

Information Retrieval
Assignment-2
Atishay Jain (MT22021)
Rishika Jain (MT22055)

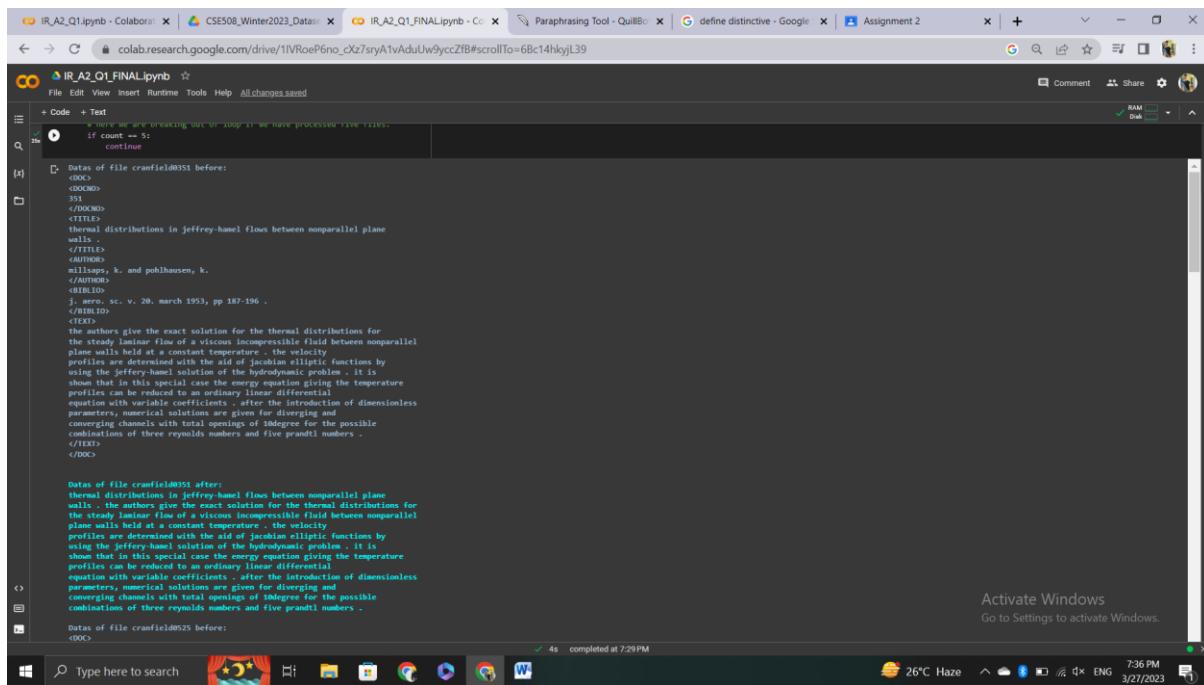
Question 1

For this question, we have used the following python libraries:

- Pandas, Numpy.
- NLTK library for pre-processing the data.

Pre-Processing:

1. We have used the `os.listdir()` function to read data files from the directory.
2. Read the contents of every file when it was opened, then put it in a dataframe.
3. Changed every word to lower case.
4. Eliminated all punctuation and special letters.
5. Using NLTK stopwords, all stopwords were eliminated from the corpus.
6. Tokenization of the document corpus was done.



The screenshot shows a Google Colab interface with multiple tabs open. The active tab is 'IR_A2_Q1_FINAL.ipynb'. The code cell contains Python code for processing files, and the output cell displays XML code representing the processed documents. The XML includes tags like <DOC>, <DOCID>, <TITLE>, <AUTHOR>, <PUBLISHER>, and <TEXT>. The text in the XML describes the flow of a viscous incompressible fluid between nonparallel plane walls held at a constant temperature, mentioning the use of Jacobi elliptic functions and dimensionless parameters to solve the hydrodynamic problem.

```
if count == 5:
    continue

<DOC>
<DOCID>
<TITLE>
<AUTHOR>
<PUBLISHER>
<TEXT>
</DOC>
```

Activate Windows
Go to Settings to activate Windows.

IR_A2_Q1.ipynb - Colaboratory

File Edit View Insert Runtime Tools Help All changes saved

[61] # printing the dataframes
print(filedata)

```
filename Data
0 cransfile00531 thermal distributions in jeffrey-hamel flows b...
1 cransfile00925 on hypersonic viscous flow over an insulated f...
2 cransfile00792 some low speed problems of high speed aircraft...
3 cransfile00332 similitude of hypersonic real-gas flows over s...
4 cransfile00804 theory of stagnation point heat transfer in di...
...
1395 cransfile00701 numerical determination of indicial lift of a t...
1396 cransfile1001 pressures, densities and temperatures in the u...
1397 cransfile00882 the variation of gust frequency with gust velo...
1398 cransfile00395 correlation of theoretical and photo-thermome...
1399 cransfile0221 effect of uniformly distributed roughness on t...
```

[1400 rows x 2 columns]

[62] # here our preprocessing starts

[63] # here we are performing tokenization on the text
wordToken = nltk.tokenize.WhitespaceTokenizer()

[64] def tokenize_text(text):
 return wordToken.tokenize(text)

[65] # here we are lowercasing the text
filedata['Data']=filedata['Data'].str.lower()
print(color.BOLD+color.DARKCYAN+"After lowercase : ")
print(filedata)

After lowercase :

```
filename Data
0 cransfile00531 thermal distributions in jeffrey-hamel flows b...
1 cransfile00925 on hypersonic viscous flow over an insulated f...
2 cransfile00792 some low speed problems of high speed aircraft...
3 cransfile00332 similitude of hypersonic real-gas flows over s...
4 cransfile00804 theory of stagnation point heat transfer in di...
...
1395 cransfile00701 numerical determination of indicial lift of a t...
1396 cransfile1001 pressures, densities and temperatures in the u...
1397 cransfile00882 the variation of gust frequency with gust velo...
1398 cransfile00395 correlation of theoretical and photo-thermome...
1399 cransfile0221 effect of uniformly distributed roughness on t...
```

```

# Here we are performing tokenization
filedata['Data_tokens'] = filedata.Data.apply(tokenized_text)
print(color.BOLDGREEN+ "After tokenization : ")
print(filedata)

After tokenization :
filename Data Data_tokens
0 cranfield0351 thermal distributions jeffrey hamel flows nonp... [thermal, distributions, jeffrey, hamel, flows, ...
1 cranfield0525 hypersonic viscous flow insulated flat plate ... [hypersonic, viscous, flow, insulated, flat, p...
2 cranfield0792 low speed problems high speed aircraft first p... [low, speed, problems, high, speed, aircraft, ...
3 cranfield0332 similitude hypersonic real gas flows slender b... [similitude, hypersonic, real, gas, flows, s...
4 cranfield0024 theory stagnation point heat transfer dissoci... [theory, stagnation, point, heat, transfer, di...
...
1395 cranfield0701 numerical determination indicl lift two dimen... [numerical, determination, indicl, lift, two, ...
1396 cranfield1103 pressures densities temperatures upper atmosph... [pressures, densities, temperatures, upper, at...
1397 cranfield0882 variation gust frequency gust velocity altitud... [variation, gust, frequency, gust, velocity, a...
1398 cranfield195 correlation theoretical photo thermoelastic re... [correlation, theoretical, photo, thermoelasti...
1399 cranfield1212 effect uniformly distributed roughness turbule... [effect, uniformly, distributed, roughness, tu...

[1400 rows x 3 columns]

