MACHINE LEARNING &
COGNITIVE COMPUTING
EXPOSITION

LATENT DIRICHLET ALLOCATION -
A NATURAL LANGUAGE PROCESSING
ALGORITHM FOR TOPIC MODELLING

Contents

List of Figures and Tables

Figures

Tables

Introduction

The exponential growth in the amount of unstructured text data being created from blogs, social media, emails and online reviews has led to an increasing interest in developing automated methods to extract or 'mine' valuable information from text. Topic Modelling is one such statistical technique within the realm of Natural Language Processing (NLP); a highly active subset of artificial intelligence which seeks to automatically draw information from natural or human language. Topic Modelling was initially focussed on information retrieval, but more recently these techniques have been used to derive information from social networks, customer analytics, advertising and even computer vision. This paper discusses one of the most popular algorithms associated with Topic Modelling, Latent Dirichlet Allocation.

Topic Modelling

Overview

The use of Topic Modelling to automatically identify abstract topics within text is based upon the assumptions that each document consists of a mixture of topics and each topic consists of a collection of words and certain words would be expected to occur more frequently. For example, a document about the economy would be expected to contain words such as 'GDP', 'current account' and 'inflation' more often than words such as 'cake', 'children' and 'birthdays'. In practice documents will often contain more than one topic and in different proportions and therefore the amount of words associated with each of the topics will vary. Topic models captures these hidden or latent topics by clustering similar words together.

Classification, Clustering and Topic Models

Topic Modelling is often confused with Text Classification which is a *supervised* machine learning method used in spam filters and sentiment analysis and Text Clustering, an *unsupervised* clustering model that groups text based on text similarity. Hard clustering assigns data to one cluster or another and soft clustering groups text in such a way that it can belong to one or more clusters based on probabilities. Text is represented as vectors and the distance between vectors represents the similarity in the text. Use cases include document retrieval, fake news detection and language translation.

Topic Modelling can be viewed as a type of soft clustering, with the focus not on partitioning data across documents, rather documents are viewed as a collection of latent topics and each topic will have a cluster of related words with associated probabilities of that word occurring within that topic. As documents can belong to multiple topics, and words can appear in multiple topics, this provides a topical fuzziness that language requires.

Topic Model Algorithms

Statistical methods that have been developed for use in Topic Modelling include:

- Latent Semantic Indexing also known as Latent Semantic Analysis
- Latent Dirichlet Allocation
- Non-negative Matrix Factorization
- Correlated Topic Model

An explanation of Latent Dirichlet Allocation (LDA) requires brief discussion of related models discussed in the following section.

Historical Development and Motivation

Vector Space Models

The roots of topic modelling lie within document retrieval. G. Salton, A. Wong, and C. S. Yang [1], presented a Vector Space Model (VSM) as a means of automatic indexing for document retrieval. In a VSM, documents are represented as vectors of identifiers or index terms which form the vector dimensions.

$$\text{Vector Space} = \{W_1, W_2, \ldots W_n\}$$

The number of n distinct terms across all documents is the number of dimensions and a document, D can be represented with W_{Dn} as the weight for word n:

$$D = \{W_{D1}, W_{D2}, \ldots . W_{Dn}\}$$

The weight can be in the form of a frequency count, as for a Document Term Matrix (a matrix with documents as rows and words or terms as columns). The frequency of the terms can be weighted to penalise high occurring but low importance words through the use of a Term Frequency Inverse Term Frequency (TF-IDF) matrix. The similarity between documents within the vector space, necessary for query retrieval, can be calculated using Euclidean geometry. Appendix 1 illustrates a simple example of TF-IDF.

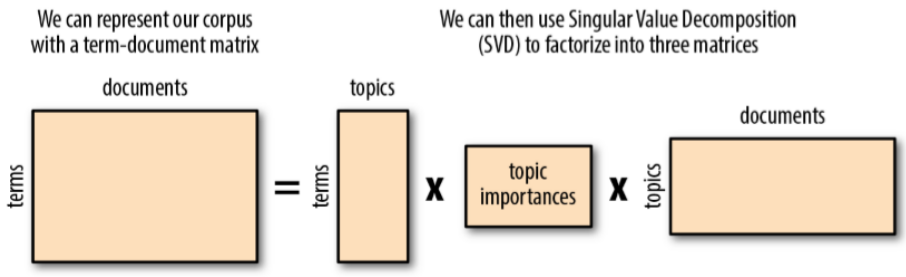
The VSM Lexigraphical models had a number of problems as a user's query was simply matched with words in a document. As a word can have multiple different meanings (polysemy) and different words can have the same or similar meaning (synonomy), search based on simple matching of terms meant relevant information was missed and irrelevant information returned.

Latent Semantic Indexing or Latent Semantic Analysis

Deerwester et al.[2], offered a new way of automatic indexing to address these issues. Using dimensionality reduction through Singular Value Decomposition (SVD) documents could be reduced to a lower semantic space by removing irrelevant elements. This left an approximation of the original data where key topics could be extracted more easily and efficiently. In summary, an (m x n) matrix M can be decomposed into a product of three simpler matrices which enable particular features of interest to be extracted. See Figure 1.

Figure 1. Latent Semantic Analysis Matrix Factorisation

$$M_{mxn} = U_{mxm} * S_{mxn} * V_{nxn}\ t$$



Source: Applied Text Analysis With Python, Bengfort, 2018

By accounting for context, LSI represented a significant improvement, in effect finding the hidden or 'latent' relationships between words. However, Lee et al. [3] note that LSI is not based on a more robust probability theory, that the number of topics could not be determined statistically and therefore relied on human judgement and it did not fully deal with the problem of polysemy.

Probabilistic Semantic Analysis

In 1999, Thomas Hofmann [4] presented a new *probabilistic generative model,* Probabilistic Latent Semantic Analysis (PLSA) relying upon conditional probabilities and latent topics that could in effect generate the data in the document-term matrix, rather than using word frequencies and SVD. Discriminative models used in supervised machine learning such as Logistic Regression, directly model the target variable y given the feature variables x so that $P(Y|X = x)$. Generative models model the target and features *together* with a joint probability distribution $P(Y, X)$ and are suited to unsupervised learning such as topic modelling.

Given a document (d), the model will look at the probability of a document given a topic (t) and the probability of a topic given the word (w) such that:

$$P(d, w) = P(d) \sum_t P(t|d)P(w|t)$$

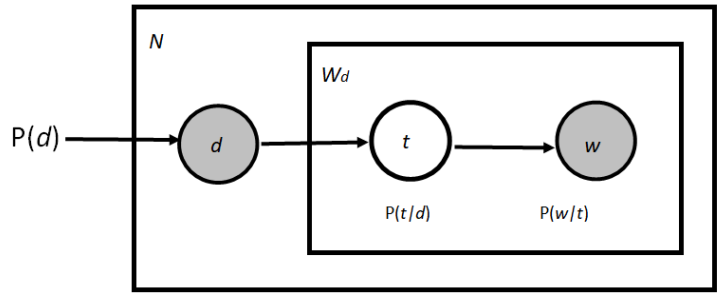
- $P(d)$ – determined from the corpus
- $P(t|d)$ and $P(w|t)$ – modelled as multinomial distributions and trained using the expectation-maximization algorithm to estimate maximum likelihood.

The probability of each co-occurrence is modelled as a mixture of conditionally independent multinomial distributions. The parameters are estimated and optimised, repeating the process towards convergence. The model can also be modelled from the perspective of the topic to generate the document and word:

$$P(d, w) = \sum_t P(t)P(d|t)P(w|t)$$

Figure 2. shows the plate diagram, a common way of representing probabilistic models. The documents (d) and words (w) are shaded since they are known and the topic (t) is latent. N represents the corpus of documents and each is comprised of W_d words.

Figure 2 The PLSA Model



Source: Amended diagram based on that provided by Joyce Xu, 2018 [5]

From the diagram:

- A document is selected or generated with probability P(d).
- Given a document (d), the probability of a topic (t) being present in the document is P(t|d).

- Given a topic (t), the probability of a word (w) being present in the topic is P(w|t).

Problems with the model include the fact that there are no parameters to model P(d), so when presented with a new document, there is no way to assign probabilities and the number of topics grows linearly with respect to the number of documents which can lead to overfitting [3].

Latent Dirichlet Allocation

Model Overview

PLSA laid the foundation for a similar and one of the most widely used techniques, Latent Dirichlet Allocation (LDA). David Blei et al. [6] published a paper in the Journal for Machine Learning in 2003 describing LDA as the following:

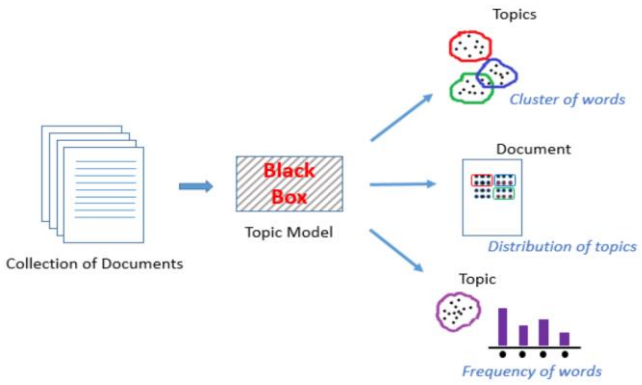
> *"LDA is a three-level hierarchical Bayesian model, in which each item of a collection is modelled as a finite mixture over an underlying set of topics. Each topic is, in turn, modelled as an infinite mixture over an underlying set of topic probabilities." [Blei et al., 2003].*

As with PLSA, Latent Dirichlet Allocation is based on the idea that the meaning or semantics of a document is based upon a range of hidden or *latent* topics and similar topics will use a similar group of words. LDA is able to discover topics across a corpus (collection) of documents by finding groups of words that frequently occur together within the documents. LDA has two basic assumptions:

- Documents that contain similar topics will contain similar groups of words.
- Latent topics can be derived by finding groups of words commonly found together

The model assumes that documents are probability distributions over topics and topics are probability distributions over words. Figure 3 shows how topics are distributed over documents; topics consist of clusters of words which are distributed over the topic and the topic model is a process or algorithm represented by the black box.

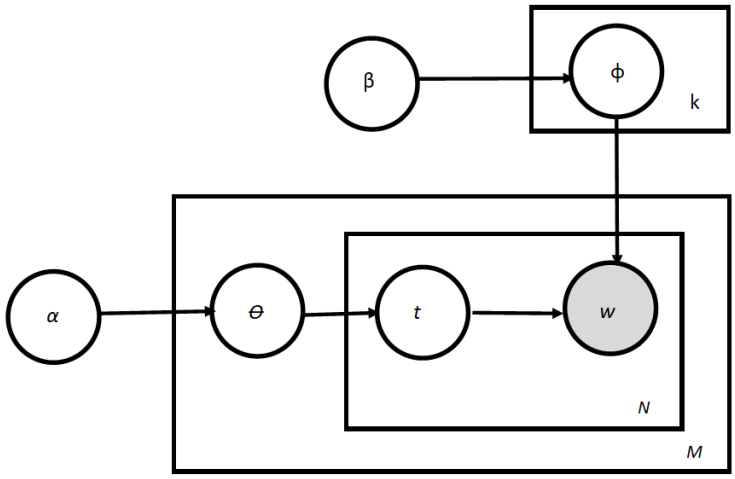
Figure 3. LDA Model



Source: Christine Doig, PyTexas 2015 [7]

The plate diagram in Figure 4 shows the relationships between the parameters of the model and can be compared to that for PLSA. M represents the corpus level (total number of documents) and N represents the number of words in a document. Where the parameters are placed indicates the level at which they operate in the model, i.e. at the document or word level or both.

Figure 4. LDA Plate Diagram



Source: Adapted from Wikipedia, Latent Dirichlet Allocation [8]

Points to note:

- Words are represented by w, topics by t and k is the number of topics.
- Alpha and beta are corpus level parameters which represent the Dirichlet prior probability distributions and these are determined by the model.
- Alpha relates to the per document topic distributions; a high value of alpha means each document is likely to contain a mixture of most of the topics and a low alpha indicates it will contain only a few.
- Beta relates to per topic word distributions; a high beta value means that each topic contains a mixture of most words and a low beta, just a few.
- Theta is document level variable and represents the topic distribution for document.
- Phi is the topic level variable and represents the word distribution for a topic.
- t_{mn} is the topic for the nth word in document m and w_{mn} is the actual word.

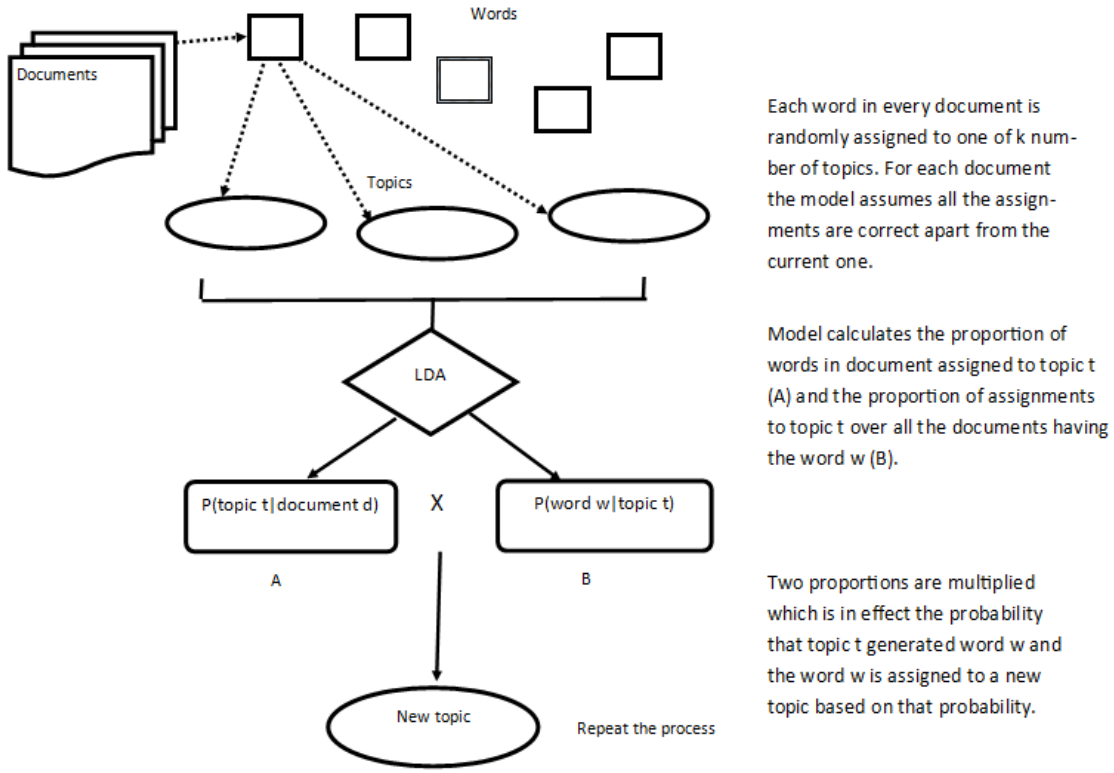
The Probabilistic Generative Process in More Depth

LDA attempts to work out how a document *could* have been created by generating topics and word distributions over the corpus. It is assumed that documents are created by firstly deciding on the number of words (N) the document will have. A document topic mix is chosen according to the Dirichlet [1] probability distribution over a fixed k number of latent topics. For example, document one may contain 50% topic A, 40% topic B and 10% topic C, and document two may contain 50% topic A and 50% topic B.

