Topic Modeling with Gensim (Python)

by Selva

Prabhakaran /

Topic Modeling is a technique to extract the hidden topics from large volumes of text. Latent Dirichlet Allocation(LDA) is a popular algorithm for topic modeling with excellent implementations in the Python's Gensim package. The challenge, however, is how to extract good quality of topics that are clear, segregated and meaningful. This depends heavily on the quality of text preprocessing and the strategy of finding the optimal number of topics. This tutorial attempts to tackle both of these problems.

1.

Contents

1. Introduction

One of the primary applications of natural language processing is to automatically extract what topics people are discussing from large volumes of text. Some examples of large text could be feeds from social media, customer reviews of hotels, movies, etc, user feedbacks, news stories, e-mails of customer complaints etc.

Knowing what people are talking about and understanding their problems and opinions is highly valuable to businesses, administrators, political campaigns. And it's really hard to manually read through such large volumes and compile the topics.

Thus is required an automated algorithm that can read through the text documents and automatically output the topics discussed.

In this tutorial, we will take a real example of the '20 Newsgroups' dataset and use LDA to extract the naturally discussed topics.

I will be using the Latent Dirichlet Allocation (LDA) from Gensim package along with the Mallet's implementation (via Gensim). Mallet has an efficient implementation of the LDA. It is known to run faster and gives better topics segregation.

We will also extract the volume and percentage contribution of each topic to get an idea of how important a topic is.

Let's begin!

Topic Modeling with Gensim in Python. Photo by Jeremy Bishop.

2. Prerequisites – Download nltk stopwords and spacy model

We will need the `stopwords` from NLTK and spacy's `en` model for text pre-processing. Later, we will be using the spacy model for lemmatization.

Lemmatization is nothing but converting a word to its root word. For example: the lemma of the word 'machines' is 'machine'. Likewise, 'walking' –> 'walk', 'mice' –> 'mouse' and so on.

```
 # Run in python console
import nltk; nltk.download('stopwords')

# Run in terminal or command prompt
python3 -m spacy download en
```

3. Import Packages

The core packages used in this tutorial are `re`, `gensim`, `spacy` and `pyLDAvis`. Besides this we will also using `matplotlib`, `numpy` and `pandas` for data handling and visualization. Let's import them.

```
 import re
import numpy as np
import pandas as pd
from pprint import pprint

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

# spacy for lemmatization
import spacy

# Plotting tools
import pyLDAvis
import pyLDAvis.gensim  # don't skip this
import matplotlib.pyplot as plt
%matplotlib inline

# Enable logging for gensim - optional
import logging
logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', level=logging.ERROR)

import warnings
warnings.filterwarnings("ignore",category=DeprecationWarning)
```

LDA’s

4. What does LDA do?

approach to topic modeling is it considers each document as a collection of topics in a certain proportion. And each topic as a collection of keywords, again, in a certain proportion.

Once you provide the algorithm with the number of topics, all it does it to rearrange the topics distribution within the documents and keywords distribution within the topics to obtain a good composition of topic-keywords distribution.

When I say topic, what is it actually and how it is represented?

A topic is nothing but a collection of dominant keywords that are typical representatives. Just by looking at the keywords, you can identify what the topic is all about.

The following are key factors to obtaining good segregation topics:

1. The quality of text processing.
2. The variety of topics the text talks about.
3. The choice of topic modeling algorithm.
4. The number of topics fed to the algorithm.
5. The algorithms tuning parameters.

5. Prepare Stopwords

We have already downloaded the stopwords. Let’s import them and make it available in `stop_words`.

```
# NLTK Stop words
from nltk.corpus import stopwords
stop_words = stopwords.words('english')
stop_words.extend(['from', 'subject', 're', 'edu', 'use'])
```

6. Import Newsgroups Data

We will be using the 20-Newsgroups dataset for this exercise. This version of the dataset contains about 11k newsgroups posts from 20 different topics. This is available as newsgroups.json.

