Topic Models

June 9, 2025

1 Assignment 5.1: Topic Modeling

Course: ADS 509 - Applied Text Mining
Assignment: Topic Modeling with NMF, LSA, and LDA
Student: Gabriel Elohi
Date: $(date)

1.1 Overview

In this assignment, we will build and compare three different topic modeling approaches: 1. **NMF (Non-negative Matrix Factorization)** model 2. **LSA (Latent Semantic Analysis)** model 3. **LDA (Latent Dirichlet Allocation)** model

We will work with the Brown University corpus from NLTK and compare the resulting topic allocations with the official document classifications.

1.2 AI Tool Attribution

If any AI tools (ChatGPT, Gemini, GitHub Copilot, etc.) are used in this assignment, they will be explicitly disclosed and cited here with explanations of their contributions.

1.3 1. Setup and Data Exploration

Let's start by importing the necessary libraries and exploring the Brown corpus.

[1]:
```
# Import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from collections import Counter, defaultdict
import warnings
warnings.filterwarnings('ignore')

# NLTK imports
import nltk
from nltk.corpus import brown, stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
```

```
from nltk.tag import pos_tag

# Scikit-learn imports for topic modeling
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.decomposition import NMF, TruncatedSVD, LatentDirichletAllocation
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.preprocessing import normalize

# Download required NLTK data
nltk.download('brown', quiet=True)
nltk.download('stopwords', quiet=True)
nltk.download('punkt', quiet=True)
nltk.download('wordnet', quiet=True)
nltk.download('averaged_perceptron_tagger', quiet=True)

print("Libraries imported successfully!")
```

```
Libraries imported successfully!
```

1.3.1 1.1 Exploring the Brown Corpus

[2]:
```
# Explore the Brown corpus structure
print("Brown Corpus Categories:")
categories = brown.categories()
print(f"Number of categories: {len(categories)}")
print(f"Categories: {categories}")

print("\nDocument counts per category:")
for category in categories:
    file_count = len(brown.fileids(categories=category))
    print(f"{category}: {file_count} documents")
```

```
Brown Corpus Categories:
Number of categories: 15
Categories: ['adventure', 'belles_lettres', 'editorial', 'fiction',
'government', 'hobbies', 'humor', 'learned', 'lore', 'mystery', 'news',
'religion', 'reviews', 'romance', 'science_fiction']

Document counts per category:
adventure: 29 documents
belles_lettres: 75 documents
editorial: 27 documents
fiction: 29 documents
government: 30 documents
hobbies: 36 documents
humor: 9 documents
learned: 80 documents
lore: 48 documents
```

```
mystery: 24 documents
news: 44 documents
religion: 17 documents
reviews: 17 documents
romance: 29 documents
science_fiction: 6 documents
```

[3]:
```
# Get sample documents from different categories
print("Sample documents from different categories:")
for i, category in enumerate(categories[:3]):
    file_id = brown.fileids(categories=category)[0]
    sample_text = ' '.join(brown.words(file_id)[:50])
    print(f"\n{category.upper()} - {file_id}:")
    print(sample_text + "...")
```

```
Sample documents from different categories:

ADVENTURE - cn01:
Dan Morgan told himself he would forget Ann Turner . He was well rid of her . He
certainly didn't want a wife who was fickle as Ann . If he had married her ,
he'd have been asking for trouble . But all of this was rationalization .
Sometimes…

BELLES_LETTRES - cg01:
Northern liberals are the chief supporters of civil rights and of integration .
They have also led the nation in the direction of a welfare state . And both in
their objectives of non-discrimination and of social progress they have had
ranged against them the Southerners who are called Bourbons…

EDITORIAL - cb01:
Assembly session brought much good The General Assembly , which adjourns today ,
has performed in an atmosphere of crisis and struggle from the day it convened .
It was faced immediately with a showdown on the schools , an issue which was met
squarely in conjunction with the governor…
```

1.3.2 1.2 Data Preprocessing

[4]:
```
# Initialize preprocessing tools
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

def preprocess_text(text):
    """
    Preprocess text by tokenizing, removing stopwords, and lemmatizing.
    """
    # Convert to lowercase and tokenize
    tokens = word_tokenize(text.lower())
```

```
    # Remove non-alphabetic tokens and stopwords
    tokens = [token for token in tokens if token.isalpha() and token not in
 ↪stop_words]

    # Lemmatize tokens
    tokens = [lemmatizer.lemmatize(token) for token in tokens]

    # Filter out very short tokens
    tokens = [token for token in tokens if len(token) > 2]

    return ' '.join(tokens)

print("Preprocessing function defined.")
```

```
Preprocessing function defined.
```