Words are generated for each document by picking a topic based on the multinomial distribution and then picking a word based on the topic's own distribution of words. In reality, the model actually works backwards from the documents to find the set of topics that are likely to have generated the collections. Figure 5. Illustrates this process where the model determines the mix of topics and words in a corpus.

[1] *"The Dirichlet distribution is a way to model random probability mass functions (PMFs) for finite sets. It is also sometimes used as a prior in Bayesian statistics. The distribution creates n positive numbers (a set of random vectors X1…Xn) that add up to **1**; Therefore, it is closely related to the multinomial distribution, which also requires n numbers that sum to 1" [Statistics How To, 2016].*

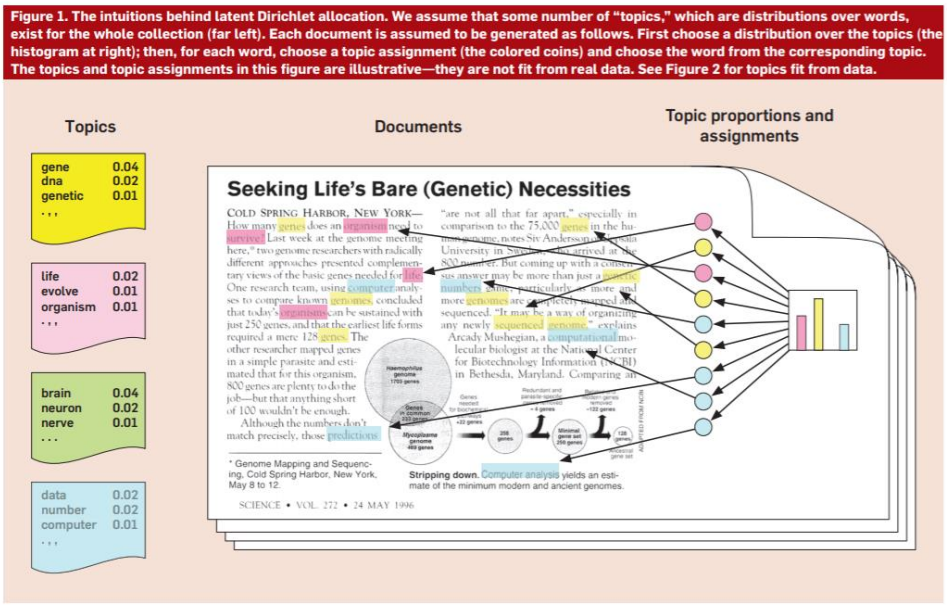
Figure 5 LDA Generative Process



Generated from Explanation by Sullivan, (YouTube,2017) [9]

Figure 6 below from Blei's research paper [10] uses an example from 17,000 articles in Science Magazine, which clearly shows the topic and word distributions.

Figure 6. LDA Topic and Word Distributions



Source: David Blei, Communications of the ACM, 2012

Comparing LDA to a common text classification model such as Naïve Bayes (NB), the distinction is that each document is viewed to be a mixture of topics and each word on a different topic, whereas NB assumes each document is on a *single topic*. However, each topic in LDA can essentially be viewed as a NB model.

Parameters and Hyperparameters

Document Term Matrix (DTM)

The data input to the model is in the form of a DTM where each row relates to a document, each column to a word/term and the values in the matrix are word frequencies, with word order being unimportant. Note that as the DTM contains every word for every document in the corpus it can become a large sparse matrix, which can be computationally heavy. It can be converted to a TF-IDF matrix but although sometimes useful, Blei et al. notes that weighting terms is not necessary for LDA to infer topics [6].

Number of Topics

The number of topics, k is a hyperparameter determined by the user and the iterations allow the process to be repeated until the topics are felt to make sense. K can be determined using Perplexity Scoring, discussed further under Model Performance.

Number of Iterations

A word will initially be assigned randomly to a topic and each iteration will allow words to be reassigned to topics. Over several iterations the topic mixtures for each document are determined and then elements of the topic are generated from the words that relate to that topic using a statistical method known as Collapsed Gibbs Sampling [2]. Note that Grid Search can be used to find the optimal value of hyperparameters in the model and the code section discusses this further.

Alpha and Beta

These hyperparameters refer to the topic and word densities. At low values of alpha, most of the weights in the topic distribution are placed on a single topic and when high it is more evenly spread. The same applies for the word weights for each topic beta.

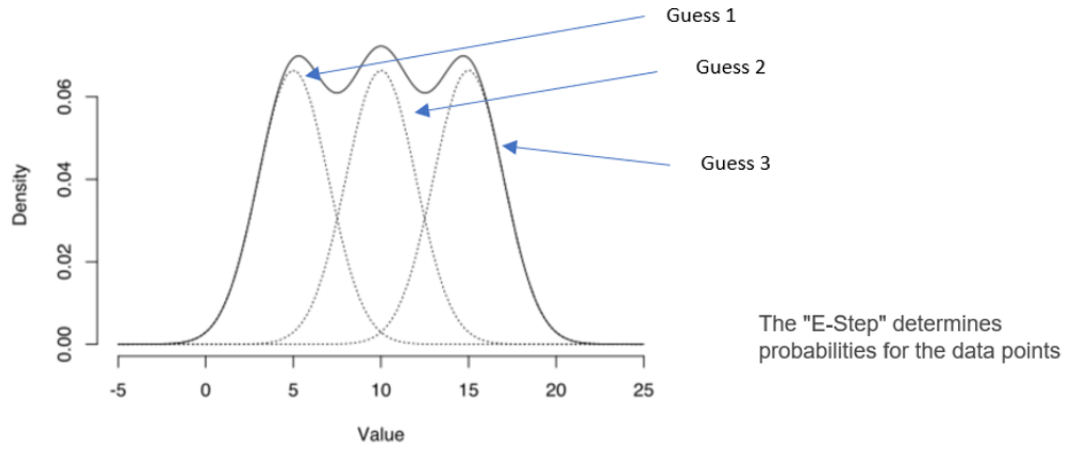
Fitting the Model

Similar to PLSA, the model is trained using the Expectation Maximisation (EM) algorithm. The EM is used as an approximation to the maximum likelihood function which does not perform well with latent variables. The values of parameters are such that they maximise the likelihood that the model produces the data that is actually observed. The method works by:

1. Choosing random values, creating a probability distribution known as the 'E-Step' or 'Expected' Distribution.
2. Using those values in the model, adjusting the probability distribution known as the 'M-Step'.
3. Repeating these steps until convergence when the distribution remains stable [11]. See Figure 7.

[2] *..an algorithm for obtaining a sequence of observations which are approximately from a specified multivariate probability distribution, when direct sampling is difficult. This sequence can be used to approximate the joint distribution to approximate ….some subset of the variables (for example, the unknown parameters or latent variables) [Wikipedia, 2020].*

Figure 7. The EM Algorithm



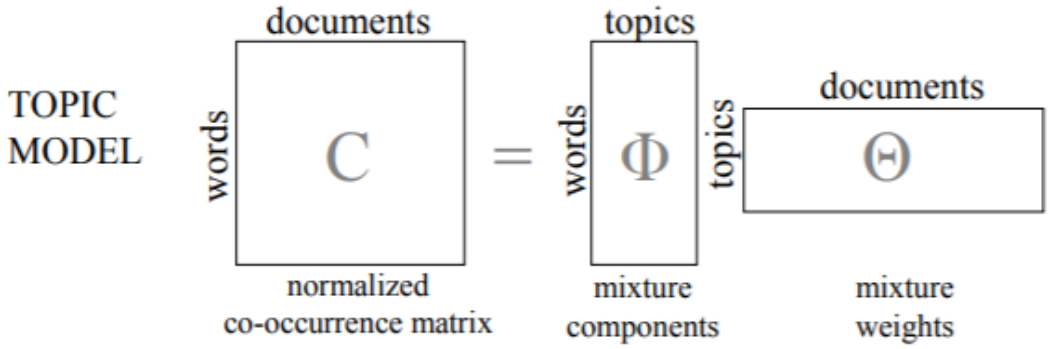
Source: Adapted from Statistics How To, 2015

The algorithm can incorrectly focus in on local maxima rather than global maxima, so multiple runs may be necessary. Also, it can be slow, especially with high dimensional data used for topic modelling.

Model Output

The model output is in the form of two matrices of probability distributions each of which sum to 1.0, including a Word Topic Matrix and a Topic Document Matrix. Figure 8 summarises this.

Figure 8. LDA Model Output Matrices



Source: Steyvers & Griffiths, 2004 [12]

Model Performance

Measuring performance of topic models can be complex. Evaluation methods can be grouped as manual and automatic and some of the methods are discussed below and demonstrated in the walkthrough section.

Manual evaluation

The aim of manual or qualitative evaluation is to ensure the topics are coherent, there is limited overlapping, topics are not missed, and the semantic structure of the corpus is captured and often referred to as 'human-in-the-loop' evaluation.

Top 'n' Words

The top 'n' words for each topic generated by the model are evaluated as to whether they make sense. This can be difficult to do in practice.

Visual Tools

PyLDAVis is a tool that firstly reduces dimensionality to two dimensions and then shows topics as circles, with size reflecting importance of a topic and proximity to other circles reflecting semantic relationships. Termite developed by Chuang et al. at Stanford University [13], uses tables to compare terms across and within topics and is based on pairwise conditional probabilities to determine how likely words are to occur together. Visual tools can be helpful but again interpretation can be difficult.

Word and Topic Intrusion

Word intrusion involves picking a topic and the top five words in that topic, then picking another word at the bottom of the list which is the top word in another topic. These are presented to a person to pick the odd one out. Topic intrusion involves picking the top three most likely topics and then one from the low probability topics and then show a person a section of the document and get them to pick the odd one out. Manual methods rely on human judgement, so can be prone to error and can be time consuming and subjective.

Automatic evaluation

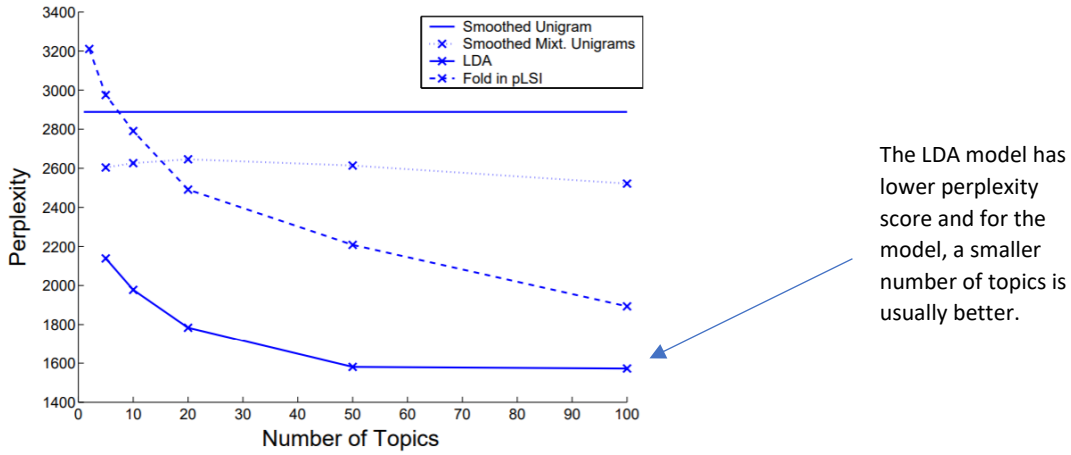
Perplexity and Log Likelihood Score

Perplexity is a very commonly used metric used to measure uncertainty in a model. The score evaluates how well the model performs against a test set with k number of topics based on the normalised log-likelihood:

Perplexity = (exp (-1 x log-likelihood per word))

A lower perplexity score is preferable. The score is actually not particularly useful on its own but if the model is run with varying values of k, the scores can be compared, and it can be used to compare topic models. Blei [6] notes that lower perplexity indicates a better generalization performance and Figure 9 shows an example of the score for different values of k for various models.

Figure 9. Perplexity Score



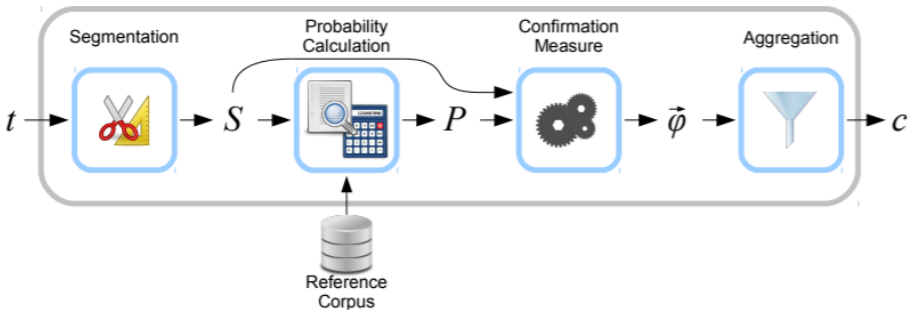
Source: Blei et al., 2003 [6]

Perplexity scores have been criticised for not adequately capturing the semantics of the corpus and optimising against perplexity may not result in topics that are easily interpretable. However, they remain a widely used metric.

