DATA SOCIETY®

Introduction to text mining - part 3

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

Welcome back

Before we get started, here's a scenario to get you thinking.

- Suppose you've been tasked with analyzing about **3000 responses** to an **open-ended survey question** (i.e. "What else could we improve?")
  - What **steps** would you need to take to **manually code** this data?
  - How could you **determine** the type and number of **categories**?
  - What **problems** could arise in attempting to draw **conclusions**?

Recap and next steps

The topics covered in the last module:

- Visualizing text data
  - n-grams and a bit about their probabilities vs. single words (unigrams)
  - Creating the term frequency-inverse document frequency (TF-IDF) matrix

In this module, we will learn about:

- Using Latent Dirichlet Allocation (LDA) for topic modeling
- Evaluating LDA using topic coherence
- Visualizing LDA topics with pyLDAvis for data exploration
- Extracting document-topic information
- Measuring document similarity using cosine similarity metric

Module completion checklist

Objective	Complete
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	
Explore and interpret LDAvis plot	
Extract document-topic information and inspect original article headlines	
Describe cosine similarity as a metric and how it applies to text	

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 a solution to both of these statements by running a LDA model on the corpus

An introduction to LDA

- **Latent Dirichlet Allocation (LDA) is a popular algorithm for topic modeling for many reasons, 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
- **The algorithm is summarized in three steps:**
  - Tell the algorithm how many topics you think there are
  - Algorithm will assign every word to a temporary topic
  - 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

Directory settings

- In order to maximize the efficiency of your workflow, you should encode your directory structure into `variables`
- Let the `main_dir` be the variable corresponding to your `booz-allen-hamilton` folder

```
from pathlib import Path
# Set `home_dir` to the root directory of your computer.
home_dir = Path.home()

# Set `main_dir` to the location of your `booz-allen-hamilton` folder.
main_dir = home_dir / "Desktop" / "booz-allen-hamilton"

# Make `data_dir` from the `main_dir` and remainder of the path to data directory.
data_dir = main_dir / "data"
```

Working directory

- Set working directory to the `data_dir` variable we set

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

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

```
/home/[user-name]/Desktop/booz-allen-hamilton/data
```

Import packages

- Let's import the packages

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

```
# Cosine similarity and clustering packages.
from sklearn.metrics.pairwise import cosine_similarity
from scipy.cluster.hierarchy import ward, dendrogram, fcluster

# Network creation and visualization.
import networkx as nx
from pyvis.network import Network

import pyLDAvis
import pyLDAvis.gensim
import matplotlib.pyplot as plt
```

```
/Library/Frameworks/R.framework/Versions/3.6/Resources/library/reticulate/python/rpytools/loader.py:19:
DeprecationWarning: the imp module is deprecated in favour of importlib; see the module's documentation
for alternative uses
  module = _import(
```

```
import gensim
from gensim import matutils
from gensim.models.coherencemodel import CoherenceModel
```

Import data we saved

```
# Load pickled data and models.
processed_docs = pickle.load(open("NYT_clean.sav","rb"))
dictionary = pickle.load(open("dictionary.sav","rb"))
corpus_tfidf = pickle.load(open("corpus_tfidf.sav","rb"))
DTM = pickle.load(open("DTM.sav","rb"))
```

```
# Load NYT article data from original file.
NYT = pd.read_csv('NYT_article_data.csv')

# Load pre-saved word counts array we pickled.
word_counts_array = pickle.load(open("word_counts_array.sav","rb"))
```

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 **ate salad** and **bananas**
  - My brother bought a **cat** and a **hamster**
  - My **cute dog** loves to **munch** on **bananas**

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**
- **Two guiding principles:**
  - LDA treats each **document** as a mixture of **topics**, such that a given document can be comprised of multiple topics
  - LDA treats each **topic** as a mixture of **words**, and a given word can be shared across multiple topics

Defining LDA at a document level

- Remember how we applied the TF-IDF transformation to each document?
- This will help you understand 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:**
  - I **ate salad** and **bananas** : 100% topic A (food)
  - My brother bought a **cat** and a **hamster**: 100% topic B (animals)
  - My **cute dog** loves to **munch** on **bananas**: 50% topic A (food), 50% topic B (animals)
- **We can derive the proportions that each word constitutes in given topics**
- LDA might produce something like:
  - **Topic A** might comprise words in the following proportions: 40% bananas, 20% ate, 20% salad, 20% munch
  - **Topic B** might comprise words in the following proportions: 25% cat, 25% hamster, 25% dog, 25% cute

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
	cat		hamster
	bananas		munch
	cat		salad

Step 1: number of topics

- **The first step is that we tell the algorithm how many topics you think there are**, usually based on:
  - previous analysis
  - informed decision by a subject matter expert
  - random guess
- **Trying out different estimates allows you to select 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** and 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	
B	dog	?	dog
B	dog	B	dog
B	cat	B	hamster
A	bananas	A	munch
B	cat	A	salad

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 B** and not seen in topic A
- An instance of "dog" picked **randomly** would more likely be about **topic B**

Document 1		Document 2	
B	**dog**	**?**	**dog**
B	**dog**	**B**	**dog**
B	cat	B	hamster
A	bananas	A	munch
B	cat	A	salad

Step 3b: topics in the document

How prevalent are the topics in the document?

- Just within document 2, *the categorized words are split 50/50 between* **topic A** and **topic B**
- Therefore, when inspecting the document, the uncategorized **dog** has a 50/50 chance of referring to either topic

Document 1		Document 2	
B	dog	?	dog
B	dog	**B**	**dog**
B	cat	**B**	**hamster**
A	bananas	**A**	**munch**
B	cat	**A**	**salad**

Which topic then?

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

- **And what could topic A be about?**

Step 3c: assign topic

- The process that we used to allocate **dog** to **topic B** 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	
B	dog	**B**	**dog**
B	dog	**B**	**dog**
B	cat	**B**	**hamster**
A	bananas	**A**	**munch**
B	cat	**A**	**salad**

Module completion checklist

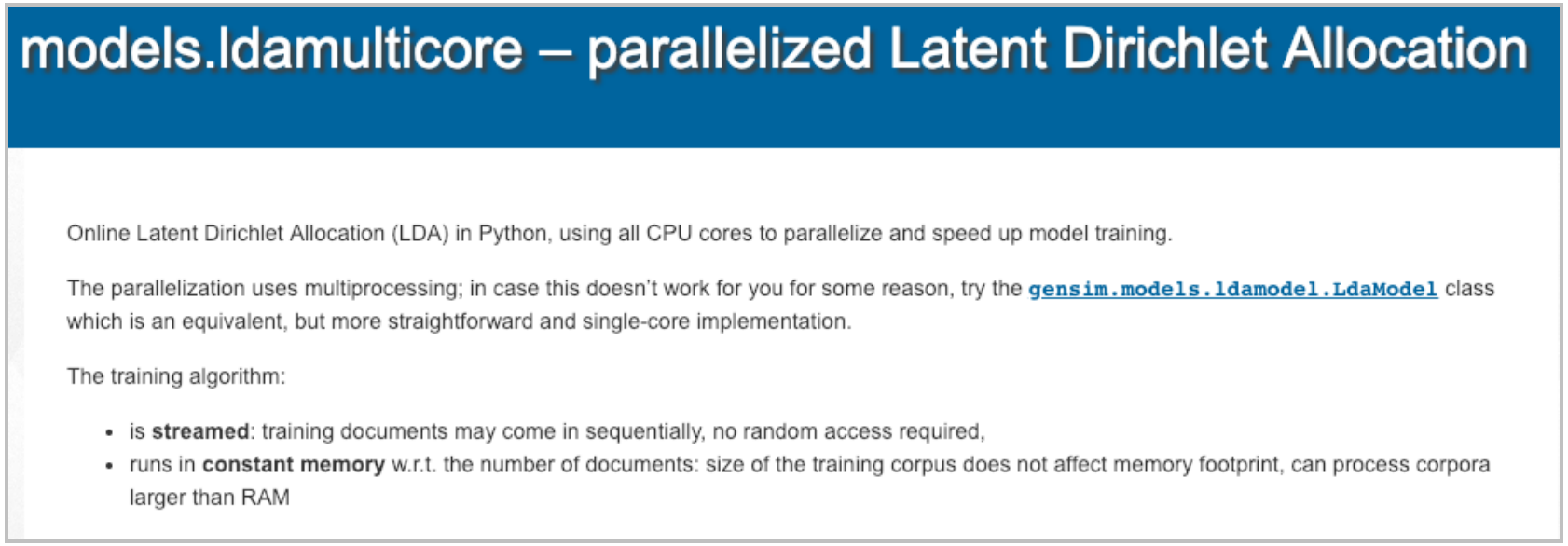
Objective	Complete
Summarize the concept of topic modeling	✔
Perform latent dirichlet allocation (LDA) on frequency counts	
Evaluate results and choose optimal number of topics	
Visualize results of LDA using interactive LDAvis plot	
Explore and interpret LDAvis plot	
Extract document-topic information and inspect original article headlines	
Describe cosine similarity as a metric and how it applies to text	

