ADS 509 Assignment 5.1: Topic Modeling

This notebook holds Assignment 5.1 for Module 5 in ADS 509, Applied Text Mining. Work through this notebook, writing code and answering questions where required.

In this assignment you will work with a categorical corpus that accompanies `nltk`. You will build the three types of topic models described in Chapter 8 of *Blueprints for Text Analytics using Python*: NMF, LSA, and LDA. You will compare these models to the true categories.

General Assignment Instructions

These instructions are included in every assignment, to remind you of the coding standards for the class. Feel free to delete this cell after reading it.

One sign of mature code is conforming to a style guide. We recommend the Google Python Style Guide. If you use a different style guide, please include a cell with a link.

Your code should be relatively easy-to-read, sensibly commented, and clean. Writing code is a messy process, so please be sure to edit your final submission. Remove any cells that are not needed or parts of cells that contain unnecessary code. Remove inessential `import` statements and make sure that all such statements are moved into the designated cell.

Make use of non-code cells for written commentary. These cells should be grammatical and clearly written. In some of these cells you will have questions to answer. The questions will be marked by a "Q:" and will have a corresponding "A:" spot for you. *Make sure to answer every question marked with a* `Q:` *for full credit.*

In [ ]:
```
#github
```

In [1]:
```
from nltk.corpus import brown

import numpy as np
import pandas as pd
from tqdm.auto import tqdm

import pyLDAvis
import pyLDAvis.lda_model
import pyLDAvis.gensim_models

import spacy
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.decomposition import NMF, TruncatedSVD, LatentDirichletAllocation

from spacy.lang.en.stop_words import STOP_WORDS as stopwords

from collections import Counter, defaultdict

nlp = spacy.load('en_core_web_sm')

import warnings
warnings.filterwarnings("ignore")
```

```
C:\Users\gauth\AppData\Roaming\Python\Python39\site-packages\pandas\core\arrays\masked.py:62: UserWarning: Pandas requires version '1.3.4' or newer of 'bottleneck' (version '1.3.2' currently installed).
  from pandas.core import (
```

