CS 688 : Assignment 5

Unsupervised Topic Modeling in the 20 Newsgroups Dataset

Prepared By:
Tejas Yogesh Pawar

INTRODUCTION

This project delves into the realm of unsupervised learning through Topic Modeling, applying it to the fetch_20newsgroups dataset from sklearn.datasets. The goal is to uncover thematic structures in this extensive collection of newsgroup documents, which span a variety of subjects.

Topic Modeling:

Topic Modeling, encompassing techniques like Latent Dirichlet Allocation (LDA) or Latent Semantic Analysis (LSA), aims to identify dominant topics within a corpus. This process involves text preprocessing steps such as tokenization, removal of punctuation, stopwords, and stemming, culminating in the creation of a bag of words. These steps are crucial in transforming raw data into a structured format for effective topic discovery.

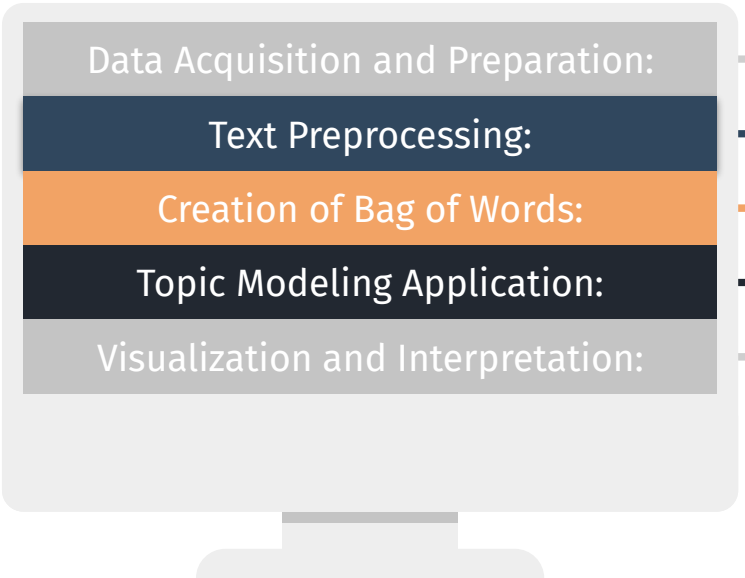
OBJECTIVES

To extract and interpret key topics from the 20 Newsgroups dataset usir Topic Modeling.

To Visualize these topics through word counts and word clouds for an intuitive understanding

The project not only highlights the application of Topic Modeling in text analytics but also showcases its potential in extracting meaningful insights from large, unstructured text data, relevant across various fields like journalism, market research, and social media analysis.

METHODOLOGY



Utilize the `fetch_20newsgroups` dataset from sklearn.datasets, a comprehensive collection of approximately 20,000 newsgroup documents. Perform initial data cleaning and preprocessing to prepare the dataset for analysis.

The textual data undergoes crucial steps including tokenization (splitting into words/tokens), punctuation removal and lowercasing for consistency, stop words removal to filter out low-value words, and stemming to simplify words to their root forms.

Utilize a TF-IDF (Term Frequency-Inverse Document Frequency) approach to convert preprocessed text into a numerical format, emphasizing word frequency while accounting for their commonness across documents, resulting in a refined document-term matrix.

Apply a Topic Modeling technique, such as Latent Dirichlet Allocation (LDA), to the bag of words. This step involves identifying clusters of words that frequently occur together in documents, which are indicative of underlying topics.

Use visualization tools, like word clouds and frequency plots, to illustrate the identified topics. Analyze and interpret the significance of these topics, discussing their relevance and distribution across the dataset. This step is crucial for presenting the findings.

Data Acquisition and Preparation:

```
from sklearn.datasets import fetch_20newsgroups
newsgroups = fetch_20newsgroups(subset='all')
```

```
topics = list(set(newsgroups.target_names))
topics.sort()

topics
```

Data Collection: The dataset is sourced from `sklearn.datasets` using the `fetch_20newsgroups` function. The `subset='all'` parameter ensures the inclusion of the entire collection of newsgroup documents.

