Team Members

Name	SRN
Naren Chandrashekhar	PES2UG20CS216
Nitin Jayachandran	PES2UG20CS231
Rohan Amin	PES2UG20CS416
Anurag B	PES2UG20CS404

Section 1

- Corpus Details: The BBC News Dataset contains 2225 news articles from the BBC, spanning 5 different categories: business, entertainment, politics, sport, and tech. Each article is labeled with its corresponding category.

The dataset is stored in a CSV file, with each row representing a single article and the following columns:

category: the category label for the article (one of "business", "entertainment", "politics", "sport", or "tech").

text: the full text of the article.

- Source: https://www.kaggle.com/datasets/sahilkirpekar/bbcnews-dataset?resource=download (https://www.kaggle.com/datasets/sahilkirpekar/bbcnews-dataset?resource=download)

Libraries Used

In [10]:
```
import pandas as pd
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize
nltk.download('stopwords')
nltk.download('punkt')
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

from collections import defaultdict
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Knowing the Dataset

In [11]:
```
df = pd.read_csv("/content/BBCNews.csv")
print(df.head())
```

```
   Unnamed: 0                                              descr  \
0           0  chelsea sack mutu  chelsea have sacked adrian ...
1           1  record fails to lift lacklustre meet  yelena i...
2           2  edu describes tunnel fracas  arsenals edu has ...
3           3  ogara revels in ireland victory  ireland flyha...
4           4  unclear future for striker baros  liverpool fo...

                                                tags
0  sports, stamford bridge, football association,...
1  sports, madrid, birmingham, france, scotland, ...
2  sports, derby, brazil, tunnel fracasedu, food,...
3  sports, bbc, united kingdom, ireland, brian o'...
4  sports, liverpool, daily sport, millennium sta...
```

Preprocessing Of Data

In [12]:
```
#Lowercase all the words in the dataset
df['descr'] = df['descr'].str.lower()

# display the updated dataset
print(df.head())
```

```
   Unnamed: 0                                              descr  \
0           0  chelsea sack mutu  chelsea have sacked adrian ...
1           1  record fails to lift lacklustre meet  yelena i...
2           2  edu describes tunnel fracas  arsenals edu has ...
3           3  ogara revels in ireland victory  ireland flyha...
4           4  unclear future for striker baros  liverpool fo...

                                                tags
0  sports, stamford bridge, football association,...
1  sports, madrid, birmingham, france, scotland, ...
2  sports, derby, brazil, tunnel fracasedu, food,...
3  sports, bbc, united kingdom, ireland, brian o'...
4  sports, liverpool, daily sport, millennium sta...
```

In [13]:
```
#Remove stop words
# create a set of stop words
stop_words = set(stopwords.words('english'))

# remove stop words from the 'descr' column
df['descr'] = df['descr'].apply(lambda x: ' '.join([word for word in x.split()

# display the updated dataset
print(df.head())
```

```
   Unnamed: 0                                              descr  \
0           0  chelsea sack mutu chelsea sacked adrian mutu f...
1           1  record fails lift lacklustre meet yelena isinb...
2           2  edu describes tunnel fracas arsenals edu lifte...
3           3  ogara revels ireland victory ireland flyhalf r...
4           4  unclear future striker baros liverpool forward...

                                                tags
0  sports, stamford bridge, football association,...
1  sports, madrid, birmingham, france, scotland, ...
2  sports, derby, brazil, tunnel fracasedu, food,...
3  sports, bbc, united kingdom, ireland, brian o'...
4  sports, liverpool, daily sport, millennium sta...
```

In [14]:
```
# Initialize the stemmer
stemmer = PorterStemmer()

# Define a function to stem the text
def stem_text(text):
    # Tokenize the text
    tokens = word_tokenize(text)
    # Stem each token and add to a new list
    stemmed_tokens = [stemmer.stem(token) for token in tokens]
    # Join the tokens back into a string
    text = ' '.join(stemmed_tokens)
    return text

# Apply the stem function to the 'clean_text' column of the dataframe
df['descr'] = df['descr'].apply(stem_text)
print(df.head())
```

```
   Unnamed: 0                                              descr  \
0           0  chelsea sack mutu chelsea sack adrian mutu fai...
1           1  record fail lift lacklustr meet yelena isinbay...
2           2  edu describ tunnel fraca arsen edu lift lid sc...
3           3  ogara revel ireland victori ireland flyhalf ro...
4           4  unclear futur striker baro liverpool forward m...

