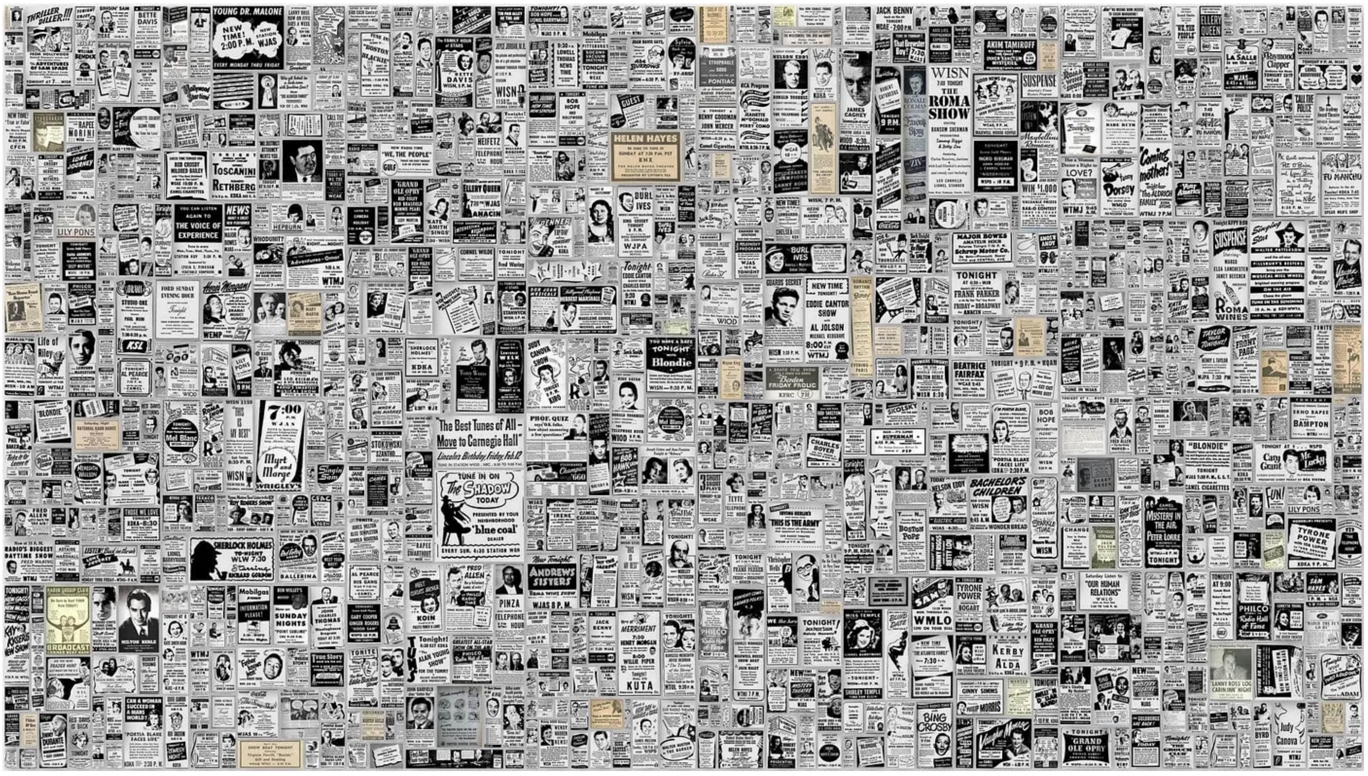


http://getwallpapers.com/image/398564

IN-DEPTH ANALYSIS

Topic Modeling in Python: Latent Dirichlet Allocation (LDA)

How to get started with topic modeling using LDA in Python

Shashank Kapadia · Follow

Published in Towards Data Science

7 min read · Apr 14, 2019

Preface: This article aims to provide consolidated information on the underlying topic and is not to be considered as the original work. The information and the code are repurposed through several online articles, research papers, books, and open-source code

Introduction

Topic Models, in a nutshell, are a type of statistical language models used for uncovering hidden structure in a collection of texts. In a practical and more intuitively, you can think of it as a task of:

Dimensionality Reduction, where rather than representing a text *T* in its feature space as {Word_i: count(Word_i, T) for Word_i in Vocabulary}, you can represent it in a topic space as {Topic_i: Weight(Topic_i, T) for Topic_i in Topics}

Unsupervised Learning, where it can be compared to clustering, as in the case of clustering, the number of topics, like the number of clusters, is an output parameter. By doing topic modeling, we build clusters of words rather than clusters of texts. A text is thus a mixture of all the topics, each having a specific weight

