4

Topic Modeling

Present 1-2: The Title slide and the Learning Objectives slide. An overview of what we will achieve in this course.

Introduction

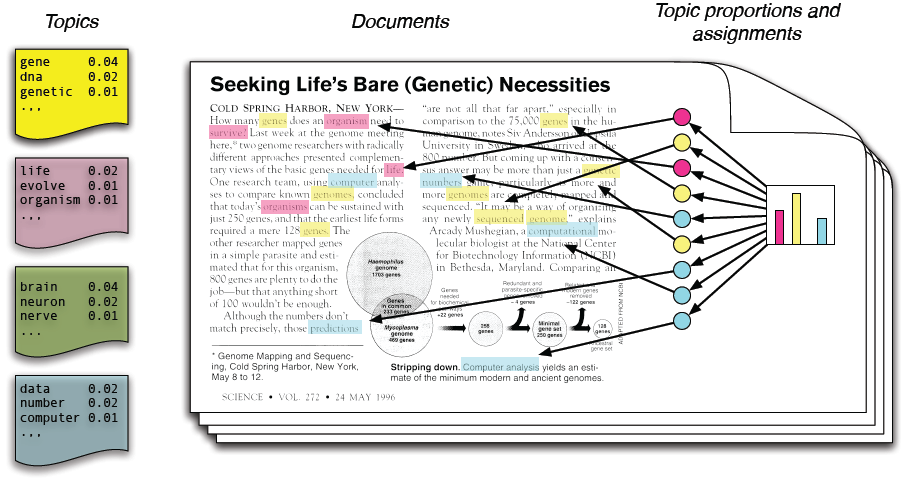
Topic modeling is a popular concept within natural language processing mainly because it is a simple way for us to capture the meanings of collection of documents. Essentially, we want to be able to understand at a high-level what topics are being spoken about in the texts in which we are interested. It is primarily a way of sorting or organizing documents into categories with the category being the topics into which you could sort the documents into.

When we speak of **topics** here, we are refering to the concepts being discussed in the documents – and these generally relate to what the authors were interested in discussing. If you do topic modeling on a subreddit for instance, you would expect the topics to relate to the what people tend to discuss on that subreddit – which could be many different things depending on the subreddit. In a way the topics that we find being discussed is dependent on what people are interested in.

Now a model is a way of capturing information about items in the real world in a software structure – bits in a computer. A topic model then captures information about the concepts contained in a set of texts. We can then use the topic model to do useful things – and the most useful of which is to organize the documents into categories – perhaps as a catalog on a website, or a way to present related documents to a user. But there may also be other interesting use cases that involve visually presenting the topics that we have found, such as when we want to learn more about some set of documents, such as a collection of emails.

Topic modelling is mostly done using **unsupervised learning** algorithms. In these types of machine learning algorithms, we have the algorithm detect the topics on its own. Unsupervised des not mean completely unguided- in fact we often give the algorithms some parameters with which to operate. Unsupervised just means that we do not present the algorithm with the sample texts plus the topics into which they fall. This helps us discover interesting topics that might exist and we also don’t have to spend manual effort labeling tests with topics. On the other hand, we now have to depend on the topics that the algorithm finds – which almost certainly would not necessarily align with human defined topics. That is one feature of unsupervised topic modeling – there is an extra step required to understand the topics that were found.

We will look at a few popular topic modeling algorithms including during this lesson. We will also look at techniques for getting text documents for topic modeling. Bear in mind here that when we speak of documents, we mean any coherent collection of words which could be as short as a tweet or as long as an article. It all depends on the project that we are interested in.



[This Photo](https://www.lucypark.kr/courses/2015-ba/text-mining.html) by Unknown Author is licensed under [CC BY](https://creativecommons.org/licenses/by/3.0/)

Figure 5.1: scalars, vectors, matrices and tensors

## Note:

## There is a subreddit for Topic Modelling

https://www.reddit.com/r/topicmodeling/

Discuss 1: Discuss ways in which raw text is messy and lacks the regular structure needed for computer algorithms to detect patterns in

## Topic Discovery

The main goal of topic modelling is to find a set of **n** topics into which to classify a set of documents. Underlying this goal is the assumption that there are implicit topics into which the documents can be classified. These topics are *implicit* because we do not know what they are beforehand and the topics are unnamed. We just generally assume that some documents are similar to each other and that we can sort into topics by how similar documents are to each other. So, topic modelling is just sorting documents by similarity.

The number of topics is usually a small – from **2** to **10** although there are some use cases in which you may want to have up to 100 (or even more) topics. Since it is the computer algorithm that discovers the topics and they do not need to directly correspond to a human selected topic are you generally can have an arbitrary number. In practice though, the number of topics should be orders of magnitude less than the number of documents. This helps the topic modelling algorithm in the sorting process as the more examples there are of documents the better the accuracy with which it can sort and place documents into categories.

The number of topics chosen depends on the documents and the objectives of the project. You may want to increase the number of topics if you have a large number of documents or if the documents fairly diverse. Conversely, if you are analyzing a narrow set of documents you may want to decrease the number of topics. This generally flows from your assumptions about the documents – if you think that inherently the document set might contain a large number of topics you should configure the algorithm to look for the similar number of topics. Essentially here you are guiding the algorithm to discover what is already inherent in the documents, and you could have already gotten a fair idea from sampling a few documents and seeing what types of topics they contain.

Topics Covered

* Loading and preprocessing text documents for topic modeling
* The main algorithms underlying topic modeling including Latent Dirichlet Allocation
* Visualizing topics using charts

Why Topic Modeling

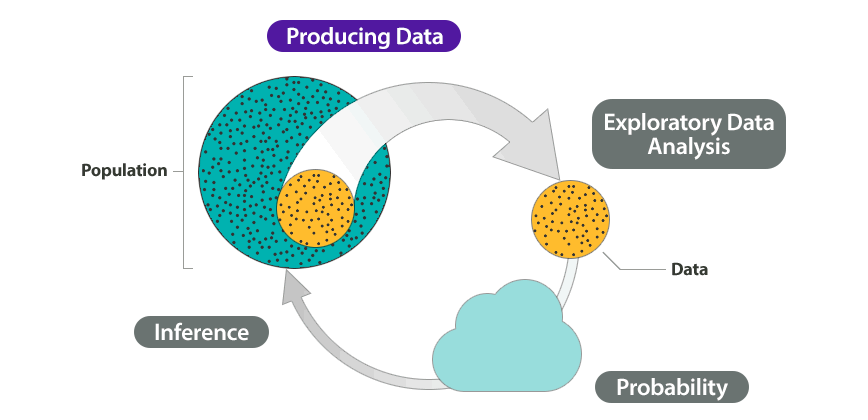
Topic modelling provides ways of automatically understanding, organizing, searching and sorting large amounts of data. The online world today is marked by large amounts of data that prove challenging for us to organize. For example, we have Wikipedia which is a collection of documents about facts about the world, Reddit which is a collection of articles that people are interested in. Reddit and Wikipedia each have their own way of organizing themselves but what if we wanted to organize a different way for a different purpose. Let’s say you wanted to analyze articles in a specific subreddit to

## Discovering Themes

Often when looking at a collection of text documents you would like to discover generally what themes or topics are contained in the documents. You could be doing this as a forensic analysis

## Exploratory Data Analysis

## It is recommended when starting a machine learning project to do exploratory data analysis at the beginning prior to performing any machine learning. This helps you learn about the data, and the probability distributions of the items within it. You will then be in a better position to choose the specific algorithms to use.



[This Photo](https://courses.lumenlearning.com/wmopen-concepts-statistics/chapter/why-it-matters-why-it-matters-types-of-statistical-studies-and-producing-data/) by Unknown Author is licensed under [CC BY](https://creativecommons.org/licenses/by/3.0/)

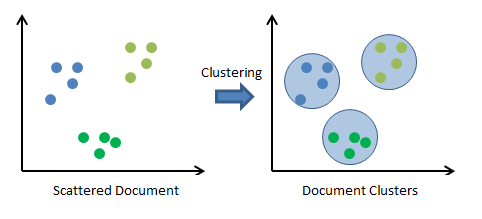
Figure 5.2: scalars, vectors, matrices and tensors

For natural language processing topic modelling is a great way to explore the text data and to see the ways in which it is naturally grouped. Topic modelling can help you understand if the data is balanced or skewed in any particular way. You can create charts of the resulting topic models perhaps for an Exploratory Data Analysis notebook to be presented to fellow data scientists for discussion on the approach to be taken.

