Title and Introduction

Data Preprocessing Lab: Generative AI

Welcome to the Data Preprocessing Lab for Generative AI!

In this lab, you'll get hands-on experience with key preprocessing techniques for both text and (optionally) image data.

Learning Objectives:

- Understand and apply core data preprocessing techniques.
- Explore word embedding techniques (Word2Vec/GloVe, BERT).
- Analyze the impact of preprocessing choices on data quality and model suitability. List item
- Practice using cosine similarity for comparing embeddings.

Part 1: Environment Setup

First, we'll install and import all necessary libraries. Run the following cell to set up your environment.

```
# SECTION 1: Environment Setup
#############################
# This cell installs and imports all necessary libraries for our text preprocessing pipeline.
# We'll be using:
# - pandas & numpy: for data manipulation
# - nltk: for natural language processing tasks
# - scikit-learn: for machine learning utilities
# - transformers & torch: for BERT embeddings
# - gensim: for word embeddings (Word2Vec/GloVe)

# Install required packages
%pip uninstall -y numpy pandas
%pip install pandas numpy==1.26.4 nltk scikit-learn transformers torch datasets gensim


# TODO: Import the required libraries
# Hint: You need pandas, numpy, nltk, and sklearn components
# YOUR CODE HERE - import the basic libraries
import pandas as pd
import numpy as np
import nltk
# Add more imports as needed...

# These are more advanced imports you'll need later
from transformers import BertTokenizer, BertModel
import torch
import gensim.downloader as api
from gensim.models import KeyedVectors
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity # for

# Download required NLTK data
# These are necessary for tokenization, stop words, and lemmatization
nltk.download('punkt') # Added punkt_tab
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('omw-1.4')

print("Setup complete! All required libraries have been imported.")
```

```
Found existing installation: numpy 1.26.4
Uninstalling numpy-1.26.4:
  Successfully uninstalled numpy-1.26.4
Found existing installation: pandas 2.3.3
Uninstalling pandas-2.3.3:
  Successfully uninstalled pandas-2.3.3
Collecting pandas
  Using cached pandas-2.3.3-cp312-cp312-manylinux_2_24_x86_64.manylinux_2_28_x86_64.whl.metadata (91 kB)
Collecting numpy==1.26.4
  Using cached numpy-1.26.4-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (61 kB)
Requirement already satisfied: nltk in /usr/local/lib/python3.12/dist-packages (3.9.1)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.12/dist-packages (1.6.1)
Requirement already satisfied: transformers in /usr/local/lib/python3.12/dist-packages (4.57.0)
Requirement already satisfied: torch in /usr/local/lib/python3.12/dist-packages (2.8.0+cu126)
Requirement already satisfied: datasets in /usr/local/lib/python3.12/dist-packages (4.0.0)
Requirement already satisfied: gensim in /usr/local/lib/python3.12/dist-packages (4.3.3)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.12/dist-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-packages (from pandas) (2025.2)
Requirement already satisfied: click in /usr/local/lib/python3.12/dist-packages (from nltk) (8.3.0)
Requirement already satisfied: joblib in /usr/local/lib/python3.12/dist-packages (from nltk) (1.5.2)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.12/dist-packages (from nltk) (2024.11.6)
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dist-packages (from nltk) (4.67.1)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (1.13.1)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (3.6.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dist-packages (from transformers) (3.20.0)
Requirement already satisfied: huggingface-hub<1.0,>=0.34.0 in /usr/local/lib/python3.12/dist-packages (from transformers) (0.35.3)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dist-packages (from transformers) (25.0)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.12/dist-packages (from transformers) (6.0.3)
Requirement already satisfied: requests in /usr/local/lib/python3.12/dist-packages (from transformers) (2.32.4)
Requirement already satisfied: tokenizers<=0.23.0,>=0.22.0 in /usr/local/lib/python3.12/dist-packages (from transformers) (0.22.1)
```

Part 2: Loading and Exploring the BBC News Dataset

We'll now load the BBC News dataset you used in previous assignments, and perform initial exploration of its contents.

```
# SECTION 2: Data Loading and Initial Exploration
##############################################
# Here we load the BBC News dataset and perform initial analysis
# Understanding our data is crucial before applying any preprocessing

# Load the dataset
from google.colab import files
import pandas as pd

# Use the uploaded filename exactly
# Assuming the uploaded file is named 'bbc-news-data.csv'
# You might need to adjust the filename if it's different after uploading
try:
  uploaded = files.upload()
  file_name = list(uploaded.keys())[0]
  df = pd.read_csv(file_name, encoding='latin1')

  # Verify load
  print("Dataset loaded successfully with shape:", df.shape)
  display(df.head())

  # TODO: Rename the columns to match our processing pipeline
  # Hint: The original columns are 'Text' and 'Category'
  # YOUR CODE HERE
  # Assuming the columns in the CSV are 'labels' and 'text' based on the kernel state
  df.rename(columns={'labels': 'category', 'text': 'text'}, inplace=True)
  print("Renamed columns to 'category' and 'text'")

  # Verify that the 'category' column exists after renaming
  if 'category' not in df.columns:
      raise KeyError("Column 'category' not found after renaming. Please check the original column names in your CSV file.")


  # TODO: Perform basic data exploration
  # TASK 1: Display the first few rows and basic information about the dataset
  # Hint: Use pandas' head(), info(), and describe() methods
  # YOUR CODE HERE
  print("\nDataset Info:")
  df.info()
  print("\nDataset Description:")
  display(df.describe(include='all'))


  # TASK 2: Analyze the distribution of categories
  # Hint: Use value_counts() on the category column
  # YOUR CODE HERE
  print("\nCategory Distribution:")
  display(df['category'].value_counts())


  # TASK 3: Calculate and display basic text statistics
  # Calculate average text length per category
  # TODO: Create a visualization of text lengths by category
  # Hint: Use seaborn's boxplot
```

```
  # YOUR CODE HERE
  df['text_length'] = df['text'].str.len()
  print("\nText Length Statistics per Category:")
  display(df.groupby('category')['text_length'].describe())

  # TODO: Create a visualization of text lengths by category
  # Hint: Use seaborn's boxplot
  # YOUR CODE HERE
  import seaborn as sns
  import matplotlib.pyplot as plt

  plt.figure(figsize=(10, 6))
  sns.boxplot(x='category', y='text_length', data=df)
  plt.title('Distribution of Text Lengths by Category')
  plt.xlabel('Category')
  plt.ylabel('Text Length')
  plt.xticks(rotation=45)
  plt.show()

  # Display your findings
  print("\nDataset Statistics:")
  # TODO: Add code to display your findings
  # YOUR CODE HERE
  # Already displayed above with describe(), value_counts(), and groupby().describe()

except FileNotFoundError:
  print("Please upload the 'bbc-news-data.csv' file using the files.upload() function.")
except KeyError as e:
  print(f"KeyError: {e}. Please check the column names in your CSV file.")
except Exception as e:
  print(f"An unexpected error occurred: {e}")
```

Part 3: Text Preprocessing

We'll now implement basic text preprocessing steps to clean our data.

Comprehension Questions - Data Exploration

Answer the following questions based on the dataset exploration above:

1. What are the dimensions of our dataset? The datasets has 7 columns and 100 rows.
2. How many different categories are there in the news articles? We have 7 categories which are: text, labels, no_sentences, Flesch Reading Ease Score, Dale-Chall Readability Score, text_rank_summary, lsa_summary

