Topic Modeling

June 11, 2025

1 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.

1.1 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.*

[65]:
```
# These libraries may be useful to you

#!pip install pyLDAvis==3.4.1 --user  #You need to restart the Kernel after␣
 ↪installation.
# You also need a Python version => 3.9.0
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')
```

[66]:
```
# add any additional libaries you need here

import matplotlib
```

[67]:
```
# 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)))
```

1.2 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.

[68]:
```
# 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.

```
[69]: 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
```

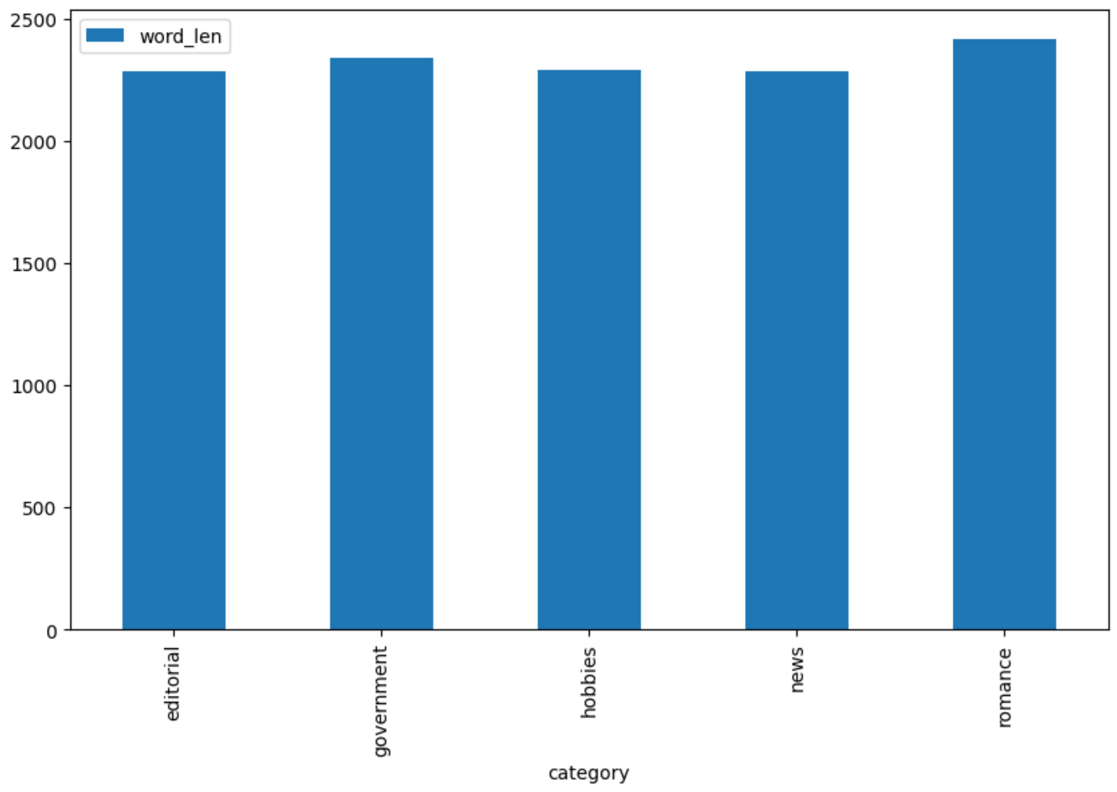
```
[69]: (166, 3)
```

```
[70]: # 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()))
```

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

```
/Users/sahilwadhwa/ads509env/lib/python3.11/site-
packages/IPython/core/pylabtools.py:77: DeprecationWarning: backends is
deprecated since IPython 8.24, backends are managed in matplotlib and can be
externally registered.
  warnings.warn(
```

```
[71]: <Axes: xlabel='category'>
```



Now do our TF-IDF and Count vectorizations.

```
[72]: 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
```

```
/Users/sahilwadhwa/ads509env/lib/python3.11/site-
packages/sklearn/feature_extraction/text.py:402: UserWarning: Your stop_words
may be inconsistent with your preprocessing. Tokenizing the stop words generated
tokens ['ll', 've'] not in stop_words.
  warnings.warn(
```

```
[72]: (166, 4941)
```

```
[73]: 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
```

```
[73]: (166, 4941)
```

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

A: The count_text_vectors data frame holds a document-term matrix where the values are how many times each word appears in each document. The tfidf_text_vectors data frame holds a document-term matrix where the values are TF-IDF scores.

1.3 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.

[74]:
```
nmf_text_model = NMF(n_components=5, random_state=314)
W_text_matrix = nmf_text_model.fit_transform(tfidf_text_vectors)
H_text_matrix = nmf_text_model.components_
```

[75]:
```
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 (0.39)
  development (0.36)
  tax (0.33)
  sales (0.30)
  program (0.25)

Topic 03
  mrs (2.61)
  mr (0.78)
  said (0.63)
  miss (0.52)
  car (0.51)

Topic 04
  game (1.02)
  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.