Topic Coherence

A coherence score measures the relative distance between the top N words within a topic and is often viewed as superior to perplexity. The method is presented as separate components that are combined and can be viewed as a pipeline. Figure 10 illustrates the pipeline presented by Roder et al. [14].

Figure 10. Topic Coherence Pipeline



Source: Roder et al., 2015 [14]

From the diagram:

- The top n words by frequency are segmented into pairs of words as subsets and probabilities calculated based on a reference corpus.
- These words and probabilities are passed to the confirmation measure which computes how strongly one words supports the other in the pair using an intrinsic or extrinsic measure.
- Scores are aggregated into a coherence value based on the average.

$$\text{Coherence} = \sum_{i<j} \text{score}(w_i, w_j)$$

There are a range of confirmation measures and two commonly used methods include intrinsic UMass measure [15] and extrinsic measure UCI [16]. A detailed discussion of these measures is not covered here but UMAs compares a word to preceding and succeeding words in an ordered word set and the UCI compares every word with every other word.

There is the possibility that larger models will pick up 'junk' information and be judged less coherent when in fact they do pick up useful information. Relying on automatic evaluation by interpreting standalone scores can be difficult but can be useful for monitoring progress of training [17]. A combination of manual and automatic methods is usually the best option.

Model Extensions

Online Method

The LDA model was extended by Hoffman, Blei and Bach in 2010, [18] to accommodate the use of massive document collections and streaming data, based on online stochastic optimization. This 'online variational Bayes algorithm' was found to be much faster for topic models on huge datasets.

Hierarchical Dirichlet Process

Instead of having the number of topics be fixed, the Hierarchical Dirichlet Process is an extension of LDA, where the number of topics is determined automatically and the more documents there are, the more topics are generated. The set of possible topics is determined and then a distribution for each document is sampled from this set [20].

Neural Network and Hybrid Models

Word2vec, a model introduced by Mikolov et al. in 2013 is a two-layer neural network model based on vectorised words which is able to capture context and semantics [22]. These more complex models have become increasingly popular due to their higher performance. Newer hybrid models such as Chris Moody's LDA2vec combine traditional LDA with word embedding by building document representations on top of word embeddings [23].

Walkthrough with SciKit Learn LDA Module

Methodology and Dataset

In this section, the LDA model is demonstrated using the DBpedia Ontology Classification Dataset 2014 [19][21]. The data is composed of 560,000 training and 70,000 test samples over 14 topic classes. See Figure 11.

Note: Although the dataset is effectively labelled, topic models are used with unlabelled data and are unsupervised so comparisons with classification models such as Naïve Bayes are not considered.

Figure 11. Topic Classes

Class	Total	Train	Test
Company	63,058	40,000	5,000
Educational Institution	50,450	40,000	5,000
Artist	95,505	40,000	5,000
Athlete	268,104	40,000	5,000
Office Holder	47,417	40,000	5,000
Mean Of Transportation	47,473	40,000	5,000
Building	67,788	40,000	5,000
Natural Place	60,091	40,000	5,000
Village	159,977	40,000	5,000
Animal	187,587	40,000	5,000
Plant	50,585	40,000	5,000
Album	117,683	40,000	5,000
Film	86,486	40,000	5,000
Written Work	55,174	40,000	5,000

Source: Text Understanding From Scratch [19]

The code is at this Colab Link.

Importing Libraries and Dataset

```
!pip install pyldavis
!pip install -U gensim
```

```
# Data manipulation
import numpy as np
import pandas as pd

# Visualisation
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
%matplotlib inline
import pyLDAvis
import pyLDAvis.sklearn

# Modelling and feature engineering
import sklearn.decomposition
from sklearn.decomposition import LatentDirichletAllocation
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import GridSearchCV
from sklearn.manifold import TSNE
import gensim

# Text processing
import nltk
```

Pre-processing

```
# Read in datasets
data = pd.read_csv('/content/MyDrive/My Drive/dbpediatrain.csv',names = ["label","title","text"])
data_test = pd.read_csv('/content/MyDrive/My Drive/dbpediatest.csv',names = ["label","title","text"])
```

```
# Take a sample of the larger dataset to work with as training set.
data = data.sample(8000, random_state = 0)

# Take a sample of the original test set to use to predict
test_data = data_test.sample(2000, random_state = 42)
test_data.reset_index(inplace = True, drop = True)
```

Examining the Dataset

```
# Are there any null entries and what are the features?
data.shape
```

```
(8000, 3)
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8000 entries, 245601 to 476345
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   label   8000 non-null   int64 
 1   title   8000 non-null   object
 2   text    8000 non-null   object
dtypes: int64(1), object(2)
memory usage: 250.0+ KB
```

```
# Examine sample of data
data.sample(5)
```

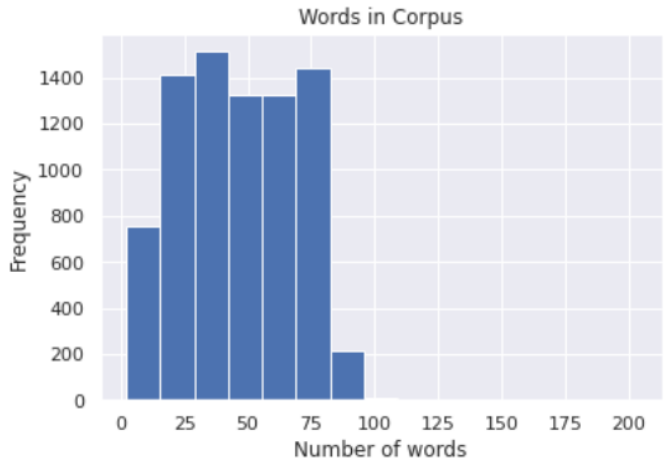
	label	title	text
474677	12	Sgrech	Sgrech is a 'best of' compilation album by th...
463024	12	Time After Time (Etta James album)	Time After Time is the twenty-first studio al...
460838	12	All the Pretty Little Horses	All the Pretty Little Horses (TheInmostLightl...
465384	12	Shut Up and Dance (The Dance Mixes)	Shut Up and Dance: The Dance Mixes was a Paul...
122690	4	Sérvulo Barbosa Bessa	Sérvulo Barbosa Bessa or simply Sérvulo (born...

```
# Look at one document
data.text.iloc[300]
```

```
' Freeworld Entertainment was a record label formed in 1997 by Dallas Austin and Kevin Czinger.'
```

Note: only the text column is extracted as this is an unsupervised technique.

```
# Number of words in document
def doc_word(text):
  doc_word = round(text.str.split().apply(len),2)
  return doc_word

length_doc = doc_word(data.text)

# Histogram of length of each document
plt.hist(length_doc,bins = 15)
plt.title("Words in Corpus")
plt.xlabel("Number of words")
ax = plt.ylabel("Frequency")
```




```
print("Maximum length:",length_doc.max())
print("Minimum_length:",length_doc.min())
print("Average_length:",round(length_doc.mean(),2))
```

```
Maximum length: 203
Minimum_length: 2
Average_length: 46.09
```

```
# Look for outliers
sort_doc = length_doc.sort_values(ascending = False)
sort_doc[0:13]
```

```
270940    203
54812     172
9658      148
552696    118
207675    118
380810    114
108194    107
110176    107
61739     106
58962     106
146207    100
535959    100
498466     98
Name: text, dtype: int64
```

Analysis:

- The average length of document is 46 words within a range of 2 to 203
- There are 12 documents over 100 words and one over 200.
- Apart from the few outliers, the rest of the data seems roughly normally distributed.

Creating the Document Term Matrix

Bag of Words - Unigrams

The LDA model takes in a bag-of-words based on word counts rather than a weighted td-idf as noted Blei notes that the model does not require weights. Various parameters can be set within the CountVectorizer to tweak the words that are included.

```
# Tokensize, lower case, remove accents and stopwords. Max_df - ignores words found in more than
# 95% documents and min_df ignores those in less than 5 documents. Only unigrams included.

vectorizer = CountVectorizer(strip_accents="ascii",max_df = 0.95,min_df = 5,stop_words='english')
```

```
# The vectorized dataset is created by running fit an transform on the count_vectorizer object.
# The bag of words has documents as rows and unique words as columns.
vec = vectorizer.fit_transform(data.text)
print(vec.shape)
```

```
(8000, 5528)
```

```
# The bag of words or document term matrix is a sparse matrix.
print(type(vec))
#Percentage of cells with non-zero values
data_dense = vec.todense()
print("Sparsicity: ", ((data_dense > 0).sum()/data_dense.size)*100, "%")
```

```
<class 'scipy.sparse.csr.csr_matrix'>
Sparsicity:  0.3339114507959479 %
```

Two thirds of the matrix is zero.

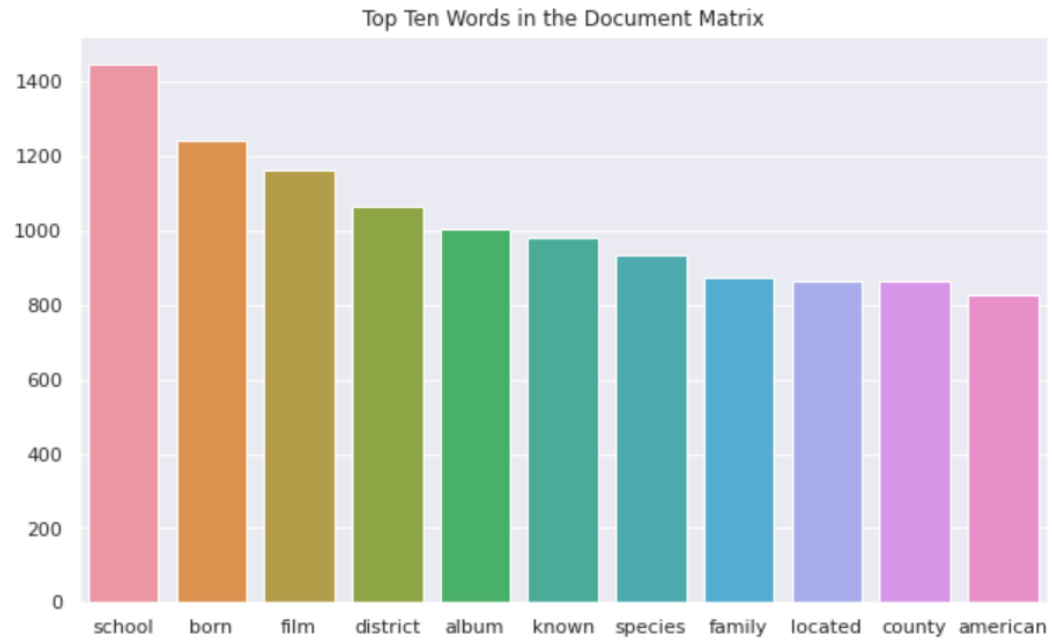
```
# Extract words from the data vector
feature_names = vectorizer.get_feature_names()
len(feature_names)
```

```
5528
```

```
# Find top words in bag of words with frequencies
all_words = vec.sum(axis=0)
freq_word = [(word, all_words[0, idx]) for word, idx in vectorizer.vocabulary_.items()]
freq_word =sorted(freq_word, key = lambda x: x[1], reverse=True)
top_ten_freq = freq_word[0:11]
top_ten_freq
```

```
[('school', 1450),
 ('born', 1241),
 ('film', 1166),
 ('district', 1066),
 ('album', 1005),
 ('known', 981),
 ('species', 936),
 ('family', 875),
 ('located', 867),
 ('county', 866),
 ('american', 828)]
```

```
# Extract and plot top words
word_name = []
word_count = []
for word in top_ten_freq:
  w = word[0]
  c = word[1]
  word_name.append(w)
  word_count.append(c)

plt.figure(figsize = (10,6))
plt.title("Top Ten Words in the Document Matrix")
ax = sns.barplot(x=word_name,y=word_count)
```



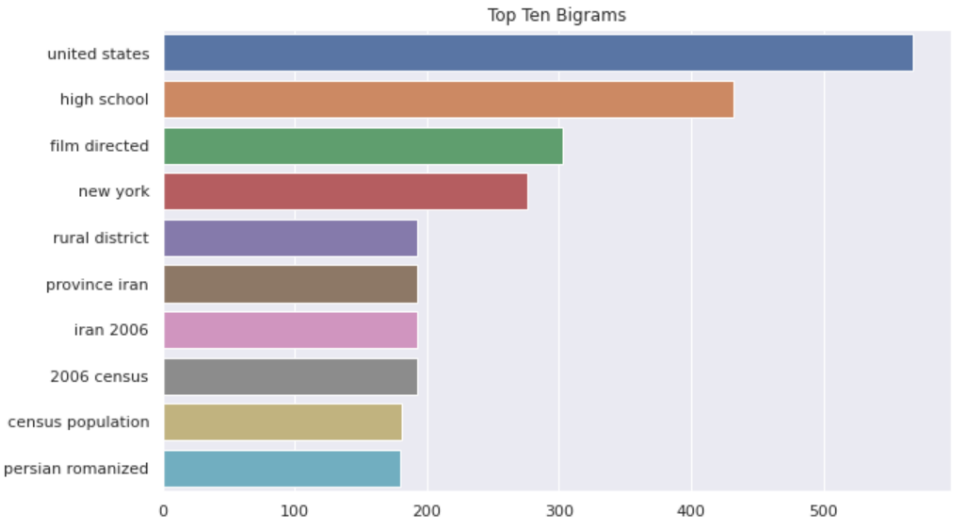

```
# Examine a slice from the dtm
feature_names[2500:2510]
```

```
['herb',
 'herbaceous',
 'herbert',
 'herbs',
 'heritage',
 'herman',
 'hermann',
 'hero',
 'herzegovina',
 'hesperiidae']
```

Bigrams

```
# Find the top bigrams - words that occur together
def bigram(text,n):
    vectorizer_2 = CountVectorizer(strip_accents="ascii",ngram_range = (n,n),
                            max_df = 0.95,min_df = 5,stop_words='english')
    vec_ngram = vectorizer_2.fit_transform(text)
    vec_sum = vec_ngram.sum(axis=0)
    freq_ngram = [(word, vec_sum[0, idx])
                  for word, idx in vectorizer_2.vocabulary_.items()]
    freq_ngram_sorted =sorted(freq_ngram, key = lambda x: x[1], reverse=True)
    return freq_ngram_sorted[:10]
```

```
top_bigrams = bigram(data["text"],2)
x,y=map(list,zip(*top_bigrams))
plt.figure(figsize = (10,6))
plt.title("Top Ten Bigrams")
ax = sns.barplot(x=y,y=x)
```



Analysis:

- The top unigrams show the word 'school' occurs across many of the documents and it is likely that it will be shared across topics.
- United States is the most frequent bigram occurring in more than 500 documents so this should be borne in mind when analysing the output of the model, if these two words occur together in word frequency.