```

[69] FileData

filename	Data	Data_tokens
0	cranfield0351	[thermal, distributions, jeffrey, hamel, flows, ...]
1	cranfield0525	[hypersonic, viscous, flow, insulated, flat, p...
2	cranfield0792	[low, speed, problems, high, speed, aircraft, ...]
3	cranfield0332	[similitude, hypersonic, real, gas, flows, s...
4	cranfield0024	[theory, stagnation, point, heat, transfer, di...
...
1395	cranfield0701	[numerical, determination, indicl, lift, two, ...]
1396	cranfield1103	[pressures, densities, temperatures, upper, at...
1397	cranfield0882	[variation, gust, frequency, gust, velocity, a...
1398	cranfield195	[correlation, theoretical, photo, thermoelasti...
1399	cranfield1212	[effect, uniformly, distributed, roughness, tu...

```

else:
    TokenCount[word] = 1
    distinctiveTokens.append(word)

FileData

filename Data Data_tokens
0 cranfield0351 thermal distributions jeffrey hamel flows nonp... [thermal, distributions, jeffrey, hamel, flows, ...
1 cranfield0525 hypersonic viscous flow insulated flat plate ... [hypersonic, viscous, flow, insulated, flat, p...
2 cranfield0792 low speed problems high speed aircraft first p... [low, speed, problems, high, speed, aircraft, ...
3 cranfield0332 similitude hypersonic real gas flows slender b... [similitude, hypersonic, real, gas, flows, s...
4 cranfield0024 theory stagnation point heat transfer dissoci... [theory, stagnation, point, heat, transfer, di...
...
1395 cranfield0701 numerical determination indicl lift two dimen... [numerical, determination, indicl, lift, two, ...
1396 cranfield1103 pressures densities temperatures upper atmosph... [pressures, densities, temperatures, upper, at...
1397 cranfield0882 variation gust frequency gust velocity altitud... [variation, gust, frequency, gust, velocity, a...
1398 cranfield195 correlation theoretical photo thermoelastic re... [correlation, theoretical, photo, thermoelasti...
1399 cranfield1212 effect uniformly distributed roughness turbule... [effect, uniformly, distributed, roughness, tu...

[1400 rows x 3 columns]

```

[72] # here we are taking the users input through the function , pre-processing it and then returning the preprocessed query.
def Enterquery():
 query=input('Enter your query :')

Activate Windows
Go to Settings to activate Windows.

METHODOLOGY:

A. TF-IDF Matrix:

For each document, we generated a dictionary that lists the terms and the number of words that are used in that document.

After that, we determined the IDF score for each distinct word in the corpus.

IDF score is equal to the log of the total number of files divided by the length of the posting list for that word.

TF-IDF matrix for every Five scoring schemes:-

❖ **BINARY:** In the TF-IDF matrix, ROW represents "Number of Documents" and COLUMN represents "Number of Unique Words in the Word Corpus."

When a token is included in the document, the TF value is 1, otherwise it is 0.

Score for TF-IDF: $tf * idf$

The screenshot shows a Jupyter Notebook interface in Google Colab. The code in cell [81] defines a function for the 'Binary Weighting scheme' which initializes a matrix of zeros and iterates through documents to calculate TF-IDF scores. Cell [83] prints the resulting matrix, which is a sparse matrix of zeros with some non-zero values. Cell [84] prints the 'Raw Count Weighting scheme'. The notebook has tabs for 'IR_A2_Q1.ipynb', 'CSE508_Winter2023_Data...', 'IR_A2_Q1_FINAL.ipynb', 'Paraphrasing Tool - QuillBot...', 'define distinctive - Google...', and 'Assignment 2'. The status bar at the bottom shows system information like temperature, battery level, and network status.

```
[81] # Binary Weighting scheme
print(color.BOLD+color.BLUE+'Binary Weighting Scheme')

[82]
matrix_tf_idf_binary=np.zeros((FileData.shape[0],len(distinctiveTokens)))
value=1
for i in range(FileData.shape[0]):
    doc=meta_doc_dict[FileData.iloc[i,0]]
    tokens=list(doc.keys())
    for word in tokens:
        tf=value
        idf=idf_values[word]
        overall_score=tf*idf
        matrix_tf_idf_binary[i,distinctiveTokens.index(word)]=overall_score

[83] print(matrix_tf_idf_binary)
[[2.7783194 2.28138289 6.55108034 ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [2.7783194 2.28138289 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]

[84] # Raw Count Weighting scheme
print(color.BOLD+color.BLUE+'Raw Count Weighting Scheme')
```

```

len_df=FileData.shape[0]
query=EnterQuery()
final_ans={}
for x in range(len_df):
    score=0
    doc=matrix_tf_idf_binary[x,:]
    for token in query:
        try:
            score+=doc[distinctiveTokens.index(token)]
        except:
            pass
    final_ans[FileData.iloc[x,0]]=score

final_sorted_ans=dict(sorted(final_ans.items(),key=lambda item:item[1],reverse=True))
count=0
print(color.BOLD+color.DARKCYAN+"Using Binary Weighting Scheme")
for k,v in final_sorted_ans.items():
    count+=1

    print(f'{count}: {k}    score: {v}')
    if(count>5):
        break

```

Activate Windows
Go to Settings to activate Windows.

❖ **RAW COUNT:** In the TF-IDF matrix, ROW represents "Number of Documents" and COLUMN represents "Number of Unique Words in the Word Corpus."
TF value is stored in the dictionary.
Score for TF-IDF: $tf * idf$

```

matrix_tf_idf_raw_count=np.zeros((FileData.shape[0],len(distinctiveTokens)))
value=1
for i in range(FileData.shape[0]):
    doc=meta_doc_dict[FileData.iloc[i,0]]
    tokens=list(doc.keys())
    for word in tokens:
        tf=doc[word]
        idf=idf_values[word]
        overall_score=tf*idf
        matrix_tf_idf_raw_count[i,distinctiveTokens.index(word)]=overall_score

```

```

[[ 5.55663879  4.56276577  6.55108034 ...  0.       0.
  [ 0.       0.       0.       ...  0.       0.
  [ 0.       0.       0.       ...  0.       0.
  [ 0.       0.       0.       ...  0.       0.
  [ 0.       0.       0.       ...  0.       0.
  ...
  [ 0.       0.       0.       ...  0.       0.
  [ 0.       0.       0.       ...  0.       0.
  [13.89159698  2.28138289  0.       ...  0.       0.
  [ 0.       0.       0.       ...  0.       0.
  [ 0.       0.       0.       ...  0.       0.]]]