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 that 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 introduce `models.LdaModel`



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 using `gensim.models.LdaModel`, a robust module from the `gensim` package
- The algorithm we just walked through with the **two documents** will now be applied to **248 documents**

LdaModel

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

```
gensim.models.LdaModel(corpus = None,
                       num_topics = 100,
                       id2word = None,
                       passes = 1,
                       random_state = 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
- `passes`: number of passes through the corpus during training, e.g. how many times to classify each word to each topic
- `random_state`: either a seed to create a random state or a random state object to generate reproducible results

Running LdaModel

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

```
lda_model_tfidf = gensim.models.LdaModel(corpus_tfidf,
                                         num_topics = 5,
                                         id2word = dictionary,
                                         passes = 2,
                                         random_state = 1)
```

- We have our `LdaModel` object now

```
print(lda_model_tfidf)
```

```
LdaModel(num_terms=155, num_topics=5, decay=0.5, chunksize=2000)
```

LDA output

- Let's look at the output - the top words in each of the 5 topics we've chosen

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

```
Topic: 0 Word: 0.034*"state" + 0.023*"even" + 0.020*"charg" + 0.020*"said" + 0.020*"north" +
0.019*"court" + 0.017*"china" + 0.017*"local" + 0.016*"case" + 0.016*"like"
Topic: 1 Word: 0.037*"new" + 0.029*"year" + 0.025*"world" + 0.024*"presid" + 0.023*"two" +
0.020*"offici" + 0.020*"trump" + 0.020*"move" + 0.018*"democrat" + 0.017*"hous"
Topic: 2 Word: 0.031*"latest" + 0.028*"time" + 0.027*"say" + 0.025*"local" + 0.023*"one" +
0.022*"young" + 0.022*"defend" + 0.021*"win" + 0.019*"final" + 0.017*"investig"
Topic: 3 Word: 0.032*"leagu" + 0.022*"first" + 0.020*"billion" + 0.019*"premier" + 0.018*"group" +
0.017*"start" + 0.016*"forward" + 0.016*"saturday" + 0.016*"back" + 0.016*"south"
Topic: 4 Word: 0.031*"said" + 0.024*"accus" + 0.022*"week" + 0.021*"night" + 0.019*"play" +
0.018*"monday" + 0.018*"help" + 0.017*"show" + 0.017*"tuesday" + 0.017*"friday"
```

- 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 `processed_docs` are clustered into the topics identified by LDA

```
# 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']
```

Classify our documents within topics (cont'd)

```
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: 0.7982504367828369
Topic: 0.032*"leagu" + 0.022*"first" + 0.020*"billion" + 0.019*"premier" + 0.018*"group" +
0.017*"start" + 0.016*"forward" + 0.016*"saturday" + 0.016*"back" + 0.016*"south"

Score: 0.0506187379360199
Topic: 0.031*"latest" + 0.028*"time" + 0.027*"say" + 0.025*"local" + 0.023*"one" + 0.022*"young" +
0.022*"defend" + 0.021*"win" + 0.019*"final" + 0.017*"investig"

Score: 0.050531044602394104
Topic: 0.034*"state" + 0.023*"even" + 0.020*"charg" + 0.020*"said" + 0.020*"north" + 0.019*"court" +
0.017*"china" + 0.017*"local" + 0.016*"case" + 0.016*"like"

Score: 0.050302185118198395
Topic: 0.037*"new" + 0.029*"year" + 0.025*"world" + 0.024*"presid" + 0.023*"two" + 0.020*"offici" +
0.020*"trump" + 0.020*"move" + 0.018*"democrat" + 0.017*"hous"

Score: 0.050297632813453674
Topic: 0.031*"said" + 0.024*"accus" + 0.022*"week" + 0.021*"night" + 0.019*"play" + 0.018*"monday" +
0.018*"help" + 0.017*"show" + 0.017*"tuesday" + 0.017*"friday"
```

Knowledge check 1

Exercise 1

Module completion checklist

Objective	Complete
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	
Explore and interpret LDAvis plot	
Extract document-topic information and inspect original article headlines	
Describe cosine similarity as a metric and how it applies to text	

LDA - how to evaluate results?

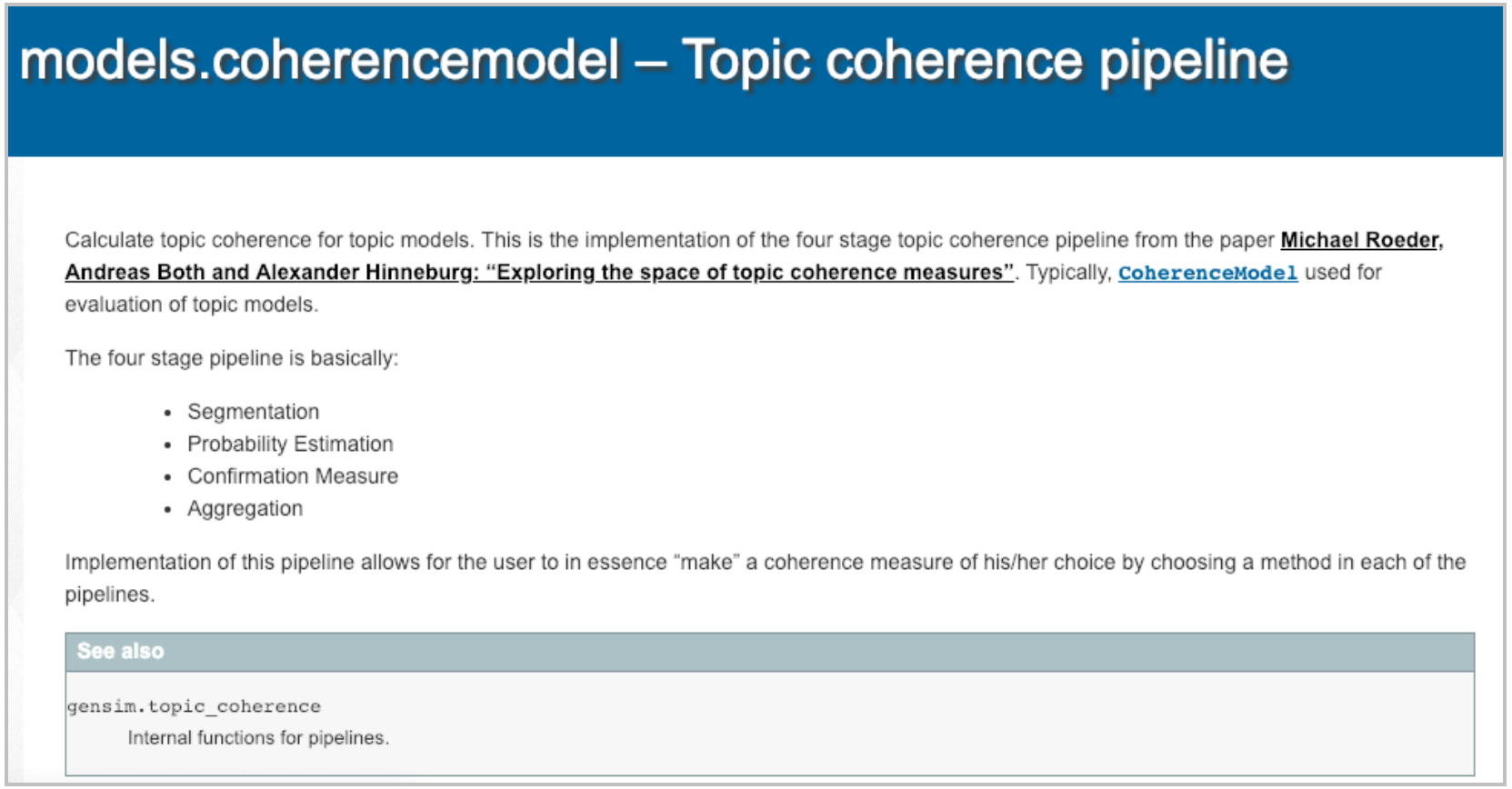
- By **manual inspection** of documents within detected topics
  - A person must assess whether the topics make sense and whether the documents actually represent those topics
  - *This method is subjective, manual, and not scalable*, although it's often the best way to assess the model results for best interpretation by humans
- By **using a metric** that allows us to assess the model in an automated way
  - This method should be used to assess model performance when manual inspection is not possible OR in combination with human assessment

LDA - topic coherence as evaluation metric

- The measure we will focus on today is **topic coherence**
- Topic coherence is a score that ranges between `0` and `1` - *the higher the coherence, the better the topic model*
- With **topic coherence**, we will be able to **evaluate our results**
- It would also allow us to **build a plot** that will help us **choose the optimal number of topics** for our LDA model

LDA - CoherenceModel function

- 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 metric and its origins



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:** quantitatively measuring of the subtopic quality
  - **Confirmation measure:** determining quality based on some predefined measure
  - **Aggregation:** combining all quality numbers and deriving one number for overall quality
- There are **two types of measures** in topic coherence:
  - **Intrinsic:** compares a word only to the preceding and succeeding words respectively
  - You need the ordered word set
  - 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: compute the score

- To generate a coherence model, you need to instantiate an object that belongs to `CoherenceModel` class

