Topic Modeling in NLP - Topic modeling - 2

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

Module completion checklist

Objective	Complete
Load data and perform text cleaning on one document	
Perform text cleaning on entire corpus	

Load packages

- Let's import all packages needed for this course

```
# Helper packages.
import os
from pathlib import Path
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
# Packages with tools for text processing.
import nltk
nltk.download('punkt')
```

```
nltk.download('stopwords')
```

```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem.porter import PorterStemmer
from sklearn.feature_extraction.text import CountVectorizer
import gensim
from gensim import corpora, models
from pprint import pprint
from gensim.models.coherencemodel import CoherenceModel

# Other plotting tools.
import pyLDAvis
import pyLDAvis.gensim
import matplotlib.pyplot as plt
```

Directory settings

- In order to maximize the efficiency of your workflow, you should encode your directory structure into `variables`
- We will use the `pathlib` library
- Let the `main_dir` be the variable corresponding to your `course` folder
- Let `data_dir` be the variable corresponding to your `data` folder

```
# Set 'main_dir' to location of the project folder
home_dir = Path(".").resolve()
main_dir = home_dir.parent.parent
print(main_dir)
```

```
data_dir = str(main_dir) + "/data"
print(data_dir)
```

```
plot_dir = str(main_dir) + "/plots"
if not os.path.exists(plot_dir):
    os.makedirs(plot_dir)
print(plot_dir)
```

Dataset

- In order to implement what you learn in this course, we will be using the NYT_article_data.csv dataset
- We will be working with columns from the dataset such as:
  - `web_url`
  - `headline`
  - `snippet`
  - `word_count`
  - `source`

Load data

- We will now load the dataset and save it as `df`

```
# Let's load and prepare the dataset for creating Document-Term Matrix
df = pd.read_csv(str(data_dir)+"/"+ "NYT_article_data.csv")
print(df.head())
```

```
                                              web_url  ...  id
0  https://www.nytimes.com/reuters/2019/01/01/spo...  ...   8
1  https://www.nytimes.com/reuters/2019/01/01/wor...  ...   9
2  https://www.nytimes.com/aponline/2019/01/01/sp...  ...  10
3  https://www.nytimes.com/2019/01/09/arts/design...  ...  11
4  https://www.nytimes.com/aponline/2019/01/10/sp...  ...  12

[5 rows x 8 columns]
```

- We will be focusing on the `snippet` column for this course

Check for NAs

- Check and drop any NAs or empty values from the `snippet` column as they are not relevant for our analysis

```
# Print total number of NAs.
print(df["snippet"].isna().sum())
```

```
0
```

```
# Drop NAs if any.
df = df.dropna(subset = ["snippet"]).reset_index(drop=True)
print(df["snippet"].isna().sum())
```

```
0
```

```
# Isolate the `snippet` column.
df_text = df["snippet"]
```

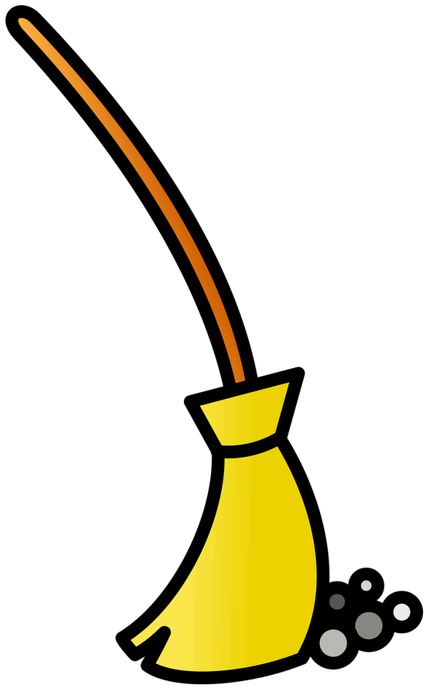
Tokenization: split each document into words

- NLTK's functions operate on **tokens**
- A **token** is the smallest unit of text of interest - in our case, it will be a **word**
- We will use `word_tokenize()` method to split each document into tokens

```
# Tokenize each document into a large list of tokenized documents.
df_tokenized = [word_tokenize(df_text[i]) for i in range(0, len(df_text))]
```

“Bag-of-words” analysis: cleaning text flow

1. Convert all characters to lowercase
2. Remove stop words
3. Remove punctuation, numbers, and all other symbols that are not letters of the alphabet
4. Stem words
5. Remove extra white space (if needed)