In [2]:
```
 import sys
 !{sys.executable} -m pip install spacy
 print(sys.executable)
import nltk
 nltk.download('brown')
```

```
Requirement already satisfied: spacy in c:\users\gauth\miniconda3\lib\site-packages (3.6.1)
Requirement already satisfied: spacy-legacy<3.1.0,>=3.0.11 in c:\users\gauth\miniconda3\lib\site-packages (from spacy) (3.0.12)
Requirement already satisfied: spacy-loggers<2.0.0,>=1.0.0 in c:\users\gauth\miniconda3\lib\site-packages (from spacy) (1.0.5)
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in c:\users\gauth\miniconda3\lib\site-packages (from spacy) (1.0.10)
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in c:\users\gauth\miniconda3\lib\site-packages (from spacy) (2.0.8)
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in c:\users\gauth\miniconda3\lib\site-packages (from spacy) (3.0.9)
Requirement already satisfied: thinc<8.2.0,>=8.1.8 in c:\users\gauth\miniconda3\lib\site-packages (from spacy) (8.1.12)
Requirement already satisfied: wasabi<1.2.0,>=0.9.1 in c:\users\gauth\miniconda3\lib\site-packages (from spacy) (1.1.2)
Requirement already satisfied: srsly<3.0.0,>=2.4.3 in c:\users\gauth\miniconda3\lib\site-packages (from spacy) (2.4.7)
Requirement already satisfied: catalogue<2.1.0,>=2.0.6 in c:\users\gauth\miniconda3\lib\site-packages (from spacy) (2.0.9)
Requirement already satisfied: typer<0.10.0,>=0.3.0 in c:\users\gauth\miniconda3\lib\site-packages (from spacy) (0.9.0)
Requirement already satisfied: pathy>=0.10.0 in c:\users\gauth\miniconda3\lib\site-packages (from spacy) (0.10.2)
Requirement already satisfied: smart-open<7.0.0,>=5.2.1 in c:\users\gauth\miniconda3\lib\site-packages (from spacy) (6.4.0)
Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in c:\users\gauth\miniconda3\lib\site-packages (from spacy) (4.62.3)
Requirement already satisfied: numpy>=1.15.0 in c:\users\gauth\appdata\roaming\python\python39\site-packages (from spacy) (1.26.0)
Requirement already satisfied: requests<3.0.0,>=2.13.0 in c:\users\gauth\miniconda3\lib\site-packages (from spacy) (2.27.1)
Requirement already satisfied: pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.4 in c:\users\gauth\miniconda3\lib\site-packages (from spacy) (2.3.0)
Requirement already satisfied: jinja2 in c:\users\gauth\miniconda3\lib\site-packages (from spacy) (3.0.3)
Requirement already satisfied: setuptools in c:\users\gauth\miniconda3\lib\site-packages (from spacy) (58.0.4)
Requirement already satisfied: packaging>=20.0 in c:\users\gauth\miniconda3\lib\site-packages (from spacy) (21.3)
Requirement already satisfied: langcodes<4.0.0,>=3.2.0 in c:\users\gauth\miniconda3\lib\site-packages (from spacy) (3.3.0)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in c:\users\gauth\miniconda3\lib\site-packages (from packaging>=20.0->spacy) (3.0.7)
Requirement already satisfied: annotated-types>=0.4.0 in c:\users\gauth\miniconda3\lib\site-packages (from pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.4->spacy) (0.5.0)
Requirement already satisfied: pydantic-core==2.6.3 in c:\users\gauth\miniconda3\lib\site-packages (from pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.4->spacy) (2.6.3)
Requirement already satisfied: typing-extensions>=4.6.1 in c:\users\gauth\miniconda3\lib\site-packages (from pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.4->spacy) (4.8.0)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in c:\users\gauth\miniconda3\lib\site-packages (from requests<3.0.0,>=2.13.0->spacy) (1.26.7)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\gauth\miniconda3\lib\site-packages (from requests<3.0.0,>=2.13.0->spacy) (2021.10.8)
Requirement already satisfied: charset-normalizer~=2.0.0 in c:\users\gauth\miniconda3\lib\site-packages (from requests<3.0.0,>=2.13.0->spacy) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in c:\users\gauth\miniconda3\lib\site-packages (from requests<3.0.0,>=2.13.0->spacy) (3.3)
Requirement already satisfied: blis<0.8.0,>=0.7.8 in c:\users\gauth\miniconda3\lib\site-packages (from thinc<8.2.0,>=8.1.8->spacy) (0.7.10)
Requirement already satisfied: confection<1.0.0,>=0.0.1 in c:\users\gauth\miniconda3\lib\site-packages (from thinc<8.2.0,>=8.1.8->spacy) (0.1.3)
Requirement already satisfied: colorama in c:\users\gauth\miniconda3\lib\site-packages (from tqdm<5.0.0,>=4.38.0->spacy) (0.4.6)
Requirement already satisfied: click<9.0.0,>=7.1.1 in c:\users\gauth\miniconda3\lib\site-packages (from typer<0.10.0,>=0.3.0->spacy) (8.1.7)
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\gauth\miniconda3\lib\site-packages (from jinja2->spacy) (2.1.0)
C:\Users\gauth\miniconda3\python.exe
[nltk_data] Downloading package brown to
[nltk_data]     C:\Users\gauth\AppData\Roaming\nltk_data...
[nltk_data]   Package brown is already up-to-date!
```

Out[2]:
```
True
```

In [3]:
```
# This function comes from the BTAP repo.

def display_topics(model, features, no_top_words=5):
    for topic, words in enumerate(model.components_):
        total = words.sum()
        largest = words.argsort()[::-1] # invert sort order
        print("\nTopic %02d" % topic)
        for i in range(0, no_top_words):
            print("  %s (%2.2f)" % (features[largest[i]], abs(words[largest[i]]*100.0/total)))
```

Getting to Know the Brown Corpus

Let's spend a bit of time getting to know what's in the Brown corpus, our NLTK example of an "overlapping" corpus.