```

Activate Windows
Go to Settings to activate Windows.

```

len_df=FileData.shape[0]
query=EnterQuery()
final_ans={}
for x in range(len_df):
    score=0
    doc=matrix_tf_idf_raw_count[x,:]
    for token in query:
        try:
            score+=doc[distinctiveTokens.index(token)]
        except:
            pass
    final_ans[FileData.iloc[x,0]]=score

final_sorted_ans=dict(sorted(final_ans.items(),key=lambda item:item[1],reverse=True))
count=0
print(color.BOLD+color.DARKCYAN+"Using Raw Count Weighting Scheme")
for k,v in final_sorted_ans.items():
    count+=1
    print(f'{count}: {k} score: {v}')
    if(count>=5):
        break

```

Enter your query :numerical determination indic lift
Using Raw Count Weighting Scheme

Rank	Document ID	Score
1:	cranfield0701	39.21171446462013
2:	cranfield0699	26.908507788967743
3:	cranfield1380	23.389527371649205
4:	cranfield0702	20.500699256730076
5:	cranfield1291	18.711621897319365

Activate Windows
Go to Settings to activate Windows.

❖ **TERM FREQUENCY:** In the TF-IDF matrix, ROW represents "Number of Documents" and COLUMN represents "Number of Unique Words in the Word Corpus."

TF value is equals to the Word count stored in the dictionary created divided by the total number of words in that document.

TF-IDF score = tf * idf

```

[87] print(color.BOLD+color.BLUE+"Term frequency Weighting Scheme")
Term frequency Weighting Scheme

[88] matrix_tf_idf_term_freq=np.zeros((FileData.shape[0],len(distinctiveTokens)))
value=1
for i in range(FileData.shape[0]):
    doc=meta_doc_dict[FileData.iloc[i,0]]
    tokens=list(doc.keys())
    overall_tf=0
    for w in tokens:
        overall_tf+=doc[w]
    for word in tokens:
        tf=doc[word]/overall_tf
        idf=idf_values[word]
        overall_score=tf*idf
        matrix_tf_idf_term_freq[i,distinctiveTokens.index(word)]=overall_score

```

```

[89] print(matrix_tf_idf_term_freq)
[[ 0.07508971  0.061659  0.08852811 ... 0.       0.       0.      ]
 [ 0.       0.       ... 0.       0.       0.       ]
 [ 0.       0.       ... 0.       0.       0.       ]
 ...
 [ 0.       0.       ... 0.       0.       0.       ]
 [ 0.13891597  0.02281383 0.       ... 0.       0.       ]
 [ 0.       0.       ... 0.       0.       0.       ]]

```

```

[90] # log normalization Weighting scheme
print(color.BOLD+color.BLUE+"log normalization Weighting Scheme")
log_normalization Weighting Scheme

```

Activate Windows
Go to Settings to activate Windows.

```

len_df=fileData.shape[0]
query=Enterquery()
final_ans={}
for x in range(len_df):
    score=0
    doc=matrix_tf_idf_term_freq[x,:]
    for token in query:
        try:
            score+=doc[distinctiveTokens.index(token)]
        except:
            pass
    final_ans[fileData.iloc[x,0]]=score

final_sorted_ans=dict(sorted(final_ans.items(),key=lambda item:item[1],reverse=True))
count=0
print(color.BOLD+color.DARKCYAN+"Using Term Frequency Weighting Scheme")
for k,v in final_sorted_ans.items():
    count+=1

    print(f'{count}: {k} score: {v}')
    if(count>5):
        break

```

Enter your query :numerical determination indic lift
Using Term Frequency Weighting Scheme

Rank	Term	Score
1:	cranfield0701	0.2435102151937966
2:	cranfield0699	0.226312191419300623
3:	cranfield0702	0.20297121335767313
4:	cranfield0698	0.19736107421877824
5:	cranfield1256	0.17991944152637851

Activate Windows
Go to Settings to activate Windows.

❖ **LOG NORMALIZATION:** In the TF-IDF matrix, ROW represents "Number of Documents" and COLUMN represents "Number of Unique Words in the Word Corpus."
TF value = LOG(Word count stored in the dictionary created divided by the total number of words in that document)

TF-IDF score = $tf * idf$

```

[90] # log normalization Weighting scheme
print(color.BOLD+color.BLUE+"log normalization Weighting Scheme")

log normalization Weighting Scheme

[91] matrix_tf_idf_log_norm=np.zeros((fileData.shape[0],len(distinctiveTokens)))
value=1
for i in range(fileData.shape[0]):
    doc=meta_doc_dict[fileData.iloc[i,0]]
    tokens=list(doc.keys())
    for word in tokens:
        tf=math.log(doc[word]+1)
        idf=idf_values[word]
        overall_score=tf*idf
        matrix_tf_idf_log_norm[i,distinctiveTokens.index(word)]=overall_score

print(matrix_tf_idf_log_norm)

[[ 3.05229583  2.50635527  4.54086286 ... 0.          0.          0.          ],
 [ 0.          0.          0.          ... 0.          0.          0.          ],
 [ 0.          0.          0.          ... 0.          0.          0.          ],
 ...
 [ 0.          0.          0.          ... 0.          0.          0.          ],
 [ 4.97808009  1.58133411  0.          ... 0.          0.          0.          ],
 [ 0.          0.          0.          ... 0.          0.          0.          ]]

[92] # Double Normalization Weighting scheme
print(color.BOLD+color.BLUE+"Double Normalization Weighting Scheme")

Double Normalization Weighting Scheme

```

Activate Windows
Go to Settings to activate Windows.

```

len_df=filedata.shape[0]
query=EnterQuery()
final_ans={}
for x in range(len_df):
    score=0
    doc=matrix_tf_idf_log_norm[x,:]
    for token in query:
        try:
            score+=doc[distinctiveTokens.index(token)]
        except:
            pass
    final_ans[filedata.iloc[x,0]]=score

final_sorted_ans=dict(sorted(final_ans.items(),key=lambda item:item[1],reverse=True))
count=0
print(color.BOLD+color.DARKCYAN+"Using Log Normalization Weighting Scheme")
for k,v in final_sorted_ans.items():
    count+=1

    print(f'{count}: {k}      score: {v}')
    if(count>=5):
        break

```

Activate Windows
Go to Settings to activate Windows.