This is imported using `pandas.read_json` and the resulting dataset has 3 columns as shown.

```
 # Import Dataset
df = pd.read_json('https://raw.githubusercontent.com/selva86/datasets/master/newsgroups.json')
print(df.target_names.unique())
df.head()
```

```
 ['rec.autos' 'comp.sys.mac.hardware' 'rec.motorcycles' 'misc.forsale'
 'comp.os.ms-windows.misc' 'alt.atheism' 'comp.graphics'
 'rec.sport.baseball' 'rec.sport.hockey' 'sci.electronics' 'sci.space'
 'talk.politics.misc' 'sci.med' 'talk.politics.mideast'
 'soc.religion.christian' 'comp.windows.x' 'comp.sys.ibm.pc.hardware'
 'talk.politics.guns' 'talk.religion.misc' 'sci.crypt']
```

20 Newsgroups Dataset

7. Remove emails and newline characters

As you can see there are many emails, newline and extra spaces that is quite distracting. Let's get rid of them using regular expressions.

```
 # Convert to list
data = df.content.values.tolist()

# Remove Emails
data = [re.sub('\S*@\S*\s?', '', sent) for sent in data]

# Remove new line characters
data = [re.sub('\s+', ' ', sent) for sent in data]

# Remove distracting single quotes
data = [re.sub("\'", "", sent) for sent in data]

pprint(data[:1])
```

```
['From: (wheres my thing) Subject: WHAT car is this!? Nntp-Posting-Host: '
 'rac3.wam.umd.edu Organization: University of Maryland, College Park Lines: '
 '15 I was wondering if anyone out there could enlighten me on this car I saw '
 'the other day. It was a 2-door sports car, looked to be from the late 60s/ '
 'early 70s. It was called a Bricklin. The doors were really small. In '
 'addition, the front bumper was separate from the rest of the body. This is '
 'all I know.  (..truncated..)]
```

After removing the emails and extra spaces, the text still looks messy. It is not ready for the LDA to consume. You need to break down each sentence into a list of words through tokenization, while clearing up all the messy text in the process.

Gensim's simple_preprocess is great for this.

8. Tokenize words and Clean-up text

Let's tokenize each sentence into a list of words, removing punctuations and unnecessary characters altogether.

Gensim's simple_preprocess() is great for this. Additionally I have set deacc=True to remove the punctuations.

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

data_words = list(sent_to_words(data))

print(data_words[:1])
```

```
[['from', 'wheres', 'my', 'thing', 'subject', 'what', 'car', 'is', 'this', 'nntp', 'posting', 'hos
```

9. Creating Bigram and Trigram Models

Bigrams are two words

frequently occurring together in the document. Trigrams are 3 words frequently occurring.

Some examples in our example are: 'front_bumper', 'oil_leak', 'maryland_college_park' etc.

Gensim's **Phrases** model can build and implement the bigrams, trigrams, quadgrams and more. The two important arguments to **Phrases** are **min_count** and **threshold**. The higher the values of these param, the harder it is for words to be combined to bigrams.

```
 # Build the bigram and trigram models
bigram = gensim.models.Phrases(data_words, min_count=5, threshold=100) # higher threshold fewer phr
trigram = gensim.models.Phrases(bigram[data_words], threshold=100)

# Faster way to get a sentence clubbed as a trigram/bigram
bigram_mod = gensim.models.phrases.Phraser(bigram)
trigram_mod = gensim.models.phrases.Phraser(trigram)

# See trigram example
print(trigram_mod[bigram_mod[data_words[0]]])
```

```
 ['from', 'wheres', 'my', 'thing', 'subject', 'what', 'car', 'is', 'this', 'nntp_posting_host', 'ra
```

10. Remove Stopwords, Make Bigrams and Lemmatize

The bigrams model is ready. Let's define the functions to remove the stopwords, make bigrams and lemmatization and call them sequentially.

```
 # Define functions for stopwords, bigrams, trigrams and lemmatization
def remove_stopwords(texts):
    return [[word for word in simple_preprocess(str(doc)) if word not in stop_words] for doc in tex

def make_bigrams(texts):
    return [bigram_mod[doc] for doc in texts]

def make_trigrams(texts):
    return [trigram_mod[bigram_mod[doc]] for doc in texts]

def lemmatization(texts, allowed_postags=['NOUN', 'ADJ', 'VERB', 'ADV']):
    """https://spacy.io/api/annotation"""
    texts_out = []
    for sent in texts:
        doc = nlp(" ".join(sent))
        texts_out.append([token.lemma_ for token in doc if token.pos_ in allowed_postags])
    return texts_out
```

Let's call the functions in order.

```
 # Remove Stop Words
data_words_nostops = remove_stopwords(data_words)

# Form Bigrams
data_words_bigrams = make_bigrams(data_words_nostops)

# Initialize spacy 'en' model, keeping only tagger component (for efficiency)
# python3 -m spacy download en
nlp = spacy.load('en', disable=['parser', 'ner'])

# Do lemmatization keeping only noun, adj, vb, adv
data_lemmatized = lemmatization(data_words_bigrams, allowed_postags=['NOUN', 'ADJ', 'VERB', 'ADV'])

print(data_lemmatized[:1])
```

```
 [['where', 's', 'thing', 'car', 'nntp_post', 'host', 'rac_wam', 'umd', 'organization', 'university
```

11. Create the Dictionary and Corpus needed for Topic Modeling

The two main inputs to the LDA topic model are the dictionary(id2word) and the corpus. Let's create them.

