MSc in AI – Assessment Cover Sheet

Course Code and Title:IT9002– Natural Language Processing

Assessment Title: Individual Project

Learning Outcomes:

LO1 – Understanding the critical knowledge of fundamental concepts, algorithms, and models in Natural Language processing (NLP) for performing various linguistic NLP tasks.

LO2 - Demonstrate professional levels of insight, interpretation by utilizing the various NLP packages like Natural Language Tool Kit (NLTK) to apply, solve, implement, evaluate, and improve the real time significant applications of NLP

Student IDs: 202306728

Student Name :Sara Husain Hammad

Tutor:Sini Raj Pulari

Individual Project

Submission Date : 13th december 2023

By submitting this assessment for marking, either electronically or as hard copy, I confirm the following:

- This assignment is our own work
- Any information used has been properly referenced.
- I understand that a copy of my work may be used for moderation.
- I have kept a copy of this assignment

TABLE OF CONTENT

References [At Least 3]

TASK 1 : PROBLEM STATEMENT FORMULATION & DEFINITION

Motivation:

The motivation behind exploring the Women's E-Commerce Clothing Reviews dataset is to gain valuable insights into customer sentiments and preferences related to women's clothing. By analyzing reviews, we aim to understand the factors influencing customer satisfaction and provide meaningful recommendations for the e-commerce clothing business

Problem Statement / Project Definition:

My aim is to achieve a sentiment analysis model that accurately classifies reviews into positive, neutral, or negative sentiments. Additionally, we seek to uncover trends in customer feedback, such as commonly praised or criticized aspects of clothing items. The ultimate goal is to provide actionable insights for the e-commerce platform to enhance customer satisfaction and optimize their product offerings

Scope of the Project Topic:

The project scope encompasses the analysis of a diverse range of customer reviews, considering aspects like product quality, sizing, style, and overall satisfaction. We will explore existing works on sentiment analysis, customer feedback analysis, and related fields to inform our approach. The specific challenges and opportunities posed by the Women's E-Commerce Clothing Reviews dataset will guide the project's focus.

TASK 2 : SELECTION OF AN APPROPRIATE DATA SET(DATA COLLECTION)

Dataset Selection and Justification:

For this project, I selected the Women's E-Commerce Clothing Reviews dataset from Kaggle. The dataset provides a rich source of information related to online clothing reviews, making it suitable for classification or prediction tasks in the e-commerce domain. The dataset is sourced from Kaggle (Women's E-Commerce Clothing Reviews), offering a diverse range of features to explore.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
import string
```

```
nltk.download('all')
```

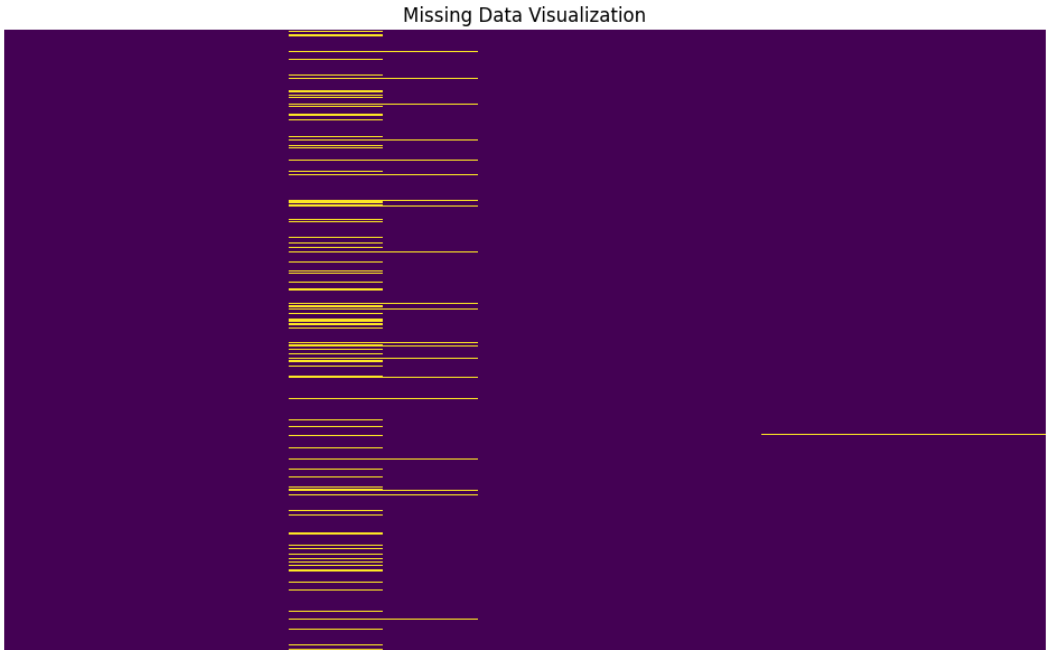
```
[nltk_data] Downloading collection 'all'
[nltk_data]    |
[nltk_data]    | Downloading package abc to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/abc.zip.
[nltk_data]    | Downloading package alpino to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/alpino.zip.
[nltk_data]    | Downloading package averaged_perceptron_tagger to
[nltk_data]    |     /root/nltk_data...
```

```
[nltk_data]    |   Unzipping taggers/averaged_perceptron_tagger.zip.
[nltk_data]    | Downloading package averaged_perceptron_tagger_ru to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Unzipping
[nltk_data]    |       taggers/averaged_perceptron_tagger_ru.zip.
[nltk_data]    | Downloading package basque_grammars to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Unzipping grammars/basque_grammars.zip.
[nltk_data]    | Downloading package bcp47 to /root/nltk_data...
[nltk_data]    | Downloading package biocreative_ppi to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Unzipping corpora/biocreative_ppi.zip.
[nltk_data]    | Downloading package bllip_wsj_no_aux to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Unzipping models/bllip_wsj_no_aux.zip.
[nltk_data]    | Downloading package book_grammars to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Unzipping grammars/book_grammars.zip.
[nltk_data]    | Downloading package brown to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/brown.zip.
[nltk_data]    | Downloading package brown_tei to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/brown_tei.zip.
[nltk_data]    | Downloading package cess_cat to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/cess_cat.zip.
[nltk_data]    | Downloading package cess_esp to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/cess_esp.zip.
[nltk_data]    | Downloading package chat80 to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/chat80.zip.
[nltk_data]    | Downloading package city_database to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Unzipping corpora/city_database.zip.
[nltk_data]    | Downloading package cmudict to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/cmudict.zip.
[nltk_data]    | Downloading package comparative_sentences to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Unzipping corpora/comparative_sentences.zip.
[nltk_data]    | Downloading package comtrans to /root/nltk_data...
[nltk_data]    | Downloading package conll2000 to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/conll2000.zip.
[nltk_data]    | Downloading package conll2002 to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/conll2002.zip.
[nltk_data]    | Downloading package conll2007 to /root/nltk_data...
[nltk_data]    | Downloading package crubadan to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/crubadan.zip.
[nltk_data]    | Downloading package dependency_treebank to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Unzipping corpora/dependency_treebank.zip.
[nltk_data]    | Downloading package dolch to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/dolch.zip.
[nltk_data]    | Downloading package europarl_raw to
```

Initial Data Visualization:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Load dataset
data=pd.read_csv("/content/drive/MyDrive/NLP /Project/Womens_Clothing_Feedback/Dataset/Womens (

# Source of Dataset - Reason why I chose the dataset
print("Source of Dataset: Women's E-Commerce Clothing Reviews on Kaggle")
print("Reason: Chose this dataset for sentiment analysis on customer reviews in the e-commerce

# Visualize the initial data
plt.figure(figsize=(12, 8))
sns.heatmap(data.isnull(), cmap='viridis', cbar=False, yticklabels=False)
plt.title('Missing Data Visualization')
plt.show()

# Display the shape of the dataset
print("Dataset shape:", data.shape)

# Display the first few rows of the dataset
print("Initial Data:")
print(data.head())
```

```
Source of Dataset: Women's E-Commerce Clothing Reviews on Kaggle
Reason: Chose this dataset for sentiment analysis on customer reviews in
```




```
#load the dataset
data=pd.read_csv("/content/drive/MyDrive/NLP /Project/Womens_Clothing_Feedback/Dataset/Womens
```

Handling Labels:

In the exploration of the Women's E-Commerce Clothing Reviews dataset, I examined and processed the labels associated with the reviews. The initial step involved displaying the shape of the dataset to understand its dimensionality.