❖ **DOUBLE NORMALIZATION:** In the TF-IDF matrix, ROW represents "Number of Documents" and COLUMN represents "Number of Unique Words in the Word Corpus."
 $\text{TF value} = 0.5 + 0.5 * (\text{f}(t,d) / \max(\text{f}(t',d)))$
 $\text{TF-IDF score} = \text{tf} * \text{idf}$

Result:

```

matrix_tf_idf_double_norm=np.zeros((FileData.shape[0],len(distinctiveTokens)))
value=1
for i in range(FileData.shape[0]):
    doc=meta_doc_dict[FileData.iloc[i,0]]
    tokens=list(doc.keys())
    tf_max=0
    for w in tokens:
        temp=doc[w]
        if(tf_max<temp):
            tf_max=temp
    tf_idf=temp/tf_max
    for word in tokens:
        tf=0.5*(doc[word]/tf_max)+0.5
        idf=idf_values[word]
        overall_score=tf*idf
        matrix_tf_idf_double_norm[i,distinctiveTokens.index(word)]=overall_score

```

Activate Windows
Go to Settings to activate Windows.

```

len_df=fileData.shape[0]
query=EnterQuery()
final_ans={}
for x in range(len_df):
    score=0
    doc=matrix_tf_idf_double_norm[x,:]
    for token in query:
        try:
            score+=doc[distinctiveTokens.index(token)]
        except:
            pass
    final_ans[fileData.iloc[x,0]]=score

final_sorted_ans=dict(sorted(final_ans.items(),key=lambda item:item[1],reverse=True))
count=0
print(color.BOLD+color.DARKCYAN+"Using Double Normalization Weighting Scheme")
for k,v in final_sorted_ans.items():
    count+=1
    print(f'{count}: {k}      score: {v}')
    if(count>=5):
        break

```

Activate Windows
Go to Settings to activate Windows.

completed at 7:29PM

Pros and cons of using each scoring scheme:

1. JACCARD COEFFICIENT

Benefits:

The sets don't have to be the same size.

The Jaccard similarity rating usually ranges from 0 to 1.

Cons:

The term frequency is not taken into account when determining similarity.

We are unable to determine which terms are more and less frequent .

The length normalisation is not taken into account.

2. TF-IDF

Binary Weighting Scheme:

Pros:

Simple and easy to implement.

Efficient in terms of computation time.

Cons:

Ignores the frequency of the term in the document, which can lead to inaccurate results.

All terms are given equal weight, which can result in less relevant documents being ranked

Higher

.

Raw Count Weighting Scheme:

Pros:

Takes into account the frequency of the term in the document, which can improve the accuracy of the results.

Allows for different weighting of terms based on their frequency.

Cons:

Does not take into account the frequency of the term in the collection, which can lead to skewed results.

Can be biased towards longer documents, as they may have more occurrences of a given term.

Term Frequency (TF) Weighting Scheme:

Pros:

Takes into account the frequency of the term in the document, which can improve the accuracy of the results.

Allows for different weighting of terms based on their frequency.

Cons:

Does not take into account the frequency of the term in the collection, which can lead to skewed results.

Can be biased towards longer documents, as they may have more occurrences of a given term.

Log Normalization Weighting Scheme:

Pros:

Reduces the impact of very high term frequencies.

Helps to account for differences in document length.

Cons:

Can be sensitive to very low term frequencies.

May require tuning of the parameter to achieve optimal results.

Double Normalization Weighting Scheme:

Pros:

Helps to account for differences in document length and term frequency.

Can improve the accuracy of the results compared to other weighting schemes.

Cons:

Can be sensitive to very low term frequencies.

May require tuning of the parameters to achieve optimal results.

B. Jaccard Coefficient

- With the help of the pre-processing techniques mentioned above, we were able to obtain distinct tokens for each of the queries and the word corpus. Following that, we created sets for the word corpus and query tokens.
- Then, using the intersection and union operations of Python sets, we collected the union and intersection between the set of query tokens and the set of document tokens for each file.

The Jaccard score for a given document was determined after obtaining the union and intersection by dividing the length of the intersection between the query and document corpus by the length of the union between the two.

- The top ten documents with the highest Jaccard coefficient were then determined by ranking the results from the previous steps.

Result:

```
try:
    comparisons=0
    query=EnterQuery()
    query=set(query)
    top_10=[]
    for i in range(len(data.shape[0])):
        tokens=set(data.iloc[i,2])
        intersecquery.intersection(tokens)
        unionquery.union(tokens)
        jaccardCoeff=intersecquery/unionquery
        top_10.append(jaccardCoeff)
    answer=sorted(top_10.items(),reverse=True)
    answer.sorted(top_5.items(),reverse=True)
    print(color.BOLD+color.DARKCYAN+"The list of top-10 document names retrieved:\n")
    count=0
    for k,v in answer:
        if(count>10):
            break
        print(color.BOLD+color.BLUE+"(v): Jaccard Coefficients: (k)")
        count+=1
except:
    print(color.BOLD+color.RED+"Enter Valid Input!!!")

Enter your query numerical determination indiclifft
The list of top-10 document names retrieved:

cranfield0092: Jaccard Coefficient: 0.07406407407407407
cranfield0032: Jaccard Coefficient: 0.06666666666666667
cranfield0078: Jaccard Coefficient: 0.06263578947368425
cranfield0219: Jaccard Coefficient: 0.064675
cranfield0091: Jaccard Coefficient: 0.06166666666666666
cranfield0256: Jaccard Coefficient: 0.0635742857428571
cranfield0039: Jaccard Coefficient: 0.03333333333333333
cranfield0025: Jaccard Coefficient: 0.0378888524590164
```

[24] # Here we are creating a meta document dictionary, which is nested.
meta_doc_dict={}
here we are creating a list of unique tokens.
distinctiveTokens=[]
for i in range(len(data.shape[0])):
 token_list=data.iloc[i,2]
 temp_dict={}
 for j in token_list:
 if(j in temp_dict):
 temp_dict[j]=j+1
 else:
 temp_dict[j]=1
 distinctiveTokens.append(temp_dict)
meta_doc_dict[i]=temp_dict

Activate Windows
Go to Settings to activate Windows.

26°C Haze 7:38 PM 3/27/2023

Question 2

1. Preprocessing the dataset:

```
trainDataFrame= pd.read_csv("/content/drive/MyDrive/IR_ASSIGNMENT2/Q2/BBC News Train.csv")
trainDataFrame.head()

[226] class color:
    PURPLE = '\u033[95m'
    CYAN = '\u033[96m'
    DARKCYAN = '\u033[36m'
    BLUE = '\u033[94m'
    GREEN = '\u033[92m'
    YELLOW = '\u033[93m'
    RED = '\u033[91m'
    UNDERLINE = '\u033[4m'
    END = '\u033[0m'

[227] import itertools
colors = itertools.cycle(["r", "b", "g"])

[228] trainDataFrame.shape
(1490, 3)

[229] # removing unnecessary columns.
data = trainDataFrame.drop(columns=['ArticleId'])

[230] # here we are removing column named 'ArticleId'.
```

Activate Windows
Go to Settings to activate Windows.

