Discussion: Topic Modeling

Group Names and Roles

- david (driver)
- rashi (reviewer)
- lauren(proposer)

Intro

In this Discussion activity, we'll continue with with topic modeling. Recall that topic modeling can often be used to infer themes (or "topics") from sets of text data. Today, we will work through an example in which we download some data, prepare it appropriately, and deploy a topic model to obtain insights about the general themes present in the data.

Our data set for this activity consists of the texts of a number of Associated Press articles. It was originally collected by David Blei. I retrieved this data set here (https://github.com/tdhopper/topic-modeling-datasets/tree/master/data/lda-c/blei-ap).

Run the following code chunk to create a large string `s` containing the entire data set.

In [1]:
```
import urllib
def retrieve_text(url):
    """
    Retrieve text from the specified url and return
    it as a string
    """

    # grab the data and parse it
    filedata = urllib.request.urlopen(url)
    data = filedata.read()

    return(data.decode())

url = 'https://raw.githubusercontent.com/PhilChodrow/PIC16A/master/datasets/blei-ap.txt'
s = retrieve_text(url)
```

Part A

Inspect `s`. Don't print out the entire string; just take a look at the first 5,000 characters or so. Write a function which, when `s` is provided as input, will return a list of document texts. It should exclude the excess tags and other metadata. Call this function to create a new list variable called `texts`, where each element of `texts` is the complete text of one news story.

- ***Hint***: *First, split `s` on the newline character `"\n"`. Then, return a list of elements of the result with length longer than 100. This can be done with a for-loop, but a conditional list comprehension will be more compact*

The resulting list of news stories should have length 2226.

Comments and docstrings are not necessary for this function.

In [8]:
```
def myFunction(s):
    myString = s.split('\n')
    l = [i for i in myString if len(i) > 100]
    return l


texts = myFunction(s)
```

In [9]:
```
# check the length of the result
len(texts)
```

Out[9]:
```
2226
```

Part B

Create a `pandas` data frame called `df` with a single column called `text`, whose rows are the entries of `texts`. This data frame should have 2226 rows. Show your data frame to check that it looks ok.

In [10]:
```
import pandas as pd
```

In [11]:
```
df = pd.DataFrame({
    'text' : texts
})
```

In [12]:
```
df
```

Out[12]:

	text
0	A 16-year-old student at a private Baptist sc...
1	The Bechtel Group Inc. offered in 1985 to sel...
2	A gunman took a 74-year-old woman hostage aft...
3	Today is Saturday, Oct. 29, the 303rd day of ...
4	Cupid has a new message for lovers this Valen...
...	...
2221	The dollar rose in quiet European trading thi...
2222	Here are the companies known to be conducting...
2223	Bloodstains on a pillowcase and exercise bar ...
2224	When Ron Thompson sat down for lunch on New Y...
2225	A Navy anti-submarine helicopter crashed whil...

2226 rows × 1 columns

Part C

Create the term-document matrix. The group's **Reviewer** might want to check the lecture notes (https://nbviewer.jupyter.org/github/PhilChodrow/PIC16A/blob/master/content/NLP/NLP_2.ipynb) on topic modeling for some code to do this. Add the term-document matrix to `df`. Make sure that the columns are labeled with the relevant word.

I found that, for the purposes of the later parts of this exercise, the following arguments to `CountVectorizer` worked well:

```
max_df = 100, min_df=0.01, stop_words='english'
```

However, please feel free to experiment.

Call the new data frame with counts `big_df`.

In [16]:
```
import numpy as np
```

In [19]:
```
from sklearn.feature_extraction.text import CountVectorizer
vec = CountVectorizer(max_df = 100, min_df=0.01, stop_words='english')
counts = vec.fit_transform(df['text'])
counts = counts.toarray()
big_df = pd.DataFrame(counts, columns = vec.get_feature_names())
big_df = pd.concat((df, big_df), axis = 1)
big_df
```

Out[19]:

	text	00	06	07	110	120	125	130	150	152	...	wrote	yard	yards	yen	yes	yield	yields	younger	youth	youths
0	A 16-year-old student at a private Baptist sc...	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	1	0
1	The Bechtel Group Inc. offered in 1985 to sel...	0	0	0	0	0	0	0	0	0	...	0	0	0	0	2	0	0	0	0	0
2	A gunman took a 74-year-old woman hostage aft...	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	Today is Saturday, Oct. 29, the 303rd day of ...	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	Cupid has a new message for lovers this Valen...	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2221	The dollar rose in quiet European trading thi...	1	0	0	0	0	0	0	0	0	...	0	0	0	5	0	0	0	0	0	0
2222	Here are the companies known to be conducting...	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2223	Bloodstains on a pillowcase and exercise bar ...	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2224	When Ron Thompson sat down for lunch on New Y...	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2225	A Navy anti-submarine helicopter crashed whil...	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

2226 rows × 2542 columns

Part D

Create an input matrix `X` which is identical to `big_df` but drops the `text` column. Then, create a Nonnegative Matrix Factorization (NMF) model and fit it to `X`. Start with 10 components.

In [48]:
```
from sklearn.decomposition import NMF
from matplotlib import pyplot as plt


X = big_df.drop(['text'], axis = 1 )
model = NMF(n_components=10, init='random', random_state=0)
W = model.fit_transform(X)

'''
X = big_df.drop(['text'], axis = 1)
model = NMF(n_components = 10, init = "random", random_state = 0)
model.fit(X)
orders = np.argsort(model.components_, axis = 1)
important_words = np.array(X.columns)[orders]
weights = model.transform(X)
fig, ax = plt.subplots(1)
ax.imshow(weights.T)


#model.components_.shape
orders

'''
```

Out[48]:
```
'\nX = big_df.drop([\'text\'], axis = 1)\nmodel = NMF(n_components = 10, init = "random", random_state = 0)\nmodel.fit
(X)\n\n\norders = np.argsort(model.components_, axis = 1)\n\n\n\nimportant_words = np.array(X.columns)[orders]\n\n\nweig
hts = model.transform(X)\n\nfig, ax = plt.subplots(1)\n\nax.imshow(weights.T)\n\n\n#model.components_.shape\norders\n\n'
```

Part E

The following code (from lecture) will extract the top words within each topic. Run this code.

In [ ]:
```
import numpy as np
def top_words(X, model, component, num_words):
    """
    Extract the top words from the specified component
    for a topic model trained on data.
    X: a term-document matrix, assumed to be a pd.DataFrame
    model: a sklearn model with a components_ attribute, e.g. NMF
    component: the desired component, specified as an integer.
        Must be less than than the total number of components in model
    num_words: the number of words to return.
    """
    orders = np.argsort(model.components_, axis = 1)
    important_words = np.array(X.columns)[orders]
    return important_words[component][-num_words:]
```

Use this code to investigate the topics constructed by the model. Can you interpret any of them? Keep in mind that many of these news articles are from the 1980s and 1990s. That's before many of you were born, you whippersnappers!

I was able to find U.S. political party conventions; fluctuations in the price of oil; U.S. / Soviet tensions; and international finance news, among other things. Show the top words for a few different topics, and see whether any of them look interpretable to you.

In [ ]:
```
# show the top words for a topic
```

In [ ]:
```
# show the top words for a topic
```

In [ ]:
```
# show the top words for a topic
```

In [ ]:
```
# show the top words for a topic
```