[76]:
```
# Your code here
df['nmf_topic'] = W_text_matrix.argmax(axis=1)
nmf_counts = df.groupby('nmf_topic')['category'].value_counts().
  ↪unstack(fill_value = 0)

print("Distribution per topic (NMF)")
display(nmf_counts)
```

```
Distribution per topic (NMF)
```

category nmf_topic	editorial	government	hobbies	news	romance
0	20	4	0	8	0
1	4	0	8	0	29
2	2	26	26	11	0
3	0	0	1	17	0
4	1	0	1	8	0

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

A: The five-topic NMF model has some simalarities to the original Brown categories but its not exactly the same

1.4 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.

[77]:
```
# Your code here

lsa_model = TruncatedSVD(n_components=5, random_state=314)
W_lsa_matrix = lsa_model.fit_transform(tfidf_text_vectors)
H_lsa_matrix = lsa_model.components_

df['lsa_topic'] = W_lsa_matrix.argmax(axis=1)
```

```
lsa_counts = df.groupby('lsa_topic')['category'].value_counts().
 ↪unstack(fill_value = 0)

print("Distribution per topic (LSA)")
display(lsa_counts)
```

```
Distribution per topic (LSA)

category   editorial  government  hobbies  news  romance
lsa_topic
0                 27          30       36    34       21
1                  0           0        0     0        8
3                  0           0        0     3        0
4                  0           0        0     7        0
```

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

A: The five-topic NMF model has some simalarities to the original Brown categories but its not exactly the same

[78]:
```
# call display_topics on your model
display_topics(lsa_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 (3.12)
  mr (1.70)
  said (1.06)
  kennedy (0.82)
  khrushchev (0.77)

Topic 03
  mrs (29.45)
  club (6.53)
  game (6.12)
  jr (5.60)
  university (5.20)

Topic 04
  game (4.54)
  league (3.27)
  baseball (3.22)
  ball (3.10)
  team (2.94)
```

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

A: My interpretation of the display topics output is that each topic has some commonalities among the words

1.5 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.

[79]:
```
# Fit your LDA model here
lda_text_model = LatentDirichletAllocation(n_components = 5, random_state =␣
↪314, learning_method = 'online')
lda_text_model.fit(count_text_vectors)
```

[79]:
```
LatentDirichletAllocation(learning_method='online', n_components=5,
                          random_state=314)
```

[80]:
```
# Call `display_topics` on your fitted model here
display_topics(lda_text_model, count_text_vectorizer.get_feature_names_out())
```

```
Topic 00
  mrs (4.40)
  mr (0.99)
  miss (0.59)
  jr (0.51)
  daughter (0.39)

Topic 01
  use (0.42)
  water (0.33)
  development (0.33)
  work (0.32)
  good (0.30)

Topic 02
  clay (0.70)
  world (0.54)
  old (0.48)
  left (0.36)
  place (0.34)

Topic 03
  said (0.83)
  president (0.63)
  state (0.62)
  mr (0.58)
  government (0.38)

Topic 04
  said (0.74)
  day (0.47)
  good (0.43)
  feed (0.42)
  little (0.40)
```

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

A: The inference I draw from the displayed topics for my LDA model is that there is some themes in each topic grouping but some topics have clearer themes than others.

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

A: My five-topic LDA model doesn't compare that well to the original Brown categories as many of the topics contain mixed categories.

[81]:
```python
lda_transform = lda_model.transform(count_text_vectors)
df['lda_topic'] = lda_transform.argmax(axis=1)
lda_counts = df.groupby('lda_topic')['category'].value_counts().
 ↪unstack(fill_value = 0)

print("Distribution per topic (LDA)")
display(lda_counts)
```

```
Distribution per topic (LDA)

category   editorial  government  hobbies  news  romance
lda_topic
0                  0           0        0     2        0
1                  0          12       18     2        0
2                  2           1        7    10        1
3                 20          10        1    28        0
4                  5           7       10     2       28
```

[82]:
```python
lda_display = pyLDAvis.lda_model.prepare(lda_text_model, count_text_vectors,␣
 ↪count_text_vectorizer, sort_topics=False)
```

[83]:
```python
pyLDAvis.display(lda_display)
```

```
[83]: <IPython.core.display.HTML object>
```

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

A: The conclusion from the scatterplot is that each circle represents a topic and the size of each circle represents the prevalence of the topic. The salient terms graph represents the overall term frequency for each word but more importandly shows the most important words for each selected topic.