                                                tags
0  sports, stamford bridge, football association,...
1  sports, madrid, birmingham, france, scotland, ...
2  sports, derby, brazil, tunnel fracasedu, food,...
3  sports, bbc, united kingdom, ireland, brian o'...
4  sports, liverpool, daily sport, millennium sta...
```

In [15]:
```
# Define a function to remove special characters
def remove_special_chars(text):
    # Define the pattern to match special characters
    pattern = r'[^a-zA-Z0-9\s]'
    # Replace special characters with a space
    text = re.sub(pattern, ' ', text)
    return text

# Apply the function to the 'clean_text' column of the dataframe
df['descr'] = df['descr'].apply(remove_special_chars)
print(df.head())
```

```
   Unnamed: 0                                              descr  \
0           0  chelsea sack mutu chelsea sack adrian mutu fai...
1           1  record fail lift lacklustr meet yelena isinbay...
2           2  edu describ tunnel fraca arsen edu lift lid sc...
3           3  ogara revel ireland victori ireland flyhalf ro...
4           4  unclear futur striker baro liverpool forward m...

                                                tags
0  sports, stamford bridge, football association,...
1  sports, madrid, birmingham, france, scotland, ...
2  sports, derby, brazil, tunnel fracasedu, food,...
3  sports, bbc, united kingdom, ireland, brian o'...
4  sports, liverpool, daily sport, millennium sta...
```

Section 2

Set of

- Free Text Test Queries
- Wild card Queries
- Phase Queries

Free Text Test Queries

- For Boolean Retrieval: Results shown in Section 4.1

1. query: white house terrorist
2. query: federer championship
3. query: ronaldo manchester united
4. query: tax law fraud
5. query: windows microsoft
6. query: economy us bank loan

- For Inverted Index: Results shown in Section 4.2

1. query: UK election
2. query: white house
3. query: chelsea
4. query: robot

5. query: hollywood
6. query: apple
7. query: blockchain
8. query: pear

Wild Card Queries

Results shown in Section 4.3

1. query: robo*
2. query: *tech
3. query: cyber*attack
4. query: mark*crash
5. query: *energy*

Phrase Queries

Results shown in Section 4.4

1. query: climate change
2. query: gender pay gap
3. query: global economic
4. query: foood waste reduction initiatives
5. query: human rights
6. query: economic growth

Section 3

Data structures used with brief reasons and similarity scheme

Data Structures Used

1. Inverted Index: An inverted index is a data structure that stores a mapping between terms and the documents that contain them. Inverted index is efficient for keyword-based search because it can quickly locate documents containing a particular term or a combination of terms. It is widely used in search engines to efficiently retrieve relevant documents based on user queries.
2. Sets: A set is an unordered collection of unique items, which can be used to store and manipulate sets of terms that occur in a document or query. In particular, sets can be used to represent the set of terms that appear in a document or query, or to store the set of documents that contain a particular term. One common use of sets in information retrieval is for document representation using the bag-of-words model. In this model, a document is represented as a set of terms, where the frequency of each term in the document is not considered. The set of terms that appear in a document can be efficiently computed using a set data structure. Another use of sets in information retrieval is for computing set operations such as union, intersection, and difference.
3. Dictionaries: A dictionary is a collection of key-value pairs, where each key is unique and maps to a corresponding value. In information retrieval, dictionaries can be used to represent mappings between terms and their corresponding document frequency or term frequency in a collection of documents. One common use of dictionaries in information

retrieval is for building an inverted index. In this index, each term in the vocabulary is mapped to a list of document IDs that contain that term. The inverted index can be implemented using a dictionary, where each key is a term and each value is a list of document IDs. The dictionary can be efficiently updated as new documents are added to the collection, and can be used to quickly retrieve the set of documents that contain a given term.

4. Lists: A list is an ordered collection of items, which can be used to store and manipulate sequences of terms or documents in a particular order. In particular, lists can be used to represent sequences of documents that match a query, or to store sequences of terms in a document. One common use of lists in information retrieval is for document ranking using the vector space model. In this model, each document is represented as a vector of weights, where each weight corresponds to the importance of a particular term in the document. To rank documents based on their similarity to a query, the model computes the cosine similarity between the query vector and each document vector, and returns a ranked list of documents. The ranked list of documents can be represented as a list of document IDs in order of decreasing similarity score.

Similarity Scheme

Similarity schemes are methods used to measure the similarity between a query and a document in information retrieval. The goal of a similarity scheme is to quantify how relevant a document is to a query based on the similarity between their respective representations.

1. Cosine similarity: This is a commonly used similarity scheme in vector space models. Cosine similarity measures the cosine of the angle between the query vector and the document vector, and is given by the dot product of the two vectors divided by their magnitudes. Cosine similarity is widely used in text retrieval because it is efficient and easy to implement.
2. BM25: This is a similarity scheme that is commonly used in ranking algorithms for text retrieval. BM25 measures the relevance of a document to a query based on the frequency of the query terms in the document and the frequency of the query terms in the entire collection of documents. It is designed to be robust to differences in document lengths and to handle rare terms.
3. TF-IDF (Term Frequency-Inverse Document Frequency) is a weighting scheme commonly used in information retrieval to rank documents based on their relevance to a query. While TF-IDF is not a similarity scheme in and of itself, it is often used as a component of similarity schemes such as the vector space model. The basic idea behind TF-IDF is to weigh the importance of a term in a document based on how frequently it appears in the document (term frequency) and how rarely it appears in the collection of all documents (inverse document frequency). This is done by multiplying the term frequency by the inverse document frequency, resulting in a weight that is high if the term appears frequently in the document and rarely in the collection, and low if the term appears rarely in the document or frequently in the collection.

Section 4.1

Result of Boolean Retrieval: On free text queries

Below are some Boolean Retrieved documents which specify the document id where the queries have found a match. The documents contain each of the given words in the queries. We have provided some useful queries for which the required documents have been specified

In [16]:
```
df = pd.read_csv("/content/BBCNews.csv")
# define the boolean retrieval function
def boolean_search(query, df):
    # split the query into individual terms
    query_terms = query.lower().split()

