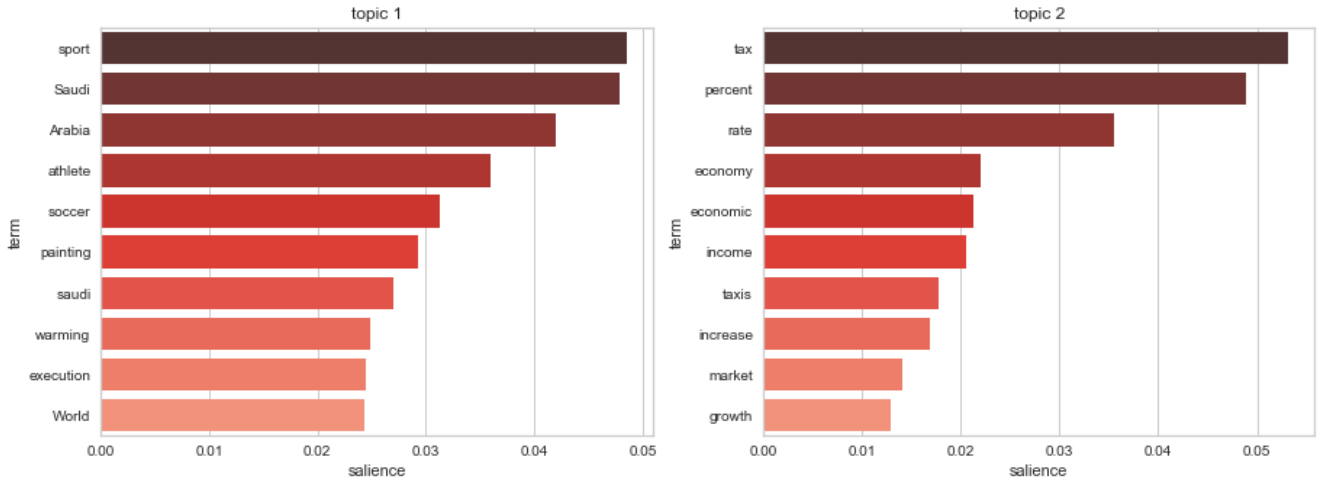
Sign in Get started

Follow 555K Followers · Editors' Picks Features Deep Dives Grow Contribute About

You have **2** free member-only stories left this month. Sign up for Medium and get an extra one

Building a Topic Modeling Pipeline with spaCy and Gensim

Jonathan Keller Sep 17, 2019 · 5 min read ★



Python, like most many programming languages, has a huge amount of exceptional libraries and modules to choose from. Generally of course, this is absolutely brilliant, but it also means that sometimes the modules don't always play nicely with each other. In this short tutorial, I'm going to show you how to link up spaCy with Gensim to create a coherent topic modeling pipeline.

Yeah, great. What's a topic model?

Good question. I thought about re-writing the Wikipedia definition, then thought that I probably should just give you the Wikipedia definition:

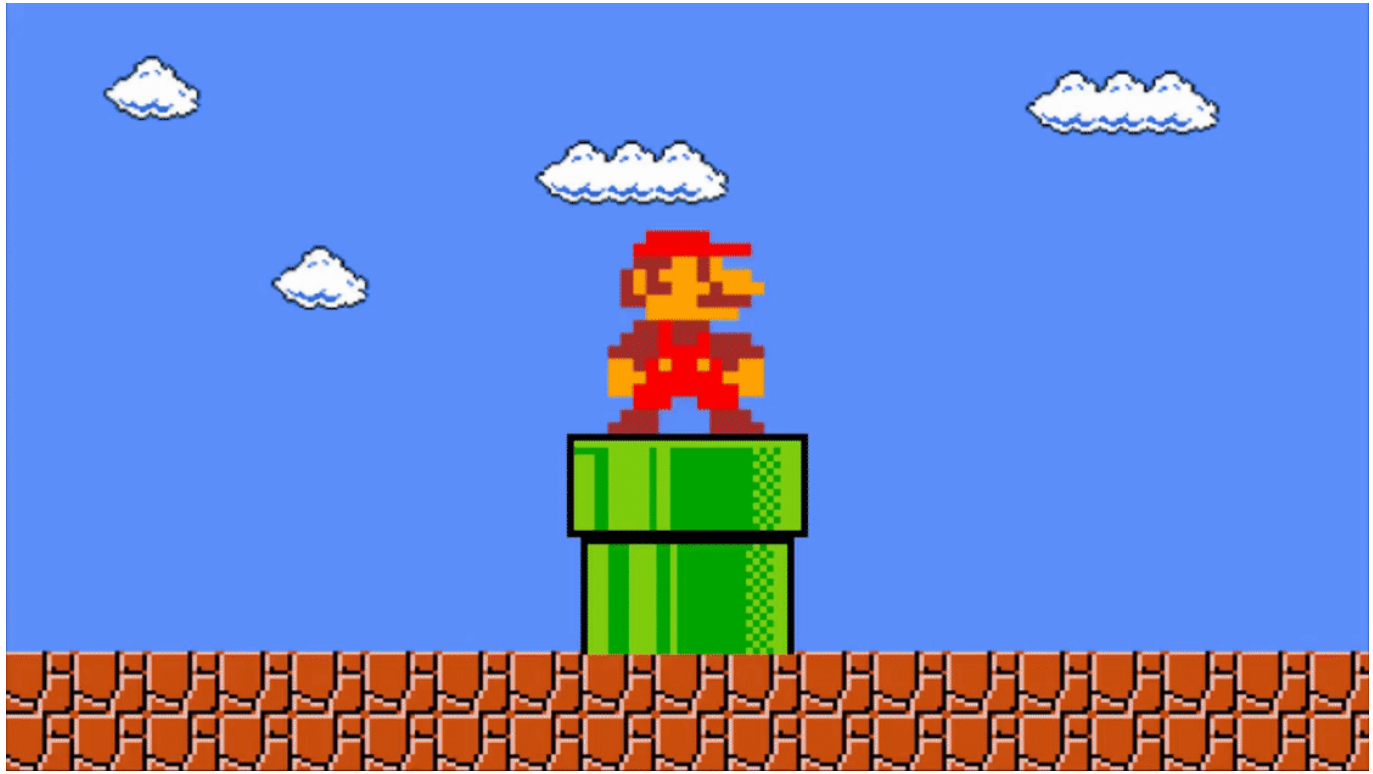
> *In machine learning and natural language processing, a* ***topic model*** *is a type of statistical model for discovering the abstract "topics" that occur in a collection of documents.*

Basically, we're looking for what collections of words, or topics, are most relevant to discussing the content of the corpus. For this tutorial we'll be using Latent Dirichlet Allocation (LDA).

A short note about libraries

For those of you folks who aren't aware, Gensim is one of the pre-eminent libraries for topic modeling. Meanwhile, spaCy is a powerful natural language processing library that has won a lot of admirers in the last few years.

Building the pipeline



First things first, let's import our libraries:

```
import numpy as np
import pandas as pd

import gensim
import gensim.corpora as corpora
from gensim.utils import simple_preprocess
from gensim.models import CoherenceModel

import spacy
from spacy.lemmatizer import Lemmatizer
```

```
from spacy.lang.en.stop_words import STOP_WORDS
import en_core_web_lg

from tqdm import tqdm_notebook as tqdm
from pprint import pprint
```

gistfile1.txt hosted with ❤ by **GitHub** **view raw**

A couple of things to note here:

- If you're new to data science in general and Python in particular, note that you'll have to pip install both Gensim and spaCy.
- spaCy has several different models to choose from. I'm using the large, general purpose web module for illustration purposes, but use what makes sense to you. This will need to be downloaded through the command line. Check out the model page here for more info.
- If you haven't used tqdm before, it's a module that allows you to create progress bars to track how long your code is taking to process.
- pprint is to make our topics formatted a little nicer when we take a look.