Fitting the Base Model

The LDA model in Scikit Learn:

class `sklearn.decomposition.` **`LatentDirichletAllocation`**(*n_components=10, doc_topic_prior=None, topic_word_prior=None, learning_method='batch', learning_decay=0.7, learning_offset=10.0, max_iter=10, batch_size=128, evaluate_every=-1, total_samples=1000000.0, perp_tol=0.1, mean_change_tol=0.001, max_doc_update_iter=100, n_jobs=None, verbose=0, random_state=None*) ¶ [source]

Source: Scikit learn documentation

```
# Topics set to 14 as per dataset, in practice usually unknown and estimated.
# Doc_topic and word_topic priors (alpha and beta) are doc-topic and word-topic
# densities. Learning method Online Variational Bayes and iterations to 15 as
# dataset is large.

topics = 14
iterations = 15
method = "online"
lda_model = LatentDirichletAllocation(n_components = topics,
                                      max_iter = iterations,random_state = 42,
                                      learning_method = method)
```

```
# Fit the model and transform
lda = lda_model.fit(data_vector)
lda_base = lda.transform(data_vector)
```

```
# Perplexity is defined as exp(-1. * log-likelihood per word)  - the lower the
# better. How 'perplexed' is the model with the words presented.
base_perplex = lda.perplexity(vec)
print("Base model perplexity:",base_perplex)
```

```
Base model perplexity: 1831.0481140174247
```

```
# Log likelihood - the higher the better. Difficult to judge as just based on
# training data
base_log = lda.score(vec)
print("Base model log likelihood:", base_log)
```

```
Base model log likelihood: -1278404.061221314
```

Examining the Output

Document-Topic Distribution (DTM)

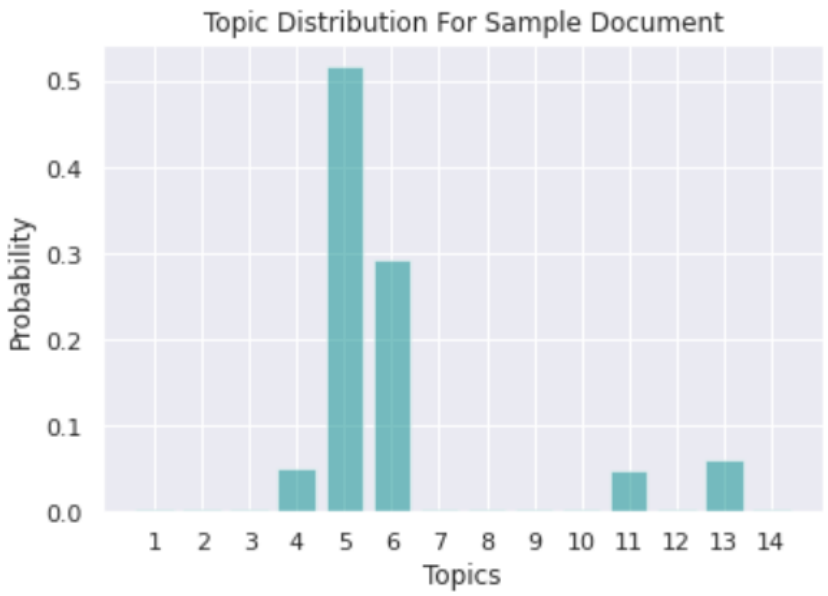
```
# Document-topic matrix shape confirmed to be as expected with documents as
# rows and topics as columns
dtm = lda_base
dtm.shape
```

```
(8000, 14)
```

```
# Example topic shows array of probabilities for a sample document over 14 topics
dtm[10]
```

```
array([0.00324676, 0.0032468 , 0.00324675, 0.05189128, 0.51730476,
       0.2924415 , 0.00324675, 0.00324679, 0.00324675, 0.00324679,
       0.04941508, 0.00324675, 0.05972648, 0.00324676])
```

```
# Plot on a graph to confirm
topics = [1,2,3,4,5,6,7,8,9,10,11,12,13,14]
probs = list(dtm[10])
plt.bar(topics, probs, align='center', alpha=0.5, color = "darkcyan")
plt.xticks(topics)
plt.ylabel('Probability')
plt.xlabel("Topics")
ax = plt.title('Topic Distribution For Sample Document')
```




```
# Create dataframe showing topic-document distributions totalling to one and dominant topics listed
df_dtm = pd.DataFrame(data=dtm, columns=topics)

# Add columns for top topic and the total sum of probabilities for each document over the topics.
df_dtm["Topic"] = df_dtm.idxmax(axis = 1)
df_dtm["Prob"] = df_dtm.iloc[:,0:14].sum(axis=1)
df_dtm.shape
```

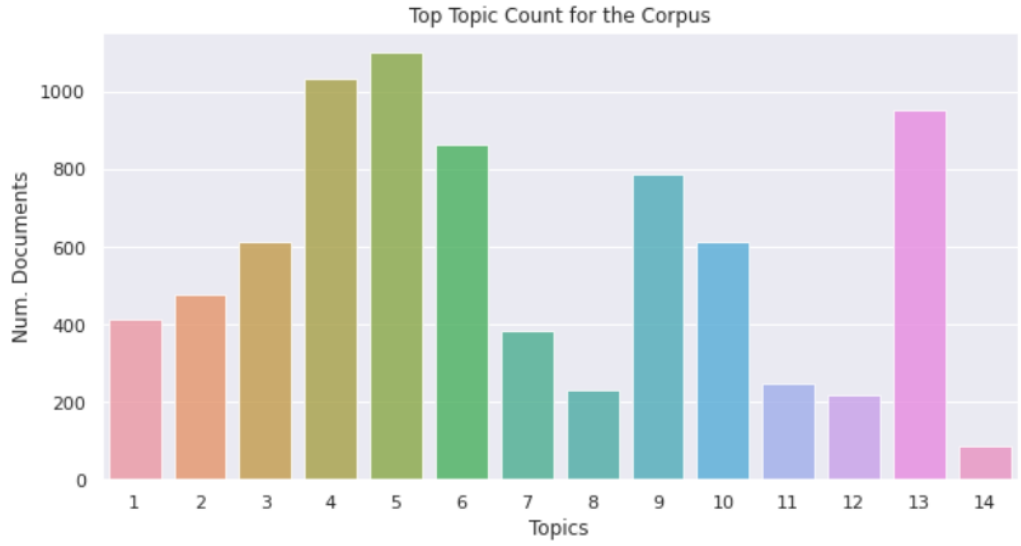
```
(8000, 16)
```

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	Topic	Prob
0	0.002041	0.002041	0.187793	0.002041	0.335662	0.002041	0.002041	0.115238	0.002041	0.002041	0.002041	0.002041	0.340899	0.002041	13	1.0
1	0.003106	0.003106	0.003106	0.095146	0.003106	0.003106	0.003106	0.266455	0.003106	0.512441	0.003106	0.003106	0.094901	0.003106	10	1.0
2	0.066457	0.415241	0.223984	0.004202	0.004202	0.004202	0.004202	0.004202	0.004202	0.004202	0.252301	0.004202	0.004202	0.004202	2	1.0
3	0.056391	0.003759	0.003759	0.003759	0.003759	0.537157	0.003759	0.056391	0.132595	0.075076	0.003759	0.003759	0.003759	0.112315	6	1.0
4	0.004202	0.004202	0.004202	0.004202	0.004202	0.743568	0.004202	0.004202	0.004202	0.206012	0.004202	0.004202	0.004202	0.004202	6	1.0

```
# Obtain value counts per topic - note: we don't know what the topics are at this point
x = pd.DataFrame(df_dtm["Topic"].value_counts())
x.T
```

	5	4	13	6	9	10	3	2	1	7	11	8	12	14
Topic	1098	1032	951	864	785	613	610	475	411	382	246	230	218	85

```
# Plot top topic for all documents
plt.figure(figsize=(10,5))
sns.barplot(x.index, x.Topic, alpha=0.8)
plt.title("Top Topic Count for the Corpus")
plt.ylabel("Num. Documents")
ax = plt.xlabel("Topics")
```



Word-Topic Distribution

```
# Word-topic distributions matrix has 14 topics as rows and number of unique words as columns
wtm = lda_model.components_
wtm.shape
```

```
(14, 5528)
```

```
# Examine example topic with word weights across all words.
wtm[11]
```

```
array([8.37616531e-06, 8.37619762e-06, 8.37653815e-06, ...,
       6.78291838e-04, 8.37620094e-06, 8.37616035e-06])
```

```
# Sum of all word weights
wtm[11].sum()
```

```
8527.603449001777
```

```
# Transform weights to probabilities that sum to one not done automatically in Scikit Learn
wtm/=wtm.sum(axis = 1)[:,np.newaxis]
```

```
# Extract top words index positions for a sample topic based on weights
example = wtm[11]
top_words_example = example.argsort()[-14:]
top_words_example
```

```
array([ 458, 2886, 3031,  626, 3914, 1748, 1479, 5413, 5330, 4734, 3557,
       1837, 2897, 3311])
```

```
# Reference index positions back to feature names
for i in top_words_example:
  print(vectorizer.get_feature_names()[i])
```

```
administrative
kilometres
lies
approximately
poland
district
county
west
village
south
north
east
km
mi
```

This topic seems to be about villages

```
# Function to print out top words (code from Scikit learn documentation)

n_top_words = 10

def top_words(lda_model, feature_names, n_top_words):
    for topic_idx, topic in enumerate(lda_model.components_):
        message = "Topic #%d: " % topic_idx
        message += " ".join([feature_names[i]
                             for i in topic.argsort()[:-n_top_words - 1:-1]])
        print(message)
    print()
```

```
# Run function
tf_feature_names = vectorizer.get_feature_names()
top_words(lda_model, tf_feature_names, n_top_words)
```

```
Topic #0: family species plant america india endemic moth indian habitat tropical
Topic #1: book published novel series story written fiction short magazine japanese
Topic #2: river class navy war united states world ship built ii
Topic #3: species genus family lake known south mountain native long common
Topic #4: film company directed based american starring science journal italian founded
Topic #5: born played football league professional player april october currently american
Topic #6: district village province 2006 population county census rural iran persian
Topic #7: church st building style century york john parish non england
Topic #8: album released music band studio records singer rock label american
Topic #9: member politician new served born york american house party state
Topic #10: built french aircraft designed services car single produced light corporation
Topic #11: mi km east north south village west county district poland
Topic #12: school located high university college historic national county public house
Topic #13: new thomas summer following zealand british robert including olympics australia
```

```
# Dataframe to show word-topic probability distributions for each word over
# every topic with total probabilities summing to one
ind = ["Topic1","Topic2","Topic3","Topic4","Topic5","Topic6","Topic7","Topic8","Topic9","Topic10",
       "Topic11","Topic12","Topic13","Topic14"]
df_word_prob = pd.DataFrame(wtm, columns = [vectorizer.get_feature_names()], index = ind)
df_word_prob["Prob"] = df_word_prob.iloc[:,:].sum(axis=1)
df_word_prob.head()
```

	00	01	08	10	100	1000	10000	101	102	103	104
Topic1	0.000008	0.000008	0.000008	0.000008	0.000008	0.000008	0.000008	0.000008	0.000008	0.000008	0.000008
Topic2	0.000006	0.000006	0.000006	0.000006	0.000006	0.000006	0.000513	0.000006	0.000006	0.000006	0.000006
Topic3	0.000006	0.000006	0.000006	0.000896	0.001412	0.000006	0.000006	0.000006	0.000006	0.000006	0.000006
Topic4	0.000004	0.000004	0.000004	0.001846	0.000004	0.000004	0.000004	0.000004	0.000004	0.000004	0.000004
Topic5	0.000413	0.000004	0.000004	0.000004	0.000004	0.000004	0.000004	0.000004	0.000004	0.000004	0.000004

5 rows × 5529 columns

```
# Looking at a sample of words with probabilities
df_word_prob.iloc[:,2500:2510].head()
```

	herb	herbaceous	herbert	herbs	heritage	herman	hermann	hero	herzegovina	hesperiidae
Topic1	0.000008	0.001660	0.000008	0.000008	0.000008	0.000008	0.000008	0.000008	0.000008	0.000008
Topic2	0.000006	0.000006	0.000006	0.000006	0.000006	0.000006	0.000006	0.000006	0.000006	0.000006
Topic3	0.000006	0.000006	0.000006	0.000006	0.000006	0.000006	0.000006	0.000709	0.000006	0.000006
Topic4	0.002395	0.000004	0.000004	0.000361	0.000004	0.000004	0.000004	0.000004	0.000004	0.000508
Topic5	0.000004	0.000004	0.000004	0.000004	0.000004	0.000004	0.000004	0.000004	0.000004	0.000004

