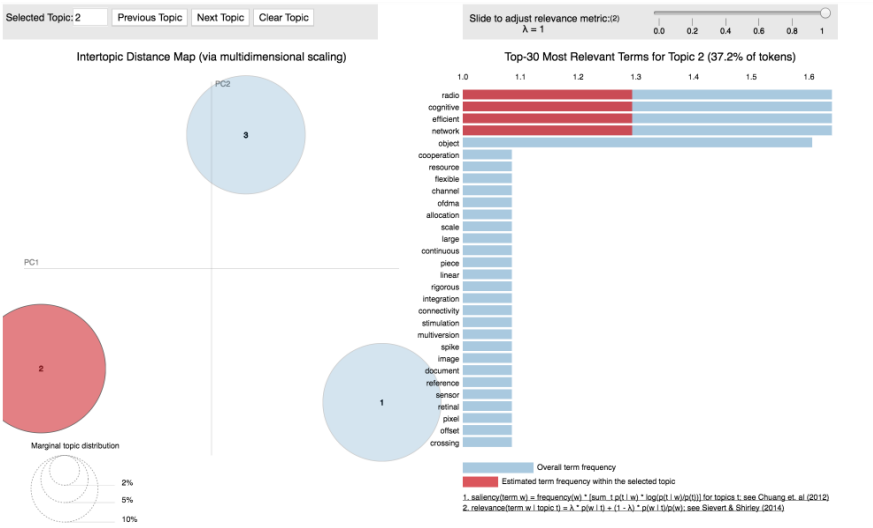
Topic Modelling in Python with NLTK and Gensim

Susan Li Follow
Mar 30, 2018 · 6 min read



In this post, we will learn how to identity which topic is discussed in a document, called topic modelling. In particular, we will cover Latent Dirichlet Allocation (LDA): a widely used topic modelling technique. And we will apply LDA to convert set of research papers to a set of topics.

Research paper topic modelling is an unsupervised machine learning method that helps us discover hidden semantic structures in a paper, that allows us to learn topic representations of papers in a corpus. The model can be applied to any kinds of labels on documents, such as tags on posts on the website.

The Process

- We pick the number of topics ahead of time even if we're not sure what the topics are.
- Each document is represented as a distribution over topics.
- Each topic is represented as a distribution over words.

The research paper text data is just a bunch of unlabeled texts and can be found here.

Text Cleaning

We use the following function to clean our texts and return a list of tokens:

```
import spacy
spacy.load('en')
from spacy.lang.en import English
parser = English()
```

```
def tokenize(text):
    lda_tokens = []
    tokens = parser(text)
    for token in tokens:
        if token.orth_.isspace():
            continue
        elif token.like_url:
            lda_tokens.append('URL')
        elif token.orth_.startswith('@'):
            lda_tokens.append('SCREEN_NAME')
        else:
            lda_tokens.append(token.lower_)
    return lda_tokens
```

We use NLTK's Wordnet to find the meanings of words, synonyms, antonyms, and more. In addition, we use WordNetLemmatizer to get the root word.

```
import nltk
nltk.download('wordnet')
```

```
from nltk.corpus import wordnet as wn
def get_lemma(word):
    lemma = wn.morphy(word)
    if lemma is None:
        return word
    else:
        return lemma

from nltk.stem.wordnet import WordNetLemmatizer
def get_lemma2(word):
    return WordNetLemmatizer().lemmatize(word)
```

Filter out stop words:

```
nltk.download('stopwords')
en_stop = set(nltk.corpus.stopwords.words('english'))
```

Now we can define a function to prepare the text for topic modelling:

```
def prepare_text_for_lda(text):
    tokens = tokenize(text)
    tokens = [token for token in tokens if len(token) > 4]
    tokens = [token for token in tokens if token not in
en_stop]
    tokens = [get_lemma(token) for token in tokens]
    return tokens
```

Open up our data, read line by line, for each line, prepare text for LDA, then add to a list.