In [4]:
```
# categories of articles in Brown corpus
for category in brown.categories() :
    print(f"For {category} we have {len(brown.fileids(categories=category))} articles.")
```

```
For adventure we have 29 articles.
For belles_lettres we have 75 articles.
For editorial we have 27 articles.
For fiction we have 29 articles.
For government we have 30 articles.
For hobbies we have 36 articles.
For humor we have 9 articles.
For learned we have 80 articles.
For lore we have 48 articles.
For mystery we have 24 articles.
For news we have 44 articles.
For religion we have 17 articles.
For reviews we have 17 articles.
For romance we have 29 articles.
For science_fiction we have 6 articles.
```

Let's create a dataframe of the articles in of hobbies, editorial, government, news, and romance.

In [5]:
```
categories = ['editorial','government','news','romance','hobbies']

category_list = []
file_ids = []
texts = []

for category in categories :
    for file_id in brown.fileids(categories=category) :

        # build some lists for a dataframe
        category_list.append(category)
        file_ids.append(file_id)

        text = brown.words(fileids=file_id)
        texts.append(" ".join(text))



df = pd.DataFrame()
df['category'] = category_list
df['id'] = file_ids
df['text'] = texts

df.shape
```

Out[5]:
```
(166, 3)
```

In [6]:
```
df.head()
```

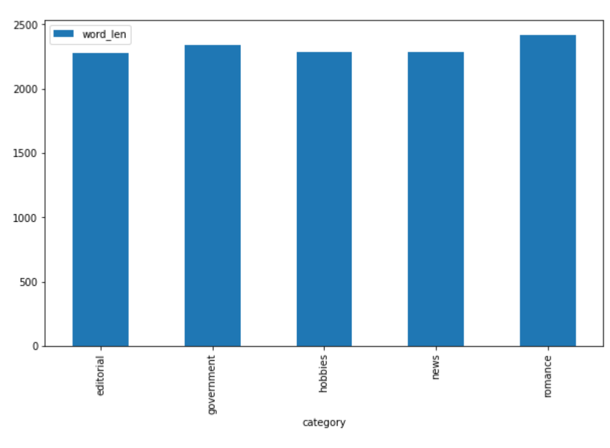
Out[6]:

	category	id	text
0	editorial	cb01	Assembly session brought much good The General...
1	editorial	cb02	Must Berlin remain divided ? ? The inference h...
2	editorial	cb03	A good man departs . Goodby , Mr. Sam . Sam Ra...
3	editorial	cb04	A shock wave from Africa Word of Dag Hammarskj...
4	editorial	cb05	Help when needed If the Dominican Republic ach...

In [7]:
```
# Let's add some helpful columns on the df
df['char_len'] = df['text'].apply(len)
df['word_len'] = df['text'].apply(lambda x: len(x.split()))
```

In [8]:
```
%matplotlib inline
df.groupby('category').agg({'word_len': 'mean'}).plot.bar(figsize=(10,6))
```

Out[8]:
```
<Axes: xlabel='category'>
```



Now do our TF-IDF and Count vectorizations.

In [9]:
```
count_text_vectorizer = CountVectorizer(stop_words=list(stopwords), min_df=5, max_df=0.7)
count_text_vectors = count_text_vectorizer.fit_transform(df["text"])
count_text_vectors.shape
```

Out[9]:
```
(166, 4941)
```

In [10]:
```
tfidf_text_vectorizer = TfidfVectorizer(stop_words=list(stopwords), min_df=5, max_df=0.7)
tfidf_text_vectors = tfidf_text_vectorizer.fit_transform(df['text'])
tfidf_text_vectors.shape
```

Out[10]:
```
(166, 4941)
```

Q: What do the two data frames `count_text_vectors` and `tfidf_text_vectors` hold?