    # initialize an empty set of documents
    result = set()

    # loop over the documents in the dataframe
    for i in range(len(df)):
        # get the text of the current document
        text = df.iloc[i]["descr"].lower()

        # check if the document contains all of the query terms
        contains_all_terms = all(term in text for term in query_terms)

        # if the document contains all of the query terms, add it to the resul
        if contains_all_terms:
            result.add(i)

    return result

# test the boolean retrieval function
query = "white house terrorist"
results = boolean_search(query, df)
print(results)
```

```
{1186}
```

In [17]:

```
df = pd.read_csv("/content/BBCNews.csv")
# define the boolean retrieval function
def boolean_search(query, df):
    # split the query into individual terms
    query_terms = query.lower().split()

    # initialize an empty set of documents
    result = set()

    # loop over the documents in the dataframe
    for i in range(len(df)):
        # get the text of the current document
        text = df.iloc[i]["descr"].lower()

        # check if the document contains all of the query terms
        contains_all_terms = all(term in text for term in query_terms)

        # if the document contains all of the query terms, add it to the resul
        if contains_all_terms:
            result.add(i)

    return result

# test the boolean retrieval function
query = "federer championship"
results = boolean_search(query, df)
print(results)
```

```
{339, 157, 31}
```

In [18]:

```
df = pd.read_csv("/content/BBCNews.csv")
# define the boolean retrieval function
def boolean_search(query, df):
    # split the query into individual terms
    query_terms = query.lower().split()

    # initialize an empty set of documents
    result = set()

    # loop over the documents in the dataframe
    for i in range(len(df)):
        # get the text of the current document
        text = df.iloc[i]["descr"].lower()

        # check if the document contains all of the query terms
        contains_all_terms = all(term in text for term in query_terms)

        # if the document contains all of the query terms, add it to the resul
        if contains_all_terms:
            result.add(i)

    return result

# test the boolean retrieval function
query = "ronaldo manchester united"
results = boolean_search(query, df)
print(results)
```

```
{192, 5, 7, 1551, 1491, 116}
```

In [19]:

```
df = pd.read_csv("/content/BBCNews.csv")
# define the boolean retrieval function
def boolean_search(query, df):
    # split the query into individual terms
    query_terms = query.lower().split()

    # initialize an empty set of documents
    result = set()

    # loop over the documents in the dataframe
    for i in range(len(df)):
        # get the text of the current document
        text = df.iloc[i]["descr"].lower()

        # check if the document contains all of the query terms
        contains_all_terms = all(term in text for term in query_terms)