Dataset Overview: The dataset comprises a diverse range of topics, which are extracted and listed in a sorted order to provide a clear understanding of the scope and variety within the dataset. This step is crucial for setting the stage for subsequent topic modeling and analysis.

Text Preprocessing:

1. **Tokenization:** The text from the dataset is broken down into individual words or tokens, a crucial step for parsing and understanding the textual content. The code iterates through the data, utilizing a natural language processing tool (nlp) for tokenization, and stores the tokens for each document.

```
full_data = newsgroups.data

# Tokenizing the dataset
tokenized_texts = []
for text in full_data:
    doc = nlp(text)
    tokens = [token.text for token in doc]
    tokenized_texts.append(tokens)
```

2. **Lowercasing:** To maintain consistency and avoid redundancy, all characters in the tokenized text are converted to lowercase. This ensures that words with different case forms (like "Email" and "email") are treated identically.

```
# Lowercasing the tokens in the tokenized dataset
lowercased_texts = []
for document in tokenized_texts:
    lowercased_document = [token.lower() for token in document]
    lowercased_texts.append(lowercased_document)

# Displaying the first few lowercased tokens of the first document as an example
print(lowercased_texts[0][:20])
```

Text Preprocessing (Continued):

3. Removing Punctuation and Special Characters: Punctuation and special characters are often extraneous for text analysis, as they usually do not contribute to the core meaning of the text. A custom function is created to filter out these elements from each token.

```
# Create a function that removes punctuation from each token.
def remove_punctuation(tokens):
    return [token for token in tokens if token not in string.punctuation]
```

```
# Use this function to process each document in your dataset.
cleaned_texts = []
for document in lowercased_texts:
    cleaned_document = remove_punctuation(document)
    cleaned_texts.append(cleaned_document)

print(cleaned_texts[0][:20])
```

4. Removing Stop Words: Stop words, such as "is", "and", "the", etc., are common words that typically carry little semantic value and are thus removed to focus on more meaningful content. The **spaCy** language model provides a list of such stop words for effective filtering.

```
# Get the List of Stop Words: spaCy's language model includes a list of stop words.
stop_words = nlp.Defaults.stop_words
```

```
# Create a function that filters out stop words from the tokens.
def remove_stop_words(tokens):
    return [token for token in tokens if token not in stop_words]
```

```
# Use this function to process each document in your dataset.
no_stop_words_texts = []
for document in cleaned_texts:  # Assuming 'cleaned_texts' is your data after removing punctuation
    no_stop_word_document = remove_stop_words(document)
    no_stop_words_texts.append(no_stop_word_document)
print(no_stop_words_texts[0][:20])
```

Text Preprocessing (Continued):

5. Removing Numbers and Nonstandard Words: To enhance the focus on meaningful text, a function is implemented to remove numbers and nonstandard words (those with less than three characters) from the tokenized data.

```
def remove_numbers_and_nonstandard_words(tokens):
    cleaned_tokens = []
    for token in tokens:
        if not re.search(r'\d', token) and len(token) > 2:
            cleaned_tokens.append(token)
    return cleaned_tokens

# Apply this function to your tokenized data
no_numbers_nonstandard_texts = [remove_numbers_and_nonstandard_words(doc) for doc in no_stop_words_texts]
```

6. Removing Email Addresses and URLs: Email addresses and URLs are often irrelevant for text analysis and are removed using regular expressions. The tokenized data is first converted back into string format, then cleaned to remove these elements, and tokenized again.

```
# Regex patterns for email addresses and URLs
email_pattern = r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b'
url_pattern = r'http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[!*\\(\\),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+'

def remove_emails_and_urls(text):
    text = re.sub(email_pattern, '', text)
    text = re.sub(url_pattern, '', text)
    return text

# Convert each document from a list of tokens to a single string
string_texts = [' '.join(doc) for doc in no_numbers_nonstandard_texts]

# Apply the function to remove emails and URLs
cleaned_texts = [remove_emails_and_urls(doc) for doc in string_texts]

# Tokenize again if needed
tokenized_cleaned_texts = [[token.text for token in nlp(doc)] for doc in cleaned_texts]
```