[5]:
```
# Prepare the corpus for topic modeling
print("Preparing corpus for topic modeling...")

documents = []
document_categories = []
document_ids = []

# Process documents from each category
for category in categories:
    file_ids = brown.fileids(categories=category)
    for file_id in file_ids:
        # Get raw text
        raw_text = ' '.join(brown.words(file_id))

        # Preprocess text
        processed_text = preprocess_text(raw_text)

        # Only include documents with sufficient content
        if len(processed_text.split()) > 50:
            documents.append(processed_text)
            document_categories.append(category)
            document_ids.append(file_id)

print(f"Total documents prepared: {len(documents)}")
print(f"Average document length: {np.mean([len(doc.split()) for doc in
 ↪documents]):.1f} words")
```

```
Preparing corpus for topic modeling…
Total documents prepared: 500
Average document length: 1021.7 words
```

1.4 2. NMF (Non-negative Matrix Factorization) Topic Model

Let's start with building an NMF model for topic modeling.

[6]:
```
# Create TF-IDF vectorizer for NMF
print("Creating TF-IDF matrix for NMF...")

# Parameters for vectorization
max_features = 1000  # Limit vocabulary size
min_df = 2  # Ignore terms that appear in less than 2 documents
max_df = 0.8  # Ignore terms that appear in more than 80% of documents

tfidf_vectorizer = TfidfVectorizer(
    max_features=max_features,
    min_df=min_df,
    max_df=max_df,
    ngram_range=(1, 2),  # Include unigrams and bigrams
    stop_words='english'
)

tfidf_matrix = tfidf_vectorizer.fit_transform(documents)
feature_names = tfidf_vectorizer.get_feature_names_out()

print(f"TF-IDF matrix shape: {tfidf_matrix.shape}")
print(f"Vocabulary size: {len(feature_names)}")
```

```
Creating TF-IDF matrix for NMF…
TF-IDF matrix shape: (500, 1000)
Vocabulary size: 1000
```

[7]:
```
# Check scikit-learn version for NMF parameter compatibility
import sklearn
print(f"Scikit-learn version: {sklearn.__version__}")

# Note: NMF parameters changed in different sklearn versions:
# - Older versions used 'alpha' parameter
# - Newer versions use 'alpha_W' and 'alpha_H' parameters
# We'll use basic parameters for maximum compatibility
```

```
Scikit-learn version: 1.6.1
```

[8]:
```
# Fit NMF model
print("Fitting NMF model...")

n_topics = 10  # Number of topics to extract
random_state = 42

# Create NMF model with compatible parameters
nmf_model = NMF(
```

```
    n_components=n_topics,
    random_state=random_state,
    init='nndsvd',  # Non-negative double SVD initialization
    solver='cd',  # Coordinate descent solver
    max_iter=200
)

nmf_topics = nmf_model.fit_transform(tfidf_matrix)

print(f"NMF model fitted successfully!")
print(f"Document-topic matrix shape: {nmf_topics.shape}")
```

```
Fitting NMF model…
NMF model fitted successfully!
Document-topic matrix shape: (500, 10)
```

