DATA SOCIETY®

Text mining - day 2

"One should look for what is and not what he thinks should be."
-Albert Einstein.

Module completion checklist

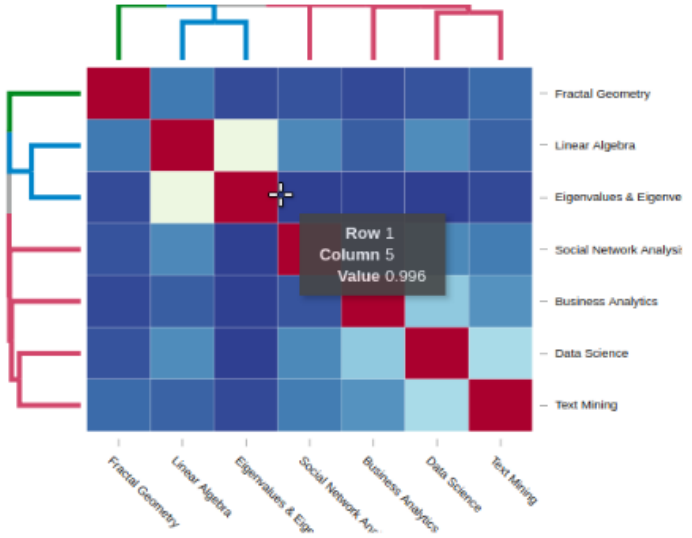
Objective	Complete
Explain use cases for bag-of-words	
Weight text data with term frequency inverse document frequency (TF-IDF)	
Summarize the concept of topic modeling	
Perform latent dirichlet allocation (LDA) on frequency counts	
Evaluate results and choose optimal number of topics	
Visualize results of LDA using interactive LDAvis plot	

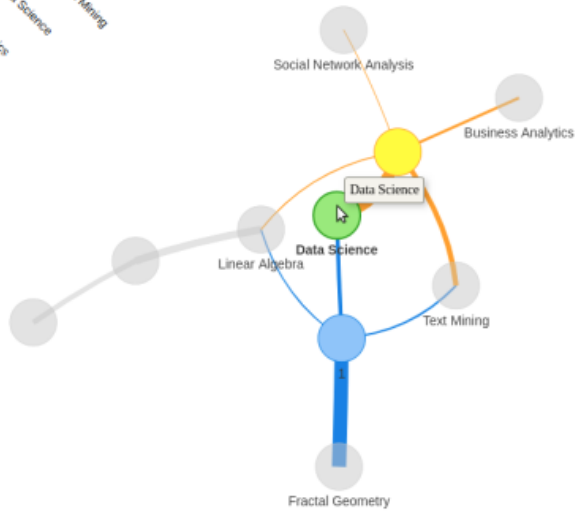
Article snippet analysis

- So far, the steps we have taken are:
  - **Load** the corpus, where each 'document' is actually one article snippet
  - **Clean** the text, removing punctuation, numbers, special characters and stop words
  - *Stem* the words to their root forms
  - **Create** a document-term matrix (DTM) with counts of each word recorded for each document
- Today, we will build the final, optimized matrix - a weighted **T**erm **F**requence - **I**nverse **D**ocument **F**requency (TF-IDF) by
  - **Transforming** the DTM to be a weighted TF-IDF

"Bag-of-words" analysis: use cases

- What can be done with such a seemingly *crude* approach?
- Quite a few things, actually! They include:
  - Topic modeling
  - Word and document similarity query processing
  - Word and document clustering
  - Sentiment analysis
  - Automated document summarization





"Bag-of-words" analysis: article snippets

- Today, we are going to dive deeper into one of these use cases with **topic modeling**
- We are going to use a very popular method called **Latent Dirichlet Allocation (LDA)**
- **This will help us understand broad topics within our corpus of article snippets**

Data exploration

Remember, a data scientist must be able to:

1. **Explore** the data to generate a hypothesis

LDA is a great way to explore and understand your text data, we will learn more today

Module completion checklist

Objective	Complete
Explain use cases for bag-of-words	✔
Weight text data with term frequency inverse document frequency (TF-IDF)	
Summarize the concept of topic modeling	
Perform latent dirichlet allocation (LDA) on frequency counts	
Evaluate results and choose optimal number of topics	
Visualize results of LDA using interactive LDAvis plot	

LDA - unsupervised text analysis

- How does LDA fall into the category of **unsupervised learning**?
  - **Topics** are formed from **unlabeled** text
  - **Clustering documents into "topics" is the basis of this algorithm**
  - **Clustering is one of the best known unsupervised techniques**
- We will learn about LDA in detail later today
- First we need to learn how to **transform our DTM to a TF-IDF weighted matrix**

Datasets for today - class

- Working with **NY Times article snippets**
  - To learn about working with text data using Python
  - To detect patterns in text utilizing text processing and analyzing techniques
  - To apply the above methods to find articles that belong to certain groups (i.e. topics)

Datasets for today - exercises

- Working with **UN agreement titles**
  - To learn about working with text data using Python
  - To detect patterns in text utilizing text processing and analyzing techniques
  - To apply the above methods to find UN agreements that belong to certain groups (i.e. topics)

Knowledge check 1

Module completion checklist

Objective	Complete
Explain use cases for bag-of-words	✔
Weight text data with term frequency inverse document frequency (TF-IDF)	
Summarize the concept of topic modeling	
Perform latent dirichlet allocation (LDA) on frequency counts	
Evaluate results and choose optimal number of topics	
Visualize results of LDA using interactive LDAvis plot	

Directory settings

- Before going into building the TF-IDF, you may want to encode your directory structure into `variables`
- Let the `main_dir` be the variable corresponding to your `af-werx` folder

```
# Set `main_dir` to the location of your `af-werx` folder (for Linux).
main_dir = "/home/[username]/af-werx"
```

```
# Set `main_dir` to the location of your `af-werx` folder (for Mac).
main_dir = '/Users/[username]/af-werx'
```

```
# Set `main_dir` to the location of your `af-werx` folder (for Windows).
main_dir = "C:\\Users\\[username]\\af-werx"
```

```
# Make `data_dir` from the `main_dir` and
# remainder of the path to data directory (for Mac).
data_dir = main_dir + "/data"
```

```
# Make `data_dir` from the `main_dir` and
# remainder of the path to data directory (for Windows).
data_dir = main_dir + "\\data"
```

Loading packages

```
# Helper packages.
import os
import pandas as pd
import numpy as np
import pickle
import matplotlib.pyplot as plt
```

```
# Packages with tools for text processing.
import nltk

# Packages for working with text data.
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
```

```
# Packages for getting data ready and building a LDA model.
import gensim
from gensim import corpora, models
from pprint import pprint
from gensim.models.coherencemodel import CoherenceModel
```

- **Note:** If you have `scikit-learn v0.20` or higher, substitute `sklearn.cross_validation` with `sklearn.model_selection` to import `train_test_split` package!

Gensim package

- Today, we introduce the `gensim` package, which we'll use for LDA
- It is a Python package made for topic modeling, click ***here*** for complete documentation
- We will be using `gensim` to transform our DTM to a TF-IDF matrix as well as to build the LDA model