Text Preprocessing (Continued):

7. Handling or Removing HTML Tags: HTML tags in the dataset, which are irrelevant for textual analysis, need to be processed and removed. This is achieved using `BeautifulSoup,` a Python library for parsing HTML and XML documents.

```
# Load the English tokenizer from spaCy
nlp = spacy.load("en_core_web_sm")
```

```
# Function to remove HTML tags using BeautifulSoup
def remove_html_tags(text):
    soup = BeautifulSoup(text, "html.parser")
    return soup.get_text()

# Join the tokens into strings
joined_texts = [' '.join(doc) for doc in tokenized_cleaned_texts]

# Remove HTML tags
no_html_texts = [remove_html_tags(doc) for doc in joined_texts]

# Retokenize the texts
re_tokenized_texts = [[token.text for token in nlp(doc)] for doc in no_html_texts]
```

Text Refinement - Stemming and Lemmatization:

8. Stemming: This process reduces words to their root or base form, standardizing variations of a word. For instance, "running", "runs", and "ran" are all stemmed to "run". We use NLTK's `PorterStemmer` for this purpose.

```
# Use NLTK's PorterStemmer for stemming.
stemmer = PorterStemmer()


stemmed_texts = [[stemmer.stem(token) for token in doc] for doc in cleaned_tokenized_texts]

# Example: Displaying the first few stemmed tokens of the first document
print(stemmed_texts[0][:20])
```

9. Lemmatization: Going beyond stemming, lemmatization considers the context and meaning of words, linking similar meanings to a base word. For instance, "better" is lemmatized to "good". This step is performed using spaCy, a powerful NLP library. The stemmed texts are first joined into sentences, then lemmatized using spaCy.

```
# Join the stemmed tokens back into sentences
joined_stemmed_texts = [' '.join(doc) for doc in stemmed_texts]

# Apply lemmatization to the joined texts
lemmatized_texts = [[token.lemma_ for token in nlp(doc)] for doc in joined_stemmed_texts]

# Example: Displaying the first few lemmatized tokens of the first document
print(lemmatized_texts[0][:20])
```

Creation of Bag of Words:

- **Transforming Text into Numerical Format:** The refined text data is converted into a numerical format using a Bag of Words model. This model quantifies the text by counting the frequency of each word, leading to the formation of a document-term matrix.

- **Process of Bag of Words Creation:**
  - First, the preprocessed and corrected tokens from each document are joined into a single string, forming a consolidated text for each document.

```
# Join the Tokens into a Single String for Each Document:
processed_texts = [' '.join(doc) for doc in spell_corrected_texts]
```

  - The Bag of Words model is created using sklearn's `CountVectorizer`. This tool converts the text into a matrix of token counts, effectively translating the textual data into a numerical format. The parameters `max_df` and `min_df` are set to filter out terms that are too frequent or too rare, enhancing the model's focus on relevant words. Additionally, English stop words are excluded to maintain the model's emphasis on meaningful content.

```
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer(max_df=0.95, min_df=2, stop_words='english')
X = vectorizer.fit_transform(processed_texts)
```

Topic Modeling Application:

- **Topic Modeling with LDA:** Latent Dirichlet Allocation (LDA) is an advanced unsupervised learning technique in Topic Modeling, used to identify themes and topics within large text corpora. In this project, LDA is applied to the 20 Newsgroups dataset, where it operates under the premise that each document is a mixture of various topics and each topic is characterized by a specific distribution of words.

```
from sklearn.decomposition import LatentDirichletAllocation

n_topics = 20  # Define the number of topics
lda = LatentDirichletAllocation(n_components=n_topics, max_iter=10, learning_method='online', random_state=0)
lda.fit(X)
```

```
LatentDirichletAllocation(learning_method='online', n_components=20,
                          random_state=0)
```

- **The LDA process involves:**

  1. Random Initialization: Initially, every word in each document is randomly assigned to one of the pre-defined number of topics.
  2. Iterative Optimization: Through iterative refinement, LDA adjusts these word-topic assignments based on two key probabilities: the prevalence of each word in topics and the distribution of topics in each document.
  3. Convergence: After a set number of iterations or upon convergence, the model arrives at a stable distribution where each document is represented as a mixture of different topics, and each topic is defined by a cluster of words.