## Soft Clustering

## Clustering is grouping a set of items in such as way that similar items are grouped together. The results of a clustering project are a set of cluster ids and the data items that they are associated with. You can for example cluster customers by features like buying income, location, age etc. and then use these clusters for predictions or for general data analysis.

Hard clustering associates each instance with only one cluster. This is useful in some cases but often documents tend to have more than one topic being discussed within them.



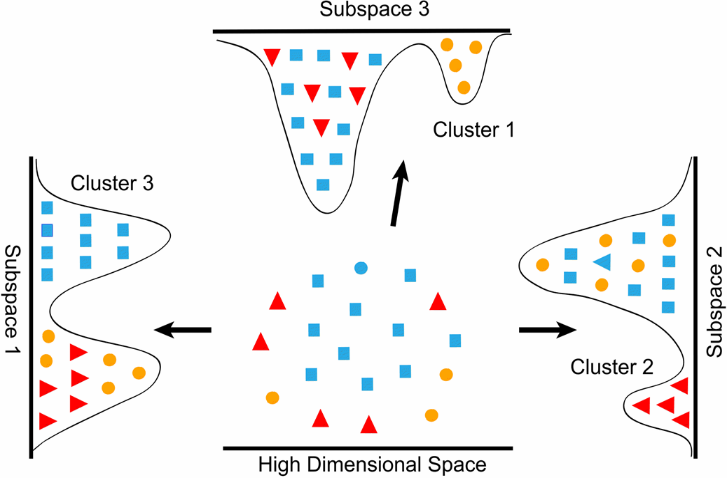
[This Photo](https://stats.stackexchange.com/questions/253926/what-are-the-x-and-y-axes-of-clustering-plots) by Unknown Author is licensed under [CC BY-SA](https://creativecommons.org/licenses/by-sa/3.0/)

Figure 5.3: scalars, vectors, matrices and tensors

Topic modelling allows for soft clustering – meaning that each document can be associated with more than one topic. This gives you a richer understanding of a document and a way to build search indexes that can have documents in multiple categories.

## Dimensionality Reduction

One of the challenges of doing machine learning is handling high dimensional data. A dimension is an axis on which your data varies. Documents are high dimensional because they contain many different words and each word can be considered a feature axis on which the document varies. If you look at the document for this lesson for example you can see that it is inherently complex once you start focusing on each individual word and what that word could possibly mean. If you instead tried to summarize this lesson by the topics that it focuses on you will see that there are many fewer topics than words. By working with the topics, you will have reduced the number of dimensions with which to work.



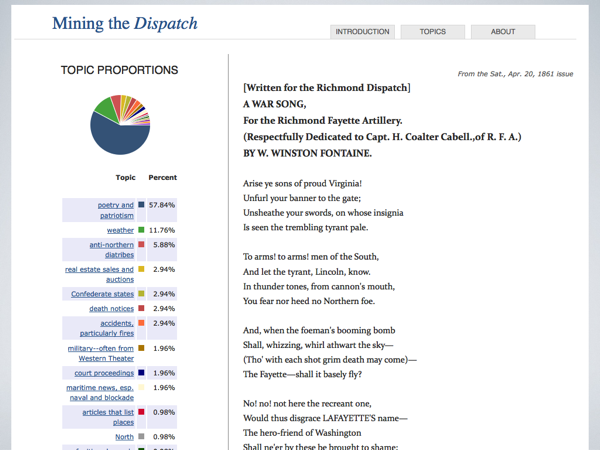
[This Photo](https://datascience.stackexchange.com/questions/130/what-is-dimensionality-reduction-what-is-the-difference-between-feature-selecti) by Unknown Author is licensed under [CC BY-SA](https://creativecommons.org/licenses/by-sa/3.0/)

Figure 5.4: scalars, vectors, matrices and tensors

So, topic modelling is one way to perform dimensionality reduction in preparation for machine learning projects.

## Historical Analysis

History is literally dimensionality reduction – historians take the vast amount of written information and constructing themes and stories that help us understand what happened in the past. It turns out that topic modelling can be a really useful tool for historians.

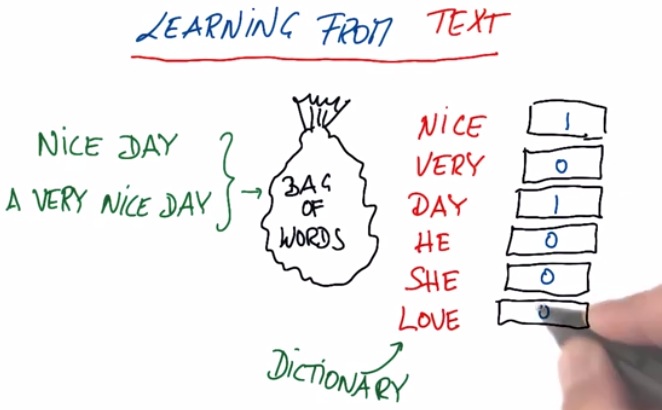


[This Photo](https://www.briancroxall.net/2014/03/07/the-red-herring-of-big-data/) by Unknown Author is licensed under [CC BY](https://creativecommons.org/licenses/by/3.0/)

Figure 5.5: scalars, vectors, matrices and tensors

## Bag of Words

Before we go into how the topic modelling algorithms work, we should make one or two simplifying assumptions. Firstly, we treat documents as a **bag-of-words**, meaning we ignore the *structure* or *grammar* of the document and just use the counts of words in the document to infer patterns about it.



[This Photo](http://napitupulu-jon.appspot.com/posts/text-learning-ud120.html) by Unknown Author is licensed under [CC BY-SA-NC](https://creativecommons.org/licenses/by-nc-sa/3.0/)

Figure 5.6: scalars, vectors, matrices and tensors

Ignoring structure, sequences and grammar makes it so that we can use certain algorithms that rely on counts and probability to make inferences. On the other hand, we lose some information when we ignore sequences and structure. We will look at approaches that explicitly model sequences in later lesson.

## Document

A document is a collection (bag) of words. Documents mean whatever you want it to mean for the purposes of your project. You can even do topic modelling on images if you can represent the image as a collection of features.

Topic Modelling Algorithms

Topic modelling algorithms operate using the following assumptions

1. Topics contain a set of words
2. Documents are made up of a set of topics

Topics are not observed but are assumed to be hidden generators of words. After these assumptions the algorithms diverge in how they go about discovering topics.

## Latent Semantic Analysis

We will start by looking at Latent Semantic Analysis (LSA). LSA actually predates the World Wide Web – it was first described in 1988. LSA is also known by the alternative acronym Latent Semantic Indexing (LSI) particularly when used to provide semantic search in document indexes. The goal of LSA is to uncover the latent topics that underlie documents and words. The assumption is that these latent topics drive the distribution of words in the document.

Note: The original patent was

[Computer information retrieval using latent semantic structure](http://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO1&Sect2=HITOFF&d=PALL&p=1&u=%2Fnetahtml%2FPTO%2Fsrchnum.htm&r=1&f=G&l=50&s1=4,839,853.PN.&OS=PN/4,839,853&RS=PN/4,839,853)  
Inventors: Scott C. Deerwester, Susan T. Dumais, George W. Furnas, Richard A. Harshman, Thomas K. Landauer, Karen E. Lochbaum, and Lynn A. Streeter

Assigned to: Bell Communications Research, Inc.  
US Patent: 4,839,853  
Granted: June 13, 1989  
Filed: September 15, 1988

## LSA – How It Works

Let’s start at the very beginning when we have a collection of documents and these documents are made up of words. Our goal is to perform statistical analysis in order to uncover the latent topics in the documents. Note that even though the documents are comprised of **words**, we generally sometimes think of them as **terms**, so you when you see phrases such as term-to-document it roughly corresponds to a word.

So, in the beginning we have a collection of documents that we can represent as a **term-document** matrix. This term-document matrix has terms as rows and documents as columns.

