
Topic 05: Topic Models

Topic Models

Topic Modelling is a process to automatically identify topics present in a collection of documents. It is a useful tool for helping to organize, search and understand vast amounts of information. Given the right application, topic modelling may allow us to discover hidden themes in collections, annotate documents by the themes we discover and apply automation to categorisation. Unlike many of the other approaches in NLP, topic modelling is strictly unsupervised.

Theory and Terminology

Before we continue talking about topic modelling, we should understand what the theoretical basis underlying the approach. As with other NLP tasks, we assume we have a collection of documents called a ***corpus***. For purposes of simplification, we further assume this collection is in text (i.e. not speech) and all in the same language. A corpus contains one or more ***documents***, ($D = D_1, D_2, D_3, \ldots, D_n$), which is simply a series of tokens (words or terms). Each document has one or more writers, and the writers' intent is to convey one or more ***topics***, ($T = T_1, T_2, T_3, \ldots, T_p$), "repeating patterns of co-occurring" tokens, ($W = W_1, W_2, W_3, \ldots, W_m$), in a document. For example, if a writer's intent is to discuss healthcare, we may observe tokens like "health", "doctor", "patient" or "hospital." If a writer discusses a different topic — such as entertainment — our tokens would likely be different. While people are generally good at picking out topics from a document or corpus, our models are not.

Figure 1 shows Blei's image of topic modelling, and it's become so ubiquitous that we should all become familiar with it. One important to note from the description above and from Figure 1 is that a document may contain more than one topic.
The US Federal government has some examples of topic models at work as a kind of information extraction tool based on topics rather than terms. Hopefully, the site at https://federalreporter.nih.gov/ will still be up by the time we discuss it. Try *Text Search* (eg.

health) and select *Topics* for an interesting user interface. While topic modelling is not itself graphical, it often goes hand-in-hand with visuals.

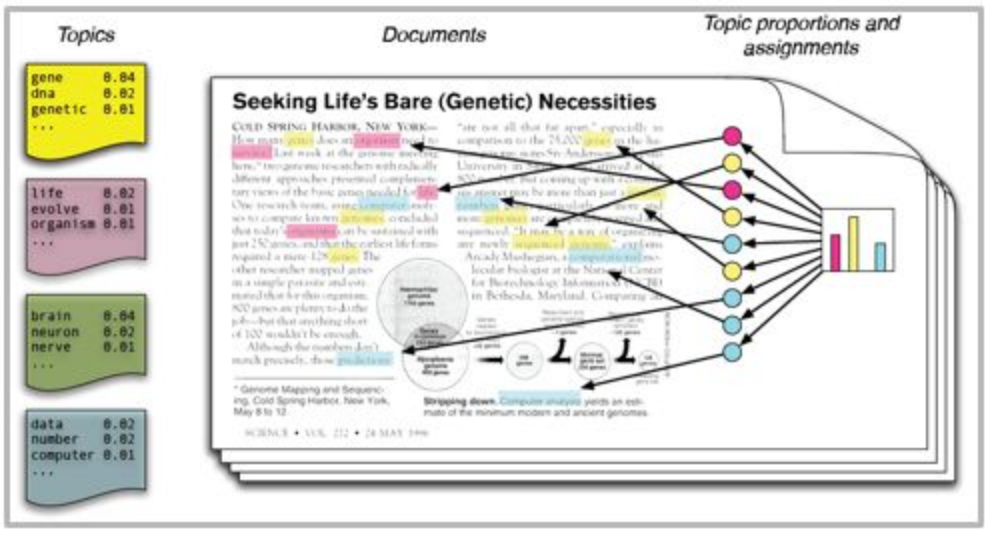

Figure 1: Conceptual Topic Modelling

LDA

There are many approaches for obtaining topics from a corpus. Term Frequency and Inverse Document Frequency (TF-IDF) and non-negative matrix factorization (NMF) were two popular approaches. Latent Dirichlet Allocation (LDA) is the most popular current topic modeling approach. Let's discuss LDA briefly.

LDA assumes documents are produced from a mixture of topics. Those topics then generate words based on the probability distribution of those words in the topic. Given a corpus, LDA backtracks and creates hypotheses about the topics which created each document. Specifically, we assume that we can represent a corpus as a matrix of D x W (Documents x Words) such as in Table 1, a document-term matrix, with each cell showing the number of times a word appears in a document.