Topic Modeling Application: (Continued)

- **Topic Display Function:** A custom function, display_topics, is defined to showcase the topics extracted by the LDA model. This function enumerates through each topic, displaying the top words associated with each, providing a clear and concise overview of the topic's thematic focus.

```
# Display the Topics:
def display_topics(model, feature_names, no_top_words):
    for topic_idx, topic in enumerate(model.components_):
        print("Topic %d:" % (topic_idx))
        print(" ".join([feature_names[i] for i in topic.argsort()[:-no_top_words - 1:-1]]))

no_top_words = 10
display_topics(lda, tfidf_vectorizer.get_feature_names_out(), no_top_words)
```

- **Insights and Interpretation:** The output from this function offers valuable insights into the dataset's themes. Each topic is represented by a set of words that most strongly define it, allowing for an intuitive understanding of the diverse subjects covered in the dataset.

This application of the LDA model is crucial in uncovering hidden patterns and themes within the data, highlighting the power of topic modeling in understanding and categorizing large volumes of text data.

Visualization and Interpretation:

Word Clouds for Top 10 Words: To effectively convey the essence of each topic identified by the Latent Dirichlet Allocation (LDA) model, we generate word clouds that visualize the top 10 words of each topic. This approach provides a clear and engaging representation of the most significant words within each topic.

Visualization Technique: Using the WordCloud library in Python, we create word clouds where the size of each word corresponds to its frequency or importance in the topic. The larger the word, the more central it is to the topic's theme.

Visualization and Interpretation: WordCloud (Continued)

Word Clouds for Top 10 Words of all Topics

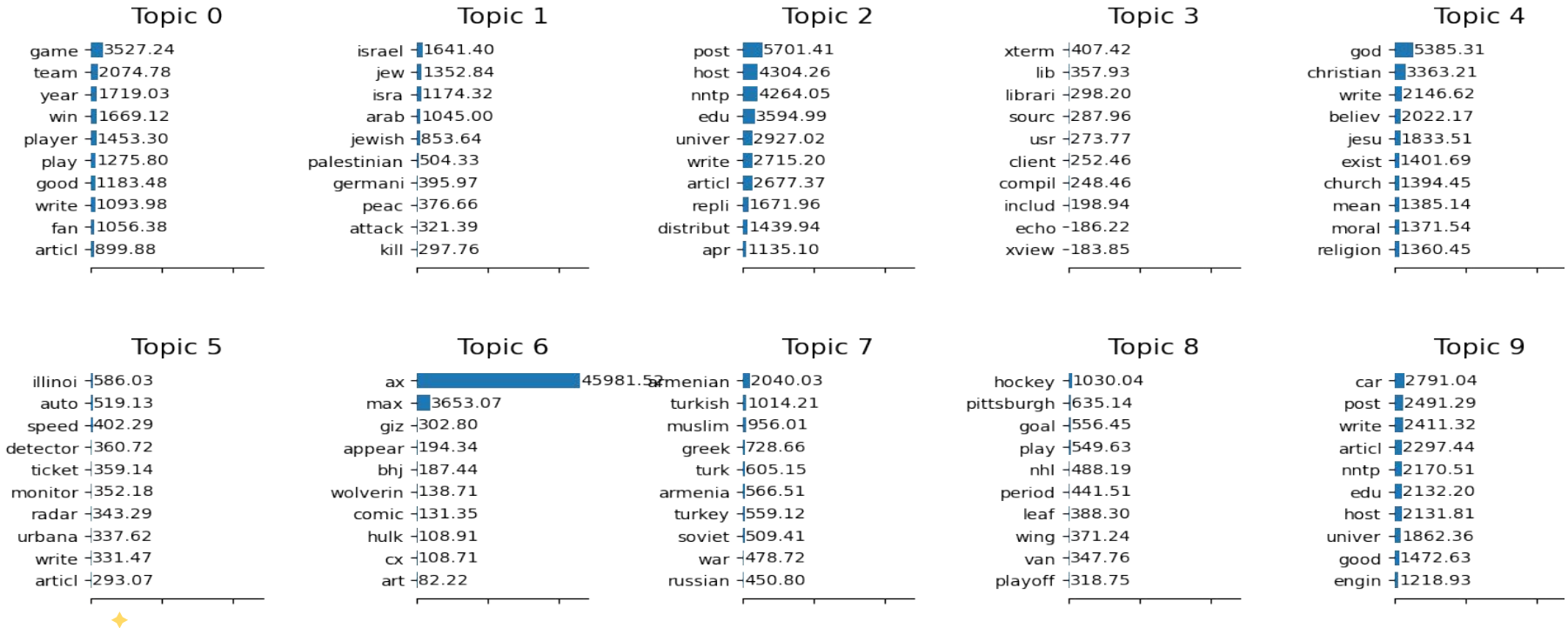
Visualization and Interpretation: WordCloud (Continued)