Working directory

- Set working directory to `data_dir`

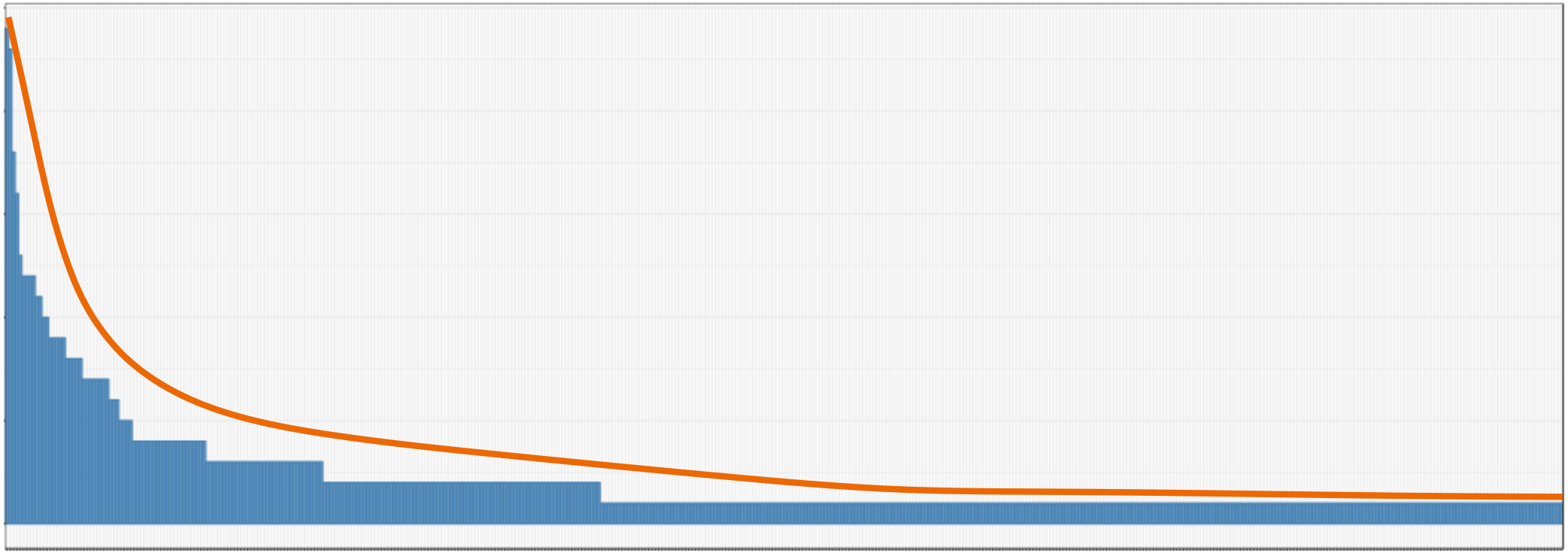
```
# Set working directory.
os.chdir(data_dir)
```

```
# Check working directory.
print(os.getcwd())
```

```
/home/[user-name]/af-werx/data
```

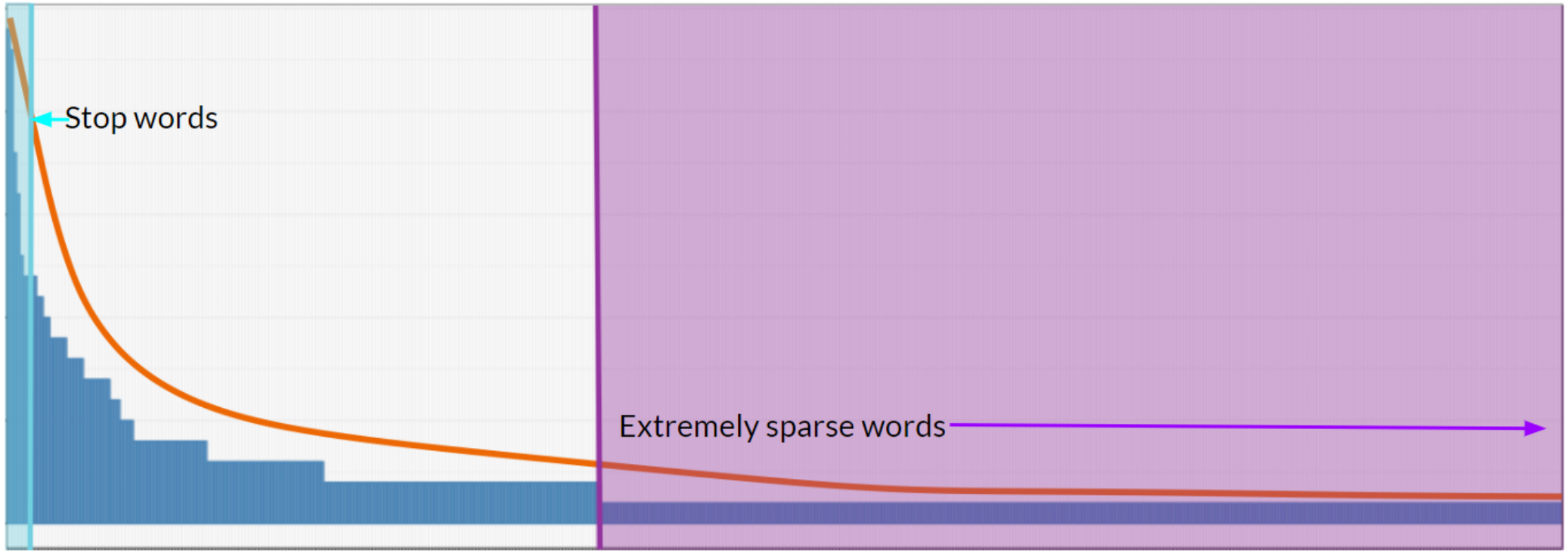
Word distribution in a language and corpus

- The distribution of words in a corpus and in a language is **highly skewed right**, which indicates that few words have very high frequencies and most words have very low counts!
- This phenomenon has implications in many areas of research and applications like information theory, natural language processing, search engines and many more



Word distribution in a language and corpus

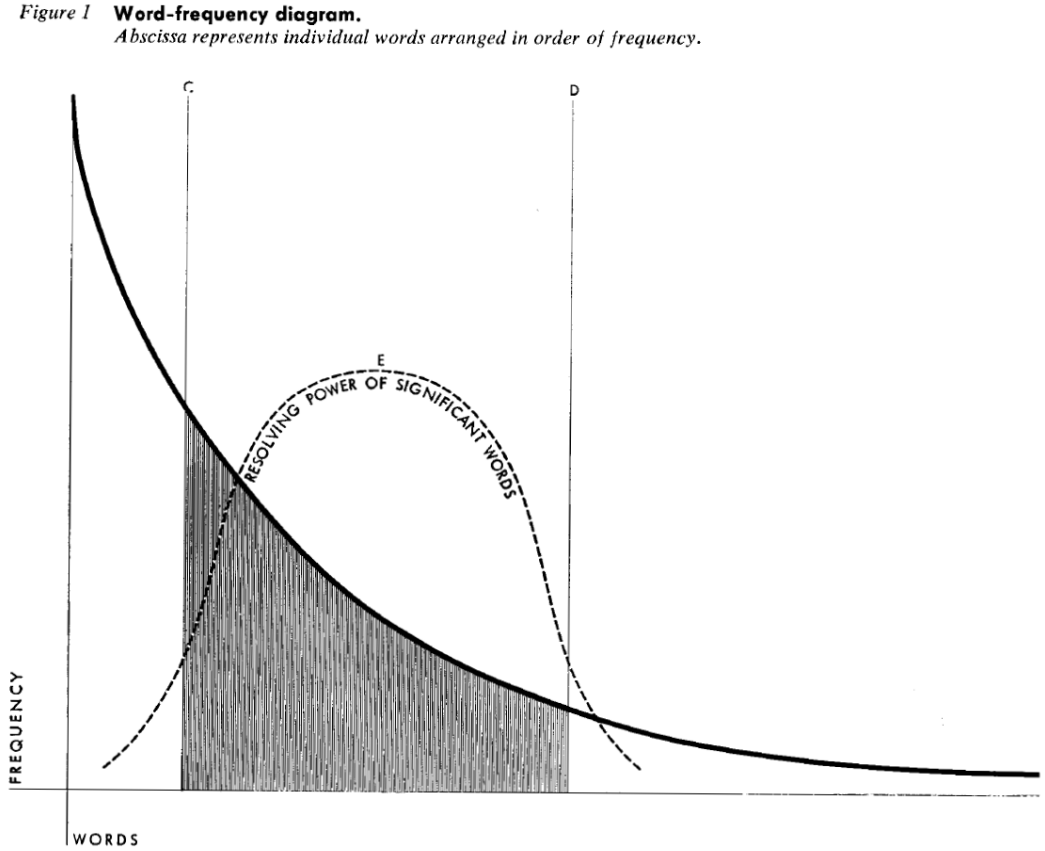
- Words that have extremely high frequencies are considered **stop words**
- Those that have extremely low frequencies are considered **extremely sparse words**



Power of significant words

- By removing *BOTH* ends of the distribution, we **reduce the dimensionality** of our data and **avoid "overfitting"** our text model

- In fact, it has been proven by *H.P. Luhn* (a researcher for IBM) *in 1958* that the **power of significant words is approximately normally distributed** and the top of the bell-shaped curve **coincides with the mid portion of the word frequency distribution**