```
coherence_model = CoherenceModel(model = your_lda_model,
                                 texts = your_documents,
                                 dictionary = your_model_dictionary,
                                 coherence = 'c_v') #<- coherence metric
```

- Once you generate the model object, you need to call the `get_coherence()` method to get the actual coherence score

```
coherence_score = coherence_model.get_coherence()
```

Topic coherence: compute the score (cont'd)

- Let's calculate topic coherence on `lda_model_tfidf` model; the parameters we need are:
  - original documents: `processed_docs`
  - dictionary built based on our corpus: `dictionary`
  - lda model: `lda_model_tfidf`

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

```
Coherence Score:  0.47755503368452945
```

- The score of about `0.5` is neither good nor particularly bad
- Scores close to `1` are very rare and unrealistic, but the ideal is for a score > `0.5`
- Scores between `0.4` and `0.5` usually indicate that the number of topics chosen was not optimal

Finding the optimal number of topics

- To find the optimal number of topics, let's look at the coherence scores for a **range of topic numbers**
- We can build a convenience 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 to compute coherence

```
def compute_coherence_values(dictionary, corpus, texts, limit, start = 2, step = 1):
    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 above function outputs 2 things:
  - `model_list`: a list of LDA topic models
  - `coherence_values`: coherence values corresponding to the LDA model with respective number of topics

Run compute_coherence_values function

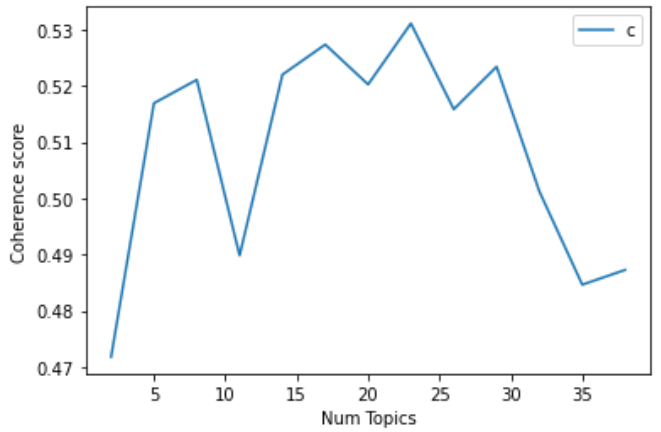
```
np.random.seed(1)

start = 2  #<- let's start with 2 topics
limit = 40 #<- and set a max value at 40 (arbitrary number, something that a human can interpret)
step = 3   #<- set step to be neither too small, nor to big, so it doesn't skip over too much
topic_range = range(start, limit, step)


model_list, coherence_values = compute_coherence_values(dictionary = dictionary,
                                                         corpus = corpus_tfidf,
                                                         texts = processed_docs,
                                                         start = start,
                                                         limit = limit,
                                                         step = step)
```

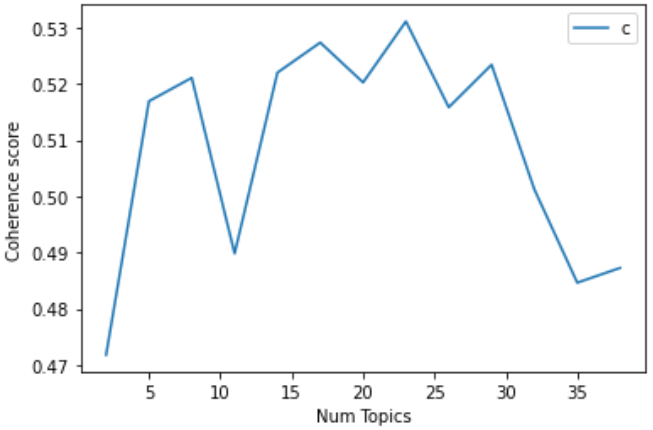
Plot coherence scores

```
# Plot graph of topic list.
plt.plot(topic_range, coherence_values)
plt.xlabel("Num Topics")
plt.ylabel("Coherence score")
plt.legend(("coherence_values"), loc = 'best')
plt.show()
```



Interpret the coherence score plot

- We can see from this plot that the coherence score peaks in a few places, and reaches **maximum between 20 and 25 topics**
- Since a human can actually interpret this number of topics, we can use it to get the optimal number
- Sometimes the coherence scores won't have clear peaks, in which case we:
  - Pick a coherence score value that is fairly high but that that a human can still interpret,
  - Tune ***other LDA model parameters***, or
  - Opt for an alternative topic modeling algorithm



Get optimal number of topics

- To get the maximum value from a list or array, we can use `numpy`'s native function `max`

```
max_coherence = np.max(coherence_values)
print(max_coherence)
```

```
0.5310675148866303
```

- To get the index at which the maximum value is located in a list or array, we can use `numpy`'s native function `argmax`
- Knowing the index of the maximum value, we can get the number of topics from the `topic_range` list

```
optimal_num_topics = topic_range[np.argmax(coherence_values)]
print(optimal_num_topics)
```

```
23
```

- We can improve the score by setting the number of topics in our model to `23`
- Since this number of topics is fairly reasonable for a human being to interpret, we'll stick with it!

Knowledge check 2

Module completion checklist

Objective	Complete
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	
Explore and interpret LDAvis plot	
Extract document-topic information and inspect original article headlines	
Describe cosine similarity as a metric and how it applies to text	

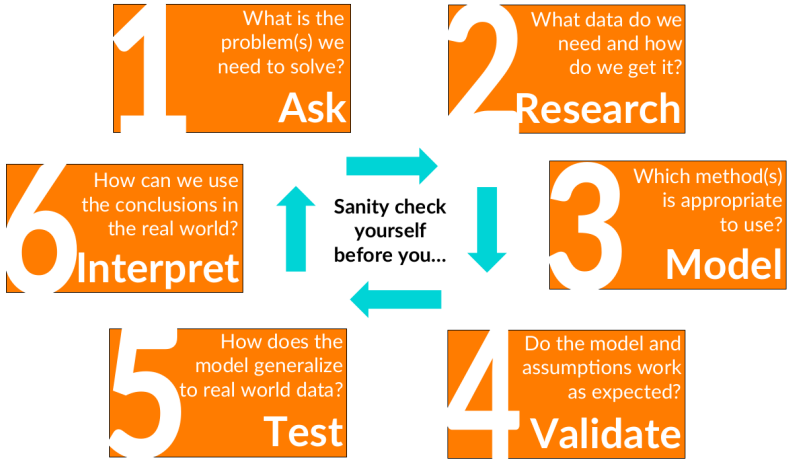
Data wrangling and exploration

Remember, a data scientist must be able to:

- **Explore** the data to generate a hypothesis

Clustering and visualization are two great methods to explore and look for patterns in your data!

