NLP LAB EXERCISE 3

TEXT SUMMARIZATION-TRISHAA S

PIPELINE

Preprocessing:

Clean text, remove noise.

Tokenize into words.

Segment into sentences.

Text Representation:

Convert to numerical format (e.g., BoW, TF-IDF, word embeddings).

Sentence Scoring:

Define features for each sentence.

Assign scores based on features.

Sentence Selection:

Set a score threshold.

Rank and select top sentences.

Summarization:

Fuse selected sentences.

Control summary length.

Verify fluency and coherence.

Post-processing:

Refine using original document.

Polish for grammar and style.

Generate the final summarized output.

```
### IMPORTING MODULES
import re
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize, sent_tokenize
from sklearn.feature_extraction.text import CountVectorizer

nltk.download('punkt')
nltk.download('stopwords')
def preprocess_text(text):
    # Convert to lowercase
    text = text.lower()

    # Remove special characters and numbers
    text = re.sub(r'[^a-zA-Z\s]', '', text)

    # Tokenize the text into sentences and then words
    sentences = sent_tokenize(text)
    words = [word_tokenize(sentence) for sentence in sentences]

    # Remove stop words
    stop_words = set(stopwords.words('english'))
    words = [[word for word in sentence if word not in stop_words] for sentence in words]

    # Join the words back into sentences
    sentences = [' '.join(sentence) for sentence in words]

    # Join sentences into a single string
    preprocessed_text = ' '.join(sentences)
```

```
    return preprocessed_text


original_text = """
```

Two senior officials at one of China's top commercial banks have reportedly disappeared after funds worth up to $120m (£64m) went missing.

The pair both worked at Bank of China in the northern city of Harbin, the South China Morning Post reported. The latest scandal at Bank of China will do nothing to reassure foreign investors that China's

big four banks are ready for international listings. Government policy sees the bank listings as vital economic reforms. Bank of China is one of two frontrunners in the race to list overseas. The other is China Construction Bank. Both are expected to list abroad during 2005.

They shared a $45bn state bailout in 2003, to help clean up their balance sheets in preparation for a foreign stock market debut.

However, a report in the China-published Economic Observer said on Monday that the two banks may

have scrapped plans to list in New York because of the cost of meeting regulatory requirements imposed since the Enron scandal. Bank of China is the country's biggest foreign exchange dealer, while

China Construction Bank is the largest deposit holder. China's banking sector is burdened with at least

$190bn of bad debt according to official data, though most observers believe the true figure is far higher. Officially, one in five loans is not being repaid. Attempts to strengthen internal controls and tighten lending policies have uncovered a succession of scandals involving embezzlement by bank officials and loans-for-favours. The most high-profile case involved the ex-president of Bank of China, Wang Xuebing, jailed for 12 years in 2003. Although, he committed the offences whilst running Bank of China in New York, Mr.Wang was head of China Construction Bank when the scandal broke. Earlier this month, a China Construction Bank branch manager was jailed for life in a separate case.

China's banks used to act as cash offices for state enterprises and did not require checks on credit worthiness. The introduction of market reforms has been accompanied by attempts to modernize the

banking sector, but links between banks and local government remain strong. Last year, China's premier, Wen Jiabao, targeted bank lending practices in a series of speeches, and regulators ordered

all big loans to be scrutinized, in an attempt to cool down irresponsible lending. China's leaders see reforming the top four banks as vital to distribute capital to profitable companies and protect the health

of China's economic boom. But two problems persist. First, inefficient state enterprises continue to receive protection from bankruptcy because they employ large numbers of people. Second, many questionable loans come not from the big four, but from smaller banks. Another high-profile financial firm, China Life, is facing shareholder lawsuits and a probe by the US Securities and Exchange Commission following its 2004 New York listing over its failure to disclose accounting irregularities at its parent company.