[9]:

```
# Display top words for each NMF topic
def display_topics(model, feature_names, n_top_words=10, model_name="Model"):
    """
    Display the top words for each topic.
    """
    print(f"\n=== {model_name} Topics ===")

    for topic_idx, topic in enumerate(model.components_):
        top_words_idx = topic.argsort()[-n_top_words:][::-1]
        top_words = [feature_names[i] for i in top_words_idx]
        top_weights = [topic[i] for i in top_words_idx]

        print(f"\nTopic {topic_idx + 1}:")
        for word, weight in zip(top_words, top_weights):
            print(f"  {word}: {weight:.3f}")

# Display NMF topics
display_topics(nmf_model, feature_names, n_top_words=8, model_name="NMF")
```

```

=== NMF Topics ===

Topic 1:
  said: 1.231
  like: 0.477
  got: 0.395
  know: 0.384
  woman: 0.352
  went: 0.343
  room: 0.341
  thought: 0.337
```

```
Topic 2:
  state: 0.450
  nation: 0.422
  united: 0.370
  war: 0.366
  soviet: 0.357
  american: 0.332
  government: 0.323
  united state: 0.298

Topic 3:
  human: 0.264
  man: 0.245
  life: 0.241
  experience: 0.231
  literature: 0.217
  idea: 0.204
  form: 0.203
  century: 0.203

Topic 4:
  temperature: 0.381
  surface: 0.353
  used: 0.285
  cell: 0.274
  data: 0.237
  pressure: 0.231
  material: 0.230
  fig: 0.222

Topic 5:
  school: 0.845
  student: 0.610
  college: 0.593
  child: 0.407
  education: 0.320
  teacher: 0.305
  university: 0.278
  class: 0.186

Topic 6:
  music: 0.495
  song: 0.340
  new: 0.303
  performance: 0.266
  new york: 0.258
  york: 0.254
  miss: 0.254
```

```
  jazz: 0.254

Topic 7:
  church: 1.098
  god: 0.490
  christian: 0.388
  catholic: 0.298
  christ: 0.293
  john: 0.218
  member: 0.184
  new: 0.158

Topic 8:
  man: 0.383
  men: 0.354
  horse: 0.321
  water: 0.309
  head: 0.298
  foot: 0.293
  eye: 0.292
  gun: 0.290

Topic 9:
  cost: 0.348
  company: 0.325
  sale: 0.310
  industry: 0.279
  program: 0.250
  plant: 0.249
  market: 0.246
  tax: 0.241

Topic 10:
  president: 0.698
  kennedy: 0.458
  said: 0.416
  state: 0.360
  committee: 0.344
  house: 0.326
  election: 0.280
  party: 0.261
```

1.4.1 2.1 NMF Topic Interpretation

Analysis of NMF Topics:

[Add your interpretation of the NMF topics here. Discuss what themes or subjects each topic seems to represent based on the top words.]

1.5 3. LSA (Latent Semantic Analysis) Topic Model

Now let's build an LSA model using Truncated SVD.

[10]:
```
# Fit LSA model using TruncatedSVD
print("Fitting LSA model...")

lsa_model = TruncatedSVD(
    n_components=n_topics,
    random_state=random_state,
    algorithm='randomized'
)

lsa_topics = lsa_model.fit_transform(tfidf_matrix)

print(f"LSA model fitted successfully!")
print(f"Document-topic matrix shape: {lsa_topics.shape}")
print(f"Explained variance ratio: {lsa_model.explained_variance_ratio_.sum():.
↪3f}")
```

```
Fitting LSA model…
LSA model fitted successfully!
Document-topic matrix shape: (500, 10)
Explained variance ratio: 0.155
```

[11]:
```
# Display LSA topics
display_topics(lsa_model, feature_names, n_top_words=8, model_name="LSA")
```

```
=== LSA Topics ===

Topic 1:
  said: 0.219
  man: 0.135
  like: 0.128
  new: 0.112
  year: 0.110
  day: 0.097
  state: 0.092
  house: 0.089

Topic 2:
  said: 0.235
  like: 0.122
  eye: 0.109
  got: 0.106
  door: 0.103
  looked: 0.095
  knew: 0.090
```

```
  went: 0.088

Topic 3:
  said: 0.240
  state: 0.217
  president: 0.165
  government: 0.108
  kennedy: 0.106
  united: 0.101
  house: 0.099
  tax: 0.096

Topic 4:
  church: 0.213
  god: 0.135
  student: 0.106
  school: 0.106
  college: 0.103
  christian: 0.101
  world: 0.093
  life: 0.091

Topic 5:
  school: 0.328
  student: 0.222
  college: 0.206
  child: 0.164
  miss: 0.155
  university: 0.121
  education: 0.108
  year: 0.107

Topic 6:
  church: 0.248
  said: 0.237
  school: 0.124
  christian: 0.112
  social: 0.110
  god: 0.107
  catholic: 0.102
  law: 0.099

Topic 7:
  church: 0.531
  god: 0.232
  john: 0.170
  christian: 0.159
  christ: 0.146
```

```
  river: 0.124
  catholic: 0.122
  water: 0.109

Topic 8:
  school: 0.266
  college: 0.250
  student: 0.249
  child: 0.159
  teacher: 0.132
  education: 0.115
  war: 0.105
  president: 0.105

Topic 9:
  church: 0.234
  president: 0.185
  kennedy: 0.165
  soviet: 0.120
  game: 0.119
  miss: 0.102
  room: 0.098
  music: 0.096

Topic 10:
  president: 0.198
  cell: 0.178
  john: 0.133
  kennedy: 0.133
  temperature: 0.129
  trial: 0.114
  fig: 0.111
  house: 0.109
```

1.5.1 3.1 LSA Topic Interpretation

Analysis of LSA Topics:

[Add your interpretation of the LSA topics here. Compare with NMF results and discuss similarities/differences.]