Convert characters to lowercase

- To convert characters to lowercase, we will use `.lower()` function
  - We call the method: `character_string.lower()`
- Since every element of the `document_words` list is a character string, we will convert each word in the document using a **list comprehension**

```
# Let's take a look at the first tokenized document
document_words = df_tokenized[0]
print(document_words)
```

```
['Nick', 'Kyrgios', 'started', 'his', 'Brisbane', 'Open', 'title', 'defense', 'with', 'a',
'battling', '7-6', '(', '5', ')', '5-7', '7-6', '(', '5', ')', 'victory', 'over',
'American', 'Ryan', 'Harrison', 'in', 'the', 'opening', 'round', 'on', 'Tuesday', '.']
```

- Let's test out the cleaning flow on this single document first
- Then we will apply the cleaning steps to all documents in our corpus

```
# 1. Convert to lowercase.
document_words = [word.lower() for word in document_words]
print(document_words[:10])
```

```
['nick', 'kyrgios', 'started', 'his', 'brisbane', 'open', 'title', 'defense', 'with', 'a']
```

Remove stop words

- In any language, there are words that carry little specific subject-related meaning, but are necessary to bind words into coherent sentences together
- Such words are the most frequent and they usually need to be taken out, as they create unnecessary noise
- They are called **stop words** and NLTK has a corpus of such words that can be called by using `stopwords` module
  - To get English stop words, we will call `stopwords.words('english')` method

```
# 2. Remove stop words.
# Get common English stop words.
stop_words = stopwords.words('english')
print(stop_words[:10])
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're"]
```

```
# Remove stop words.
document_words = [word for word in document_words if not word in stop_words]
print(document_words[:10])
```

```
['nick', 'kyrgios', 'started', 'brisbane', 'open', 'title', 'defense', 'battling', '7-6',
'(']
```

Remove non-alphabetical characters

- For text analysis based on “bag-of-words” approach, we only use words
- We remove numbers, punctuation or anything other than alphabetical characters
- We will use `.isalpha()` method to check whether the characters are alphabetical or not
  - We call the method: `character_string.isalpha()`
  - Since every element of the `document_words` list is a character string, we will check each token in the document using a **conditional** `if` inside of the **list comprehension**

```
# 3. Remove punctuation and any non-alphabetical characters.
document_words = [word for word in document_words if word.isalpha()]
print(document_words[:10])
```

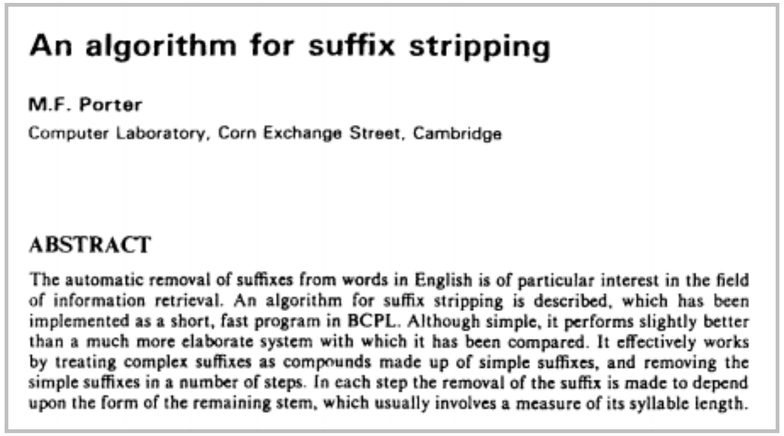
```
['nick', 'kyrgios', 'started', 'brisbane', 'open', 'title', 'defense', 'battling',
'victory', 'american']
```

Word stemming

- **Stemming** reduces words to their root form - it allows us to:
  - Treat different forms of the same word as one (i.e. “reads” and “reading” would both be stemmed to “read”)
  - Shrink the total number of unique terms, which reduces the noise in data and dimensionality of the data, which we will discuss shortly
- **PorterStemmer** is a Python implementation for the most famous stemming algorithm in existence - the Porter stemmer

Porter stemmer

- The algorithm is named after its inventor, computational linguist *Dr. Martin Porter*
- His paper **“An algorithm for suffix stripping”** is one of the most sited works in Information Retrieval (over **12,000** times according to Google Scholar)!