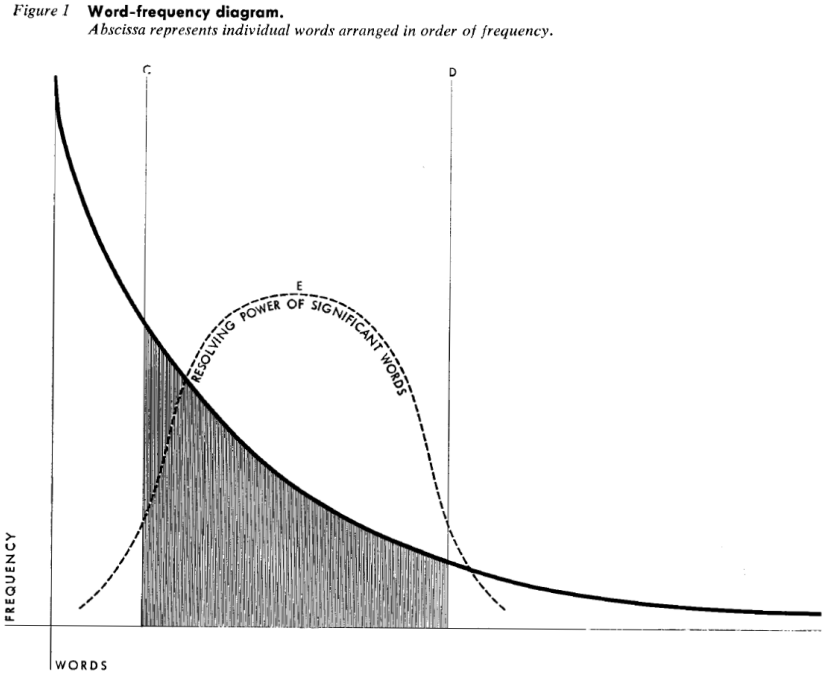
Source: H.P. Luhn, "The Automatic Creation of Literature Abstracts", *Presented at IRE National Convention, New York, March 24, 1958. Published in IBM JOURNAL APRIL 1958*

Why bother?

- **Avoid** having **too many dimensions** (in text analysis words represent variables, i.e. dimensions)
- **Speed up** computations
- **Reduce noise** and **prevent overfitting**

Ways to overcome extreme skewness

- Since the word significance is maximized towards the mid portion of the skewed distribution, we need to "reshape" our data to let the words that are in that mid-portion stand out
- One of the most common ways of achieving this is **weighting** our data, which in statistics is usually known as **data transformation**



Source: H.P. Luhn, "The Automatic Creation of Literature Abstracts",*

**Presented at IRE National Convention, New York, March 24, 1958.*

Published in IBM JOURNAL APRIL 1958

Text Frequency - Inverse Document Frequency

- **TF-IDF** is a famous transformation of text data used to battle the skewness of the word distribution in a corpus and is given by the following formula:

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

- Where:
  - Subscripts w and d stand for $word$ and $document$ respectively
  - F_{wd} is the frequency of a word in a document
  - N_d is a total number of words in a given document
  - M is the total number of documents in a corpus
  - M_w is the number of documents containing the given word in the corpus

TF-IDF effect on DTM

- Normalization of **TF** (i.e. $TF = \frac{F_{wd}}{N_d}$) eliminates the size effect of documents: longer documents with more words no longer overpower smaller documents with fewer words that may contain valuable information
- **IDF** (i.e. $log\frac{M}{M_w}$) helps to adjust for the fact that some words appear more frequently in general, but carry less value in their meaning due to not being very specific to a particular concept that makes a document or a group of documents in a corpus stand out

TF-IDF effect on DTM (cont'd)

- Consider the following DTM:

DTM with raw counts

	analyze	data	fractal
Doc A	1	2	0
Doc B	0	0	1
Doc C	0	4	3

DTM with TF-IDF weights

	analyze	data	fractal
Doc A	0.528	0.390	0
Doc B	0	0	0.585
Doc C	0	0.334	0.251

- The word `data` appears in `DocA` and `DocC`, and although it appeared twice as often in `DocC`, its weight has adjusted for `DocA` to a very similar one as in `DocC`, making it of similar "value"

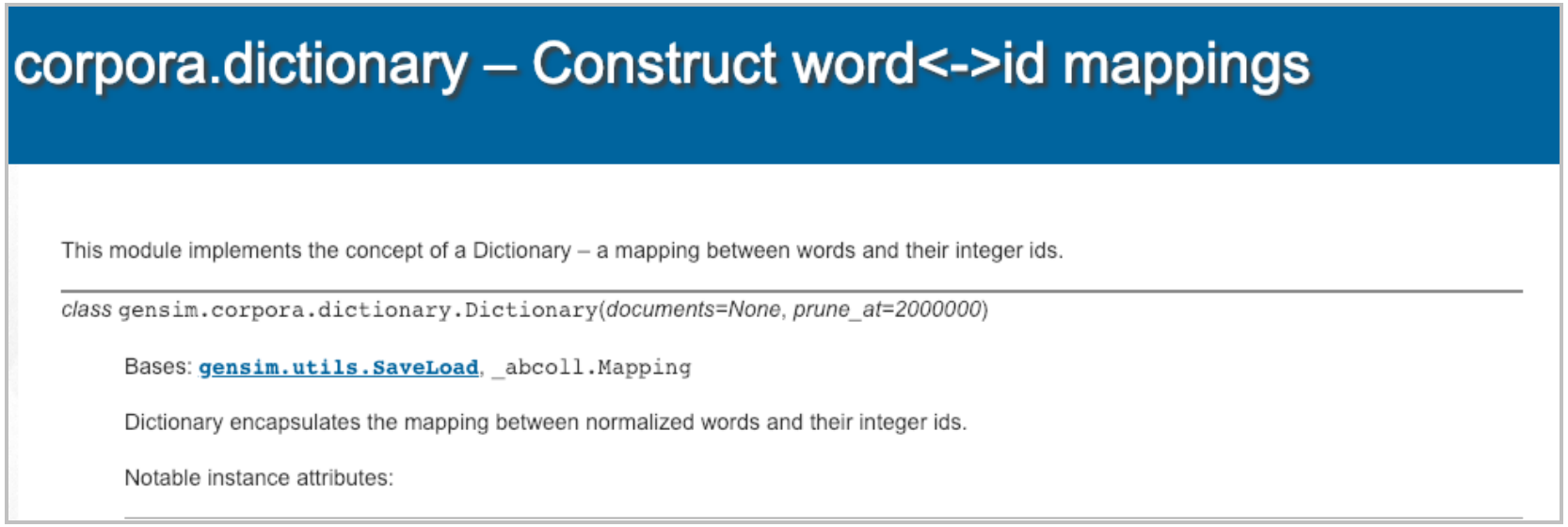
Load objects from last class

- Remember, we used `pickle` to save our objects from last class
- As a refresher, `pickle` saves objects by "flattening" them:
  - **pickle/saving: a Python object is converted into a byte stream**
  - **unpickle/loading: the inverse operation where a byte stream is converted back into an object**
- We saved objects last class, let's **unpickle** what we need today, the `NYT_clean` object:

```
processed_docs = pickle.load(open("NYT_clean.sav","rb"))  #<- the processed NYT snippets
```

Create a dictionary of counts

- Remember last class when we created a convenience function to look at counts?
- **Today, we will create a dictionary of counts using a `gensim` function `gensim.corpora.Dictionary`**
- **This will give us a dictionary from `processed_docs` that contains the number of times a given word appears within the entire corpus**



corpora.dictionary – Construct word<->id mappings

This module implements the concept of a Dictionary – a mapping between words and their integer ids.

class `gensim.corpora.dictionary.Dictionary`(*documents=None*, *prune_at=2000000*)

Bases: `gensim.utils.SaveLoad`, `_abcoll.Mapping`

Dictionary encapsulates the mapping between normalized words and their integer ids.

