Domain Specific IR SYSTEM For WIKI DOCS

Design & Documentation

NAME : ARJEE JACOB JACOB

BITS ID : 2019HT12111

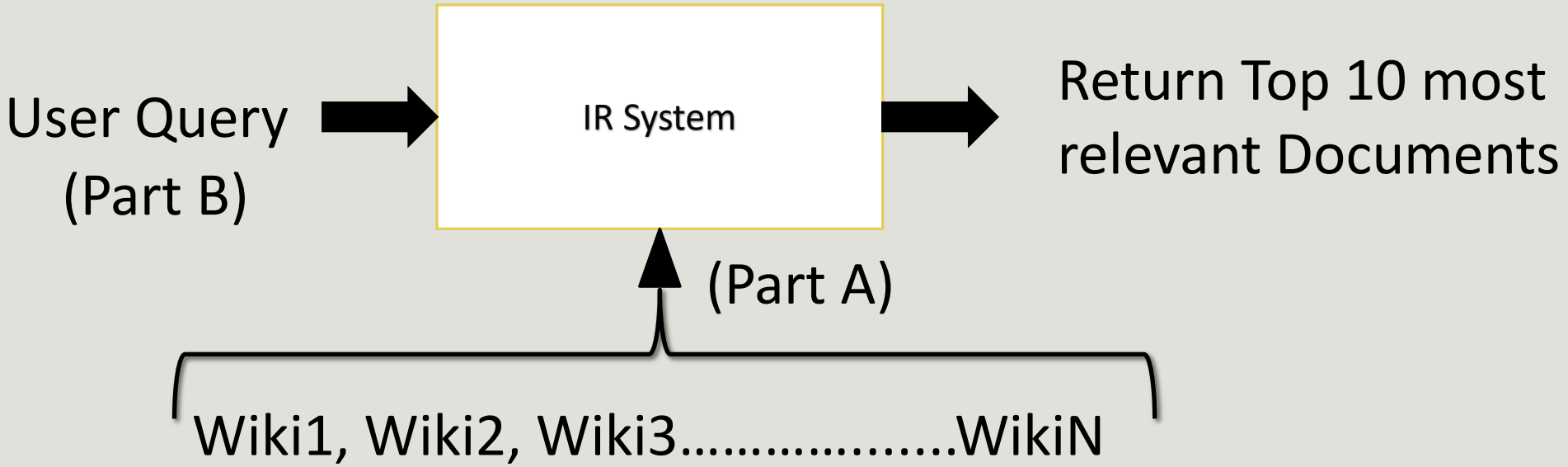
SUBJECT CODE : SSZG537

IDE : JUPYTER NOTEBOOK
PYTHON VERSION : 3.7.5