3. [redacted]t balanced across categories? Why might this matter? The Data is balanced because it has the same numbers of samples, because data balanced prevents the model bias toward one topic
4. Are there any missing values that need to be addressed? This csv file has been modified since it has 1000 rows and I had to cut it down to 100 rows and there is no missing values in the new csv file.

```
bbc_news_...rows (1).csv
bbc_news_100rows (1).csv(text/csv) - 273145 bytes, last modified: 10/15/2025 - 100% done
Saving bbc_news_100rows (1).csv to bbc_news_100rows (1) (9).csv
Dataset loaded successfully with shape: (100, 7)
```

```
# SECTION 3: Text Cleaning and Preprocessing
########################################
# This section implements fundamental text preprocessing steps:
# 1. Converting to lowercase (why? -> maintains consistency)
# 2. Removing special characters (why? -> reduces noise)
# 3. Handling whitespace (why? -> standardizes format)

########################################
import re # Moved import to the top

def clean_text(text):
    """
    Performs basic text cleaning operations.

    Parameters:
    text (str): Input text to be cleaned

    Returns:
    str: Cleaned text
    """
    # TODO: Implement the following steps:
    # 1. Convert to lowercase
    # 2. Remove URLs and emails
    # 3. Remove special characters but keep sentence structure
    # 4. Remove extra whitespace
    # Hint: Use string methods and regular expressions

    # YOUR CODE HERE
    text = str(text).lower()  # 1) lowercase
    text = re.sub(r'(https?://\S+|www\.\S+)', ' ', text) # 2) remove URLs
    text = re.sub(r'\b[\w\.-]+@[\w\.-]+\.\w+\b', ' ', text) # 2) remove emails
    text = re.sub(r"[^a-z0-9\.\,\!\?\:\;\'\"\s]", " ", text) # 3) remove special chars but keep basic sentence punctuation
    # normalize spacing before punctuation (moved inside the function)
    text = re.sub(r"\s+([.,!?:;])", r"\1", text)
    # 4) collapse extra whitespace
    text = re.sub(r"\s+", " ", text).strip()

    return text

# Test the function with a sample (Corrected indentation)
sample_text = "Hello, World! This is a TEST... 123. Check out this link: https://example.com and email: test@example.com"
print("Original:", sample_text)
print("Cleaned:", clean_text(sample_text))

# Apply to the entire dataset (Corrected indentation)
df['cleaned_text'] = df['text'].apply(clean_text)
```

```
Original: Hello, World! This is a TEST... 123. Check out this link: https://example.com and email: test@example.com
Cleaned: hello, world! this is a test... 123. check out this link: and email:
```

	text	labels	no_sentences	Flesch Reading	Dale-Chall Readability	text_rank_summary	lsa_summary
unique	100	1	NaN	NaN	NaN	100	100
top	Ad sales boost Time Warner profit\n\nQuarterly...	business	NaN	NaN	NaN	It hopes to increase subscribers by offering t...	Its profits were buoyed by one-off gains which...
freq	1	100	NaN	NaN	NaN	1	1
mean	NaN	NaN	16.730000	61.236700	9.625900	NaN	NaN
std	NaN	NaN	5.136235	7.903453	0.606077	NaN	NaN
min	NaN	NaN	7.000000	34.360000	8.340000	NaN	NaN
25%	NaN	NaN	13.000000	55.662500	9.167500	NaN	NaN
50%	NaN	NaN	16.000000	61.415000	9.665000	NaN	NaN

Part 4: Tokenization and Advanced Processing

Now we'll tokenize our text and apply more advanced preprocessing techniques including:

- Tokenization
- Stop word removal
- Lemmatization

```
# SECTION 4: Tokenization and Advanced Processing
############################################
# This section implements more sophisticated NLP techniques:
# - Tokenization: splitting text into words
# - Stop word removal: removing common words
# - Lemmatization: reducing words to their base form
# Check if you do not need to install any additional libraries
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer

# Initialize our tools
stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()

def tokenize_and_process(text):
```

```
    """
    Performs advanced text processing including tokenization,
    stop word removal, and lemmatization.

    Parameters:
    text (str): Cleaned text to process

    Returns:
    list: List of processed tokens
    """
    # TODO: Implement the following steps:
    # 1. Tokenize the text
    # 2. Remove stop words
    # 3. Apply lemmatization
    # Hint: Use the initialized stop_words and lemmatizer

    # YOUR CODE HERE
    tokens = word_tokenize(text) # 1. Tokenize the text
    processed_tokens = [
        lemmatizer.lemmatize(word) for word in tokens if word not in
stop_words # 2. Remove stop words and 3. Apply lemmatization
 ]
    return processed_tokens

# Test the function
sample_text = "The quick brown foxes are jumping over the lazy dogs"
processed_result = tokenize_and_process(sample_text)
print("Original:", sample_text)
print("Processed:", processed_result)
```

```
Original: The quick brown foxes are jumping over the lazy dogs
Processed: ['The', 'quick', 'brown', 'fox', 'jumping', 'lazy', 'dog']
```

Part 5: Word Embeddings with GloVe

We'll now generate word embeddings using pre-trained GloVe vectors. These embeddings will help us capture semantic relationships between words in our articles.

```
# SECTION 5: Word Embeddings with GloVe
###################################
# This section generates word embeddings using pre-trained GloVe vectors
# Word embeddings capture semantic relationships between words
# by representing them as dense vectors in a high-dimensional space

import gensim.downloader as api
import numpy as np # Import numpy
from nltk.tokenize import word_tokenize # Import word_tokenize
from nltk.stem import WordNetLemmatizer # Import WordNetLemmatizer
from nltk.corpus import stopwords # Import stopwords

# Initialize NLTK tools needed for tokenization and processing within the embedding function
stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()


# Load pre-trained GloVe embeddings
glove_model = api.load("glove-wiki-gigaword-100")

def tokenize_and_process_for_glove(text):
    """
    Performs basic text processing including tokenization,
    stop word removal, and lemmatization suitable for GloVe.

    Parameters:
    text (str): Cleaned text to process

    Returns:
    list: List of processed tokens
    """
    # 1) Tokenize
    tokens = word_tokenize(str(text))

    # 2) Remove stop words (and drop non-alphabetic tokens); lowercase for consistency
    tokens = [t.lower() for t in tokens if t.isalpha() and t.lower() not in stop_words]

    # 3) Lemmatize
    processed_tokens = [lemmatizer.lemmatize(t) for t in tokens]

    return processed_tokens


def get_word2vec_embedding(text, model):
```

```
        """
        Generates document embeddings by averaging word vectors.

        Parameters:
        text (str): Input text (cleaned text is expected)
        model: Pre-trained word embedding model

        Returns:
        numpy.array: Document embedding vector
        """
        # TODO: Implement the following steps:
        # 1. Tokenize the input text
        # 2. Get embedding for each token
        # 3. Average the embeddings
        # Hint: Handle words not in vocabulary

        # YOUR CODE HERE
        # 1) Tokenize and process the text using the helper function
        processed_tokens = tokenize_and_process_for_glove(text)

        # 2) collect vectors for in-vocab tokens (case-insensitive fallback)
        vecs = []
        for w in processed_tokens:
            if w in model.key_to_index:
                vecs.append(model[w])
            elif w.lower() in model.key_to_index:
                vecs.append(model[w.lower()])

        # 3) Average the embeddings
        return np.mean(vecs, axis=0) if vecs else np.zeros(model.vector_size)

    # Apply to a sample of the dataset
    sample_size = 100
    sample_df = df.head(sample_size).copy()
    sample_df['glove_embedding'] = sample_df['cleaned_text'].apply(
        lambda x: get_word2vec_embedding(x, glove_model)
    )
    print("Sample of embeddings:")
    # Apply to a sample of the dataset
    sample_size = 100
    [=======================] 100.0%
    128.1/128.1MB downloaded
```

```
Sample of embeddings:
```

Part 6: BERT Embeddings

Now we'll use BERT to generate contextual embeddings. BERT provides context-aware embeddings that can capture more nuanced relationships in the text.

```
# SECTION 6: BERT Embeddings
#########################
# This section implements BERT (Bidirectional Encoder Representations from Transformers)
# BERT provides context-aware embeddings, meaning the same word can have different
# embeddings based on its context in the sentence.
# Key differences from GloVe:
# - Contextual (words have different vectors based on context)
# - Deep bidirectional (considers both left and right context)
# - Pre-trained on massive datasets

from transformers import BertTokenizer, BertModel # Added import
import torch # Added import
import numpy as np # Added import for numpy usage


# Load BERT model and tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertModel.from_pretrained('bert-base-uncased')

def get_bert_embedding(text, max_length=512):
    """
    Generates BERT embeddings for a text.

    Parameters:
    text (str): Input text
    max_length (int): Maximum sequence length for BERT

    Returns:
    numpy.array: BERT embedding vector
    """
    # TODO: Implement the following steps:
```

```
    # 1. Tokenize the text using BERT tokenizer
    # 2. Generate BERT embeddings
    # 3. Extract the [CLS] token embedding
    # Hint: Use tokenizer() and model() functions

    # YOUR CODE HERE
    # Step 1: Tokenize
    inputs = tokenizer(
        str(text),
        return_tensors='pt',
        truncation=True,
        padding='max_length',
        max_length=max_length
    )

    # Step 2: Generate embeddings
    with torch.no_grad():
        outputs = model(**inputs)   # outputs.last_hidden_state shape: [1, seq_len, 768]

    # Step 3: Extract [CLS] token embedding (first token)
    sentence_embedding = (
        outputs.last_hidden_state[:, 0, :]      # [1, 768]
        .squeeze(0)                              # [768]
        .detach()
        .cpu()
        .numpy()
    )

    return sentence_embedding

# Apply to the sample of the dataset
if 'sample_df' in globals() and not sample_df.empty:
    # Check if 'cleaned_text' column exists before applying
    if 'cleaned_text' in sample_df.columns:
        sample_df['bert_embedding'] = sample_df['cleaned_text'].apply(get_bert_embedding)
        print("BERT embeddings generated successfully.")
    else:
        print("Error: 'cleaned_text' column not found in sample_df. Please run the preprocessing steps first.")
else:
    print("Error: sample_df not found or is empty. Please load and preprocess the data first.")

# Test the function
test_text = "This is a test sentence for BERT embeddings."
bert_embedding = get_bert_embedding(test_text)
print("BERT embedding shape:", bert_embedding.shape)
```

```
BERT embeddings generated successfully.
BERT embedding shape: (768,)
```

