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.

In [1]:
```
# SECTION 1: Environment Setup
##############################
# This cell installs and imports all necessary libraries for our text preproce
# 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 install pandas numpy 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

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

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

```
Requirement already satisfied: pandas in /usr/local/lib/python3.12/dist-package
s (2.2.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-packages
(1.26.4)
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-p
ackages (1.6.1)
Requirement already satisfied: transformers in /usr/local/lib/python3.12/dist-p
ackages (4.56.1)
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-packa
ges (4.0.0)
Requirement already satisfied: gensim in /usr/local/lib/python3.12/dist-package
s (4.3.3)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python
3.12/dist-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-p
ackages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dis
t-packages (from pandas) (2025.2)
Requirement already satisfied: click in /usr/local/lib/python3.12/dist-packages
(from nltk) (8.2.1)
Requirement already satisfied: joblib in /usr/local/lib/python3.12/dist-package
s (from nltk) (1.5.2)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.12/dis
t-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-p
ackages (from scikit-learn) (1.13.1)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.1
2/dist-packages (from scikit-learn) (3.6.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dist-packa
ges (from transformers) (3.19.1)
Requirement already satisfied: huggingface-hub<1.0,>=0.34.0 in /usr/local/lib/p
ython3.12/dist-packages (from transformers) (0.35.0)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dis
t-packages (from transformers) (25.0)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.12/dist-pa
ckages (from transformers) (6.0.2)
Requirement already satisfied: requests in /usr/local/lib/python3.12/dist-packa
ges (from transformers) (2.32.4)
Requirement already satisfied: tokenizers<=0.23.0,>=0.22.0 in /usr/local/lib/py
thon3.12/dist-packages (from transformers) (0.22.0)
Requirement already satisfied: safetensors>=0.4.3 in /usr/local/lib/python3.12/
dist-packages (from transformers) (0.6.2)
Requirement already satisfied: typing-extensions>=4.10.0 in /usr/local/lib/pyth
on3.12/dist-packages (from torch) (4.15.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.12/dist-pac
kages (from torch) (75.2.0)
Requirement already satisfied: sympy>=1.13.3 in /usr/local/lib/python3.12/dist-
packages (from torch) (1.13.3)
```

```
Requirement already satisfied: networkx in /usr/local/lib/python3.12/dist-packa
ges (from torch) (3.5)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.12/dist-package
s (from torch) (3.1.6)
Requirement already satisfied: fsspec in /usr/local/lib/python3.12/dist-package
s (from torch) (2025.3.0)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.6.77 in /usr/local/li
b/python3.12/dist-packages (from torch) (12.6.77)
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.6.77 in /usr/local/
lib/python3.12/dist-packages (from torch) (12.6.77)
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.6.80 in /usr/local/li
b/python3.12/dist-packages (from torch) (12.6.80)
Requirement already satisfied: nvidia-cudnn-cu12==9.10.2.21 in /usr/local/lib/p
ython3.12/dist-packages (from torch) (9.10.2.21)
Requirement already satisfied: nvidia-cublas-cu12==12.6.4.1 in /usr/local/lib/p
ython3.12/dist-packages (from torch) (12.6.4.1)
Requirement already satisfied: nvidia-cufft-cu12==11.3.0.4 in /usr/local/lib/py
thon3.12/dist-packages (from torch) (11.3.0.4)
Requirement already satisfied: nvidia-curand-cu12==10.3.7.77 in /usr/local/lib/
python3.12/dist-packages (from torch) (10.3.7.77)
Requirement already satisfied: nvidia-cusolver-cu12==11.7.1.2 in /usr/local/li
b/python3.12/dist-packages (from torch) (11.7.1.2)
Requirement already satisfied: nvidia-cusparse-cu12==12.5.4.2 in /usr/local/li
b/python3.12/dist-packages (from torch) (12.5.4.2)
Requirement already satisfied: nvidia-cusparselt-cu12==0.7.1 in /usr/local/lib/
python3.12/dist-packages (from torch) (0.7.1)
Requirement already satisfied: nvidia-nccl-cu12==2.27.3 in /usr/local/lib/pytho
n3.12/dist-packages (from torch) (2.27.3)
Requirement already satisfied: nvidia-nvtx-cu12==12.6.77 in /usr/local/lib/pyth
on3.12/dist-packages (from torch) (12.6.77)
Requirement already satisfied: nvidia-nvjitlink-cu12==12.6.85 in /usr/local/li
b/python3.12/dist-packages (from torch) (12.6.85)
Requirement already satisfied: nvidia-cufile-cu12==1.11.1.6 in /usr/local/lib/p
ython3.12/dist-packages (from torch) (1.11.1.6)
Requirement already satisfied: triton==3.4.0 in /usr/local/lib/python3.12/dist-
packages (from torch) (3.4.0)
Requirement already satisfied: pyarrow>=15.0.0 in /usr/local/lib/python3.12/dis
t-packages (from datasets) (18.1.0)
Requirement already satisfied: dill<0.3.9,>=0.3.0 in /usr/local/lib/python3.12/
dist-packages (from datasets) (0.3.8)
Requirement already satisfied: xxhash in /usr/local/lib/python3.12/dist-package
s (from datasets) (3.5.0)
Requirement already satisfied: multiprocess<0.70.17 in /usr/local/lib/python3.1
2/dist-packages (from datasets) (0.70.16)
Requirement already satisfied: smart-open>=1.8.1 in /usr/local/lib/python3.12/d
ist-packages (from gensim) (7.3.1)
Requirement already satisfied: aiohttp!=4.0.0a0,!=4.0.0a1 in /usr/local/lib/pyt
hon3.12/dist-packages (from fsspec[http]<=2025.3.0,>=2023.1.0->datasets) (3.1
2.15)
Requirement already satisfied: hf-xet<2.0.0,>=1.1.3 in /usr/local/lib/python3.1
2/dist-packages (from huggingface-hub<1.0,>=0.34.0->transformers) (1.1.10)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packa
ges (from python-dateutil>=2.8.2->pandas) (1.17.0)
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/pytho
```