	W_1	W_2	...	W_m
D_1	0	1	0	3
D_2	1	12	7	1
...	3	1	5	8

D_n	1	2	0	1

Table 1: Document-Term Matrix

Using LDA, we convert this document-term matrix to two lower-dimensional matrices: a document-topics matrix and topic-terms matrix. The goal is to answer a question: given an observed distribution of words in a corpus, what is the probability that a distribution of words in a document belongs to a particular topic. With:

- **k** = number of topics
- **n** = the number of documents
- **t** = number of types (“unique words”)

… the document-topics matrix has a rank of (n, k) and the topic-terms matrix has a rank of (k, t). These matrices are populated, initially with a boolean if a document contains a topic, as shown in Table 2, a sample document-topics matrix. (The topic-terms matrix is conceptually similar to this.)

	K_1	K_2	...	K_k
D_1	1	1	0	1
D_2	0	1	1	0
...	0	1	0	1
D_n	0	1	1	1

Table 2: Document-Term Matrix

The matrices are initially constructed by a random assignment of tokens to topics. This initial assignment could be a problem if not adjusted. For example, when a word like “lead” is used to construct this matrix, we do not know whether this word refers to the (lack of) leadership in the state of Michigan which caused the city of Flint to have a poisoned water supply or whether it refers to the substance used to poison the water supply — i.e. a verb or a noun. Since LDA does not use context in the initial construction of these matrices, disambiguation of this word is impossible.

Although word ambiguity could be a problem, LDA makes the assumption that a topic defines a probability distribution over a set of words (not a single word), so it iterates to find this distribution. For each iteration, a new topic, **k**, is assigned to each word, **w**, with probability **p** = p_1 x p_2, where:

- p_1 = the fraction of words in document d assigned to topic k.
- p_2 = the fraction of assignment to topic t over all documents from the word w.

Based on this, the each word in the topic-terms matrix is updated with a new topic with the probability p, which is the probability that a topic generated a term. The document-topics matrix

is updated based on this reassignment. These steps are repeated to convergence (documents do not switch topics) or until the maximum number of iterations has been achieved.

Practical Topic Modelling

Practically, topic models may require a lot of time to train and associate. The time is a function of the size of the corpus, the number of topics required and the iterations to find the number of topics. We'll start with a small corpus which should be able to give us results in seconds and ignore larger corpora such as wikipedia.

Let us practice topic modelling with two tools: TopicModelingTool, which is a graphical package on Mallet, and also with gensim. For this, let us first download the tiny corpus we used for information extraction. It contains two broad themes: children driving cars and the recent Korean presidential election.

Topic Modelling Tool

Mallet is an open-source topic modelling tool written by Andrew McCallum at the University of Massachusetts. While Mallet is used primarily as a topic modelling toolkit, it has other functions such as text classification and sequence tagging. There is a command-line interface, but since it is written in Java, we will not discuss its use (especially the APIs) directly in this course. You can read more about the Mallet project at the project's page: http://mallet.cs.umass.edu/topics.php.

A graphical user interface was created for Mallet by David Newmann and Arun Balagopalan and distributed as "TopicModelingTool," a rebranded version. We will apply this to our corpus in this way:

1) Navigate to https://code.google.com/archive/p/topic-modeling-tool/ > Downloads
2) Download TopicModelingTool.jar
3) Launch the TopicModelingTool.jar either by double-clicking on it or by typing on terminal: `java -jar /PATH_TO/TopicModelingTool.jar` — ensuring that /PATH_TO/ contains the path to the jar file.
4) Create an output directory, such as /PATH_TO/topic_model_output/.
5) Select Input File or Dir as the folder with the small corpus.
6) Set the number of topics to 2. For reference, we almost never apply topic modelling to such a small number of documents. A good number of topics depends on the number of documents, specifically:
    a) 1K - 10K = 10-40 topics
    b) 10K-100K = 50-200 topics
    c) 100K-1M = 250-1000 topics
7) Once you have the tool looking like Figure 2, click "Learn Topics."

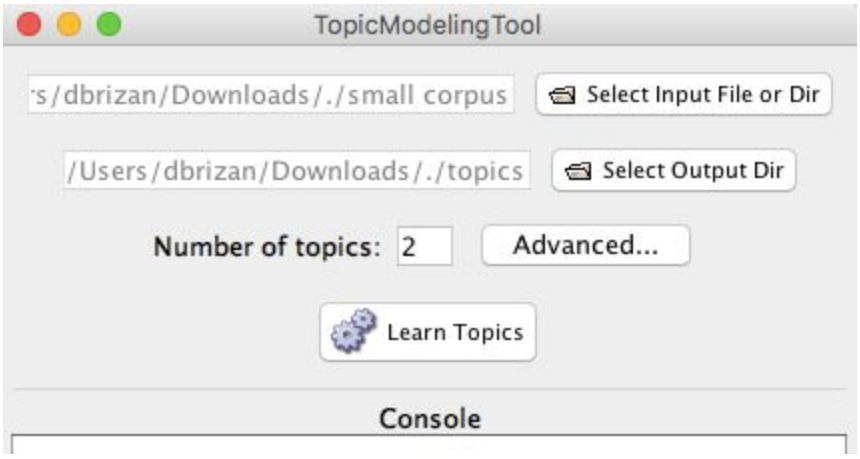

Figure 2: Setup for TopicModelingTool

There are two outputs of the TopicModelingTool: a set of CSV files useful for manipulation by a script or program and an HTML file useful for human consumption. Let's drag the HTML files into a browser and discuss the output.

In addition to the HTML output, there should be 3 CSV files:

- `DocsInTopics.csv` contains the topics, the documents sorted by topic and their ranks within each topic. We may notice that one of the documents (ir-doc01.txt) appears to be miscategorised. Why?
- `TopicsInDocs.csv` contains the document, the topics and the strength of the hypothesised correlation to the topics. We may notice that one of those documents (ir-doc04.txt) has a very weak association to the Korean election topic. Why?
- `Topics_Words.csv` contains the top words associated to the topics. Some of the words in one topic (christensen, diego, park, victory) may seem out of place. How were these topics selected?

As we discussed earlier, a document may contain more than one topic. As an exercise, let us:

1) Explore the [`Advanced…`] area, including the use of custom stopwords (the, and, …), case preservation, etc. (For example, what happens if we add "diego" to the topics?)
2) Investigate increasing the number of topics to 3 on the small corpus. What is the effect?
3) Apply the TopicModelingTool to some of the test data at https://code.google.com/archive/p/topic-modeling-tool/ > Downloads (economy file). Specifically:
    a) What is the effect of using 20 topics vs. 40 topics? (What is associated with "bush" for each of those numbers of topics?)
    b) Are the results stable when run more than once in a row?

---

gensim

Gensim is the python-based topic modelling tool written by Radim Rehurek from the Czech Republic. The name “gensim” comes from the phrase “generate similar.”

Download & install

In order to install gensim, we’ll need Python 2.7, sklearn and numpy. (This is probably not going to be a problem for any of us.) There are a couple ways to install gensim. The easy install will work well if your installation is reasonable standard. From a terminal, execute:

```
easy_install -U gensim
```

You may see a number of warnings. These can be ignored. If it executes correctly, you should see a line at the end similar to “`Finished processing dependencies for gensim`”. If you prefer pip or if the above does not work, you can try:

```
pip install --upgrade gensim
```

There may be other ways to install.

Once installed, we can read each document into a list of documents, such that documents = [doc1, doc2, …, docn]. We can use NLTK to clean, remove punctuation and and get the lemmas for tokens with something like the following:

```
def clean(doc):
    from nltk.corpus import stopwords
    from nltk.stem.wordnet import WordNetLemmatizer

    stop = set(stopwords.words('english'))
    exclude = set(string.punctuation)
    lemma = WordNetLemmatizer()

    stop_free = " ".join([i for i in doc.lower().split() if i not in
stop])
    punc_free = ''.join(ch for ch in stop_free if ch not in exclude)
    normalized = " ".join(lemma.lemmatize(word) for word in
punc_free.split())
    return normalized
```

With that complete, we can:

```
import gensim
from gensim import corpora
```

```
# Creating the term dictionary of our courpus, where every unique
term is assigned an index.
dictionary = corpora.Dictionary(doc_clean)

# Converting list of documents (corpus) into Document Term Matrix
using dictionary prepared above.
doc_term_matrix = [dictionary.doc2bow(doc) for doc in doc_clean]

# Creating the object for LDA model using gensim library
Lda = gensim.models.ldamodel.LdaModel

# Running and Trainign LDA model on the document term matrix.
ldamodel = Lda(doc_term_matrix, num_topics=3, id2word = dictionary,
passes=50)

print(ldamodel.print_topics(num_topics=3, num_words=3))
```