The Top Ten Words Per Topic

```
# Create word topic matrix dataframe

# Sort top ten words by weight and then use to look up words
all_words = np.array(feature_names)
topic_top = []
for topic_weights in wtm:
  top_word_ind = (-topic_weights).argsort()[:10]
  topic_top.append(all_words.take(top_word_ind))

# Create topic - keywords Dataframe
cols = ["Word1","Word2","Word3","Word4","Word5","Word6","Word7","Word8","Word9","Word10"]
ind = ["Topic1","Topic2","Topic3","Topic4","Topic5","Topic6","Topic7","Topic8","Topic9","Topic10",
       "Topic11","Topic12","Topic13","Topic14"]

topic_word_matrix = pd.DataFrame(topic_top,columns = cols ,index = ind)
topic_word_matrix
```

	Word1	Word2	Word3	Word4	Word5	Word6	Word7	Word8	Word9	Word10
Topic1	family	species	plant	america	india	endemic	moth	indian	habitat	tropical
Topic2	book	published	novel	series	story	written	fiction	short	magazine	japanese
Topic3	river	class	navy	war	united	states	world	ship	built	ii
Topic4	species	genus	family	lake	known	south	mountain	native	long	common
Topic5	film	company	directed	based	american	starring	science	journal	italian	founded
Topic6	born	played	football	league	professional	player	april	october	currently	american
Topic7	district	village	province	2006	population	county	census	rural	iran	persian
Topic8	church	st	building	style	century	york	john	parish	non	england
Topic9	album	released	music	band	studio	records	singer	rock	label	american
Topic10	member	politician	new	served	born	york	american	house	party	state
Topic11	built	french	aircraft	designed	services	car	single	produced	light	corporation
Topic12	mi	km	east	north	south	village	west	county	district	poland
Topic13	school	located	high	university	college	historic	national	county	public	house
Topic14	new	thomas	summer	following	zealand	british	robert	including	olympics	australia

```
# Try to infer the topics based on original dataset - some are very fuzzy!!
Inferred = ["Plant","Written","Transport","Animal", "Film",
            "Athlete","Village","Building","Album","Office Holder",
            "Company","Natural","Education","Artist"]


topic_word_matrix["Inferred Topic"] = Inferred
topic_word_matrix
```

	Word1	Word2	Word3	Word4	Word5	Word6	Word7	Word8	Word9	Word10	Inferred Topic
Topic1	family	species	plant	america	india	endemic	moth	indian	habitat	tropical	Plant
Topic2	book	published	novel	series	story	written	fiction	short	magazine	japanese	Written
Topic3	river	class	navy	war	united	states	world	ship	built	ii	Transport
Topic4	species	genus	family	lake	known	south	mountain	native	long	common	Animal
Topic5	film	company	directed	based	american	starring	science	journal	italian	founded	Film
Topic6	born	played	football	league	professional	player	april	october	currently	american	Athlete
Topic7	district	village	province	2006	population	county	census	rural	iran	persian	Village
Topic8	church	st	building	style	century	york	john	parish	non	england	Building
Topic9	album	released	music	band	studio	records	singer	rock	label	american	Album
Topic10	member	politician	new	served	born	york	american	house	party	state	Office Holder
Topic11	built	french	aircraft	designed	services	car	single	produced	light	corporation	Company
Topic12	mi	km	east	north	south	village	west	county	district	poland	Natural
Topic13	school	located	high	university	college	historic	national	county	public	house	Education
Topic14	new	thomas	summer	following	zealand	british	robert	including	olympics	australia	Artist

The model has identified most of the topics but there is still some fuzziness with words occurring over different topics.

Visualising The Model

Using Dimensionality Reduction

```
[65] # Document term matrix shape
     dtm.shape
```

```
(8000, 14)
```

```
# Reduce dimensionality of data to view on a graph using t-stochastic neighbor embedding (TSNE)
tsne = TSNE(n_components = 2, init = 'pca', random_state = 42)
```

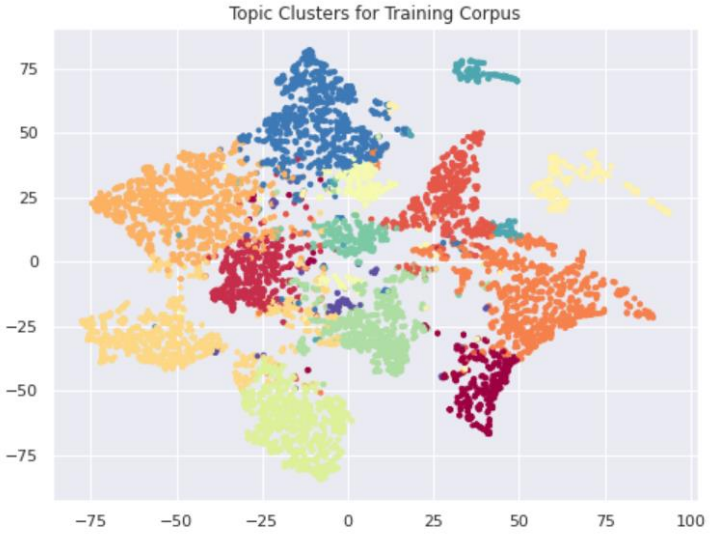
```
lda_tsne = tsne.fit_transform(dtm)
```

```
# Document term matrix reduced to 2 dimensions
df_tsne = pd.DataFrame(lda_tsne)
df_tsne.shape
```

```
(8000, 2)
```

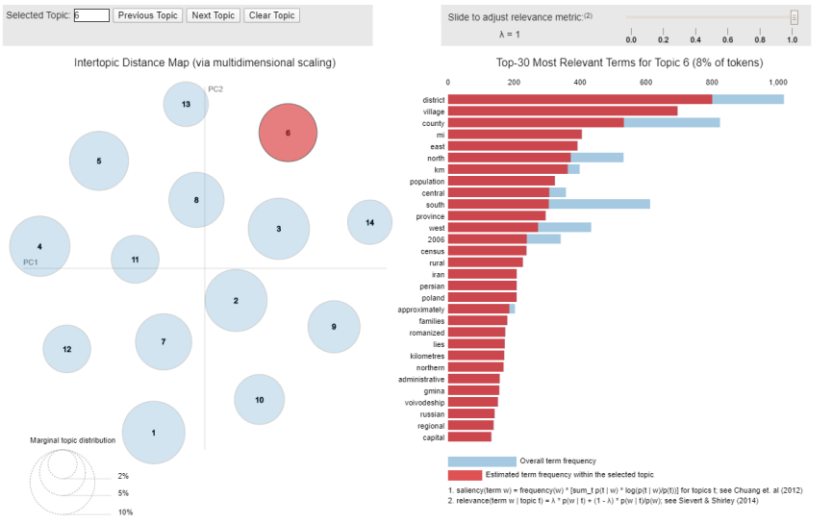
```
plt.subplots(figsize = (8,6))
plt.scatter(df_tsne[0],df_tsne[1],c = df_dtm["Topic"],cmap = plt.cm.Spectral,s=9)
ax = plt.title("Topic Clusters for Training Corpus")
```

The data is in clusters but there is some overlap with topics which reflects the fuzziness of the word-topic matrix.



PyLDAVis

```
[71] # Create the interactive visualisation
     pyLDAvis.enable_notebook()
     p = pyLDAvis.sklearn.prepare(lda_model,data_vector,vectorizer,mds = "tsne")
     p
```



A good model will have clearly defined large circles and not clustered together. If the model has too many topics there will be a lot of overlaps as the topics have many words in common and the distance between circles shows how different the topics are from each other.

- On the right panel - choose a word and it will show which topics are associated with the word e.g. 'river' is associated with one topic, so the other circles will disappear.
- On the left panel - the relative size of a circle indicates where most of the occurrences of a particular word appears.
- Choose a topic and the chart will show how often each term occurs in that topic in red

See Colab Notebook to view full functionality of the model.

Model Performance and Hyperparameter Tuning

Scikit Learn LDA models can be compared using Perplexity and Log likelihood with higher log likelihood and lower the perplexity scores better. The effect of changing some of the parameters is examined here but is not exhaustive due to project length constraints.

Grid Search and Randomized Grid Search

Grid Search was attempted (code below) to find optimal parameters for topics and learning decay, which are two of the most important parameters in the model. This was computationally extremely demanding so randomized grid search was used initially and then each of the parameters were investigated separately.

```
from sklearn.model_selection import RandomizedSearchCV
search_params = {'n_components': [5, 10, 15, 20, 25], 'learning_decay': [.5, .7, .9], 'doc_topic_prior':[0.1,0.5,1.0],'topic_word_prior':[0.1,0.5,1.0]}
lda = LatentDirichletAllocation(learning_method = 'online')
model = RandomizedSearchCV(lda, param_distributions = search_params, random_state=0)
```

```
model.best_params_
```

```
{'doc_topic_prior': 0.5,
 'learning_decay': 0.5,
 'n_components': 5,
 'topic_word_prior': 0.1}
```

```
# Trying these parameters in the model and using to derive perplexity
lda_grid = LatentDirichletAllocation(n_components = 5,doc_topic_prior = 0.5,topic_word_prior = 0.1,learning_method="online",
                                     max_iter = 15,learning_decay = 0.5)
model_grid = lda_grid.fit(vec)
model_grid.perplexity(vec)
```

```
1858.8494615208285
```

Analysis: This model returns a slightly higher perlexity than the base model.

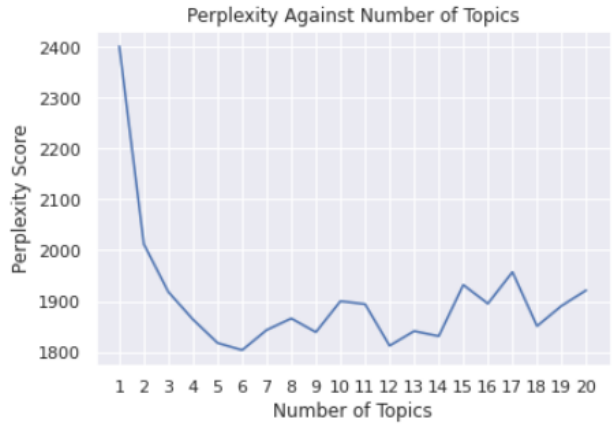
Optimum Number of Topics

```
# Function to obtain perplexity and likelihood for different numbers of topics
def scoring(topic):

  lda2_model = LatentDirichletAllocation(n_components = topic,
                                         max_iter = 15,random_state = 42,
                                         learning_method = 'online')
  lda2 = lda2_model.fit(vec)
  perplex2 = lda2_model.perplexity(vec)
  like2 = lda2_model.score(vec)

  return perplex2,like2
```

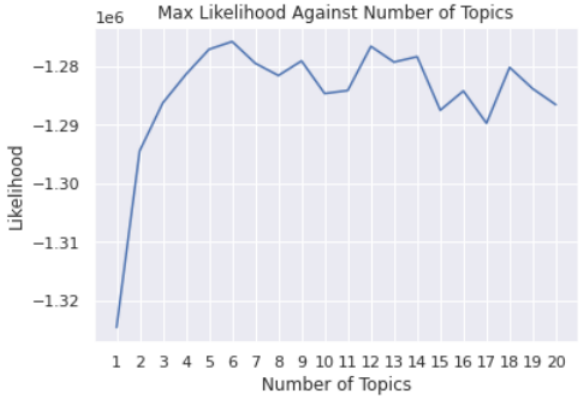
```
test = pd.DataFrame([topic_num,perplexity,log_score]).
test.columns = (["Topics","Perplexity","Log"])
test.head(10)
```

	Topics	Perplexity	Log
0	1.0	2401.690429	-1.324568e+06
1	2.0	2012.251910	-1.294462e+06
2	3.0	1917.937502	-1.286293e+06
3	4.0	1863.884691	-1.281429e+06
4	5.0	1817.681225	-1.277157e+06
5	6.0	1803.556765	-1.275830e+06
6	7.0	1843.268580	-1.279536e+06
7	8.0	1865.943240	-1.281616e+06
8	9.0	1838.900425	-1.279132e+06
9	10.0	1899.925273	-1.284688e+06

```
# Plot perplexity against number of topics
plt.plot(topic_num,perplexity)
plt.title("Perplexity Against Number of Topics")
plt.xlabel("Number of Topics")
plt.xticks(topic_num)
ax = plt.ylabel("Perplexity Score")
```




```
# Plot likelihood against number of topics
plt.plot(topic_num,log_score)
plt.title("Max Likelihood Against Number of Topics")
plt.xlabel("Number of Topics")
plt.xticks(topic_num)
ax = plt.ylabel("Likelihood")
```



Higher Numbers of Topics

```
# Set to 30,50 and 100 topics
r1 = scoring(30)
r2 = scoring(50)
r3 = scoring(100)
print("30 topics:",r1)
print("50 topics:",r2)
print("100 topics:",r3)
```

```
30 topics: (1984.492974283586, -1292098.2352548614)
50 topics: (2278.6569664205913, -1315619.1887880042)
100 topics: (2994.631962490516, -1362114.7895291573)
```

Analysis:

- Perplexity falls and log-likelihood increases sharply up to 5 topics.
- The lowest perplexity and highest log-likelihood is at 6 topics. This could be because several topic categories share words. The Word-Topic matrix clearly illustrates this as it struggles to separate some of the words into distinct topics and the TSNE graph shows several clusters overlapping. Interestingly, perplexity and likelihood dip and peak respectively at 12 topics as granularity in the data increases.
- Above 12 topics perplexity seems to oscillate and running this model with 30,50 and 100 topics it tends to increase.
- The 14 topic model is retained in order to be able to compare results with the original dataset, since 6 topics would clearly not capture the different topics in the data.