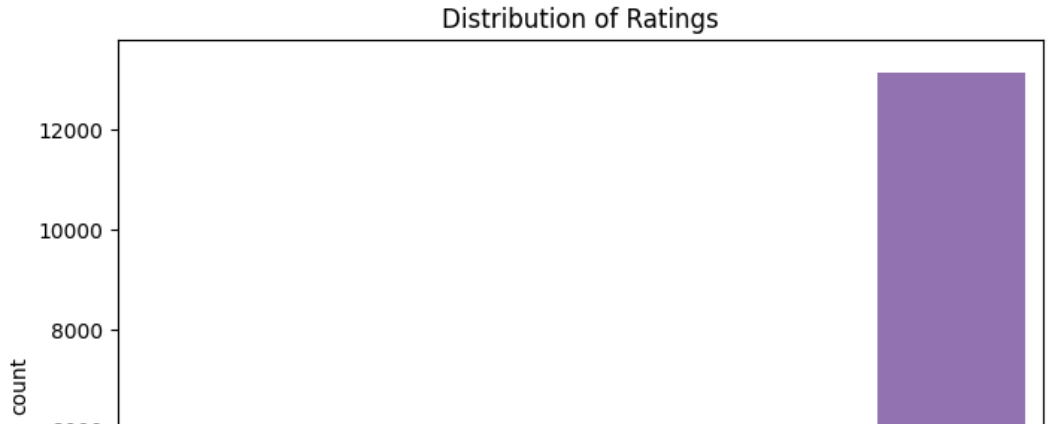
Following this, I visualized the distribution based on the original 'Rating' labels using a count plot. This provided an overview of the distribution of ratings across the dataset.

To enhance the interpretability of the labels for classification tasks, I transformed the numeric 'Rating' labels into categorical sentiments: 'dislike,' 'neutral,' and 'like.' This categorical representation facilitates a more intuitive understanding of sentiments associated with each review.

Subsequently, I visualized the distribution based on the newly created 'Sentiment' categories. This allowed for a clearer understanding of the sentiment distribution within the dataset.

```
data=pd.read_csv("/content/drive/MyDrive/NLP /Project/Womens_Clothing_Feedback/Dataset/Womens (
# Display the shape of the dataset
print("Dataset shape:", data.shape)
if 'Rating' in data.columns:
    # Visualize distribution based on labels
    plt.figure(figsize=(8, 6))
    sns.countplot(x='Rating', data=data)
    plt.title('Distribution of Ratings')
    plt.show()

    # Convert labels to categories (dislike, neutral, like)
    data['Sentiment'] = data['Rating'].apply(lambda x: 'dislike' if x in [1, 2] else ('neutral

    # Visualize distribution after conversion
    plt.figure(figsize=(8, 6))
    sns.countplot(x='Sentiment', data=data)
    plt.title('Distribution of Sentiments')
    plt.show()
```

```
Dataset shape: (23486, 11)
```



TASK 3 :TEXT PREPROCESSING

HANDLING MISSING VALUES:

```
# Data cleaning: Remove unnecessary columns
columns_to_remove = ['Unnamed: 0', 'Clothing ID', 'Age', 'Title', 'Recommended IND', 'Positive
data_cleaned = data.drop(columns=columns_to_remove, axis=1)
```

Handling NaN Values:

```
#Handling NaN Values
nan_check = data['Review Text'].isnull().any()

if nan_check:
    print("NaN values found in 'Review Text' column. Handling NaN values...")

    # Step 2: Replace NaN values with an empty string
    data['Review Text'].fillna('', inplace=True)

    print("NaN values handled.")
else:
    print("No NaN values found in 'Review Text' column.")
```

```
NaN values found in 'Review Text' column. Handling NaN values...
NaN values handled.
```


Convert to lowercase:

```
# Convert to lowercase
data['Review Text'] = data['Review Text'].str.lower()
```

Remove punctuation:

```
# Remove punctuation
data['Review Text'] = data['Review Text'].apply(lambda x: x.translate(str.maketrans('', '', st
```

Remove stop words:

```
# Remove stop words
stop_words = set(stopwords.words('english'))
data['Review Text'] = data['Review Text'].apply(lambda x: ' '.join([word for word in word_toke
```

Tokenization, stemming:

```
# Tokenization, stemming
ps = PorterStemmer()
data['Review Text'] = data['Review Text'].apply(lambda x: ' '.join([ps.stem(word) for word in
```

Remove numeric characters:

```
# Remove numeric characters
data['Review Text'] = data['Review Text'].str.replace('\d+', '')
```

```
<ipython-input-15-5cebfbbfd0a0>:2: FutureWarning: The default value of regex will change fr
  data['Review Text'] = data['Review Text'].str.replace('\d+', '')
```

Visualize the cleaned data:

```
# Visualize the cleaned data
print("\nCleaned data:")
print(data_cleaned.head())
```

```
Cleaned data:
                                         Review Text  Rating Sentiment
0  Absolutely wonderful - silky and sexy and comf...       4      like
1  Love this dress!  it's sooo pretty.  i happene...       5      like
2  I had such high hopes for this dress and reall...       3   neutral
3  I love, love, love this jumpsuit. it's fun, fl...       5      like
4  This shirt is very flattering to all due to th...       5      like
```

```
# Display Review Text data
print("Cleaned data:")
print(data['Review Text'].head())
```

```
Cleaned data:
0                    absolut wonder silki sexi comfort
1    love dress sooo pretti happen find store im gl...
```

```
2    high hope dress realli want work initi order p...
3    love love love jumpsuit fun flirti fabul everi...
4    shirt flatter due adjust front tie perfect len...
Name: Review Text, dtype: object
```

```
# Display the shape of the cleaned dataset
print("\nShape of the cleaned dataset:")
print(data_cleaned.shape)
```

```
Shape of the cleaned dataset:
(23486, 3)
```

TEXT PRE-PROCESSING TECHNIQUES:

1. Tokenization:

Tokenization is the process of breaking down a text into individual words or tokens. The 'Review Text' column in the dataset is tokenized using the NLTK (Natural Language Toolkit) library's word_tokenize function.

Visualizing Token Length Distribution:

The code computes the length of each tokenized text and creates a histogram to visualize the distribution of token lengths. The histogram provides insights into the range of token lengths in the dataset.

Displaying Tokenized Text:

The code prints the tokenized text for the first few rows in the 'Tokenized_Text' column.

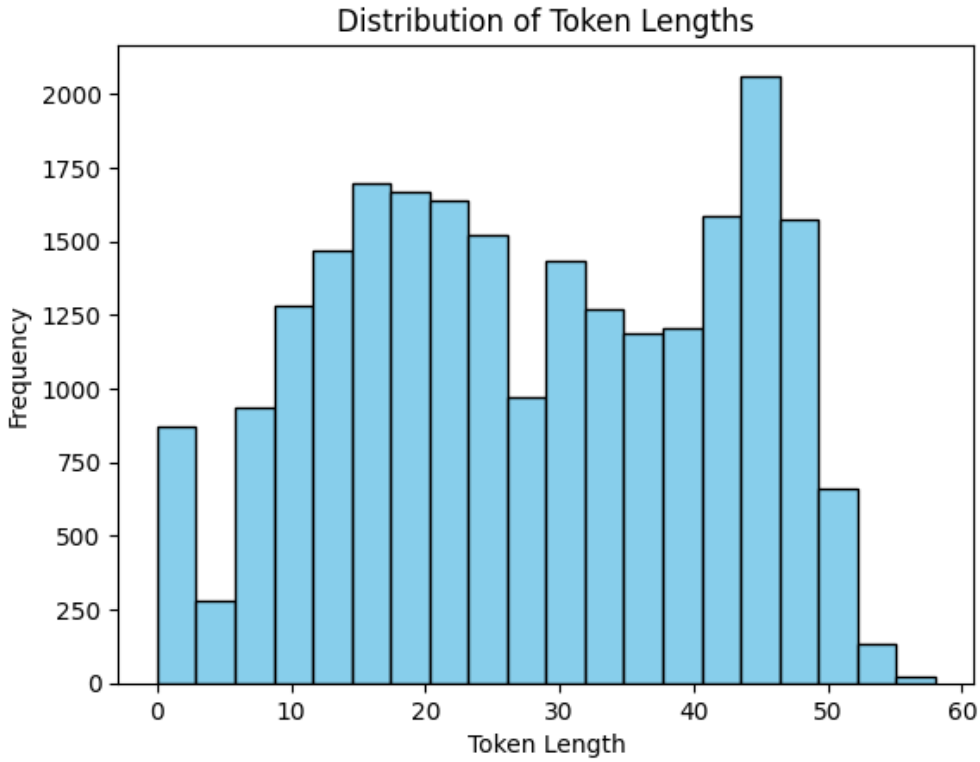
Why Tokenization?