A:Each cell in tfidf_text_vectors represents the TF-IDF value, whereas each cell in count_text_vectors reflects the term frequency (frequency of a term in a document). To gauge a term's significance in a document in relation to the full corpus, TF-IDF combines the term frequency and inverse document frequency.

Fitting a Non-Negative Matrix Factorization Model

In this section the code to fit a five-topic NMF model has already been written. This code comes directly from the BTAP repo, which will help you tremendously in the coming sections.

In [11]:
```
# fit a five-topic NMF model
nmf_text_model = NMF(n_components=5, random_state=7)
W_text_matrix = nmf_text_model.fit_transform(tfidf_text_vectors)
H_text_matrix = nmf_text_model.components_
```

In [12]:
```
display_topics(nmf_text_model, tfidf_text_vectorizer.get_feature_names_out())
```

```
Topic 00
  mr (0.51)
  president (0.45)
  kennedy (0.43)
  united (0.42)
  khrushchev (0.40)

Topic 01
  said (0.88)
  didn (0.46)
  ll (0.45)
  thought (0.42)
  man (0.37)

Topic 02
  state (0.40)
  development (0.36)
  tax (0.33)
  sales (0.30)
  program (0.25)

Topic 03
  mrs (2.61)
  mr (0.78)
  said (0.64)
  miss (0.52)
  car (0.51)

Topic 04
  game (1.01)
  league (0.74)
  ball (0.72)
  baseball (0.71)
  team (0.66)
```

Now some work for you to do. Compare the NMF factorization to the original categories from the Brown Corpus.

We are interested in the extent to which our NMF factorization agrees or disagrees with the original categories in the corpus. For each topic in your NMF model, tally the Brown categories and interpret the results.

In [13]:
```
# For each topic in your NMF model, tally the Brown categories
topic_to_category = defaultdict(lambda: defaultdict(int))

for idx, row in enumerate(W_text_matrix):
    topic = np.argmax(row)
    category = df['category'].iloc[idx]
    topic_to_category[topic][category] += 1
topic_to_category
```

Out[13]:
```
defaultdict(<function __main__.<lambda>()>,
            {2: defaultdict(int,
                         {'editorial': 2,
                          'government': 26,
                          'news': 11,
                          'hobbies': 26}),
             0: defaultdict(int,
                         {'editorial': 20, 'government': 4, 'news': 8}),
             1: defaultdict(int,
                         {'editorial': 4, 'romance': 29, 'hobbies': 8}),
             4: defaultdict(int, {'editorial': 1, 'news': 8, 'hobbies': 1}),
             3: defaultdict(int, {'news': 17, 'hobbies': 1})})
```

Q: How does your five-topic NMF model compare to the original Brown categories?

A: The five-topic NMF model shows some agreement with the original Brown categories. For example, topic 02, which is dominated by terms such as "state," "tax," "program," and "development," seems to align well with the 'government' category. Similarly, topic 00, with terms like "mr," "president," "kennedy," and "united," appears to fit with categories such as 'government,' 'news,' or 'editorial. However, topic 01 and 03 with term like "mrs," "miss," "car," and "said" or "said," "didn," "thought," and "man" are harder to intepret if it is 'romance', 'hobies', or 'news'.

Fitting an LSA Model

In this section, follow the example from the repository and fit an LSA model (called a "TruncatedSVD" in `sklearn`). Again fit a five-topic model and compare it to the actual categories in the Brown corpus. Use the TF-IDF vectors for your fit, as above.

To be explicit, we are once again interested in the extent to which this LSA factorization agrees or disagrees with the original categories in the corpus. For each topic in your model, tally the Brown categories and interpret the results.