Varying Learning Decay

The Learning decay controls the learning rate in the online learning method which determines the step size at each iteration as the model converges to the minimum.

```
# Learning Decay varies the learning rate over time.
# Function to try different learning rate - must be between 0.5 and 1.0

def learn(topics,rate):

  lda3_model = LatentDirichletAllocation(n_components = topics, max_iter = 15,
                                         learning_method = "online",
                                         learning_decay = rate)

  lda3 = lda3_model.fit_transform(vec)

  perplex3 = lda3_model.perplexity(vec)
  like3 = lda3_model.score(vec)
```

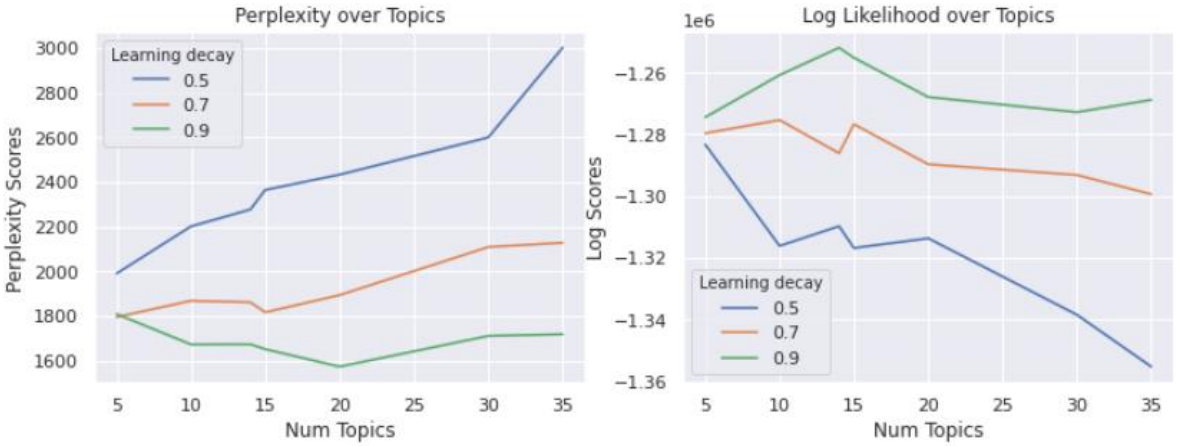
```
# Learning rate 0.5
rate = 0.5
topics = [5,10,14,15,20,30,35]
p1 = []
l1 = []
for i in topics:
  p1.append(learn(i,rate)[0])
  l1.append(learn(i,rate)[1])
```

Repeat the function above for learning rates of 0.7 and 0.9.

```
# Perplexity dataframe
decay = pd.DataFrame([topics,p1,p2,p3]).T
decay.columns = (["Topics","0.5","0.7","0.9"])
decay
```

	Topics	0.5	0.7	0.9
0	5.0	1990.918365	1796.137955	1809.910171
1	10.0	2201.485315	1868.913891	1673.864738
2	14.0	2277.205980	1862.182070	1674.465891
3	15.0	2364.689870	1817.407804	1652.695220
4	20.0	2432.853770	1894.339820	1574.445529
5	30.0	2599.228333	2109.281400	1711.739760
6	35.0	3000.425140	2128.284778	1719.172867

```
# Plot the three learning rates on a graph for each number of topics
plt.figure(figsize=(12,4))
plt.subplot(1, 2, 1)
plt.plot(topics, p1, label='0.5')
plt.plot(topics, p2, label='0.7')
plt.plot(topics, p3, label='0.9')
plt.title("Perplexity over Topics")
plt.xlabel("Num Topics")
plt.ylabel("Perplexity Scores")
plt.legend(title='Learning decay', loc='best')

plt.subplot(1, 2, 2)
plt.plot(topics, l1, label='0.5')
plt.plot(topics, l2, label='0.7')
plt.plot(topics, l3, label='0.9')
plt.title("Log Likelihood over Topics")
plt.xlabel("Num Topics")
plt.ylabel("Log Scores")
plt.legend(title='Learning decay', loc='best')
plt.show()
```



Analysis:

- Learning decay of 0.9 appears to be the optimal reaching a minimum perplexity at 20 topics and maximum log-likelihood at 14 topics.

Document and Topic Density

Doc_topic_prior represents document-topic density - higher number means documents are made up of more topics. Topic_word_prior represents topic-word density - higher number means topics are made up of most of the words in the corpus. These are the alpha and beta hyperparameters in the model.

Default = 1/number of components(topics)

```
# Function to change density (priors):
def priors(t,p):
  lda4_model = LatentDirichletAllocation(n_components = t,doc_topic_prior = p, topic_word_prior = p,max_iter = 15,
                                         learning_method = "online",learning_decay = 0.9)
  lda4 = lda4_model.fit_transform(vec)

  perplex4 = lda4_model.perplexity(vec)
  like4 = lda4_model.score(vec)

  return perplex4,like4
```

```
# Priors = 0.1
topics = [5,10,14,15,20,30]
pp1 = []
ll1 = []
for i in topics:
  pp1.append(priors(i,0.1)[0])
  ll1.append(priors(i,0.1)[1])
```

Repeat the function for priors of 0.5 and 0.05.

```
# Perplexity dataframe
density = pd.DataFrame([topics,pp1,pp2,pp3]).T
density.columns = (["Topics","0.1","0.5","0.05"])
density
```

	Topics	0.1	0.5	0.05
0	5.0	1851.149915	1837.299622	1954.530055
1	10.0	1648.797651	1862.663908	1660.384625
2	14.0	1639.008248	2091.129886	1696.971867
3	15.0	1674.384310	2054.614846	1742.138758
4	20.0	1632.237883	2325.618923	1631.625658
5	30.0	1703.971649	2956.721521	1695.061717

```
# Plot the three density values on a graph for each number of topics
plt.figure(figsize=(12,4))
plt.subplot(1, 2, 1)
plt.plot(topics, pp1, label='0.1')
plt.plot(topics, pp2, label='0.5')
plt.plot(topics, pp3, label='0.05')
plt.title("Perplexity over Topics")
plt.xlabel("Num Topics")
plt.ylabel("Perplexity Scores")
plt.legend(title='Density', loc='best')

plt.subplot(1, 2, 2)
plt.plot(topics, ll1, label='0.1')
plt.plot(topics, ll2, label='0.5')
plt.plot(topics, ll3, label='0.05')
plt.title("Log Likelihood over Topics")
plt.xlabel("Num Topics")
plt.ylabel("Log Scores")
plt.legend(title='Density', loc='best')
plt.show()
```

Analysis:

- 14 topics with density = 0.1 returns the lowest perplexity.
- The 0.1 and 0.05 lines are very close but higher density of 0.5 rises sharply as topic numbers increase.
- Maximum likelihood occurs at 14 topics with 0.1 density but above that the line falls below that for 0.05 density.
- 14 topics at 0.1 density is assumed for the best model.

Changing The Number of Iterations

```
# Increased iterations for 14 topics
def num_iterations(iterations):

  lda5_model = LatentDirichletAllocation(n_components = 14,doc_topic_prior = 0.1, topic_word_prior = 0.1,max_iter = iterations,
                                         learning_method = "online",learning_decay = 0.9)

  lda5 = lda5_model.fit_transform(vec)
  perplex5 =lda5_model.perplexity(vec)
  like5 = lda5_model.score(vec)

  return perplex5,like5
```

```
# Increase iterations for 14 topics
iterations = [15,30,50,100]
pi = []
li = []
for i in iterations:
  pi.append(num_iterations(i)[0])
  li.append(num_iterations(i)[1])
```

```
# Perplexity dataframe
iterate = pd.DataFrame([iterations,pi,li]).T
iterate.columns = (["Iterations","Perplexity","Likelihood"])
iterate
```

	Iterations	Perplexity	Likelihood
0	15.0	1639.063709	-1.262850e+06
1	30.0	1534.626740	-1.258720e+06
2	50.0	1653.232270	-1.250561e+06
3	100.0	1656.677909	-1.250790e+06

Analysis

- Doubling the number of iterations appears to improve the model.

Pre-processing Text

Pre-processing can often help increase the performance of text models, so a range of processing tasks were performed.

```
# import libraries
import re
from nltk.tokenize import RegexpTokenizer
nltk.download('stopwords')
nltk.download('wordnet')
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.stem.porter import PorterStemmer
```

```
# Make a copy of the data to perform pre-processing
d1 = data_s.copy()
```

```
# Create pre-processing function
def process(text):

  # tokenize and remove punctuation
  text = text.text
  tokenizer = RegexpTokenizer(r'\w+')
  text = text.apply(lambda x: tokenizer.tokenize(x.strip().lower()))

  # Remove numbers, except words that contain numbers.
  text = text.apply(lambda x: [n for n in x if not n.isnumeric()])

  # Remove non-ascii characters
  text = text.apply(lambda x: [e for e in x if e.encode("ascii","ignore")])

  # Remove common/stopwords
  stop = stopwords.words('english')
  add_stop = ['wa','mile','ft','km',"ii","st","mi","born","mp"]
  stop.extend(add_stop)
  text = text.apply(lambda x: [w for w in x if w not in stop])
```

```
# Lemmatize text
lemmatizer = WordNetLemmatizer()
def lemmatize_text(text):
  return [lemmatizer.lemmatize(w) for w in text]
text = text.apply(lemmatize_text)

# Stem text - run only to check if improves results
#stemmer to PorterStemmer()
#def stem_text(text):
  #return [stemmer.stem(w) for w in text]
#text = text.apply_stem(text)

# Convert list to string
text = text.apply(lambda x: ' '.join(x))

return text
```

```
process(d1)
```

```
245601    waterway experiment station also known wes ori...
252469    ibm building story skyscraper metropolitan tra...
227560    royal enfield fury british motorcycle made roy...
144727    andrés gómez santos february guayaquil ecuador...
196076    kumaree rajeshree deerpalsing january also kno...
                              ...
13769     bw offshore previously known bergesen worldwid...
52931     beechworth secondary college state secondary c...
278431    phase also known phase lesbian bar nightclub 8...
61737     horndean technology college large school situa...
476345    woman touch seventeenth studio album singer so...
Name: text, Length: 8000, dtype: object
```

Note Stemming was also tried but did not provide significantly better results.

Summarising the Results

The perplexity scores showing the effect of the changes made to the base model are summarised in Table 1.

Table 1. Summary of Perplexity Scores for Various Models

Model	Perplexity	Topics	Tuning and model adjustment
Base Model	1831	14	15 iterations, learning method 'online', all other params = default
Random Grid	1858	5	15 iterations, 'online', alpha(0.5), beta(0.1), learning decay = 0.5
Model 2	1674	14	Learning decay increased from 0.7 to 0.9
Model 3	1639	14	Document and topic densities from 1/n to 0.1
Model 4	1534	14	Iterations from 15 to 30
Model 5	1345	14	Text processing – stopwords, lemmatization etc.

Fit The Best Model so Far

Bring all the tasks together into one function:

Cont…

```
# Create pipeline function for processed text.

# set column names for topics
ind_col = [1,2,3,4,5,6,7,8,9,10,11,12,13,14]

def best(x):

  topics = 14

  # Set best model parameters
  best_model = LatentDirichletAllocation(n_components = topics,max_iter = 30,
                                         doc_topic_prior = 0.1,
                                         topic_word_prior = 0.1,
                                         learning_method = "online",batch_size = 100,
                                         learning_decay = 0.9,random_state = 1)
  # pre-process data function
  x = process(x)

  # Convert to numpy array and create vectorizer object
  d1_array = x.to_numpy()
  vectorizer = CountVectorizer(strip_accents="ascii",max_df = 0.95,min_df = 5,stop_words='english')
  d1_vector = vectorizer.fit_transform(d1_array)

  # Fit model and extract dtm, wtm
  lda_best = best_model.fit_transform(d1_vector)
  word_prob_best = best_model.components_

  # Print perplexity and log score for best model
  print("Perplexity for final model:",best_model.perplexity(d1_vector))
  print("Log score for final model:",best_model.score(d1_vector))

  # Create document topic matrix with top two topics per document
  df_dtm_best = pd.DataFrame(data= lda_best, columns=ind_col)
  df_dtm_best["Top_topic"] = df_dtm_best.idxmax(axis = 1)
  g = df_dtm_best.apply(lambda s, n: pd.Series(s.nlargest(n).index), axis=1, n=3)
  g.drop([0,1],axis = 1,inplace = True)
  g.columns = ["2nd"]
  df_dtm_best["2nd"] = g["2nd"]

  # Create word topic matrix of top ten words for each topic
  feature_names = vectorizer.get_feature_names()
  words = np.array(feature_names)
  top_topics = []
  for topic_weights in word_prob_best:
      word_locs = (-topic_weights).argsort()[:10]
      top_topics.append(words.take(word_locs))
  df_top_words = pd.DataFrame(top_topics,index = ind_col)

  return vectorizer,df_dtm_best, df_top_words, feature_names, best_model
```

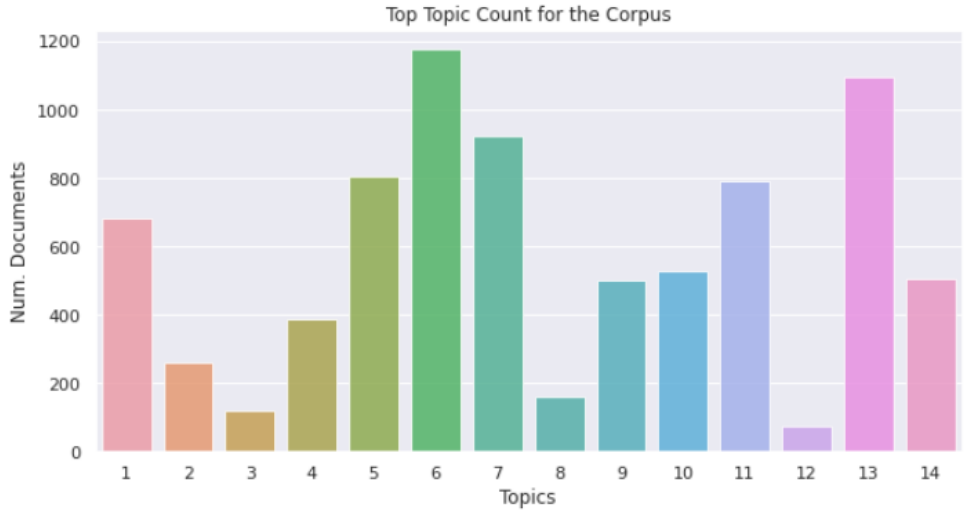
```
result = best(d1)
```

```
Perplexity for final model: 1345.9075495508048
Log score for final model: -1124435.2395091401
```

```
# Get document topic probabilities
df_dtm_best = result[1]
df_dtm_best.head()
```