- So far, we have used LDA to obtain **5 topics** and also found that the optimal number of topics is "
- We will use the model with 5 topics in the slides to follow for the demonstration purposes



Visualize topics generated with LDA

- **In the previous module, we went over the algorithm in three steps:**
  - told the algorithm how many topics we think there are
  - algorithm assigned every word to a temporary topic
  - algorithm checked and updated topic assignments
- We have performed LDA on the **NY Times article snippets** and assessed the numerical metrics of our model's performance
- Now let's look at how all of those metrics and numbers fit together by **visualizing the LDA**
- We will be using `pyLDAvis` package that is a Python wrapper around a very popular R package called `LDAvis`
- You can find the original publication of `LDAvis` ***here***

Visualize topics generated with LDA: pyLDAvis

- We created our LDA model using `gensim`, which integrates easily with `pyLDAvis` through 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

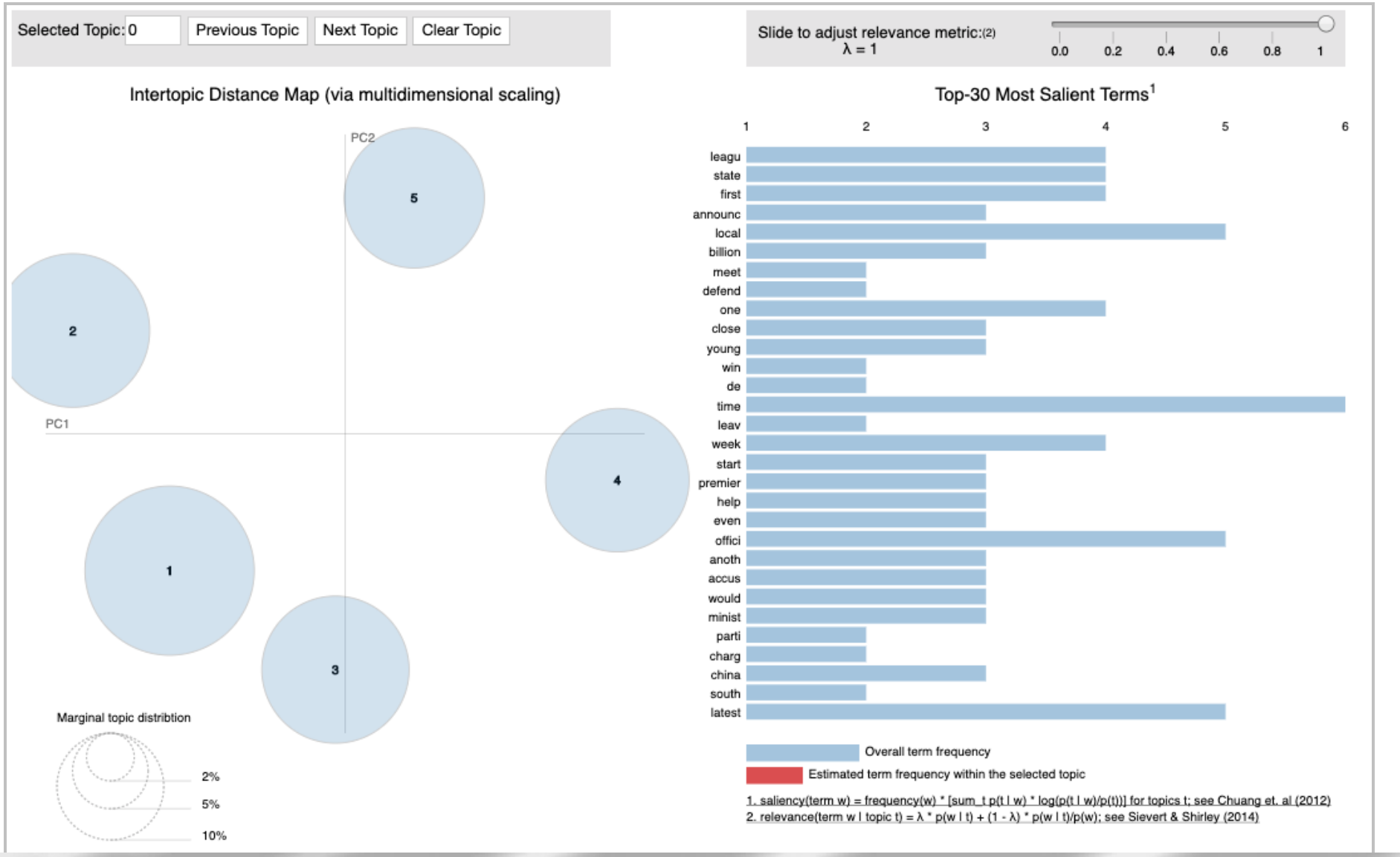
```
# Prepare LDA vis object by providing:
vis = pyLDAvis.gensim.prepare(lda_model_tfidf,    #<- model object
                              corpus_tfidf,       #<- corpus object
                              dictionary)         #<- dictionary object
```

- 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)



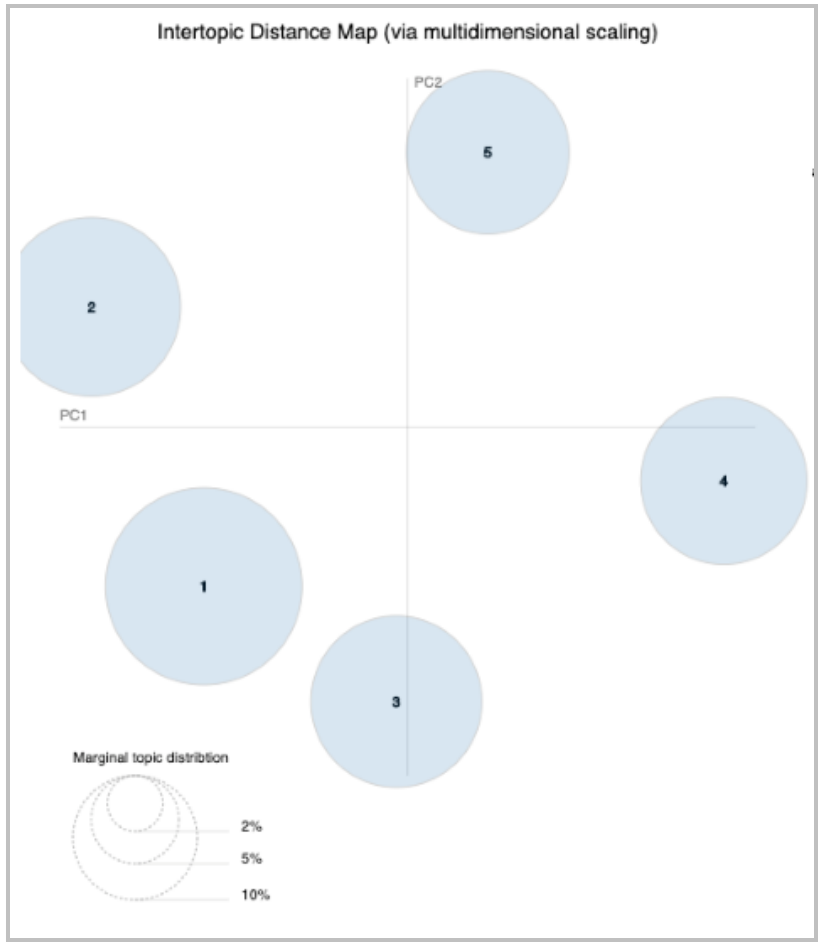
Exercise 2

Module completion checklist

Objective	Complete
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	✔
Explore and interpret LDAvis plot	
Extract document-topic information and inspect original article headlines	
Describe cosine similarity as a metric and how it applies to text	