In [14]:
```
# Fit the LSA model
lsa_text_model = TruncatedSVD(n_components=5, random_state=7)
lsa_text_matrix = lsa_text_model.fit_transform(tfidf_text_vectors)
```

In [15]:
```
# For each topic in your NMF model, tally the Brown categories
topic_to_category = defaultdict(lambda: defaultdict(int))

for idx, row in enumerate(lsa_text_matrix):
    topic = np.argmax(row)
    category = df['category'].iloc[idx]
    topic_to_category[topic][category] += 1
topic_to_category
```

Out[15]:
```
defaultdict(<function __main__.<lambda>()>,
            {0: defaultdict(int,
                         {'editorial': 27,
                          'government': 30,
                          'news': 34,
                          'romance': 21,
                          'hobbies': 36}),
             4: defaultdict(int, {'news': 7}),
             3: defaultdict(int, {'news': 3}),
             1: defaultdict(int, {'romance': 8})})
```

Q: How does your five-topic LSA model compare to the original Brown categories?

A: Topic 0 appears to predominate throughout all categories, whereas topics 4 and 3 are related to the "news" category in particular and topic 1 is related to the "romance" category. Topic 2 was absent, which indicates that it has little or no relationship to any of the categories.

In [16]:
```
# call display_topics on your model
display_topics(lsa_text_model, tfidf_text_vectorizer.get_feature_names_out())
```

```
Topic 00
  said (0.44)
  mr (0.25)
  mrs (0.22)
  state (0.20)
  man (0.17)

Topic 01
  said (3.89)
  ll (2.73)
  didn (2.63)
  thought (2.20)
  got (1.97)

Topic 02
  mrs (3.17)
  mr (1.69)
  said (1.06)
  kennedy (0.78)
  khrushchev (0.76)

Topic 03
  mrs (27.04)
  club (6.03)
  game (5.52)
  jr (5.17)
  university (4.72)

Topic 04
  game (4.12)
  league (2.95)
  baseball (2.94)
  ball (2.82)
  team (2.68)
```

Q: What is your interpretation of the display topics output?

A: Some interpretations can be formed based on the display topics generated by the LSA model, such as the following: topics 00, 01, and 02 are more like stories or discourse about people who are difficult to interpret, while topics 03 and 04 are probably about sports or hobbies. Topic 00 is about baseball or something sports-related.

Fitting an LDA Model

Finally, fit a five-topic LDA model using the count vectors (`count_text_vectors` from above). Display the results using `pyLDAvis.display` and describe what you learn from that visualization. t

In [17]:
```
# Fit the LDA model
lda_text_model = LatentDirichletAllocation(n_components=5, random_state=7)
lda_text_matrix = lda_text_model.fit_transform(count_text_vectors)
```

In [18]:
```
# Call `display_topics` on your fitted model here
display_topics(lda_text_model, tfidf_text_vectorizer.get_feature_names_out())
```

```
Topic 00
  use (0.59)
  water (0.45)
  work (0.38)
  good (0.37)
  high (0.35)

Topic 01
  said (1.41)
  man (0.58)
  old (0.53)
  little (0.49)
  know (0.43)

Topic 02
  said (0.65)
  state (0.61)
  president (0.55)
  tax (0.43)
  city (0.35)

Topic 03
  mrs (1.52)
  mr (0.53)
  clay (0.44)
  work (0.39)
  student (0.38)

Topic 04
  state (1.41)
  states (1.32)
  united (1.23)
  government (0.89)
  shall (0.71)
```

In [19]:
```
# For each topic in your NMF model, tally the Brown categories
topic_to_category = defaultdict(lambda: defaultdict(int))

for idx, row in enumerate(lda_text_matrix):
    topic = np.argmax(row)
    category = df['category'].iloc[idx]
    topic_to_category[topic][category] += 1
topic_to_category
```

Out[19]:
```
defaultdict(<function __main__.<lambda>()>,
            {2: defaultdict(int,
                         {'editorial': 13,
                          'government': 14,
                          'news': 28,
                          'hobbies': 3}),
             1: defaultdict(int,
                         {'editorial': 11,
                          'government': 1,
                          'news': 6,
                          'romance': 29,
                          'hobbies': 5}),
             3: defaultdict(int,
                         {'editorial': 1,
                          'government': 2,
                          'news': 9,
                          'hobbies': 8}),
             4: defaultdict(int,
                         {'editorial': 2,
                          'government': 8,
                          'news': 1,
                          'hobbies': 1}),
             0: defaultdict(int, {'government': 5, 'hobbies': 19})})
```

Q: What inference do you draw from the displayed topics for your LDA model?