"""

```
preprocessed_text = preprocess_text(original_text)
print(preprocessed_text)

vectorizer = CountVectorizer()
bow_representation = vectorizer.fit_transform([preprocessed_text])

bow_array = bow_representation.toarray()

# Display the Bag of Words representation
print("Bag of Words Representation:")
print(bow_array)
print("Feature names:", vectorizer.get_feature_names_out())
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vectorizer = TfidfVectorizer()
tfidf_representation = tfidf_vectorizer.fit_transform([preprocessed_text])
tfidf_array = tfidf_representation.toarray()

# Display the TF-IDF representation
print("TF-IDF Representation:")
```

```
print(tfidf_array)
print("Feature names:", tfidf_vectorizer.get_feature_names_out())
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Embedding, Lambda
from tensorflow.keras.preprocessing.text import Tokenizer
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

# Convert the corpus to a sequence of integers
tokenizer = Tokenizer()
tokenizer.fit_on_texts(preprocessed_text)
sequences = tokenizer.texts_to_sequences(preprocessed_text)
print("After converting our words in the corpus \
into vector of integers:")
print(sequences)
from gensim.models import Word2Vec
from nltk.tokenize import word_tokenize, sent_tokenize
sentences = [word_tokenize(sent) for sent in sent_tokenize(preprocessed_text)]

# Generate Skip-gram model
skipgram_model = Word2Vec(sentences, vector_size=100, window=5, sg=1, min_count=1)

# Get the word vectors for each sentence
sentence_vectors_skipgram = []
for sentence in sentences:
    sentence_vector = sum(skipgram_model.wv[word] for word in sentence if word in
skipgram_model.wv)
    sentence_vectors_skipgram.append(sentence_vector)
```

```
# Display the Skip-gram representations for each sentence
for i, vector in enumerate(sentence_vectors_skipgram):
    print(f"Sentence {i + 1} Skip-gram Representation:")
    print(vector)
    print()



import spacy

nlp = spacy.load("en_core_web_sm")
doc = nlp(preprocessed_text)

# Get the Word2Vec vectors
word_vectors_spacy = [token.vector for token in doc]
print("Word Vectors (spaCy Word2Vec):")
print(word_vectors_spacy)

!wget http://nlp.stanford.edu/data/glove.6B.zip
!unzip glove*.zip
!git clone https://github.com/facebookresearch/fastText.git
import fasttext
import fasttext.util
#fasttext.download_model('en', if_exists='ignore')
from nltk.tokenize import word_tokenize
import numpy as np

# Load pre-trained GloVe word vectors
glove_file_path = '/content/glove.6B.100d.txt'  # Replace with the actual path to your GloVe file

def load_glove_model(glove_file):
    with open(glove_file, 'r', encoding='utf-8') as file:
```

```
        word_vectors = {}
        if len(file.readline().split()) > 2:
            file.seek(0)
        for line in file:
            values = line.split()
            word = values[0]
            vector = np.array(values[1:], dtype='float32')
            word_vectors[word] = vector
    return word_vectors

# Load GloVe model
glove_model = load_glove_model(glove_file_path)

sentences = [word_tokenize(sent) for sent in sent_tokenize(preprocessed_text)]

sentence_vectors_glove = []
for sentence in sentences:
    sentence_vector = np.mean([glove_model[word] for word in sentence if word in glove_model],
axis=0)
    sentence_vectors_glove.append(sentence_vector)

# Display the GloVe representations for each sentence
for i, vector in enumerate(sentence_vectors_glove):
    print(f"Sentence {i + 1} GloVe Representation:")
    print(vector)
    print()

from sklearn.feature_extraction.text import TfidfVectorizer
from gensim.models import Word2Vec
import numpy as np
import spacy
```

```
import fasttext
from nltk.tokenize import word_tokenize, sent_tokenize

import spacy



# Tokenize sentences
sentences = [word_tokenize(sent) for sent in sent_tokenize(preprocessed_text)]

# TF-IDF
tfidf_vectorizer = TfidfVectorizer()
tfidf_representation = tfidf_vectorizer.fit_transform([preprocessed_text])
tfidf_array = tfidf_representation.toarray()
print("TF-IDF Representation:")
print(tfidf_array)

# CBOW (using Gensim)
cbow_model = Word2Vec(sentences, vector_size=100, window=5, sg=0, min_count=1)
word_vectors_cbow = cbow_model.wv
print("Word Vectors (CBOW):")
print(word_vectors_cbow)

# Skip-gram (using Gensim)
skipgram_model = Word2Vec(sentences, vector_size=100, window=5, sg=1, min_count=1)
word_vectors_skipgram = skipgram_model.wv
print("Word Vectors (Skip-gram):")
print(word_vectors_skipgram)

# Word2Vec (spaCy)
nlp = spacy.load("en_core_web_sm")
doc = nlp(preprocessed_text)
```

```
word_vectors_spacy = [token.vector for token in doc]
print("Word Vectors (spaCy Word2Vec):")
print(word_vectors_spacy)

# GloVe
glove_model_path = '/content/glove.6B.100d.txt'
glove_model = {}
with open(glove_model_path, 'r', encoding='utf-8') as file:
    for line in file:
        values = line.split()
        word = values[0]