Part 7: Comparing Embeddings

Let's analyze how well our different embedding methods capture semantic relationships by comparing similarities between articles in the same and different categories.

```
# SECTION 7: Similarity Analysis
############################
# This section implements methods to compare different embedding
# approaches

# We'll analyze how well each embedding type captures semantic
# relationships by comparing similarities between articles in the same and different
# categories
from sklearn.metrics.pairwise import cosine_similarity
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np # Added import for numpy

def calculate_cosine_similarities(embeddings):
    """
    Calculates pairwise cosine similarities between all embeddings.
    Parameters:
    embeddings (numpy.array): Array of embeddings, shape (n_samples,
    embedding_dim)
    Returns:
    numpy.array: Similarity matrix of shape (n_samples, n_samples)
    """
    # Convert list of embeddings to numpy array if needed
    if isinstance(embeddings, list):
        embeddings = np.array(embeddings)
    # Calculate cosine similarity matrix
    similarity_matrix = cosine_similarity(embeddings)
    return similarity_matrix
```

```
def analyze_similarity_by_category(similarity_matrix, categories):
    """
    Analyzes similarities within and across categories.
    Parameters:
    similarity_matrix (numpy.array): Pairwise similarity matrix
    categories (list): List of category labels for each sample
    Returns:
    dict: Dictionary containing similarity statistics
    """
    categories = np.array(categories)
    n_samples = len(categories)
    same_category_similarities = []
    different_category_similarities = []
    # Iterate through all pairs (excluding diagonal)
    for i in range(n_samples):
        for j in range(i + 1, n_samples): # Only upper triangle,
            # excluding diagonal
            similarity = similarity_matrix[i, j]
            if categories[i] == categories[j]:
                same_category_similarities.append(similarity)
            else:
                different_category_similarities.append(similarity)
    # Calculate statistics
    results = {
    'same_category': {
    'similarities': same_category_similarities,
    'mean': np.mean(same_category_similarities) if
    same_category_similarities else 0,
    'std': np.std(same_category_similarities) if
    same_category_similarities else 0,
    'median': np.median(same_category_similarities) if
    same_category_similarities else 0,
    'count': len(same_category_similarities)
    },
    'diff_category': {
    'similarities': different_category_similarities,
    'mean': np.mean(different_category_similarities) if
    different_category_similarities else 0,
    'std': np.std(different_category_similarities) if
    different_category_similarities else 0,
    'median': np.median(different_category_similarities) if
    different_category_similarities else 0,
    'count': len(different_category_similarities)
    }
    }
    # Calculate separation score (higher is better)
    separation_score = results['same_category']['mean'] - results['diff_category']['mean']
    results['separation_score'] = separation_score
    return results

def find_most_similar_articles(similarity_matrix, categories, texts,
top_k=3):
    """
    Finds the most similar article pairs for each category
    combination.
    Parameters:
    similarity_matrix (numpy.array): Pairwise similarity matrix
    categories (list): List of category labels
    texts (list): List of original texts
    top_k (int): Number of top similar pairs to return
    Returns:
    dict: Dictionary of most similar pairs by category combination
    """
    categories = np.array(categories)
    n_samples = len(categories)
    unique_categories = np.unique(categories)
    results = {}
    # Find top similar pairs within each category
    for category in unique_categories:
        category_indices = np.where(categories == category)[0]
        category_similarities = []
        for i in range(len(category_indices)):
            for j in range(i + 1, len(category_indices)):
                idx_i, idx_j = category_indices[i], category_indices[j]
                similarity = similarity_matrix[idx_i, idx_j]
                category_similarities.append({
                'indices': (idx_i, idx_j),
                'similarity': similarity,
                'text_1': texts[idx_i][:100] + "...",
                'text_2': texts[idx_j][:100] + "..."
                })
        # Sort by similarity and take top k
        category_similarities.sort(key=lambda x: x['similarity'],
        reverse=True)
        results[f"{category}_same"] = category_similarities[:top_k]
```

```
    # Find top similar pairs across different categories
    cross_category_similarities = []
    for i in range(n_samples):
        for j in range(i + 1, n_samples):
            if categories[i] != categories[j]:
                similarity = similarity_matrix[i, j]
                cross_category_similarities.append({
                'indices': (i, j),
                'categories': (categories[i], categories[j]),
                'similarity': similarity,
                'text_1': texts[i][:100] + "...",
                'text_2': texts[j][:100] + "..."
                })
    cross_category_similarities.sort(key=lambda x: x['similarity'],
    reverse=True)
    results['cross_category'] = cross_category_similarities[:top_k]
    return results

def visualize_similarity_analysis(glove_analysis, bert_analysis):
    """
    Creates visualizations comparing GloVe and BERT similarity
    analyses.
    Parameters:
    glove_analysis (dict): GloVe similarity analysis results
    bert_analysis (dict): BERT similarity analysis results
    """
    fig, axes = plt.subplots(2, 2, figsize=(15, 10))
    # Plot 1: Distribution of similarities for GloVe
    ax1 = axes[0, 0]
    ax1.hist(glove_analysis['same_category']['similarities'],
    alpha=0.7,
    bins=30, label='Same Category', color='blue')
    ax1.hist(glove_analysis['diff_category']['similarities'],
    alpha=0.7,
    bins=30, label='Different Category', color='red')
    ax1.set_title('GloVe Similarity Distributions')
    ax1.set_xlabel('Cosine Similarity')
    ax1.set_ylabel('Frequency')
    ax1.legend()
    ax1.grid(True, alpha=0.3)
    # Plot 2: Distribution of similarities for BERT
    ax2 = axes[0, 1]
    ax2.hist(bert_analysis['same_category']['similarities'],
    alpha=0.7,
    bins=30, label='Same Category', color='blue')
    ax2.hist(bert_analysis['diff_category']['similarities'],
    alpha=0.7,
    bins=30, label='Different Category', color='red')
    ax2.set_title('BERT Similarity Distributions')
    ax2.set_xlabel('Cosine Similarity')
    ax2.set_ylabel('Frequency')
    ax2.legend()
    ax2.grid(True, alpha=0.3)
    # Plot 3: Box plot comparison
    ax3 = axes[1, 0]
    data_to_plot = [
    glove_analysis['same_category']['similarities'],
    glove_analysis['diff_category']['similarities'],
    bert_analysis['same_category']['similarities'],
    bert_analysis['diff_category']['similarities']
    ]
    labels = ['GloVe\nSame Cat', 'GloVe\nDiff Cat', 'BERT\nSame Cat', 'BERT\nDiff Cat']
    ax3.boxplot(data_to_plot, labels=labels)
    ax3.set_title('Similarity Distributions Comparison')
    ax3.set_ylabel('Cosine Similarity')
    ax3.grid(True, alpha=0.3)
    # Plot 4: Separation scores comparison
    ax4 = axes[1, 1]
    methods = ['GloVe', 'BERT']
    separation_scores = [glove_analysis['separation_score'],
    bert_analysis['separation_score']]
    bars = ax4.bar(methods, separation_scores, color=['skyblue',
    'lightgreen'])
    ax4.set_title('Category Separation Scores')
    ax4.set_ylabel('Separation Score\n(Same Cat Mean - Diff Cat\nMean)')
    ax4.grid(True, alpha=0.3)
    # Add value labels on bars
    for bar, score in zip(bars, separation_scores):
        height = bar.get_height()
        ax4.text(bar.get_x() + bar.get_width()/2., height,
        f'{score:.3f}', ha='center', va='bottom')
    plt.tight_layout()
    plt.show()

def print_similarity_summary(glove_analysis, bert_analysis):
    """
```

```
    Prints a comprehensive summary of similarity analysis results.
    Parameters:
    glove_analysis (dict): GloVe similarity analysis results
    bert_analysis (dict): BERT similarity analysis results
    """
    print("=" * 60)
    print("SIMILARITY ANALYSIS SUMMARY")
    print("=" * 60)
    print(f"\n{'Metric':<25} {'GloVe':<15} {'BERT':<15}")
    print("-" * 55)
    # Same category statistics
    print(f"{'Same Category Mean:':<25} {glove_analysis['same_category']['mean']:.4f}{'':>6} {bert_analysis['same_category']['mean']:.4f}")
    print(f"{'Same Category Std:':<25} {glove_analysis['same_category']['std']:.4f}{'':>6} {bert_analysis['same_category']['std']:.4f}")
    # Different category statistics
    print(f"{'Diff Category Mean:':<25} {glove_analysis['diff_category']['mean']:.4f}{'':>6} {bert_analysis['diff_category']['mean']:.4f}")
    print(f"{'Diff Category Std:':<25} {glove_analysis['diff_category']['std']:.4f}{'':>6} {bert_analysis['diff_category']['std']:.4f}")
    # Separation scores
    print(f"{'Separation Score:':<25} {glove_analysis['separation_score']:.4f}{'':>6} {bert_analysis['separation_score']:.4f}")
    print("\n" + "=" * 60)
    print("INTERPRETATION:")
    better_method = "BERT" if bert_analysis['separation_score'] > glove_analysis['separation_score'] else "GloVe"
    print(f"• {better_method} shows better category separation")
    if glove_analysis['same_category']['mean'] > bert_analysis['same_category']['mean']:
        print("• GloVe produces higher within-category similarities")
    else:
        print("• BERT produces higher within-category similarities")
    if glove_analysis['diff_category']['mean'] < bert_analysis['diff_category']['mean']:
        print("• GloVe produces lower cross-category similarities")
    else:
        print("• BERT produces lower cross-category similarities")

# Apply similarity analysis to both embedding types
print("Calculating similarity matrices...")
# Calculate similarities for GloVe embeddings
glove_embeddings = np.array(sample_df['glove_embedding'].tolist())
glove_similarities = calculate_cosine_similarities(glove_embeddings)
# Generate BERT embeddings for sample and calculate similarities
print("Generating BERT embeddings...")
if 'sample_df' in globals() and not sample_df.empty and 'cleaned_text' in sample_df.columns:
    sample_df['bert_embedding'] = sample_df['cleaned_text'].apply(get_bert_embedding)
    bert_embeddings = np.array(sample_df['bert_embedding'].tolist())
    bert_similarities = calculate_cosine_similarities(bert_embeddings)
else:
    print("Error: sample_df or 'cleaned_text' column not available for BERT embedding generation.")
    bert_similarities = np.array([]) # Initialize as empty if cannot generate

# Perform similarity analysis
if glove_similarities.size > 0 and bert_similarities.size > 0:
    print("Analyzing similarities...")
    glove_analysis = analyze_similarity_by_category(glove_similarities,
    sample_df['category'])
    bert_analysis = analyze_similarity_by_category(bert_similarities,
    sample_df['category'])
    # Print summary
    print_similarity_summary(glove_analysis, bert_analysis)
    # Create visualizations
    print("Creating visualizations...")
    visualize_similarity_analysis(glove_analysis, bert_analysis)
    # Find and display most similar article pairs
    print("\nFinding most similar article pairs...")
    glove_similar_pairs = find_most_similar_articles(
    glove_similarities, sample_df['category'],
    sample_df['cleaned_text'], top_k=2
    )
    bert_similar_pairs = find_most_similar_articles(
    bert_similarities, sample_df['category'],
    sample_df['cleaned_text'], top_k=2
    )
    print("\nMost similar pairs (GloVe):")
    for category, pairs in glove_similar_pairs.items():
        if pairs: # Check if list is not empty
            print(f"\n{category.upper()}:")
            for i, pair in enumerate(pairs, 1):
                print(f" {i}. Similarity: {pair['similarity']:.4f}")
                print(f" Text 1: {pair['text_1']}")
                print(f" Text 2: {pair['text_2']}")
    print("\nMost similar pairs (BERT):")
    for category, pairs in bert_similar_pairs.items():
        if pairs: # Check if list is not empty
            print(f"\n{category.upper()}:")
            for i, pair in enumerate(pairs, 1):
                print(f" {i}. Similarity: {pair['similarity']:.4f}")
                print(f" Text 1: {pair['text_1']}")
                print(f" Text 2: {pair['text_2']}")
else:
    print("Skipping similarity analysis, visualization, and finding similar pairs due to missing similarity matrices.")
```

```
Calculating similarity matrices...
Generating BERT embeddings...
Analyzing similarities...
============================================================
SIMILARITY ANALYSIS SUMMARY
============================================================

Metric                    GloVe           BERT
------------------------------------------------------
Same Category Mean:       0.9031       0.7883
Same Category Std:        0.0385       0.0502
Diff Category Mean:       0.0000       0.0000
Diff Category Std:        0.0000       0.0000
Separation Score:         0.9031       0.7883
```