Now we can see how our text data are converted:

```
import random
text_data = []
with open('dataset.csv') as f:
    for line in f:
        tokens = prepare_text_for_lda(line)
        if random.random() > .99:
            print(tokens)
            text_data.append(tokens)
```

['sociocrowd', 'social', 'network', 'base', 'framework', 'crowd', 'simulation']
['detection', 'technique', 'clock', 'recovery', 'application']
['voltage', 'syllabic', 'companding', 'domain', 'filter']
['perceptual', 'base', 'coding', 'decision']
['cognitive', 'mobile', 'virtual', 'network', 'operator', 'investment', 'pricing', 'supply', 'uncertainty']
['clustering', 'query', 'search', 'engine']
['psychological', 'engagement', 'enterprise', 'starting', 'london']
['10-bit', '200-ms', 'digitally', 'calibrate', 'pipelined', 'using', 'switching', 'opamps']
['optimal', 'allocation', 'resource', 'distribute', 'information', 'network']
['modeling', 'synaptic', 'plasticity', 'within', 'network', 'highly', 'accelerate', 'i&f', 'neuron']
['tile', 'interleave', 'multi', 'level', 'discrete', 'wavelet', 'transform']
['security', 'cross', 'layer', 'protocol', 'wireless', 'sensor', 'network']

['objectivity', 'industrial', 'exhibit']
['balance', 'packet', 'discard', 'improve', 'performance', 'network']
['bodyqos', 'adaptive', 'radio', 'agnostic', 'sensor', 'network']
['design', 'reliability', 'methodology']
['context', 'aware', 'image', 'semantic', 'extraction', 'social']
['computation', 'unstable', 'limit', 'cycle', 'large', 'scale', 'power', 'system', 'model']
['photon', 'density', 'estimation', 'using', 'multiple', 'importance', 'sampling']
['approach', 'joint', 'blind', 'space', 'equalization', 'estimation']
['unify', 'quadratic', 'programming', 'approach', 'mix', 'placement']

LDA with Gensim

First, we are creating a dictionary from the data, then convert to bag-of-words corpus and save the dictionary and corpus for future use.

```
from gensim import corpora
dictionary = corpora.Dictionary(text_data)corpus =
[dictionary.doc2bow(text) for text in text_data]
```

```
import pickle
pickle.dump(corpus, open('corpus.pkl', 'wb'))
dictionary.save('dictionary.gensim')
```

We are asking LDA to find 5 topics in the data:

```
import gensim
NUM_TOPICS = 5
ldamodel = gensim.models.ldamodel.LdaModel(corpus,
num_topics = NUM_TOPICS, id2word=dictionary, passes=15)
ldamodel.save('model5.gensim')
```

```
topics = ldamodel.print_topics(num_words=4)
for topic in topics:
    print(topic)
```

(0, '0.034"processor" + 0.019*"database" + 0.019*"issue" + 0.019*"overview"')*
(1, '0.051"computer" + 0.028*"design" + 0.028*"graphics" + 0.028*"gallery"')*
(2, '0.050"management" + 0.027*"object" + 0.027*"circuit" + 0.027*"efficient"')*

(3, '0.019"cognitive" + 0.019*"radio" + 0.019*"network" + 0.019*"distribute"')*
(4, '0.029"circuit" + 0.029*"system" + 0.029*"rigorous" + 0.029*"integration"')*

Topic 0 includes words like "processor", "database", "issue" and "overview", sounds like a topic related to database. Topic 1 includes words like "computer", "design", "graphics" and "gallery", it is definite a graphic design related topic. Topic 2 includes words like "management", "object", "circuit" and "efficient", sounds like a corporate management related topic. And so on.

With LDA, we can see that different document with different topics, and the discriminations are obvious.

Let's try a new document:

```
new_doc = 'Practical Bayesian Optimization of Machine
Learning Algorithms'
new_doc = prepare_text_for_lda(new_doc)
new_doc_bow = dictionary.doc2bow(new_doc)
print(new_doc_bow)
print(ldamodel.get_document_topics(new_doc_bow))
```

[(38, 1), (117, 1)]
[(0, 0.06669136), (1, 0.40170625), (2, 0.06670282), (3, 0.39819494), (4, 0.066704586)]

My new document is about machine learning algorithms, the LDA out put shows that topic 1 has the highest probability assigned, and topic 3 has the second highest probability assigned. We agreed!

Remember that the above 5 probabilities add up to 1.