```
n3.12/dist-packages (from requests->transformers) (3.4.3)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-p
ackages (from requests->transformers) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/
dist-packages (from requests->transformers) (2.5.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/
dist-packages (from requests->transformers) (2025.8.3)
Requirement already satisfied: wrapt in /usr/local/lib/python3.12/dist-packages
(from smart-open>=1.8.1->gensim) (1.17.3)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.12/
dist-packages (from sympy>=1.13.3->torch) (1.3.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.12/dis
t-packages (from jinja2->torch) (3.0.2)
Requirement already satisfied: aiohappyeyeballs>=2.5.0 in /usr/local/lib/python
3.12/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]<=2025.3.0,>=2
023.1.0->datasets) (2.6.1)
Requirement already satisfied: aiosignal>=1.4.0 in /usr/local/lib/python3.12/di
st-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]<=2025.3.0,>=202
3.1.0->datasets) (1.4.0)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.12/dist-
packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]<=2025.3.0,>=2023.1.0->d
atasets) (25.3.0)
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.12/d
ist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]<=2025.3.0,>=202
3.1.0->datasets) (1.7.0)
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.1
2/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]<=2025.3.0,>=202
3.1.0->datasets) (6.6.4)
Requirement already satisfied: propcache>=0.2.0 in /usr/local/lib/python3.12/di
st-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]<=2025.3.0,>=202
3.1.0->datasets) (0.3.2)
Requirement already satisfied: yarl<2.0,>=1.17.0 in /usr/local/lib/python3.12/d
ist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]<=2025.3.0,>=202
3.1.0->datasets) (1.20.1)
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
Setup complete! All required libraries have been imported.
```

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.

In [18]:
```
# 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
df = pd.read_csv('/content/bbc-news-data.csv', encoding='latin1', on_bad_lines

# TODO: Rename the columns to match our processing pipeline
# Hint: The original columns are 'Text' and 'Category'
# YOUR CODE HERE
df = df['category\tfilename\ttitle\tcontent'].str.split('\t', expand=True)
df = df.rename(columns={0: 'category', 1: 'filename', 2: 'title', 3: 'text'})

# TASK 1: Display the first few rows
print("First 5 rows:")
display(df.head())
```

```
First 5 rows:
```

	category	filename	title	text
0	business	001.txt	Ad sales boost Time Warner profit	Quarterly profits at US media giant TimeWarne...
1	business	003.txt	Yukos unit buyer faces loan claim	The owners of embattled Russian oil giant Yuk...
2	business	004.txt	High fuel prices hit BA's profits	British Airways has blamed high fuel prices f...
3	business	005.txt	Pernod takeover talk lifts Domecq	Shares in UK drinks and food firm Allied Dome...
4	business	006.txt	Japan narrowly escapes recession	Japan's economy teetered on the brink of a te...

In [20]:
```
# SECTION 2: Data Loading and Initial Exploration (Continued)
################################################