Notable instance attributes:

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(processed_docs)

# 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 africa
1 attack
2 back
3 batsmen
4 claw
5 find
6 handl
7 like
8 live
9 must
10 newland
```

Document to bag-of-words dictionary

- Now we will use `gensim` libraries `doc2bow` to transform each document to a dictionary
- Each document will become a dictionary that has how many words and how many 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 processed_docs object.
bow_corpus = [dictionary.doc2bow(doc) for doc in processed_docs]

# Let's look at the 1st document.
print(bow_corpus[0])
```

```
[(0, 1), (1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (6, 1), (7, 1), (8, 1), (9, 1), (10, 1), (11, 1), (12,
1), (13, 1), (14, 1), (15, 1), (16, 1), (17, 1), (18, 1), (19, 1), (20, 1), (21, 2), (22, 1)]
```

Document to bag-of-words

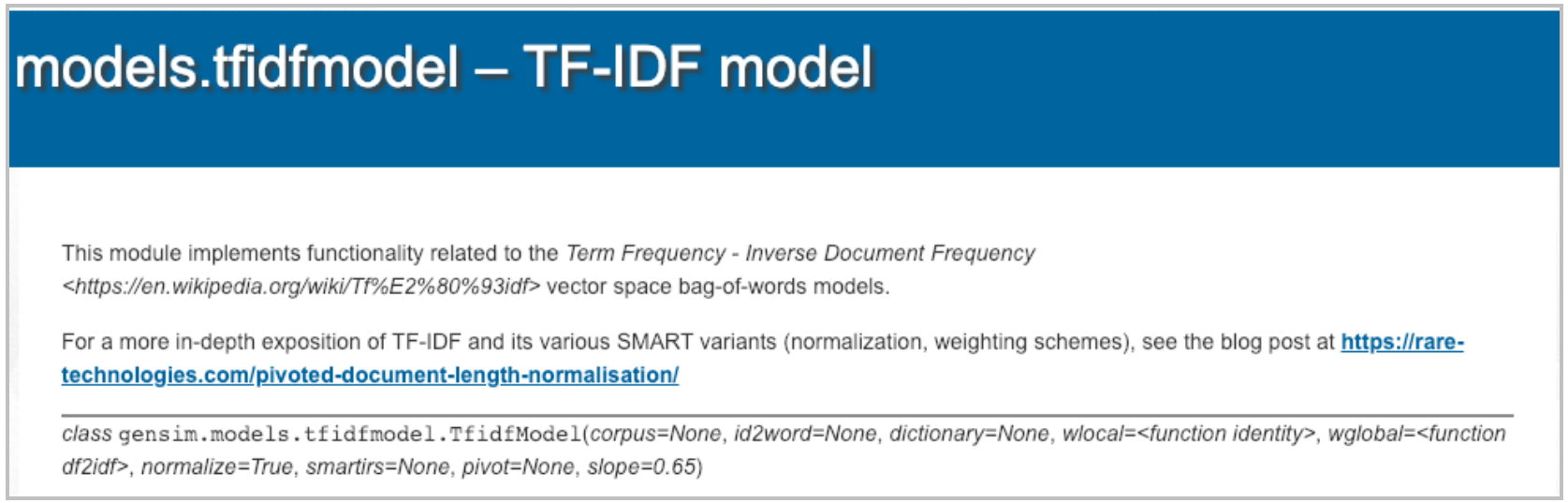
- Let's preview bag-of-words for the first document

```
# 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 ("africa") appears 1 time.
Word 1 ("attack") appears 1 time.
Word 2 ("back") appears 1 time.
Word 3 ("batsmen") appears 1 time.
Word 4 ("claw") appears 1 time.
Word 5 ("find") appears 1 time.
Word 6 ("handl") appears 1 time.
Word 7 ("like") appears 1 time.
Word 8 ("live") appears 1 time.
Word 9 ("must") appears 1 time.
Word 10 ("newland") appears 1 time.
Word 11 ("pace") appears 1 time.
Word 12 ("pakistan") appears 1 time.
Word 13 ("potent") appears 1 time.
Word 14 ("second") appears 1 time.
Word 15 ("seri") appears 1 time.
Word 16 ("south") appears 1 time.
```

Transform counts with TfidfModel

- To transform a Document-Term matrix with `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