```
 # Create Dictionary
id2word = corpora.Dictionary(data_lemmatized)

# Create Corpus
texts = data_lemmatized

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

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

```
 [[(0, 1), (1, 2), (2, 1), (3, 1), (4, 1), (5, 1), (6, 5), (7, 1), (8, 1), (9, 2), (10, 1), (11, 1)
```

Gensim creates a unique id for each word in the document. The produced corpus shown above is a mapping of (word_id, word_frequency).

For example, (0, 1) above implies, word id 0 occurs once in the first document. Likewise, word id 1 occurs twice and so on.

This is used as the input by the LDA model.

If you want to see what word a given id corresponds to, pass the id as a key to the dictionary.

```
id2word[0]
```

```
'addition'
```

Or, you can see a human-readable form of the corpus itself.

```
 # Human readable format of corpus (term-frequency)
[[(id2word[id], freq) for id, freq in cp] for cp in corpus[:1]]
```

```
[[('addition', 1),
  ('anyone', 2),
  ('body', 1),
  ('bricklin', 1),
  ('bring', 1),
  ('call', 1),
  ('car', 5),
  ('could', 1),
  ('day', 1),
  ('door', 2),
  ('early', 1),
  ('engine', 1),
  ('enlighten', 1),
  ('front_bumper', 1),
  ('maryland_college', 1),
  (..truncated..)]]
```

Alright, without digressing further let's jump back on track with the next step: Building the topic model.

12. Building the Topic Model

We have everything required to train the LDA model. In addition to the corpus and dictionary, you need to provide the number of topics as well.

Apart from that, **alpha** and **eta** are hyperparameters that affect sparsity of the topics. According to the Gensim docs, both defaults to 1.0/num_topics prior.

chunksize is the number of documents to be used in each training chunk. update_every determines how often the model parameters should be updated and passes is the total number of training passes.

```
# Build LDA model
lda_model = gensim.models.ldamodel.LdaModel(corpus=corpus,
                                           id2word=id2word,
                                           num_topics=20,
                                           random_state=100,
                                           update_every=1,
                                           chunksize=100,
                                           passes=10,
                                           alpha='auto',
                                           per_word_topics=True)
```

13. View the topics in LDA model

The above LDA model is built with 20 different topics where each topic is a combination of keywords and each keyword contributes a certain weightage to the topic.

You can see the keywords for each topic and the weightage(importance) of each keyword using lda_model.print_topics() as shown next.