**Term to Document**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Doc-1** | **Doc-1** | **Doc-3** |
| **Water** |  |  |  |
| **Dog** |  |  |  |
| **Willow** |  |  |  |
| **Cart** |  |  |  |
| **Pill** |  |  |  |
| **Stone** |  |  |  |

Figure 5.7: scalars, vectors, matrices and tensors

We want to break this matrix down into separate matrices - **topic-to-document**, **topic-to-words** and **topic-importance.** This separation is done by Singular Value Decomposition (SVD) – a matrix factorization technique for separating a rectangular matrix into other matrices.



**Term to Document**

**Term to topics**

**Topic Importance**

**=**

**x**

**x**

**m** x **n**

**m** x **m**

**n** x **n**

**n** x **m**

**Topic to Documents**

Figure 5.8: scalars, vectors, matrices and tensors

## Term Topic Matrix

## Topic Importance

## Topic to Document

## Singular Value Decomposition

We have seen how LSA relies on being able to split the document-term matrix into multiple matrices. This is one by using Singular Value Decomposition (SVD). SVD is a matrix factorization method that separates a rectangular matrix M into the product of other matrices.

Let’s see the equation for SVD. The process takes a matrix M and splits it

M = U∑V\*

* M is an m×m matrix
* U is a m×n left singular matrix
* Σ is a n×n diagonal matrix with non-negative real numbers.
* V is a m×n right singular matrix and V\* is n×m matrix, which is the transpose of the V.

## LSA Sampling Process

**Documents in Corpus**

**Words in Documents**

For each document in the corpus:

        For each word in the document:

        Select a distribution of words from the distribution of topics.

               Select a word from that distribution of words.

Figure 5.9: scalars, vectors, matrices and tensors

## Building Topic Models with the Gensim Library

For topic modelling

## Exercise 1: Analyzing Reuters New Articles with Latent Semantic Analysis

In this exercise we will analyze Reuters new articles.

The Reuters news article dataset

We will use gensim LsiModel

1. Open a Jupyter notebook to implement this exercise
2. Add the following code in a new code cell. This imports the main gensim packages that we will use for the exercise

from gensim import corpora

from gensim.models import LsiModel

from gensim.parsing.preprocessing import preprocess\_string

1. Now we will have a code section that focuses on cleaning the text. We clean text to remove characters that do not add useful information to our model, and the choice of what characters to clean often depends on the project. One decision would be to remove non-alphanumeric characters especially since the data for our project are Reuters articles, which are unlikely to have useful special characters like emoticons which would be more common in more casual text sources.

So, we will remove non-alphanumeric characters and we will replace numbers with the # mark. Normalizing numbers in this way allows our model to use the numbers in the articles as tokens which may add some information.

import re

def clean\_text(x):

pattern = r'[^a-zA-z0-9\s]'

text = re.sub(pattern, '', x)

return x

def clean\_numbers(x):

if bool(re.search(r'\d', x)):

x = re.sub('[0-9]{5,}', '#####', x)

x = re.sub('[0-9]{4}', '####', x)

x = re.sub('[0-9]{3}', '###', x)

x = re.sub('[0-9]{2}', '##', x)

return x

def clean(x):

x = clean\_text(x)

x = clean\_numbers(x)

return x

1. The Reuters news articles are contained in files with the extension \*.sgm. These files are actually XML files in a format proprietary to Reuters. Each file contains multiple articles and the text of each article is within the <BODY></BODY> XML element. We will ignore the other fields for now and focus on retrieving the article text from this element from all the files.

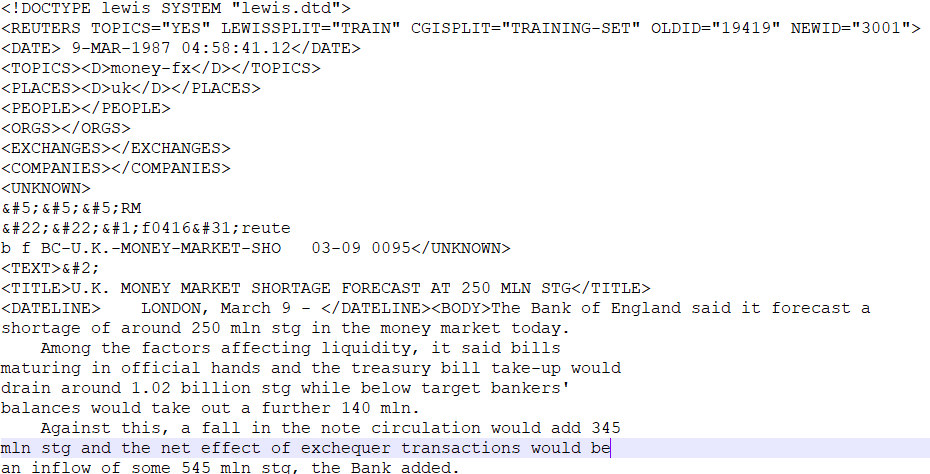


Figure 5.10: scalars, vectors, matrices and tensors

To read XML in Python we have at least two good choice. We can use **ElementTree** which is the inbuilt Python module for reading XML. We can also use **BeautifulSoup** a parser for HTML and XML. Because BeutifulSoup is designed to parse HTML it is considerably more forgiving of unusual characters, and as it turns out the Reuters SGM files do have come characters that trip up the ElementTree parser. So we will use BeautifulSoup.

The code below loops through each file, reads the contents, and creates a BeautifulSoup instance from it. Then it yields the text of the <BODY> element.

from pathlib import Path

from bs4 import BeautifulSoup

import re

def load\_articles(data\_dir):

reuters = Path(data\_dir)

for path in reuters.glob('\*.sgm'):

with path.open() as sgm\_file:

contents = sgm\_file.read()

soup = BeautifulSoup(contents)

for article in soup.find\_all('body'):

yield article.text

1. The **load\_articles** function we created above is specialized for reading the Reuters SGM file. Because it uses the yield keyword it operates as a generator and we will need to turn this into a list of documents for our model training. So, now we will add code that will load all the documents into a list. The function **load\_documents** will use the **load\_articles** function and return the list of documents.

def load\_documents(document\_dir):

print(f'Loading from {document\_dir}')

documents = list(load\_articles(document\_dir))

print(f'Loaded {len(documents)} documents')

return documents

1. After we load the documents, we need to run a step to prepare the documents for the model. In this step, we will clean the text using the clean functions that we created earlier. Then we will run each document through a series of text processing functions that are often required for LSA and other similar models.

We will use the gensim **preprocess\_string** function. This function rokenizes the documents into individual tokens and also applies the following list of functions that will be applied to each document.

* [**strip\_tags()**](https://radimrehurek.com/gensim/parsing/preprocessing.html#gensim.parsing.preprocessing.strip_tags),
* [**strip\_punctuation()**](https://radimrehurek.com/gensim/parsing/preprocessing.html#gensim.parsing.preprocessing.strip_punctuation),
* [**strip\_multiple\_whitespaces()**](https://radimrehurek.com/gensim/parsing/preprocessing.html#gensim.parsing.preprocessing.strip_multiple_whitespaces),
* [**strip\_numeric()**](https://radimrehurek.com/gensim/parsing/preprocessing.html#gensim.parsing.preprocessing.strip_numeric),
* [**remove\_stopwords()**](https://radimrehurek.com/gensim/parsing/preprocessing.html#gensim.parsing.preprocessing.remove_stopwords),
* [**strip\_short()**](https://radimrehurek.com/gensim/parsing/preprocessing.html#gensim.parsing.preprocessing.strip_short),
* [**stem\_text()**](https://radimrehurek.com/gensim/parsing/preprocessing.html#gensim.parsing.preprocessing.stem_text).

def prepare\_documents(documents):

print('Preparing documents')

documents = [clean(document) for document in documents]

documents = [preprocess\_string(doc) for doc in documents]

return documents

1. Now we will create our model. The model we will use is the LsiModel. LSI stands for **Latent Semantic Indexing** which is used somewhat interchangeably with Latent Semantic Analysis.