Tokenization is a fundamental step in text processing, breaking down the text into smaller units for analysis. It helps convert unstructured text data into a format suitable for machine learning algorithms. Analyzing token lengths provides an understanding of the text complexity and can guide subsequent preprocessing steps.

Effects on the Dataset:

Tokenization transforms the original text into a structured format of individual words or tokens. Visualizing token lengths can reveal patterns in the distribution of text complexity. The 'Tokenized_Text' column is added to the dataset, containing lists of tokens for each corresponding 'Review Text' entry.

```
from nltk.tokenize import word_tokenize
import matplotlib.pyplot as plt

# Tokenize the 'Review Text' column
data['Tokenized_Text'] = data['Review Text'].apply(word_tokenize)

# Visualize the distribution of token lengths
token_lengths = data['Tokenized_Text'].apply(len)
plt.hist(token_lengths, bins=20, color='skyblue', edgecolor='black')
plt.title('Distribution of Token Lengths')
plt.xlabel('Token Length')
plt.ylabel('Frequency')
plt.show()

# Display the tokenized text
print("Tokenized Text:")
print(data['Tokenized_Text'].head())
```




```
Tokenized Text:
0              [absolut, wonder, silki, sexi, comfort]
1    [love, dress, sooo, pretti, happen, find, stor...
2    [high, hope, dress, realli, want, work, initi,...
3    [love, love, love, jumpsuit, fun, flirti, fabu...
4    [shirt, flatter, due, adjust, front, tie, perf...
Name: Tokenized_Text, dtype: object
```

2. Stemming:

Stemming is a text normalization process that reduces words to their root or base form. The Porter stemming algorithm, implemented in the NLTK library, is applied to the tokenized text in the 'Tokenized_Text' column.

Displaying Stemmed Text:

The code prints the stemmed text for the first few rows in the 'Stemmed_Text' column.

Why Stemming?

Stemming helps reduce words to their base or root form, simplifying the analysis of variations of words. It is a form of text normalization that aids in feature reduction and can improve the efficiency of text-based models.

Effects on the Dataset:

The 'Tokenized_Text' column is a prerequisite for stemming, so it assumes that tokenization has already been performed. Stemming results in a new column, 'Stemmed_Text,' containing lists of stemmed words for each corresponding 'Review Text' entry. In summary, this code applies stemming to the tokenized text, reducing words to their base forms. The 'Stemmed_Text' column is added to the dataset, providing a processed version of the original text data for further analysis.

```
from nltk.stem import PorterStemmer

# Apply stemming to the 'Review Text' column
ps = PorterStemmer()
data['Stemmed_Text'] = data['Tokenized_Text'].apply(lambda x: [ps.stem(word) for word in x])

# Display the stemmed text
print("Stemmed Text:")
print(data['Stemmed_Text'].head())
```

```
Stemmed Text:
0              [absolut, wonder, silki, sexi, comfort]
1    [love, dress, sooo, pretti, happen, find, stor...
2    [high, hope, dress, realli, want, work, initi,...
3    [love, love, love, jumpsuit, fun, flirti, fabu...
4    [shirt, flatter, due, adjust, front, tie, perf...
Name: Stemmed_Text, dtype: object
```

3. Lemmatization:

Lemmatization is a text normalization process that reduces words to their base or dictionary form (lemma). The WordNet Lemmatizer, provided by the NLTK library, is applied to the tokenized text in the 'Tokenized_Text' column.

Displaying Lemmatized Text:

The code prints the lemmatized text for the first few rows in the 'Lemmatized_Text' column.

Why Lemmatization?

Lemmatization, like stemming, aims to bring words to their base form, but it considers the context and meaning of words. It is a more sophisticated normalization technique compared to stemming.

Effects on the Dataset:

The 'Tokenized_Text' column is a prerequisite for lemmatization, so it assumes that tokenization has already been performed. Lemmatization results in a new column, 'Lemmatized_Text,' containing lists of lemmatized words for each corresponding 'Review Text' entry.

In summary, this code applies lemmatization to the tokenized text, producing a more contextually accurate representation of the base forms of words. The 'Lemmatized_Text' column is added to the dataset for further analysis.

```
from nltk.stem import WordNetLemmatizer

# Download the WordNet resource for lemmatization
nltk.download('wordnet')

# Apply lemmatization to the 'Review Text' column
lemmatizer = WordNetLemmatizer()
data['Lemmatized_Text'] = data['Tokenized_Text'].apply(lambda x: [lemmatizer.lemmatize(word) f

# Display the lemmatized text
print("Lemmatized Text:")
print(data['Lemmatized_Text'].head())
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
Lemmatized Text:
0              [absolut, wonder, silki, sexi, comfort]
1    [love, dress, sooo, pretti, happen, find, stor...
2    [high, hope, dress, realli, want, work, initi,...
3    [love, love, love, jumpsuit, fun, flirti, fabu...
4    [shirt, flatter, due, adjust, front, tie, perf...
Name: Lemmatized_Text, dtype: object
```

4. TF-IDF (Term Frequency-Inverse Document Frequency):

It converts the 'Review Text' column into a list of strings (corpus). TF-IDF (Term Frequency-Inverse Document Frequency) vectorization is applied to the corpus using the TfidfVectorizer from scikit-learn.

Visualizing TF-IDF Scores:

The TF-IDF scores for the first document (first row in the dataset) are visualized. The top 10 words with the highest TF-IDF scores are displayed in a horizontal bar plot.

Why TF-IDF Vectorization?

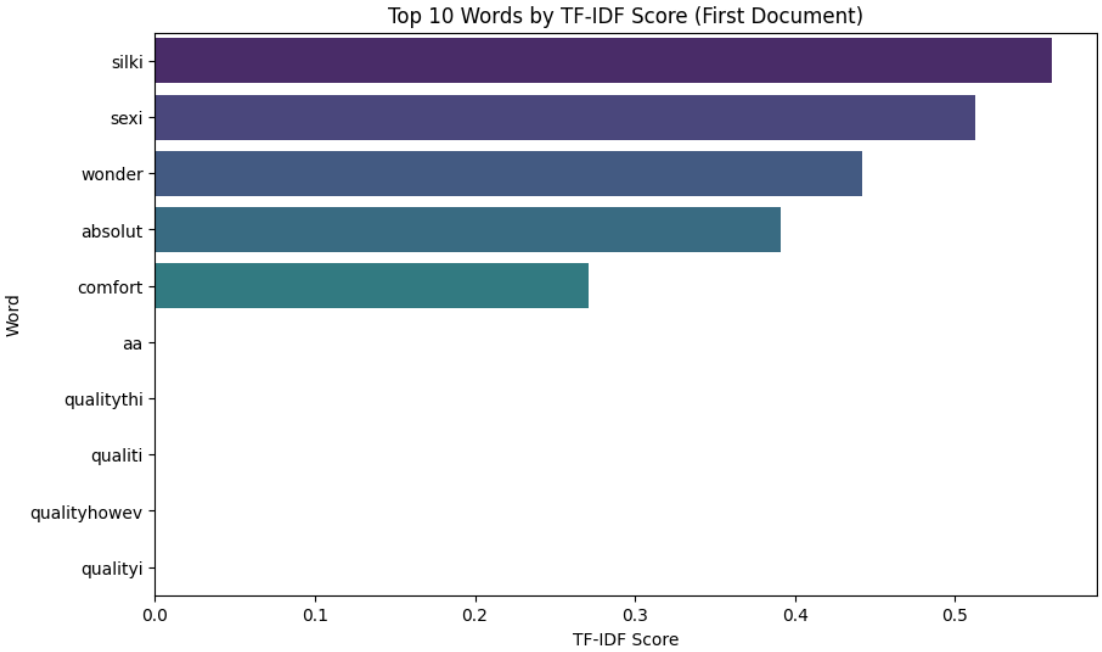
TF-IDF is a numerical statistic that reflects the importance of a word in a document relative to a collection of documents (corpus). It is commonly used in natural language processing tasks to represent text data numerically.