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

```
[(0,
  '0.016*"car" + 0.014*"power" + 0.010*"light" + 0.009*"drive" + 0.007*"mount" '
  '+ 0.007*"controller" + 0.007*"cool" + 0.007*"engine" + 0.007*"back" + '
  '0.006*"turn"'),
 (1,
  '0.072*"line" + 0.066*"organization" + 0.037*"write" + 0.032*"article" + '
  '0.028*"university" + 0.027*"nntp_post" + 0.026*"host" + 0.016*"reply" + '
  '0.014*"get" + 0.013*"thank"'),
 (2,
  '0.017*"patient" + 0.011*"study" + 0.010*"slave" + 0.009*"wing" + '
  '0.009*"disease" + 0.008*"food" + 0.008*"eat" + 0.008*"pain" + '
  '0.007*"treatment" + 0.007*"syndrome"'),
 (3,
  '0.013*"key" + 0.009*"use" + 0.009*"may" + 0.007*"public" + 0.007*"system" + '
  '0.007*"order" + 0.007*"government" + 0.006*"state" + 0.006*"provide" + '
  '0.006*"law"'),
 (4,
  '0.568*"ax" + 0.007*"rlk" + 0.005*"tufts_university" + 0.004*"ei" + '
  '0.004*"m" + 0.004*"vesa" + 0.004*"differential" + 0.004*"chz" + 0.004*"lk" '
  '+ 0.003*"weekly"'),
 (5,
  '0.029*"player" + 0.015*"master" + 0.015*"steven" + 0.009*"tor" + '
  '0.009*"van" + 0.008*"king" + 0.008*"scripture" + 0.007*"cal" + '
  '0.007*"helmet" + 0.007*"det"'),
 (6,
  '0.028*"system" + 0.020*"problem" + 0.019*"run" + 0.018*"use" + 0.016*"work" '
  '+ 0.015*"do" + 0.013*"window" + 0.013*"driver" + 0.013*"bit" + 0.012*"set"'),
 (7,
  '0.017*"israel" + 0.011*"israeli" + 0.010*"war" + 0.010*"armenian" + '
  '0.008*"kill" + 0.008*"soldier" + 0.008*"attack" + 0.008*"government" + '
  '0.007*"lebanese" + 0.007*"greek"'),
 (8,
  '0.018*"money" + 0.018*"year" + 0.016*"pay" + 0.012*"car" + 0.010*"drug" + '
  '0.010*"president" + 0.009*"rate" + 0.008*"face" + 0.007*"license" + '
  '0.007*"american"'),
 (9,
  '0.028*"god" + 0.020*"evidence" + 0.018*"christian" + 0.012*"believe" + '
  '0.012*"reason" + 0.011*"faith" + 0.009*"exist" + 0.008*"bible" + '
  '0.008*"religion" + 0.007*"claim"'),
 (10,
  '0.030*"physical" + 0.028*"science" + 0.012*"direct" + 0.012*"st" + '
  '0.012*"scientific" + 0.009*"waste" + 0.009*"jeff" + 0.008*"cub" + '
  '0.008*"brown" + 0.008*"msg"'),
 (11,
  '0.016*"wire" + 0.011*"keyboard" + 0.011*"md" + 0.009*"pm" + 0.008*"air" + '
  '0.008*"input" + 0.008*"fbi" + 0.007*"listen" + 0.007*"tube" + '
  '0.007*"koresh"'),
 (12,
  '0.016*"motif" + 0.014*"serial_number" + 0.013*"son" + 0.013*"father" + '
  '0.011*"choose" + 0.009*"server" + 0.009*"event" + 0.009*"value" + '
  '0.007*"collin" + 0.007*"prediction"'),
 (13,
  '0.098*"_" + 0.043*"max" + 0.015*"dn" + 0.011*"cx" + 0.009*"eeg" + '
  '0.008*"gateway" + 0.008*"c" + 0.005*"mu" + 0.005*"mr" + 0.005*"eg"'),
 (14,
  '0.024*"book" + 0.009*"april" + 0.007*"group" + 0.007*"page" + '
  '0.007*"new_york" + 0.007*"iran" + 0.006*"united_state" + 0.006*"author" + '
  '0.006*"include" + 0.006*"club"'),
 (15,
  '0.020*"would" + 0.017*"say" + 0.016*"people" + 0.016*"think" + 0.014*"make" '
  '+ 0.014*"go" + 0.013*"know" + 0.012*"see" + 0.011*"time" + 0.011*"get"'),
 (16,
  '0.026*"file" + 0.017*"program" + 0.012*"window" + 0.012*"version" + '
  '0.011*"entry" + 0.011*"software" + 0.011*"image" + 0.011*"color" + '
  '0.010*"source" + 0.010*"available"'),
 (17,
  '0.027*"game" + 0.027*"team" + 0.020*"year" + 0.017*"play" + 0.016*"win" + '
  '0.010*"good" + 0.009*"season" + 0.008*"fan" + 0.007*"run" + 0.007*"score"'),
 (18,
  '0.036*"drive" + 0.024*"card" + 0.020*"mac" + 0.017*"sale" + 0.014*"cpu" + '
  '0.010*"price" + 0.010*"disk" + 0.010*"board" + 0.010*"pin" + 0.010*"chip"'),
 (19,
  '0.030*"space" + 0.010*"sphere" + 0.010*"earth" + 0.009*"item" + '
  '0.008*"launch" + 0.007*"moon" + 0.007*"mission" + 0.007*"nasa" + '
  '0.007*"orbit" + 0.006*"research"')]
```

How to interpret this?

Topic 0 is a represented as _0.016*“car”* + *0.014*“power” + 0.010*“light”* + *0.009*“drive” + 0.007*“mount”* + *0.007*“controller” + 0.007*“cool”* + *0.007*“engine” + 0.007*“back”* + *‘0.006*“turn”.

It means the top 10 keywords that contribute to this topic are: ‘car’, ‘power’, ‘light’.. and so on and the weight of ‘car’ on topic 0 is 0.016.

The weights reflect how important a keyword is to that topic.

Looking at these keywords, can you guess what this topic could be? You may summarise it either are 'cars' or 'automobiles'.

Likewise, can you go through the remaining topic keywords and judge what the topic is?

Inferring Topic from Keywords

Model

14. Compute Model Perplexity and Coherence Score

perplexity and topic coherence provide a convenient measure to judge how good a given topic model is. In my experience, topic coherence score, in particular, has been more helpful.

```
 # Compute Perplexity
print('\nPerplexity: ', lda_model.log_perplexity(corpus))  # a measure of how good the model is. lo

# Compute Coherence Score
coherence_model_lda = CoherenceModel(model=lda_model, texts=data_lemmatized, dictionary=id2word, co
coherence_lda = coherence_model_lda.get_coherence()
print('\nCoherence Score: ', coherence_lda)
```

```
 Perplexity:  -8.86067503009

Coherence Score:  0.532947587081
```

There you have a coherence score of 0.53.

Now that the LDA

15. Visualize the topics-keywords

model is built, the next step is to examine the produced topics and the associated keywords. There is no better tool than pyLDAvis package's interactive chart and is designed to work well with jupyter notebooks.

```
# Visualize the topics
pyLDAvis.enable_notebook()
vis = pyLDAvis.gensim.prepare(lda_model, corpus, id2word)
vis
```

pyLDAvis Output

So how to infer pyLDAvis's output?

Each bubble on the left-hand side plot represents a topic. The larger the bubble, the more prevalent is that topic.

A good topic model will have fairly big, non-overlapping bubbles scattered throughout the chart instead of being clustered in one quadrant.

A model with too many topics, will typically have many overlaps, small sized bubbles clustered in one region of the chart.

Alright, if you move the cursor over one of the bubbles, the words and bars on the right-hand side will update. These words are the salient keywords that form the selected topic.

We have successfully built a good looking topic model.

Given our prior knowledge of the number of natural topics in the document, finding the best model was fairly straightforward.

Upnext, we will improve upon this model by using Mallet's version of LDA algorithm and then we will focus on how to arrive at the optimal number of topics given any large corpus of text.

16. Building LDA Mallet Model

So far you have seen Gensim's inbuilt version of the LDA algorithm. Mallet's version, however, often gives a better quality of topics.

Gensim provides a wrapper to implement Mallet's LDA from within Gensim itself. You only need to download the zipfile, unzip it and provide the path to mallet in the unzipped directory to `gensim.models.wrappers.LdaMallet`. See how I have done this below.

```
# Download File: http://mallet.cs.umass.edu/dist/mallet-2.0.8.zip
mallet_path = 'path/to/mallet-2.0.8/bin/mallet' # update this path
ldamallet = gensim.models.wrappers.LdaMallet(mallet_path, corpus=corpus, num_topics=20, id2word=id2
```

```
# Show Topics
pprint(ldamallet.show_topics(formatted=False))

# Compute Coherence Score
coherence_model_ldamallet = CoherenceModel(model=ldamallet, texts=data_lemmatized, dictionary=id2wo
coherence_ldamallet = coherence_model_ldamallet.get_coherence()
print('\nCoherence Score: ', coherence_ldamallet)
```

```
[(13,
  [('god', 0.022175351915726671),
   ('christian', 0.017560827817656381),
   ('people', 0.0088794630371958616),
   ('bible', 0.008215251235200895),
   ('word', 0.0077491376899412696),
   ('church', 0.0074112053696280414),
   ('religion', 0.0071198844038407759),
   ('man', 0.0067936049221590383),
   ('faith', 0.0067469935676330757),
   ('love', 0.0064556726018458093)]),
 (1,
  [('organization', 0.10977647987951586),
   ('line', 0.10182379194445974),
   ('write', 0.097397469098389255),
   ('article', 0.082483883409554246),
   ('nntp_post', 0.079894209047330425),
   ('host', 0.069737542931658306),
   ('university', 0.066303010266865026),
   ('reply', 0.02255404338163719),
   ('distribution_world', 0.014362591143681011),
   ('usa', 0.010928058478887726)]),
 (8,
  [('file', 0.028166900140084050),
   ('line', 0.021396171035954908),
   ('problem', 0.013508104862917751),
   ('program', 0.013157894736842105),
   ('read', 0.012607564538723234),
   ('follow', 0.011106663998399040),
   ('number', 0.011056633980388232),
   ('set', 0.010522980454939631),
   ('error', 0.010172770328863986),
   ('write', 0.010039356947501835)]),
 (7,
  [('include', 0.0091670556506405262),
   ('information', 0.0088169700741662776),
   ('national', 0.0085576474249260924),
   ('year', 0.0077667133447435295),
   ('report', 0.0070406099268710129),
   ('university', 0.0070406099268710129),
   ('book', 0.0068979824697889113),
   ('program', 0.0065219646283906432),
   ('group', 0.0058866241377521916),
   ('service', 0.0057180644157460714)]),
 (..truncated..)]


Coherence Score:  0.632431683088
```

Just by changing the LDA algorithm, we increased the coherence score from .53 to .63. Not bad!

17. How to find the optimal number of topics for LDA?

My approach to finding the optimal number of topics is to build many LDA models with different values of number of topics (k) and pick the one that gives the highest coherence value.

Choosing a 'k' that marks the end of a rapid growth of topic coherence usually offers meaningful and interpretable topics. Picking an even higher value can sometimes provide more granular sub-topics.

If you see the same keywords being repeated in multiple topics, it's probably a sign that the 'k' is too large.

The `compute_coherence_values()` (see below) trains multiple LDA models and provides the models and their corresponding coherence scores.

```
def compute_coherence_values(dictionary, corpus, texts, limit, start=2, step=3):
    """
    Compute c_v coherence for various number of topics

    Parameters:
    ----------
    dictionary : Gensim dictionary
    corpus : Gensim corpus
    texts : List of input texts
    limit : Max num of topics

    Returns:
    -------
    model_list : List of LDA topic models
    coherence_values : Coherence values corresponding to the LDA model with respective number of to
    """
    coherence_values = []
    model_list = []
    for num_topics in range(start, limit, step):
        model = gensim.models.wrappers.LdaMallet(mallet_path, corpus=corpus, num_topics=num_topics,
        model_list.append(model)
        coherencemodel = CoherenceModel(model=model, texts=texts, dictionary=dictionary, coherence=
        coherence_values.append(coherencemodel.get_coherence())

    return model_list, coherence_values
```

```
# Can take a long time to run.
model_list, coherence_values = compute_coherence_values(dictionary=id2word, corpus=corpus, texts=da
```

```
# Show graph
limit=40; start=2; step=6;
x = range(start, limit, step)
plt.plot(x, coherence_values)
plt.xlabel("Num Topics")
plt.ylabel("Coherence score")
plt.legend(("coherence_values"), loc='best')
plt.show()
```

Choosing the optimal number of LDA topics

```
# Print the coherence scores
for m, cv in zip(x, coherence_values):
    print("Num Topics =", m, " has Coherence Value of", round(cv, 4))
```

```
Num Topics = 2  has Coherence Value of 0.4451
Num Topics = 8  has Coherence Value of 0.5943
Num Topics = 14  has Coherence Value of 0.6208
Num Topics = 20  has Coherence Value of 0.6438
Num Topics = 26  has Coherence Value of 0.643
Num Topics = 32  has Coherence Value of 0.6478
Num Topics = 38  has Coherence Value of 0.6525
```

If the coherence score seems to keep increasing, it may make better sense to pick the model that gave the highest CV before flattening out. This is exactly the case here.

So for further steps I will choose the model with 20 topics itself.

```
# Select the model and print the topics
optimal_model = model_list[3]
model_topics = optimal_model.show_topics(formatted=False)
pprint(optimal_model.print_topics(num_words=10))
```

```
[(0,
  '0.025*"game" + 0.018*"team" + 0.016*"year" + 0.014*"play" + 0.013*"good" + '
  '0.012*"player" + 0.011*"win" + 0.007*"season" + 0.007*"hockey" + '
  '0.007*"fan"'),
 (1,
  '0.021*"window" + 0.015*"file" + 0.012*"image" + 0.010*"program" + '
  '0.010*"version" + 0.009*"display" + 0.009*"server" + 0.009*"software" + '
  '0.008*"graphic" + 0.008*"application"'),
 (2,
  '0.021*"gun" + 0.019*"state" + 0.016*"law" + 0.010*"people" + 0.008*"case" + '
  '0.008*"crime" + 0.007*"government" + 0.007*"weapon" + 0.007*"police" + '
  '0.006*"firearm"'),
 (3,
  '0.855*"ax" + 0.062*"max" + 0.002*"tm" + 0.002*"qax" + 0.001*"mf" + '
  '0.001*"giz" + 0.001*"_" + 0.001*"ml" + 0.001*"fp" + 0.001*"mr"'),
 (4,
  '0.020*"file" + 0.020*"line" + 0.013*"read" + 0.013*"set" + 0.012*"program" '
  '+ 0.012*"number" + 0.010*"follow" + 0.010*"error" + 0.010*"change" + '
  '0.009*"entry"'),
 (5,
  '0.021*"god" + 0.016*"christian" + 0.008*"religion" + 0.008*"bible" + '
  '0.007*"life" + 0.007*"people" + 0.007*"church" + 0.007*"word" + 0.007*"man" '
  '+ 0.006*"faith"'),
 (..truncated..)]
```

Those were the topics for the chosen LDA model.

18. Finding the dominant topic in each sentence

One of the practical application of topic modeling is to determine what topic a given document is about.

To find that, we find the topic number that has the highest percentage contribution in that document.

The `format_topics_sentences()` function below nicely aggregates this information in a presentable table.

```
def format_topics_sentences(ldamodel=lda_model, corpus=corpus, texts=data):
    # Init output
    sent_topics_df = pd.DataFrame()

    # Get main topic in each document
    for i, row in enumerate(ldamodel[corpus]):
        row = sorted(row, key=lambda x: (x[1]), reverse=True)
        # Get the Dominant topic, Perc Contribution and Keywords for each document
        for j, (topic_num, prop_topic) in enumerate(row):
            if j == 0:  # => dominant topic
                wp = ldamodel.show_topic(topic_num)
                topic_keywords = ", ".join([word for word, prop in wp])
                sent_topics_df = sent_topics_df.append(pd.Series([int(topic_num), round(prop_topic,
            else:
                break
    sent_topics_df.columns = ['Dominant_Topic', 'Perc_Contribution', 'Topic_Keywords']

    # Add original text to the end of the output
    contents = pd.Series(texts)
    sent_topics_df = pd.concat([sent_topics_df, contents], axis=1)
    return(sent_topics_df)


df_topic_sents_keywords = format_topics_sentences(ldamodel=optimal_model, corpus=corpus, texts=data

# Format
df_dominant_topic = df_topic_sents_keywords.reset_index()
df_dominant_topic.columns = ['Document_No', 'Dominant_Topic', 'Topic_Perc_Contrib', 'Keywords', 'Te

# Show
df_dominant_topic.head(10)
```

Dominant Topic For Each Document

19. Find the most representative document for each topic

Sometimes just the topic keywords may not be enough to make sense of what a topic is about. So, to help with understanding the topic, you can find the documents a given topic has contributed to the most and infer the topic by reading that document. Whew!!

```
 # Group top 5 sentences under each topic
sent_topics_sorteddf_mallet = pd.DataFrame()

sent_topics_outdf_grpd = df_topic_sents_keywords.groupby('Dominant_Topic')

for i, grp in sent_topics_outdf_grpd:
    sent_topics_sorteddf_mallet = pd.concat([sent_topics_sorteddf_mallet,
                                             grp.sort_values(['Perc_Contribution'], ascending=[0]).
                                            axis=0)

# Reset Index
sent_topics_sorteddf_mallet.reset_index(drop=True, inplace=True)

# Format
sent_topics_sorteddf_mallet.columns = ['Topic_Num', "Topic_Perc_Contrib", "Keywords", "Text"]

# Show
sent_topics_sorteddf_mallet.head()
```

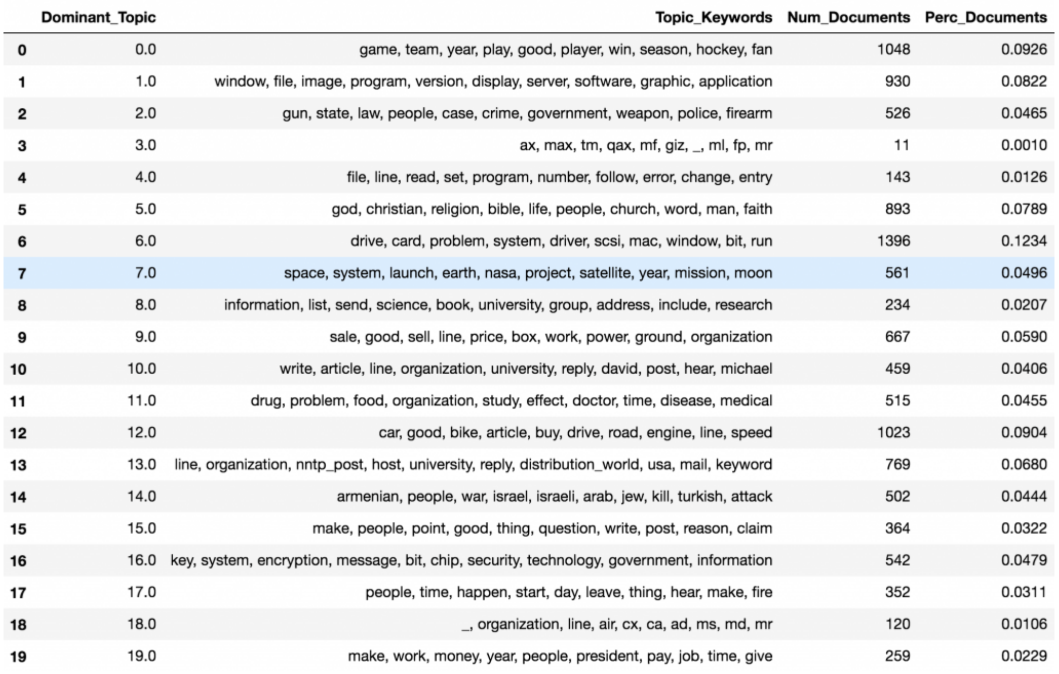
Most Representative Topic For Each Document