1.6 4. LDA (Latent Dirichlet Allocation) Topic Model

Finally, let's build an LDA model. LDA works better with count data rather than TF-IDF.

[12]:
```python
# Create count vectorizer for LDA
print("Creating count matrix for LDA...")

count_vectorizer = CountVectorizer(
```

```
    max_features=max_features,
    min_df=min_df,
    max_df=max_df,
    ngram_range=(1, 1),  # Only unigrams for LDA
    stop_words='english'
)

count_matrix = count_vectorizer.fit_transform(documents)
count_feature_names = count_vectorizer.get_feature_names_out()

print(f"Count matrix shape: {count_matrix.shape}")
print(f"Vocabulary size: {len(count_feature_names)}")
```

```
Creating count matrix for LDA…
Count matrix shape: (500, 1000)
Vocabulary size: 1000
```

[13]:
```
# Note: LDA parameter names in scikit-learn:
# - doc_topic_prior (equivalent to alpha in other LDA implementations)
# - topic_word_prior (equivalent to beta in other LDA implementations)
print("Setting up LDA model with scikit-learn parameter names...")
```

```
Setting up LDA model with scikit-learn parameter names…
```

[14]:
```
# Fit LDA model
print("Fitting LDA model...")

# Create LDA model with correct parameter names
lda_model = LatentDirichletAllocation(
    n_components=n_topics,
    random_state=random_state,
    doc_topic_prior=0.1,  # Document-topic concentration (alpha)
    topic_word_prior=0.01,  # Topic-word concentration (beta)
    max_iter=100,
    learning_method='batch'
)

lda_topics = lda_model.fit_transform(count_matrix)

print(f"LDA model fitted successfully!")
print(f"Document-topic matrix shape: {lda_topics.shape}")
print(f"Log likelihood: {lda_model.score(count_matrix):.2f}")
```

```
Fitting LDA model…
LDA model fitted successfully!
Document-topic matrix shape: (500, 10)
Log likelihood: -1349398.43
```

[15]:
```python
# Display LDA topics
display_topics(lda_model, count_feature_names, n_top_words=8, model_name="LDA")
```

```
=== LDA Topics ===

Topic 1:
  new: 397.064
  city: 310.589
  south: 219.396
  year: 204.915
  town: 179.862
  day: 173.985
  north: 161.176
  old: 141.456

Topic 2:
  social: 310.731
  law: 284.947
  state: 282.828
  people: 269.414
  community: 215.618
  fact: 210.616
  policy: 210.547
  question: 194.406

Topic 3:
  number: 317.757
  point: 218.305
  line: 183.520
  form: 183.381
  value: 178.789
  data: 157.376
  information: 156.605
  used: 149.535

Topic 4:
  said: 1539.817
  like: 973.071
  man: 772.658
  know: 565.163
  hand: 536.752
  little: 526.372
  day: 507.702
  came: 480.362

Topic 5:
  life: 337.321
```

```
  work: 334.181
  new: 301.967
  world: 258.195
  great: 233.810
  say: 229.262
  man: 228.720
  book: 224.913

Topic 6:
  school: 630.635
  child: 398.718
  student: 311.352
  year: 309.762
  college: 298.311
  university: 218.312
  program: 184.637
  education: 183.863

Topic 7:
  state: 644.998
  year: 474.620
  cost: 357.573
  program: 278.977
  tax: 246.010
  business: 244.003
  service: 228.834
  development: 227.840

Topic 8:
  church: 428.934
  god: 299.323
  life: 158.465
  john: 157.679
  new: 148.876
  christian: 146.015
  man: 139.350
  religious: 119.690

Topic 9:
  state: 415.560
  president: 360.202
  new: 322.374
  war: 321.515
  american: 310.930
  united: 268.857
  said: 246.264
  nation: 225.687
```

```
Topic 10:
  used: 249.765
  surface: 210.217
  water: 182.212
  material: 167.622
  area: 162.748
  use: 161.222
  temperature: 161.010
  high: 126.234
```

1.6.1 4.1 LDA Topic Interpretation

Analysis of LDA Topics:

[Add your interpretation of the LDA topics here. Compare with NMF and LSA results.]