The LSA model is created from a document-term matrix, a dictionary of words and the number of topics which you have to specify.

def create\_lsa\_model(documents, dictionary, number\_of\_topics):

print(f'Creating LSA Model with {number\_of\_topics} topics')

document\_terms = [dictionary.doc2bow(doc) for doc in documents]

return LsiModel(document\_terms,

num\_topics=number\_of\_topics,

id2word = dictionary)

def run\_lsa\_process(documents, number\_of\_topics=10):

documents = prepare\_documents(documents)

dictionary = corpora.Dictionary(documents)

lsa\_model = create\_lsa\_model(documents, dictionary,

number\_of\_topics)

return documents, dictionary, lsa\_model

1. In the code cell below, we will run the main series of steps for the exercise. This will load the documents and run the LSA process. Then we end up with the documents a dictionary and a model.

document\_dir ='data/reuters'

articles = list(load\_articles(document\_dir))

documents, dictionary, model = run\_lsa\_process(articles, number\_of\_topics=8)

1. Once created our model contains information about the topics that we specified and the word tokens that contributed to each topic. We can use the **print\_topics** function to see this information.

The LSI model used the text data that we provided tried to separate the word tokens into the number of topics that we specified. Since we specified 8, then you will see 8 topics. The topics are unnamed, and they are overlapping so you will see some words assigned to multiple topics. We will look later at ho to specify the number of topics for our model to make sure that the topic creation and assignment is as accurate as possible.

Let’s call the **print\_topics** function

model.print\_topics()

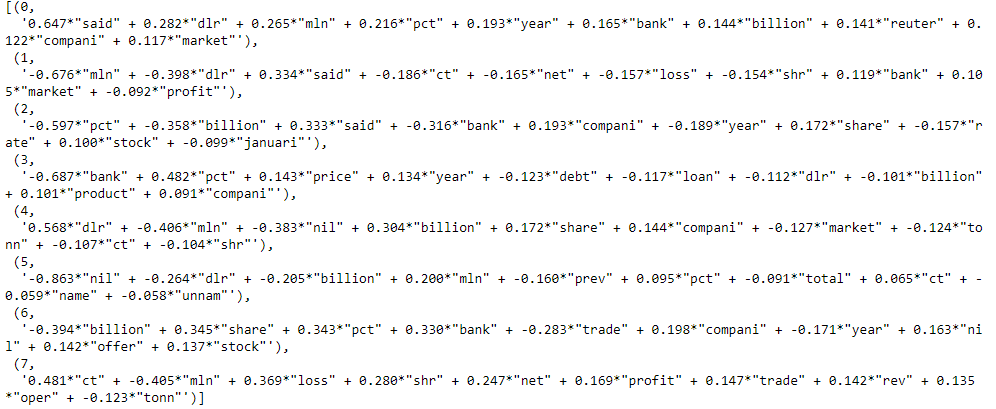


Figure 5.11: This image showing the result of calling the print\_topics function on the LsiModel

1. To create our LsiModel we had to decide up-front how many topics we wanted. This does not necessarily match with what is intrinsic to the Reuters articles. There could in fact be more or fewer natural topics than the 8 that we chose. In order to find a good number of topics we can use the **CoherenceModel** that gensim provides to see how well our model’s topics were chosen and how well they fit together. Generally we do not want the topics to overlap too much and the coherence score will let us know if that is the case.

Add the following code in a new code cell

from gensim.models.coherencemodel import CoherenceModel

def calculate\_coherence\_score(documents, dictionary, model):

coherence\_model = CoherenceModel(model=model,

texts=documents,

dictionary=dictionary,

coherence='c\_v')

return coherence\_model.get\_coherence()

def get\_coherence\_values(start, stop):

for num\_topics in range(start, stop):

print(f'\nCalculating coherence for {num\_topics} topics')

documents, dictionary, model = run\_lsa\_process(articles,

number\_of\_topics=num\_topics)

coherence = calculate\_coherence\_score(documents,

dictionary,

model)

yield coherence

Figure 5.10: This image shows our function calculating coherence score on different number of topics

1. We want to calculate coherence scores for a range between 2 and 20. Add the following to a new code cell

min\_topics, max\_topics = 2,20

coherence\_scores = list(get\_coherence\_values(min\_topics, max\_topics))

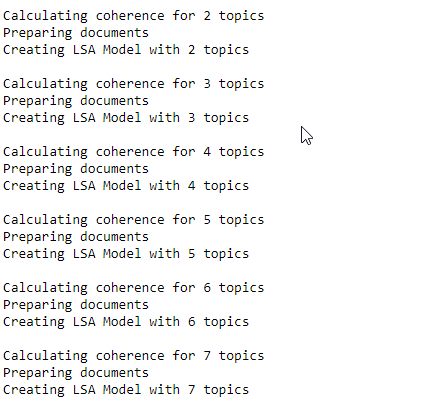


Figure 5.10: This image shows our function calculating coherence score on different number of topics

1. Now we can chart the coherence scores to see which number of topics would be the best choice. We will use matplotlib.

Add the following code in a new code cell.

import matplotlib.pyplot as plt

import matplotlib.style as style

style.use('fivethirtyeight')

%matplotlib inline

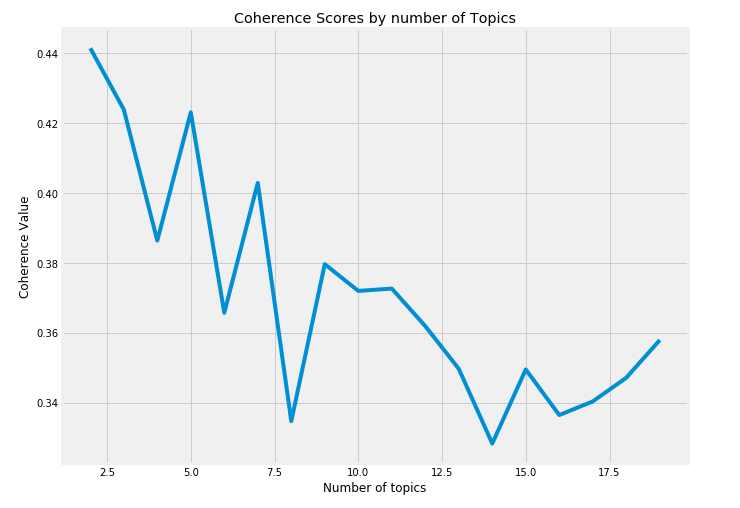
x = [int(i) for i in range(min\_topics, max\_topics)]

plt.plot(x, coherence\_scores)

plt.xlabel('Number of topics')

plt.ylabel('Coherence Value')

Figure 5.11: scalars, vectors, matrices and tensors



As we can see from the chart the appropriate number of topics is quite low – 2 topics would be the best choice. This means that the

## LSA and Overfitting

LSA is prone to overfitting on the training data. After you have trained your model and you add new documents that the model will get progressively worse until you retrain the model.

Latent Dirichlet Allocation

**Latent Dirichlet Allocation** (LDA) is a generative statistical model that allows a set of **items** to be sorted into by *unobserved* **groups** by similarity. LDA applied to items that are each made up of parts and on which similarity sorting can be done using the statistical patterns of the parts.

LDA can be done on any collection of things that are made up of parts – employees and skills, sports teams and individuals, documents and words etc. Since this lesson is about natural language processing, we are primarily interested in documents and words.

In topic modelling the groups are *unobserved* since we do not know ahead of time what they are in the documents. However, what we can observe are the *words* in the documents and we use the statistical patterns about the words to figure out the groups – which in topic modelling are the topics.

Latent means unobserved or hidden. The topics are hidden from us but implicitly affect the words in the document. We an attempt to find the topics by reverse engineering the statistical pattern in the words.

Each document is assumed to have a small number of topics associated with it. For example, a document could be CAT-Related. In CAT-Related documents there is higher probability that you will find words such as *meow, purr, kitty*, while in DOG-Related documents there is a higher probability of finding words like *bone, bark, wag*. This probability is called a prior probability which is an assumption about something before actually measuring it.

LDA is a way of measuring the statistical patterns of words in the documents so that we can deduce the topic. So, if we tend to see more words that are CAT-Topic like then we can infer that the document cold be CAT-Topic related.

LDA was first described as a means of topic discovery in 2003 by Andrew Ng, David Blei, and Michael I. Jordan.