LDA visualization: topic distribution

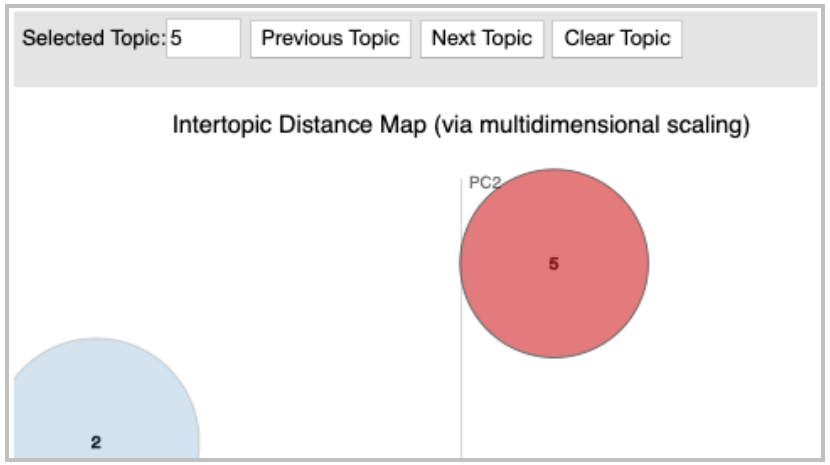
- The left-hand side shows the **topics** represented by the circles



- 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?*

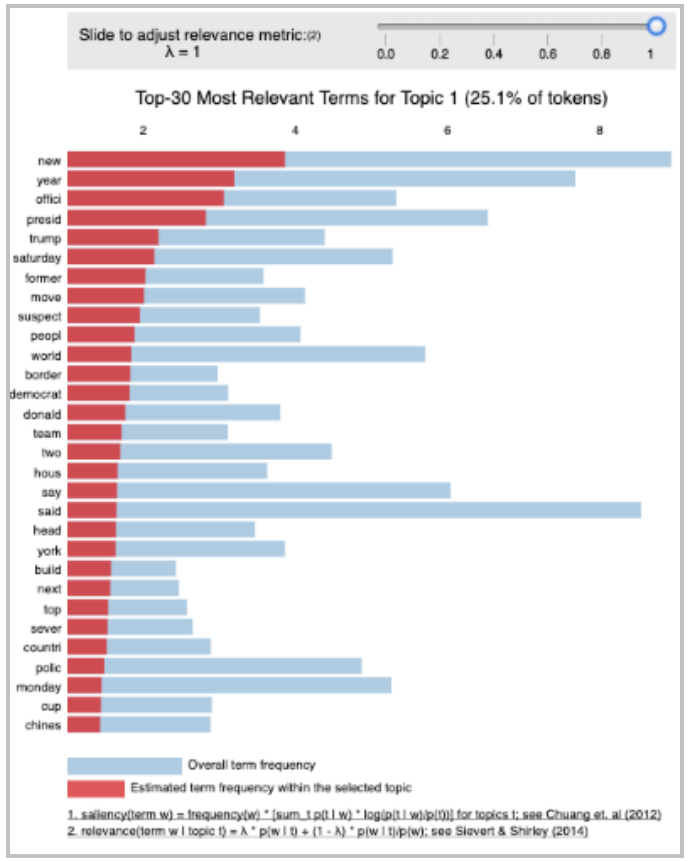
LDA visualization: select topic

- 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

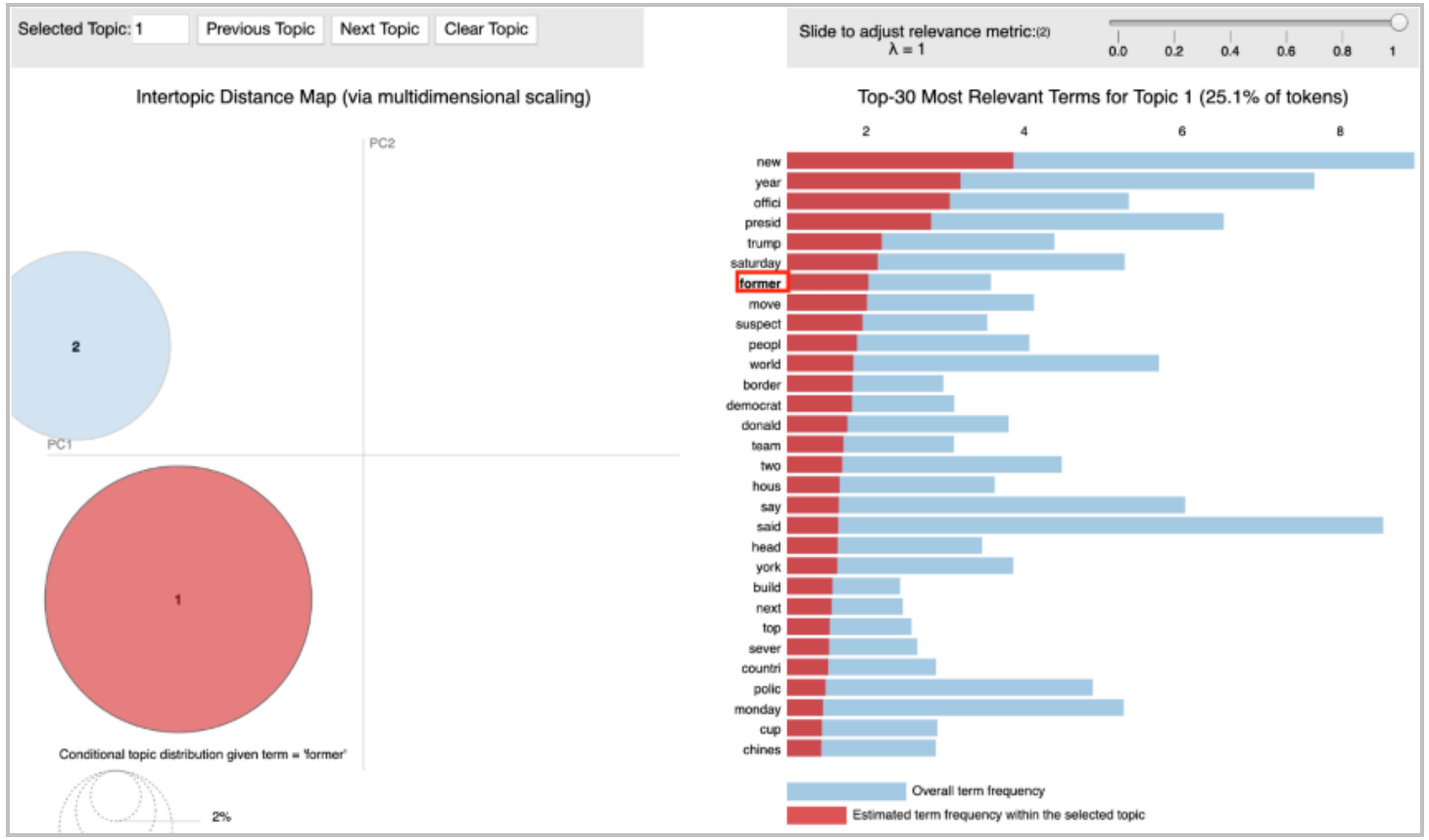
- The right-hand side shows the most relevant terms in each topic, once the topic is selected



- The **blue bars** represent the overall term frequency in corpus
- The **maroon bars** represent term frequency in a 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 a topic with respect to its frequency within a 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, when hovered over the term `former`, we see that it appears in topics `1` and `2`, but it is more prominent in topic `1`