- 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, 0.20123812213894005),
 (1, 0.16169735155283377),
 (2, 0.1814677368458869),
 (3, 0.23503581903178308),
 (4, 0.23503581903178308),
 (5, 0.19035769396664654),
 (6, 0.19035769396664654),
 (7, 0.13068668575528808),
 (8, 0.20123812213894005),
 (9, 0.23503581903178308),
 (10, 0.23503581903178308),
 (11, 0.21526543373872994),
 (12, 0.20123812213894005),
 (13, 0.23503581903178308),
 (14, 0.20123812213894005),
 (15, 0.20123812213894005),
 (16, 0.1814677368458869),
 (17, 0.16744042524609698),
 (18, 0.20123812213894005),
```

"Bag-of-words" analysis: key elements

What we need	What we have learned
A corpus of documents cleaned and processed in a certain way • All words a converted to lower case • All punctuation, numbers and special characters are removed • Stopwords are removed • Words are stemmed to their root form	✔
A Document-Term Matrix (DTM): with counts of each word recorded for each document	✔
A transformed representation of a Document-Term Matrix (i.e. weighted with TF-IDF weights)	✔

Knowledge check 2

Exercise 1

Module completion checklist

Objective	Complete
Explain use cases for bag-of-words	✔
Weight text data with term frequency inverse document frequency (TF-IDF)	✔
Summarize the concept of topic modeling	
Perform latent dirichlet allocation (LDA) on frequency counts	
Evaluate results and choose optimal number of topics	
Visualize results of LDA using interactive LDAvis plot	

Article snippet - topic modeling

- So far, the steps we have taken are:
  - **Load** the corpus, where each 'document' is actually one chat message
  - **Clean** the text, removing punctuation, numbers, special characters and stop words
  - *Stem* the words to their root forms
  - **Create** a document-term matrix (DTM) with counts of each word recorded for each document
  - **Transform** the DTM to be a weighted term frequency - inverse document frequency matrix

TF-IDF weighted corpus to LDA

- We have our final transformation of our processed documents, `corpus_tfidf`
- The next step is to find out what topics seem to stand out within these 248 articles
  - **Are there groups of articles that all fall under certain topics?**
  - **How can we subset these articles into larger groups than 248 separate documents?**
- We can find an answer to both of these questions by running a LDA model on the corpus

An introduction to LDA

- **Latent Dirichlet Allocation (LDA) is a popular algorithm for topic modeling because it allows us to:**
  - Reduce dimensionality amongst large bodies of documents
  - Apply other machine learning algorithms to the reduced corpus
  - Uncover themes and patterns within your data
- **Today, we are going to go over the algorithm in three steps:**

1. Tell the algorithm how many topics you think there are
2. Algorithm will assign every word to a temporary topic
3. Algorithm will check and update topic assignments

- Here is the ***original paper*** written on the algorithm by David M. Blei, Andrew Y. Ng and Michael I. Jordan

LDA on a simple corpus

- Let's use a **simple corpus** as an example
- It consists of **three documents** which are actually just three sentences
  - I **eat chicken** and **vegetables**
  - **Chicken** live on **farms**
  - My **dog** loves to eat my **leftover chicken**

LDA on a simple corpus (cont'd)

- **What do you think LDA will do with these documents?**
- **LDA could:**
  - Classify the **bold** words under the topic A, which we then might inspect and label **food**
  - Classify the **blue** words under the topic B, which we then might inspect and label **animals and livestock**
- **LDA is actually defining each of these documents as a bag-of-words and you then label the topics as you see fit**

Defining LDA at a document level

- Remember how we applied the TF-IDF transformation to each document?
- This example illustrates why there is a benefit of LDA defining topics on a word level
- **We can infer the content spread of each sentence by a word count:**
  - **Sentence 1:** 100% topic A
  - **Sentence 2:** 100% topic B
  - *Sentence 3*: 66% topic B, 33% topic A
- **We can derive the proportions that each word constitutes in given topics**
  - **Topic A** might comprise words in the following proportions: 40% chicken, 20% leftover, 20% vegetables, 20% eat
  - **Topic B** might comprise words in the following proportions: 33% chicken, 33% dogs, 33% farm

LDA in three steps

Let's go back to the three steps of LDA

1. Tell the algorithm how many topics you think there are
2. Algorithm will assign every word to a temporary topic
3. Algorithm will check and update topic assignments

Now, instead of three sentences, let's imagine we have two documents with the following words:

Document 1		Document 2	
	dog		dog
	dog		dog
	chicken		vegetables
	chicken		leftovers
	farm		leftovers

Step 1: number of topics

- **The first step is to tell the algorithm how many topics you think there are**, this is usually based on:
  - previous analysis
  - informed decision by a subject matter expert
  - random guess
- **In trying different estimates, you may pick the one that generates topics to your desired level of interpretability**
- In our example, we can probably guess the number of topics by eyeballing the documents, since they are tiny
- **We will guess that there are two topics**

Step 2: algorithm assigns temporary topic

- **The second step is when the algorithm assigns every word in each document to a temporary topic**
  - Topic assignments are temporary, they will be updated in **Step 3**
  - Temporary topics are assigned according to a ***Dirichlet distribution***
  - If a certain word appears twice, it may be assigned to two different topics
- **Let's look at how topics have been assigned in our small example, remember we are dealing with topic A and topic B**

Document 1		Document 2	
A	dog	**?**	dog
A	dog	A	dog
A	chicken	A	vegetables
A	chicken	B	leftovers
A	farm	B	leftovers

Step 3: checking and updating topic assignments

- **Step 3 is the iterative step of the algorithm, where topics are checked and updated as the algorithm loops through each word in every document**
- The algorithm is looking at two main criteria:
  - **a)** How prevalent is the word across topics?
  - **b)** How prevalent are the topics in the document?
- **Remember the question marked item in *Document 2* from step 2?**
- We will now see how the algorithm iterates and updates the topic for the **?** from step 2, and the assignment for **dog** in document 2

Step 3a: word across topics

How prevalent is the word across topics?

- **Dog seems to be prevalent within topic A and not seen in topic B**
- A **dog** word **picked randomly would more likely be about topic A**

Document 1		Document 2	
A	**dog**	**?**	**dog**
A	**dog**	**A**	**dog**
A	chicken	A	vegetables
A	chicken	B	leftovers
A	farm	B	leftovers

Step 3b: topics in the document

How prevalent are the topics in the document?

- Now we look within document 2
- We see that *the words are split 50/50 between* **topic A** and **topic B**
- Therefore, when inspecting the document, **dog** has a 50/50 chance of either topic

Document 1		Document 2	
A	dog	**?**	**dog**
A	dog	**A**	**dog**
A	chicken	**A**	**vegetables**
A	chicken	**B**	**leftovers**
A	farm	**B**	**leftovers**

Which topic then?

- We weigh both criteria and see that **dog** from document 2 seems to fit more within **topic A**
- **What could topic A be about?**

- **And topic B?**

Step 3c: assign topic

- **The process that we used to allocate dog to topic A is repeated on each word in each document, and this cycle occurs multiple times**
- *This iterative updating is the **key** feature of LDA that allows us to arrive finally at **coherent topics***

Document 1		Document 2	
A	dog	**A**	**dog**
A	dog	**A**	**dog**
A	chicken	**A**	**vegetables**
A	chicken	**B**	**leftovers**
A	farm	**B**	**leftovers**

Module completion checklist

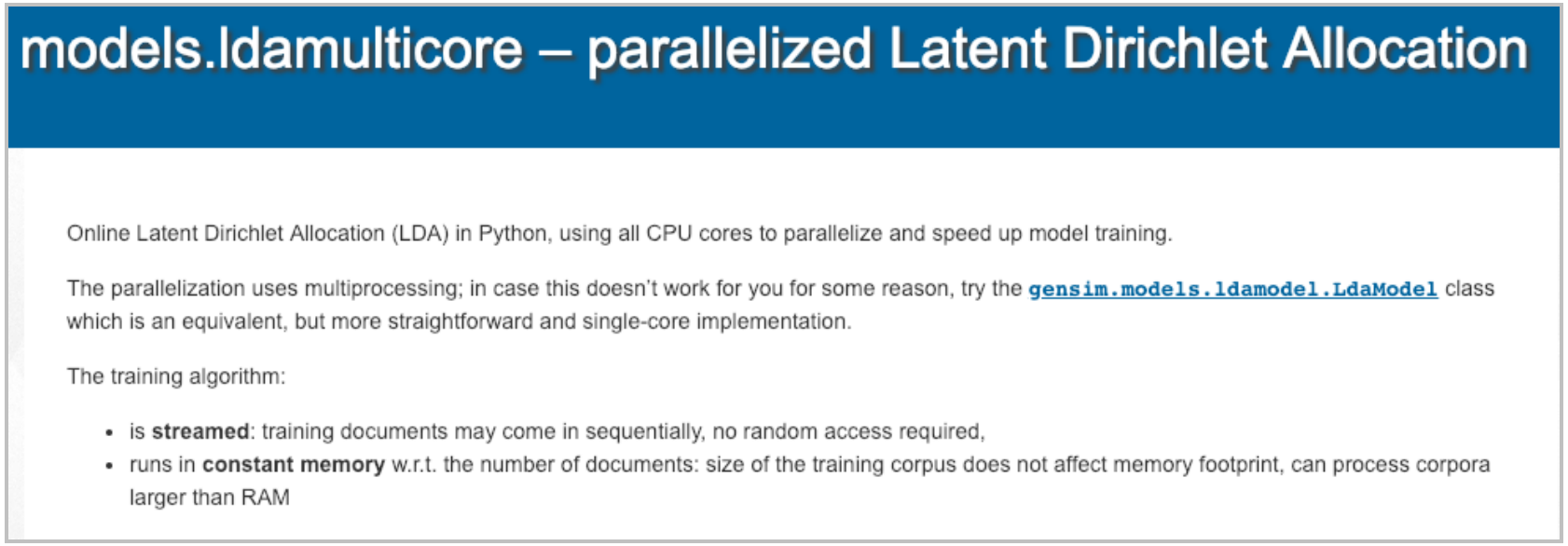
Objective	Complete
Explain use cases for bag-of-words	✔
Weight text data with term frequency inverse document frequency (TF-IDF)	✔
Summarize the concept of topic modeling	✔
Perform latent dirichlet allocation (LDA) on frequency counts	
Evaluate results	

LDA on article snippets

- **Let's quickly review what has already been done to the snippets**
  - **Load** the corpus, where each 'document' is actually one chat message
  - **Clean** the text, removing punctuation, numbers, special characters and stop words
  - *Stem* the words to their root forms
  - **Create** a document-term matrix (DTM) with counts of each word recorded for each document
  - **Transform** the DTM to be a weighted term frequency - inverse document frequency matrix
- Now we understand the basic idea of LDA and how it works
- Let's apply it to our corpus of **NYT article snippets**

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 2 for now, as most machines will have 2 cores
- The algorithm we just walked through with the **two documents** will now be applied to **248 documents**

LdaMulticore

- Before running the model, let's make sure we understand the main parameters of the model:

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

- `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 topic

Running LdaMulticore

- Ok, 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)
```

```
LdaModel(num_terms=1925, num_topics=5, decay=0.5, chunksize=2000)
```

LDA output

- Let's look at the 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: 0.003*"suspect" + 0.002*"year" + 0.002*"make" + 0.002*"local" + 0.002*"first" +
0.002*"leagu" + 0.002*"head" + 0.002*"time" + 0.002*"presid" + 0.002*"new"
Topic: 1 Word: 0.003*"said" + 0.002*"best" + 0.002*"tuesday" + 0.002*"would" + 0.002*"brief" +
0.002*"argument" + 0.002*"reach" + 0.002*"never" + 0.002*"hous" + 0.002*"secretari"
Topic: 2 Word: 0.002*"new" + 0.002*"week" + 0.002*"time" + 0.002*"york" + 0.002*"profession" +
0.002*"murder" + 0.002*"offici" + 0.002*"championship" + 0.002*"appear" + 0.002*"final"
Topic: 3 Word: 0.003*"local" + 0.002*"time" + 0.002*"govern" + 0.002*"latest" + 0.002*"new" +
0.002*"say" + 0.002*"world" + 0.002*"eve" + 0.002*"show" + 0.002*"order"
Topic: 4 Word: 0.003*"presid" + 0.002*"say" + 0.002*"donald" + 0.002*"trump" + 0.002*"border" +
0.002*"kill" + 0.002*"said" + 0.002*"secur" + 0.002*"even" + 0.002*"hous"
```

- 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 in our next class!

Classify our documents within topics

- Let's see how we would classify our processed_docs as one of the five topics:

```
# Now, we can check where each of our documents would be classified.
# Let's look at our first document as an example:

print(processed_docs[0])
```

```
['pakistan', 'struggl', 'batsmen', 'must', 'find', 'way', 'handl', 'south', 'africa', 'potent', 'pace',
'attack', 'claw', 'way', 'back', 'seri', 'second', 'test', 'start', 'like', 'live', 'newland',
'wicket', 'thursday']
```

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