## LDA: How it works

The goal of LDA is to infer topics using statistical analysis. It uses the statistical counts of the data to infer the *latent* or *unobserved* topics. It keeps track of the statistical counts of the ***composites***, the ***parts*** and the ***groups***.

For topic modelling of text documents, the following holds

|  |  |  |
| --- | --- | --- |
| Composites | = | Documents |
| Parts | = | Words |
| Groups | = | Topics |

Let’s look at a specific example. We have four documents that contains only 3 unique words – **cat**, **dog** and **hippo**. The table below shows the documents and the number of times each word is found in each document. For example, the word Cat is found 10 times in Document 1 and Document 4 and 0 times in documents 2 and 3. Document 4 contains all three words 10 times each.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Cat** | **Dog** | **Hippo** |
| **Document 1** | **10** | 0 | 0 |
| **Document 2** | 0 | **10** | 0 |
| **Document 3** | 0 | 0 | **10** |
| **Document 4** | **10** | **10** | **10** |

For its analysis LDA maintains two probability tables. The first table tracks the probability of selecting a specific **word** when sampling a specific **topic**. The second table keeps track of the probability of selecting a specific topic when sampling a particular document.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Words vs Topics** | | | |  | **Documents vs Topics** | | | |
|  | | | |  |  | | | |
|  | **Topic 1** | **Topic 2** | **Topic 3** |  |  | **Topic 1** | **Topic 2** | **Topic 3** |
| **Cat** | 0.00 | 0.00 | **0.99** |  | **Document 1** | 0.030 | 0.030 | 0.939 |
| **Dog** | **0.99** | 0.00 | 0.00 |  | **Document 2** | 0.939 | 0.030 | 0.030 |
| **Hippo** | 0.00 | **0.99** | 0.00 |  | **Document 3** | 0.030 | 0.939 | 0.030 |
|  |  |  |  |  | **Document 4** | 0.33 | 0.33 | 0.33 |

The probability tables reflect how likely it would be to get a particular word if you sampled from each topic. If you sampled a word from Topic 3 it would likely be Cat (probability 99%). If you sampled Document 4 then because it contains all three words in equal proportions then there is a one-third chance of getting each of the topics.

## Hyper parameters

Alpha controls the sampling of **topics** from **documents**. A low alpha places more weight on having each document composed of only a few dominant topics.

Beta controls the sampling of words from topics. A low beta value places more weight on having each topic composed of a few dominant words.

**Documents in Corpus**

**Words in Documents**

**alpha**

**beta**

For each document in the corpus:

Select a topic mixture distribution from a Dirichlet distribution.

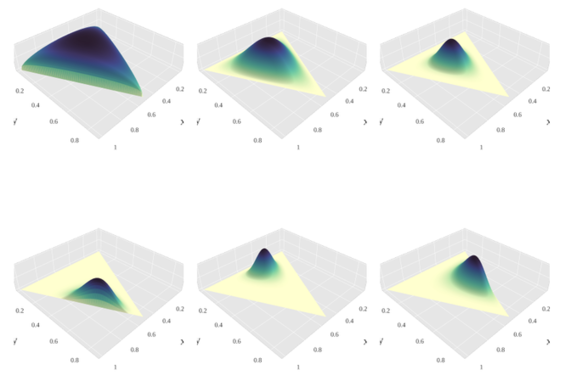
For each word in the document:

Select a topic from the topic mixture distribution.

       Select a word from the word distribution selected above

## The Dirichlet Distribution

LDA is based on the Dirichlet distribution. Dirichlet is a multivariate distribution – it describes the distribution of multiple variables. It is used for topic modelling since we are usually concerned about multiple topics. Dirichlet helps us specify our assumptions about the probability distributions about before doing the analysis. In other words it helps us with our prior assumptions.



[This Photo](https://en.wikipedia.org/wiki/Dirichlet_distribution) by Unknown Author is licensed under [CC BY-SA](https://creativecommons.org/licenses/by-sa/3.0/)

Figure 5.12: scalars, vectors, matrices and tensors

Note: The Dirichlet distribution is named for a German mathematician Peter Gustave Lejeune Dirichlet who was trained in Paris France. One of the first steps in understanding Dirichlet is learning how to pronounce it as there is some debate as to whether to pronounce it the French way *DirishLAY* or the German way *DiriKLET*



[This Photo](https://fr.wikipedia.org/wiki/Johann_Peter_Gustav_Lejeune_Dirichlet) by Unknown Author is licensed under [CC BY-SA](https://creativecommons.org/licenses/by-sa/3.0/)

Dirichlet was born in **Düren** in 1805 when that city was part of the French Empire under Napoleon before it reverted 1815. Because of this multicultural origin, there probably isn’t a right way to pronounce the name – either might do.

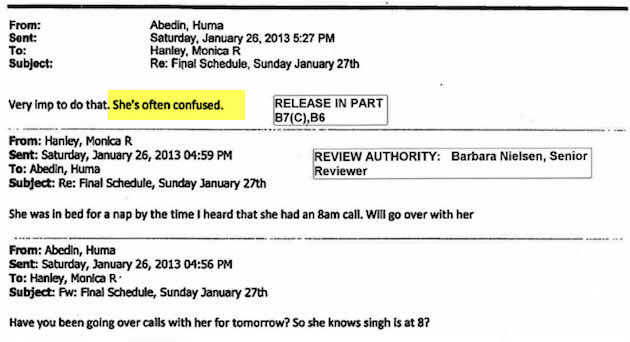
For more detail on this follow the debate on Wikipedia here https://en.wikipedia.org/wiki/Talk:Peter\_Gustav\_Lejeune\_Dirichlet

## Exercise 2: Topic Modelling the Hilary Clinton Emails

During the 2016 US presidential elections there was an intense ongoing controversy surrounding Hilary Clinton’s use of a personal email server while she was Secretary of State. In response to Freedom of Information requests, the State Department released batches of these emails as PDF documents to the public.

Some enterprising data scientists have taken the opportunity to parse these PDFs into CSV datasets thereby making them available for data science projects. You can find the email parsing project on github at <https://github.com/benhamner/hillary-clinton-emails>. The dataset and more details on it are also hosted on Kaggle at <https://www.kaggle.com/kaggle/hillary-clinton-emails>.

In the image below you can see samples of the emails.



[This Photo](http://impiousdigest.com/father-of-doctor-treating-hillary-clinton-for-dementia-mysteriously-dies/) by Unknown Author is licensed under [CC BY](https://creativecommons.org/licenses/by/3.0/)

Figure 5.13: scalars, vectors, matrices and tensors

In this exercise we will perform topic modeling on the Clinton emails in order to discover what topics were being discussed. We will use the Latent Dirichlet Allocation algorithm as provided by the **gensim** library.

1. Open a Jupyter notebook to implement this activity
2. The email data has already been downloaded to the lesson data directory and is located in the Emails.csv files. We will use the pandas library to read the emails into a dataframe in order to do the text processing and build the LDA topic model. For starters we will add a few import statements.

import pandas as pd

import warnings

warnings.filterwarnings('ignore')

pd.set\_option('display.max\_colwidth', 900)

We import pandas and set the display column width to a large enough value so that the email contents can be shown on the screen. There may also be some technical warnings being emitted by some of the libraries that we so we will filter them out. That is technically not required, but it avoids some confusion as to whether there is a real issue in the program. In your own real world program you probably shouldn’t ignore warnings unless you are producing a notebook for distribution to others.

1. Now we read the email csv into a dataframe. We will drop records where the ExtractedBodyText is NA – meaning there was no body content that could be extracted from the email.

emails = pd.read\_csv('data/Hillary-clinton-emails/Emails.csv')

emails = emails.dropna(subset=['ExtractedBodyText'])

1. After loading into a dataframe we can safely select only those columns that contain text content for the email – the main ones being the **RawText** and **ExtractedBodyText**. We will also include the columns that start with “**Extr**” because they will give us the ability to further remove some of this from the text of the emails as part of cleaning.