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	Top_topic	2nd
0	0.268657	0.002825	0.002825	0.002825	0.002825	0.002825	0.002825	0.002825	0.237736	0.101418	0.079247	0.287516	0.002825	0.002825	12	1
1	0.004673	0.004673	0.004674	0.004673	0.004674	0.004673	0.004673	0.566443	0.377478	0.004673	0.004674	0.004673	0.004673	0.004673	8	9
2	0.006098	0.570592	0.006098	0.006098	0.006098	0.006098	0.006098	0.006100	0.006098	0.006100	0.309534	0.052792	0.006100	0.006098	2	11
3	0.006494	0.006495	0.006495	0.006495	0.073678	0.329221	0.282220	0.006496	0.006494	0.006494	0.006495	0.006494	0.249938	0.006494	6	7
4	0.008772	0.008774	0.008772	0.008773	0.008772	0.885956	0.008773	0.008773	0.008773	0.008772	0.008772	0.008772	0.008773	0.008773	6	2

```
df_topic_counts = df_dtm_best['Top_topic'].value_counts().reset_index()
df_topic_counts.columns = ['Topic', 'Documents']

# Plot top topic for all documents
plt.figure(figsize=(10,5))
sns.barplot(df_topic_counts["Topic"],df_topic_counts["Documents"],alpha=0.8)
plt.title("Top Topic Count for the Corpus")
plt.ylabel("Num. Documents")
ax = plt.xlabel("Topics")
```




```
# Get word topic matrix - top six words per topic
df_top_words = result[2]
df_top_words
```

	0	1	2	3	4	5	6	7	8	9
1	school	high	university	college	located	public	student	state	education	secondary
2	genus	family	specie	canadian	known	canada	america	ontario	member	toronto
3	australian	international	australia	summer	citation	world	olympics	needed	competed	airline
4	book	published	novel	series	story	magazine	written	fiction	author	writer
5	album	released	record	band	music	studio	singer	song	rock	label
6	member	played	state	politician	american	october	december	served	football	june
7	film	directed	best	american	starring	star	award	journal	life	based
8	new	york	city	game	known	california	woman	san	people	video
9	historic	church	house	building	located	built	national	place	register	county
10	company	based	founded	group	art	service	museum	largest	united	bank
11	river	navy	class	built	war	ship	state	world	united	tributary
12	new	red	thomas	spanish	brown	zealand	orchid	castle	ireland	irish
13	specie	family	plant	lake	mountain	endemic	south	known	native	north
14	district	village	county	population	province	east	central	north	west	south

```
# Add inferred guesses based on the word matrix

Inferred = ["Education","Animal","Athlete","Written","Album","Office Holder",
"Film","Artist","Building","Company","Transport","Natural","Plant","Village"]

df_top_words["Inferred"] = Inferred
df_top_words
```

	0	1	2	3	4	5	6	7	8	9	Inferred
1	school	high	university	college	located	public	student	state	education	secondary	Education
2	genus	family	specie	canadian	known	canada	america	ontario	member	toronto	Animal
3	australian	international	australia	summer	citation	world	olympics	needed	competed	airline	Athlete
4	book	published	novel	series	story	magazine	written	fiction	author	writer	Written
5	album	released	record	band	music	studio	singer	song	rock	label	Album
6	member	played	state	politician	american	october	december	served	football	june	Office Holder
7	film	directed	best	american	starring	star	award	journal	life	based	Film
8	new	york	city	game	known	california	woman	san	people	video	Artist
9	historic	church	house	building	located	built	national	place	register	county	Building
10	company	based	founded	group	art	service	museum	largest	united	bank	Company
11	river	navy	class	built	war	ship	state	world	united	tributary	Transport
12	new	red	thomas	spanish	brown	zealand	orchid	castle	ireland	irish	Natural
13	specie	family	plant	lake	mountain	endemic	south	known	native	north	Plant
14	district	village	county	population	province	east	central	north	west	south	Village

Analysis

- Evaluating unsupervised learning models is not easy since there is no base against which we can test the output of our model. For this dataset however, there is the original labelling.
- Topics are inferred based on the words in the word-topic matrix but the topics are blury and overlapping highlighting that this is a difficult task.

Prediction

To see how the model performs with predicting on the unseen test data.

A Single Data Point

```
# Take sample item from test set and note topic
example = test_data.sample(1)
example
```

	label	title	text
582	10	Mitrella hernandezi	Mitrella hernandezi is a species of sea snail...

```
def predict(x):

  # Process text and create dtm
  prep = process(example)
  test_vector = result[0].transform(prep)
  dtm_test = result[4].transform(test_vector)
  df_dtm_test = pd.DataFrame(dtm_test,columns = list(range(1,15)))

  # Sort to find top topic for each document
  df_dtm_test["Top"] = df_dtm_test.idxmax(axis = 1)
  h = df_dtm_test.apply(lambda s, n: pd.Series(s.nlargest(n).index), axis=1, n=3)
  h.drop([0,1],axis = 1,inplace = True)
  h.columns = ["2nd"]
  df_dtm_test["2nd"] = h

  top_topic = df_dtm_test["Top"]
  second_topic = df_dtm_test["2nd"]
  topic_guess = df_top_words.iloc[top_topic]["Inferred"]
  second_guess = df_top_words.iloc[second_topic]["Inferred"]

  return topic_guess,second_guess
```

```
# Predict sample
pd.Series(predict(example))
```

```
0      2    Animal
Name: Inferred, dtype: object
1    14    Village
Name: Inferred, dtype: object
dtype: object
```

The model seems to be working with the top topic being animal.

Using the Whole Test Set

```
# Create df to show original data with actual topic subjects as per the dataset
dict = {1:"Company",2:"Education",3:"Artist",4:"Athlete",5:"Office Holder",
       6:"Transport",7:"Building",8:"Natural",9:"Village",10:"Animal",
       11:"Plant",12:"Album",13:"Film",14:"Written"}
top = pd.DataFrame(dict.items(),columns = ["label","topic"])

# Join to test data set
test_df = test_data.merge(top, on='label', how='left')
test_df.head(10)
```

	label	title	text	topic
0	10	Platymetopus	Platymetopus is a genus of beetles in the fam...	Animal
1	10	Sicera	Sicera is a genus of moth in the family Gelec...	Animal
2	9	Strzeczonka	Strzeczonka [stʂɛ t͡ʂɔnka] is a village in th...	Village
3	7	Rose Hill Manor	Rose Hill Manor is a historic home located at...	Building
4	9	Eslamabad-e Bezahrud	Eslamabad-e Bezahrud (Persian: اسلام اباد بزه...	Village
5	3	Jonathan Singleton	Jonathan Singleton is an American country mus...	Artist
6	5	Hans-Josef Fell	Hans-Josef Fell (born 7 January 1952 in Hamme...	Office Holder
7	5	Darlene Mealy	Darlene Mealy is a member of the New York Cit...	Office Holder
8	1	Air Orient	Air Orient was an airline based in France. Cr...	Company
9	12	Out There (Eric Dolphy album)	Out There is a 1960 jazz album by Eric Dolphy...	Album

```
# Function to infer topic
def infer(x):

  # Preprocessing function on the test set and convert to dtm
  prep = process(test_data)
  test_vector = result[0].transform(prep)
  whole_test_df = pd.DataFrame(result[4].transform(test_vector),columns = ind_col)

  # Create dataframe of top words and themes
  themes = df_top_words["Inferred"]
  themes = themes.reset_index()
  themes.columns = ["Top","Inferred"]

  # Create dataframe of 'inferred topics'
  inferred = pd.DataFrame(whole_test_df.idxmax(axis = 1),columns = ["Top"])
  inferred_2 = whole_test_df.apply(lambda s,n: pd.Series(s.nlargest(n).index), axis=1, n=2)
  inferred_2.drop([0],axis = 1,inplace = True)
  inferred_2.columns = ["Top"]
  inferred_2.index +=1

  # Join the two
  Topic_names = inferred.merge(themes, on = 'Top', how = 'left')
  test_df["Guess"] = Topic_names["Inferred"]
  Second_topic = inferred_2.merge(themes, on = "Top", how = 'left')
  test_df["2nd_Guess"] = Second_topic["Inferred"]

  return test_df
```

```
test_final = infer(test_data)
test_final.head(15)
```

	label	title	text	topic	Guess	2nd_Guess
0	10	Platymetopus	Platymetopus is a genus of beetles in the fam...	Animal	Animal	Plant
1	10	Sicera	Sicera is a genus of moth in the family Gelec...	Animal	Animal	Building
2	9	Strzeczonka	Strzeczonka [stʂɛ t͡ʂonka] is a village in th...	Village	Village	Written
3	7	Rose Hill Manor	Rose Hill Manor is a historic home located at...	Building	Building	Office Holder
4	9	Eslamabad-e Bezahrud	Eslamabad-e Bezahrud (Persian: اسلام اباد بزه...	Village	Village	Animal
5	3	Jonathan Singleton	Jonathan Singleton is an American country mus...	Artist	Album	Film
6	5	Hans-Josef Fell	Hans-Josef Fell (born 7 January 1952 in Hamme...	Office Holder	Office Holder	Film
7	5	Darlene Mealy	Darlene Mealy is a member of the New York Cit...	Office Holder	Office Holder	Artist
8	1	Air Orient	Air Orient was an airline based in France. Cr...	Company	Athlete	Company
9	12	Out There (Eric Dolphy album)	Out There is a 1960 jazz album by Eric Dolphy...	Album	Album	Office Holder
10	6	HMS Repulse (1892)	HMS Repulse was a Royal Sovereign-class pre-d...	Transport	Transport	Building
11	7	Occum Hydroelectric Plant and Dam	Occum Hydroelectric Plant and Dam is a histor...	Building	Building	Plant
12	1	GDF Suez Energy Resources NA	GDF Suez Energy Resources NA is the retail en...	Company	Company	Education
13	4	Ruthie Matthes	Ruthie Matthes (born November 11 1965) is an ...	Athlete	Office Holder	Natural
14	1	Toast Records	Toast Records is an independent record label ...	Company	Album	Natural

The model appears to be performing well on the first part of the test data but what is the accuracy across all the 2000 items in the test set?

Based on the top topic for each document:

```
[ ] incorrect = test_final[test_final["topic"] != test_final["Guess"]].shape[0]
    correct = test_final.shape[0]-incorrect
    percent_correct = correct/(incorrect+correct)*100
    print("Percentage guessed correctly by model:",percent_correct)
```

```
Percentage guessed correctly by model: 61.7
```

Adding in the second guess as well as the first

```
# How many are correct guesses based on top topic and second topic guess?
incorrect = test_final[(test_final['topic'] != test_final["Guess"]) & (test_final['topic'] != test_final["2nd_Guess"])].shape[0]
correct = test_final.shape[0]-incorrect
percent_correct = correct/(incorrect+correct)*100
print("Percentage guessed correctly by model:",percent_correct)
```

```
Percentage guessed correctly by model: 70.8
```

Analysis

- The model is correct with 62% of the test_data, not much better than flipping a coin but not too bad considering the complexity of text data and a dataset with several very closely related and overlapping topics.
- When the second top topic guess is added in, so that the topic can be either of the top one or two topics, then the model returns 71% accuracy.

Gensim Library

Gensim is considered to be a more sophisticated library for NLP. An example of this library and results from using the dataset is contained in Colab in Section 12 but not included in this report due to word constraints.

Conclusions

1. LDA models can produce good results but it can be extremely difficult to design them well.
2. The key parameter is the number of topics, which is found by the process of iteration and by reference to perplexity scores, although inferring topics still relies heavily on the skill of the modeller.
3. Using the LDA function in Scikit Learn, the lowest perplexity was recorded for 6 topics, rather than the 14 in the dataset, highlighting significant fuzziness in the topics which the model struggled to define.
4. Hyperparameter tuning such as changing the topic and document priors, adjusting learning decay, increasing iterations and performing text processing all helped to improve the results but needs to be balanced with the significant time and computing resources required.
5. Based on the top-word for each document, the best model returned a 62% accuracy which increased to 71% when the second highest weighted word was included.
6. Further improvements could include using more training data, more detailed text processing methods, for example parts-of-speech tagging, a higher number of iterations and further parameter tuning, which all must be balanced with the greater compute resources required.
7. The Gensim package appeared to return good results on test data, although the overall accuracy could not be directly compared with the Scikit Learn model, since topics could not easily be inferred by reference to the top words.
8. Newer hybrid models are available (LDA2Vec) which account for context by combining word-embeddings with traditional LDA bag-of-words and these can return greatly improved performance for topic modelling, especially when trained using Recurrent Neural Networks.

References

[1] G. Salton, A. Wong, and C. S. Yang (1975), "A Vector Space Model for Automatic Indexing" Communications of the ACM, vol. 18, nr. 11, pages 613–620.

[2] Deerwester, S., Dumais, S., Landauer, T., Furnas, G., Harshman, R. Indexing by latent semantic analysis. J. Am. Soc. Inform. Sci. 41, 6 (1990), 391–407.

[3] Sangno Lee, Jaeki Song & Yongjin Kim (2010) An Empirical Comparison of Four Text Mining Methods, Journal of Computer Information Systems, 51:1, 1-10, DOI: 10.1080/08874417.2010.11645444

[4] Hofmann, T. Probabilistic Latent Semantic Analysis, In *Uncertainty in Artificial Intelligence* (UAI) (1999).

[5] Xu, Joyce. Medium. 2020. Topic Modelling With LSA, PSLA, LDA & Lda2vec. [online] Available at: https://medium.com/nanonets/topic-modeling-with-lsa-psla-lda-and-lda2vec-555ff65b0b05.

[6] Blei, David M.; Ng, Andrew Y.; Jordan, Michael I(January 2003). Lafferty, John (ed.). "Latent Dirichlet Allocation". Journal of Machine Learning Research. 3 (4–5): pp. 993–1022.

[7] Doig, Christine. PyVideo.org. 2020. Topic Modelling With Python. [online] Available at: https://pyvideo.org/pytexas-2015/topic-modeling-with-python.html.

[8] En.wikipedia.org. 2020. Latent Dirichlet Allocation. [online] Available at: https://en.wikipedia.org/wiki/Latent_Dirichlet_allocation.