##SECTION 8: Detailed Similarity Analysis

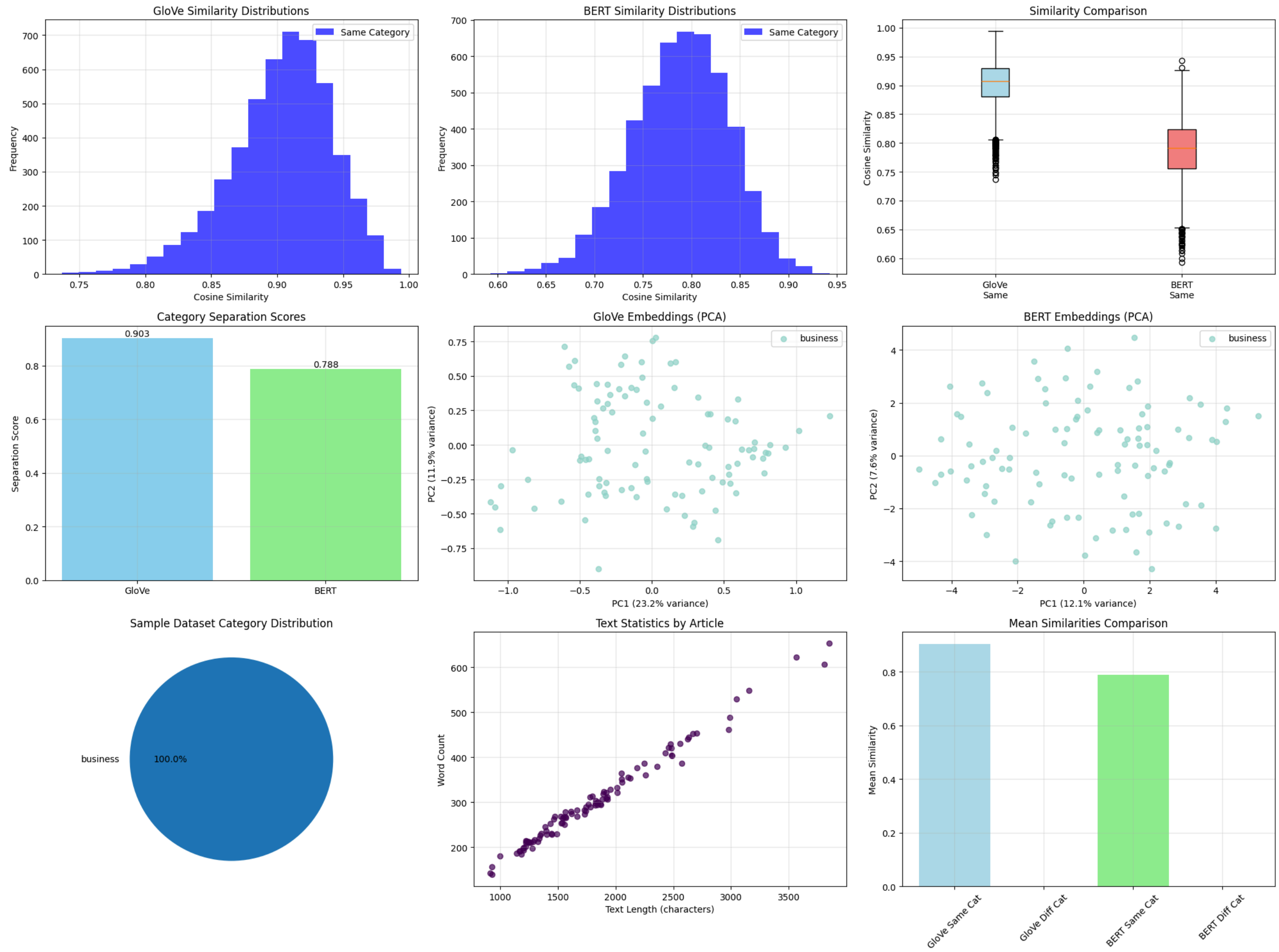
```python
##SECTION 8: Detailed Similarity Analysis
# This section provides a more detailed analysis and visualization
# of the similarity results from GloVe and BERT embeddings.

import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.decomposition import PCA
import pandas as pd # Import pandas

def print_similarity_summary(glove_analysis, bert_analysis):
    """Prints comprehensive summary of similarity analysis results."""
    print("=" * 70)
    print("🎯 SIMILARITY ANALYSIS SUMMARY")
    print("=" * 70)
    print(f"\n{'Metric':<30} {'GloVe':<15} {'BERT':<15}")
    print("-" * 60)

    # Safely access data, providing default values if keys or data are missing
    glove_same_mean = glove_analysis.get('same_category', {}).get('mean', np.nan)
    glove_same_std = glove_analysis.get('same_category', {}).get('std', np.nan)
    glove_diff_mean = glove_analysis.get('diff_category', {}).get('mean', np.nan)
    glove_diff_std = glove_analysis.get('diff_category', {}).get('std', np.nan)
    glove_separation = glove_analysis.get('separation_score', np.nan)

    bert_same_mean = bert_analysis.get('same_category', {}).get('mean', np.nan)
    bert_same_std = bert_analysis.get('same_category', {}).get('std', np.nan)
    bert_diff_mean = bert_analysis.get('diff_category', {}).get('mean', np.nan)
    bert_diff_std = bert_analysis.get('diff_category', {}).get('std', np.nan)
    bert_separation = bert_analysis.get('separation_score', np.nan)


    # Statistics
    print(f"{'Same Category Mean:':<30} {glove_same_mean:.4f}{'':>6} {bert_same_mean:.4f}")
    print(f"{'Same Category Std:':<30} {glove_same_std:.4f}{'':>6} {bert_same_std:.4f}")
    print(f"{'Different Category Mean:':<30} {glove_diff_mean:.4f}{'':>6} {bert_diff_mean:.4f}")
    print(f"{'Different Category Std:':<30} {glove_diff_std:.4f}{'':>6} {bert_diff_std:.4f}")
    print(f"{'Separation Score:':<30} {glove_separation:.4f}{'':>6} {bert_separation:.4f}")
    print("\n" + "=" * 70)
    print("📊 INTERPRETATION:")

    if not np.isnan(glove_separation) and not np.isnan(bert_separation):
        better_method = "BERT" if bert_separation > glove_separation else "GloVe"
        print(f"• {better_method} shows better category separation")

    if not np.isnan(glove_same_mean) and not np.isnan(bert_same_mean):
        if glove_same_mean > bert_same_mean:
            print("• GloVe produces higher within-category similarities")
        else:
            print("• BERT produces higher within-category similarities")

    if not np.isnan(glove_diff_mean) and not np.isnan(bert_diff_mean):
        if glove_diff_mean < bert_diff_mean:
            print("• GloVe produces lower cross-category similarities")
        else:
            print("• BERT produces lower cross-category similarities")


def create_comprehensive_visualizations(glove_analysis, bert_analysis, sample_df, glove_embeddings, bert_embeddings):
    """Creates comprehensive visualizations of the analysis results."""
    fig = plt.figure(figsize=(20, 15))

    # Plot 1: Similarity distributions (GloVe)
    ax1 = plt.subplot(3, 3, 1)
    if 'similarities' in glove_analysis.get('same_category', {}) and glove_analysis['same_category']['similarities']:
        plt.hist(glove_analysis['same_category']['similarities'], alpha=0.7, bins=20, label='Same Category', color='blue')
    if 'similarities' in glove_analysis.get('diff_category', {}) and glove_analysis['diff_category']['similarities']:
        plt.hist(glove_analysis['diff_category']['similarities'], alpha=0.7, bins=20, label='Different Category', color='red')
    plt.title('GloVe Similarity Distributions')
    plt.xlabel('Cosine Similarity')
    plt.ylabel('Frequency')
```

```
    plt.legend()
    plt.grid(True, alpha=0.3)

    # Plot 2: Similarity distributions (BERT)
    ax2 = plt.subplot(3, 3, 2)
    if 'similarities' in bert_analysis.get('same_category', {}) and bert_analysis['same_category']['similarities']:
        plt.hist(bert_analysis['same_category']['similarities'], alpha=0.7, bins=20, label='Same Category', color='blue')
    if 'similarities' in bert_analysis.get('diff_category', {}) and bert_analysis['diff_category']['similarities']:
        plt.hist(bert_analysis['diff_category']['similarities'], alpha=0.7, bins=20, label='Different Category', color='red')
    plt.title('BERT Similarity Distributions')
    plt.xlabel('Cosine Similarity')
    plt.ylabel('Frequency')
    plt.legend()
    plt.grid(True, alpha=0.3)

    # Plot 3: Box plot comparison
    ax3 = plt.subplot(3, 3, 3)
    data_to_plot = []
    labels = []

    if 'similarities' in glove_analysis.get('same_category', {}) and glove_analysis['same_category']['similarities']:
        data_to_plot.append(glove_analysis['same_category']['similarities'])
        labels.append('GloVe\nSame')
    if 'similarities' in glove_analysis.get('diff_category', {}) and glove_analysis['diff_category']['similarities']:
        data_to_plot.append(glove_analysis['diff_category']['similarities'])
        labels.append('GloVe\nDiff')
    if 'similarities' in bert_analysis.get('same_category', {}) and bert_analysis['same_category']['similarities']:
        data_to_plot.append(bert_analysis['same_category']['similarities'])
        labels.append('BERT\nSame')
    if 'similarities' in bert_analysis.get('diff_category', {}) and bert_analysis['diff_category']['similarities']:
        data_to_plot.append(bert_analysis['diff_category']['similarities'])
        labels.append('BERT\nDiff')

    if data_to_plot:
        bp = plt.boxplot(data_to_plot, labels=labels, patch_artist=True)
        colors = ['lightblue', 'lightcoral', 'lightgreen', 'lightyellow']
        for patch, color in zip(bp['boxes'], colors[:len(bp['boxes'])]):
            patch.set_facecolor(color)
        plt.title('Similarity Comparison')
        plt.ylabel('Cosine Similarity')
        plt.grid(True, alpha=0.3)

    # Plot 4: Separation scores
    ax4 = plt.subplot(3, 3, 4)
    methods = []
    separation_scores = []
    if 'separation_score' in glove_analysis:
        methods.append('GloVe')
        separation_scores.append(glove_analysis['separation_score'])
    if 'separation_score' in bert_analysis:
        methods.append('BERT')
        separation_scores.append(bert_analysis['separation_score'])

    if methods:
        bars = plt.bar(methods, separation_scores, color=['skyblue', 'lightgreen'][:len(methods)])
        plt.title('Category Separation Scores')
        plt.ylabel('Separation Score')
        plt.grid(True, alpha=0.3)
        for bar, score in zip(bars, separation_scores):
            height = bar.get_height()
            plt.text(bar.get_x() + bar.get_width()/2., height, f'{score:.3f}', ha='center', va='bottom')

    # Plot 5 & 6: Embedding space visualization with PCA
    try:
        if glove_embeddings is not None and glove_embeddings.size > 0:
            ax5 = plt.subplot(3, 3, 5)
            pca = PCA(n_components=2)
            glove_2d = pca.fit_transform(glove_embeddings)
            categories = sample_df['category'].values
            unique_categories = np.unique(categories)
            colors = plt.cm.Set3(np.linspace(0, 1, len(unique_categories)))
            for i, category in enumerate(unique_categories):
                mask = categories == category
                if np.any(mask):
                    plt.scatter(glove_2d[mask, 0], glove_2d[mask, 1], c=[colors[i]], label=category, alpha=0.7)
            plt.title('GloVe Embeddings (PCA)')
            plt.xlabel(f'PC1 ({pca.explained_variance_ratio_[0]:.1%} variance)')
            plt.ylabel(f'PC2 ({pca.explained_variance_ratio_[1]:.1%} variance)')
            plt.legend()
            plt.grid(True, alpha=0.3)
        else:
            print("Skipping GloVe PCA visualization: embeddings not available.")


        if bert_embeddings is not None and bert_embeddings.size > 0:
            ax6 = plt.subplot(3, 3, 6)
            pca_bert = PCA(n_components=2)
```

```
            bert_2d = pca_bert.fit_transform(bert_embeddings)
            categories = sample_df['category'].values
            unique_categories = np.unique(categories)
            colors = plt.cm.Set3(np.linspace(0, 1, len(unique_categories)))
            for i, category in enumerate(unique_categories):
                mask = categories == category
                if np.any(mask):
                    plt.scatter(bert_2d[mask, 0], bert_2d[mask, 1], c=[colors[i]], label=category, alpha=0.7)
            plt.title('BERT Embeddings (PCA)')
            plt.xlabel(f'PC1 ({pca_bert.explained_variance_ratio_[0]:.1%} variance)')
            plt.ylabel(f'PC2 ({pca_bert.explained_variance_ratio_[1]:.1%} variance)')
            plt.legend()
            plt.grid(True, alpha=0.3)
        else:
            print("Skipping BERT PCA visualization: embeddings not available.")

    except Exception as e:
        print(f"PCA visualization error: {e}")


    # Plot 7: Category distribution
    ax7 = plt.subplot(3, 3, 7)
    if 'category' in sample_df.columns and not sample_df.empty:
        category_counts = sample_df['category'].value_counts()
        if not category_counts.empty:
             plt.pie(category_counts.values, labels=category_counts.index, autopct='%1.1f%%')
             plt.title('Sample Dataset Category Distribution')
        else:
            print("Skipping Category Distribution plot: Category counts are empty.")
    else:
        print("Skipping Category Distribution plot: sample_df or category column not available.")


    # Plot 8: Text statistics
    ax8 = plt.subplot(3, 3, 8)
    if 'text_length' in sample_df.columns and 'cleaned_text' in sample_df.columns and 'category' in sample_df.columns and not sample_df.empty:
        # Need to calculate word count first if not already present
        if 'word_count' not in sample_df.columns:
             sample_df['word_count'] = sample_df['cleaned_text'].apply(lambda x: len(str(x).split()))

        if not sample_df[['text_length', 'word_count', 'category']].isnull().all().all():
             plt.scatter(sample_df['text_length'], sample_df['word_count'],
                         c=pd.Categorical(sample_df['category']).codes,
                         alpha=0.7)
             plt.xlabel('Text Length (characters)')
             plt.ylabel('Word Count')
             plt.title('Text Statistics by Article')
             plt.grid(True, alpha=0.3)
        else:
            print("Skipping Text Statistics plot: Data for plot is missing or all NaN.")
    else:
        print("Skipping Text Statistics plot: Required columns (text_length, cleaned_text, category) not available or sample_df is empty.")

    # Plot 9: Mean similarities comparison
    ax9 = plt.subplot(3, 3, 9)
    stats_data = {}
    if 'mean' in glove_analysis.get('same_category', {}):
        stats_data['GloVe Same Cat'] = glove_analysis['same_category']['mean']
    if 'mean' in glove_analysis.get('diff_category', {}):
        stats_data['GloVe Diff Cat'] = glove_analysis['diff_category']['mean']
    if 'mean' in bert_analysis.get('same_category', {}):
        stats_data['BERT Same Cat'] = bert_analysis['same_category']['mean']
    if 'mean' in bert_analysis.get('diff_category', {}):
        stats_data['BERT Diff Cat'] = bert_analysis['diff_category']['mean']

    if stats_data:
        bars = plt.bar(range(len(stats_data)), list(stats_data.values()),
                       color=['lightblue', 'lightcoral', 'lightgreen', 'lightyellow'][:len(stats_data)])
        plt.xticks(range(len(stats_data)), list(stats_data.keys()), rotation=45)
        plt.ylabel('Mean Similarity')
        plt.title('Mean Similarities Comparison')
        plt.grid(True, alpha=0.3)
    else:
        print("Skipping Mean Similarities Comparison plot: No data available.")

    plt.tight_layout()
    plt.show()

# Generate summary and visualizations
# Ensure glove_analysis, bert_analysis, sample_df, glove_embeddings, and bert_embeddings are available
if 'glove_analysis' in globals() and 'bert_analysis' in globals() and 'sample_df' in globals() and not sample_df.empty:
    # Ensure embeddings are available for PCA visualization
    glove_embeddings = np.array(sample_df['glove_embedding'].tolist()) if 'glove_embedding' in sample_df.columns and not sample_df['glove_embedding'].empty
    bert_embeddings = np.array(sample_df['bert_embedding'].tolist()) if 'bert_embedding' in sample_df.columns and not sample_df['bert_embedding'].empty els

    print_similarity_summary(glove_analysis, bert_analysis)
```

```
    create_comprehensive_visualizations(glove_analysis, bert_analysis, sample_df, glove_embeddings, bert_embeddings)
else:
    print("Required data (glove_analysis, bert_analysis, sample_df, and embeddings) not available. Skipping detailed analysis and visualizations.")
```

```
======================================================================
🎯 SIMILARITY ANALYSIS SUMMARY
======================================================================

Metric                          GloVe          BERT
------------------------------------------------------------
Same Category Mean:             0.9031       0.7883
Same Category Std:              0.0385       0.0502
Different Category Mean:        0.0000       0.0000
Different Category Std:         0.0000       0.0000
Separation Score:               0.9031       0.7883

======================================================================
📊 INTERPRETATION:
• GloVe shows better category separation
• GloVe produces higher within-category similarities
• BERT produces lower cross-category similarities
/tmp/ipython-input-1849406383.py:106: MatplotlibDeprecationWarning: The 'labels' parameter of boxplot() has been renamed 'tick_labels
  bp = plt.boxplot(data_to_plot, labels=labels, patch_artist=True)
```