```

Score: 0.967782735824585
Topic: 0.002*"new" + 0.002*"week" + 0.002*"time" + 0.002*"york" + 0.002*"profession" + 0.002*"murder" +
0.002*"offici" + 0.002*"championship" + 0.002*"appear" + 0.002*"final"
```

Knowledge check 3

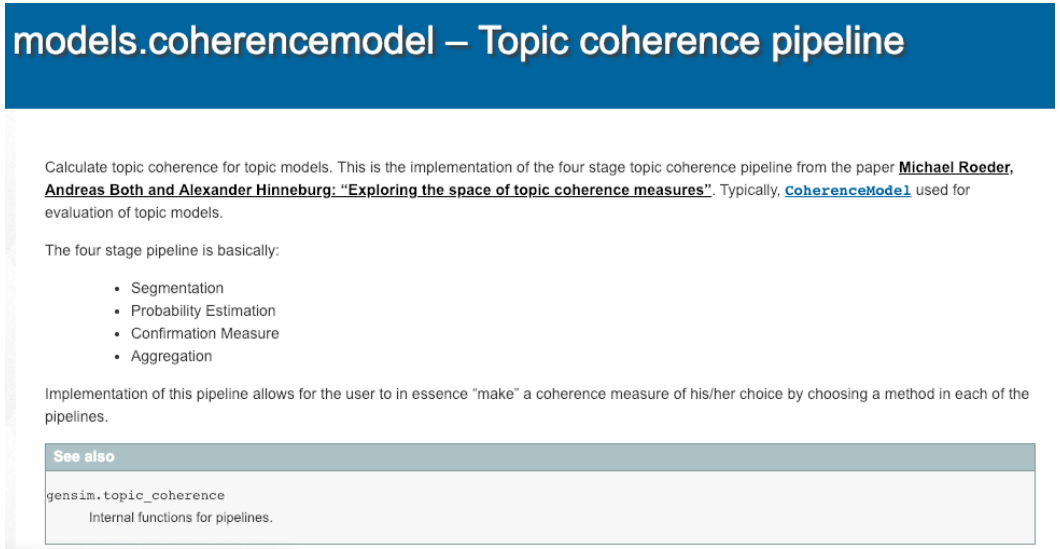
Exercise 2

Module completion checklist

Objective	Complete
Explain use cases for bag-of-words	✔
Weight text data with term frequency inverse document frequency (TF-IDF)	✔
Summarize the concept of topic modeling	✔
Perform latent dirichlet allocation (LDA) on frequency counts	✔
Evaluate results and choose optimal number of topics	
Visualize results of LDA using interactive LDAvis plot	

LDA - evaluate results

- How do we evaluate our LDA model?
- The measure we will focus on today 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`

Topic coherence quick overview

- We will now calculate topic coherence on our `lda_model_tfidf`
- The parameters we need are:
  - original doc, `processed_docs`
  - 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 = processed_docs, dictionary =
dictionary, coherence = 'c_v')
coherence_lda = coherence_model_lda.get_coherence()
print('\nCoherence Score: ', coherence_lda)
```

```
Coherence Score:  0.4937424088619259
```

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
- Let's 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

```
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
```

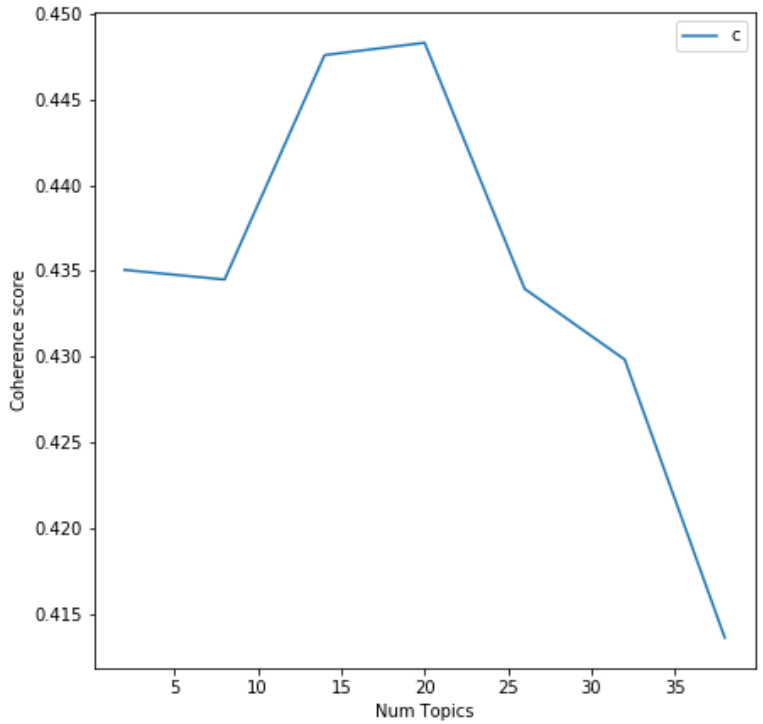
The output of the function is:

```
- `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

```
model_list, coherence_values =
compute_coherence_values(dictionary =
dictionary, corpus = bow_corpus, texts =
processed_docs, start = 2, limit = 40, step = 6)
```

```
# Plot graph of topic list.
# Show graph.
import matplotlib.pyplot as plt
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()
```



Visualize topics generated with LDA

- We have performed LDA on the NY Times article snippets and assessed the numerical metrics of our model's performance
- Now is the time to look at the overall picture and how all of those metrics and numbers fit together
- The best way to do it is to visualize the LDA
- We will be using `pyLDAvis` package that is a Python wrapper around a very popular R package called `LDAvis`
- If you are interested to learn more about the inner workings and detailed explanation of `LDAvis`, you can find the original publication ***here***

Visualize topics generated with LDA: pyLDAvis

- Since there are a few different libraries and packages in Python that create LDA models, `pyLDAvis` has a few methods that handle data from the most popular ones
- We created our LDA model using `gensim`, which integrates easily with `pyLDAvis` through a module `pyLDAvis.gensim`

- The method that generates a visualization object is called `pyLDAvis.gensim.prepare()` and it takes the following `gensim` objects as arguments:
  - LDA model object
  - the corpus object
  - the dictionary
- We have created all three in the previous module and imported them already as the following variables:
  - `lda_model_tfidf`
  - `corpus_tfidf`
  - `dictionary`

Visualize topics generated with LDA

- Let's prepare the visualization object for plotting

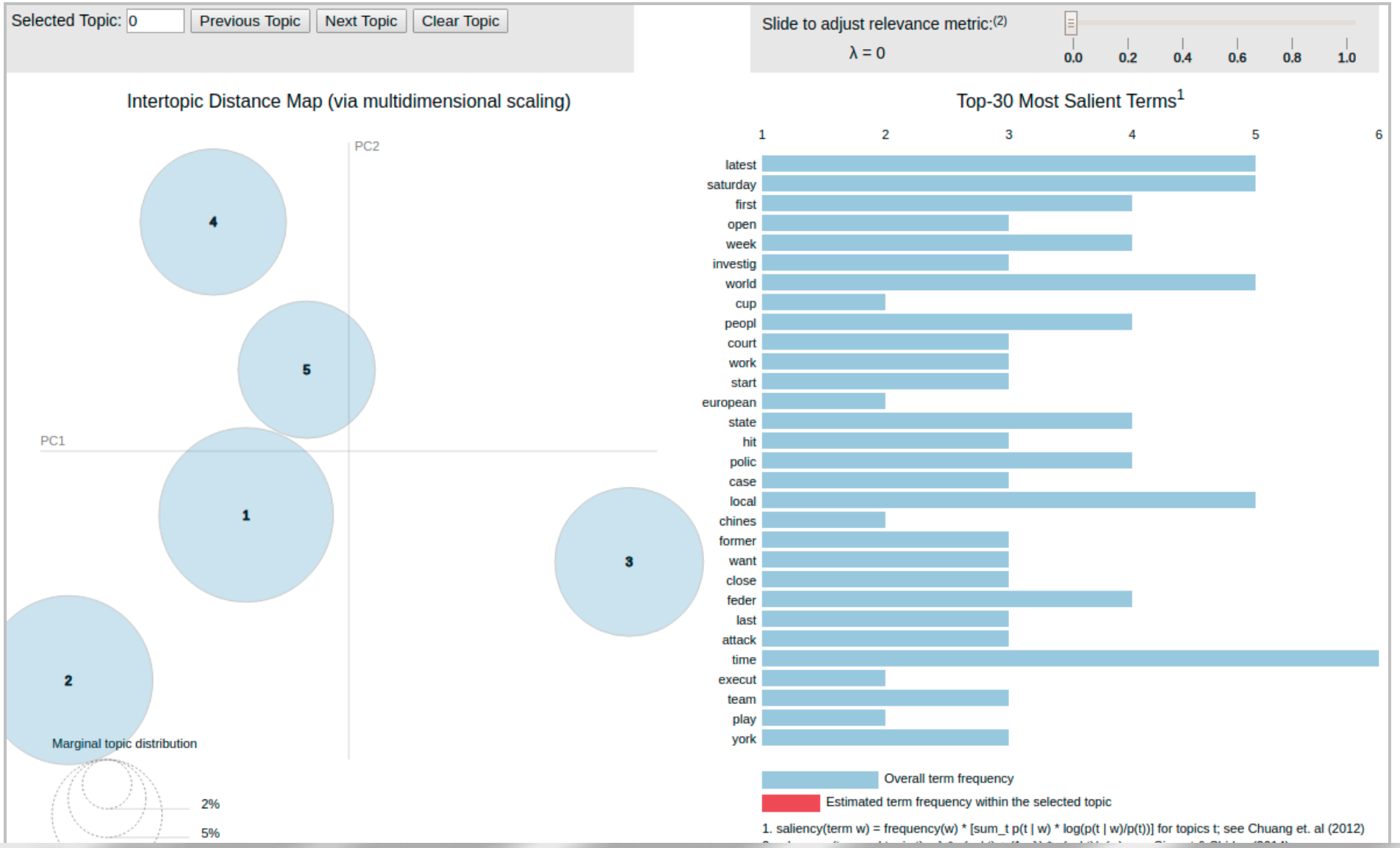
```
# Prepare LDA vis object by providing:
vis = pyLDAvis.gensim.prepare(lda_model_tfidf,    #<- model object
                              corpus_tfidf,#<- corpus object
                              dictionary)  #<- dictionary object
```

- To display the results in `Jupyter`, you just need to use `pyLDAvis.display()` function

