Topic Modeling in NLP - Topic modeling - 3

One should look for what is and not what he thinks should be. (Albert Einstein)

Module completion checklist

Objective	Complete
Perform latent dirichlet allocation (LDA) on frequency counts	
Evaluate results and choose optimal number of topics	

Create a dictionary of counts

- Let's create the dictionary using `gensim.corpora.Dictionary` and look at our output using a small loop

```
# Set the seed.
np.random.seed(1)
dictionary = gensim.corpora.Dictionary(df_clean)

# The loop below iterates through the first 10 items of the dictionary and prints out the key and
value.
count = 0
for k, v in dictionary.iteritems():
    print(k, v)
    count += 1
    if count > 10:
        break
```

```
0 american
1 battl
2 brisban
3 defens
4 harrison
5 kyrgio
6 nick
7 open
8 round
9 ryan
10 start
```

Create a dictionary of counts

- We can filter out words by their frequency in the dictionary
- `.filter_extremes()` will remove all values in the dictionary that are:
    - less frequent than `no_below` documents
    - more than `no_above` documents (fraction of total corpus size, not absolute number)
    - keep only first `keep_n` most frequent tokens

```
dictionary.filter_extremes(no_below = 4, no_above = 0.5, keep_n = 200)

# How many words are left in the dictionary?
len(dictionary)
```

```
200
```

Document to bag-of-words

- Now we will use `gensim` library `doc2bow` to transform each document to a dictionary
- Each document will become a dictionary that has the number of words and has the number of times each of those words appear
- This is the object we will use to build our TF-IDF matrix

```
# We use a list comprehension to transform each doc within our df_clean object.
bow_corpus = [dictionary.doc2bow(doc) for doc in df_clean]

# Let's look at the first document.
print(bow_corpus[0])
```

```
[(0, 1), (1, 1), (2, 2), (3, 1), (4, 1), (5, 1), (6, 1)]
```

Document to bag-of-words (cont'd)

- Let's preview bag-of-words for the first document

```
# Isolate the first document.
bow_doc_1 = bow_corpus[0]

# Iterate through each dictionary item using the index.
# Print out each actual word and how many times it appears.
for i in range(len(bow_doc_1)):
    print("Word {} (\"{}\") appears {} 
time.".format(bow_doc_1[i][0], 

          dictionary[bow_doc_1[i][0]], 
          bow_doc_1[i][1]))
```

```
Word 0 ("american") appears 1 time.
Word 1 ("defens") appears 1 time.
Word 2 ("open") appears 2 time.
Word 3 ("round") appears 1 time.
Word 4 ("start") appears 1 time.
Word 5 ("tuesday") appears 1 time.
Word 6 ("victori") appears 1 time.
```

Transform counts with TfidfModel

- To transform a Document-Term Matrix, which is a “bag-of-words” representation `bow_corpus` that we created above, into `TF–IDF`, we will use `TfidfModel` from `gensim` library’s `model` module for working with text

models.tfidfmodel – TF-IDF model

This module implements functionality related to the *Term Frequency - Inverse Document Frequency* *<https://en.wikipedia.org/wiki/Tf%E2%80%93idf>* vector space bag-of-words models.

For a more in-depth exposition of TF-IDF and its various SMART variants (normalization, weighting schemes), see the blog post at https://rare-technologies.com/pivoted-document-length-normalisation/

class `gensim.models.tfidfmodel.TfidfModel`(*corpus=None, id2word=None, dictionary=None, wlocal=<function identity>, wglobal=<function df2idf>, normalize=True, smartirs=None, pivot=None, slope=0.65*)

Transform counts with TfidfModel (cont'd)

- We will now activate the `TfidfModel` function and transform our `bow_corpus`
- Our output will be the TF-IDF transformation applied to each document:

$$TF \times IDF = \frac{F_{wd}}{N_d} \times log \frac{M}{M_w}$$

```
# This is the transformation.
tfidf = models.TfidfModel(bow_corpus)

# Apply the transformation to the entire corpus.
corpus_tfidf = tfidf[bow_corpus]

# Preview TF-IDF scores for the first document.
for doc in corpus_tfidf:
    pprint(doc)
    break
```

```
[(0, 0.31942373876087665),
 (1, 0.3549009519669791),
 (2, 0.6118718565633235),
 (3, 0.3549009519669791),
 (4, 0.3059359282816618),
 (5, 0.22829905152454918),
 (6, 0.3549009519669791)]
```

LDA on `snippet`

We need the following data objects for topic modeling:

- `df_clean`: the corpus, where:
  - each 'document' is one entry in `snippet`
  - each document is cleaned, and punctuation, numbers, special characters and stop words removed
- `dictionary`: a dictionary containing the **number of times a given word appears** within the entire corpus
- `corpus_tfidf`: a Document-Term Matrix (DTM) transformed to be a weighted term frequency - inverse document frequency matrix

Let's apply the idea of LDA to our corpus of **snippet**

LDA with the gensim package

- We will continue using `gensim` and now introduce `models.LdaMulticore`

models.ldamulticore – parallelized Latent Dirichlet Allocation

Online Latent Dirichlet Allocation (LDA) in Python, using all CPU cores to parallelize and speed up model training.

The parallelization uses multiprocessing; in case this doesn't work for you for some reason, try the `gensim.models.ldamodel.LdaModel` class which is an equivalent, but more straightforward and single-core implementation.

The training algorithm:

- is **streamed**: training documents may come in sequentially, no random access required,
- runs in **constant memory** w.r.t. the number of documents: size of the training corpus does not affect memory footprint, can process corpora larger than RAM

- We are going to take our `corpus_tfidf` object we created and run `LDA` on it
- `gensim.models.LdaMulticore` is a powerful package that allows our machine to run on multiple cores (if they exist)
- We will use two for now, as most machines will have two cores
- The algorithm we just walked through with the **two documents** will now be applied to all the documents

LdaMulticore

- We run the LdaMulticore model using:

```
gensim.models.ldamulticore.LdaMulticore(corpus = None,
                                        num_topics = 100,
                                        id2word = None, workers = None,
                                        passes = 1)
```

- Before running the model, let's make sure we understand the main parameters of the model:
  - `corpus`: stream of document vectors or sparse matrix of shape
  - `num_topics`: default is 100, make sure to change according to number of topics you decide on
  - `id2word`: mapping from word IDs to words
  - `workers`: number of cores being used, if `None` then all available cores will be used
  - `passes`: number of passes through the corpus during training, e.g., how many times to classify each word to each topic

Running LdaMulticore

- Let's run the model on our transformed matrix `corpus_tfidf` using `dictionary` as the `id2word` object

```
lda_model_tfidf = gensim.models.LdaMulticore(corpus_tfidf,
                                             num_topics = 5,
                                             id2word = dictionary,
                                             workers = 4,
                                             passes = 2)
```

- We have our `LdaMulticore` object now

```
print(lda_model_tfidf)
```

```
LdaMulticore<num_terms=200, num_topics=5, decay=0.5, chunksize=2000>
```

LDA output

- We chose 5 topics, we are now going to print out each topic and the top words within the topics

```
for idx, topic in lda_model_tfidf.print_topics(-1):
    print('Topic: {} Word: {}'.format(idx, topic))
```

```
Topic: 0 Word: 0.020*"offici" + 0.020*"like" + 0.017*"say" + 0.017*"week" + 0.017*"warn" +
0.015*"show" + 0.015*"state" + 0.014*"thursday" + 0.014*"help" + 0.014*"new"
Topic: 1 Word: 0.023*"time" + 0.023*"latest" + 0.022*"world" + 0.021*"local" + 0.020*"new" +
0.016*"year" + 0.015*"meet" + 0.013*"york" + 0.013*"tenni" + 0.013*"leader"
Topic: 2 Word: 0.021*"said" + 0.020*"friday" + 0.018*"billion" + 0.018*"know" + 0.015*"set"
+ 0.015*"say" + 0.013*"accus" + 0.012*"want" + 0.012*"govern" + 0.012*"court"
Topic: 3 Word: 0.020*"investig" + 0.020*"saturday" + 0.019*"presid" + 0.017*"polic" +
0.014*"said" + 0.014*"expect" + 0.013*"citi" + 0.013*"suspect" + 0.013*"year" +
0.013*"first"
Topic: 4 Word: 0.022*"young" + 0.020*"tuesday" + 0.018*"look" + 0.018*"said" + 0.016*"talk"
+ 0.016*"close" + 0.015*"move" + 0.015*"new" + 0.015*"south" + 0.014*"year"
```

- We can interpret this by looking at the **top words by topic**
    - These are the words that contribute most to each topic
- This is a very raw version of the output, we are going to learn more about how to clean this up and interpret it later!