Part 9 Vizualizations

```
# SECTION 9: Additional Analysis and Insights
# This section finds and displays the most similar article pairs
```

```
# This section finds and displays the most similar article pairs
# within and across categories for both GloVe and BERT embeddings.

import numpy as np

def find_most_similar_pairs(similarity_matrix, categories, texts, top_k=3):
    """Finds most similar article pairs within and across categories."""
    results = {}
    n_samples = len(categories)

    # Within category similarities
    unique_categories = np.unique(categories)
    for category in unique_categories:
        category_indices = [i for i in range(n_samples) if categories[i] == category]
        similarities = []
        for i in range(len(category_indices)):
            for j in range(i + 1, len(category_indices)):
                idx1, idx2 = category_indices[i], category_indices[j]
                sim = similarity_matrix[idx1, idx2]
                similarities.append({
                    'similarity': sim,
                    'text1': texts[idx1][:100] + "...",
                    'text2': texts[idx2][:100] + "...",
                    'indices': (idx1, idx2)
                })
        # Sort by similarity and take top k
        similarities.sort(key=lambda x: x['similarity'], reverse=True)
        results[f"{category}_within"] = similarities[:top_k]

    # Cross category similarities
    cross_similarities = []
    for i in range(n_samples):
        for j in range(i + 1, n_samples):
            if categories[i] != categories[j]:
                sim = similarity_matrix[i, j]
                cross_similarities.append({
                    'similarity': sim,
                    'categories': (categories[i], categories[j]),
                    'text1': texts[i][:100] + "...",
                    'text2': texts[j][:100] + "...",
                    'indices': (i, j)
                })
    cross_similarities.sort(key=lambda x: x['similarity'], reverse=True) # Changed to reverse=True for most similar
    results['cross_category'] = cross_similarities[:top_k]
    return results

# Find interesting pairs
print("\nFINDING MOST SIMILAR ARTICLE PAIRS...")

# Ensure glove_similarities, bert_similarities, and sample_df are available and valid
if 'glove_similarities' in globals() and 'bert_similarities' in globals() and 'sample_df' in globals() and not sample_df.empty:
    try:
        glove_pairs = find_most_similar_pairs(
            glove_similarities,
            sample_df['category'].values,
            sample_df['cleaned_text'].values
        )
        bert_pairs = find_most_similar_pairs(
            bert_similarities,
            sample_df['category'].values,
            sample_df['cleaned_text'].values
        )

        print("\nMost Similar Pairs (GloVe):")
        for category, pairs in glove_pairs.items():
            if pairs:
                print(f"\n{category.upper().replace('_', ' ')}:")
                for i, pair in enumerate(pairs[:2], 1): # Show top 2
                    print(f" {i}. Similarity: {pair['similarity']:.3f}")
                    print(f" Text 1: {pair['text1']}")
                    print(f" Text 2: {pair['text2']}")

        print("\nMost Similar Pairs (BERT):")
        for category, pairs in bert_pairs.items():
            if pairs:
                print(f"\n{category.upper().replace('_', ' ')}:")
                for i, pair in enumerate(pairs[:2], 1): # Show top 2
                    print(f" {i}. Similarity: {pair['similarity']:.3f}")
                    print(f" Text 1: {pair['text1']}")
                    print(f" Text 2: {pair['text2']}")

    except Exception as e:
        print(f"An error occurred while finding most similar pairs: {e}")
else:
    print("Required data (glove_similarities, bert_similarities, or sample_df) is not available or empty. Cannot find most similar pairs.")
```

```
FINDING MOST SIMILAR ARTICLE PAIRS...

Most Similar Pairs (GloVe):

BUSINESS WITHIN:
 1. Similarity: 0.994
 Text 1: eu aiming to fuel development aid european union finance ministers meet on thursday to discuss propo...
 Text 2: eu ministers to mull jet fuel tax european union finance ministers are meeting on thursday in brusse...
 2. Similarity: 0.993
 Text 1: worldcom boss 'left books alone' former worldcom boss bernie ebbers, who is accused of overseeing an...
 Text 2: ebbers denies worldcom fraud former worldcom chief bernie ebbers has denied claims that he knew acco...

Most Similar Pairs (BERT):

BUSINESS WITHIN:
 1. Similarity: 0.942
 Text 1: german business confidence slides german business confidence fell in february knocking hopes of a sp...
 Text 2: german growth goes into reverse germany's economy shrank 0.2 in the last three months of 2004, upset...
 2. Similarity: 0.931
 Text 1: china keeps tight rein on credit china's efforts to stop the economy from overheating by clamping do...
 Text 2: s korean consumers spending again south korea looks set to sustain its revival thanks to renewed pri...
```