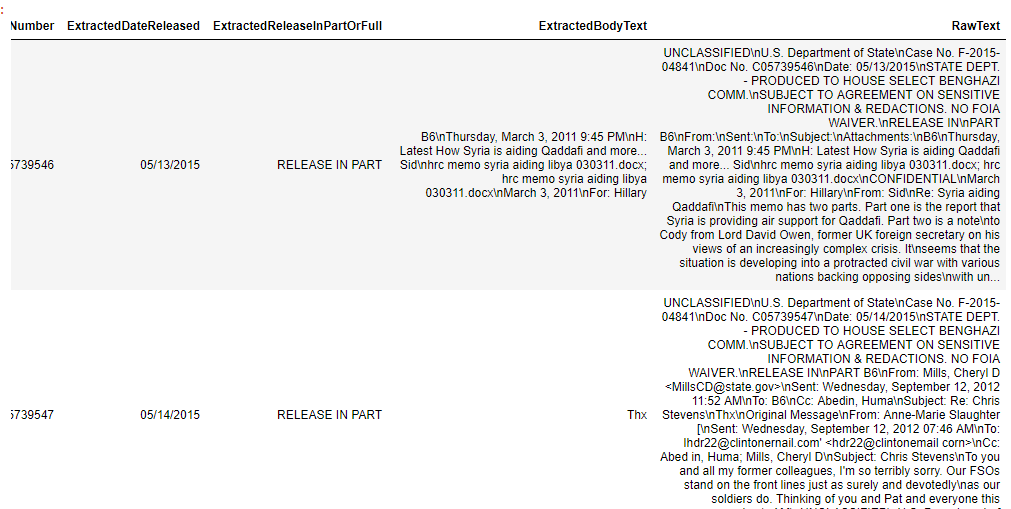
cols = [col for col in emails.columns if

col.startswith('Extr') or col=='RawText']

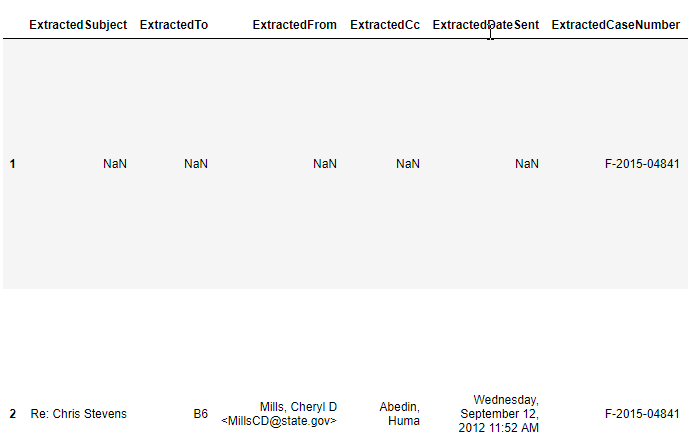
emails = emails[cols]

1. We can now take a look at the emails just to see what the content looks like. Use the **dataframe** **head**() function.

emails.head(4)



1. Some of the fields still contain Nan values as shown in the image below. Nans are floats and we want them as strings so that we can use them to clean the emails.



The solution is to fill the NA values with a special string “-999” then convert the column type to str. Rather than specify the columns names explicitly we can loop through the columns of the datafframe that start with “Extr” and set the column type to str.

NAN\_VALUE = str(-999)

emails = emails.fillna(NAN\_VALUE)

for col in emails.columns:

if col.startswith('Extr'):

emails[col] = emails[col].astype(str)

1. We now are at the text cleaning step. We will use the Python regex package re for this and so we will add a number of regular expressions that we will use to clean the strings.

import re

DATE = '[A-Z][a-z]{5,}, [A-Z][a-z]{3,} [0-9]{1,2}, 201[0-9]( [0-9]{1,2}:[0-9]{1,2} [AP]M)?'

SHORT\_DATE = '[A-Z][a-z]{3,}, [0-9]{1,2} [A-Z][a-z]{2} 201[0-9]'

TIME = '[0-9]{1,2}:[0-9]{1,2} [AP]M'

EMAIL\_ADDRESS = '[a-zA-Z0-9\_.+-][+@[a-zA-Z0-9-]+\.?[a-zA-Z0-9-](mailto:+@[a-zA-Z0-9-]+\.?%5ba-zA-Z0-9-).]+'

EMAIL\_FIELDS = '(Sent|Subject|From|Fw|Re|Cc|To|Attachments|Date):?'

ALL\_UPPERCASE = '[A-Z]{2,}'

STATE\_DEPT\_TEXT = 'U.S. Department of State| Case No|Doc No|B5|B6'

1. Now apply the clean function to the emails dataframe. We use the regexes that we defined above and the re.sub function to replace the values with a blank string. At the end we create a new column called **CleanText** that contains the values of the RawText column after applying the clean function

def clean(text):

text = re.sub('\n', ' ', text)

text = re.sub(DATE, '', text)

text = re.sub(SHORT\_DATE, '', text)

text = re.sub(TIME, '', text)

text = re.sub(EMAIL\_FIELDS, '', text)

text = re.sub(EMAIL\_ADDRESS, '', text)

text = re.sub(ALL\_UPPERCASE, '', text)

text = re.sub(STATE\_DEPT\_TEXT, '', text)

return text

emails['CleanText'] = emails.RawText.apply(clean)

1. In this cleaning step will take the values of the extracted columns and replace them in the **CleanText** column if they are found.

def remove\_extracted\_field(text, field\_value):

if field\_value == NAN\_VALUE:

return text

return text.replace(str(field\_value), '')

for col in emails.columns:

if col.startswith('Extracted'):

emails.CleanText = emails.apply(lambda d:

remove\_extracted\_field(d.CleanText,

d[col]),

axis=1)

1. At the end we should have a much cleaner column that we can use. It isn’t perfect of course, and there are still some person names in the emails that might affect what the LDA model will learn. For now, we will accept this but in the future we can improve this by doing some entity recognition to detect the person names so we can remove them.

To display the results so far, use the head function.

emails[['BodyText']].head(5)



1. After text cleaning we are ready to use the gensim library to build our model. There is one more step required which is the step to convert the data into the form required for gensim. The gensim LDA model requires the data to be in a list of list of tokens – meaning each entry in the list is a list of the tokens that make up each individual document. This means that we need to tokenize each email body – breaking it into individual words.

There are other preprocessing steps required. One of these is stemming, which is reducing a word to its basic form – a stem. For example, the words *office*, *official*, and *officials* are all reduced to the token *offic*. This in theory means that the model captures the root meaning of the word and so is better able to infer the topics.

from gensim.parsing.preprocessing import preprocess\_string

email\_documents = emails.BodyText.apply(preprocess\_string).tolist()

From the gensim documentation the preprocess\_string fucntion applies the following filters.

* [**strip\_tags()**](https://radimrehurek.com/gensim/parsing/preprocessing.html#gensim.parsing.preprocessing.strip_tags),
* [**strip\_punctuation()**](https://radimrehurek.com/gensim/parsing/preprocessing.html#gensim.parsing.preprocessing.strip_punctuation),
* [**strip\_multiple\_whitespaces()**](https://radimrehurek.com/gensim/parsing/preprocessing.html#gensim.parsing.preprocessing.strip_multiple_whitespaces),
* [**strip\_numeric()**](https://radimrehurek.com/gensim/parsing/preprocessing.html#gensim.parsing.preprocessing.strip_numeric),
* [**remove\_stopwords()**](https://radimrehurek.com/gensim/parsing/preprocessing.html#gensim.parsing.preprocessing.remove_stopwords),
* [**strip\_short()**](https://radimrehurek.com/gensim/parsing/preprocessing.html#gensim.parsing.preprocessing.strip_short),
* [**stem\_text()**](https://radimrehurek.com/gensim/parsing/preprocessing.html#gensim.parsing.preprocessing.stem_text).

An alternative would be to use gensim.utils.simple\_preprocess which does not apply all these functions but tokenizes and lowercases the words.

1. For the LDA model we need to create a dictionary of each token in the dataset. The dictionary captures each unique word and gives it an index.

from gensim import corpora

from gensim.models.ldamodel import LdaModel

dictionary = corpora.Dictionary(email\_documents)

corpus = [dictionary.doc2bow(text) for text in email\_documents]

1. Now we will create an LDAModel that will learn on 8 topics. We specify that the model will make 8 passes through the data.