Now we are asking LDA to find 3 topics in the data:

```
ldamodel = gensim.models.ldamodel.LdaModel(corpus,
num_topics = 3, id2word=dictionary, passes=15)
ldamodel.save('model3.gensim')
topics = ldamodel.print_topics(num_words=4)
for topic in topics:
    print(topic)
```

(0, '0.029"processor" + 0.016*"management" + 0.016*"aid" + 0.016*"algorithm"')*
(1, '0.026"radio" + 0.026*"network" + 0.026*"cognitive" + 0.026*"efficient"')*
(2, '0.029"circuit" + 0.029*"distribute" + 0.016*"database" + 0.016*"management"')*

We can also find 10 topics:

```
ldamodel = gensim.models.ldamodel.LdaModel(corpus,
num_topics = 10, id2word=dictionary, passes=15)
ldamodel.save('model10.gensim')
topics = ldamodel.print_topics(num_words=4)
for topic in topics:
    print(topic)
```

(0, '0.055"database" + 0.055*"system" + 0.029*"technical" + 0.029*"recursive"')*
(1, '0.038"distribute" + 0.038*"graphics" + 0.038*"regenerate" + 0.038*"exact"')*
(2, '0.055"management" + 0.029*"multiversion" + 0.029*"reference" + 0.029*"document"')*
(3, '0.046"circuit" + 0.046*"object" + 0.046*"generation" + 0.046*"transformation"')*
(4, '0.008"programming" + 0.008*"circuit" + 0.008*"network" + 0.008*"surface"')*
(5, '0.061"radio" + 0.061*"cognitive" + 0.061*"network" + 0.061*"connectivity"')*
(6, '0.085"programming" + 0.008*"circuit" + 0.008*"subdivision" + 0.008*"management"')*
(7, '0.041"circuit" + 0.041*"design" + 0.041*"processor" + 0.041*"instruction"')*
(8, '0.055"computer" + 0.029*"efficient" + 0.029*"channel" + 0.029*"cooperation"')*
(9, '0.061"stimulation" + 0.061*"sensor" + 0.061*"retinal" + 0.061*"pixel"')*

pyLDAvis

pyLDAvis is designed to help users interpret the topics in a topic model that has been fit to a corpus of text data. The package extracts information from a fitted LDA topic model to inform an interactive web-based visualization.

Visualizing 5 topics:

```
dictionary =
gensim.corpora.Dictionary.load('dictionary.gensim')
corpus = pickle.load(open('corpus.pkl', 'rb'))
lda = gensim.models.ldamodel.LdaModel.load('model5.gensim')
```

```
import pyLDAvis.gensim
lda_display = pyLDAvis.gensim.prepare(lda, corpus,
dictionary, sort_topics=False)
pyLDAvis.display(lda_display)
```

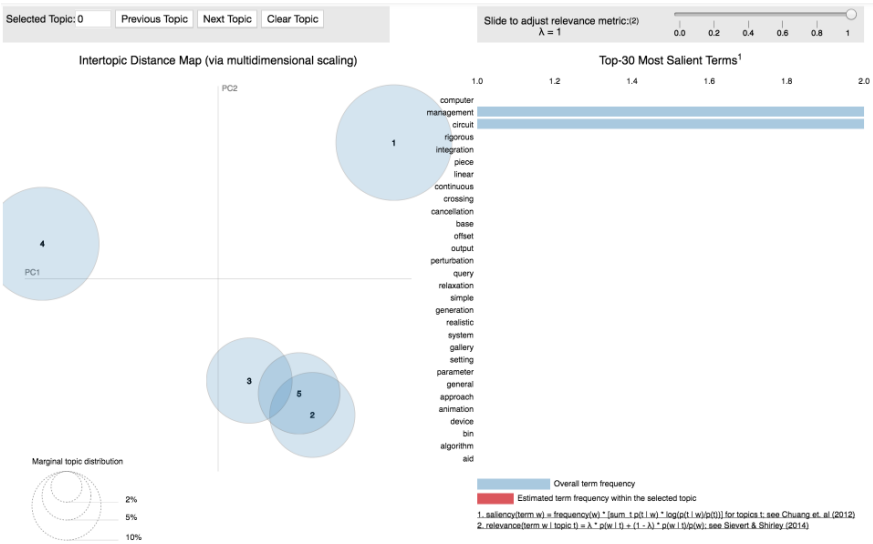


Figure 1

Saliency: a measure of how much the term tells you about the topic.

Relevance: a weighted average of the probability of the word given the topic and the word given the topic normalized by the probability of the topic.