Word Clouds for Top 10 Words of all Topics

In summary, the use of word clouds to visualize the top 10 words from each topic significantly aids in our analysis, making it easier to interpret the topics and communicate our findings effectively.

Visualization and Interpretation: BarChart (Continued)

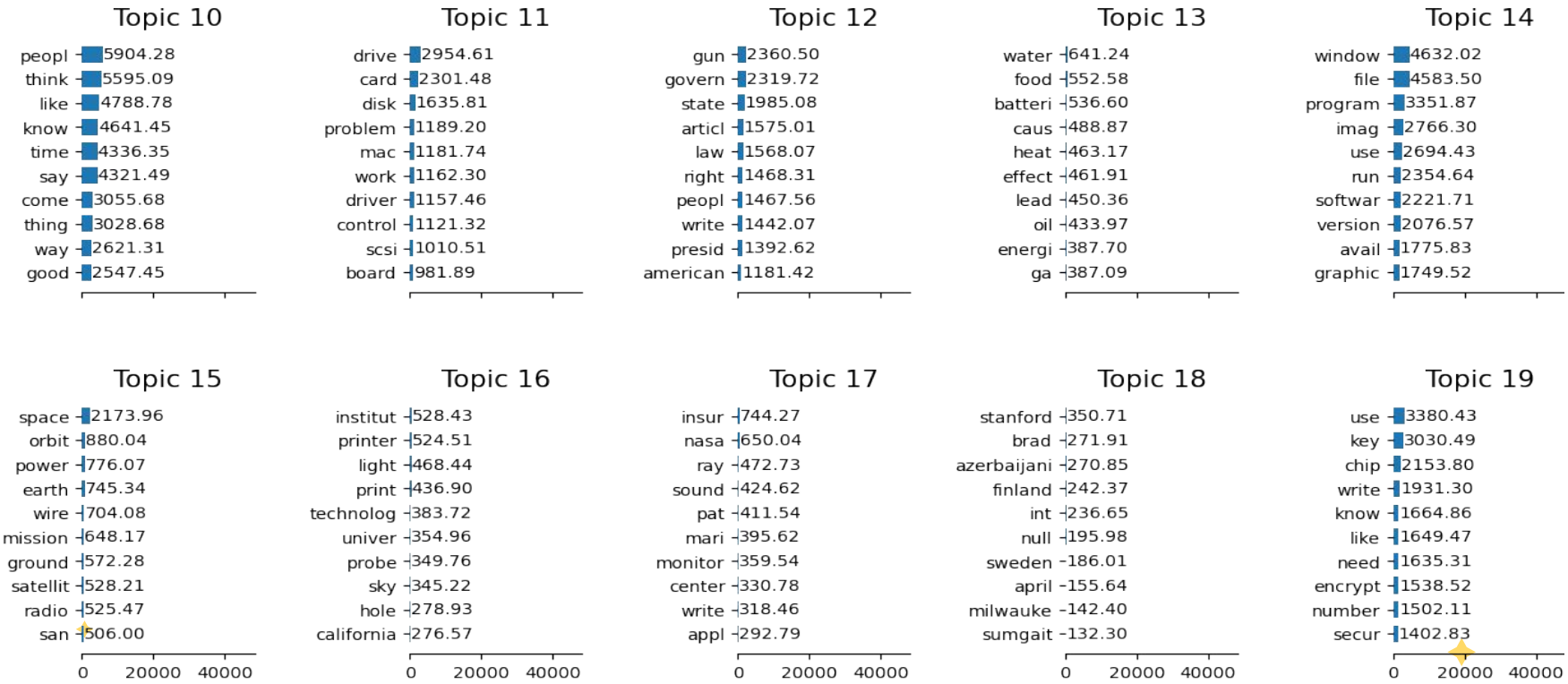
Bar Chart Visualization of Top Words in Each Topic