The tabular output above actually has 20 rows, one each for a topic. It has the topic number, the keywords, and the most representative document.
The `Perc_Contribution` column is nothing but the percentage contribution of the topic in the given document.

20. Topic distribution across documents

Finally, we want to understand the volume and distribution of topics in order to judge how widely it was discussed. The below table exposes that information.

```
 # Number of Documents for Each Topic
topic_counts = df_topic_sents_keywords['Dominant_Topic'].value_counts()

# Percentage of Documents for Each Topic
topic_contribution = round(topic_counts/topic_counts.sum(), 4)

# Topic Number and Keywords
topic_num_keywords = df_topic_sents_keywords[['Dominant_Topic', 'Topic_Keywords']]

# Concatenate Column wise
df_dominant_topics = pd.concat([topic_num_keywords, topic_counts, topic_contribution], axis=1)

# Change Column names
df_dominant_topics.columns = ['Dominant_Topic', 'Topic_Keywords', 'Num_Documents', 'Perc_Documents'

# Show
df_dominant_topics
```



	Dominant_Topic	Topic_Keywords	Num_Documents	Perc_Documents
0	0.0	game, team, year, play, good, player, win, season, hockey, fan	1048	0.0926
1	1.0	window, file, image, program, version, display, server, software, graphic, application	930	0.0822
2	2.0	gun, state, law, people, case, crime, government, weapon, police, firearm	526	0.0465
3	3.0	ax, max, tm, qax, mf, giz, _, ml, fp, mr	11	0.0010
4	4.0	file, line, read, set, program, number, follow, error, change, entry	143	0.0126
5	5.0	god, christian, religion, bible, life, people, church, word, man, faith	893	0.0789
6	6.0	drive, card, problem, system, driver, scsi, mac, window, bit, run	1396	0.1234
7	7.0	space, system, launch, earth, nasa, project, satellite, year, mission, moon	561	0.0496
8	8.0	information, list, send, science, book, university, group, address, include, research	234	0.0207
9	9.0	sale, good, sell, line, price, box, work, power, ground, organization	667	0.0590
10	10.0	write, article, line, organization, university, reply, david, post, hear, michael	459	0.0406
11	11.0	drug, problem, food, organization, study, effect, doctor, time, disease, medical	515	0.0455
12	12.0	car, good, bike, article, buy, drive, road, engine, line, speed	1023	0.0904
13	13.0	line, organization, nntp_post, host, university, reply, distribution_world, usa, mail, keyword	769	0.0680
14	14.0	armenian, people, war, israel, israeli, arab, jew, kill, turkish, attack	502	0.0444
15	15.0	make, people, point, good, thing, question, write, post, reason, claim	364	0.0322
16	16.0	key, system, encryption, message, bit, chip, security, technology, government, information	542	0.0479
17	17.0	people, time, happen, start, day, leave, thing, hear, make, fire	352	0.0311
18	18.0	_, organization, line, air, cx, ca, ad, ms, md, mr	120	0.0106
19	19.0	make, work, money, year, people, president, pay, job, time, give	259	0.0229

Topic Volume Distribution

21. Conclusion

We started with understanding what topic modeling can do. We built a basic topic model using Gensim's LDA and visualize the topics using pyLDAvis. Then we built mallet's LDA implementation. You saw how to find the optimal number of topics using coherence scores and how you can come to a logical understanding of how to choose the optimal model.