23°C Haze 10:38 PM 3/27/2023

```

File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
(1496, 3)
[229] # removing unnecessary columns.
[230] # here we are removing column named 'ArticleId'.
data = trainDataFrame.drop(columns=['ArticleId'])

list_of_stopwords_and_punctuation = stopwords.words('english') + list(string.punctuation)

# here we are initializing the WordNetLemmatizer
lemmatizer = WordNetLemmatizer()

# here we are creating a function to preprocess the text
def preprocessing(text):
    # here we are lowercasing the text
    text = text.lower()

    # here we are tokenizing the text
    words = word_tokenize(text)

    # here we are Removing the stopwords and punctuation marks
    words = [word for word in words if word not in list_of_stopwords_and_punctuation]

    # here we are Lemmatizing the words
    words = [lemmatizer.lemmatize(word) for word in words]

    return words

# here are applying the preprocessing function
data['Text'] = data['Text'].apply(preprocessing)

[232] # for the TfidfVectorizer we need to convert the list of words to a string
data['Text'] = data['Text'].apply('.join')

# Initializing

```

2. the dataset:

```

# for the TfidfVectorizer we need to convert the list of words to a string
data['Text'] = data['Text'].apply('.join')

# Initializing
c_v = CountVectorizer()
tfidf_transformer = TfidfTransformer()

# here we are Fitting and transforming the vectorizer on the 'Text' column
term_doc_matrix = c_v.fit_transform(data['Text'])

# here we are Generating the TF-IDF matrix
tfidf_matrix = tfidf_transformer.fit_transform(term_doc_matrix).toarray()

# here we are evaluating the ICF values
num_docs = len(data)
icf_values = np.log(num_docs / np.count_nonzero(tfidf_matrix, axis=0))

# here we are converting tfidf matrix and icf values to sparse matrices
tfidf_matrix = csr_matrix(tfidf_matrix)
icf_values = csr_matrix(icf_values)

# here we are evaluating tf_icf matrix
tf_icf_matrix = tfidf_matrix.multiply(icf_values)

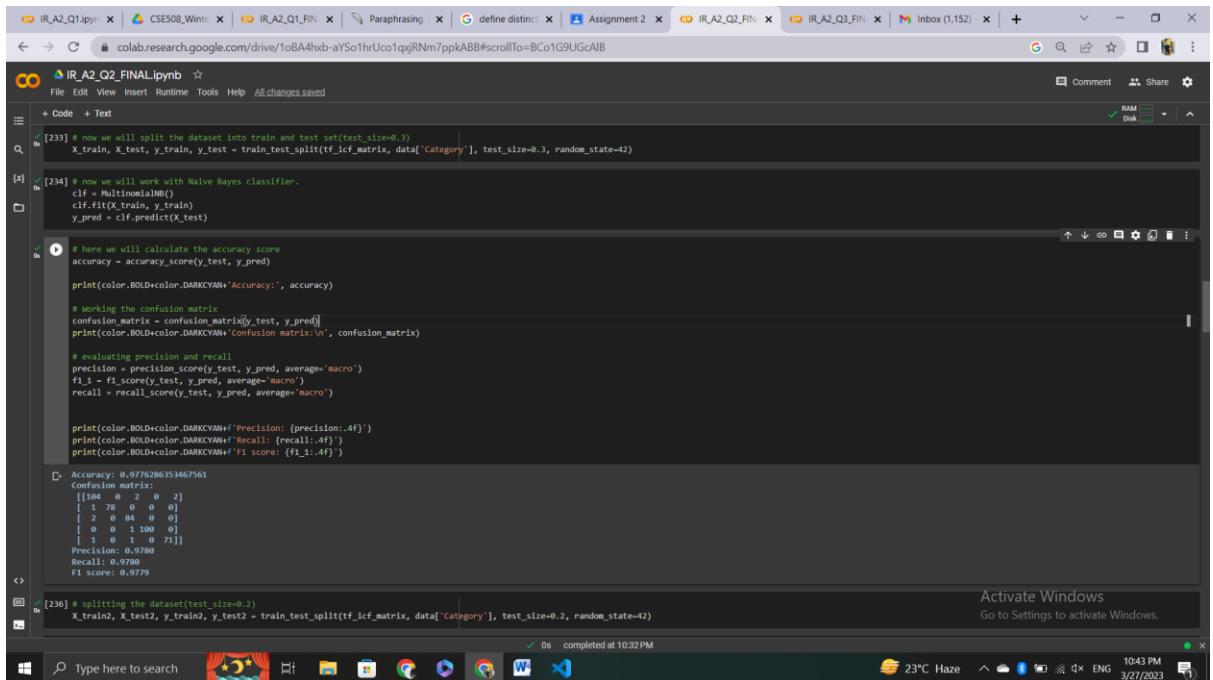
[233] # now we will split the dataset into train and test set(test_size=0.3)
X_train, X_test, y_train, y_test = train_test_split(tf_icf_matrix, data['Category'], test_size=0.3, random_state=42)

# now we will work with Naive Bayes classifier.
clf = MultinomialNB()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

[235] # here we will calculate the accuracy score