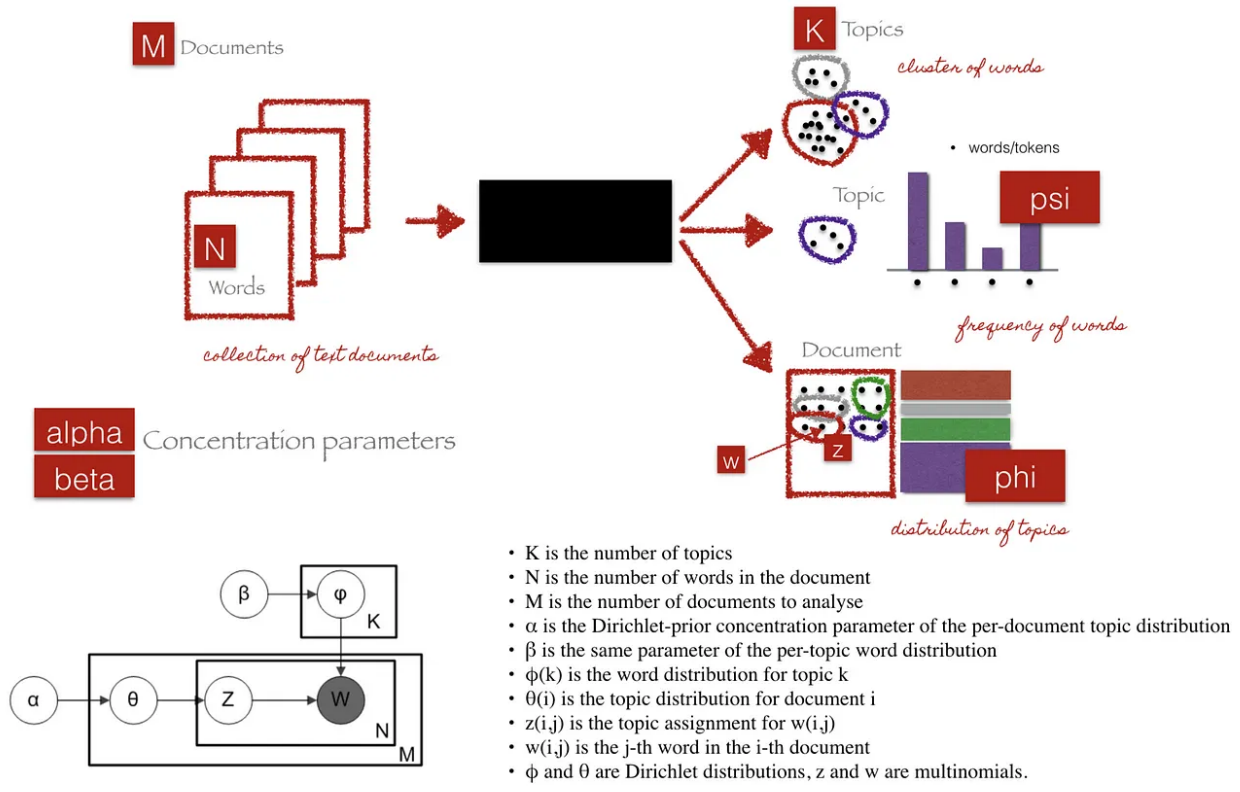
Tagging, abstract "topics" that occur in a collection of documents that best represents the information in them.

There are several existing algorithms you can use to perform the topic modeling. The most common of it are, *Latent Semantic Analysis (LSA/LSI), Probabilistic Latent Semantic Analysis (pLSA), and Latent Dirichlet Allocation (LDA)*

In this article, we'll take a closer look at LDA, and implement our first topic model using the *sklearn* implementation in *python 2.7*

Theoretical Overview

LDA is a generative probabilistic model that assumes each topic is a mixture over an underlying set of words, and each document is a mixture of over a set of topic probabilities.



- K is the number of topics
- N is the number of words in the document
- M is the number of documents to analyse
- α is the Dirichlet-prior concentration parameter of the per-document topic distribution
- β is the same parameter of the per-topic word distribution
- ϕ(k) is the word distribution for topic k
- θ(i) is the topic distribution for document i
- z(i,j) is the topic assignment for w(i,j)
- w(i,j) is the j-th word in the i-th document
- ϕ and θ are Dirichlet distributions, z and w are multinomials.

http://chdoig.github.io/pytexas2015-topic-modeling/#/3/4

We can describe the generative process of LDA as, given the *M* number of documents, *N* number of words, and prior *K* number of topics, the model trains to output:

psi, the distribution of words for each topic *K*

phi, the distribution of topics for each document *i*

Parameters of LDA

> ***Alpha parameter** is Dirichlet prior concentration parameter that represents document-topic density — with* a higher alpha, documents are assumed to be made up of more topics and result in more specific topic distribution per document.
>
> ***Beta parameter** is the same prior concentration parameter that represents topic-word density —* with high beta, topics are assumed to made of up most of the words and result in a more specific word distribution per topic.