Effects on the Dataset:

TF-IDF vectorization results in a sparse matrix representation of the textual data. The visualization provides insights into the importance of words in the first document based on their TF-IDF scores.

In summary, this code applies TF-IDF vectorization to the 'Review Text' column, allowing for a numerical representation of the textual data. The visualization highlights the words with the highest TF-IDF scores in the first document.

```
from sklearn.feature_extraction.text import TfidfVectorizer
import seaborn as sns

# Convert the 'Review Text' column to a list of strings
corpus = data['Review Text'].tolist()

# Apply TF-IDF vectorization
vectorizer = TfidfVectorizer()
tfidf_matrix = vectorizer.fit_transform(corpus)

# Visualize TF-IDF scores for the first document
feature_names = vectorizer.get_feature_names_out()
tfidf_scores = tfidf_matrix[0].toarray().flatten()
tfidf_data = pd.DataFrame({'Word': feature_names, 'TF-IDF Score': tfidf_scores})
tfidf_data = tfidf_data.sort_values(by='TF-IDF Score', ascending=False).head(10)

plt.figure(figsize=(10, 6))
sns.barplot(x='TF-IDF Score', y='Word', data=tfidf_data, palette='viridis')
plt.title('Top 10 Words by TF-IDF Score (First Document)')
plt.show()
```



TASK 4 : TEXT REPRESENTATION

I'll use three different text representation techniques: Bag of Words (BoW), Word Embeddings (using Word2Vec), and N-Grams.

1. Bag of Words (BoW):

Bag of Words (BoW) Representation:

It converts the 'Review Text' column into a list of strings (corpus). The Bag of Words representation is applied using the CountVectorizer from scikit-learn.

Visualizing the Bag of Words Matrix:

The entire Bag of Words matrix for the dataset is visualized using a heatmap. The heatmap displays the occurrence counts of words (features) in the first 50 words of the vocabulary.

Why Bag of Words Representation?

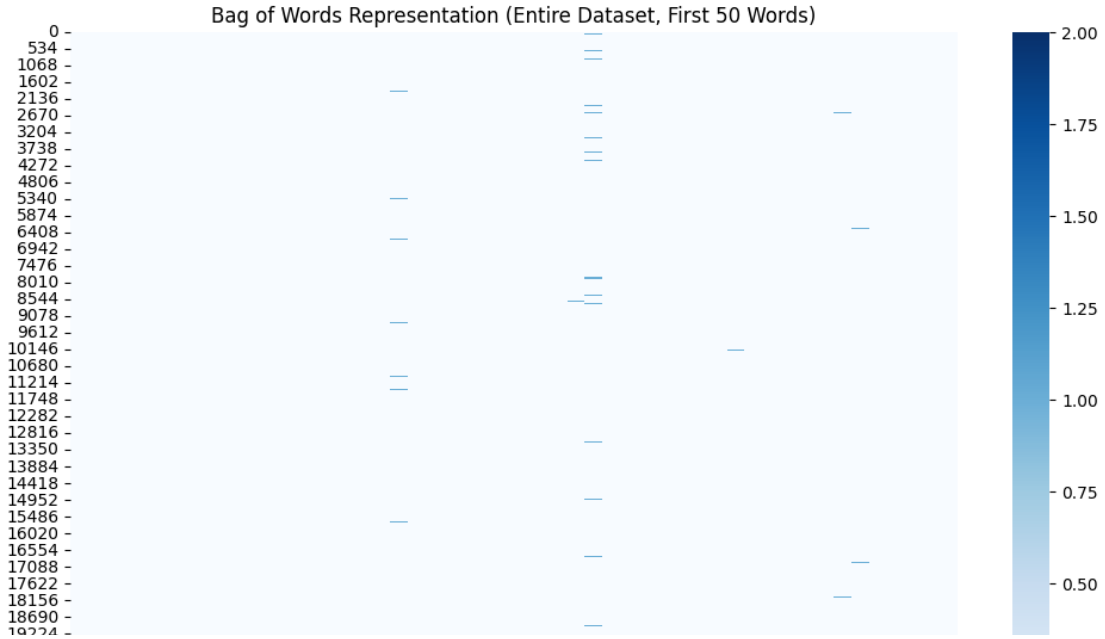
Bag of Words is a common text representation technique that focuses on the occurrence of words in a document, disregarding their order. It is a simple and effective way to convert textual data into a numerical format for machine learning models.

Output and Insights:

The heatmap visualizes the Bag of Words matrix for the entire dataset, specifically showing the occurrence counts of the first 50 words in the vocabulary. Darker shades represent higher occurrence counts. The output provides a numerical representation of the textual data, with each row corresponding to a document (review) and each column corresponding to a unique word in the vocabulary. This representation is suitable for further analysis or as input to machine learning models.

In summary, the code applies the Bag of Words representation to the 'Review Text' column, and the heatmap visually represents the occurrence counts of words in the dataset, offering insights into the distribution of words across different documents.

```
from sklearn.feature_extraction.text import CountVectorizer

# Convert the 'Review Text' column to a list of strings
corpus = data['Review Text'].tolist()

# Apply Bag of Words representation
vectorizer = CountVectorizer()
bow_matrix = vectorizer.fit_transform(corpus)

# Visualize the Bag of Words matrix for the entire dataset
bow_df = pd.DataFrame(bow_matrix.toarray(), columns=vectorizer.get_feature_names_out())
plt.figure(figsize=(12, 8))
sns.heatmap(bow_df.iloc[:, :50], cmap='Blues', annot=False)
plt.title('Bag of Words Representation (Entire Dataset, First 50 Words)')
plt.show()
```



2. Word Embeddings (Word2Vec):

The below code performs the following tasks:

Tokenization:

It tokenizes the 'Review Text' column using the word_tokenize function from the NLTK library.

Training Word2Vec Model:

It trains a Word2Vec model on the tokenized reviews using the Word2Vec class from the Gensim library.

Parameters: vector_size: Dimensionality of the word vectors (100 in this case). window: Maximum distance between the current and predicted word within a sentence (5 in this case). min_count: Ignores all words with a total frequency lower than this (set to 1, including infrequent words). workers: Number of CPU cores to use for model training (4 in this case).

Checking the Vocabulary:

It prints the vocabulary, which consists of unique words mapped to their corresponding indices.

Why Word2Vec Model?

Word2Vec is a popular word embedding technique that represents words as dense vectors in a continuous vector space. It captures semantic relationships between words and is capable of understanding the context and similarity between words.

Output and Insights:

The output is the vocabulary of the Word2Vec model, which includes unique words from the 'Review Text' column along with their corresponding indices. Each word in the vocabulary is associated with a dense vector representation of size 100.

In summary, the code tokenizes the text and trains a Word2Vec model to obtain word embeddings for the words in the dataset. The vocabulary provides a mapping of words to their vector representations, allowing for semantic understanding and similarity analysis.

The below code is print the vocabulary of my Word2Vec model, so I can select the words for the Word Embedding method

```
from gensim.models import Word2Vec
from nltk.tokenize import word_tokenize

# Tokenize the 'Review Text' column
tokenized_reviews = data['Review Text'].apply(word_tokenize)

# Train Word2Vec model
word2vec_model = Word2Vec(sentences=tokenized_reviews, vector_size=100, window=5, min_count=1,

# Check the vocabulary
vocab = word2vec_model.wv.key_to_index
print("Vocabulary:", list(vocab.keys()))
```

```
Vocabulary: ['dress', 'love', 'fit', 'size', 'look', 'top', 'wear', 'like', 'color', 'great
```

The below code performs the following:

Loading the Dataset:

It loads the dataset from a CSV file, specifically the 'Womens Clothing E-Commerce Reviews' dataset. Data Cleaning:

It removes rows with NaN values in the "Review Text" column to ensure data quality.

Tokenization:

It tokenizes the 'Review Text' column using the word_tokenize function from the NLTK library.

Training Word2Vec Model:

It trains a Word2Vec model on the tokenized reviews using the Word2Vec class from the Gensim library.

Parameters:

vector_size: Dimensionality of the word vectors (100 in this case). window: Maximum distance between the current and predicted word within a sentence (5 in this case). min_count: Ignores all words with a total frequency lower than this (set to 1, including infrequent words). workers: Number of CPU cores to use for model training (4 in this case).

Accessing Word Embeddings:

It accesses the word embeddings obtained from the trained Word2Vec model. Finding Similar Words:

It finds the most similar words to a specified word ('word' in this case) and extracts their vectors. Scatter Plot of Word Vectors:

It plots the word vectors in a scatter plot, where each point represents a word's vector in a 2D space (using the first two dimensions). The plot is annotated with the corresponding words.