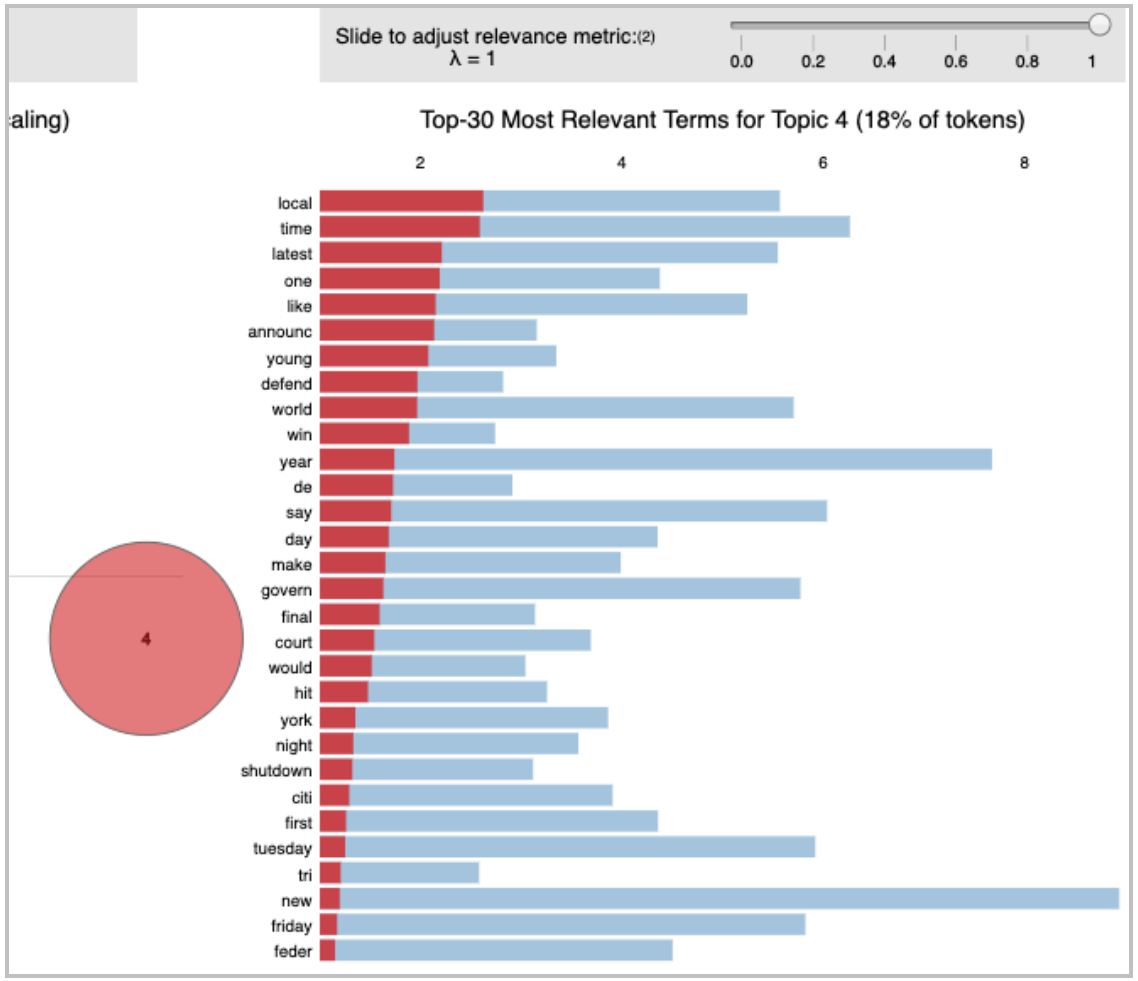


LDA visualization: how to determine topics

- **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**, and thus make better inferences
- This process requires the subject matter expertise and some common sense, as the naming conventions of topics are subjective

LDA visualization: name topic 4

- Let's take a look at the most relevant terms in topic 4:



- 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?

Note: The same can be performed for all other topics as well

LDA visualization: name other topics

Here's something to consider and interpret more on the results :

- Which topics were the hardest to label?
- Why do you think that's the case?
- What can be done to improve the model overall? *Hint: Text cleanup*

Module completion checklist

Objective	Complete
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	✔
Explore and interpret LDAvis plot	✔
Extract document-topic information and inspect original article headlines	
Describe cosine similarity as a metric and how it applies to text	

Extracting documents for each topic

- This graph has provided us with a lot of interesting insights, but it doesn't show **which topics are assigned to each document**
- We can easily extract that information from our data and model and supplement the graph
- We have previously loaded the original `NYT` data and the `word_counts_array`
- We will use them to trace the document IDs in our corpus to documents in the original data and inspect document assignments

Get topic probabilities for a document

- Let's say we would like to get topic probabilities for the first document in our corpus

```
# Select the index of the document in corpus.
doc_num = 0
```

```
# Extract the vector of tf_idf weights for the document.
doc_vec = corpus_tfidf[doc_num]
print(doc_vec)
```

```
[(0, 0.25954540320095165), (1, 0.29127945804505345), (2, 0.3055490022471601), (3, 0.3055490022471601),
(4, 0.20976922764421047), (5, 0.29127945804505345), (6, 0.268763787812797), (7, 0.25954540320095165),
(8, 0.23076512567108073), (9, 0.5825589160901069)]
```

```
# Extract topic probabilities for that document.
doc_topics = lda_model_tfidf.get_document_topics(doc_vec)
print(doc_topics)
```

```
[(0, 0.05053181), (1, 0.050302185), (2, 0.05061681), (3, 0.79825157), (4, 0.05029764)]
```

- We can see that topic `4` has the highest probability for document `1` (remember, it's listed as `3` because the index starts at `0`)

Get topic probabilities for a document (cont'd)

- Let's get the best topic and its probability for the document programmatically

```
# Initialize maximum probability score.
max_prob = 0
# Initialize best topic.
best_topic = 0

# Loop over all topics for the document.
for topic in doc_topics:

    if max_prob <= topic[1]:  #<- if current topic's probability is as high as max
        max_prob = topic[1]   #<- make current topic's probability the new max
        best_topic = topic[0] #<- make current topic best

# Create a tuple with information we just got.
doc_topic_pair = (doc_num, best_topic, max_prob)
print(doc_topic_pair)
```

```
(0, 3, 0.79825157)
```

- From this tuple, the best topic for document `1` is `4` and its probability is almost `80%`
- Let's define a function that will let us extract this information for a document given an LDA model

Get topic probabilities for a document (cont'd)

- The function we create here is based upon skills we've already learned, we just wrap the code into a `def` structure and add a `return` statement!

```
# Put it all together into a function that returns a tuple
# with the index of the document, the best fit topic, and its probability.
def GetDocTopicPair(doc_num, corpus, lda_model_tfidf):

    # Extract the vector of tf_idf weights for the document.
    doc_vec = corpus[doc_num]

    # Extract topic probabilities for that document.
    doc_topics = lda_model_tfidf.get_document_topics(doc_vec)

    max_prob = 0
    best_topic = 0

    for topic in doc_topics:

        if max_prob <= topic[1]:
            max_prob = topic[1]
            best_topic = topic[0]

    doc_topic_pair = (doc_num, best_topic, max_prob)

    return(doc_topic_pair)
```

Get topic probabilities for all documents

- Now that we have a function that extracts information for a document, we can apply it to each document in our corpus by using a loop

```
# Create an empty list of the same length as the number of documents.
doc_topic_pairs = [None]*dictionary.num_docs

# Loop through a range of document indices.
for i in range(dictionary.num_docs):
    # For each document index, get the document-topic tuple.
    doc_topic_pairs[i] = GetDocTopicPair(i, corpus_tfidf, lda_model_tfidf)

print(doc_topic_pairs[:10])
```

```
[(0, 3, 0.7982507), (1, 3, 0.74036914), (2, 1, 0.7577274), (3, 1, 0.72685695), (4, 2, 0.7742631), (5,
4, 0.7311213), (6, 0, 0.79679924), (7, 0, 0.7129077), (8, 3, 0.7747945), (9, 2, 0.5289931)]
```

- We now have a list of tuples that contain:
  - document id
  - topic id
  - topic probability for that document