The size of the bubble measures the importance of the topics, relative to the data.

First, we got the most salient terms, means terms mostly tell us about what's going on relative to the topics. We can also look at individual topic.

Visualizing 3 topics:

```
lda3 = gensim.models.ldamodel.LdaModel.load('model3.gensim')
lda_display3 = pyLDAvis.gensim.prepare(lda3, corpus,
dictionary, sort_topics=False)
pyLDAvis.display(lda_display3)
```

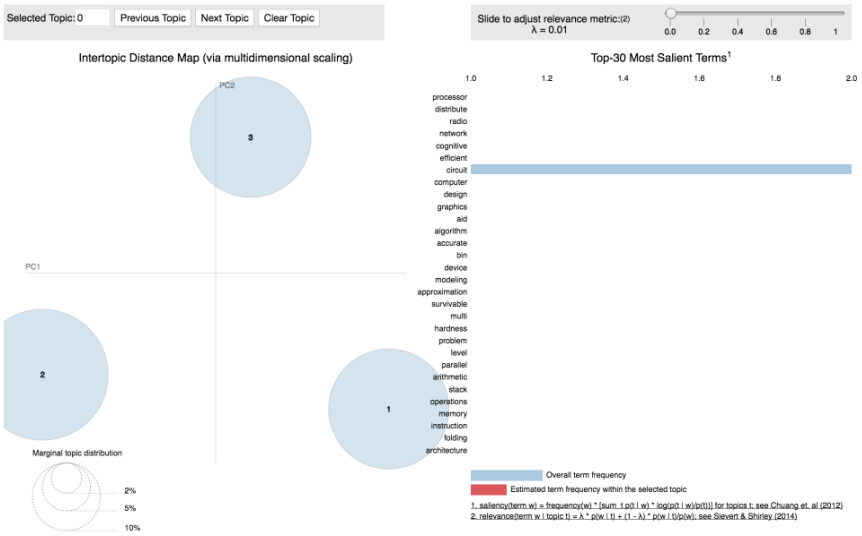


Figure 2

Visualizing 10 topics:

```
lda10 =
gensim.models.ldamodel.LdaModel.load('model10.gensim')
lda_display10 = pyLDAvis.gensim.prepare(lda10, corpus,
dictionary, sort_topics=False)
pyLDAvis.display(lda_display10)
```

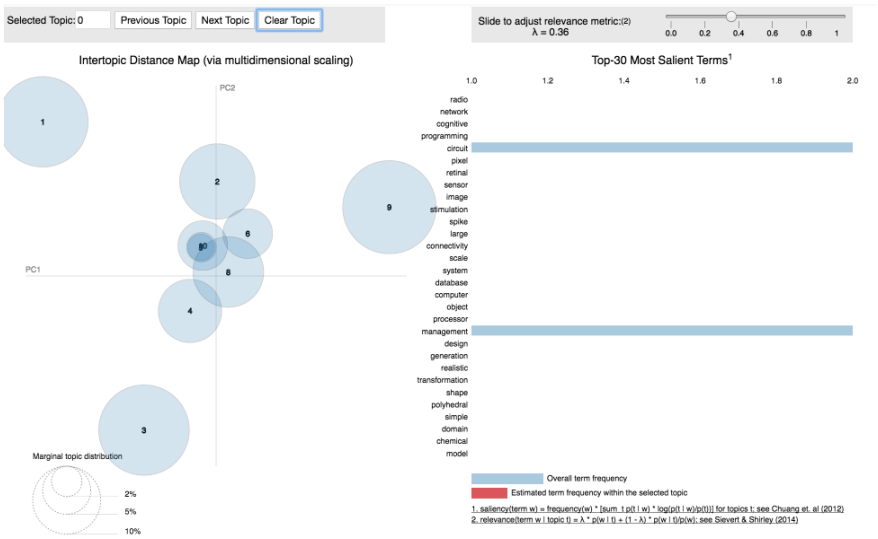


Figure 3

When we have 5 or 10 topics, we can see certain topics are clustered together, this indicates the similarity between topics. What a a nice way to visualize what we have done thus far!

Try it out, find a text dataset, remove the label if it is labeled, and build a topic model yourself!

Source code can be found on Github. I look forward to hearing any feedback or questions.