```

3. Training, Testing and improving the classifier:



```
[233] # now we will split the dataset into train and test set(test_size=0.3)
X_train, X_test, y_train, y_test = train_test_split(tf_icf_matrix, data['Category'], test_size=0.3, random_state=42)

[234] # now we will work with Naive Bayes classifier.
clf = MultinomialNB()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

# here we will calculate the accuracy score
accuracy = accuracy_score(y_test, y_pred)

print(color.BOLD+color.DARKCYAN+'Accuracy:', accuracy)

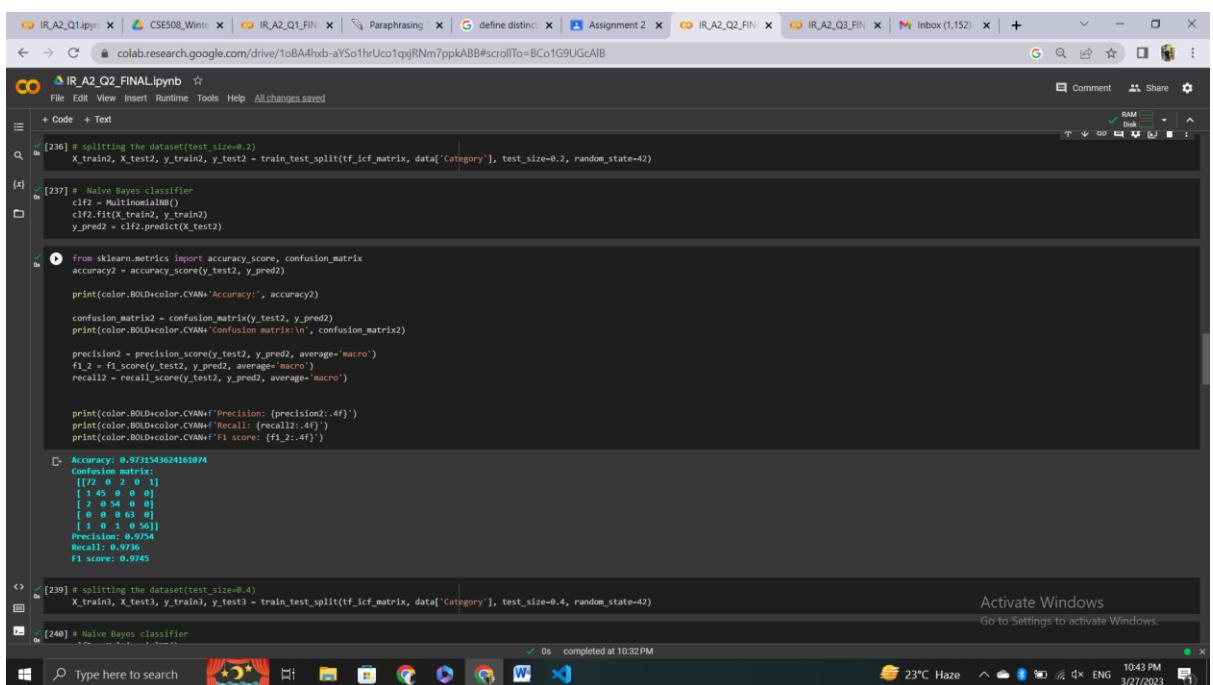
# working the confusion matrix
confusion_matrix = confusion_matrix(y_test, y_pred)
print(color.BOLD+color.DARKCYAN+'Confusion matrix:\n', confusion_matrix)

# evaluating precision and recall
precision = precision_score(y_test, y_pred, average='macro')
f1_1 = f1_score(y_test, y_pred, average='macro')
recall = recall_score(y_test, y_pred, average='macro')

print(color.BOLD+color.DARKCYAN+'Precision: (precision:.4f)')
print(color.BOLD+color.DARKCYAN+'Recall: (recall:.4f)')
print(color.BOLD+color.DARKCYAN+'F1 score: (f1_1:.4f)')

Accuracy: 0.9736286353467561
Confusion matrix:
[[104  0  2  0]
 [ 1 78  0  0]
 [ 2  0 84  0]
 [ 0  0  0 100]
 [ 1  0  1  71]]
Precision: 0.9780
Recall: 0.9780
F1 score: 0.9779

[236] # splitting the dataset(test_size=0.2)
X_train2, X_test2, y_train2, y_test2 = train_test_split(tf_icf_matrix, data['Category'], test_size=0.2, random_state=42)
```



```
[236] # splitting the dataset(test_size=0.2)
X_train2, X_test2, y_train2, y_test2 = train_test_split(tf_icf_matrix, data['Category'], test_size=0.2, random_state=42)

[237] # Naive Bayes classifier
clf2 = MultinomialNB()
clf2.fit(X_train2, y_train2)
y_pred2 = clf2.predict(X_test2)

from sklearn.metrics import accuracy_score, confusion_matrix
accuracy2 = accuracy_score(y_test2, y_pred2)

print(color.BOLD+color.CYAN+'Accuracy:', accuracy2)

confusion_matrix2 = confusion_matrix(y_test2, y_pred2)
print(color.BOLD+color.CYAN+'Confusion matrix:\n', confusion_matrix2)

precision2 = precision_score(y_test2, y_pred2, average='macro')
f1_2 = f1_score(y_test2, y_pred2, average='macro')
recall2 = recall_score(y_test2, y_pred2, average='macro')

print(color.BOLD+color.CYAN+'Precision: (precision2:.4f)')
print(color.BOLD+color.CYAN+'Recall: (recall2:.4f)')
print(color.BOLD+color.CYAN+'F1 score: (f1_2:.4f)')

Accuracy: 0.9731543624161074
Confusion matrix:
[[77  0  2  0]
 [ 1 45  0  0]
 [ 2  0 54  0]
 [ 0  0  0 63]
 [ 1  0  0 56]]
Precision: 0.9784
Recall: 0.9786
F1 score: 0.9745

[238] # splitting the dataset(test_size=0.4)
X_train3, X_test3, y_train3, y_test3 = train_test_split(tf_icf_matrix, data['Category'], test_size=0.4, random_state=42)

[240] # Naive Bayes classifier
```

```

IR_A2_Q1.ipynb x CSE508_Wint... x IR_A2_Q1.FIN x Paraphrasing x define distinct... x Assignment 2 x IR_A2_Q2.FIN x IR_A2_Q3.FIN x Inbox (1,152) x + - _ X
← → ⌂ colab.research.google.com/drive/1oBA4hxh-aYSo1hrUco1qxjRNm7ppkABB#scrollTo=BCo1G9UGcAIB
File Edit View Insert Runtime Tools Help All changes saved
Comment Share RAM Disk
+ Code + Text
[239] # splitting the dataset(test_size=0.4)
X_train3, X_test3, y_train3, y_test3 = train_test_split(tf_lcf_matrix, data['Category'], test_size=0.4, random_state=42)
[240] # Naive Bayes classifier
clf3 = MultinomialNB()
clf3.fit(X_train3, y_train3)
y_pred3 = clf3.predict(X_test3)

accuracy3 = accuracy_score(y_test3, y_pred3)
print(color.BOLD+color.GREEN+'Accuracy:', accuracy3)

confusion_matrix3 = confusion_matrix(y_test3, y_pred3)
print(color.BOLD+color.GREEN+'Confusion matrix:\n', confusion_matrix3)

precision3 = precision_score(y_test3, y_pred3, average='macro')
f1_3 = f1_score(y_test3, y_pred3, average='macro')
recall3 = recall_score(y_test3, y_pred3, average='macro')

print(color.BOLD+color.GREEN+f'Precision: {precision3:.4f}')
print(color.BOLD+color.GREEN+f'Recall: {recall3:.4f}')
print(color.BOLD+color.GREEN+f'F1 score: {f1_3:.4f}')

Activation: 0.8748322474651006
Confusion matrix:
[[150  0  0  0]
 [ 1 300  1  0]
 [ 2  0 107  0]
 [ 0  1  120  0]
 [ 1  0  0  200]]
Precision: 0.9747
Recall: 0.9748
F1 score: 0.9748

[242] # here we are calculating the frequency of each category in the training set
categoryProbability = y_train.value_counts(normalize=True)
print(color.BOLD+color.BLUE+'Category probability:\n', categoryProbability)

```

Activate Windows
Go to Settings to activate Windows.

```

IR_A2_Q1.ipynb x CSE508_Wint... x IR_A2_Q1.FIN x Paraphrasing x define distinct... x Assignment 2 x IR_A2_Q2.FIN x IR_A2_Q3.FIN x Inbox (1,152) x + - _ X
← → ⌂ colab.research.google.com/drive/1oBA4hxh-aYSo1hrUco1qxjRNm7ppkABB#scrollTo=BCo1G9UGcAIB
File Edit View Insert Runtime Tools Help All changes saved
Comment Share RAM Disk
+ Code + Text
[242] # here we are calculating the frequency of each category in the training set
categoryProbability = y_train.value_counts(normalize=True)
print(color.BOLD+color.BLUE+'Category probability:\n', categoryProbability)

Category probability:
sport      0.214889
Business   0.142857
entertainment 0.188083
Tech       0.188083
politics   0.188083
Name: Category, dtype: float64

[243] data['Text']
0    worldcom ex-boss launch defence lawyer defend...
1    german business confidence slide german busine...
2    bbc poll indicates economic gloom citizen major...
3    lifestyle governors mobile choice faster better ...
4    enron boss 168 payout eighteen former enron d...
1485  double eviction big brother model caprice hol...
1486  dj double act revamp chart show dj duo jk joel...
1487  weak dollar hit reuters revenue median group r...
1488  apple ipod family expands market apple expand...
1489  sony sony sony sony sony sony sony sony sony s...
Name: Text, length: 1400, dtype: object

[244] y_train
701    entertainment
1142      tech
490      tech
10      politics
147      business
...
138      politics
1295     business
868      politics
1459     entertainment
1126      sport
Name: Category, length: 1043, dtype: object

```

Activate Windows
Go to Settings to activate Windows.

OR_A2_Q1.ipynb | CSE508_Winte | IR_A2_Q1_FIN | Paraphrasing | define distinct | Assignment 2 | IR_A2_Q2_FIN | IR_A2_Q3_FIN | Inbox (1,152) + - _

colab.research.google.com/drive/1oBA4hxbaYSo1hrUco1qjRNm?usp=scrollTo=FrCewGaBcg3

```
[x] In [246]: data2 = traindataframe.drop(columns=['ArticleID'])

[x] In [247]: data2['Text'] = data2['Text'].apply(preprocessing)

[ ] In [248]: ngram_range = (1,2)

[ ] In [249]: data2['Text'] = data2['Text'].apply(lambda x: ' '.join(
    vectorizer1.fit_transform(ngram_range=ngram_range).toarray().sum(1).nonzero()[0].tolist()))
    vectorizer2 = TfidfVectorizer(ngram_range=ngram_range)
    tfidf_matrix2 = vectorizer2.fit_transform(data2['Text'])

    # Here we are converting sparse matrix into dense matrix
    tfidf_matrix2 = tfidf_matrix2.todense()

    # Here we are creating a data frame from a dense matrix
    tfidf_dfs2 = pd.DataFrame(tfidf_matrix2, columns=vectorizer2.get_feature_names_out())

    # Here we are adding a column named Category to the data frame
    tfidf_dfs2['Category'] = data2['Category']

    # Here we are evaluating the mean of each feature
    tfidf_dfs2 = tfidf_dfs2.groupby('Category').mean()

[x] In [250]: # Splitting the dataset(test_size=0.3)
    X_train4, X_test4, y_train4, y_test4 = train_test_split(tfidf_dfs2.drop(['Category'], axis=1), tfidf_dfs2['Category'], test_size=0.3, random_state=42)

[x] In [251]: # naive Bayes classifier
    cfb = MultinomialNB()
    cfb.fit(X_train4, y_train4)
    y_pred4 = cfb.predict(X_test4)

[x] In [252]: accuracy4 = accuracy_score(y_test4, y_pred4)
    print(color.BOLD+color.PURPLE+'Accuracy:', accuracy4)

    confusion_matrix4 = confusion_matrix(y_test4, y_pred4)
    print(color.BOLD+color.PURPLE+'Confusion matrix:\n', confusion_matrix4)

    precision4 = precision_score(y_test4, y_pred4, average='macro')
    f1_4 = f1_score(y_test4, y_pred4, average='macro')
    recall4 = recall_score(y_test4, y_pred4, average='macro')

    print(color.BOLD+color.PURPLE+'Precision: ',precision4)
    print(color.BOLD+color.PURPLE+'Recall: ',recall4)
    print(color.BOLD+color.PURPLE+'F1 score: ',f1_4)

    Accuracy: 0.90304469755867
    Confusion matrix:
[[4848  1  1  1]
 [ 2 72  1  1]
 [ 1  1  1  1]
 [ 1  0  0  101]
 [ 1  0  0  703]
 Precision: 0.9036
Recall: 0.9046
F1 score: 0.9046
```

Activate Windows
Go to Settings to activate Windows.

4. Conclusion:

- Write a brief report summarizing your findings.

the code is performing text classification on the BBC News dataset using Naive Bayes classifier.

First, the dataset is read using Pandas and the 'ArticleId' column is removed. Then, the NLTK library is used to preprocess the text data by converting it to lowercase,

tokenizing it, removing stop words and punctuation marks, and lemmatizing the words.

After preprocessing the data, the TF-IDF vectorizer is used to generate a term-document matrix and the TF-IDF matrix is computed from it. ICF (Inverse Category Frequency) values are then computed from the TF-IDF matrix. The TF-ICF matrix is computed from the element-wise multiplication of the TF-IDF matrix and the ICF matrix.

The dataset is split into training and testing sets using the `train_test_split` function from scikit-learn. The Naive Bayes classifier is then initialized and fit on the training data. The model is evaluated on the testing data by predicting the categories of the articles and calculating various evaluation metrics such as accuracy, precision, recall, and F1 score.

The code is also performing the same text classification task on the dataset for two other test sizes (20% and 40%) and calculating the evaluation metrics for each of them separately.

- **Discuss the performance of the classifier and the impact of different preprocessing techniques, features, and weighting schemes on the results.**

The performance of a classifier can be evaluated using various metrics such as accuracy, precision, recall, F1-score, and area under the curve (AUC) of the receiver operating characteristic (ROC) curve. The impact of different preprocessing techniques, features, and weighting schemes can also have a significant impact on the performance of a classifier.

Preprocessing techniques such as data cleaning, normalization, and feature selection can help to improve the performance of a classifier. For example, removing irrelevant features and handling missing values can reduce noise and improve the accuracy of the model. Similarly, normalization can help to standardize the range of values for each feature, which can prevent some features from dominating others and improve the stability of the model.