Create a dataframe with doc-topic assignments

- Let's put all of these tuples into a neat dataframe, which will contain the following columns:
  - `doc_id`
  - `best_topic`
  - `best_probability`

```
# Make a dataframe out of a list of tuples.
doc_topic_pairs_df = pd.DataFrame(doc_topic_pairs)

# Assign column names to the dataframe.
doc_topic_pairs_df.columns = ["doc_id", "best_topic", "best_probability"]
print(doc_topic_pairs_df.head())
```

```
   doc_id  best_topic  best_probability
0       0           3          0.798251
1       1           3          0.740369
2       2           1          0.757727
3       3           1          0.726857
4       4           2          0.774263
```

Matching document ids to NYT data

- When cleaning, we had to remove all documents with word counts under `5`, offsetting our document indexing
- Let's retrieve it and assign original index from `NYT` data to our `doc_topic_pairs_df` dataframe
- Article `122` isn't in the corpus, so it shouldn't be in the dataframe

```
# Find indices of articles that we kept.
valid_snippets = np.where(word_counts_array >= 5)[0]
print(valid_snippets[120:125])
```

```
[120 121 123 124 125]
```

```
# Now assign the index of the original article to be the index of the dataframe.
doc_topic_pairs_df.index = valid_snippets
print(doc_topic_pairs_df.iloc[120:125, ])
```

```
     doc_id  best_topic  best_probability
120     120           1          0.798328
121     121           1          0.727164
123     122           2          0.659608
124     123           0          0.545718
125     124           4          0.740956
```

Inspect documents for a given topic

- Now that we have all pieces in place, we can retrieve all documents assigned to a topic and inspect them
- For demonstration purposes, we will do this for topic `3` and will only output the top `10` documents in that topic

```
# Filter and sort all documents assigned to topic 3 by probability in descending order.
topic3_docs = doc_topic_pairs_df.query("best_topic==2")
topic3_docs = topic3_docs.sort_values(by = "best_probability", ascending = False)
print(topic3_docs.head())
```

```
     doc_id  best_topic  best_probability
75       75           2          0.804347
181     179           2          0.794047
70       70           2          0.781515
171     169           2          0.780105
184     182           2          0.775857
```

```
# Let's see how many documents were assigned to that topic.
print(topic3_docs.shape)
```

```
(45, 3)
```

Inspect documents for a given topic (cont'd)

- Get the indices of the top `10` documents in the topic and then the headlines of the top `10` documents in the topic from the original `NYT` dataframe

```
# Let's get the top 10 documents that were assigned to that topic.
top_10 = topic3_docs.index[0:10,]
```

```
# Inspect the top 10 documents in topic 3.
NYT_articles_topic3 = NYT.loc[top_10, :]
print(NYT_articles_topic3[['headline']])
```

```
                                              headline
75   Greek Minister Criticizes Police Over Clashes ...
181  India Force Rare Australia Follow-On in Sydney...
70   Air Traffic Controllers' Union Sues Over Unpai...
171  Britain Testing 'No-Deal' Scenario as Brexit V...
184  Australia Dismissed for 300, India Enforce Fol...
144  Froome, Thomas to Skip Giro d'Italia and Focus...
4    Froome, Thomas to Skip Giro d'Italia and Focus...
97   Netflix's 'Sex Education' Is a Trans-Atlantic ...
16   Reaction to Andy Murray's Impending Retirement...
163  The Latest: White House Invites Hill Leaders t...
```

*Note: the topic indexing in the model and in the visualization is **not** the same! Be careful and match them up based on the relevant words instead. In this case, topic 3 corresponds to topic 2 in the LDAvis*

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")
```

Knowledge check 3

Exercise 3

Module completion checklist

Objective	Complete
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	✔
Explore and interpret LDAvis plot	✔
Extract document-topic information and inspect original article headlines	✔
Describe cosine similarity as a metric and how it applies to text	

Field trip

- Visit ***databasic.io/en/samediff/***
- Select two artists under the “use samples” menu and hit “COMPARE”
- How similar are the artists?

- Select two different artists. Are they more or less similar than the first pair you selected?

Comparing terms and documents in text

- One of the most common analyses in text mining is **finding similar terms and documents**
- Using contextual comparison allows us to sort large amounts of documents or terms, instead of using an exact search
- Just like with any other two data points, to find **two terms or documents** that are **similar** in their meaning, we need to **measure the distance** between them

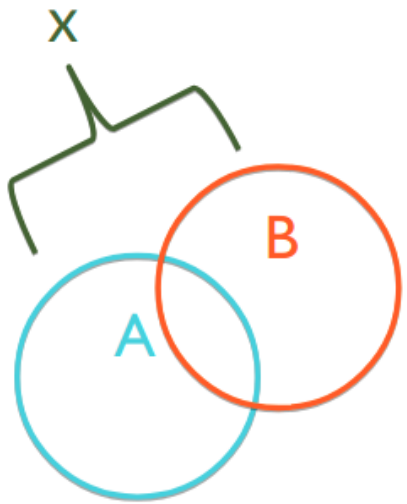
Measuring similarity in text

- When measuring similarity between two data points, the usual metric would be ***Euclidean distance***
- When working with text data, we use a metric called **cosine similarity**
- Since we will be using cosine similarity today, let's go into a little more detail.

Cosine similarity vs distance

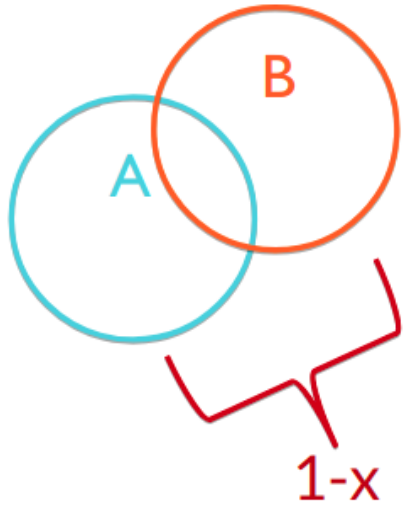
Cosine similarity

- Measure of how close two objects are
- Similarity between `A` and `B` is high, because the objects are close to each other
- If `similarity(A, B) = x`



Cosine distance

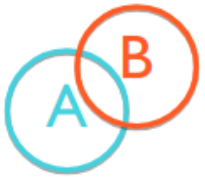
- Measure of how far away two objects are
- Distance between `A` and `B` is small, because the objects are close to each other
- Then `distance(A, B) = 1-x`



Cosine similarity vs distance (cont'd)

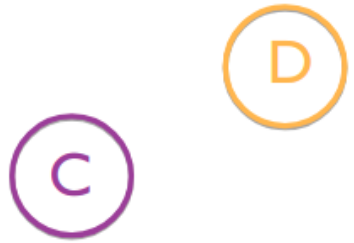
Cosine similarity

- If `similarity(A, B) = 0.8`
- If `similarity(C, D) = 0.1`



Cosine distance

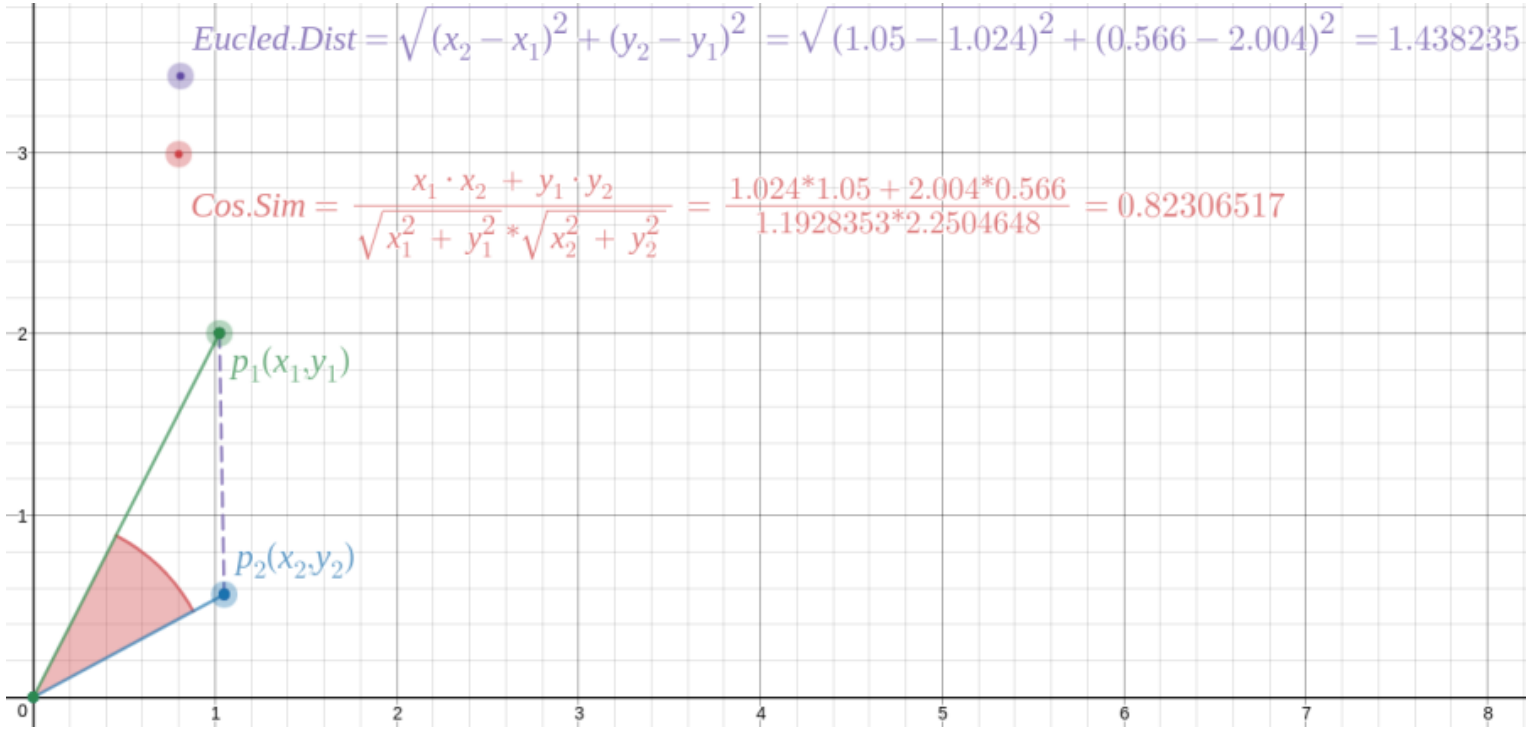
- then `distance(A, B) = 0.2`
- then `distance(C, D) = 0.9`



Note that cosine similarity & cosine distance will always be values between `[0,1]`!

Cosine similarity: what is it?

- **Cosine similarity** is $cos(a)$, where a is an angle between the two *vectors*
- These vectors are either *terms* or *documents*, depending on what we would like to compare
- In order **to compute cosine similarity between documents, we need to know their numeric representation**, which we already have in the form of `corpus_tfidf`



Cosine similarity using scikit-learn

- We'll use `cosine_similarity` to compute cosine similarity for each possible pair of documents
- The input for this function is a matrix of row vectors
- `DTM` weighted with `TfIdf` is a matrix where each row represents a document vector with term weights

sklearn.metrics.pairwise.cosine_similarity

`sklearn.metrics.pairwise.` **`cosine_similarity`** (*X*, *Y=None*, *dense_output=True*) [source]

Compute cosine similarity between samples in X and Y.

Cosine similarity, or the cosine kernel, computes similarity as the normalized dot product of X and Y:

K(X, Y) = <X, Y> / (||X||*||Y||)

On L2-normalized data, this function is equivalent to linear_kernel.

Read more in the User Guide.

Parameters:

X : *ndarray or sparse array, shape: (n_samples_X, n_features)*
Input data.

Y : *ndarray or sparse array, shape: (n_samples_Y, n_features)*
Input data. If `None`, the output will be the pairwise similarities between all samples in `X`.

dense_output : *boolean (optional), default True*
Whether to return dense output even when the input is sparse. If `False`, the output is sparse if both input arrays are sparse.

New in version 0.17: parameter `dense_output` for dense output.

Returns:

kernel matrix : *array*
An array with shape (n_samples_X, n_samples_Y).

Compute TF-IDF weighted TDM

- To convert a `gensim` corpus object like `corpus_tfidf`, we will use the `matutils` module from `gensim`
- The function `matutils.corpus2dense()` needs three things
  - a corpus object
  - number of unique terms in the corpus
  - number of documents in the corpus
- It will generate a `TDM` (i.e. a `Term-Document Matrix`), which is a **transpose** of the `DTM`

Compute TF-IDF weighted TDM (cont'd)

- Generate a `TDM` from corpus weighted with TF-IDF
- Check the dimensions of the type of the `TDM`