DESIGN

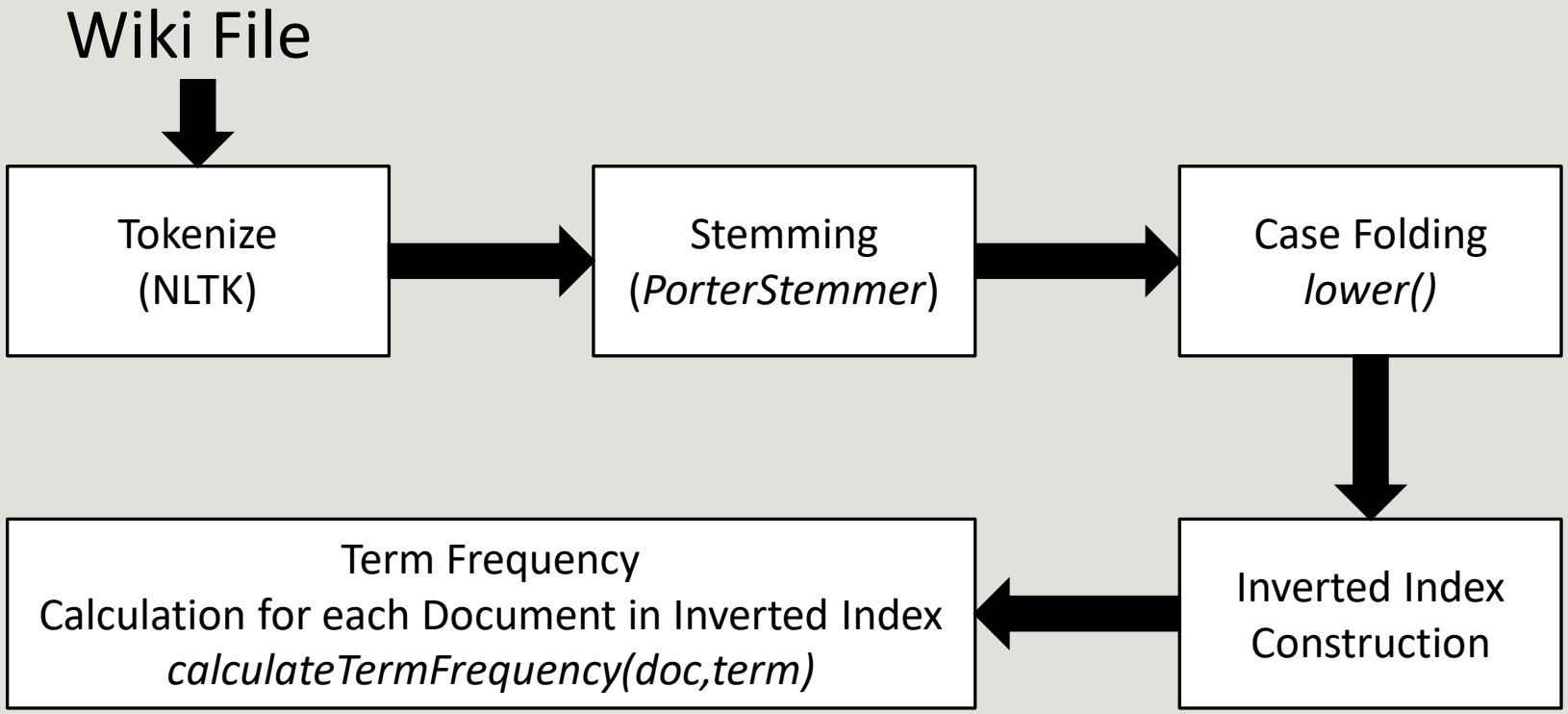
This Section provides information on the design of the system, the input source used, and the outputs expected as well as a report on a few query operations

HIGH LEVEL Architecture

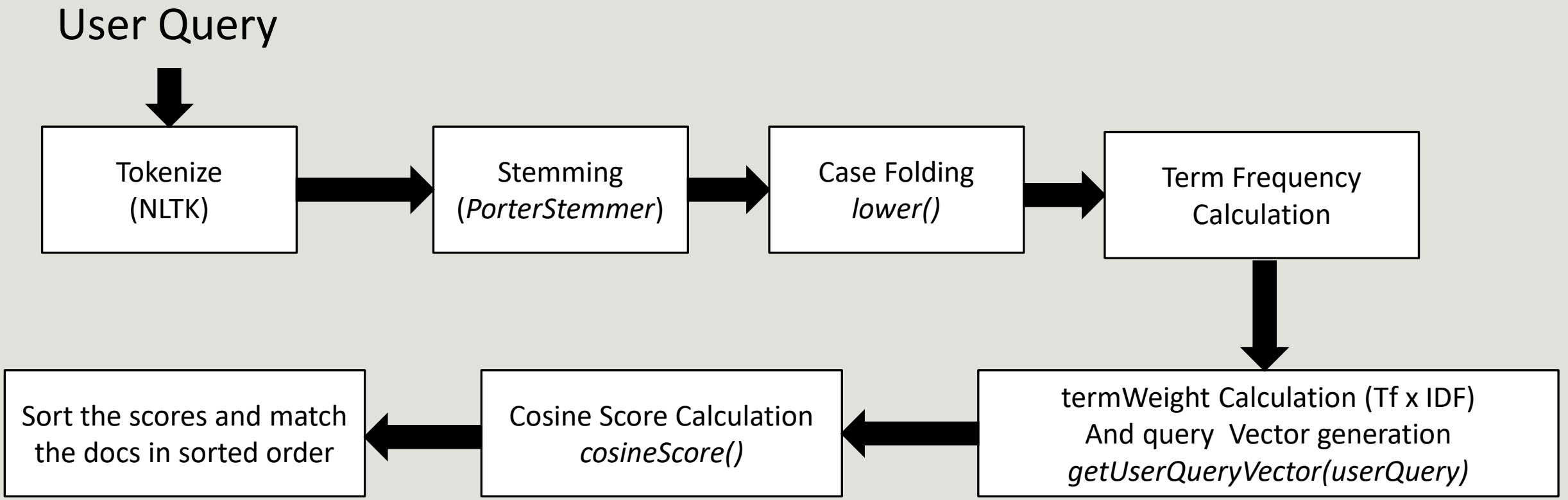


Each Wiki file is a collection of documents in xml format

IR SYSTEM Architecture – Part A



IR SYSTEM Architecture – Part B



SYSTEM Objectives

Tokenize the Wiki document and user Query

Apply Stemming using Porter Stemmer

Generate Inverted Index

Calculate TF-IDF Weights and Cosine Similarity

Return top 10 document IDs from input User Query

PYTHON LIBRARIES USED

NLTK – For Tokenizing and Porter Stemmer

Regular Expression Library “re” - for matching purposes

Math Library - To perform logarithmic operations for calculating Term Frequency

Json Library - To help in Inverted Index construction

EXECUTION TIME

Reading the Wiki File : 3-5 seconds

Tokenization, Stemming and Inverted Index Construction : 40-43 seconds

Considering other Fields to execute, Consider Total Time to be close to 1 min.

Keep an Eye out on the following 2 Code sections as they take time to execute.

START OF MAIN PROGRAM

First Tokenize the documents from the input file

```
1 print("Reading Wiki File. This will take a few seconds...")
2 soup = BeautifulSoup(open(wikiPath, encoding="utf8"), "html.parser")
3 print("Reading Complete. Generated Beautiful soup object from Wiki File")
```

```
Reading Wiki File. This will take a few seconds...
Reading Complete. Generated Beautiful soup object from Wiki File
```

Note : Executing the following will take a while. Please be patient.

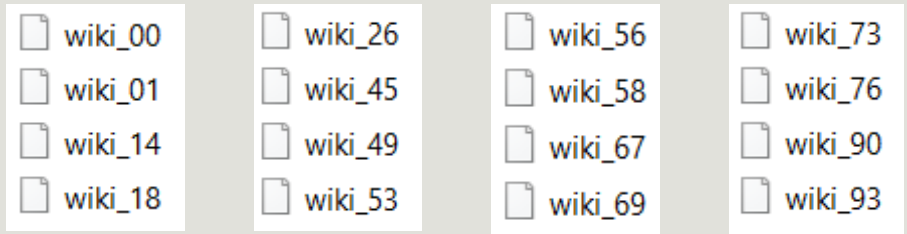
```
1 #Initializing the Inverted Index Structure
2 InvertedIndex = {'documentCount' : 0 , 'terms': {}}
3
4 #Create Porter Stemmer object for Stemming
5 ps = PorterStemmer()
6
7 print("Generating Tokens, Stemming and creating the Inverted Index.")
8 print("This will take a while. Please Wait till it says Complete...")
9
```

INPUT SOURCE

The input source is a wiki file. One wiki file will contain a collection of multiple documents along with their document ids in the form of an XML.

Wiki Files have been uploaded to the following google drive location
https://drive.google.com/drive/folders/1i54s7ouUQUkNIX1R65TRjUO3WwLoM3SO?usp=sharing

The Following Wiki Files have been Provided to try



In case the link doesn't work, 2 sample files have been attached to the zip file as well

How to execute part 1 - Input

First Provide the file path of “any one” of the Wiki Files in the **wikiPath** variable

```
#Provide the path of the wiki file here
wikiPath = "C:/Users/dnv786/Desktop/Zebra/Personal/Mtech/Semester2/IR/Assignment/Sem2Assignment/AA/wiki_18"
```

Each file contains a set of documents with **doc ids**, in xml format

```
1 <doc id="12" url="https://en.wikipedia.org/wiki?curid=12"
  title="Anarchism">
2 Anarchism
3
4 Anarchism is a <a href="political%20philosophy">political
  philosophy</a> that advocates <a
  href="self-governance">self-governed</a> societies based
  on voluntary institutions. These are often described as
  <a href="stateless%20society">stateless societies</a>,
  although several authors have defined them more
  specifically as institutions based on non-<a
  href="Hierarchy">hierarchical</a> <a
  href="Free%20association%20%28communism%20and%20anarchism%2
  9">free associations</a>. Anarchism considers the <a
```

Note:

An **InvertedIndex.json** will get generated in the location where the program is run. It is only an intermediate **output** that is generated for the user to inspect.

How to execute part 2 - Input

Provide your required Query at the Bottom of the Python *.ipynb* Document

Input User Query Here

The userQuery will be the input to the IR system

The query will then be input to the top most method i.e. retrieveTop10Docs

This will then retrieve the IDs of the Top 10 documents of the sample Query

```
1 userQuery = "What is Mahjong?"
2 retrieveTop10Docs(userQuery)
```

How to execute part 3 - Output

After providing the wiki file path & user query, start executing from the 1st cell to the last cell in sequence. The OUTPUT RESULT will be returned as below in the last cell.

```
1 userQuery = "What is the meaning of Anarchism?"
2 retrieveTop10Docs(userQuery)
```

```
Query Tokens       : ['what', 'is', 'the', 'mean', 'of', 'anarch']
Query Token Count  : {'what': 1, 'is': 1, 'the': 1, 'mean': 1, 'of': 1, 'anarch': 1}
Query Vector : {'what': 0.34516395356044266, 'is': 0.03557615426119612, 'the': 0.03451818910486234, 'mean': 0.2681487692693255
7, 'of': 0.031359669860032684, 'anarch': 2.1712387562612694}
Top 10 Documents that match the sample input query, with corresponding scores :
12 0.1306231786481284
1023 0.1093620818053134
339 0.08474952699982599
696 0.0661941066714508
1176 0.06263976031918828
1212 0.05136417468626282
1158 0.05088670516230048
643 0.04972585571201736
1160 0.04948656879034355
1309 0.0447406502511525
```

OUTPUT EXPLANATION

The first column shows the doc id in the wiki file

The second column shows the cosine score of the corresponding doc id.

Here the doc ids **12** and **1023** have high scores of **0.13** and **0.10** respectively which makes them more relevant to the query that the user had provided as input

```
Top 10 Documents that match the sample input query, with corresponding scores :
12 0.1306231786481284
1023 0.1093620818053134
339 0.084749526999825 99
696 0.06619410667145 08
1176 0.06263976031918828
1212 0.05136417468626282
1158 0.05088670516230048
643 0.04972585571201736
1160 0.04948656879034355
1309 0.0447406502511525
```

EXAMPLE QUERY REPORT 1

Query	Wiki File	Top 10 Document IDs returned	Document-Query Score	Search Relevance
Who is Jimi Hendrix?	wiki_14	15181	0.157144018	Yes
		15268	0.083441986	No
		15521	0.075422853	No
		15624	0.075214989	No
		15422	0.072996678	No
		15210	0.071610995	No
		15695	0.069093887	No
		15125	0.06869318	No
		15782	0.067903674	No
		15447	0.064475963	No

EXAMPLE QUERY REPORT 2

Query	Wiki File	Top 10 Document IDs returned	Document-Query Score	Search Relevance
What is Mahjong?	wiki_18	19496	0.177541521	Yes
		19328	0.046862029	No
		19327	0.045480866	No
		20041	0.039803807	No
		19581	0.031829796	No
		19958	0.031362642	No
		19372	0.029993652	No
		19738	0.029105519	No
		19719	0.028389043	No
		20088	0.025843326	No

DOCUMENTATION

This Section provides information implementation methods and Libraries used to achieve in retrieving the documents based on their relevance

INVERTED INDEX CONSTRUCTION – 1 Defining the Structure

The **InvertedIndex.json** will be an intermediate file generated

This will be generated in the same location as where the python files are located

```
indexFilePath = 'InvertedIndex.json'
```

We use a JSON Data Structure to define the Inverted Index Structure

```
#Initializing the Inverted Index Structure
InvertedIndex = {'documentCount' : 0 , 'terms': {}}
```

INVERTED INDEX CONSTRUCTION – 2 Preprocessing

READING THE WIKI FILE

We use the BeautifulSoup library to parse the Wiki File

```
1 print("Reading Wiki File. This will take a few seconds...")
2 soup = BeautifulSoup(open(wikiPath, encoding="utf8"), "html.parser")
3 print("Reading Complete. Generated Beautiful soup object from Wiki File")
```

```
Reading Wiki File. This will take a few seconds...
Reading Complete. Generated Beautiful soup object from Wiki File
```

BeautifulSoup is a powerful library to parse the XML/HTML based files.

INVERTED INDEX CONSTRUCTION – 3 Preprocessing

TOKENIZATION & STEMMING

Next we use the NLTK library to perform Tokenization operations

```
#Tokenize the Each document using  NLTK Tokenizer
documentTokens = nltk.word_tokenize(eachDocument.get_text())
```

We also use the PorterStemmer class to do Stemming operations

```
#Create Porter Stemmer object for Stemming
ps = PorterStemmer()
```

```
#Apply Stemming operation using Porter Stemmer
stemmedTokens = [ps.stem(eachWord) for eachWord in documentTokens]
```

INVERTED INDEX CONSTRUCTION – 4 Preprocessing

CASE-FOLDING AND REMOVAL OF PUCTUATIONS & WHITESPACES

We use the Regular Expression Library library to match non-word types and remove them as well as perform case-folding using *lower()* method

```
#Bring all tokens to lower case
cleanTokens = []
for eachToken in stemmedTokens:
    #Checks if Punctuations are not present like [. , "'"] etc
    if not re.match(r'^[_\W]+$',eachToken):
        cleanTokens.append(eachToken.lower())
```

INVERTED INDEX CONSTRUCTION – 5 Index Building

The program will next, build on the Inverted index using the pre-processed tokens whenever new tokens are encountered in each document referred by the doc id.

```
for eachToken in cleanTokens:
    if eachToken not in InvertedIndex['terms']:
        #Initialize the Inverted Index structure for every New term
        InvertedIndex['terms'][eachToken] = {'docs' : {docID : {'count':1,'tf':0}},'total' : 1, 'docs_count':1}
    else:
        if docID not in InvertedIndex['terms'][eachToken]['docs']:
            InvertedIndex["terms"][eachToken]['docs'][docID] = {'count':1,'tf': 0}
            InvertedIndex["terms"][eachToken]['docs_count'] +=1
            InvertedIndex["terms"][eachToken]['total'] +=1
        else:
            InvertedIndex["terms"][eachToken]['docs'][docID]['count'] +=1
            InvertedIndex["terms"][eachToken]['total'] +=1
```

INVERTED INDEX CONSTRUCTION – 6 Resultant Inverted Index

This is a snippet of how the tokenized, stemmed and case-folded Inverted Index will look like.


```
{"documentCount": 479, "terms": {"economi": {"docs": {
"19296": {"count": 48, "tf": 2.681241237375587,
"tf_norm": 0.04309554961302 21}, "19300": {"count": 3,
"tf": 1.477121254719662 4, "tf_norm":
0.03345527896013 2964}, "19301": {"count": 7, "tf":
1.8450980400142567, "tf_norm": 0.03175005475099545},
"19302": {"count": 1, "tf": 1.0, "tf_norm":
0.02954080949940489}, "19305": {"count": 1, "tf": 1.0,
"tf_norm": 0.03472208633093 2524}, "19310": {"count": 2,
 "tf": 1.3010299956639813, "tf_norm":
0.05739255919417131}, "19323": {"count": 6, "tf":
```

CALCULATING TERM FREQUENCY

$$w_{t,d} = \begin{cases} 1 + \log_{10} \mathrm{tf}_{t,d}, & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0, & \text{otherwise} \end{cases}$$

Code Snippet : Imported Math Library to perform log operations

```
1  def calculateTermFrequency(doc, term):
2      if doc not in InvertedIndex["terms"][term]['docs']: return 1
3      else:
4          count = InvertedIndex["terms"][term]['docs'][doc]['count']
5          termFrequency = 1 + math.log(count, 10)
6      return termFrequency
```

INVERSE DOC FREQUENCY & TERM WEIGHT CALCULATION

$$\mathrm{idf}_t = \log_{10}(N/\mathrm{df}_t)$$

$$\mathrm{w}_{t,d} = (1+\log \mathrm{tf}_{t,d}) \times \log_{10}(N/\mathrm{df}_t)$$

Code Snippet : Imported Math Library to perform log operations

```
# Inverse Document Frequency = log(N/docFrequency) , Where N - Total Documents in Inverted Index
inverseDocumentFrequency = math.log(totalDocumentsInInvertedIndex/documentFrequency, 10)

## Term Weight = Term Frequency x Inverse Document Frequency
termWeight = termFrequency * inverseDocumentFrequency
```

Cosine Normalization

$$c\ (\text{cosine}) \qquad \frac{1}{\sqrt{w_1^2+w_2^2+\ldots+w_M^2}}$$

Code Snippet : Using the sqrt and math functions, we perform normalization operation

```
document_norm = {}
for eachTerm in InvertedIndex["terms"]:
    for docID in InvertedIndex["terms"][eachTerm]['docs']:
        termFrequency = InvertedIndex["terms"][eachTerm]['docs'][docID]['tf']
        if docID not in document_norm:
            document_norm[docID] = termFrequency**2
        else:
            document_norm[docID] += termFrequency**2

document_norm = {key: math.sqrt(value) for key, value in document_norm.items()}
print(document_norm)
```

Document-Query Score Calculation

The score for a document-query pair is calculated by summation over the terms in both the query and the document

$$\text{Score}(q,d) = \sum_{t \in q \cap d} W_{t,\,d}$$

Code Snippet on calculating Document-Query Score.

```
def cosineScore(userQueryVector):
    scores = {}
    for token, qnorm in userQueryVector.items():
        for doc, data in InvertedIndex["terms"][token]['docs'].items():
            score = qnorm * data['tf_norm']
            if doc not in scores:
                scores[doc] = score
            else:
                scores[doc] += score
    return scores
```

FINAL RETRIEVAL OF TOP 10 DOCS

The final step in the algorithm is to retrieve the docs in sorted score order. For this we use the *sorted* method to sift through the acquired score set of document-query score pairs. We break the loop the moment count reaches 10.

```
def retrieveTop10Docs(userQuery):
    userQueryVector = getUserQueryVector(userQuery)
    print("Query Vector : " + str(userQueryVector))
    scores = cosineScore(userQueryVector)
    top10Count = 0
    #Retrieve the first 10 documents from the document list that has the best scores matching the input sample query
    print('Top 10 Documents that match the sample input query, with corresponding scores : ')
    for k, v in sorted(scores.items(), key=lambda item: item[1], reverse=True):
        print(k, v)
        top10Count += 1
        if top10Count >= 10:
            break
```

THANK YOU

“Torture the data, and it will confess to anything.”

– Ronald Coase