LDA Implementation

The complete code is available as a Jupyter Notebook on GitHub

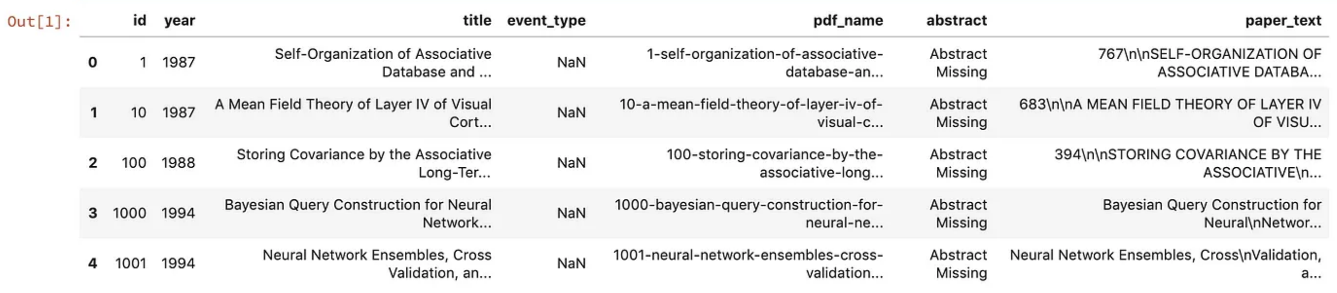
1. Loading data
2. Data cleaning
3. Exploratory analysis
4. Preparing data for LDA analysis
5. LDA model training
6. Analyzing LDA model results

Loading data

For this tutorial, we'll use the dataset of papers published in NeurIPS (NIPS) conference which is one of the most prestigious yearly events in the machine learning community. The CSV data file contains information on the different NeurIPS papers that were published from 1987 until 2016 (29 years!). These papers discuss a wide variety of topics in machine learning, from neural networks to optimization methods, and many more.

Let's start by looking at the content of the file

```
# Importing modules
import pandas as pd
import os

os.chdir('..')

# Read data into papers
papers = pd.read_csv('./data/NIPS Papers/papers.csv')

# Print head
papers.head()
```



Out[1]:

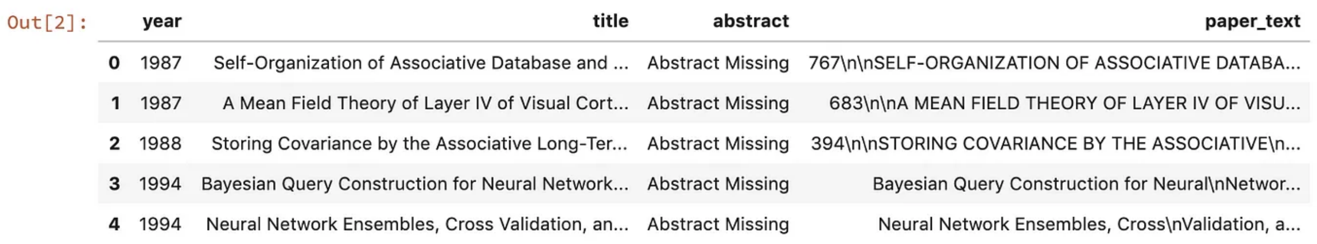
	id	year	title	event_type	pdf_name	abstract	paper_text
0	1	1987	Self-Organization of Associative Database and ...	NaN	1-self-organization-of-associative-database-an...	Abstract Missing	767\n\nSELF-ORGANIZATION OF ASSOCIATIVE DATABA...
1	10	1987	A Mean Field Theory of Layer IV of Visual Cort...	NaN	10-a-mean-field-theory-of-layer-iv-of-visual-c...	Abstract Missing	683\n\nA MEAN FIELD THEORY OF LAYER IV OF VISU...
2	100	1988	Storing Covariance by the Associative Long-Ter...	NaN	100-storing-covariance-by-the-associative-long...	Abstract Missing	394\n\nSTORING COVARIANCE BY THE ASSOCIATIVE\n...
3	1000	1994	Bayesian Query Construction for Neural Network...	NaN	1000-bayesian-query-construction-for-neural-ne...	Abstract Missing	Bayesian Query Construction for Neural\nNetwor...
4	1001	1994	Neural Network Ensembles, Cross Validation, an...	NaN	1001-neural-network-ensembles-cross-validation...	Abstract Missing	Neural Network Ensembles, Cross\nValidation, a...