        # if the document contains all of the query terms, add it to the resul
        if contains_all_terms:
            result.add(i)

    return result

# test the boolean retrieval function
query = "tax law fraud"
results = boolean_search(query, df)
print(results)
```

```
{1729, 1763, 1891, 1915, 2276, 2373, 1737, 1900, 2235, 1679, 2389, 1656, 117
7, 1659, 1630}
```

In [20]:

```
df = pd.read_csv("/content/BBCNews.csv")
# define the boolean retrieval function
def boolean_search(query, df):
    # split the query into individual terms
    query_terms = query.lower().split()

    # initialize an empty set of documents
    result = set()

    # loop over the documents in the dataframe
    for i in range(len(df)):
        # get the text of the current document
        text = df.iloc[i]["descr"].lower()

        # check if the document contains all of the query terms
        contains_all_terms = all(term in text for term in query_terms)

        # if the document contains all of the query terms, add it to the resul
        if contains_all_terms:
            result.add(i)

    return result

# test the boolean retrieval function
query = "windows microsoft"
results = boolean_search(query, df)
print(results)
```

```
{1272, 1282, 1413, 774, 778, 1290, 1418, 1293, 1294, 408, 413, 544, 800, 803,
676, 677, 678, 806, 810, 434, 690, 567, 440, 573, 701, 449, 581, 454, 712, 45
9, 464, 1362, 595, 1237, 472, 475, 476, 604, 1252, 1385, 1258, 620, 621, 139
1, 1267, 756, 1399, 760, 1277}
```

In [21]:
```
df = pd.read_csv("/content/BBCNews.csv")
# define the boolean retrieval function
def boolean_search(query, df):
    # split the query into individual terms
    query_terms = query.lower().split()

    # initialize an empty set of documents
    result = set()

    # loop over the documents in the dataframe
    for i in range(len(df)):
        # get the text of the current document
        text = df.iloc[i]["descr"].lower()

        # check if the document contains all of the query terms
        contains_all_terms = all(term in text for term in query_terms)

        # if the document contains all of the query terms, add it to the resul
        if contains_all_terms:
            result.add(i)

    return result

# test the boolean retrieval function
query = "economy us loan bank"
results = boolean_search(query, df)
print(results)
```

```
{2249, 1771, 1709, 2349, 1746, 1714, 2228}
```

Section 4.2

Result with inverted index: On free text queries with rank.

The first celll block is a basic code to give the inverted index for each word in the document. The following cell blocks give the result of a free text query with the ranking.

Explaination as to how the query works, various examples with explaination has been provided after each cell output.

Ranking here specifies the most relevant document retrieved with respect to the query given. The output of the document ranking is in descending order.

In [22]:
```python
# Define a regular expression to tokenize the text
tokenizer = re.compile(r"\w+")

# Create an empty dictionary to store the inverted index
inverted_index = defaultdict(list)

# Loop over each document in the dataset
for i, row in df.iterrows():
    # Tokenize the text
    tokens = tokenizer.findall(row['descr'])
    # Count the frequency of each token
    term_freq = defaultdict(int)
    for token in tokens:
        term_freq[token] += 1
    # Add the document to the inverted index for each unique token
    for token, freq in term_freq.items():
        inverted_index[token].append((i, freq))

# Print the inverted index
for token, postings in inverted_index.items():
    print(f"{token}: {postings}")
```

```
chelsea: [(0, 8), (4, 1), (12, 8), (24, 3), (25, 1), (42, 2), (47, 2), (10
8, 1), (110, 5), (122, 1), (124, 1), (126, 4), (129, 1), (137, 2), (139,
8), (141, 3), (144, 2), (146, 2), (151, 1), (154, 3), (155, 8), (163, 4),
(166, 1), (192, 1), (197, 5), (346, 3), (358, 7), (361, 4), (362, 2), (36
5, 3), (383, 3), (389, 3), (392, 1), (395, 1), (396, 1), (453, 1), (631,
2), (632, 3), (635, 1), (637, 1), (638, 2), (643, 4), (652, 2), (662, 1),
(922, 1), (1187, 2), (1271, 1), (1451, 3), (1456, 1), (1462, 1), (1483,
3), (1485, 4), (1486, 5), (1493, 7), (1501, 2), (1509, 3), (1523, 2), (152
4, 3), (1534, 1), (1535, 1), (1540, 2), (1550, 1), (1558, 3), (1565, 7),
(1573, 3), (1581, 3), (1598, 4), (1604, 1), (2308, 3)]
sack: [(0, 2), (147, 1), (206, 1), (207, 1), (298, 1), (309, 1), (954, 1),
(1222, 1), (1427, 1), (1598, 2), (1605, 1)]
mutu: [(0, 6), (652, 6), (1598, 6)]
have: [(0, 5), (1, 10), (2, 1), (3, 5), (5, 1), (6, 1), (7, 1), (8, 1), (1
1, 2), (12, 3), (15, 1), (16, 3), (19, 1), (21, 3), (22, 4), (23, 2), (25,
2), (26, 7), (27, 6), (28, 1), (29, 3), (30, 6), (31, 1), (32, 3), (33,
3), (34, 6), (36, 1), (37, 1), (38, 1), (39, 1), (40, 5), (42, 1), (44,
1), (47, 8), (48, 1), (49, 3), (50, 2), (51, 1), (52, 1), (53, 3), (56,
4), (57, 2), (58, 1), (59, 3), (61, 2), (63, 2), (66, 2), (67, 3), (68,
```

In [23]:

