretations can you make from the illustration? Do you think the topics are coherent? Does t-SNE have an effect (try changing some of its parameters)? Discuss these with your lab demonstrator.