1.7 5. Model Comparison and Analysis

Let's compare the three topic modeling approaches and analyze their performance.

[16]:
```python
# Create a comparison of topic assignments
def get_dominant_topic(doc_topic_matrix):
    """
    Get the dominant topic for each document.
    """
    return np.argmax(doc_topic_matrix, axis=1)

# Get dominant topics for each model
nmf_dominant_topics = get_dominant_topic(nmf_topics)
lsa_dominant_topics = get_dominant_topic(lsa_topics)
lda_dominant_topics = get_dominant_topic(lda_topics)

# Create comparison DataFrame
comparison_df = pd.DataFrame({
    'document_id': document_ids,
    'true_category': document_categories,
    'nmf_topic': nmf_dominant_topics,
    'lsa_topic': lsa_dominant_topics,
    'lda_topic': lda_dominant_topics
})

print("Topic assignment comparison:")
print(comparison_df.head(10))
```

```
Topic assignment comparison:
  document_id true_category  nmf_topic  lsa_topic  lda_topic
0        cn01     adventure          0          0          3
1        cn02     adventure          7          0          3
2        cn03     adventure          7          0          3
```

```
3         cn04     adventure          7           0           3
4         cn05     adventure          0           0           3
5         cn06     adventure          0           0           3
6         cn07     adventure          7           0           3
7         cn08     adventure          7           0           3
8         cn09     adventure          0           0           3
9         cn10     adventure          0           0           3
```

[17]:

```python
# Analyze topic distribution by true categories
print("\nTopic distribution analysis:")

for model_name, topic_col in [('NMF', 'nmf_topic'), ('LSA', 'lsa_topic'),␣
 ↪('LDA', 'lda_topic')]:
    print(f"\n{model_name} Topic Distribution by Category:")
    topic_category_crosstab = pd.crosstab(comparison_df['true_category'],␣
 ↪comparison_df[topic_col])
    print(topic_category_crosstab)
```

```
Topic distribution analysis:

NMF Topic Distribution by Category:
nmf_topic         0   1   2   3  4   5   6   7   8   9
true_category
adventure        12   0   0   0  0   0   0  17   0   0
belles_lettres    5  10  31   1  5  10   2   6   0   5
editorial         0  16   0   0  2   3   3   1   1   1
fiction          14   0   0   0  0   2   3  10   0   0
government        0   7   1   3  1   0   0   0  16   2
hobbies           0   1   0  12  4   5   0   4  10   0
humor             5   0   0   0  0   3   0   0   0   1
learned           0   4  17  38  6   2   0   1   9   3
lore              1   5   9   4  7   4   4   7   4   3
mystery          18   0   0   0  0   0   0   6   0   0
news              0   4   0   0  7  11   2   1   7  12
religion          0   0   5   0  0   0  11   1   0   0
reviews           0   2   1   1  0  13   0   0   0   0
romance          22   0   0   0  1   0   1   4   1   0
science_fiction   3   0   2   0  0   0   1   0   0   0

LSA Topic Distribution by Category:
lsa_topic         0  2  4  5  6  7   9
true_category
adventure        29  0  0  0  0  0   0
belles_lettres   74  0  0  0  0  1   0
editorial        27  0  0  0  0  0   0
fiction          27  0  0  0  2  0   0
government       25  3  1  0  0  0   1
```

```
hobbies          34  0  1  0  0  0   1
humor             9  0  0  0  0  0   0
learned          62  1  4  2  0  1  10
lore             41  0  2  1  3  1   0
mystery          24  0  0  0  0  0   0
news             43  1  0  0  0  0   0
religion         14  0  0  0  3  0   0
reviews          17  0  0  0  0  0   0
romance          29  0  0  0  0  0   0
science_fiction   6  0  0  0  0  0   0

LDA Topic Distribution by Category:
lda_topic         0   1   2   3   4  5   6  7   8   9
true_category
adventure         0   0   0  29   0  0   0  0   0   0
belles_lettres    3  16   2  15  30  0   0  0   9   0
editorial         0   0   0   3   4  0   1  0  19   0
fiction           0   0   0  28   1  0   0  0   0   0
government        0   3   1   0   0  1  18  0   5   2
hobbies           6   0   1   3   5  2   4  1   1  13
humor             0   0   0   8   1  0   0  0   0   0
learned           2   8  19   2  13  3   7  0   3  23
lore              4   8   0  11   8  6   2  1   5   3
mystery           0   0   0  24   0  0   0  0   0   0
news             10   1   0   4   1  8   7  0  13   0
religion          0   5   1   2   3  0   0  6   0   0
reviews           1   0   0   0  15  0   0  0   1   0
romance           0   0   0  29   0  0   0  0   0   0
science_fiction   0   0   0   6   0  0   0  0   0   0
```