Classify our documents within topics

- Let’s see how we would classify our df_clean as one of the five topics

```
# Let's look at our first document as
an example:
print(df_clean[0])
```

```
['nick', 'kyrgio', 'start',
'brisban', 'open', 'titl', 'defens',
'battl', 'victori', 'american',
'ryan', 'harrison', 'open', 'round',
'tuesday']
```

```
for index, score in
sorted(lda_model_tfidf[corpus_tfidf[0]], key=lambda tup:
-1*tup[1]):
    print("\nScore: {}\t \nTopic: {}".format(score,
lda_model_tfidf.print_topic(index, 10)))
```

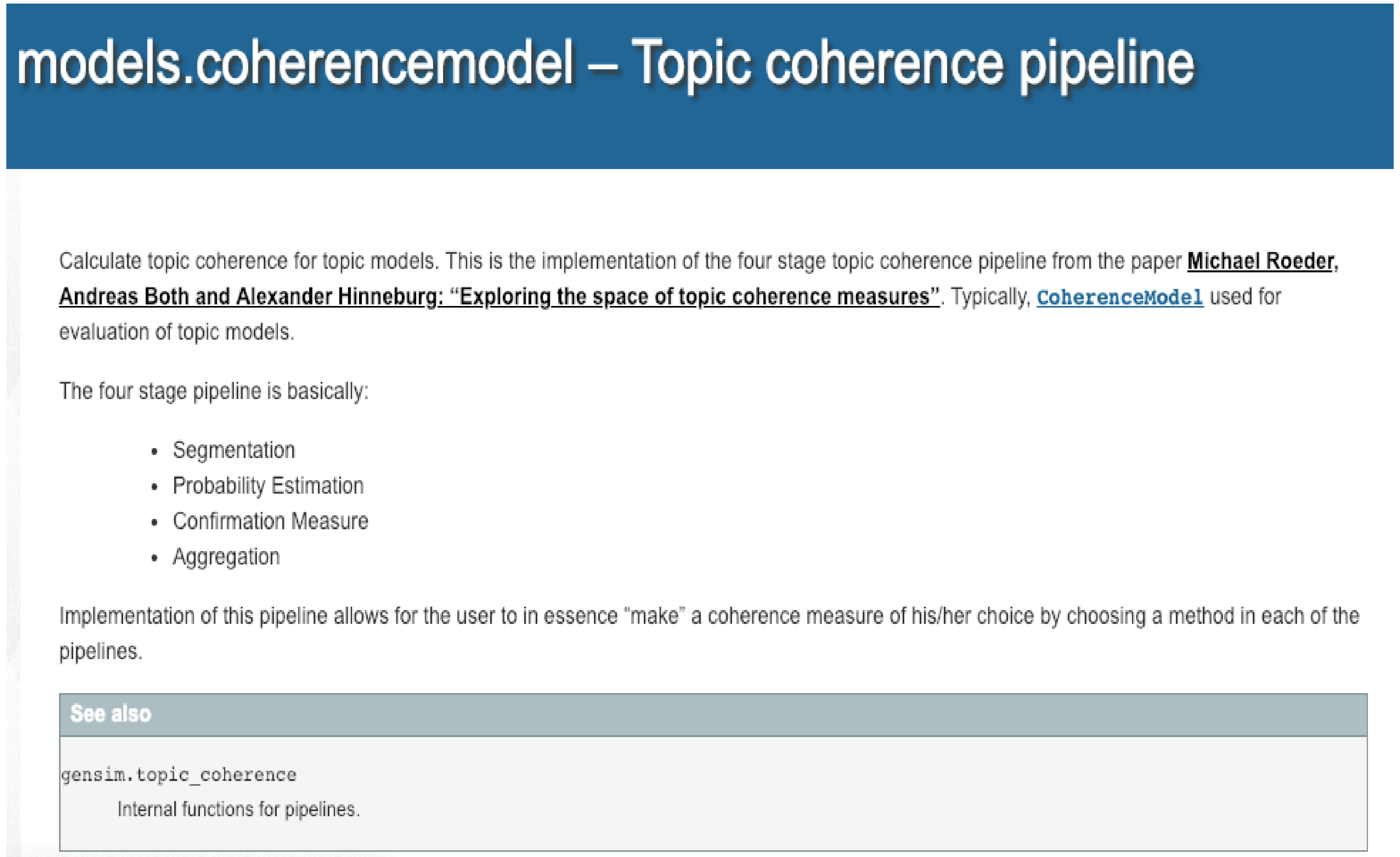
```

Score: 0.7646151185035706
Topic: 0.020*"investig" + 0.020*"saturday" +
0.019*"presid" + 0.017*"polic" + 0.014*"said" +
0.014*"expect" + 0.013*"citi" + 0.013*"suspect" +
0.013*"year" + 0.013*"first"

Score: 0.060962967574596405
Topic: 0.020*"offici" + 0.020*"like" + 0.017*"say" +
0.017*"week" + 0.017*"warn" + 0.015*"show" +
0.015*"state" + 0.014*"thursday" + 0.014*"help" +
0.014*"new"

Score: 0.058861665427684784
Topic: 0.022*"young" + 0.020*"tuesday" + 0.018*"look"
+ 0.018*"said" + 0.016*"talk" + 0.016*"close" +
0.015*"move" + 0.015*"new" + 0.015*"south" +
0.014*"year"
```