An algorithm for suffix stripping

M.F. Porter

Computer Laboratory, Corn Exchange Street, Cambridge

ABSTRACT

The automatic removal of suffixes from words in English is of particular interest in the field of information retrieval. An algorithm for suffix stripping is described, which has been implemented as a short, fast program in BCPL. Although simple, it performs slightly better than a much more elaborate system with which it has been compared. It effectively works by treating complex suffixes as compounds made up of simple suffixes, and removing the simple suffixes in a number of steps. In each step the removal of the suffix is made to depend upon the form of the remaining stem, which usually involves a measure of its syllable length.

Source: https://tartarus.org/martin/PorterStemmer/

Stem words

- The `PorterStemmer()` module of NLTK contains a method `.stem()`
- To stem a word, we would simply call `PorterStemmer().stem(word)` for every word in the `document_words` list using a **list comprehension**

```
# 4. Stem words.
document_words = [PorterStemmer().stem(word) for word in document_words]
print(document_words[:10])
```

```
['nick', 'kyrgio', 'start', 'brisban', 'open', 'titl', 'defens', 'battl', 'victori',
'american']
```

Module completion checklist

Objective	Complete
Load data and perform text cleaning on one document	✔
Perform text cleaning on entire corpus	

Clean the entire corpus

- Now that we have successfully implemented text cleaning steps on a single document, we can do that for the entire corpus of documents

```
# Create a list for clean documents.
df_clean = [None] * len(df_tokenized)
# Create a list of word counts for each clean document.
word_counts_per_document = [None] * len(df_tokenized)

# Process words in all documents.
for i in range(len(df_tokenized)):
    # 1. Convert to lowercase.
    df_clean[i] = [document.lower() for document in df_tokenized[i]]

    # 2. Remove stop words.
    df_clean[i] = [word for word in df_clean[i] if not word in stop_words]

    # 3. Remove punctuation and any non-alphabetical characters.
    df_clean[i] = [word for word in df_clean[i] if word.isalpha()]

    # 4. Stem words.
    df_clean[i] = [PorterStemmer().stem(word) for word in df_clean[i]]
```

Clean the entire corpus (cont'd)

```
# Convert word counts list and documents list to NumPy arrays.
word_counts_array = np.array(word_counts_per_document)
df_array = np.array(df_clean, dtype= object)

# Find indices of all documents where there are greater than or equal to 5 words.
valid_documents = np.where(word_counts_array >= 5)[0]

# Subset the df_array to keep only those where there are at least 5 words.
df_array = df_array[valid_documents]

# Convert the array back to a list.
df_clean = df_array.tolist()   #<- the processed documents

print(df_clean[0:2])
```

```

# Convert word counts list and documents list to NumPy arrays.
word_counts_array = np.array(word_counts_per_document)
df_array = np.array(df_clean, dtype= object)
```

Clean corpus

```
# Print the first 5 documents of clean corpus
for i in range(5):
  print(df_array[i])
```

```
['nick', 'kyrgio', 'start', 'brisban', 'open', 'titl', 'defens', 'battl', 'victori',
'american', 'ryan', 'harrison', 'open', 'round', 'tuesday']
['british', 'polic', 'confirm', 'tuesday', 'treat', 'stab', 'attack', 'injur', 'three',
'peopl', 'manchest', 'victoria', 'train', 'station', 'terrorist', 'investig', 'search',
'address', 'cheetham', 'hill', 'area', 'citi']
['marcellu', 'wiley', 'still', 'fenc', 'let', 'young', 'son', 'play', 'footbal', 'former',
'nfl', 'defens', 'end', 'fox', 'sport', 'person', 'tell', 'podcaston', 'sport', 'like',
'nfl', 'tri', 'make', 'footbal', 'safer', 'game', 'de']
['still', 'reckon', 'fallout', 'emmett', 'till', 'paint', 'chasten', 'artist', 'reveal',
'controversi', 'chang', 'even', 'move', 'forward', 'new', 'galleri', 'show']
['far', 'arik', 'ogunbowal', 'coach', 'muffet', 'mcgraw', 'concern', 'notr', 'dame',
'victori', 'louisvil', 'thursday', 'night', 'anoth', 'atlant', 'coast', 'confer', 'game',
'januari']
```

Knowledge check

Module completion checklist

Objective	Complete
Load data and perform text cleaning on one document	✔
Perform text cleaning on entire corpus	✔

Congratulations on completing this module!

You are now ready to try Tasks 1-5 in the Exercise for this topic