We're going to use a dataset of NY Times front page articles. The actual articles themselves are in the "content" column of our dataframe.

```
1  nytimes = pd.read_csv('nytimes.csv')
2  newest_doc = nytimes['content']
```

gistfile2.txt hosted with ❤ by **GitHub** **view raw**

Now for the fun part - we'll build the pipeline! The default spaCy pipeline is laid out like this:

- **Tokenizer**: Breaks the full text into individual tokens.
- **Tagger**: Tags each token with the part of speech.
- **Parser**: Parses into noun chunks, amongst other things.
- **Named Entity Recognizer (NER)**: Labels named entities, like U.S.A.

We don't really need all of these elements as we ultimately won't be passing spaCy's native output (.docs composed of .tokens), so tagged information for example, won't pass through. The NER is going to be useful to us however. I'll leave them all in for now, again for illustration purposes although it'll make the pipeline somewhat less efficient.

spaCy also allows you to build a custom pipeline using your own functions, in addition to what they have out of the box, and that's where we will be

getting the real value. spaCy has a robust stop words list and lemmatizer built in, but we'll need to add that functionality into the pipeline.

For those of you not familiar with stop words, they are basically common words that don't really add a lot of predictive value to your model. If you don't remove the word "the" from your corpus for example, it will likely show up in every topic you generate, given how often "the" is used in the English language.

Note that no word, even one as ubiquitous as "the" is automatically a stop word. Stop words are usually determined by the particular task at hand. For example, my task is to topic model front page articles from *The New York Times*, I'll have stop words that you wouldn't consider in a different context. I can add them in to spaCy's default stop words list like so:

```
nlp= spacy.load("en")

# My list of stop words.
stop_list = ["Mrs.","Ms.","say","WASHINGTON","'s","Mr.",]

# Updates spaCy's default stop words list with my additional words.
nlp.Defaults.stop_words.update(stop_list)

# Iterates over the words in the stop words list and resets the "is_stop" flag.
for word in STOP_WORDS:
```

```
11      lexeme = nlp.vocab[word]
12      lexeme.is_stop = True
```

gistfile3.txt hosted with ❤ by **GitHub** **view raw**

I haven't gone too crazy with my stop word updates, just adding a few select ones to show you my thought process:

- *The New York Times* normally uses honorifics in its articles, so we're going to get a lot of "Mr.", "Mrs.", etc…that we don't need.
- Since the articles will be filled with quotes, we're also adding "say" to our list.
- The byline is included with many articles and since most are political, "WASHINGTON" comes up frequently.
- Finally, spaCy is smart enough to break some contractions into their component words when lemmatized (like "isn't" into "is" and "not"), but that's not the case for "'s". So that'll need to go as well.
- Another big caveat here is that I'm not doing any ngrams here, which is when you are looking for the most common two and above word phrases in the text. I would probably make slightly different choices in that case.

We'll also need to lemmatize the texts, which is simply reducing each word to it's root. So for example, "going" and "goes" both are reduced to "go". "Better" would be reduced to "good".

Ok, so now let's build out our pipeline, adding functions for both lemmatization and stop word removal:

```
def lemmatizer(doc):
    # This takes in a doc of tokens from the NER and lemmatizes them.
    # Pronouns (like "I" and "you" get lemmatized to '-PRON-', so I'm removing those.
    doc = [token.lemma_ for token in doc if token.lemma_ != '-PRON-']
    doc = u' '.join(doc)
    return nlp.make_doc(doc)

def remove_stopwords(doc):
    # This will remove stopwords and punctuation.
    # Use token.text to return strings, which we'll need for Gensim.
    doc = [token.text for token in doc if token.is_stop != True and token.is_punct != Tr
    return doc

# The add_pipe function appends our functions to the default pipeline.
nlp.add_pipe(lemmatizer,name='lemmatizer',after='ner')
nlp.add_pipe(remove_stopwords, name="stopwords", last=True)
```

gistfile4.txt hosted with ❤ by **GitHub** **view raw**

The important thing here is to make sure you're returning token.text for your final output. That will turn each token object into a string object, so it can be used in Gensim.