NUM\_TOPICS = 8

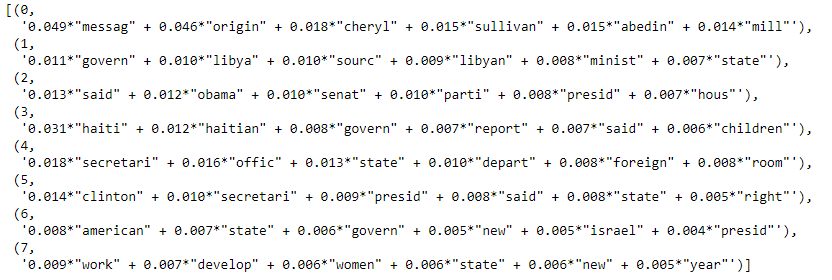
ldamodel = LdaModel(corpus,

num\_topics = NUM\_TOPICS,

id2word=dictionary, passes=15)

1. After training we can access the topics as determined by the model. One method is by using the **print\_topics** function. This gives us a list of each topic and the words that are attributed to each topic.

ldamodel.print\_topics(num\_words=6)



The first topic seems to have a concentration of person names which means we could do additional cleaning to remove these. The second topic deals with Libya, which from what we know was one of the actual topics discussed in the Hilary Clinton emails. The third topic seems to be about President Obama and the Senate.

Note that because as part of preprocessing, we stemmed the words then what is printed here are the stemmed words e.g. *presid* instead of *president.*

With topic modelling you often have to use intuition in order to decipher what the model found. Another thing we can do is to see if we have chosen the correct number of topics. We will look at how to select the number of topics later in this lesson.

1. Adsd

from gensim.models.coherencemodel import CoherenceModel

def calculate\_coherence\_score(documents, dictionary, model):

coherence\_model = CoherenceModel(model=model,

texts=documents,

dictionary=dictionary,

coherence='c\_v')

return coherence\_model.get\_coherence()

def get\_coherence\_values(start, stop):

for num\_topics in range(start, stop):

print(f'\nCalculating coherence for {num\_topics} topics')

ldamodel = LdaModel(corpus,

num\_topics = num\_topics,

id2word=dictionary, passes=2)

coherence = calculate\_coherence\_score(email\_documents,

dictionary,

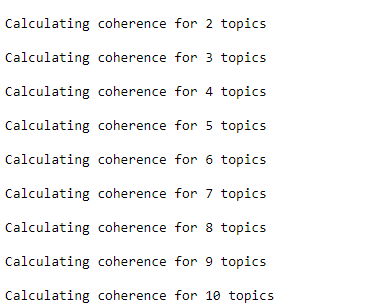
ldamodel)

yield coherence

1. Dasda

min\_topics, max\_topics = 2,30

coherence\_scores = list(get\_coherence\_values(min\_topics, max\_topics))



1. Asdsadasd

import matplotlib.pyplot as plt

import matplotlib.style as style

from matplotlib.ticker import MaxNLocator

style.use('fivethirtyeight')

%matplotlib inline

x = [int(i) for i in range(min\_topics, max\_topics)]

ax = plt.figure(figsize=(10,8))

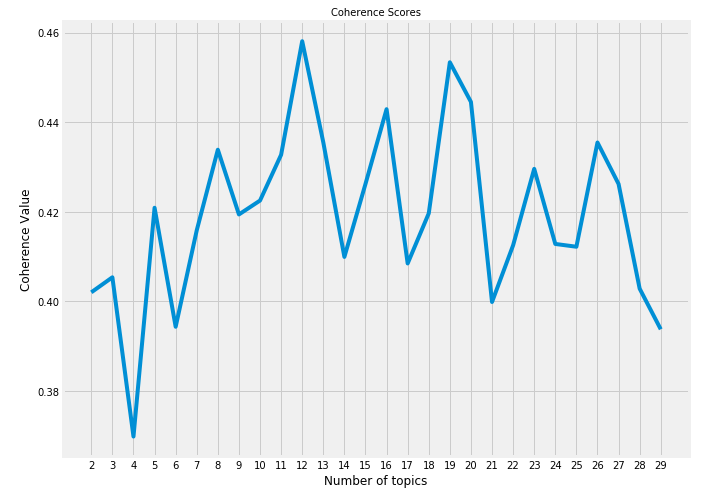
plt.xticks(x)

plt.plot(x, coherence\_scores)

plt.xlabel('Number of topics')

plt.ylabel('Coherence Value')

plt.title('Coherence Scores', fontsize=10);



Topic Vectors

* 1. Open a Jupyter notebook to implement this activity
  2. Add the following import statement for the pandas **dataframe** package. We also set the pandas option to set the column display width to 800 so that we can easily view the contents of the description column.

import pandas as pd

pd.set\_option('display.max\_colwidth', 800)

* 1. The OpenData inventory data is located in a CSV file in the lesson folder. We will use pandas to read the data into a dataframe. We want to load only a few of the columns including the description\_en column, which we will use as the target for topic modelling.

OPEN\_DATA\_URL = 'data/canada-open-data/inventory.csv'

COLUMNS = ['title\_en', 'description\_en','date\_released']

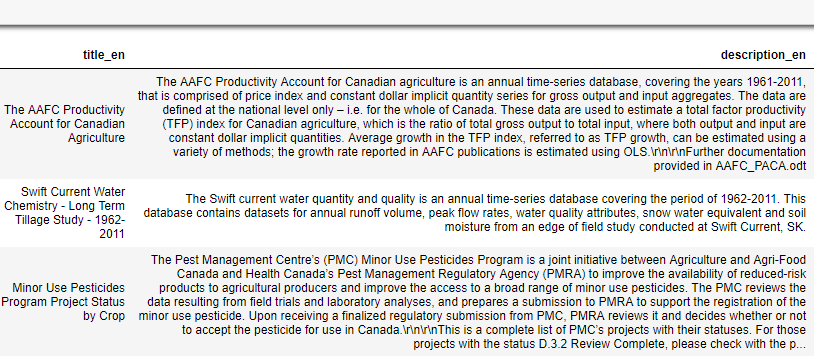
catalog = pd.read\_csv(OPEN\_DATA\_URL, usecols=COLUMNS)

catalog = catalog.dropna(subset=['description\_en'])

We also drop rows that contain NA values in the **description\_en** column.

* 1. You can now view the contents of the catalog. Add this code in a new code cell.

catalog



* 1. Now we are at the text processing step. First, we convert the text to lowercase then we remove stopwords like “the”, “and”, “so” since they do not add any value to the model.

Finally, we need to create a list of lists of tokens. Each item in the outer list is the description split into individual tokens. For this we use the gensim simple\_preprocess function.

Add the following code in a new code cell:

from gensim.parsing.preprocessing import remove\_stopwords

from gensim.utils import simple\_preprocess

def text\_to\_tokens(text):

text = text.lower()

text = remove\_stopwords(text)

tokens = simple\_preprocess(text)

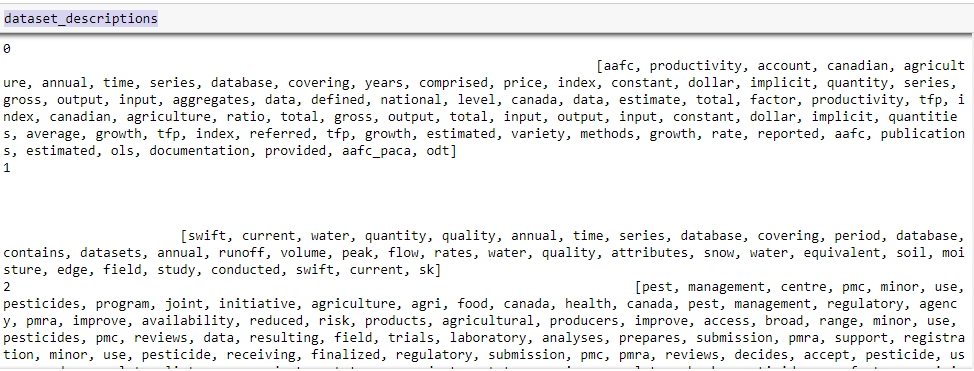
return tokens

* 1. Now we can create a new description dataset by applying the text\_to\_tokens function. This will be the input for the topic model.