Sample of raw data

Data Cleaning

Since the goal of this analysis is to perform topic modeling, let's focus only on the text data from each paper, and drop other metadata columns. Also, for the demonstration, we'll only look at 100 papers

```
# Remove the columns
papers = papers.drop(columns=['id', 'event_type', 'pdf_name'],
axis=1).sample(100)

# Print out the first rows of papers
papers.head()
```



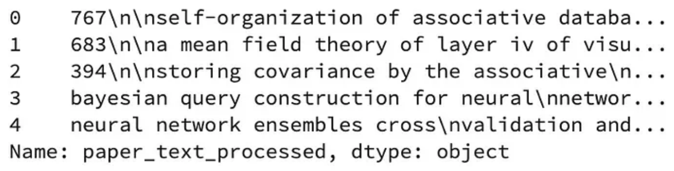
Out[2]:

	year	title	abstract	paper_text
0	1987	Self-Organization of Associative Database and ...	Abstract Missing	767\n\nSELF-ORGANIZATION OF ASSOCIATIVE DATABA...
1	1987	A Mean Field Theory of Layer IV of Visual Cort...	Abstract Missing	683\n\nA MEAN FIELD THEORY OF LAYER IV OF VISU...
2	1988	Storing Covariance by the Associative Long-Ter...	Abstract Missing	394\n\nSTORING COVARIANCE BY THE ASSOCIATIVE\n...
3	1994	Bayesian Query Construction for Neural Network...	Abstract Missing	Bayesian Query Construction for Neural\nNetwor...
4	1994	Neural Network Ensembles, Cross Validation, an...	Abstract Missing	Neural Network Ensembles, Cross\nValidation, a...

Remove punctuation/lower casing

Next, let's perform a simple preprocessing on the content of *paper_text* column to make them more amenable for analysis, and reliable results. To do that, we'll use a regular expression to remove any punctuation, and then *lowercase* the text

```
# Load the regular expression library
import re

# Remove punctuation
papers['paper_text_processed'] = \
papers['paper_text'].map(lambda x: re.sub('[,\.!?]', '', x))

# Convert the titles to lowercase
papers['paper_text_processed'] = \
papers['paper_text_processed'].map(lambda x: x.lower())

# Print out the first rows of papers
papers['paper_text_processed'].head()
```

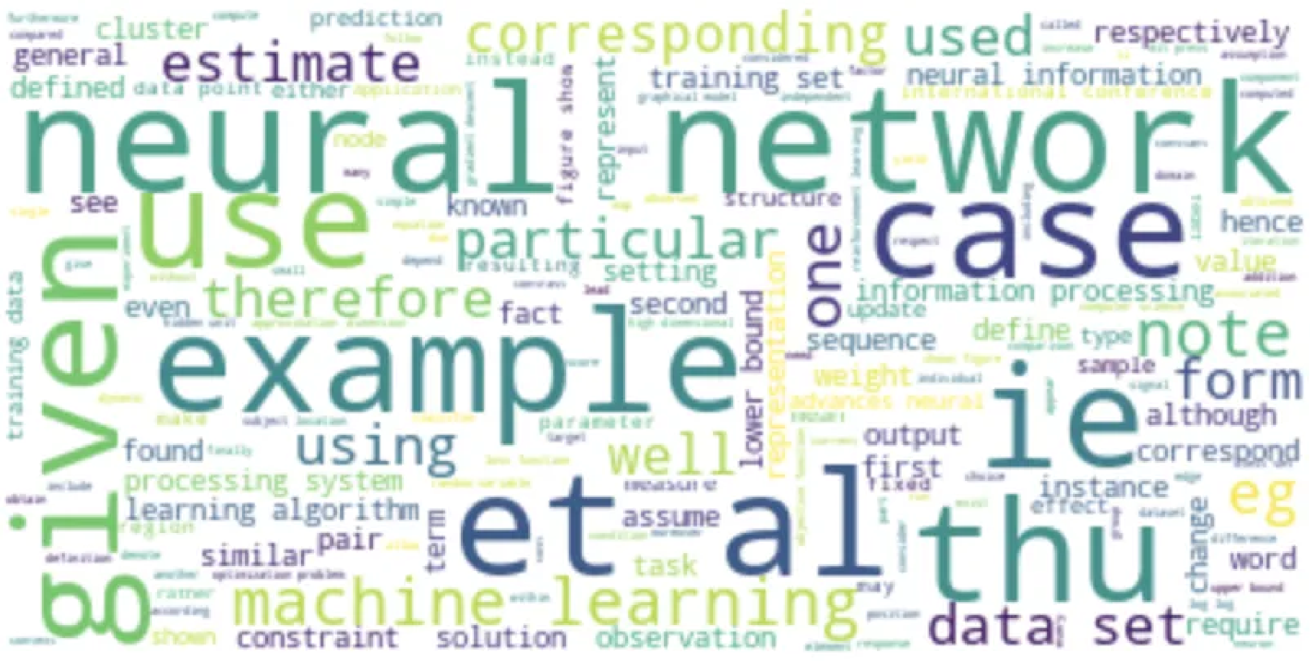


```
0    767\n\nself-organization of associative databa...
1    683\n\na mean field theory of layer iv of visu...
2    394\n\nstoring covariance by the associative\n...
3    bayesian query construction for neural\nnnetwor...
4    neural network ensembles cross\nvalidation and...
Name: paper_text_processed, dtype: object
```