Module completion checklist

Objective	Complete
Perform latent dirichlet allocation (LDA) on frequency counts	✔
Evaluate results and choose optimal number of topics	

LDA: evaluate results

- How do we evaluate our LDA model?
- The measure we will focus on in this module is **topic coherence**
- With **topic coherence**, we will be able to evaluate our results as well as build a plot that will help us choose the optimal number of topics for our LDA model
- The function we will use is from the `gensim` package, `CoherenceModel`
- You can read the paper ***"Exploring the space of topic coherence measures"*** for an in-depth understanding of topic coherence



models.coherencemodel – Topic coherence pipeline

Calculate topic coherence for topic models. This is the implementation of the four stage topic coherence pipeline from the paper Michael Roeder, Andreas Both and Alexander Hinneburg: "Exploring the space of topic coherence measures". Typically, `CoherenceModel` used for evaluation of topic models.

The four stage pipeline is basically:

- Segmentation
- Probability Estimation
- Confirmation Measure
- Aggregation

Implementation of this pipeline allows for the user to in essence "make" a coherence measure of his/her choice by choosing a method in each of the pipelines.

See also

`gensim.topic_coherence`
Internal functions for pipelines.

Topic coherence: quick overview

- Topic coherence is based on four main concepts
  - **Segmentation**: dividing the topics into smaller subsets
  - **Probability estimation**: quantitative measure of the subtopic quality
  - **Confirmation measure**: determine quality based on some predefined measure
  - **Aggregation**: combine all quality numbers and derive one number for overall quality
- There are two measures in topic coherence
  - **Intrinsic** : compares a word only to the preceding and succeeding words respectively, so you need the ordered word set for this. It uses a pairwise score function which is the empirical conditional log-probability with smoothing count
  - **Extrinsic** : every single word is paired with every other single word
- Both intrinsic and extrinsic measures compute the coherence score c

Calculate topic coherence

- We will now calculate topic coherence on our `lda_model_tfidf`
- The parameters we need are:
  - original doc, `df_clean`
  - dictionary built of the corpora, `dictionary`
  - lda model, `lda_model_tfidf`
  - coherence metric, `c_v` (based on the ***paper*** referenced above)

```
# Compute Coherence Score using c_v.
coherence_model_lda = CoherenceModel(model = lda_model_tfidf, texts = df_clean, dictionary =
dictionary, coherence = 'c_v')
coherence_lda = coherence_model_lda.get_coherence()
```

```
print('Coherence Score: ', coherence_lda)
```

```
Coherence Score:  0.5082391247655911
```

Find optimal topic number

- We see we have a pretty low coherence score
- We can look at the coherence score for a range of topic numbers and choose the optimal topic number
- We now build a function that will allow us compute `c_v` coherence for various number of topics
- The parameters are:
  - `dictionary` : Gensim dictionary
  - `corpus` : Gensim corpus
  - `texts` : list of input texts
  - `limit` : max num of topics

Convenience function

- Here's a convenience function to run LDA and compute coherence values:

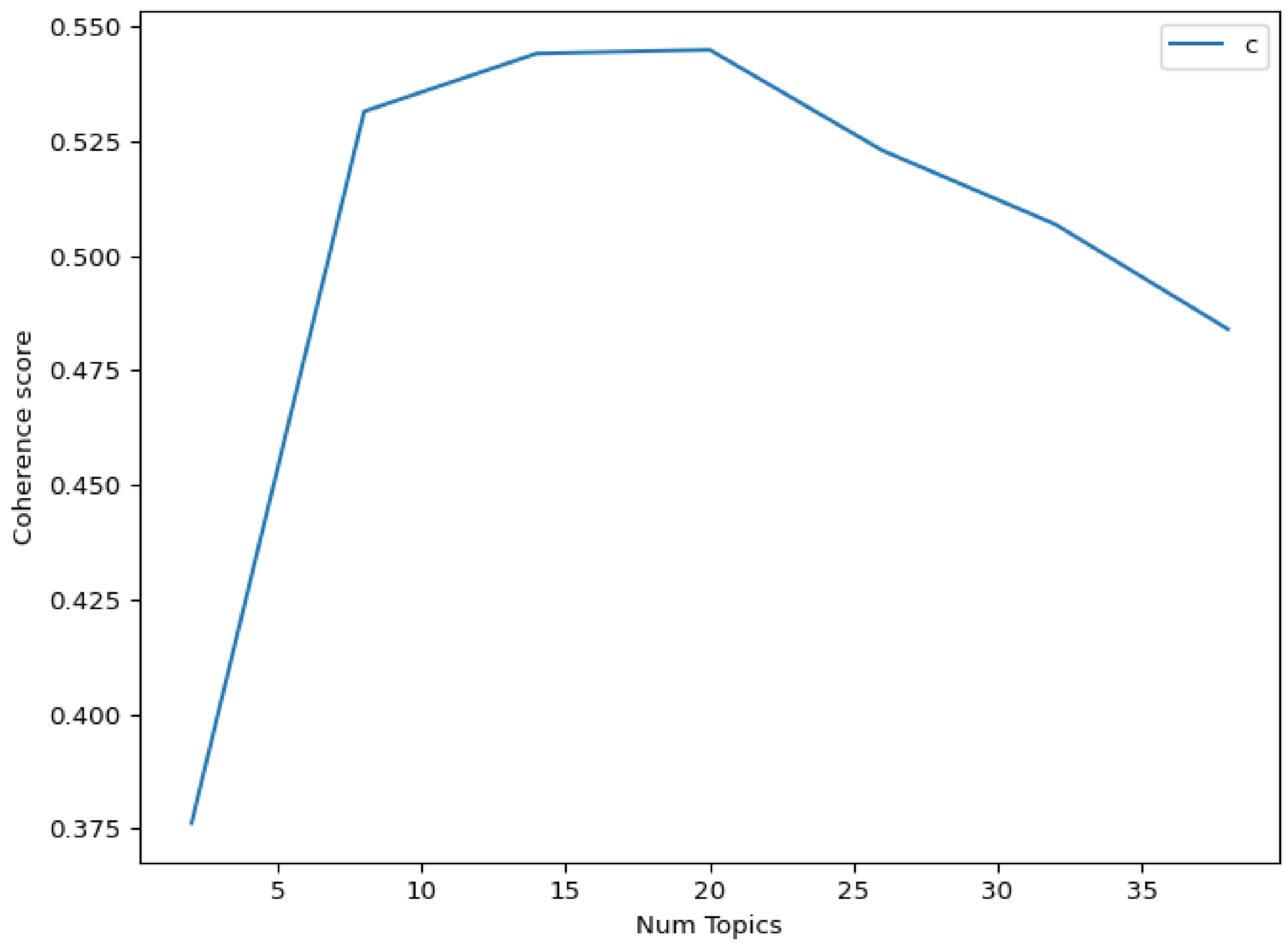
```
def compute_coherence_values(dictionary, corpus, texts, limit, start = 2, step = 3):
    coherence_values = []
    model_list = []
    for num_topics in range(start, limit, step):
        model = gensim.models.LdaMulticore(corpus = corpus,
                                           id2word = dictionary, num_topics = num_topics)
        model_list.append(model)
        coherencemodel = CoherenceModel(model = model, texts = texts,
                                        dictionary = dictionary, coherence = 'c_v')
        coherence_values.append(coherencemodel.get_coherence())

    return model_list, coherence_values
```

- The output of the function is:
  - `model_list` : list of LDA topic models
  - `coherence_values` : coherence values corresponding to the LDA model with respective number of topics

Run compute_coherence_values function

```
np.random.seed(1)
model_list, coherence_values =
compute_coherence_values(dictionary = dictionary,

corpus = corpus_tfidf,

texts = df_clean,

start = 2,

limit = 40,

step = 6)
```

```
# Plot graph of topic list.
# 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()
```



Final thoughts

- **What do you think the optimal number of topics looks like?**
- A couple takeaways about LDA:
    - LDA does better with more text, larger pieces of text / documents
    - Sentiment analysis would do well on a smaller amount of data like what we have here

Knowledge check

Module completion checklist

Objective	Complete
Perform latent dirichlet allocation (LDA) on frequency counts	✔
Evaluate results and choose optimal number of topics	✔

Congratulations on completing this module!

You are now ready to try Tasks 6-9 in the Exercise for this topic