A: Topic 00 seems to focus on the use of water, work, and quality. Topic 01 may be discussion about a man, and his age. Topic 02 involves discussions related to the state, president, taxes, and cities. Topic 03 focuses on individuals such as Mrs., Mr., work, and students. Topic 04 is about the state, United States, government.

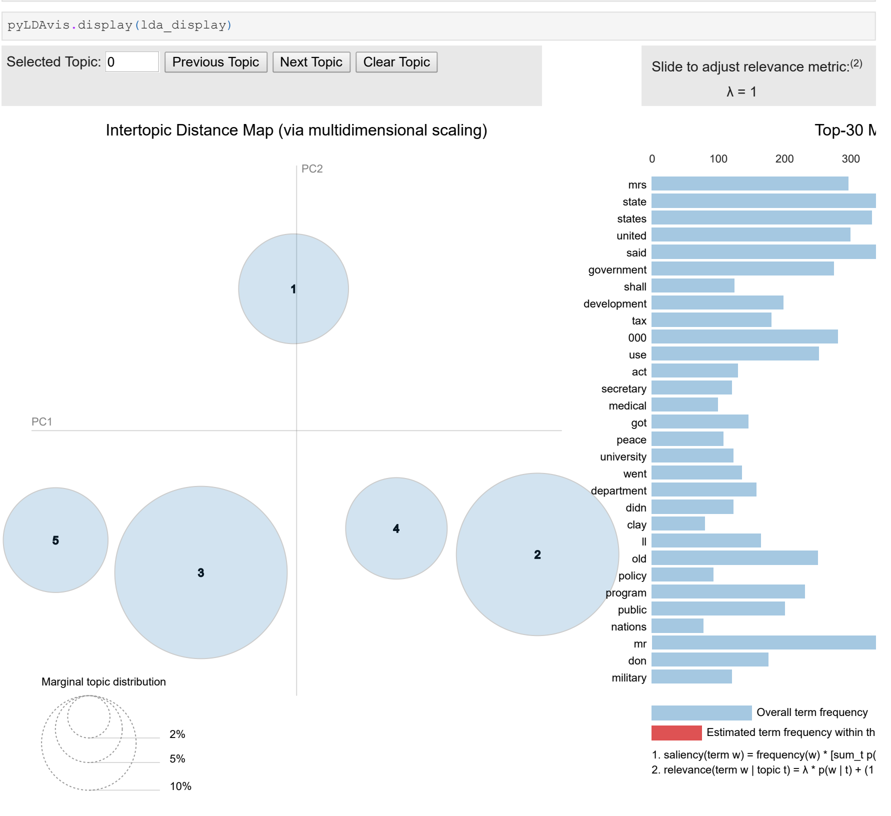
Q: Repeat the tallying of Brown categories within your topics. How does your five-topic LDA model compare to the original Brown categories?

A: Mosts topics from LDA model contains different Brown categories. Topic 2 and 4 seems most aligned to the ctergory of government, editorials and news. Other topics are hareder to compare.

In [20]:
```
lda_display = pyLDAvis.lda_model.prepare(lda_text_model, count_text_vectors, count_text_vectorizer, sort_topics
```

In [21]:
```
pyLDAvis.display(lda_display)
```

Out[21]:



Q: What conclusions do you draw from the visualization above? Please address the principal component scatterplot and the salient terms graph.

A: Topics 3 and 2 appear to have the largest representation in the dataset, according to the principal component scatterplot, indicating that these two topics are more frequently discussed than other topics. Additionally, it can be seen that 3 and 5 share the most similarities, along with 2 and 4. Topic 1 appears to be the most distinctive from other topics, however. Additionally, the salient terms graph highlights the most significant and common terms for each topic within the corpus. Finding the main ideas and concepts related to each issue is helpful. These graphs offer excellent details and insightful information.

In [ ]: