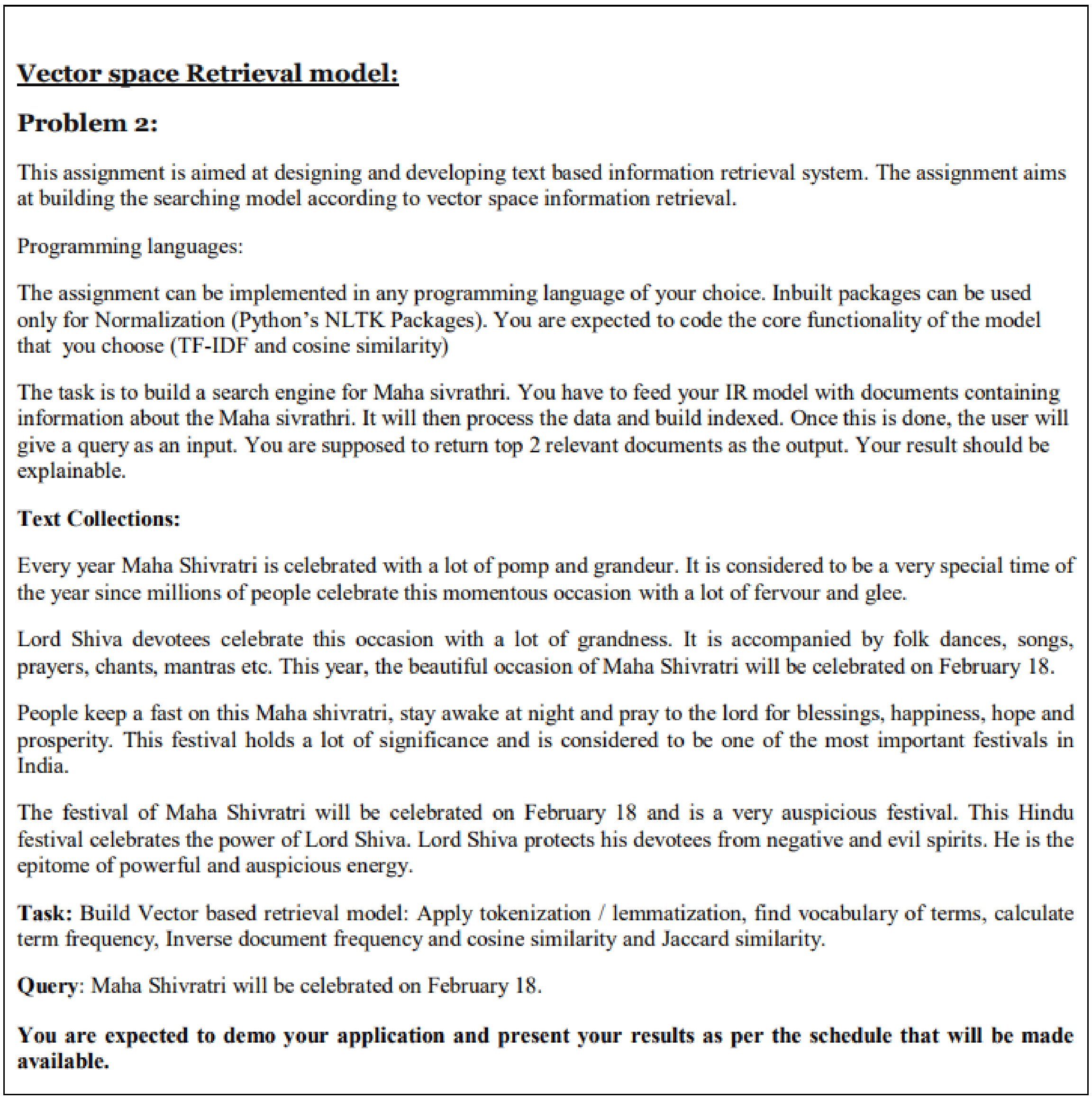
INFORMATION RETRIEVAL AND ORGANISATION(CSE3087)

DIGITAL ASSIGNMENT 2

TEAM MEMBERS:

1. PIYALI SAHA-20MIA1066
2. ZUBAIR MD-20MIA1135
3. APURV RAI-20MIA1086
4. SAKSHI CHAUHAN-20MIA1164

QUESTION:



<u>Vector space Retrieval model:</u>

Problem 2:

This assignment is aimed at designing and developing text based information retrieval system. The assignment aims at building the searching model according to vector space information retrieval.

Programming languages:

The assignment can be implemented in any programming language of your choice. Inbuilt packages can be used only for Normalization (Python's NLTK Packages). You are expected to code the core functionality of the model that you choose (TF-IDF and cosine similarity)

The task is to build a search engine for Maha sivrathri. You have to feed your IR model with documents containing information about the Maha sivrathri. It will then process the data and build indexed. Once this is done, the user will give a query as an input. You are supposed to return top 2 relevant documents as the output. Your result should be explainable.

Text Collections:

Every year Maha Shivratri is celebrated with a lot of pomp and grandeur. It is considered to be a very special time of the year since millions of people celebrate this momentous occasion with a lot of fervour and glee.

Lord Shiva devotees celebrate this occasion with a lot of grandness. It is accompanied by folk dances, songs, prayers, chants, mantras etc. This year, the beautiful occasion of Maha Shivratri will be celebrated on February 18.

People keep a fast on this Maha shivratri, stay awake at night and pray to the lord for blessings, happiness, hope and prosperity. This festival holds a lot of significance and is considered to be one of the most important festivals in India.

The festival of Maha Shivratri will be celebrated on February 18 and is a very auspicious festival. This Hindu festival celebrates the power of Lord Shiva. Lord Shiva protects his devotees from negative and evil spirits. He is the epitome of powerful and auspicious energy.

Task: Build Vector based retrieval model: Apply tokenization / lemmatization, find vocabulary of terms, calculate term frequency, Inverse document frequency and cosine similarity and Jaccard similarity.

Query: Maha Shivratri will be celebrated on February 18.

You are expected to demo your application and present your results as per the schedule that will be made available.

SOLUTION:

We can use the following steps to build a vector space model for the given task:

1. Tokenize the documents: We will use Python's Natural Language Toolkit (NLTK) package to tokenize the given documents. This will split the text into individual words or tokens.
2. Lemmatize the tokens: We will use the NLTK package to lemmatize the tokens. This will convert the tokens to their base form, which will help us reduce the number of unique words in the documents.
3. Create a vocabulary: We will create a set of all the unique words in the documents, which will serve as our vocabulary.
4. Calculate the term frequency: For each document, we will calculate the frequency of each word in the document. This will give us a vector representation of the document.
5. Calculate the inverse document frequency: We will calculate the inverse document frequency for each word in the vocabulary. This will help us reduce the weight of words that appear frequently in all documents.
6. Calculate the cosine similarity: We will use the cosine similarity measure to compare the query with each document. The documents with the highest cosine similarity to the query will be returned as the top 2 relevant documents.
7. Calculate the Jaccard similarity: We will also calculate the Jaccard similarity measure to compare the query with each document. The documents with the highest Jaccard similarity to the query will be returned as the top 2 relevant documents.

Data Preparation

First, we need to prepare the data by cleaning and processing it. We will tokenize the documents and apply lemmatization to reduce words to their base form. We will also remove stop words and punctuation marks.

```
import math

from collections import Counter

from nltk.tokenize import word_tokenize

from nltk.stem import WordNetLemmatizer

import nltk

nltk.download('punkt')

nltk.download('wordnet')

nltk.download('omw-1.4')

# Documents

docs =[ "Every year Maha Shivratri is celebrated with a lot of pomp and grandeur. It is considered to be a very special time of the year since millions of people celebrate this momentous occasion with a lot of fervour and glee.",      "Lord Shiva devotees celebrate this occasion with a lot of grandness. It is accompanied by folk dances, songs, prayers, chants, mantras etc. This year, the beautiful occasion of Maha Shivratri will be celebrated on February 18.",      "People keep a fast on this Maha shivratri,
stay awake at night and pray to the lord for blessings, happiness, hope and prosperity. This festival
holds a lot of significance and is considered to be one of the most important festivals in India.",
"The festival of Maha Shivratri will be celebrated on February 18 and is a very auspicious festival.
This Hindu festival celebrates the power of Lord Shiva. Lord Shiva protects his devotees from
negative and evil spirits. He is the epitome of powerful and auspicious energy"]


# Query
query = "Maha Shivratri will be celebrated on February 18."
# Preprocessing
lemmatizer = WordNetLemmatizer()


def preprocess(text):
    tokens = word_tokenize(text.lower())
    lemmas = [lemmatizer.lemmatize(token) for token in tokens]
    return lemmas


doc_tokens = [preprocess(doc) for doc in docs]
query_tokens = preprocess(query)
```

Vocabulary and Document Representation

Next, we need to build the vocabulary of terms from the documents and represent each

document

as a vector in the vector space. We will use the TF-IDF (Term Frequency-Inverse Document

Frequency) weighting scheme to represent the documents as vectors.

```
# Vocabulary of Terms
all_tokens_set = set([token for tokens in doc_tokens for token in tokens])
print("Vocabulary of terms:")
print(sorted(all_tokens_set))
# Term Frequency
doc_word_counts = [Counter(tokens) for tokens in doc_tokens]
doc_term_frequency = []
for word_count in doc_word_counts:
    frequencies = {word: count/sum(word_count.values()) for word, count in word_count.items()}

    doc_term_frequency.append(frequencies)

# Inverse Document Frequency

def inverse_document_frequency(term, all_docs_tokens):

    num_docs_with_term = sum(1 for doc in all_docs_tokens if term in doc)

    if num_docs_with_term > 0:

        return math.log(len(all_docs_tokens) / num_docs_with_term)

    else:

        return 0


doc_inverse_document_frequency = {token: inverse_document_frequency(token, doc_tokens) for
token in all_tokens_set}

# print results

print("Term frequency:")

for i, frequencies in enumerate(doc_term_frequency):

    print(f"Document {i+1}: {frequencies}")


print("Inverse document frequency:")

for term, idf in doc_inverse_document_frequency.items():

    print(f"{term}: {idf}")
```

Usage of cosine and Jaccard similarity

```
# cosine similarity

def cosine_similarity(doc_vector, query_vector):

    numerator = sum([doc_vector.get(term, 0) * query_vector.get(term, 0) for term in
set(doc_vector.keys()) & set(query_vector.keys())])

    denominator = math.sqrt(sum([count**2 for count in doc_vector.values()])) *
math.sqrt(sum([count**2 for count in query_vector.values()]))

    if denominator > 0:

        return numerator / denominator

    else:

        return 0
```

```
doc_vectors = []

for frequencies in doc_term_frequency:

   vector = {term: frequency*doc_inverse_document_frequency[term] for term, frequency in
frequencies.items()}

   doc_vectors.append(vector)


query_vector = {term: query_tokens.count(term)/len(query_tokens) *
doc_inverse_document_frequency[term] for term in query_tokens}


def cosine_similarity(doc_vector, query_vector):

   numerator = sum([doc_vector.get(term, 0) * query_vector.get(term, 0) for term in
set(doc_vector.keys()) & set(query_vector.keys())])

   denominator = math.sqrt(sum([count**2 for count in doc_vector.values()])) *
math.sqrt(sum([count**2 for count in query_vector.values()]))

   if denominator > 0:

      return numerator / denominator

   else:

      return 0


similarities = [cosine_similarity(doc_vector, query_vector) for doc_vector in doc_vectors]

print()

for i, score in enumerate(similarities):

   print(f"Document {i+1} cosine similarity score: {score}")
```

#Jaccard similarity

```
def Jaccard_Similarity(doc1, doc2):

   # List the unique words in a document

   words_doc1 = set(doc1.lower().split())

   words_doc2 = set(doc2.lower().split())

   # Find the intersection of words list of doc1 & doc2

   intersection = words_doc1.intersection(words_doc2)

   # Find the union of words list of doc1 & doc2

   union = words_doc1.union(words_doc2)
```

```
# Calculate Jaccard similarity score
# using length of intersection set divided by length of union set
return float(len(intersection)) / len(union)
```