Exploratory Analysis

To verify whether the preprocessing, we'll make a word cloud using the wordcloud package to get a visual representation of most common words. It is key to understanding the data and ensuring we are on the right track, and if any more preprocessing is necessary before training the model.

```
# Import the wordcloud library
from wordcloud import WordCloud

# Join the different processed titles together.
long_string = ','.join(list(papers['paper_text_processed'].values))

# Create a WordCloud object
wordcloud = WordCloud(background_color="white", max_words=5000,
contour_width=3, contour_color='steelblue')

# Generate a word cloud
wordcloud.generate(long_string)

# Visualize the word cloud
wordcloud.to_image()
```



Prepare data for LDA Analysis

Next, let's work to transform the textual data in a format that will serve as an input for training LDA model. We start by tokenizing the text and removing stopwords. Next, we convert the tokenized object into a corpus and dictionary.

```
import gensim
from gensim.utils import simple_preprocess
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords

stop_words = stopwords.words('english')
stop_words.extend(['from', 'subject', 're', 'edu', 'use'])

def sent_to_words(sentences):
    for sentence in sentences:
        # deacc=True removes punctuations
        yield(gensim.utils.simple_preprocess(str(sentence),
deacc=True))

def remove_stopwords(texts):
    return [[word for word in simple_preprocess(str(doc))
             if word not in stop_words] for doc in texts]

data = papers.paper_text_processed.values.tolist()
data_words = list(sent_to_words(data))

# remove stop words
data_words = remove_stopwords(data_words)
```

```
print(data_words[:1][0][:30])
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/shashank/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
['self', 'organization', 'associative', 'database', 'applications', 'hisashi', 'suzuki'
, 'suguru', 'arimoto', 'osaka', 'university', 'toyonaka', 'osaka', 'japan', 'abstract',
'efficient', 'method', 'self', 'organizing', 'associative', 'databases', 'proposed', 't
ogether', 'applications', 'robot', 'eyesight', 'systems', 'proposed', 'databases', 'ass
ociate']
```

```
import gensim.corpora as corpora

# Create Dictionary
id2word = corpora.Dictionary(data_words)

# Create Corpus
texts = data_words

# Term Document Frequency
corpus = [id2word.doc2bow(text) for text in texts]

# View
print(corpus[:1][0][:30])
```