Why Scatter Plot of Word Vectors?

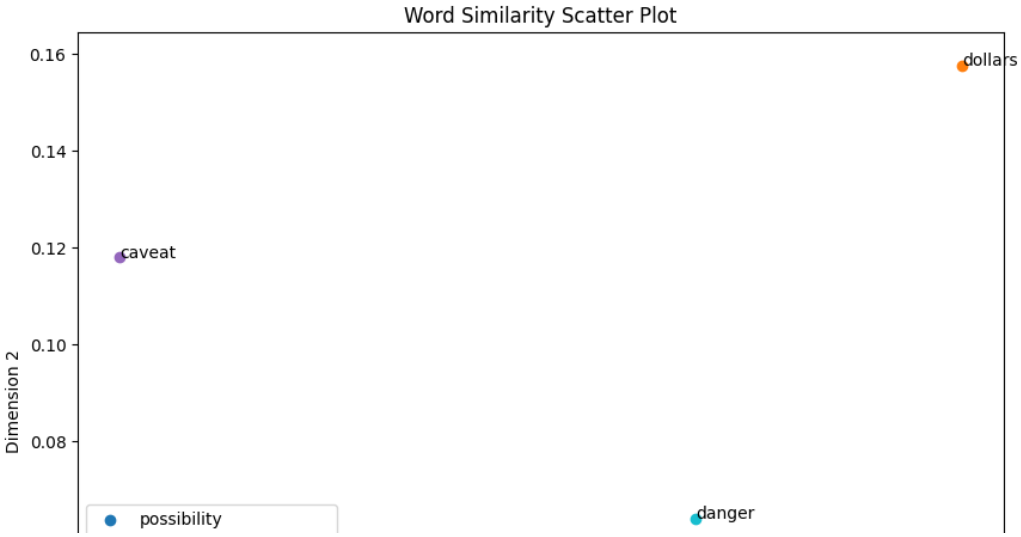
The scatter plot visually represents the similarity between words in the embedding space. Words with similar meanings or contexts are expected to be closer in the plot.

Output and Insights:

The output is a scatter plot where each point represents a word, and words with similar meanings are expected to be close to each other. The visualization provides insights into the semantic relationships captured by the Word2Vec model.

In summary, the code uses Word2Vec embeddings to visualize word similarities in a scatter plot, providing a qualitative understanding of the semantic relationships in the dataset.

```
import pandas as pd
from nltk.tokenize import word_tokenize
from gensim.models import Word2Vec
import matplotlib.pyplot as plt
import seaborn as sns

# Load your dataset
data = pd.read_csv("/content/drive/MyDrive/NLP /Project/Womens_Clothing_Feedback/Dataset/Women

# Remove rows with NaN values in the "Review Text" column
data = data.dropna(subset=['Review Text'])

# Tokenize the 'Review Text' column
tokenized_reviews = data['Review Text'].apply(word_tokenize)

# Train Word2Vec model
word2vec_model = Word2Vec(sentences=tokenized_reviews, vector_size=100, window=5, min_count=1,

# Access the word embeddings
word_embeddings = word2vec_model.wv

# Get the most similar words
similar_words = word_embeddings.most_similar('word', topn=10)

# Extract similar words and their vectors
similar_words, similarities = zip(*similar_words)
vectors = [word_embeddings.get_vector(word) for word in similar_words]

# Plot the word vectors in a scatter plot
fig, ax = plt.subplots(figsize=(10, 8))
for word, vector in zip(similar_words, vectors):
    ax.scatter(vector[0], vector[1], label=word)

# Annotate points with words
for i, word in enumerate(similar_words):
    ax.annotate(word, (vectors[i][0], vectors[i][1]))

plt.title('Word Similarity Scatter Plot')
plt.xlabel('Dimension 1')
plt.ylabel('Dimension 2')
plt.legend()
plt.show()
```



3. N-Grams:

This code performs the following tasks:

Loading the Dataset:

It loads the 'Womens Clothing E-Commerce Reviews' dataset from a CSV file.

Data Type Conversion:

It ensures that the 'Review Text' column is of string type for consistency.

Printing Preprocessed Data:

It prints the first few rows of the preprocessed data for a quick check.

Limiting the Number of Documents:

It limits the number of documents for memory efficiency (adjustable based on available memory).

N-Grams Representation:

It applies the N-Grams representation to the limited set of documents using CountVectorizer. The ngram_range parameter is set to (1, 3) to include unigrams, bigrams, and trigrams.

Handling Zero Values:

A small offset (offset) is added to avoid issues with zero values when taking the logarithm. Printing Feature Names:

It prints the first 50 feature names extracted from the N-Grams.

Visualizing the N-Grams Matrix:

It creates a heatmap to visualize the N-Grams matrix with adjustments for better visibility. The visualization is limited to the first 100 documents and the first 50 N-Grams.

Why Visualize N-Grams Matrix? **bold text** The heatmap visually represents the presence of N-Grams (combinations of adjacent words) in the dataset. It provides insights into frequently occurring word sequences.

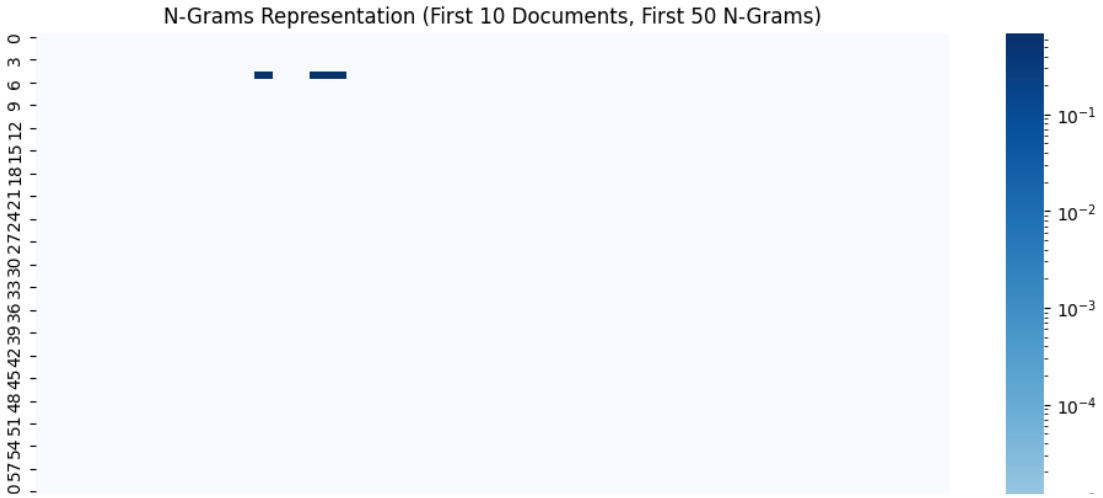
Output and Insights:

The output includes the preprocessed data and a heatmap displaying the N-Grams representation. The heatmap helps in understanding the distribution of N-Grams in the dataset.