        vector = np.array(values[1:], dtype='float32')
        glove_model[word] = vector

sentence_vectors_glove = [np.mean([glove_model[word] for word in sentence if word in
glove_model], axis=0)
                          for sentence in sentences]
print("GloVe Sentence Representations:")
print(sentence_vectors_glove)



from sklearn.metrics.pairwise import cosine_similarity
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import spacy
import fasttext
from nltk.tokenize import word_tokenize, sent_tokenize

# Tokenize sentences
sentences = [word_tokenize(sent) for sent in sent_tokenize(preprocessed_text)]
```

```
# CBOW (using Gensim)
cbow_model = Word2Vec(sentences, vector_size=100, window=5, sg=0, min_count=1)
sentence_vectors_cbow = [np.mean([cbow_model.wv[word] for word in sentence if word in cbow_model.wv], axis=0)
                         for sentence in sentences]

# Skip-gram (using Gensim)
skipgram_model = Word2Vec(sentences, vector_size=100, window=5, sg=1, min_count=1)
sentence_vectors_skipgram = [np.mean([skipgram_model.wv[word] for word in sentence if word in skipgram_model.wv], axis=0)
                             for sentence in sentences]

# Word2Vec (spaCy)
nlp = spacy.load("en_core_web_sm")
doc = nlp(preprocessed_text)
sentence_vectors_spacy = [np.mean([token.vector for token in sent] or [np.zeros_like(token.vector) for token in sent], axis=0)
                          for sent in doc.sents]

# GloVe
glove_model_path = '/content/glove.6B.100d.txt'
glove_model = {}
with open(glove_model_path, 'r', encoding='utf-8') as file:
    for line in file:
        values = line.split()
        word = values[0]
        vector = np.array(values[1:], dtype='float32')
        glove_model[word] = vector

sentence_vectors_glove = [np.mean([glove_model[word] for word in sentence if word in glove_model], axis=0)
```

```
                    for sentence in sentences]


# Define a function to compute cosine similarity between two vectors
def compute_cosine_similarity(vector1, vector2):
    return cosine_similarity([vector1], [vector2])[0][0]


# Compute cosine similarity between sentence vectors
cosine_similarity_matrix_cbow = np.array([[compute_cosine_similarity(sentence_vectors_cbow[i],
sentence_vectors_cbow[j])
                                for j in range(len(sentences))]
                               for i in range(len(sentences))])


cosine_similarity_matrix_skipgram =
np.array([[compute_cosine_similarity(sentence_vectors_skipgram[i],
                                                sentence_vectors_skipgram[j])
                                for j in range(len(sentences))]
                               for i in range(len(sentences))])


cosine_similarity_matrix_spacy = np.array([[compute_cosine_similarity(sentence_vectors_spacy[i],
sentence_vectors_spacy[j])
                                for j in range(len(sentences))]
                               for i in range(len(sentences))])


cosine_similarity_matrix_glove = np.array([[compute_cosine_similarity(sentence_vectors_glove[i],
sentence_vectors_glove[j])
                                for j in range(len(sentences))]
                               for i in range(len(sentences))])


print("CBOW Cosine Similarity:")
print(cosine_similarity_matrix_cbow)
```

```
print("\nSkip-gram Cosine Similarity:")
print(cosine_similarity_matrix_skipgram)

print("\nspaCy Word2Vec Cosine Similarity:")
print(cosine_similarity_matrix_spacy)

print("\nGloVe Cosine Similarity:")
print(cosine_similarity_matrix_glove)
```

Embedding techniques play a vital role in NLP by converting words or sentences into numerical representations. While traditional methods like Bag of Words and TF-IDF provide vector representations, they struggle to capture nuanced semantic relationships between words and sentences. In contrast, word embeddings such as Word2Vec, GloVe, and FastText offer dense vectors that capture semantic similarities effectively. However, contextual embeddings like BERT, GPT, and ELMo go even further by considering the entire context of words in sentences, providing rich, contextual representations. These embeddings, trained on large corpora, excel in capturing word semantics in various contexts and can be fine-tuned for specific tasks, making them highly effective in understanding the intricate nuances of language and improving the identification of semantic distances between sentences.

```
### Generating Summary
def generate_summary(text,feature_matrix,n):
    sentences = sent_tokenize(text)
    sentence_scores = cosine_similarity(feature_matrix[-1], feature_matrix[:-1])[0]
    print(sentence_scores)
    summary_sentences = nlargest(n, range(len(sentence_scores)), key=sentence_scores.__getitem__)
    summary = ' '.join([sentences[i] for i in sorted(summary_sentences)])
    summary = summary.split('. ')
    formatted_summary = '.\n'.join(summary)
    return formatted_summary
print(formatted_summary)
```