Feature selection is another important preprocessing technique that can help to reduce the dimensionality of the data and improve the accuracy of the model. It involves selecting a subset of the most relevant features for the classification task. Different feature selection algorithms can be used, such as the chi-squared test, mutual information, and correlation-based feature selection.

Weighting schemes are also important in improving the performance of a classifier. For example, using inverse document frequency (IDF) weighting can help to reduce the impact of frequent terms in text classification tasks. Similarly, using term frequency-inverse document frequency (TF-IDF) weighting can improve the

performance of text classification models by giving more weight to terms that are specific to a particular document or class.

Overall, the performance of a classifier depends on various factors such as the quality of the data, the choice of preprocessing techniques, the selection of features, and the weighting scheme used. Therefore, it is important to experiment with different combinations of these factors to determine the best approach for a particular classification task.

Question 3

Part 1:

Using pandas, load the dataset, and extract all of the rows having the qid:4 value.

(x)		Dataframe																				
		0	1	2	3	4	5	6	7	8	9	...	128	129	130	131	132	133	134	135	136	137
0	0	qjd4	1.3	2.0	3.2	4.0	5.3	6.1	7.0	8:0.6666667	...	127:27	128:2	129:9	130:124	131:4678	132:54	133:74	134:0	135:0	136:0	
1	0	qjd4	1.3	2.0	3.3	4.0	5.3	6.1	7.0	8:1	...	127:61	128:0	129:8	130:122	131:508	132:131	133:136	134:0	135:0	136:0	
2	0	qjd4	1.3	2.0	3.2	4.0	5.3	6.1	7.0	8:0.6666667	...	127:31	128:2	129:8	130:115	131:508	132:51	133:70	134:0	135:0	136:0	
3	0	qjd4	1.3	2.0	3.3	4.0	5.3	6.1	7.0	8:1	...	127:32	128:82	129:17	130:122	131:508	132:83	133:107	134:0	135:10	136:13.35	
4	1	qjd4	1.3	2.0	3.3	4.0	5.3	6.1	7.0	8:1	...	127:29	128:11	129:8	130:121	131:508	132:103	133:120	134:0	135:0	136:0	
...		
98	0	qjd4	1.3	2.0	3.2	4.0	5.3	6.1	7.0	8:0.6666667	...	127:62	128:35	129:1	130:153	131:4872	132:9	133:55	134:0	135:0	136:0	
99	1	qjd4	1.3	2.0	3.3	4.2	5.3	6.1	7.0	8:1	...	127:52	128:367	129:6	130:153	131:2383	132:18	133:99	134:0	135:16	136:11.31666666666667	
100	2	qjd4	1.2	2.0	3.2	4.0	5.2	6.0	6.6666667	7.0	8:0.6666667	...	127:28	128:0	129:0	130:49182	131:26966	132:15	133:69	134:0	135:193	136:21.9355595468361
101	1	qjd4	1.2	2.0	3.2	4.0	5.2	6.0	6.6666667	7.0	8:0.6666667	...	127:23	128:0	129:1	130:42877	131:26562	132:12	133:24	134:0	135:56	136:62.920604232688
102	0	qjd4	1.3	2.0	3.2	4.0	5.3	6.1	7.0	8:0.6666667	...	127:59	128:1415	129:14	130:5334	131:6434	132:4	133:17	134:0	135:0	136:0	

Part 2:

The relevance of all the terms is sorted before the DCG value is calculated to determine the maximum DCG. DCG value is calculated using the formula:

	Sorted Dataframe													...	128	129	130	131	132	133	134	135	136	137	...
	0	1	2	3	4	5	6	7	8	9	...	128	129	130	131	132	133	134	135	136	137	...			
0	3	qjd4	1.3	2.0	3.2	4.1	5.3	6.1	7.0	8.0:6666667	...	127:32	128:349	129:8	130:123	131:281	132:22	133:6	134:0	135:0	136:0	137:0			
1	2	qjd4	1.2	2.0	3.1	4.0	5.2	6.0:6666667	7.0	8.0:333333	...	127:19	128:0	129:0	130:2417	131:721	132:14	133:113	134:0	135:13	136:47.9	137:0			
2	2	qjd4	1.3	2.2	3.2	4.0	5.3	6.1	7.0:6666667	8.0:6666667	...	127:33	128:8	129:3	130:1888	131:9338	132:3	133:11	134:0	135:0	136:0	137:0			
3	2	qjd4	1.3	2.0	3.2	4.0	5.3	6.1	7.0	8.0:6666667	...	127:17	128:0	129:2	130:12028	131:11379	132:26	133:24	134:0	135:77	136:23.9595223404047	137:0			
4	2	qjd4	1.3	2.0	3.3	4.3	5.3	6.1	7.0	8:1	...	127:67	128:27	129:0	130:814	131:13555	132:108	133:113	134:0	135:0	136:0	137:0			
...		
98	0	qjd4	1.2	2.0	3.0	4.0	5.2	6.0:6666667	7.0	8:0	...	127:41	128:8	129:0	130:868	131:9260	132:246	133:88	134:0	135:0	136:0	137:0			
99	0	qjd4	1.2	2.0	3.0	4.0	5.2	6.0:6666667	7.0	8:0	...	127:38	128:4	129:0	130:797	131:9260	132:237	133:80	134:0	135:0	136:0	137:0			
100	0	qjd4	1.3	2.0	3:3	4:1	5:3	6:1	7:0	8:1	...	127:65	128:83	129:5	130:144	131:262	132:157	133:179	134:0	135:0	136:0	137:0			
101	0	qjd4	1.3	2.1	3:3	4:2	5:3	6:1	7:0:333333	8:1	...	127:65	128:195	129:8	130:124	131:206	132:103	133:121	134:0	135:0	136:0	137:0			
102	0	qjd4	1.3	2.0	3:2	4:0	5:3	6:1	7:0	8:0:6666667	...	127:59	128:1415	129:14	130:5334	131:6434	132:4	133:17	134:0	135:0	136:0	137:0			

To calculate the number of files that could be made, we calculate the factorial of each relevant class and then multiply them to get the total number of ways.

Part 3:

nDCG is calculated using the formula : $nDCG = DCG / \text{Max } DCG$

For first 50 dataset, we calculate nDCG for first 50 rows of dataset.

```
Maximum Discounted Cumulative Gain (nDCG) for whole dataset is: 0.5979226516897831
Discounted Cumulative Gain (DCG) for first 50 rows is: 0.5253808413557646
```

Part 4:

Based on the 75th feature, we rank the URL. The URL is more pertinent the higher the value of this feature. After setting all nonnegative relevance to 1, we compute precision@k and recall@k while iterating through the dataset.