In summary, the code focuses on visualizing the N-Grams representation to capture sequential word patterns in the dataset. Adjustments are made to handle zero values and enhance visualization using a heatmap.

```
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.colors import LogNorm
from scipy.sparse import csr_matrix

# Load the dataset
data=pd.read_csv("/content/drive/MyDrive/NLP /Project/Womens_Clothing_Feedback/Dataset/Womens (
data['Review Text'] = data['Review Text'].astype(str)

# Check the first few rows of the preprocessed data
print("Preprocessed Data (First 5 Documents):")
for i in range(5):
    print(f"Document {i + 1}: {data['Review Text'][i]}")

# Limit the number of documents for memory efficiency
num_documents = 1000  # You can adjust this number based on your available memory
corpus = data['Review Text'][:num_documents].tolist()

# Apply N-Grams representation
ngram_vectorizer = CountVectorizer(ngram_range=(1, 3))  # or (1, 4)
ngram_matrix = ngram_vectorizer.fit_transform(corpus)

# Add a small offset to avoid issues with zero values
offset = 1e-8
ngram_matrix_log = csr_matrix(np.log1p(ngram_matrix.toarray() + offset))

# Print the first 50 feature names
feature_names = ngram_vectorizer.get_feature_names_out()
print("Extracted N-Grams:", feature_names[:50])

# Visualize the N-Grams matrix with adjustments
ngram_df = pd.DataFrame(ngram_matrix_log.toarray(), columns=feature_names)
plt.figure(figsize=(12, 8))
sns.heatmap(ngram_df.iloc[:100, :50], cmap='Blues', annot=False, norm=LogNorm())
plt.title('N-Grams Representation (First 10 Documents, First 50 N-Grams)')
plt.show()
```

```
Preprocessed Data (First 5 Documents):
Document 1: Absolutely wonderful - silky and sexy and comfortable
Document 2: Love this dress!  it's sooo pretty.  i happened to find it in a s
Document 3: I had such high hopes for this dress and really wanted it to work
Document 4: I love, love, love this jumpsuit. it's fun, flirty, and fabulous!
Document 5: This shirt is very flattering to all due to the adjustable front
Extracted N-Grams: ['00' '00 dress' '00 dress does' '00 for' '00 for my' '00
 '00 petite fits' '00 price' '00 price tag' '03' '03 dd' '03 dd che' '0p'
 '0p fit' '0p fit snugly' '0p in' '0p in this' '0p the' '0p the zero' '10'
 '10 12' '10 12 and' '10 12 but' '10 12 in' '10 12 on' '10 12 or'
 '10 also' '10 also means' '10 and' '10 and chose' '10 and found'
 '10 and the' '10 and this' '10 and tried' '10 ankle' '10 ankle length'
 '10 australian' '10 australian us' '10 but' '10 but still' '10 depending'
 '10 depending on' '10 fit' '10 fit in' '10 fit quite' '10 got'
 '10 got the' '10 got this' '10 if' '10 if get']
```



TASK 5 : TEXT CLASSIFICATION / PREDICTION

I'll perform text classification using two models: Multinomial Naive Bayes and LSTM (Deep Learning).

Package Usage:

Multinomial Naïve Bayes:

SKLEARN for TF-IDF vectorization and model implementation.

MODEL 2: Recurrent Neural Network (RNN) with Long Short-Term Memory (LSTM)

TensorFlow/Keras for tokenization, sequence padding, and LSTM model training.

1. Data Preparation:

Ensure the dataset is cleaned, preprocessed, and suitable for text classification. Features (X): Text data 'Review Text' Labels (y): 'Sentiment' based on ratings

```
import pandas as pd
from sklearn.model_selection import train_test_split

# Load your dataset
data = pd.read_csv("/content/drive/MyDrive/NLP /Project/Womens_Clothing_Feedback/Dataset/Women

data['Sentiment'] = data['Rating'].apply(lambda x: 'dislike' if x in [1, 2] else ('neutral' if

# Remove rows with NaN values in the "Review Text" and "Sentiment" columns
data.dropna(subset=['Review Text', 'Sentiment'], inplace=True)


# Split the data into features (X) and labels (y)
X = data['Review Text']
y = data['Sentiment']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

2. Splitting into Training and Testing Sets:

```
from sklearn.model_selection import train_test_split

# Split the data into features (X) and labels (y)
X = data['Review Text']
y = data['Rating']

# Split the data into training and testing sets
# Adjust the test_size parameter based on your preference
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Now, X_train and y_train are your training features and labels, respectively
# X_test and y_test are your testing features and labels, respectively
```

Model 1: Multinomial Naïve Bayes (ML-based):

Why?

- Efficient for text classification tasks.
- Performs well with the Bag of Words or TF-IDF representations.
- Simple yet effective for sentiment analysis.

Accuracy: It indicates the overall correctness of the model. In my case, the accuracy is 0.64, meaning the model is correct in its predictions about 64% of the time.

Precision: It measures the accuracy of the positive predictions. For example, the precision for the rating "5" is 0.78, which means that when the model predicts a review to be a "5," it is correct 78% of the time.

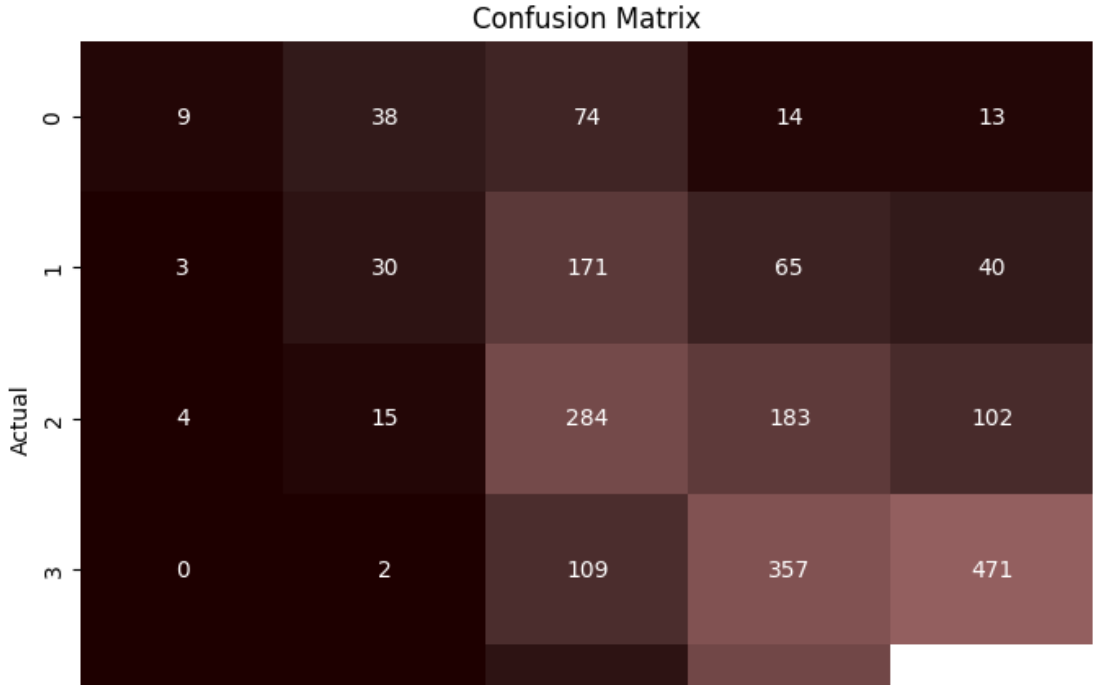
Recall: It measures the ability of the model to capture all the relevant instances. For example, the recall for the rating "5" is 0.88, indicating that the model correctly identifies 88% of the actual "5" ratings.

F1-score: It is the harmonic mean of precision and recall. It provides a balance between precision and recall. The weighted average F1-score for my model is 0.62.

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
```

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming 'data' is your DataFrame
# Replace 'Rating' with your actual label column name
# Drop any rows with missing values
data.dropna(subset=['Review Text', 'Rating'], inplace=True)

# Convert ratings to numeric values if they are categorical
data['Label'] = data['Rating'].astype(int)

# Split the data into features (X) and labels (y)
X = data['Review Text']
y = data['Label']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Apply Bag of Words representation using CountVectorizer
vectorizer = CountVectorizer()
X_train_bow = vectorizer.fit_transform(X_train)
X_test_bow = vectorizer.transform(X_test)

# Train the Multinomial Naive Bayes model
naive_bayes = MultinomialNB()
naive_bayes.fit(X_train_bow, y_train)

# Make predictions on the test set
y_pred = naive_bayes.predict(X_test_bow)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

# Display classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Create confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Define your custom color palette
'''custom_colors = [(255, 0, 0), (0, 255, 0), (0, 0, 255), (255, 255, 0), (255, 0, 255)]  # RGB
custom_palette = sns.color_palette(custom_colors, as_cmap=True)'''


# Plot confusion matrix as a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='pink', cbar=False)
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
Accuracy: 0.64
Classification Report:
              precision    recall  f1-score   support