[9] Sullivan, Scott. YouTube. 2020. LDA Algorithm Description. [online] Available at: https://www.youtube.com/watch?v=DWJYZq_fQ2A.

[10] David M. Blei. 2012. Probabilistic topic models. Commun. ACM 55, 4 (April 2012), 77–84. DOI:https://doi.org/10.1145/2133806.2133826.

[11] Statistics How To. 2020. EM Algorithm (Expectation-Maximization): Simple Definition - Statistics How To. [online] Available at: https://www.statisticshowto.datasciencecentral.com/em-algorithm-expectation-maximization.

[12] Griffiths TL, Steyvers M (2004). “Finding Scientific Topics.” Proceedings of the National Academy of Sciences of the United States of America, 101, 5228–5235.

[13] Chuang, Jason & Manning, Christopher & Heer, Jeffrey. (2012). Termite: Visualization Techniques for Assessing Textual Topic Models. 10.1145/2254556.2254572.

[14] Michael Röder, Andreas Both, and Alexander Hinneburg. 2015. Exploring the Space of Topic Coherence Measures. In Proceedings of the Eighth ACM International Conference on Web Search and Data Mining (WSDM ’15). Association for Computing Machinery, New York, NY, USA, 399–408. DOI:https://doi.org/10.1145/2684822.2685324

[15] David Mimno, Hanna M. Wallach, Edmund Talley, Miriam Leenders, and Andrew McCallum. 2011. Optimizing semantic coherence in topic models. In Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP ’11). Association for Computational Linguistics, USA, 262–272.

[16] D. Newman, J. H. Lau, K. Grieser, and T. Baldwin. Automatic evaluation of topic coherence. In Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics, pages 100–108. 2010.

[17] Mattilyra.github.io. 2020. Evaluating Topic Models. [online] Available at: https://mattilyra.github.io/2017/07/30/evaluating-topic-models.html

[18] Hoffman Matthew D, Blei David M and Bach Francis. 2010. Online learning for Latent Dirichlet Allocation. In Proceedings of the 23rd International Conference on Neural Information Processing Systems – Volume 1(NIPS'10). Curran Associates Inc., Red Hook, NY, USA, 856-864.

[19] Zhang, X. and LeCun, Y., 2015. Text understanding from scratch. arXiv preprint arXiv:1502.01710.

[20] Teh, Y., Jordan, M., Beal, M. and Blei, D., 2006. Hierarchical Dirichlet Processes. *Journal of the American Statistical Association*, 101(476), pp.1566-1581.

[21] Wiki.dbpedia.org. 2020. Dbpedia. [online] Available at: https://wiki.dbpedia.org/

[22] Mikolov, Tomas, Kai Chen, Gregory S. Corrado and Jeffrey Dean. "Efficient Estimation of Word Representations in Vector Space." *CoRR* abs/1301.3781 (2013): n. pag.

[23] Moody, Christopher E.. "Mixing Dirichlet Topic Models and Word Embeddings to Make lda2vec." *ArXiv* abs/1605.02019 (2016): n. pag.

Bibliography

Arvindpdmn, p., 2020. Topic Modelling. [online] Devopedia. Available at: https://devopedia.org/topic-modelling.

Blei, David M. 2012. Probabilistic topic models. Commun. ACM 55, 4 (April 2012), 77–84. DOI:https://doi.org/10.1145/2133806.2133826.

Bengfort, B., Bilbro, R. and Ojeda, T., 2018. Applied Text Analysis With Python. Sebastopol, CA: O'Reilly Media.

Blei, David M. 2012. Probabilistic topic models. Commun. ACM 55, 4 (April 2012), 77-84. DPO: https//doi.org/10.1145/2133806.2133826.

Blei, David M.; Ng, Andrew Y.; Jordan, Michael I(January 2003). Lafferty, John (ed.). "Latent Dirichlet Allocation". Journal of Machine Learning Research. 3 (4–5): pp. 993–1022.

Brooks, Andrew. 2020. Latent Dirichlet Allocation - Under The Hood. [online] Available at: http://brooksandrew.github.io/simpleblog/articles/latent-dirichlet-allocation-under-the-hood/.

Chuang, Jason & Manning, Christopher & Heer, Jeffrey. (2012). Termite: Visualization Techniques for Assessing Textual Topic Models. 10.1145/2254556.2254572.

Chuang, J., Gupta, S., Manning, C. & Heer, J.. (2013). Topic Model Diagnostics: Assessing Domain Relevance via Topical Alignment. Proceedings of the 30th International Conference on Machine Learning, in PMLR 28(3):612-620

Coursera. 2020. Vector Space Model - Big Data Modelling (Part 2) | Coursera. [online] Available at: https://www.coursera.org/lecture/big-data-management/vector-space-model-0GMs1

Datasciencecentral.com. 2020. Information Retrieval Document Search Using Vector Space Model In R. [online] Available at: https://www.datasciencecentral.com/profiles/blogs/information-retrieval-document-search-using-vector-space-model-in.

Datasciencecentral.com. 2020. Information Retrieval Document Search Using Vector Space Model In R. [online] Available at: https://www.datasciencecentral.com/profiles/blogs/information-retrieval-document-search-using-vector-space-model-in.

Deerwester, S., Dumais, S., Landauer, T., Furnas, G., Harshman, R. Indexing by latent semantic analysis. J. Am. Soc. Inform. Sci. 41, 6 (1990), 391–407.

Devopedia. 2020. Latent Dirichlet Allocation. [online] Available at: https://devopedia.org/latent-dirichlet-allocation.

Dl.acm.org. 2020. Evaluation Methods For Topic Models | Proceedings Of The 26Th Annual International Conference On Machine Learning. [online]. Available at: https://dl.acm.org/doi/10.1145/1553374.1553515.

Doig, Christine. PyVideo.org. 2020. Topic Modelling With Python. [online] Available at: https://pyvideo.org/pytexas-2015/topic-modeling-with-python.html.

En.wikipedia.org. 2020. Latent Dirichlet Allocation. [online] Available at: https://en.wikipedia.org/wiki/Latent_Dirichlet_allocation.

Es, Shahul, neptune.ai. 2020. *Exploratory Data Analysis For Natural Language Processing: A Complete Guide To Python Tools | Neptune's Blog*. [online] Available at: https://neptune.ai/blog/exploratory-data-analysis-natural-language-processing-tools.

Griffiths TL, Steyvers M (2004). "Finding Scientific Topics." Proceedings of the National Academy of Sciences of the United States of America, 101, 5228–5235.

Hoffman, Matthew D., Blei, David M. and Bach, Francis. 2010. Online learning for Latent Dirichlet Allocation. In Proceedings of the 23rd International Conference on Neural Information Processing Systems - Volume 1 (NIPS'10). Curran Associates Inc., Red Hook, NY, USA, 856–864.

Hofmann, T. Probabilistic Latent Semantic Analysis, In *Uncertainty in Artificial Intelligence* (UAI) (1999).

Hoffman Matthew D, Blei David M and Bach Francis (2010). Online learning for Latent Dirichlet Allocation. In Proceedings of the 23rd International Conference on Neural Information Processing Systems – Volume 1(NIPS'10). Curran Associates Inc., Red Hook, NY, USA, 856-864.

Kapadia Shashank. 2020. Evaluate Topic Models: Latent Dirichlet Allocation (LDA). [online] Available at: https://towardsdatascience.com/evaluate-topic-model-in-python-latent-dirichlet-allocation-lda-7d57484bb5d0.

Kumar, K., 2020. Evaluation Of Topic Modelling: Topic Coherence. [online] Data Science+. Available at: https://datascienceplus.com/evaluation-of-topic-modeling-topic-coherence/.
Lee Sangno, Song Jaeki & Kim Yongjin (2010) An Empirical Comparison of Four Text Mining Methods, Journal of Computer Information Systems, 51:1, 1-10, DOI: 10.1080/08874417.2010.11645444

Mattilyra.github.io. 2020. Evaluating Topic Models. [online] Available at: https://mattilyra.github.io/2017/07/30/evaluating-topic-models.html

Mikolov, Tomas, Kai Chen, Gregory S. Corrado and Jeffrey Dean. "Efficient Estimation of Word Representations in Vector Space." *CoRR* abs/1301.3781 (2013): n. pag.

Moody, Christopher E.. "Mixing Dirichlet Topic Models and Word Embeddings to Make lda2vec." *ArXiv* abs/1605.02019 (2016): n. pag.

Lehmann, Jens et al. "DBpedia - A large-scale, multilingual knowledge base extracted from Wikipedia." *Semantic Web* 6 (2015): 167-195.

Lettier, D. 2020. Your Guide To Latent Dirichlet Allocation. [online] Available at: https://medium.com/@lettier/how-does-lda-work-ill-explain-using-emoji-108abf40fa7d.

Mimno David, Wallach Hanna M., Talley Edmund, Leenders Miriam, and McCallum Andrew. 2011. Optimizing semantic coherence in topic models. In Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP '11). Association for Computational Linguistics, USA, 262–272.

Newman D., Lau J. H., Grieser K., and Baldwin T. Automatic evaluation of topic coherence. In Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics, pages 100–108. 2010.

Prabhakaran, Selva. Machine Learning Plus. 2020. *LDA - How To Grid Search Best Topic Models? (With Examples In Python)*. [online] Available at: https://www.machinelearningplus.com/nlp/topic-modeling-python-sklearn-examples.

Röder Michael, Both Andreas, and Hinneburg Alexander. 2015. Exploring the Space of Topic Coherence Measures. In Proceedings of the Eighth ACM International Conference on Web Search and Data Mining (WSDM '15). Association for Computing Machinery, New York, NY, USA, 399–408. DOI:https://doi.org/10.1145/2684822.2685324

Salton, G., Wong, A. and Yang C. S. (1975), "A Vector Space Model for Automatic Indexing" Communications of the ACM, vol. 18, nr. 11, pages 613–620.

Sievert, Carson and Kenneth Shirley. "LDAvis: A method for visualizing and interpreting topics." (2014).

Soltoff, B., 2020. Topic Modelling | Computing For The Social Sciences. [online] Computing for the Social Sciences. Available at: https://cfss.uchicago.edu/notes/topic-modeling.

Stack Abuse. 2020. Python For NLP: Topic Modelling. [online] Available at: https://stackabuse.com/python-for-nlp-topic-modeling/.

Statistics How To. 2020. Dirichlet Distribution: Simple Definition, PDF, Mean - Statistics How To. [online] Available at: https://www.statisticshowto.datasciencecentral.com/dirichlet-distribution.

Statistics How To. 2020. EM Algorithm (Expectation-Maximization): Simple Definition - Statistics How To. [online] Available at: https://www.statisticshowto.datasciencecentral.com/em-algorithm-expectation-maximization.

Sullivan, Scott. YouTube. 2020. LDA Algorithm Description. [online] Available at: https://www.youtube.com/watch?v=DWJYZq_fQ2A.

Teh, Y., Jordan, M., Beal, M. and Blei, D., 2006. Hierarchical Dirichlet Processes. *Journal of the American Statistical Association*, 101(476), pp.1566-1581.

Wiki.dbpedia.org. 2020. Dbpedia. [online] Available at: https://wiki.dbpedia.org/

Xu, Joyce. Medium. 2020. Topic Modelling With LSA, PSLA, LDA & Lda2vec. [online] Available at: https://medium.com/nanonets/topic-modeling-with-lsa-psla-lda-and-lda2vec-555ff65b0b05.

Zhang, X, Zhao, J & LeCun, Y 2015, Character-level convolutional networks for text classification. in Advances in Neural Information Processing Systems. vol. 2015-January, Neural information processing systems foundation, pp. 649-657, 29th Annual Conference on Neural Information Processing Systems, NIPS 2015, Montreal, Canada, 12/7/15.

Zhang, X. and LeCun, Y., 2015. Text understanding from scratch. *arXiv preprint arXiv:1502.01710*.

APPENDIX 1 – TF-IDF Example

Document 1: I love cats and dogs

Document 2: I hate cats and dogs

Documents are represented as a matrix of words and their frequency. Splitting words into separate tokens, removing stopwords/common words with no value such as "is" and punctuation or special characters is often done as a pre-processing step. This can help to reduce the size of the vocabulary which otherwise can be very large.

Document Term Matrix

		I	love	cats	and	hate	dogs
Doc 1	I love cats and dogs	1	1	1	1	0	1
Doc 2	I hate cats and dogs	1	0	1	1	1	1
Query	love	0	1	0	0	0	0

Each row is a document and every word is represented. This forms the vocabulary.

Bag of Words model—no word order and no word relationships captured

Tf-idf = tf X idf

Each word is represented as the product of two terms:

- Tf or term frequency
- Idf or inverse document frequency

Idf = log(N/df) where df is the document frequency or number of documents that contain a word and N is the number of documents

A word is upweighted if it occurs more frequently in a document and downweights words that occur frequently in the whole corpus (e.g. "and", "the" etc).
Note: there are many ways to calculate the tf and idf, this is just one example. See Wikipedia https://en.wikipedia.org/wiki/Tf%E2%80%93idf

Captures frequency and relevancy but not the relationships with surrounding words.

Term Frequency—Inverse Document Frequency

	Document Frequency (df)	Inverse Document Frequency (idf)
I	2	0
love	1	0.30103
cats	2	0
and	2	0
hate	1	0.30103
dogs	2	0

Number of documents (N) = 2
idf = log(N/df)

Weighted Document Term Matrix

		I	love	cats	and	hate	dogs
Doc 1	I love cats and dogs	0	0.30103	0	0	0	0
Doc 2	I hate cats and dogs	0	0	0	0	0.30103	0
Query	love	0	0.30103	0	0	0	0

APPENDIX 2 – Colab Sheet Index

1.0 Import Libraries
2.0 Pre-processing
3.0 Examining the Dataset
4.0 Creating the Document Term Matrix
5.0 Fitting the Base Model
6.0 Examining the Output
7.0 Visualising the Model
8.0 Model Performance and Hyperparameter Tuning
9.0 Text Pre-processing
10.0 Fit the Best Model
11.0 Prediction
12.0 Gensim Model Comparison