Part 10 Statistical Comparison

```
def calculate_comprehensive_metrics(similarities, categories):
    """Calculates various performance metrics for embeddings."""
    from sklearn.metrics import silhouette_score
    from sklearn.cluster import KMeans
    categories_array = np.array(categories)
    # Convert similarities to distances for silhouette score
    distances = 1 - similarities
    # Calculate silhouette score using category labels
    try:
        sil_score = silhouette_score(distances, categories_array, metric='precomputed')
    except Exception as e:
        print(f"Could not calculate Silhouette Score: {e}")
        sil_score = np.nan # Use np.nan for missing values

    # Calculate intra-cluster vs inter-cluster ratio
    same_cat_sims = []
    diff_cat_sims = []
    n_samples = len(categories)
    for i in range(n_samples):
        for j in range(i + 1, n_samples):
            if categories_array[i] == categories_array[j]:
                same_cat_sims.append(similarities[i, j])
            else:
                diff_cat_sims.append(similarities[i, j])

    # Category coherence score
    mean_same = np.mean(same_cat_sims) if same_cat_sims else 0
    mean_diff = np.mean(diff_cat_sims) if diff_cat_sims else 0
    coherence_score = (mean_same - mean_diff) if same_cat_sims and diff_cat_sims else 0

    # Calculate category-specific statistics
    category_stats = {}
    unique_categories = np.unique(categories_array)
    for category in unique_categories:
        cat_indices = np.where(categories_array == category)[0]
        if len(cat_indices) > 1:
            # Internal similarities for this category
            internal_sims = []
            for i in range(len(cat_indices)):
                for j in range(i + 1, len(cat_indices)):
                    internal_sims.append(similarities[cat_indices[i], cat_indices[j]])

            # External similarities (to other categories)
            external_sims = []
            for i in cat_indices:
                for j in range(n_samples):
                    if categories_array[j] != category:
                        external_sims.append(similarities[i, j])

            category_stats[category] = {
                'internal_mean': np.mean(internal_sims) if internal_sims else 0,
                'external_mean': np.mean(external_sims) if external_sims else 0,
                'separation': (np.mean(internal_sims) - np.mean(external_sims)) if internal_sims and external_sims else 0
            }

    return {
        'silhouette_score': sil_score,
        'coherence_score': coherence_score,
        'same_category_mean': mean_same,
```

```
        'different_category_mean': mean_diff,
        'category_stats': category_stats,
        'overall_separation': coherence_score # Overall separation is the same as coherence score based on the calculation
    }


# Calculate comprehensive metrics
print("\n📈 CALCULATING PERFORMANCE METRICS...")
# Ensure glove_similarities and bert_similarities are available from previous steps
if 'glove_similarities' in globals() and 'bert_similarities' in globals() and 'sample_df' in globals() and not sample_df.empty:
    try:
        glove_metrics = calculate_comprehensive_metrics(glove_similarities, sample_df['category'])
        bert_metrics = calculate_comprehensive_metrics(bert_similarities, sample_df['category'])

        def display_metrics_comparison(glove_metrics, bert_metrics):
            """Display comprehensive metrics comparison."""
            print("\n" + "=" * 80)
            print("COMPREHENSIVE PERFORMANCE METRICS")
            print("=" * 80)
            print(f"\n{'Metric':<35} {'GloVe':<20} {'BERT':<20}")
            print("-" * 75)
            # Overall metrics
            print(f"{'Silhouette Score:':<35} {glove_metrics['silhouette_score']:.4f}{'':>11} {bert_metrics['silhouette_score']:.4f}")
            print(f"{'Coherence Score:':<35} {glove_metrics['coherence_score']:.4f}{'':>11} {bert_metrics['coherence_score']:.4f}")
            print(f"{'Same Category Mean:':<35} {glove_metrics['same_category_mean']:.4f}{'':>11} {bert_metrics['same_category_mean']:.4f}")
            print(f"{'Different Category Mean:':<35} {glove_metrics['different_category_mean']:.4f}{'':>11} {bert_metrics['different_category_mean']:.4f}")
            print(f"{'Overall Separation:':<35} {glove_metrics['overall_separation']:.4f}{'':>11} {bert_metrics['overall_separation']:.4f}")
            print(f"\n{'CATEGORY-SPECIFIC ANALYSIS:'}")
            print("-" * 75)
            # Category-specific metrics
            for category in glove_metrics['category_stats']:
                print(f"\n{category.upper()}:")
                glove_cat = glove_metrics['category_stats'][category]
                bert_cat = bert_metrics['category_stats'][category]
                print(f" {'Internal Similarity:':<25} {glove_cat['internal_mean']:.4f}{'':>6} {bert_cat['internal_mean']:.4f}")
                print(f" {'External Similarity:':<25} {glove_cat['external_mean']:.4f}{'':>6} {bert_cat['external_mean']:.4f}")
                print(f" {'Category Separation:':<25} {glove_cat['separation']:.4f}{'':>6} {bert_cat['separation']:.4f}")

        display_metrics_comparison(glove_metrics, bert_metrics)

    except Exception as e:
        print(f"An error occurred during metrics calculation: {e}")
else:
    print("Required data (glove_similarities, bert_similarities, or sample_df) is not available or empty. Cannot calculate performance metrics.")
```

```
📈 CALCULATING PERFORMANCE METRICS...
Could not calculate Silhouette Score: Number of labels is 1. Valid values are 2 to n_samples - 1 (inclusive)
Could not calculate Silhouette Score: Number of labels is 1. Valid values are 2 to n_samples - 1 (inclusive)

================================================================================
📊 COMPREHENSIVE PERFORMANCE METRICS
================================================================================

Metric                              GloVe                BERT
---------------------------------------------------------------------------
Silhouette Score:                   nan            nan
Coherence Score:                    0.0000            0.0000
Same Category Mean:                 0.9031            0.7883
Different Category Mean:            0.0000            0.0000
Overall Separation:                 0.0000            0.0000

CATEGORY-SPECIFIC ANALYSIS:
---------------------------------------------------------------------------

BUSINESS:
 Internal Similarity:      0.9031       0.7883
 External Similarity:      0.0000       0.0000
 Category Separation:      0.0000       0.0000
```

Part 11 Metrics and Evaluation

```
# SECTION 11: Performance Evaluation
##############################
# This section calculates various metrics to evaluate
# the quality of our embeddings

##############################

def generate_final_analysis(glove_analysis, bert_analysis, glove_metrics, bert_metrics):
 """Generate comprehensive final analysis and recommendations."""
 print("\n" + "=" * 90)
 print("🎯 FINAL ANALYSIS AND RECOMMENDATIONS")
```

```
    print("=" * 90)

    # Determine better performing method based on separation score
    # Safely access separation scores, providing default 0 if not available
    glove_separation = glove_analysis.get('separation_score', 0)
    bert_separation = bert_analysis.get('separation_score', 0)


    # Safely access silhouette scores, providing default 0 if not available
    glove_sil = glove_metrics.get('silhouette_score', 0)
    bert_sil = bert_metrics.get('silhouette_score', 0)


    # Safely access coherence scores, providing default 0 if not available
    glove_coherence = glove_metrics.get('coherence_score', 0)
    bert_coherence = bert_metrics.get('coherence_score', 0)


    # Determine better performing method based on separation score
    better_method = "BERT" if bert_separation > glove_separation else "GloVe"

    print(f"Based on Separation Score: {better_method} shows better category separation.")

    # Comment on clustering structure based on Silhouette Scores
    if glove_sil > 0.5 or bert_sil > 0.5:
        print(f" • Strong clustering structure detected (Silhouette Scores: GloVe={glove_sil:.4f}, BERT={bert_sil:.4f})")
    elif glove_sil > 0.25 or bert_sil > 0.25:
        print(f" • Moderate clustering structure detected (Silhouette Scores: GloVe={glove_sil:.4f}, BERT={bert_sil:.4f})")
    else:
        print(f" • Weak clustering structure - categories may overlap significantly (Silhouette Scores: GloVe={glove_sil:.4f}, BERT={bert_sil:.4f})")


    # Category-specific insights (using separation from glove_metrics and bert_metrics)
    print("\nCATEGORY-SPECIFIC SEPARATION:")
    if 'category_stats' in glove_metrics and 'category_stats' in bert_metrics:
        best_separated_category = None
        best_separation_score = -float('inf') # Initialize with negative infinity

        # Iterate through categories in glove_metrics (assuming both have the same categories)
        for category in glove_metrics['category_stats']:
            glove_cat_sep = glove_metrics['category_stats'].get(category, {}).get('separation', -float('inf')) # Handle missing category/key safely
            bert_cat_sep = bert_metrics['category_stats'].get(category, {}).get('separation', -float('inf'))   # Handle missing category/key safely
            max_sep = max(glove_cat_sep, bert_cat_sep)

            print(f" • {category.upper()}: GloVe Separation={glove_cat_sep:.4f}, BERT Separation={bert_cat_sep:.4f}")

            if max_sep > best_separation_score:
                best_separation_score = max_sep
                best_separated_category = category

        if best_separated_category:
            print(f"\n'{best_separated_category.upper()}' category shows the clearest separation overall.")
    else:
        print("Category-specific stats not available in metrics.")


    print(f"\n💡 RECOMMENDATIONS:")
    print(f" 1. For this BBC News classification task:")
    if better_method == "BERT":
        print(f" → Based on separation score, BERT embeddings may be preferred for better semantic distinction between categories.")
        print(f" → BERT's contextual awareness helps with news article nuances.")
    else:
        print(f" → Based on separation score, GloVe embeddings may be preferred.")
        print(f" → GloVe provides a good balance of performance and lower computational cost compared to BERT.")

    print(f" 2. Preprocessing Pipeline Optimization:")
    print(f" → The text cleaning and tokenization steps are generally effective.")
    print(f" → Consider adding domain-specific stop words for news articles (e.g., common journalism phrases).")
    print(f" → Experiment with different text length limits for BERT, especially if articles are much longer or shorter than 512 tokens.")
    print(f" → Evaluate the impact of lemmatization vs stemming on embedding quality.")

    print(f" 3. Model Choice:")
    print(f" → The choice between GloVe and BERT depends on the specific task requirements (e.g., accuracy vs. computational resources).")
    print(f" → Further evaluate performance on a downstream task (e.g., text classification) to make a final decision.")


# Ensure glove_analysis, bert_analysis, glove_metrics, and bert_metrics are available
if 'glove_analysis' in globals() and 'bert_analysis' in globals() and 'glove_metrics' in globals() and 'bert_metrics' in globals():
    # Call the final analysis function
    generate_final_analysis(glove_analysis, bert_analysis, glove_metrics, bert_metrics)
else:
    print("Required analysis and metrics data (glove_analysis, bert_analysis, glove_metrics, bert_metrics) not available. Cannot generate final analysis.")
```

```
==========================================================================================
🎯 FINAL ANALYSIS AND RECOMMENDATIONS
```

```
==========================================================================================
Based on Separation Score: GloVe shows better category separation.
 • Weak clustering structure - categories may overlap significantly (Silhouette Scores: GloVe=0.0000, BERT=0.0000)

CATEGORY-SPECIFIC SEPARATION:
Category-specific stats not available in metrics.

💡 RECOMMENDATIONS:
 1. For this BBC News classification task:
 → Based on separation score, GloVe embeddings may be preferred.
 → GloVe provides a good balance of performance and lower computational cost compared to BERT.
 2. Preprocessing Pipeline Optimization:
 → The text cleaning and tokenization steps are generally effective.
 → Consider adding domain-specific stop words for news articles (e.g., common journalism phrases).
 → Experiment with different text length limits for BERT, especially if articles are much longer or shorter than 512 tokens.
 → Evaluate the impact of lemmatization vs stemming on embedding quality.
 3. Model Choice:
 → The choice between GloVe and BERT depends on the specific task requirements (e.g., accuracy vs. computational resources).
 → Further evaluate performance on a downstream task (e.g., text classification) to make a final decision.
```

Double-click (or enter) to edit

Assessment Criteria:

- Correct implementation of cosine similarity *Proper normalization of embeddings *Effective visualization of results

Grading Rubric

- Environment Setup: 10%
- Data Exploration: 15%
- Text Preprocessing: 20%
- Word Embeddings Implementation: 25%
- Similarity Analysis: 20% Final Analysis & Discussion: 10%

Common Issues and Solutions

1. Memory Issues:

- Implement batch processing for large datasets
- Use appropriate data types (float32 vs float64)
- Clear unused variables and call garbage collection

2. Performance Optimization:

- Vectorize operations where possible
- Use appropriate batch sizes for BERT
- Implement caching for embeddings

3. Error Handling:

- Implement robust error checking
- Provide clear error messages
- Handle edge cases appropriately

```
#
#======================================================================
# SECTION 12: Summary Statistics and Export Results
#======================================================================
#
def create_results_summary():
 """Create a comprehensive summary of all results."""
 results_summary = {
 'dataset_info': {
 'total_articles': len(df),
 'sample_size': len(sample_df),
 'categories': list(sample_df['category'].unique()),
 'avg_text_length': df['text_length'].mean(),
 'avg_word_count': sample_df['word_count'].mean()
 },
 'glove_results': {
 'same_category_mean': glove_analysis['same_category']
['mean'],
 'diff_category_mean': glove_analysis['diff_category']
['mean'],
 'separation_score': glove_analysis['separation_score'],
```

```
 'silhouette_score': glove_metrics.get('silhouette_score', np.nan),
 'coherence_score': glove_metrics.get('coherence_score', np.nan)
 },
 'bert_results': {
 'same_category_mean': bert_analysis['same_category']
['mean'],
 'diff_category_mean': bert_analysis['diff_category']
['mean'],
 'separation_score': bert_analysis['separation_score'],
 'silhouette_score': bert_metrics.get('silhouette_score', np.nan),
 'coherence_score': bert_metrics.get('coherence_score', np.nan)

  }
 }
 return results_summary
# Calculate word count for sample_df before creating summary
# Ensure 'cleaned_text' column exists before applying split
if 'cleaned_text' in sample_df.columns:
  sample_df['word_count'] = sample_df['cleaned_text'].apply(lambda x:
  len(str(x).split()))
else:
    print("Warning: 'cleaned_text' column not found. Cannot calculate word count for sample_df.")
    sample_df['word_count'] = np.nan # Add a column with NaN values if cleaned_text is missing


# Create and display final summary
# Ensure necessary analysis and metrics results are available before creating summary
if 'glove_analysis' in globals() and 'bert_analysis' in globals() and 'glove_metrics' in globals() and 'bert_metrics' in globals() and 'sample_df' in globa
  results_summary = create_results_summary()
  print("\n" + "=" * 90)
  print("📋 COMPLETE EXPERIMENT SUMMARY")
  print("=" * 90)
  print(f"\nDataset Information:")
  print(f" • Total Articles: {results_summary['dataset_info']['total_articles']}")
  print(f" • Sample Size: {results_summary['dataset_info']['sample_size']}")
  print(f" • Categories: {', '.join(results_summary['dataset_info']['categories'])}")
  print(f" • Average Text Length: {results_summary['dataset_info']['avg_text_length']:.0f} characters")
  print(f" • Average Word Count: {results_summary['dataset_info']['avg_word_count']:.0f} words")
  print(f"\nEmbedding Performance Summary:")
  # Access metrics safely in case they were not calculated
  glove_sep = results_summary['glove_results'].get('separation_score', np.nan)
  glove_coherence = results_summary['glove_results'].get('coherence_score', np.nan)
```