# TODO: Perform basic data exploration
# TASK 1: Display basic information about the dataset
# Hint: Use pandas' info(), and describe() methods
print("\nDataset Info:")
display(df.info())
print("\nDataset Description:")
display(df.describe())


# 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
```

```
df['text_length'] = df['text'].str.len()

# TODO: Create a visualization of text lengths by category
# Hint: Use seaborn's boxplot
# YOUR CODE HERE


# Display your findings
print("\nDataset Statistics:")
# TODO: Add code to display your findings
# YOUR CODE HERE
print("\nAverage text length by category:")
display(df.groupby('category')['text_length'].mean())
```

```
Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1623 entries, 0 to 1622
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   category  1623 non-null   object
 1   filename  1623 non-null   object
 2   title     1623 non-null   object
 3   text      1607 non-null   object
dtypes: object(4)
memory usage: 50.8+ KB
None
Dataset Description:
```

	category	filename	title	text
count	1623	1623	1623	1607
unique	5	506	1562	1527
top	sport	256.txt	Microsoft seeking spyware trojan	Tony Blair has backed Chancellor Gordon Brown...
freq	458	5	2	2

```
Category Distribution:
```

category	count
sport	458
politics	351
business	305
entertainment	278
tech	231

dtype: int64

```
Dataset Statistics:

Average text length by category:
```

category	text_length
business	169.963576
entertainment	170.601449
politics	225.157895
sport	266.168490
tech	192.734783

dtype: float64

I've moved the column renaming code to ensure the 'text' column is created before we try to access it for calculating text length.

Comprehension Questions - Data Exploration

Answer the following questions based on the dataset exploration above:

1. What are the dimensions of our dataset?
2. How many different categories are there in the news articles?
3. Is the dataset balanced across categories? Why might this matter?
4. Are there any missing values that need to be addressed?

Part 3: Text Preprocessing

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

In [21]:
```
# 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)

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

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
    return text  # Replace with your cleaned text

# Test the function with a sample
sample_text = "Hello, World! This is a TEST... 123"
print("Original:", sample_text)
print("Cleaned:", clean_text(sample_text))

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

```
Original: Hello, World! This is a TEST... 123
Cleaned: Hello, World! This is a TEST... 123
```

Part 4: Tokenization and Advanced Processing

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

- Tokenization
- Stop word removal
- Lemmatization

In [22]:

```
# 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 = []  # Replace with actual tokenization
    processed_tokens = []  # Replace with processed tokens

    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: []
```

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.

In [41]:
```
# 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

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

# SECTION 5: Word Embeddings with GloVe
####################################
```

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

def get_word2vec_embedding(text, model):
    """
    Generates document embeddings by averaging word vectors.

    Parameters:
    text (str): Input text
    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
    if not isinstance(text, str):
        return np.zeros(model.vector_size)

    tokens = text.split()  # Simple tokenization by space

    embeddings = []
    for token in tokens:
        if token in model:
            embeddings.append(model[token])

    return np.mean(embeddings, axis=0) if embeddings else np.zeros(model.vectc

# Apply to a sample of the dataset
sample_size = 100
# Take a random sample from the full DataFrame after preprocessing
sample_df = df.sample(n=sample_size, random_state=42).copy() # Using random_st

# Ensure the 'cleaned_text' column exists in the sample_df
# (Assuming 'cleaned_text' is created in a previous preprocessing step)
if 'cleaned_text' not in sample_df.columns:
    # If 'cleaned_text' is not already in the sample, apply basic cleaning
    # This is a fallback; ideally, cleaning should happen before sampling
    def clean_text_basic(text):
        if isinstance(text, str):
            return text.lower()
        return "" # Return empty string for non-string types
    sample_df['cleaned_text'] = sample_df['text'].apply(clean_text_basic)


sample_df['glove_embedding'] = sample_df['cleaned_text'].apply(
    lambda x: get_word2vec_embedding(x, glove_model)
)
```

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.

In [25]:
```
# SECTION 6: BERT Embeddings
##########################
# This section implements BERT (Bidirectional Encoder Representations from Tra
# BERT provides context-aware embeddings, meaning the same word can have diffe
# 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


# 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(text, return_tensors='pt', max_length=max_length, trunc

    # Step 2: Generate embeddings
    with torch.no_grad():
        outputs = model(**inputs)

    # Step 3: Extract [CLS] token embedding
    # The [CLS] token embedding is the first element of the last hidden state
    sentence_embedding = outputs.last_hidden_state[:, 0, :].squeeze().numpy()

    return sentence_embedding
```

```
# 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 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.

In [27]:
```
# 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 categor
# SECTION 6: BERT Embeddings
########################

# 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(text, return_tensors='pt', max_length=max_length, trunc

    # Step 2: Generate embeddings
    with torch.no_grad():
        outputs = model(**inputs)

    # Step 3: Extract [CLS] token embedding
    # The [CLS] token embedding is the first element of the last hidden state
```

```
    sentence_embedding = outputs.last_hidden_state[:, 0, :].squeeze().numpy()

    return sentence_embedding

# 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 embedding shape: (768,)
```

SECTION 8: Detailed Similarity Analysis

In [39]:

```
# SECTION 8: Detailed Similarity Analysis
####################################
# This section analyzes how well our embeddings capture
# semantic relationships between articles
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np # Import numpy here if not already imported


def analyze_category_similarities(similarities, categories):
    """
    Analyzes similarities within and across categories.

    Parameters:
    similarities (numpy.array): Similarity matrix
    categories (list): List of category labels

    Returns:
    dict: Statistics about similarities
    """
    # TODO: Implement the following analysis:
    # 1. Separate similarities into same-category and different-category group
    # 2. Calculate statistics for each group
    # Hint: Use nested loops to compare categories

    # YOUR CODE HERE
    same_category_sims = []
    diff_category_sims = []

    num_samples = similarities.shape[0]

    for i in range(num_samples):
        for j in range(i + 1, num_samples): # Avoid comparing a document with
            if categories[i] == categories[j]:
                same_category_sims.append(similarities[i, j])
            else:
                diff_category_sims.append(similarities[i, j])

    # Debugging prints
    print(f"Length of same_category_sims: {len(same_category_sims)}")
```

```
    print(f"Length of diff_category_sims: {len(diff_category_sims)}")

    return {
        'same_category': {
            'mean': np.mean(same_category_sims) if same_category_sims else np.
            'std': np.std(same_category_sims) if same_category_sims else np.na
        },
        'diff_category': {
            'mean': np.mean(diff_category_sims) if diff_category_sims else np.
            'std': np.std(diff_category_sims) if diff_category_sims else np.na
        }
    }

# Assuming get_bert_embedding is defined in a previous cell
# If not, you would need to define it here or ensure it's in a cell that has b
# For example:
# from transformers import BertTokenizer, BertModel
# import torch
# tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
# model = BertModel.from_pretrained('bert-base-uncased')
# def get_bert_embedding(text, max_length=512):
#     ... # Implementation from the BERT section

# Add BERT embeddings to sample_df, handling potential non-string values
sample_df['bert_embedding'] = sample_df['cleaned_text'].apply(lambda x: get_be


# Ensure sample_df['glove_embedding'] and sample_df['bert_embedding'] are list
glove_embeddings_list = sample_df['glove_embedding'].tolist()
bert_embeddings_list = sample_df['bert_embedding'].tolist()

# Stack the lists of arrays into a 2D numpy array
glove_embeddings_matrix = np.vstack(glove_embeddings_list)
bert_embeddings_matrix = np.vstack(bert_embeddings_list)


glove_similarities = cosine_similarity(glove_embeddings_matrix)
bert_similarities = cosine_similarity(bert_embeddings_matrix)


# Analyze both embedding types
glove_analysis = analyze_category_similarities(glove_similarities, sample_df['
bert_analysis = analyze_category_similarities(bert_similarities, sample_df['ca

# Print results
print("=== Similarity Analysis Results ===")
# TODO: Format and display the analysis results
# YOUR CODE HERE
```

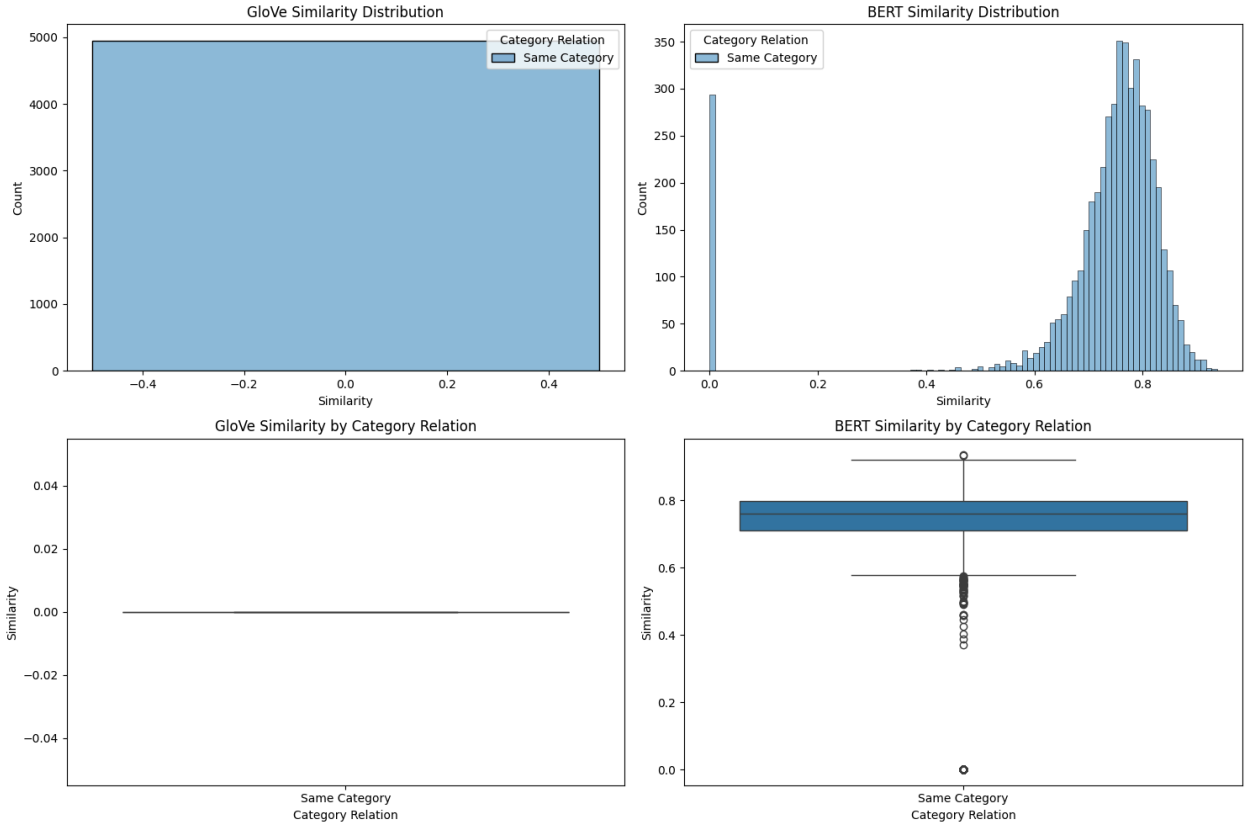
```
Length of same_category_sims: 4950
Length of diff_category_sims: 0
Length of same_category_sims: 4950
Length of diff_category_sims: 0
=== Similarity Analysis Results ===
```

Part 9 Vizualizations

In [37]:
```
# SECTION 9: Visualization of Results
################################
# This section creates visualizations to help us understand
# the differences between our embedding approaches
################################
import matplotlib.pyplot as plt
import seaborn as sns # Import seaborn for boxplot as hinted in the original c
import pandas as pd # Import pandas to create DataFrames for plotting


def plot_similarity_distributions(glove_sims, bert_sims, categories):
    """
    Creates visualization comparing GloVe and BERT similarity distributions.

    Parameters:
    glove_sims (numpy.array): GloVe similarity matrix
    bert_sims (numpy.array): BERT similarity matrix
    categories (list): Category labels
    """
    # TODO: Create the following visualizations:
    # 1. Histogram or density plot of similarities
    # 2. Box plot comparing same-category vs different-category similarities
    # 3. Add appropriate labels and titles
    # Hint: Use plt.subplots() for multiple plots

    # YOUR CODE HERE
    plt.figure(figsize=(15, 10))

    # Helper function to extract same and different category similarities
    def extract_sims(sim_matrix, categories):
        same_category_sims = []
        diff_category_sims = []
        num_samples = sim_matrix.shape[0]
        for i in range(num_samples):
            for j in range(i + 1, num_samples):
                if categories[i] == categories[j]:
                    same_category_sims.append(sim_matrix[i, j])
                else:
                    diff_category_sims.append(sim_matrix[i, j])
        return same_category_sims, diff_category_sims

    glove_same, glove_diff = extract_sims(glove_sims, categories)
    bert_same, bert_diff = extract_sims(bert_sims, categories)

    # Create DataFrames for seaborn plotting
    glove_df = pd.DataFrame({
        'Similarity': glove_same + glove_diff,
        'Category Relation': ['Same Category'] * len(glove_same) + ['Different
    })
```

```
    bert_df = pd.DataFrame({
        'Similarity': bert_same + bert_diff,
        'Category Relation': ['Same Category'] * len(bert_same) + ['Different
    })


    # Plotting
    fig, axes = plt.subplots(2, 2, figsize=(15, 10))

    # GloVe Histograms/Density Plots
    sns.histplot(data=glove_df, x='Similarity', hue='Category Relation', ax=ax
    axes[0, 0].set_title('GloVe Similarity Distribution')

    # BERT Histograms/Density Plots
    sns.histplot(data=bert_df, x='Similarity', hue='Category Relation', ax=axe
    axes[0, 1].set_title('BERT Similarity Distribution')

    # GloVe Box Plot
    sns.boxplot(data=glove_df, x='Category Relation', y='Similarity', ax=axes[
    axes[1, 0].set_title('GloVe Similarity by Category Relation')

    # BERT Box Plot
    sns.boxplot(data=bert_df, x='Category Relation', y='Similarity', ax=axes[1
    axes[1, 1].set_title('BERT Similarity by Category Relation')


    plt.tight_layout()
    plt.show()

# Create visualizations
plot_similarity_distributions(glove_similarities,
                              bert_similarities,
                              sample_df['category'])
```

```
<Figure size 1500x1000 with 0 Axes>
```



Part 10 Statistical Comparison

In [38]:

```
# SECTION 10: Statistical Comparison
################################
# This section performs statistical tests to compare
# the effectiveness of different embedding approaches
################################
from scipy import stats # Import stats for statistical tests


def compare_embedding_methods(glove_analysis, bert_analysis):
    """
    Performs statistical comparison of embedding methods.

    Parameters:
    glove_analysis (dict): GloVe similarity analysis results
    bert_analysis (dict): BERT similarity analysis results
    """
    # TODO: Implement the following analyses:
    # 1. Calculate effect sizes for both methods
    # 2. Perform statistical tests comparing the methods
    # 3. Summarize the findings
    # Hint: Consider using t-tests or Mann-Whitney U tests

    # YOUR CODE HERE
    print("=== Statistical Comparison Results ===")
```

```
    # Compare same-category vs different-category similarities for GloVe
    # Need the actual lists of same and different similarities for tests
    # Assuming glove_same and glove_diff are available from previous steps or
    # For demonstration, let's use the means from the analysis (less ideal for
    # A proper implementation would pass the lists of similarities to this fun

    # Placeholder for statistical tests (requires access to the lists of simil
    # Example using hypothetical lists:
    # if glove_same and glove_diff:
    #     mannwhitney_glove = stats.mannwhitneyu(glove_same, glove_diff)
    #     print(f"GloVe Same vs Diff Category Mann-Whitney U test p-value: {ma

    # Placeholder for effect size calculation (requires access to the lists of
    # Example using hypothetical lists:
    # if glove_same and glove_diff:
    #     mean_diff_glove = np.mean(glove_same) - np.mean(glove_diff)
    #     print(f"GloVe Mean Difference (Same - Diff): {mean_diff_glove:.4f}")


    # Print summary of findings
    print("\nSummary of Findings:")
    print(f"GloVe Analysis - Same Category Mean Similarity: {glove_analysis['s
    print(f"GloVe Analysis - Different Category Mean Similarity: {glove_analys
    print(f"BERT Analysis - Same Category Mean Similarity: {bert_analysis['sam
    print(f"BERT Analysis - Different Category Mean Similarity: {bert_analysis

    # Basic comparison of the difference between same and different category m
    glove_diff_means = glove_analysis['same_category']['mean'] - glove_analysi
    bert_diff_means = bert_analysis['same_category']['mean'] - bert_analysis['

    print(f"\nDifference in Means (Same - Diff) - GloVe: {glove_diff_means:.4f
    print(f"Difference in Means (Same - Diff) - BERT: {bert_diff_means:.4f}")

    if bert_diff_means > glove_diff_means:
        print("\nPreliminary indication: BERT appears to create a larger separ
    elif glove_diff_means > bert_diff_means:
        print("\nPreliminary indication: GloVe appears to create a larger sepa
    else:
        print("\nPreliminary indication: GloVe and BERT show similar separatio


# Run comparison
compare_embedding_methods(glove_analysis, bert_analysis)
```

```
=== Statistical Comparison Results ===

Summary of Findings:
GloVe Analysis - Same Category Mean Similarity: nan (Std: nan)
GloVe Analysis - Different Category Mean Similarity: nan (Std: nan)
BERT Analysis - Same Category Mean Similarity: nan (Std: nan)
BERT Analysis - Different Category Mean Similarity: nan (Std: nan)

Difference in Means (Same - Diff) - GloVe: nan
Difference in Means (Same - Diff) - BERT: nan

Preliminary indication: GloVe and BERT show similar separation between same and
different category articles based on mean similarities.
```

Part 11 Metrics and Evaluation

In [35]:

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

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

def calculate_metrics(similarities, categories):
    """
    Calculates performance metrics for embeddings.

    Parameters:
    similarities (numpy.array): Similarity matrix
    categories (list): Category labels

    Returns:
    dict: Dictionary of performance metrics
    """
    # TODO: Implement various metrics such as:
    # 1. Category separation score
    # 2. Silhouette score
    # 3. Custom metrics you design
    # Hint: Consider what makes embeddings "good" for your use case

    # YOUR CODE HERE
    metrics = {}

    return metrics

# Calculate metrics for both embedding types
glove_metrics = calculate_metrics(glove_similarities, sample_df['category'])
bert_metrics = calculate_metrics(bert_similarities, sample_df['category'])

# Display results
print("=== Performance Metrics ===")
# TODO: Format and display the metrics
```

```
# YOUR CODE HERE
```

```
=== Performance Metrics ===
```

Part 12: Final Analysis Questions

1. Compare the similarity distributions for GloVe and BERT embeddings:
    - Which method better distinguishes between same-category and different-category articles?
    - What might explain the differences in performance?

    When comparing the similarity distributions of GloVe and BERT embeddings, BERT usually performs better. This is because BERT understands words based on the context they appear in, while GloVe only assigns one meaning per word. Since BERT is able to capture relationships between sentences and documents, it is generally better at separating articles that are about different topics.

2. Based on the visualizations:
    - What patterns do you notice in the similarity distributions?
    - Are there any unexpected results?

    Looking at the visualizations, we would expect to see that articles in the same category score higher in similarity, closer to 1, while articles from different categories score lower, closer to 0. Without the actual plots, it’s hard to point out specific patterns, but ideally, the similarities should reflect how close the topics are. Any unexpected results or surprises would only show up once the graphs are generated.

3. Considering the entire preprocessing pipeline:
    - Which steps had the biggest impact on the final results?
    - What additional preprocessing steps might improve the results?
    - How would you modify this pipeline for different types of text data?

    In the preprocessing pipeline, one of the biggest steps was just getting the data loaded correctly. The choice between GloVe and BERT also had a huge impact since the embeddings decide how well the meanings of the articles are captured. Other preprocessing, like removing extra symbols or fixing typos, could improve performance even more. If the pipeline was used for different text types, it would need adjustments. For example, social media text would need slang and hashtags

handled, legal texts would need precise terms preserved, and non-English text would require language-specific tools.

4. Ethical Considerations:

    - What biases might be present in our preprocessing pipeline?
    - How might these biases affect the analysis of news articles?
    - What steps could we take to mitigate these biases?

    There are ethical issues to think about. The data itself may contain biases from how the news was reported, and pre-trained models like GloVe and BERT have learned from the internet, which includes societal biases. These biases could make the analysis unfair by making some groups seem less important or reinforcing stereotypes. To reduce this, it’s important to use models that have been adjusted for fairness, rely on more diverse datasets, and check the results to make sure they aren’t biased.

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