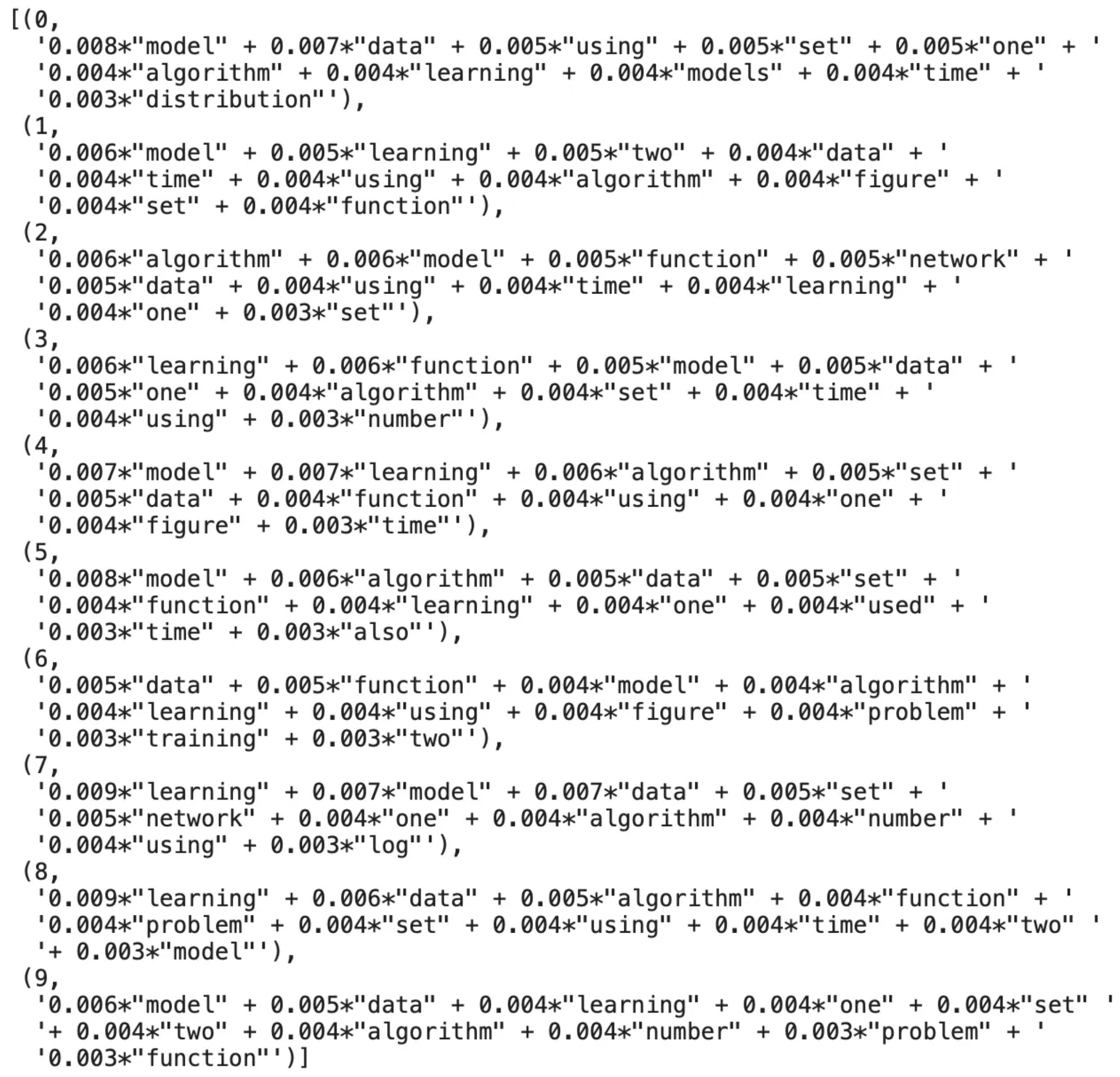
```
[(0, 1), (1, 8), (2, 1), (3, 1), (4, 1), (5, 2), (6, 1), (7, 6), (8, 1), (9, 1), (10, 3
), (11, 1), (12, 2), (13, 2), (14, 1), (15, 1), (16, 1), (17, 1), (18, 1), (19, 6), (20
, 2), (21, 4), (22, 8), (23, 5), (24, 1), (25, 1), (26, 2), (27, 2), (28, 1), (29, 1)]
```