To complement the word clouds and provide a more quantitative view of the topic analysis, we use bar charts to visualize the top words in each topic.

Visualization and Interpretation: BarChart (Continued)

Bar Chart Visualization of Top Words in Each Topic

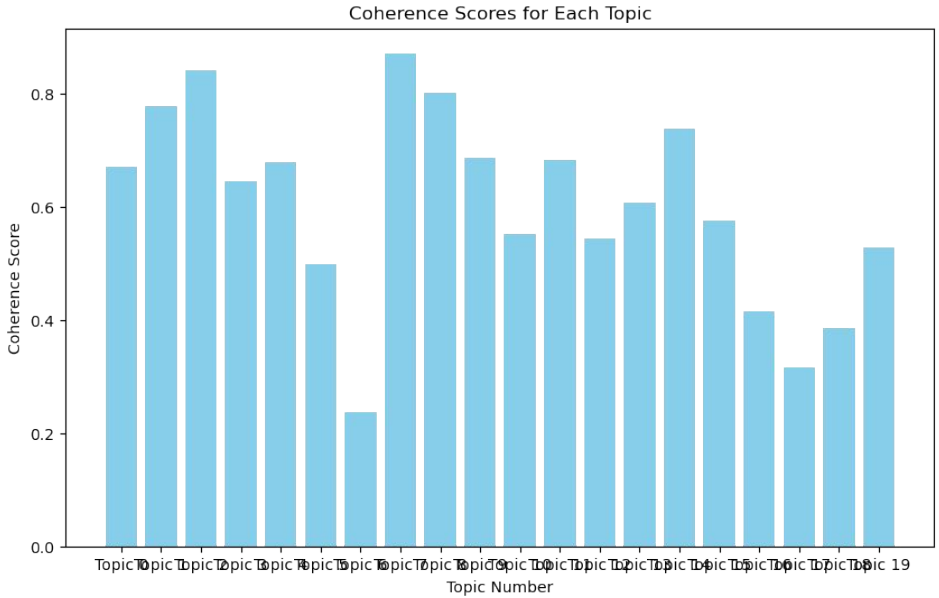


This method allows us to see not only the most significant words but also their relative weights within each topic.

Visualization and Interpretation: CoherenceScore (Continued)

LDA Models Overall Coherence Score: 0.6037321905994115

A score around 0.60 suggests that the topics are fairly interpretable. It means that the words within each topic tend to have semantic similarities, making them more meaningful when grouped together.



The coherence scores reveal a mix of clear and ambiguous topics in our LDA model: Topics 1, 2, 7, and 8 are highly coherent and well-defined; topics like 0, 4, and 14 show moderate coherence; while topics 5, 6, 16, 17, and 18 are less coherent, indicating a need for further model refinement.

CONCLUSION

In this project, we successfully implemented text preprocessing, LDA topic modeling, and visualization to extract and interpret topics from the 20 newsgroups dataset. Through preprocessing steps like tokenization, lowercasing, and removing punctuation, stop words, and numbers, we prepared the data for effective modeling. The LDA model, optimized through hyperparameter tuning, revealed topics of varying coherence, indicating the quality and clarity of the thematic extraction. Visualizations like word clouds and bar charts provided an intuitive understanding of the topics, aiding in their interpretation. This project demonstrates the synergy of technical modeling and careful analysis in uncovering meaningful insights from textual data.