```
# The function takes `vis` object that we prepared above as the main argument.
pyLDAvis.display(vis)
```

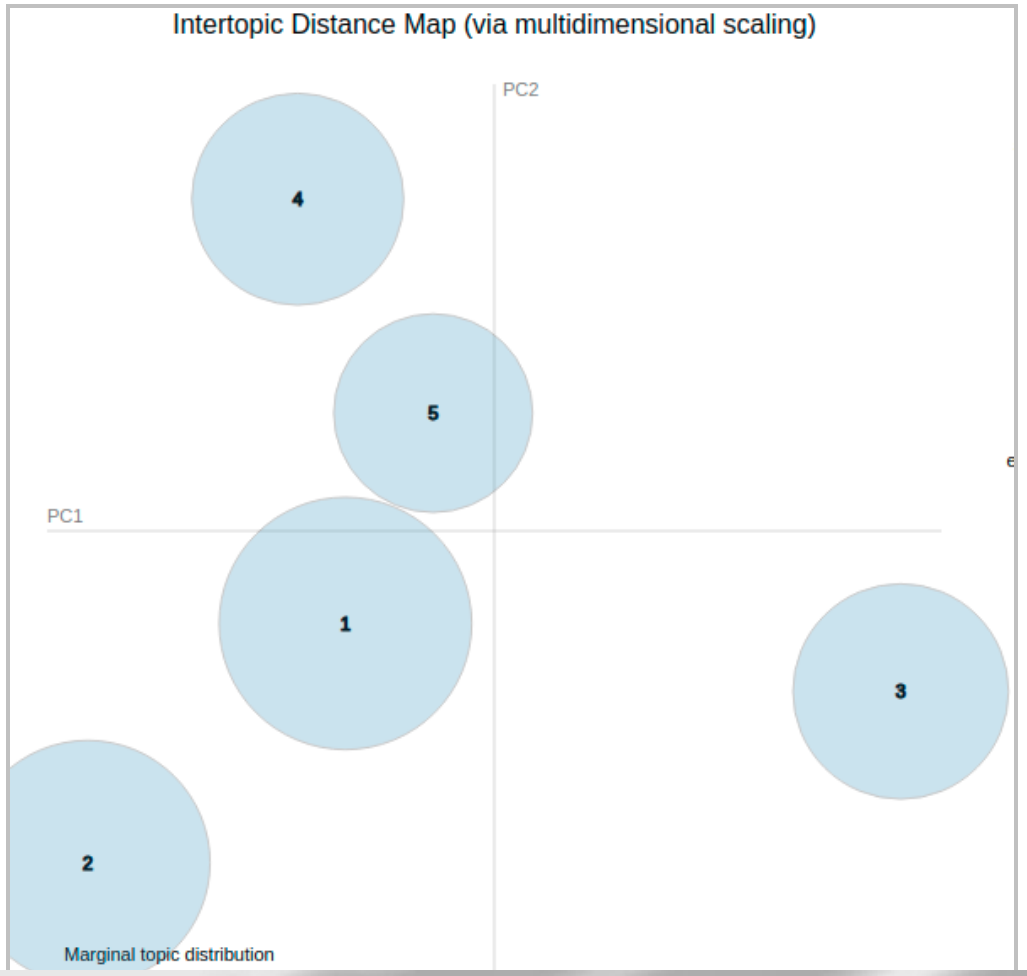
- Give the chart a moment to appear and render

Visualize topics generated with LDA (cont'd)



LDA visualization: topic distribution

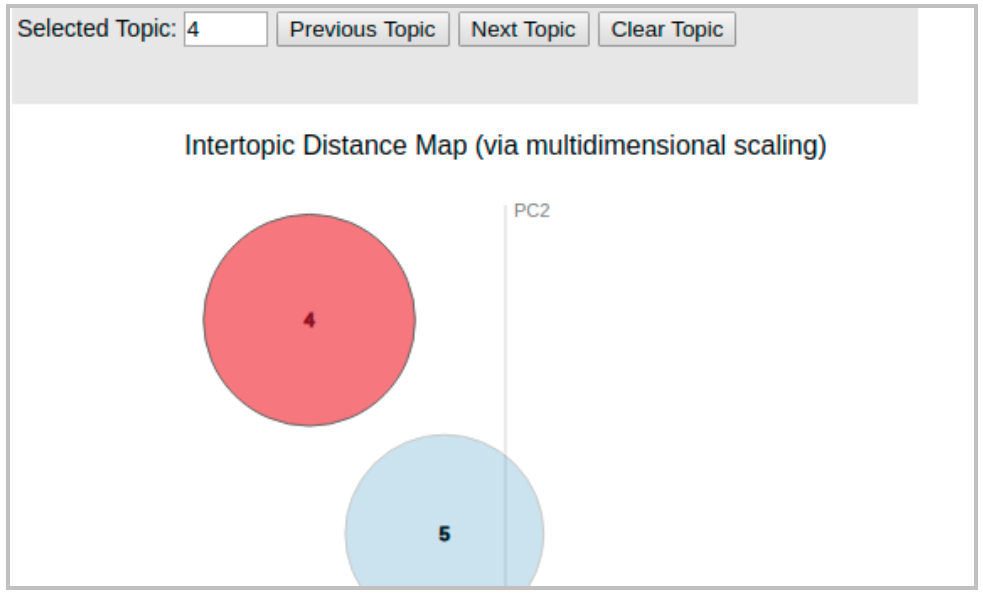
- The left-hand side shows the topics represented by the circles



- Circle **locations** are related to the topic position with respect to one another
  - topics closer to each other in space are closer to each other in meaning
  - topics farther away are more dissimilar in meaning
- Circle **size** is related to the number of documents that contain the topic
  - topics found in more documents are bigger circles
  - topics found in fewer documents are smaller circles
- *By looking at this part of the plot, what can you tell about the topics in our NYT snippets corpus?*

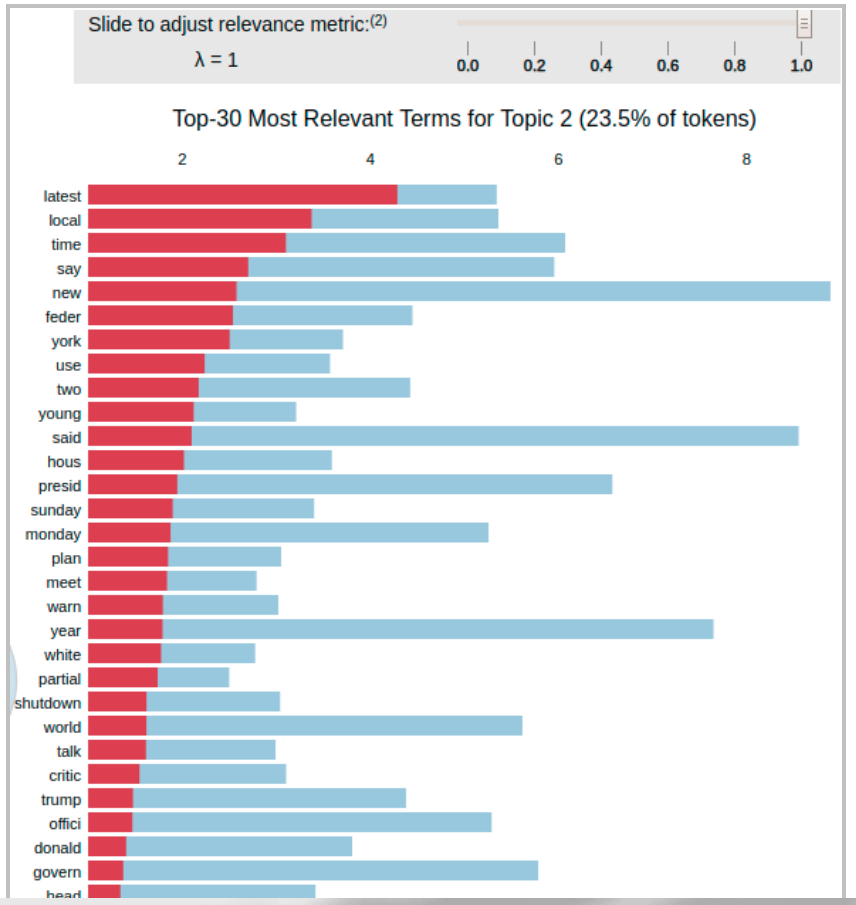
LDA visualization: select topic

- To select a particular topic, you can either
  - click on the respective circle, or
  - enter the topic number in the window in the top left corner



LDA visualization: relevant terms

- The right-hand side shows the most relevant terms in each topic, once the topic is selected



- The **blue bars** represent the overall term frequency in corpus
- The **maroon bars** represent term frequency in selected topic
- The slider at the top represents the value of λ - a relevance metric
  - Default is `1` which means that the term's place in the relevance ranking below is solely based on its frequency within a selected topic
  - When set to `0`, the ranking re-arranges itself to be based on the term's frequency within topic with respect to its frequency within corpus
  - When set to be between `0` and `1`, the ranking will depend on both of the above

LDA visualization: terms across topics

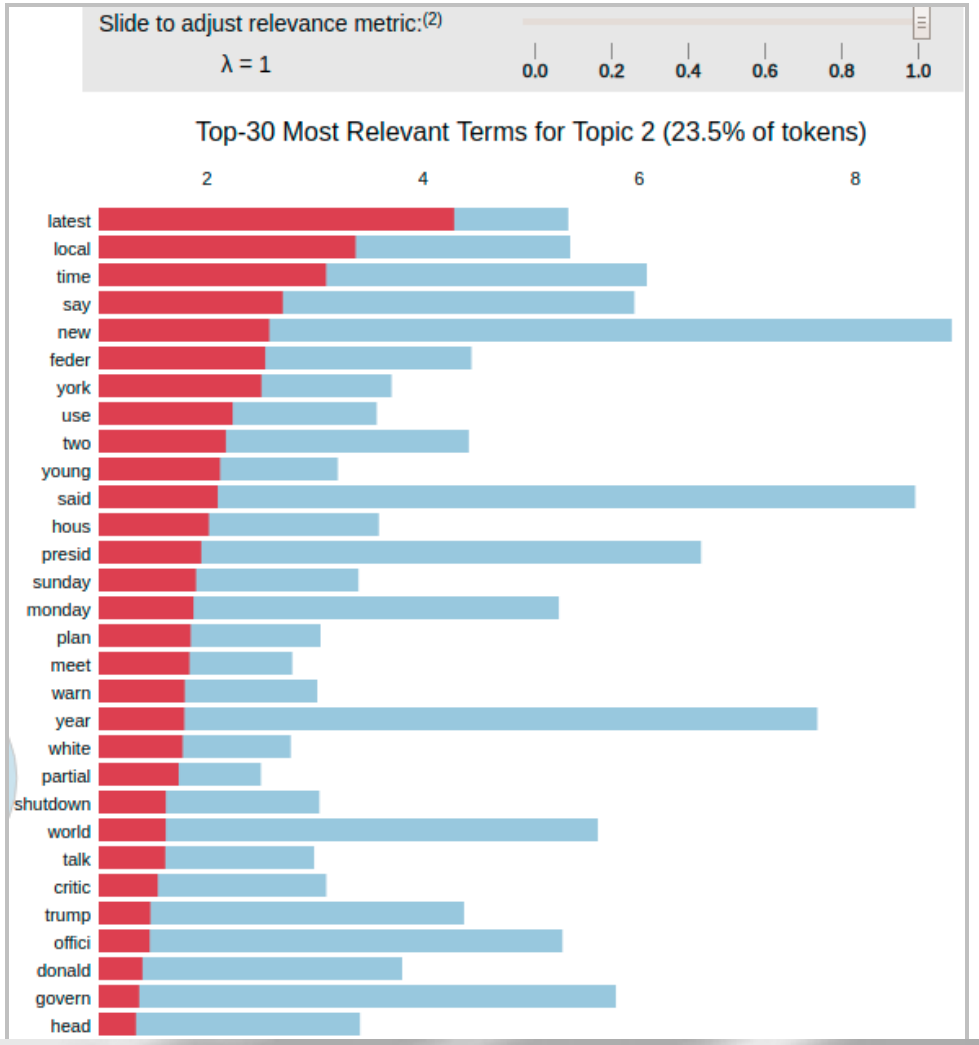
- When you hover over a term, you will see the term's distribution across topics
- For instance, `team` appears in topics `3` and `5`, but it is more prominent in topic `5`

LDA visualization: how to determine topics

- The LDA algorithm does not give us explicit names of topics
- It produces the probability scores associated with *topic distribution within each document* and *term distribution across topics*
- We can **assess** those probabilities and LDA results by interacting with LDA visualization **and inferring the topics**
- LDA visualization allows us to explore and tweak the parameters to **find terms most relevant for each topic**
- These terms will allow us to **infer** the actual topic and to name it
- This process requires the subject matter expertise and some common sense, and the naming conventions of topics are subjective

LDA visualization: name topic 2

- Let's take a look at the most relevant terms in topic 2:



- Take a look at the words with $\lambda = 1$, what are the top 10?
- Now adjust the slider to $\lambda = 0$, what are the top 10?
- Now adjust the slider to $\lambda = 0.2$, what are the top 10?
- By looking at all relevant terms in this chart, what do you think this topic is about?

LDA visualization: name other topics

- Now do the same for all other topics
- Share your thoughts
- Which topics were the hardest to label?
- Why do you think that's the case?
- What can be done to improve the model overall?

Save LDA visualization to HTML file

- To keep and distribute the visualization as fully-interactive `HTML` file, we can simply use `pyLDAvis.save_html()` method
- We need to provide it with 2 arguments:
  - visualization object we prepared earlier (i.e. `vis`)
  - file path with name where we would like to save it

```
# Save the plot as a self-contained HTML file.
pyLDAvis.save_html(vis, plot_dir+"/NYT_LDAvis.html")
```

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**
- Next class, we will discuss in detail more ways to evaluate the actual topics

Knowledge check 4

Exercise 3

Module completion checklist

Objective	Complete
Explain use cases for bag-of-words	✔
Weight text data with term frequency inverse document frequency (TF-IDF)	✔
Summarize the concept of topic modeling	✔
Perform latent dirichlet allocation (LDA) on frequency counts	✔
Evaluate results and choose optimal number of topics	✔
Visualize results of LDA using interactive LDAvis plot	✔

Workshop!

- Workshops are to be completed in the afternoon either with a dataset for a capstone project or with another dataset of your choosing
- Make sure to annotate and comment your code so that it is easy for others to understand what you are doing
- This is an exploratory exercise to get you comfortable with the content we discussed today
- Today you will:
  - Load the text corpus from last class
  - Convert the corpus to TF-IDF vector
  - Perform LDA on the corpus and compute coherence score

This completes our module

Congratulations!