LDA model training

To keep things simple, we'll keep all the parameters to default except for inputting the number of topics. For this tutorial, we will build a model with 10 topics where each topic is a combination of keywords, and each keyword contributes a certain weightage to the topic.

```
from pprint import pprint

# number of topics
num_topics = 10

# Build LDA model
lda_model = gensim.models.LdaMulticore(corpus=corpus,
                                       id2word=id2word,
                                       num_topics=num_topics)
```

```
# Print the Keyword in the 10 topics
pprint(lda_model.print_topics())
doc_lda = lda_model[corpus]
```



```
[(0,
  '0.008*"model" + 0.007*"data" + 0.005*"using" + 0.005*"set" + 0.005*"one" + '
  '0.004*"algorithm" + 0.004*"learning" + 0.004*"models" + 0.004*"time" + '
  '0.003*"distribution"'),
 (1,
  '0.006*"model" + 0.005*"learning" + 0.005*"two" + 0.004*"data" + '
  '0.004*"time" + 0.004*"using" + 0.004*"algorithm" + 0.004*"figure" + '
  '0.004*"set" + 0.004*"function"'),
 (2,
  '0.006*"algorithm" + 0.006*"model" + 0.005*"function" + 0.005*"network" + '
  '0.005*"data" + 0.004*"using" + 0.004*"time" + 0.004*"learning" + '
  '0.004*"one" + 0.003*"set"'),
 (3,
  '0.006*"learning" + 0.006*"function" + 0.005*"model" + 0.005*"data" + '
  '0.005*"one" + 0.004*"algorithm" + 0.004*"set" + 0.004*"time" + '
  '0.004*"using" + 0.003*"number"'),
 (4,
  '0.007*"model" + 0.007*"learning" + 0.006*"algorithm" + 0.005*"set" + '
  '0.005*"data" + 0.004*"function" + 0.004*"using" + 0.004*"one" + '
  '0.004*"figure" + 0.003*"time"'),
 (5,
  '0.008*"model" + 0.006*"algorithm" + 0.005*"data" + 0.005*"set" + '
  '0.004*"function" + 0.004*"learning" + 0.004*"one" + 0.004*"used" + '
  '0.003*"time" + 0.003*"also"'),
 (6,
  '0.005*"data" + 0.005*"function" + 0.004*"model" + 0.004*"algorithm" + '
  '0.004*"learning" + 0.004*"using" + 0.004*"figure" + 0.004*"problem" + '
  '0.003*"training" + 0.003*"two"'),
 (7,
  '0.009*"learning" + 0.007*"model" + 0.007*"data" + 0.005*"set" + '
  '0.005*"network" + 0.004*"one" + 0.004*"algorithm" + 0.004*"number" + '
  '0.004*"using" + 0.003*"log"'),
 (8,
  '0.009*"learning" + 0.006*"data" + 0.005*"algorithm" + 0.004*"function" + '
  '0.004*"problem" + 0.004*"set" + 0.004*"using" + 0.004*"time" + 0.004*"two" '
  '+ 0.003*"model"'),
 (9,
  '0.006*"model" + 0.005*"data" + 0.004*"learning" + 0.004*"one" + 0.004*"set" '
  '+ 0.004*"two" + 0.004*"algorithm" + 0.004*"number" + 0.003*"problem" + '
  '0.003*"function"')]
```

Analyzing LDA model results

Now that we have a trained model let's visualize the topics for interpretability. To do so, we'll use a popular visualization package, pyLDAvis which is designed to help interactively with:

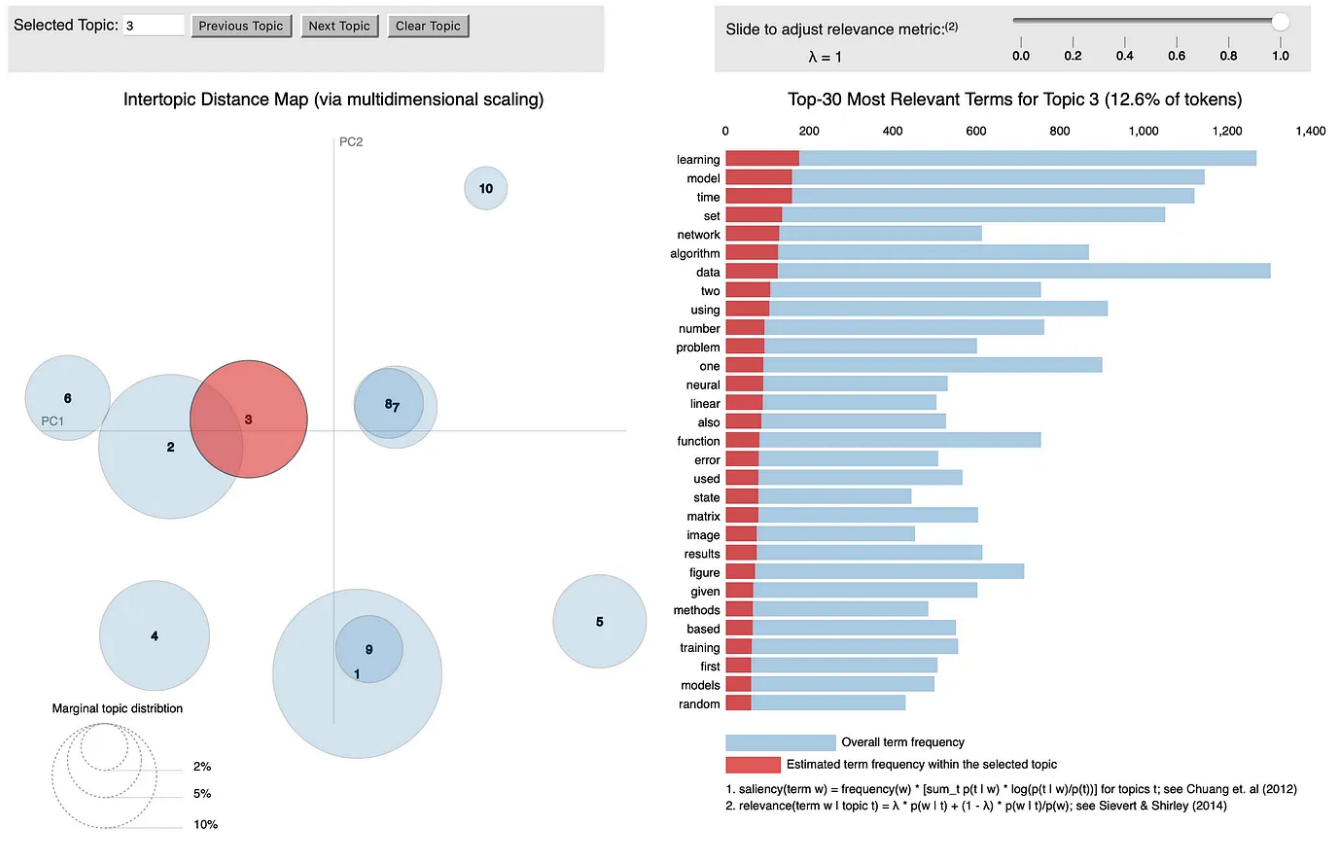
1. Better understanding and interpreting individual topics, and