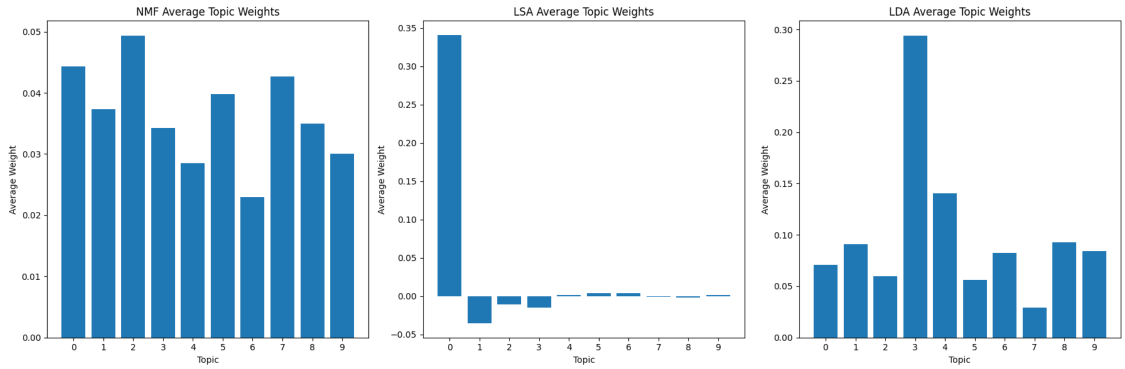
[18]:
```python
# Visualize topic distributions
fig, axes = plt.subplots(1, 3, figsize=(18, 6))

models = [('NMF', nmf_topics), ('LSA', lsa_topics), ('LDA', lda_topics)]

for idx, (model_name, topic_matrix) in enumerate(models):
    # Calculate average topic weights
    avg_topic_weights = np.mean(topic_matrix, axis=0)

    axes[idx].bar(range(len(avg_topic_weights)), avg_topic_weights)
    axes[idx].set_title(f'{model_name} Average Topic Weights')
    axes[idx].set_xlabel('Topic')
    axes[idx].set_ylabel('Average Weight')
    axes[idx].set_xticks(range(len(avg_topic_weights)))

plt.tight_layout()
plt.show()
```



1.8 6. Conclusions and Insights

1.8.1 6.1 Model Comparison Summary

NMF (Non-negative Matrix Factorization): - *[Add your analysis of NMF performance and characteristics]*

LSA (Latent Semantic Analysis): - *[Add your analysis of LSA performance and characteristics]*

LDA (Latent Dirichlet Allocation): - *[Add your analysis of LDA performance and characteristics]*

1.8.2 6.2 Comparison with Official Brown Corpus Categories

[Discuss how well each model's topics align with the official Brown corpus categories. Which model performed best at capturing the underlying document structure?]

1.8.3 6.3 Key Findings

1. *[Finding 1]*
2. *[Finding 2]*
3. *[Finding 3]*

1.8.4 6.4 Recommendations

[Based on your analysis, which topic modeling approach would you recommend for different use cases?]

1.9 7. Additional Analysis (Optional)

1.9.1 7.1 Topic Coherence Analysis

[If time permits, add topic coherence analysis or other advanced metrics]

[19]:
```
# Optional: Add any additional analysis code here
print("Assignment completed successfully!")
print("\nNext steps:")
```

```python
print("1. Fill in the interpretation sections with your analysis")
print("2. Run all cells and verify results")
print("3. Convert notebook to PDF for submission")
print("4. Commit and push to GitHub")
```

```
Assignment completed successfully!

Next steps:
1. Fill in the interpretation sections with your analysis
2. Run all cells and verify results
3. Convert notebook to PDF for submission
4. Commit and push to GitHub
```