```
# Convert corpus weighted with TF-IDF to a TDM matrix.
TDM_tf_idf_matrix = matutils.corpus2dense(corpus_tfidf,
                                          DTM.shape[1],
                                          DTM.shape[0])
print(TDM_tf_idf_matrix.shape)
print(type(TDM_tf_idf_matrix))
```

```
(1921, 248)
<class 'numpy.ndarray'>
```

- We have a 2D array with `1921` unique terms and `248` documents

Transpose TDM to DTM

- Convert a `TDM` into a `DTM` by using a `.transpose()` method (it's standard for any `numpy` array)
- Check the dimensions of the matrix

```
# Transpose matrix to get the DTM.
DTM_tf_idf_matrix = TDM_tf_idf_matrix.transpose()
print(DTM_tf_idf_matrix.shape)
```

```
(248, 1921)
```

- We have a 2D array with `248` documents and `1921` unique terms

Compute cosine similarity matrix

- Compute cosine similarity for the `DTM_tf_idf_matrix`

```
# Compute similarity matrix (a numpy 2D array).
similarity = cosine_similarity(DTM_tf_idf_matrix)
print(type(similarity))
print(similarity.shape)
```

```
<class 'numpy.ndarray'>
(248, 248)
```

Cosine similarity matrix characteristics

	doc_0	doc_1	doc_2
doc_0	**1**	0	0.021
doc_1	0	**1**	0.89
doc_2	0.021	0.89	**1**

- It is *square*: the number of rows is the same as the number of columns
- It is *symmetric*: the entries above the diagonal are mirror images of entries below
- Values along the *diagonal* are always 1

Note: This is only true when comparing documents, sentences, words, etc to each other. If we make a matrix of document-to-query similarity, for instance, the matrix is going to be neither square, nor symmetric, nor will it have 1's on the diagonal!

Cosine similarity matrix: interpretation

- When we want to compute cosine similarity between **documents**, we give the function a `DTM` that has `248` rows
  - As a result, we get `248x248` similarity matrix
  - Each row and each column represents a document
  - The value at the intersection is the similarity score between them

- If we wanted to compute cosine similarity between **terms**, what would we give as input to the function and what would be the output?
- Why do you think we have all `1`s along the diagonal?
- Why do you think the values above the diagonal are mirror images of the values below the diagonal?

Cosine distance matrix: interpretation

- If we wanted to compute cosine distance between **terms**, what would we give as input to the function and what would be the output?
  Input: TDM that has 1921 rows, Output: 1921X1921 Distance matrix Each row here represents a 'term' present in the corpus
- Why do you think we have all 0s along the diagonal?
  Hint: cosine distance of 0 means the documents are perfectly similar, 1 means they are completely dissimilar!
- Why do you think the values above the diagonal are mirror images of the values below the diagonal?
  Hint: The matrix is a distance representation of each term against every other term

Knowledge check 4

Exercise 4

Module completion checklist

Objective	Complete
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	✔
Explore and interpret LDAvis plot	✔
Extract document-topic information and inspect original article headlines	✔
Describe cosine similarity as a metric and how it applies to text	✔

This completes our module

Congratulations!

Wondering what's next?

- You may be interested in the following course, which Data Society offers for Booz Allen Hamilton:
  - **Advanced Text Mining & NLP** - In this course, students will leverage text mining skills to generate context from language patterns. Students will learn hierarchical clustering and visualization techniques to evaluate similarities and patterns and identify trends in data. By the end of this course, students will have the know-how to apply complex sentiment analysis tools and methods to amplify and fine-tune models to provide greater accuracy in interpreting language-based data.