Now for Gensim, I'm ultimately going to need to create a list of lists:

```
doc_list = []
# Iterates through each article in the corpus.
for doc in tqdm(newest_doc):
    # Passes that article through the pipeline and adds to a new list.
    pr = nlp(doc)
    doc_list.append(pr)
```

gistfile5.txt hosted with ❤ by **GitHub** **view raw**

Here's what one of those articles now looks like after having gone through the pipeline:

```
In [399]: doc_list[9]

Out[399]: ['grow',
           'campaign',
           'President',
           'Trump',
           'ardent',
           'supporter',
           'discredit',
           'special',
           'counsel',
           'Robert',
           'S.',
           'Mueller',
```

```
'law',
'enforcement',
'agency',
'assist',
'investigation',
```

Now that we've processed the articles, we'll need to create the inputs needed for our topic model:

```
# Creates, which is a mapping of word IDs to words.
words = corpora.Dictionary(doc_list)

# Turns each document into a bag of words.
corpus = [words.doc2bow(doc) for doc in doc_list]
```

gistfile6.txt hosted with ❤ by **GitHub** **view raw**

Finally we can run the model itself:

```
lda_model = gensim.models.ldamodel.LdaModel(corpus=corpus,
                                            id2word=words,
                                            num_topics=10,
                                            random_state=2,
                                            update_every=1,
                                            passes=10,
                                            alpha='auto',
                                            per_word_topics=True)
```

gistfile7.txt hosted with ❤ by **GitHub** **view raw**

And finally, some sample topics(note the pprint to help with formatting):

```
# Print the Keyword in the 10 topics
pprint(lda_model.print_topics(num_words=10))
```

```
[(0,
  '0.021*"Russia" + 0.019*"russian" + 0.015*"official" + 0.010*"email" + '
  '0.009*"Putin" + 0.009*"information" + 0.008*"intelligence" + '
  '0.007*"investigation" + 0.007*"report" + 0.007*"F.B.I."'),
 (1,
  '0.028*"$" + 0.015*"company" + 0.013*"million" + 0.011*"money" + 0.010*"pay" '
  '+ 0.009*"bank" + 0.008*"business" + 0.008*"year" + 0.008*"fund" + '
  '0.007*"financial"'),
 (2,
  '0.013*"State" + 0.011*"Islamic" + 0.011*"attack" + 0.009*"official" + '
  '0.008*"Syria" + 0.008*"military" + 0.008*"group" + 0.007*"government" + '
  '0.007*"american" + 0.006*"force"'),
 (3,
  '0.014*"case" + 0.013*"law" + 0.011*"court" + 0.009*"lawyer" + '
  '0.008*"Justice" + 0.008*"Department" + 0.007*"federal" + 0.006*"official" + '
  '0.006*"investigation" + 0.006*"state"'),
 (4,
  '0.012*"United" + 0.010*"country" + 0.010*"Obama" + 0.010*"States" + '
  '0.008*"Iran" + 0.007*"leader" + 0.007*"deal" + 0.007*"Israel" + '
  '0.006*"nuclear" + 0.006*"american"'),
 (5,
```

And that's it! Hope it was helpful!

Sign up for The Variable

By Towards Data Science

Every Thursday, the Variable delivers the very best of Towards Data Science: from hands-on tutorials and cutting-edge research to original features you don't want to

miss. Take a look.

Get this newsletter

You'll need to sign in or create an account to receive this newsletter.

Machine Learning Spacy Gensim NLP Pipeline

About Write Help Legal