dataset\_descriptions = catalog.description\_en.apply(text\_to\_tokens)

* 1. If you look at the contents of dataset\_descriptions you will see that it contains a list of the tokens extracted from each description using the text\_to\_tokens function.

dataset\_descriptions



* 1. We will be using the gensim LdaModel so we will add the import statements. We also create two objects – a dictionary containing the tokens from the dataset\_descriptions, and a corpus.

import gensim

from gensim.models import LdaModel

from gensim.parsing.preprocessing import preprocess\_string

dictionary = gensim.corpora.Dictionary(dataset\_descriptions)

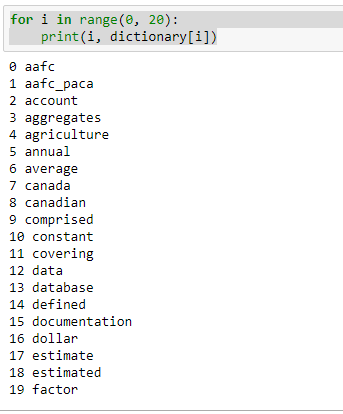
corpus = [dictionary.doc2bow(text)

for text in dataset\_descriptions]

* 1. To get a sense of what is contained in the dictionary we can loop through the first 20 items.

for i in range(0, 20):

print(i, dictionary[i])



* 1. For our topic vectors we will choose a vector size of 50. This means we will build a model with 50 topics.

VECTOR\_SIZE=50

lda\_model:LdaModel = LdaModel(corpus,

num\_topics=VECTOR\_SIZE,

passes=4)

* 1. To illustrate how we will create topic vectors let us look at the process for one description value. In the code below we choose the first catalog\_description, converted it to tokens, then converted the tokens to a bag of words. The final step is to use the model to indicate which topic that description was associated with and the relevance of that topic.

text = catalog.description\_en[0]

tokens = text\_to\_tokens(text)

bag\_of\_words = dictionary.doc2bow(tokens)

pd.DataFrame(ldaModel[bag\_of\_words],

columns=['Topic','Relevance']).set\_index('Topic')



W have 50 topics in total, so there are some topics for which the description does not apply. These we will set to 0

* 1. The function below takes a text description and converts it to a vector with 50 items. If the description fits a topic then it will have a relevance score at that topic index. Otherwise the value will be zero.

def topic\_vector(topic\_model:LdaModel, text:str):

processed\_text = text\_to\_tokens(text)

fingerprint = [0] \* topic\_model.num\_topics

for topic, prob in topic\_model[dictionary.doc2bow(processed\_text)]:

fingerprint[topic] = prob

return fingerprint

* 1. The code below creates an image from a text description so we can visualize it.

import matplotlib.pyplot as plt

import matplotlib.style as style

from IPython.display import display

style.use('fivethirtyeight')

VECTOR\_SIZE=50

%matplotlib inline

def show\_fingerprint(topic\_model, text:str):

display(text)

vector = topic\_vector(topic\_model, text)

plt.figure(figsize=(8,1))

ax = plt.bar( range(len(vector)),

vector,

0.25,

linewidth=1)

plt.ylim(top=0.4)

plt.tick\_params(axis='both',

which='both',

left=False,

bottom=False,

top=False,

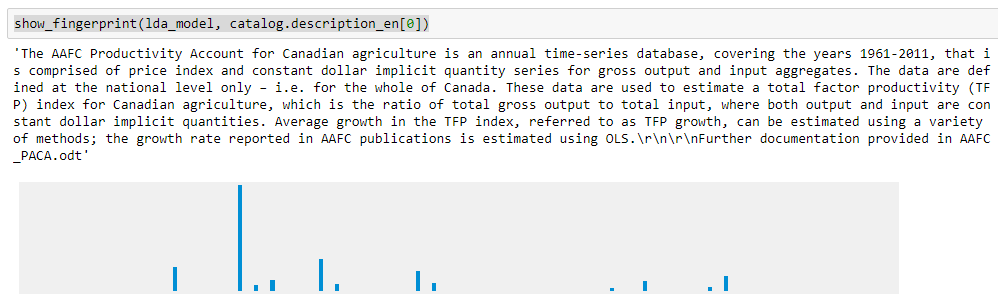
labelleft=False,

labelbottom=False)

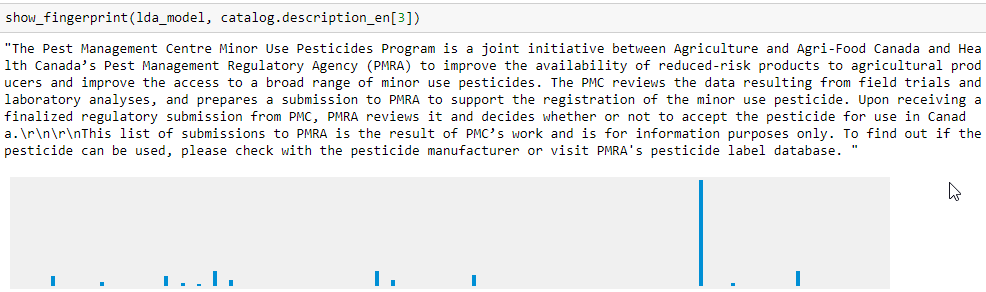
plt.grid(False)

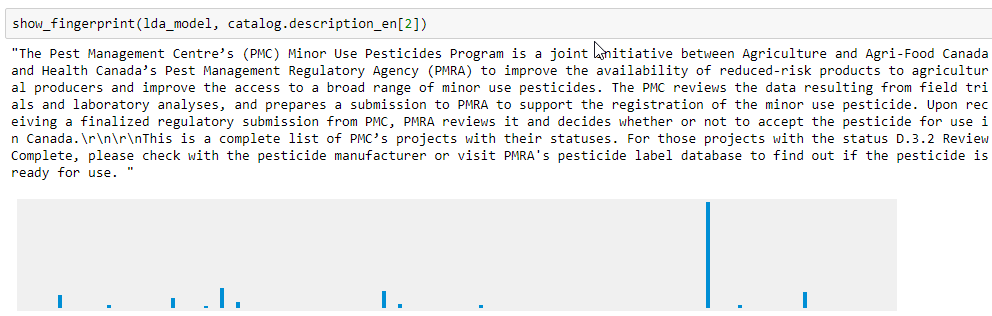
* 1. Now if we run the show\_fingerprint function on a description we can see it represented as an image.

show\_fingerprint(lda\_model, catalog.description\_en[0])



* 1. When we run the show\_fingerprint function for two similar descriptions we can see how they are nearly identical except for slight differences in the topic relevance heights. This shows that we can use topic modelling to create topic vectors which can then be used to measure text similarity.





Summary

In this lesson we looked at Topic Modelling – one of the more popular areas in natural language processing. It is popular because it does something useful – it helps us understand, categorize and sort text documents. This assumes that there are some intrinsic topics contained with the documents that can be uncovered by statistical analysis – a useful and probably true assumption. Documents are usually written based on certain themes and these themes are what data scientists try to discover.

Topic modelling has some interesting uses. We can use it to discover the themes within a set of documents allowing us to understand communities that produce content. For example, following events that affect communities – such as terrorist attacks, we can analyze forums, tweets or reddit communities to discover what topics are being discussed. We can also do topic modelling of organizational emails help in categorizing and organizing the correspondence. Topic modelling can also help companies suggest content, products or services to their customers based on document analysis.

Topic modelling is enabled by two main algorithms that try to uncover the latent topics in the text sources. The first, Latent Semantic Analysis also known as Latent Sematic Indexing, was first described in the 1980s as a way of indexing documents. It relies on Singular Value Decomposition which is a method for factoring matrixes into multiple component matrices.

Latent Dirichlet Allocation (LDA) is another method that relies on the Dirichlet distribution. It relies on our assumptions about the prior probabilities and uses that as the basic of assigning documents to topics. It is a more modern method and is thought to be more accurate.

Both LDA and LSA are available in the **gensim** package as well as other Python NLP packages and are fairly easy to use. However we also learned about their shortcomings and techniques for choosing the correct number of topics and interpreting the results.