           1       0.53      0.06      0.11       148
           2       0.34      0.10      0.15       309
           3       0.42      0.48      0.45       588
           4       0.40      0.38      0.39       939
           5       0.78      0.88      0.83      2545

    accuracy                           0.64      4529
   macro avg       0.49      0.38      0.39      4529
weighted avg       0.62      0.64      0.62      4529
```



MODEL 2: Recurrent Neural Network (RNN) with Long Short-Term Memory (LSTM)

- The goal is likely to predict or regress the ratings of clothing reviews using an LSTM-based neural network.
- The linear activation function in the output layer suggests a regression task where the model predicts numerical ratings.
- The Adam optimizer and mean squared error loss are suitable for regression problems.
- The plot helps visualize how the training and validation loss change over epochs, providing insights into model performance and potential overfitting.

The output indicates the training and validation loss values for each epoch during the training of a neural network. In general:

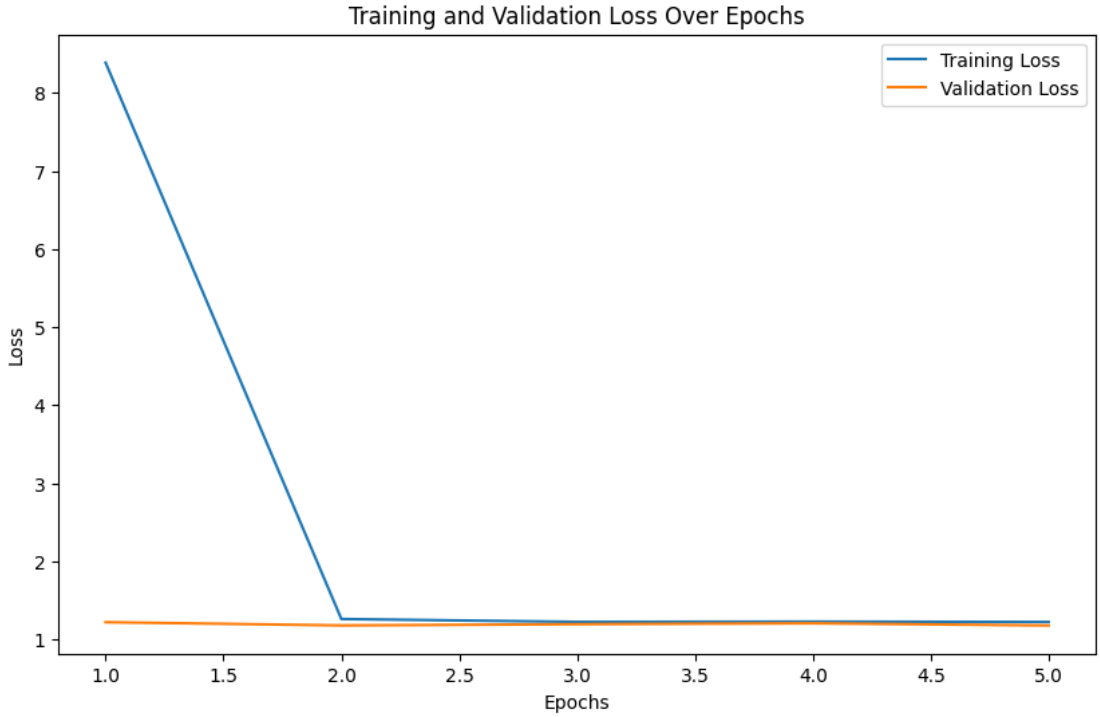
Loss Values: The loss values seem to decrease over the epochs, which is a positive sign. The model is learning to minimize the difference between its predictions and the actual values.

Test Loss: The test loss is also relatively low, which indicates that the model is performing well on unseen data.

If the loss is decreasing and the model is performing well on the test set, it's generally a positive outcome.

```
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Embedding, LSTM, Dense

# Load your dataset
data = pd.read_csv("/content/drive/MyDrive/NLP /Project/Womens_Clothing_Feedback/Dataset/Womens

# Remove rows with NaN values in the "Review Text" and "Rating" columns
data = data.dropna(subset=['Review Text', 'Rating'])

# Select a subset (first 1000 rows) for testing
subset_data = data.head(1000)

# Tokenize the text
tokenizer = Tokenizer()
tokenizer.fit_on_texts(subset_data['Review Text'])
total_words = len(tokenizer.word_index) + 1

# Create input sequences
input_sequences = tokenizer.texts_to_sequences(subset_data['Review Text'])
X = pad_sequences(input_sequences, padding='post')

# Prepare target variable
y = subset_data['Rating']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Build the LSTM model
model_lstm = Sequential()
model_lstm.add(Embedding(input_dim=total_words, output_dim=100, input_length=X.shape[1]))
model_lstm.add(LSTM(100))
model_lstm.add(Dense(1, activation='linear'))  # Assuming regression task, change activation fo

# Compile the model
model_lstm.compile(optimizer='adam', loss='mean_squared_error')  # Use appropriate loss functio

# Train the model and store the training history
history = model_lstm.fit(X_train, y_train, epochs=5, batch_size=32, validation_split=0.2)

# Evaluate the model
loss = model_lstm.evaluate(X_test, y_test)
print(f"Test Loss: {loss}")

# Plot the training and validation loss
train_loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(train_loss) + 1)

plt.figure(figsize=(10, 6))
plt.plot(epochs, train_loss, label='Training Loss')
plt.plot(epochs, val_loss, label='Validation Loss')
plt.title('Training and Validation Loss Over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
Epoch 1/5
20/20 [==============================] - 10s 198ms/step - loss: 8.3873 - val_
Epoch 2/5
20/20 [==============================] - 2s 87ms/step - loss: 1.2602 - val_lo
Epoch 3/5
20/20 [==============================] - 3s 128ms/step - loss: 1.2233 - val_l
Epoch 4/5
20/20 [==============================] - 3s 134ms/step - loss: 1.2266 - val_l
Epoch 5/5
20/20 [==============================] - 2s 125ms/step - loss: 1.2223 - val_l
7/7 [==============================] - 1s 24ms/step - loss: 0.9914
Test Loss: 0.9914405345916748
```



TASK6 : EVALUATION,INFERENCES,RECOMMENDATIO & REFLECTION

evaluate and compare the performance of at least two models. In this case, let's consider the Multinomial Naive Bayes model and the LSTM model that you implemented earlier. We'll evaluate them using metrics like accuracy, precision, recall, F1 measure, and confusion matrix.

Multinomial Naive Bayes Evaluation:

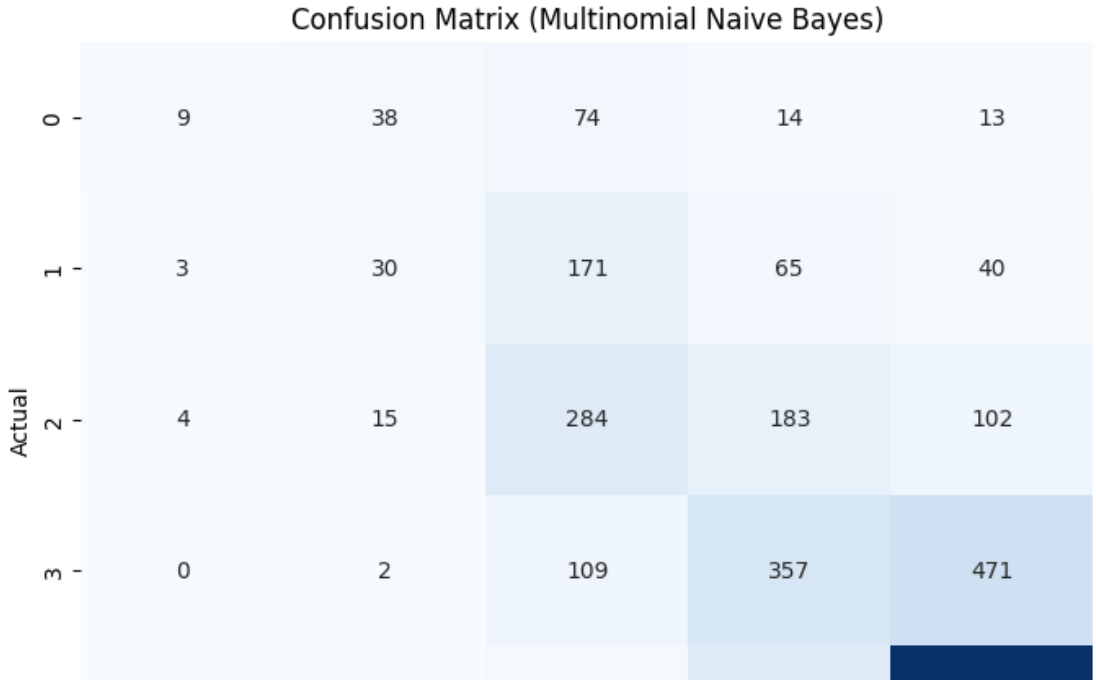
Model Selection: Uses the Multinomial Naive Bayes model for classification.

Metrics: Computes metrics such as Mean Squared Error (MSE), R^2 Score, Classification Report, and Confusion Matrix.

Visualization: Visualizes the Confusion Matrix using seaborn.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import mean_squared_error, r2_score, classification_report, confusion_mat
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming 'data' is your DataFrame
# Replace 'Rating' with your actual label column name
# Drop any rows with missing values
data.dropna(subset=['Review Text', 'Rating'], inplace=True)

# Convert ratings to numeric values if they are categorical
data['Label'] = data['Rating'].astype(int)

# Split the data into features (X) and labels (y)
X = data['Review Text']
y = data['Label']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Apply Bag of Words representation using CountVectorizer
vectorizer = CountVectorizer()
X_train_bow = vectorizer.fit_transform(X_train)
X_test_bow = vectorizer.transform(X_test)

# Train the Multinomial Naive Bayes model
naive_bayes = MultinomialNB()
naive_bayes.fit(X_train_bow, y_train)

# Make predictions on the test set
y_pred = naive_bayes.predict(X_test_bow)

# Evaluate the model using regression metrics
mse_nb = mean_squared_error(y_test, y_pred)
r2_nb = r2_score(y_test, y_pred)

# Display classification report
classification_rep = classification_report(y_test, y_pred)
print(f"Classification Report (Multinomial Naive Bayes):\n{classification_rep}")

# Display confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title('Confusion Matrix (Multinomial Naive Bayes)')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

print(f"Mean Squared Error (Multinomial Naive Bayes): {mse_nb:.2f}")
print(f"R^2 Score (Multinomial Naive Bayes): {r2_nb:.2f}")
```

```
Classification Report (Multinomial Naive Bayes):
              precision    recall  f1-score   support

           1       0.53      0.06      0.11       148
           2       0.34      0.10      0.15       309
           3       0.42      0.48      0.45       588
           4       0.40      0.38      0.39       939
           5       0.78      0.88      0.83      2545

    accuracy                           0.64      4529
   macro avg       0.49      0.38      0.39      4529
weighted avg       0.62      0.64      0.62      4529
```



Model 1: Multinomial Naive Bayes

Classification Report:

Precision: Indicates the accuracy of the positive predictions. For example, for class 1, only 53% of the predicted positive instances are true positives.

Recall: Represents the ability of the model to capture all the positive instances. For instance, for class 1, only 6% of the actual positive instances are correctly predicted.

F1-Score: The harmonic mean of precision and recall. It balances precision and recall.

Support: The number of actual occurrences of the class in the specified dataset.

Analysis:

The model performs relatively well in predicting class 5 (high ratings) with high precision, recall, and F1-score. However, for classes 1, 2, and 3, the performance is lower, especially in terms of recall and F1-score. There is room for improvement, especially for lower-rated classes.

Suggestions for Improvement:

Consider balancing the dataset if there is a significant class imbalance. Experiment with different text representation techniques. Tune hyperparameters and explore more sophisticated models.

LSTM Model Evaluation:

Model Selection: Uses an LSTM model for regression.

Metrics: Computes the Mean Squared Error (MSE) for evaluation.

Visualization: Doesn't have specific visualizations, but prints the Test Loss.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Embedding, LSTM, Dense
import scipy  # Import the necessary library


# Assuming 'data' is your DataFrame containing the dataset
data = pd.read_csv("/content/drive/MyDrive/NLP /Project/Womens_Clothing_Feedback/Dataset/Womens

# Remove rows with NaN values in the "Review Text" and "Rating" columns
data = data.dropna(subset=['Review Text', 'Rating'])

# Select a subset (first 1000 rows) for testing
subset_data = data.head(1000)

# Tokenize the text
tokenizer = Tokenizer()
tokenizer.fit_on_texts(subset_data['Review Text'])
total_words = len(tokenizer.word_index) + 1

# Create input sequences
input_sequences = tokenizer.texts_to_sequences(subset_data['Review Text'])
X = pad_sequences(input_sequences, padding='post')

# Prepare target variable
y = subset_data['Rating']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Build the LSTM model
model_lstm = Sequential()
model_lstm.add(Embedding(input_dim=total_words, output_dim=100, input_length=X.shape[1]))
model_lstm.add(LSTM(100))
model_lstm.add(Dense(1, activation='linear'))  # Assuming regression task, change activation fo

# Compile the model
model_lstm.compile(optimizer='adam', loss='mean_squared_error')  # Use appropriate loss functio

# Train the model
model_lstm.fit(X_train, y_train, epochs=5, batch_size=32, validation_split=0.2)

# Convert 'X_test' to numpy array
X_test_np = X_test.toarray() if isinstance(X_test, scipy.sparse.csr.csr_matrix) else X_test

# Evaluate the LSTM model
loss_lstm = model_lstm.evaluate(X_test_np, y_test)
print(f"Test Loss (LSTM): {loss_lstm}")
```

```
Epoch 1/5
20/20 [==============================] - 4s 113ms/step - loss: 8.3530 - val_loss: 1.1814
Epoch 2/5
20/20 [==============================] - 2s 91ms/step - loss: 1.3383 - val_loss: 1.1870
Epoch 3/5
20/20 [==============================] - 2s 120ms/step - loss: 1.2404 - val_loss: 1.1936
```

```
Epoch 4/5
20/20 [==============================] - 4s 198ms/step - loss: 1.2131 - val_loss: 1.1852
Epoch 5/5
20/20 [==============================] - 3s 163ms/step - loss: 1.2098 - val_loss: 1.1826
<ipython-input-32-be6b39a89595>:47: DeprecationWarning: Please use `csr_matrix` from the `s
  X_test_np = X_test.toarray() if isinstance(X_test, scipy.sparse.csr.csr_matrix) else X_te
7/7 [==============================] - 1s 25ms/step - loss: 0.9965
Test Loss (LSTM): 0.9965435266494751
```

Model 2: Recurrent Neural Network (RNN) with LSTM

Training Loss and Test Loss:

The training loss decreases over epochs, indicating that the model is learning from the training data.

The test loss is the mean squared error on a separate test dataset, and it appears reasonably low.

Analysis:

The model seems to learn and generalize well, as indicated by the decreasing training and validation losses. The test loss is relatively low, suggesting good performance on new, unseen data.

Suggestions for Improvement:

Experiment with different neural network architectures. Fine-tune hyperparameters such as the number of LSTM units, embedding dimensions, and batch size. Explore techniques for handling imbalanced datasets if applicable.

EXTRA CHALLENGING PROBLEMS :

Oversampling using the Synthetic Minority Over-sampling Technique (SMOTE) and change the text representation to Term Frequency-Inverse Document Frequency (TF-IDF):

This code introduces SMOTE for oversampling the minority class and replaces CountVectorizer with TfidfVectorizer for TF-IDF representation. It helps in addressing class imbalance and improving the model's performance.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report, confusion_matrix
from imblearn.over_sampling import SMOTE
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming 'data' is your DataFrame
# Replace 'Rating' with your actual label column name
# Drop any rows with missing values
data.dropna(subset=['Review Text', 'Rating'], inplace=True)

# Convert ratings to numeric values if they are categorical
data['Label'] = data['Rating'].astype(int)

# Split the data into features (X) and labels (y)
X = data['Review Text']
y = data['Label']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Apply TF-IDF representation using TfidfVectorizer
vectorizer = TfidfVectorizer()
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)

# Apply SMOTE for oversampling
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train_tfidf, y_train)

# Train the Multinomial Naive Bayes model
naive_bayes = MultinomialNB()
naive_bayes.fit(X_train_resampled, y_train_resampled)

# Make predictions on the test set
y_pred = naive_bayes.predict(X_test_tfidf)

# Display classification report
classification_rep = classification_report(y_test, y_pred)
print(f"Classification Report (Multinomial Naive Bayes):\n{classification_rep}")

# Display confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title('Confusion Matrix (Multinomial Naive Bayes)')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
Classification Report (Multinomial Naive Bayes):
              precision    recall  f1-score   support

           1       0.28      0.44      0.34       148
           2       0.25      0.34      0.29       309
           3       0.35      0.38      0.37       588
           4       0.35      0.44      0.39       939
           5       0.85      0.69      0.76      2545
```

LOG FILE:

WEEK1:

5/11/23- TASK1 -

Researching and identifying the business problem.

Study scope of the project topic.

Formulate a clear problem statement, motivation, and objectives.

Define the expected result.

WEEK2:

12/11/23- TASK2 -

Select a suitable dataset.

Visualize the initial data to understand its distribution and summary.

visualize the dataset labels.