Topic Modeling

Introduction

Another popular text analysis technique is called topic modeling. The ultimate goal of topic modeling is to find various topics that are present in your corpus. Each document in the corpus will be made up of at least one topic, if not multiple topics.

In this notebook, we will be covering the steps on how to do **Latent Dirichlet Allocation (LDA)**, which is one of many topic modeling techniques. It was specifically designed for text data.

To use a topic modeling technique, you need to provide (1) a document-term matrix and (2) the number of topics you would like the algorithm to pick up.

Once the topic modeling technique is applied, your job as a human is to interpret the results and see if the mix of words in each topic make sense. If they don't make sense, you can try changing up the number of topics, the terms in the document-term matrix, model parameters, or even try a different model.

```
Resource punkt not found.
  Please use the NLTK Downloader to obtain the resource:

  >>> import nltk
  >>> nltk.download('punkt')

  For more information see: https://www.nltk.org/data.html
```

```
Resource averaged_perceptron_tagger not found.
  Please use the NLTK Downloader to obtain the resource:

  >>> import nltk
  >>> nltk.download('averaged_perceptron_tagger')
```

In [1]:
```
import nltk
```

In [2]:
```
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /home/gong/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Out[2]: True

In [3]:
```
nltk.download('averaged_perceptron_tagger')
```

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /home/gong/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

Out[3]: True

Topic Modeling - Attempt #1 (All Text)

In [4]:
```python
# Let's read in our document-term matrix
import pandas as pd
import pickle

data = pd.read_pickle('dtm_stop.pkl')
data
```

Out[4]:

	aaaaah	aaaaahhhhhhh	aaaaauuugghhhhhh	aaaahhhhh	aaah	aah	abc	abcs	ability	abject	...	zee	zen	zeppelin	zero	zillion	zombie	zombies	zoning	zoo	éclair
ali	0	0	0	0	0	0	1	0	0	0	...	0	0	0	0	0	1	0	0	0	0
anthony	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
bill	1	0	0	0	0	0	0	1	0	0	...	0	0	0	1	1	1	1	1	0	0
bo	0	1	1	1	0	0	0	0	1	0	...	0	0	0	1	0	0	0	0	0	0
dave	0	0	0	0	1	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
hasan	0	0	0	0	0	0	0	0	0	0	...	2	1	0	1	0	0	0	0	0	0
jim	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
joe	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
john	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	1
louis	0	0	0	0	0	3	0	0	0	0	...	0	0	0	2	0	0	0	0	0	0
mike	0	0	0	0	0	0	0	0	0	0	...	0	0	2	1	0	0	0	0	0	0
ricky	0	0	0	0	0	0	0	0	1	1	...	0	0	0	0	0	0	0	0	1	0

12 rows × 7468 columns

In [5]:
```python
# Import the necessary modules for LDA with gensim
# Terminal / Anaconda Navigator: conda install -c conda-forge gensim
from gensim import matutils, models
import scipy.sparse

# import logging
# logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', level=logging.INFO)
```

In [6]:
```python
# One of the required inputs is a term-document matrix
tdm = data.transpose()
tdm.head()
```

Out[6]:

	ali	anthony	bill	bo	dave	hasan	jim	joe	john	louis	mike	ricky
aaaaah	0	0	1	0	0	0	0	0	0	0	0	0
aaaaahhhhhhh	0	0	0	1	0	0	0	0	0	0	0	0
aaaaauuugghhhhhh	0	0	0	1	0	0	0	0	0	0	0	0
aaaahhhhh	0	0	0	1	0	0	0	0	0	0	0	0
aaah	0	0	0	0	1	0	0	0	0	0	0	0

In [7]:

```python
# We're going to put the term-document matrix into a new gensim format, from df --> sparse matrix --> gensim corpus
sparse_counts = scipy.sparse.csr_matrix(tdm)
corpus = matutils.Sparse2Corpus(sparse_counts)
```

In [8]:

```python
# Gensim also requires dictionary of the all terms and their respective location in the term-document matrix
cv = pickle.load(open("cv_stop.pkl", "rb"))
id2word = dict((v, k) for k, v in cv.vocabulary_.items())
```

Now that we have the corpus (term-document matrix) and id2word (dictionary of location: term), we need to specify two other parameters - the number of topics and the number of passes. Let's start the number of topics at 2, see if the results make sense, and increase the number from there.

In [9]:

```python
# Now that we have the corpus (term-document matrix) and id2word (dictionary of location: term),
# we need to specify two other parameters as well - the number of topics and the number of passes
lda = models.LdaModel(corpus=corpus, id2word=id2word, num_topics=2, passes=10)
lda.print_topics()
```

Out[9]:

```
[(0,
  '0.008*"fucking" + 0.006*"want" + 0.006*"going" + 0.005*"love" + 0.005*"say" + 0.005*"fuck" + 0.005*"shit" + 0.004*"hes" + 0.004*"goes" + 0.004*"we
nt"'),
 (1,
  '0.006*"fucking" + 0.006*"shit" + 0.006*"fuck" + 0.005*"say" + 0.005*"theyre" + 0.005*"didnt" + 0.005*"hes" + 0.004*"cause" + 0.004*"thing" + 0.004
*"day"')]
```

In [10]:

```python
# LDA for num_topics = 3
lda = models.LdaModel(corpus=corpus, id2word=id2word, num_topics=3, passes=10)
lda.print_topics()
```

Out[10]:

```
[(0,
  '0.005*"love" + 0.005*"shit" + 0.005*"ok" + 0.004*"want" + 0.004*"stuff" + 0.004*"bo" + 0.004*"repeat" + 0.004*"fucking" + 0.003*"hes" + 0.003*"lo
t"'),
 (1,
  '0.006*"hes" + 0.006*"say" + 0.006*"thing" + 0.006*"fucking" + 0.005*"life" + 0.005*"theres" + 0.005*"theyre" + 0.005*"didnt" + 0.005*"went" + 0.00
5*"id"'),
 (2,
  '0.008*"fucking" + 0.006*"shit" + 0.006*"fuck" + 0.006*"going" + 0.006*"say" + 0.005*"theyre" + 0.005*"want" + 0.005*"didnt" + 0.005*"good" + 0.005
*"cause"')]
```

In [11]:

```python
# LDA for num_topics = 4
lda = models.LdaModel(corpus=corpus, id2word=id2word, num_topics=4, passes=10)
lda.print_topics()
```

Out[11]:

```
[(0,
  '0.007*"say" + 0.006*"didnt" + 0.005*"shit" + 0.005*"fucking" + 0.005*"hes" + 0.005*"good" + 0.005*"going" + 0.005*"life" + 0.005*"thing" + 0.005
*"fuck"'),
 (1,
  '0.007*"shit" + 0.007*"fuck" + 0.007*"fucking" + 0.006*"theyre" + 0.006*"cause" + 0.004*"say" + 0.004*"really" + 0.004*"little" + 0.004*"man" + 0.0
04*"day"'),
 (2,
  '0.012*"fucking" + 0.007*"went" + 0.006*"fuck" + 0.006*"love" + 0.006*"want" + 0.006*"going" + 0.006*"day" + 0.005*"good" + 0.005*"thing" + 0.004
*"stuff"'),
 (3,
  '0.007*"fucking" + 0.006*"shit" + 0.006*"want" + 0.005*"cause" + 0.005*"says" + 0.005*"didnt" + 0.005*"going" + 0.005*"goes" + 0.005*"guy" + 0.004
*"say"')]
```

These topics aren't looking too great. We've tried modifying our parameters. Let's try modifying our terms list as well.

Topic Modeling - Attempt #2 (Nouns Only)

One popular trick is to look only at terms that are from one part of speech (only nouns, only adjectives, etc.). Check out the UPenn tag set: https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html (https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html).

In [12]:

```
# Let's create a function to pull out nouns from a string of text
from nltk import word_tokenize, pos_tag

def nouns(text):
    '''Given a string of text, tokenize the text and pull out only the nouns.'''
    is_noun = lambda pos: pos[:2] == 'NN'
    tokenized = word_tokenize(text)
    all_nouns = [word for (word, pos) in pos_tag(tokenized) if is_noun(pos)]
    return ' '.join(all_nouns)
```

In [13]:

```
# Read in the cleaned data, before the CountVectorizer step
data_clean = pd.read_pickle('data_clean.pkl')
data_clean
```

Out[13]:

	transcript
ali	ladies and gentlemen please welcome to the sta...
anthony	thank you thank you thank you san francisco th...
bill	all right thank you thank you very much thank...
bo	bo what old macdonald had a farm e i e i o and...
dave	this is dave he tells dirty jokes for a living...
hasan	whats up davis whats up im home i had to bri...
jim	ladies and gentlemen please welcome to the ...
joe	ladies and gentlemen welcome joe rogan wha...
john	all right petunia wish me luck out there you w...
louis	introfade the music out lets roll hold there l...
mike	wow hey thank you thanks thank you guys hey se...
ricky	hello hello how you doing great thank you wow ...

In [14]:
```
# Apply the nouns function to the transcripts to filter only on nouns
data_nouns = pd.DataFrame(data_clean.transcript.apply(nouns))
data_nouns
```

Out[14]:

	transcript
ali	ladies gentlemen stage ali hi thank hello na s...
anthony	thank thank people i em i francisco city world...
bill	thank thank pleasure georgia area oasis i june...
bo	macdonald farm e i o farm pig e i i snort macd...
dave	jokes living stare work profound train thought...
hasan	whats davis whats home i netflix la york i son...
jim	ladies gentlemen stage mr jim jefferies thank ...
joe	ladies gentlemen joe fuck thanks phone fuckfac...
john	petunia thats hello hello chicago thank crowd ...
louis	music lets lights lights thank i i place place...
mike	wow hey thanks look insane years everyone i id...
ricky	hello thank fuck thank im gon youre weve money...

In [15]:

```python
# Create a new document-term matrix using only nouns
from sklearn.feature_extraction import text
from sklearn.feature_extraction.text import CountVectorizer

# Re-add the additional stop words since we are recreating the document-term matrix
add_stop_words = ['like', 'im', 'know', 'just', 'dont', 'thats', 'right', 'people',
                  'youre', 'got', 'gonna', 'time', 'think', 'yeah', 'said']
stop_words = text.ENGLISH_STOP_WORDS.union(add_stop_words)

# Recreate a document-term matrix with only nouns
cvn = CountVectorizer(stop_words=stop_words)
data_cvn = cvn.fit_transform(data_nouns.transcript)
data_dtmn = pd.DataFrame(data_cvn.toarray(), columns=cvn.get_feature_names())
data_dtmn.index = data_nouns.index
data_dtmn
```

Out[15]:

	aaaaahhhhhhh	aaaaauuugghhhhhh	aaaahhhhh	aah	abc	abcs	ability	abortion	abortions	abuse	...	yummy	ze	zealand	zee	zeppelin	zillion	zombie	zombies	zoo	éclair
ali	0	0	0	0	1	0	0	0	0	0	...	0	0	0	0	0	0	1	0	0	0
anthony	0	0	0	0	0	0	0	2	0	0	...	0	0	10	0	0	0	0	0	0	0
bill	0	0	0	0	0	1	0	0	0	0	...	0	1	0	0	0	1	1	1	0	0
bo	1	1	1	0	0	0	1	0	0	0	...	0	0	0	0	0	0	0	0	0	0
dave	0	0	0	0	0	0	0	0	1	0	...	0	0	0	0	0	0	0	0	0	0
hasan	0	0	0	0	0	0	0	0	0	0	...	0	0	0	1	0	0	0	0	0	0
jim	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
joe	0	0	0	0	0	0	0	0	0	1	...	0	0	0	0	0	0	0	0	0	0
john	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	1
louis	0	0	0	3	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
mike	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	2	0	0	0	0	0
ricky	0	0	0	0	0	0	1	0	0	0	...	1	0	0	0	0	0	0	0	1	0

12 rows × 4635 columns

In [16]:

```python
# Create the gensim corpus
corpusn = matutils.Sparse2Corpus(scipy.sparse.csr_matrix(data_dtmn.transpose()))

# Create the vocabulary dictionary
id2wordn = dict((v, k) for k, v in cvn.vocabulary_.items())
```

In [17]:

```python
# Let's start with 2 topics
ldan = models.LdaModel(corpus=corpusn, num_topics=2, id2word=id2wordn, passes=10)
ldan.print_topics()
```

Out[17]:

```
[(0,
  '0.009*"shit" + 0.009*"man" + 0.009*"life" + 0.009*"thing" + 0.008*"fuck" + 0.008*"hes" + 0.008*"day" + 0.007*"guy" + 0.007*"way" + 0.006*"caus
e"'),
 (1,
  '0.009*"day" + 0.007*"thing" + 0.007*"joke" + 0.006*"cause" + 0.006*"lot" + 0.005*"way" + 0.005*"things" + 0.005*"years" + 0.005*"id" + 0.005*"he
s"')]
```

In [18]:
```
# Let's try topics = 3
ldan = models.LdaModel(corpus=corpusn, num_topics=3, id2word=id2wordn, passes=10)
ldan.print_topics()
```

Out[18]:
```
[(0,
  '0.016*"shit" + 0.011*"fuck" + 0.011*"man" + 0.008*"gon" + 0.008*"lot" + 0.008*"dude" + 0.007*"hes" + 0.007*"guy" + 0.007*"thing" + 0.006*"life"'),
 (1,
  '0.011*"day" + 0.010*"thing" + 0.008*"life" + 0.007*"cause" + 0.007*"way" + 0.007*"hes" + 0.006*"dad" + 0.006*"shes" + 0.006*"things" + 0.006*"gu
y"'),
 (2,
  '0.010*"stuff" + 0.010*"bo" + 0.009*"repeat" + 0.008*"eye" + 0.007*"contact" + 0.006*"man" + 0.005*"brain" + 0.005*"story" + 0.005*"cos" + 0.004*"c
omedy"')]
```

In [19]:
```
# Let's try 4 topics
ldan = models.LdaModel(corpus=corpusn, num_topics=4, id2word=id2wordn, passes=10)
ldan.print_topics()
```

Out[19]:
```
[(0,
  '0.011*"thing" + 0.010*"day" + 0.009*"life" + 0.009*"man" + 0.009*"hes" + 0.008*"shit" + 0.007*"fuck" + 0.007*"way" + 0.007*"cause" + 0.007*"gu
y"'),
 (1,
  '0.010*"dad" + 0.010*"shit" + 0.008*"man" + 0.007*"fuck" + 0.007*"lot" + 0.007*"day" + 0.006*"life" + 0.006*"school" + 0.006*"way" + 0.006*"mom"'),
 (2,
  '0.000*"man" + 0.000*"life" + 0.000*"gon" + 0.000*"thing" + 0.000*"guy" + 0.000*"cause" + 0.000*"way" + 0.000*"lot" + 0.000*"fuck" + 0.000*"day"'),
 (3,
  '0.010*"cause" + 0.008*"point" + 0.007*"lot" + 0.007*"day" + 0.007*"kind" + 0.006*"gon" + 0.006*"shit" + 0.005*"way" + 0.005*"women" + 0.005*"nigh
t"')]
```

Topic Modeling - Attempt #3 (Nouns and Adjectives)

In [20]:
```
# Let's create a function to pull out nouns from a string of text
def nouns_adj(text):
    '''Given a string of text, tokenize the text and pull out only the nouns and adjectives.'''
    is_noun_adj = lambda pos: pos[:2] == 'NN' or pos[:2] == 'JJ'
    tokenized = word_tokenize(text)
    nouns_adj = [word for (word, pos) in pos_tag(tokenized) if is_noun_adj(pos)]
    return ' '.join(nouns_adj)
```

In [21]:

```python
# Apply the nouns function to the transcripts to filter only on nouns
data_nouns_adj = pd.DataFrame(data_clean.transcript.apply(nouns_adj))
data_nouns_adj
```

Out[21]:

	transcript
ali	ladies gentlemen welcome stage ali wong hi wel...
anthony	thank san francisco thank good people surprise...
bill	right thank thank pleasure greater atlanta geo...
bo	old macdonald farm e i i o farm pig e i i snor...
dave	dirty jokes living stare most hard work profou...
hasan	whats davis whats im home i netflix special la...
jim	ladies gentlemen welcome stage mr jim jefferie...
joe	ladies gentlemen joe fuck san francisco thanks...
john	right petunia august thats good right hello he...
louis	music lets lights lights thank much i i i nice...
mike	wow hey thanks hey seattle nice look crazy ins...
ricky	hello great thank fuck thank lovely welcome im...

In [22]:

```python
# Create a new document-term matrix using only nouns and adjectives, also remove common words with max_df
cvna = CountVectorizer(stop_words=stop_words, max_df=.8)
data_cvna = cvna.fit_transform(data_nouns_adj.transcript)
data_dtmna = pd.DataFrame(data_cvna.toarray(), columns=cvna.get_feature_names())
data_dtmna.index = data_nouns_adj.index
data_dtmna
```

Out[22]:

	aaaaah	aaaaahhhhhhh	aaaaauuugghhhhhh	aaaahhhhh	aah	abc	abcs	ability	abject	able	...	ze	zealand	zee	zeppelin	zero	zillion	zombie	zombies	zoo	éclair
ali	0	0	0	0	0	1	0	0	0	2	...	0	0	0	0	0	0	1	0	0	0
anthony	0	0	0	0	0	0	0	0	0	0	...	0	10	0	0	0	0	0	0	0	0
bill	1	0	0	0	0	0	1	0	0	1	...	1	0	0	0	0	1	1	1	0	0
bo	0	1	1	1	0	0	0	1	0	0	...	0	0	0	0	1	0	0	0	0	0
dave	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
hasan	0	0	0	0	0	0	0	0	0	1	...	0	0	2	0	0	0	0	0	0	0
jim	0	0	0	0	0	0	0	0	0	1	...	0	0	0	0	0	0	0	0	0	0
joe	0	0	0	0	0	0	0	0	0	2	...	0	0	0	0	0	0	0	0	0	0
john	0	0	0	0	0	0	0	0	0	3	...	0	0	0	0	0	0	0	0	0	1
louis	0	0	0	0	3	0	0	0	0	1	...	0	0	0	0	0	0	0	0	0	0
mike	0	0	0	0	0	0	0	0	0	0	...	0	0	0	2	0	0	0	0	0	0
ricky	0	0	0	0	0	0	0	1	1	2	...	0	0	0	0	0	0	0	0	1	0

12 rows × 5587 columns

In [23]:

```python
# Create the gensim corpus
corpusna = matutils.Sparse2Corpus(scipy.sparse.csr_matrix(data_dtmna.transpose()))

# Create the vocabulary dictionary
id2wordna = dict((v, k) for k, v in cvna.vocabulary_.items())
```

In [24]:

```python
# Let's start with 2 topics
ldana = models.LdaModel(corpus=corpusna, num_topics=2, id2word=id2wordna, passes=10)
ldana.print_topics()
```

Out[24]:

```
[(0,
  '0.007*"joke" + 0.005*"mom" + 0.004*"parents" + 0.003*"jokes" + 0.003*"bo" + 0.003*"hasan" + 0.003*"comedy" + 0.003*"clinton" + 0.003*"repeat" + 0.002*"anthony"'),
 (1,
  '0.003*"ass" + 0.003*"son" + 0.003*"gun" + 0.002*"ok" + 0.002*"dick" + 0.002*"door" + 0.002*"dog" + 0.002*"jenny" + 0.002*"guns" + 0.002*"friend"')]
```

In [25]:

```python
# Let's try 3 topics
ldana = models.LdaModel(corpus=corpusna, num_topics=3, id2word=id2wordna, passes=10)
ldana.print_topics()
```

Out[25]:

```
[(0,
  '0.008*"joke" + 0.004*"son" + 0.004*"jokes" + 0.004*"guns" + 0.004*"ahah" + 0.003*"anthony" + 0.003*"mad" + 0.003*"ass" + 0.003*"party" + 0.003*"gun"'),
 (1,
  '0.007*"mom" + 0.005*"parents" + 0.004*"ok" + 0.003*"hasan" + 0.003*"clinton" + 0.003*"dick" + 0.003*"president" + 0.003*"ass" + 0.003*"york" + 0.002*"door"'),
 (2,
  '0.005*"bo" + 0.004*"jenny" + 0.004*"repeat" + 0.003*"comedy" + 0.003*"eye" + 0.003*"contact" + 0.003*"love" + 0.003*"andy" + 0.003*"sense" + 0.003*"sad"')]
```

In [26]:

```python
# Let's try 4 topics
ldana = models.LdaModel(corpus=corpusna, num_topics=4, id2word=id2wordna, passes=10)
ldana.print_topics()
```

Out[26]:

```
[(0,
  '0.007*"bo" + 0.006*"repeat" + 0.005*"eye" + 0.005*"contact" + 0.004*"cos" + 0.004*"gun" + 0.004*"comedy" + 0.004*"jesus" + 0.003*"um" + 0.003*"brain"'),
 (1,
  '0.004*"ass" + 0.004*"jenny" + 0.004*"guns" + 0.003*"dog" + 0.003*"girls" + 0.003*"class" + 0.003*"dick" + 0.003*"morning" + 0.003*"tit" + 0.003*"date"'),
 (2,
  '0.014*"joke" + 0.006*"jokes" + 0.006*"anthony" + 0.005*"twitter" + 0.004*"grandma" + 0.004*"nuts" + 0.004*"dead" + 0.004*"jenner" + 0.004*"shark" + 0.004*"hampstead"'),
 (3,
  '0.007*"mom" + 0.004*"parents" + 0.004*"hasan" + 0.004*"clinton" + 0.004*"wife" + 0.004*"ahah" + 0.003*"friend" + 0.003*"gay" + 0.003*"husband" + 0.003*"door"')]
```

Identify Topics in Each Document

Out of the 9 topic models we looked at, the nouns and adjectives, 4 topic one made the most sense. So let's pull that down here and run it through some more iterations to get more fine-tuned topics.

In [27]:
```
# Our final LDA model (for now)
ldana = models.LdaModel(corpus=corpusna, num_topics=4, id2word=id2wordna, passes=80)
ldana.print_topics()
```

Out[27]:
```
[(0,
  '0.009*"ahah" + 0.007*"nigga" + 0.006*"gay" + 0.004*"son" + 0.004*"oj" + 0.004*"ghetto" + 0.004*"young" + 0.004*"motherfucker" + 0.004*"kevin" + 0.
003*"mad"'),
 (1,
  '0.004*"dog" + 0.004*"gun" + 0.004*"bo" + 0.004*"ass" + 0.004*"wife" + 0.003*"clinton" + 0.003*"guns" + 0.003*"repeat" + 0.003*"mom" + 0.003*"dic
k"'),
 (2,
  '0.008*"joke" + 0.005*"mom" + 0.004*"jokes" + 0.004*"parents" + 0.004*"hasan" + 0.003*"door" + 0.003*"anthony" + 0.003*"jenner" + 0.003*"mad" + 0.0
03*"twitter"'),
 (3,
  '0.007*"jenny" + 0.005*"husband" + 0.004*"ok" + 0.004*"accident" + 0.003*"pregnant" + 0.003*"friend" + 0.003*"scrambler" + 0.003*"marriage" + 0.003
*"argument" + 0.003*"andy"')]
```

These four topics look pretty decent. Let's settle on these for now.

- Topic 0: mom, parents
- Topic 1: husband, wife
- Topic 2: guns
- Topic 3: profanity

In [28]:
```
# Let's take a look at which topics each transcript contains
corpus_transformed = ldana[corpusna]
list(zip([a for [(a,b)] in corpus_transformed], data_dtmna.index))
```

Out[28]:
```
[(3, 'ali'),
 (2, 'anthony'),
 (1, 'bill'),
 (1, 'bo'),
 (0, 'dave'),
 (2, 'hasan'),
 (1, 'jim'),
 (2, 'joe'),
 (1, 'john'),
 (1, 'louis'),
 (3, 'mike'),
 (2, 'ricky')]
```

For a first pass of LDA, these kind of make sense to me, so we'll call it a day for now.

- Topic 0: mom, parents [Anthony, Hasan, Louis, Ricky]
- Topic 1: husband, wife [Ali, John, Mike]
- Topic 2: guns [Bill, Bo, Jim]
- Topic 3: profanity [Dave, Joe]

Additional Exercises

1. Try further modifying the parameters of the topic models above and see if you can get better topics.
2. Create a new topic model that includes terms from a different [part of speech (https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html)](https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html) and see if you can get better topics.

In [ ]:

```
 
```