2. Better understanding the relationships between the topics.

For (1), you can manually select each topic to view its top most frequent and/or “relevant” terms, using different values of the λ parameter. This can help when you’re trying to assign a human interpretable name or “meaning” to each topic.

For (2), exploring the *Intertopic Distance Plot* can help you learn about how topics relate to each other, including potential higher-level structure between groups of topics.

```
import pyLDAvis.gensim
import pickle
import pyLDAvis

# Visualize the topics
pyLDAvis.enable_notebook()

LDAvis_data_filepath =
os.path.join('./results/ldavis_prepared_'+str(num_topics))

# # this is a bit time consuming - make the if statement True
# # if you want to execute visualization prep yourself
if 1 == 1:
    LDAvis_prepared = pyLDAvis.gensim.prepare(lda_model, corpus,
id2word)
    with open(LDAvis_data_filepath, 'wb') as f:
        pickle.dump(LDAvis_prepared, f)

# load the pre-prepared pyLDAvis data from disk
with open(LDAvis_data_filepath, 'rb') as f:
    LDAvis_prepared = pickle.load(f)

pyLDAvis.save_html(LDAvis_prepared, './results/ldavis_prepared_'+
str(num_topics) +'.html')

LDAvis_prepared
```



1. saliency(term w) = frequency(w) * [sum_t p(t | w) * log(p(t | w)/p(t))] for topics t; see Chuang et. al (2012)
2. relevance(term w | topic t) = λ * p(w | t) + (1 - λ) * p(w | t)/p(w); see Sievert & Shirley (2014)

Closing Notes

Machine learning has become increasingly popular over the past decade, and recent advances in computational availability have led to exponential growth to people looking for ways how new methods can be incorporated to advance the field of Natural Language Processing.

Often, we treat topic models as black-box algorithms, but hopefully, this article addressed to shed light on the underlying math, and intuitions behind it, and high-level code to get you started with any textual data.

In the next article, we'll go one step deeper into understanding how you can evaluate the performance of topic models, tune its hyper-parameters to get more intuitive and reliable results.

References:

[1] Topic model — Wikipedia. https://en.wikipedia.org/wiki/Topic_model

[2] Distributed Strategies for Topic Modeling. https://www.ideals.illinois.edu/bitstream/handle/2142/46405/ParallelTopicModels.pdf

?sequence=2&isAllowed=y

[3] Topic Mapping — Software — Resources — Amaral Lab. https://amaral.northwestern.edu/resources/software/topic-mapping

[4] A Survey of Topic Modeling in Text Mining. https://thesai.org/Downloads/Volume6No1/Paper_21-A_Survey_of_Topic_Modeling_in_Text_Mining.pdf

Thanks for reading. *If you have any feedback, please feel to reach out by commenting on this post, messaging me on LinkedIn, or shooting me an email (shmkapadia[at]gmail.com)*

If you liked this article, visit my other articles on NLP

Evaluate Topic Models: Latent Dirichlet Allocation (LDA)
A step-by-step guide to building interpretable topic models
towardsdatascience.com

Introduction to Natural Language Processing (NLP)
A brief introduction to NLP
medium.com

Building Blocks: Text Pre-Processing
This article is the second of more to come articles on Natural Language Processing. The purpose of this series of…
towardsdatascience.com