```
import pandas as pd
import re
from collections import defaultdict

# Define a regular expression to tokenize the text
tokenizer = re.compile(r"\w+")

# Create an empty dictionary to store the inverted index
inverted_index = defaultdict(list)

# Loop over each document in the dataset
for i, row in df.iterrows():
    # Tokenize the text
    tokens = tokenizer.findall(row['descr'])
    # Count the frequency of each token
    term_freq = defaultdict(int)
    for token in tokens:
        term_freq[token] += 1
    # Add the document to the inverted index for each unique token
    for token, freq in term_freq.items():
        inverted_index[token].append((i, freq))

# Define a function to score documents based on a query
def score_documents(query, inverted_index, df):
    # Tokenize the query
    tokens = tokenizer.findall(query)
    # Count the frequency of each token
    query_freq = defaultdict(int)
    for token in tokens:
        query_freq[token] += 1
    # Compute the score for each document
    scores = defaultdict(int)
    for token, freq in query_freq.items():
        if token in inverted_index:
            for doc_id, doc_freq in inverted_index[token]:
                scores[doc_id] += freq * doc_freq
    # Sort the documents by score
    sorted_docs = sorted(scores.items(), key=lambda x: x[1], reverse=True)
    # Return the top-scoring documents as a DataFrame
    return df.iloc[[doc_id for doc_id, _ in sorted_docs]]

# Example usage: retrieve documents containing the word "election" and rank th
query = "UK election"
result_df = score_documents(query, inverted_index, df)
print(result_df.head())
```

	Unnamed: 0		descr	\
199	199	kennedy looks to election gains	they may not ...	
963	963	kennedys cautious optimism	charles kennedy is...	
1003	1003	labour mps fears over squabbling	if there is ...	
914	914	what the election should really be about	a ge...	
1000	1000	february poll claim speculation	reports that ...	

	tags
199	politics, iraq, westminster hq, liberal democr...
963	politics, united kingdom, labour government, c...
1003	politics, mr brown, africa, brown camp, labour...
914	social issues, politics, mps, united kingdom, ...
1000	politics, baghdad, the sunday times, sunday te...

In [24]:
```
# Define a regular expression to tokenize the text
tokenizer = re.compile(r"\w+")

# Create an empty dictionary to store the inverted index
inverted_index = defaultdict(list)

# Loop over each document in the dataset
for i, row in df.iterrows():
    # Tokenize the text
    tokens = tokenizer.findall(row['descr'])
    # Count the frequency of each token
    term_freq = defaultdict(int)
    for token in tokens:
        term_freq[token] += 1
    # Add the document to the inverted index for each unique token
    for token, freq in term_freq.items():
        inverted_index[token].append((i, freq))

# Define a function to score documents based on a query
def score_documents(query, inverted_index, df):
    # Tokenize the query
    tokens = tokenizer.findall(query)
    # Count the frequency of each token
    query_freq = defaultdict(int)
    for token in tokens:
        query_freq[token] += 1
    # Compute the score for each document
    scores = defaultdict(int)
    for token, freq in query_freq.items():
        if token in inverted_index:
            for doc_id, doc_freq in inverted_index[token]:
                scores[doc_id] += freq * doc_freq
    # Sort the documents by score
    sorted_docs = sorted(scores.items(), key=lambda x: x[1], reverse=True)
    # Return the top-scoring documents as a DataFrame
    return df.iloc[[doc_id for doc_id, _ in sorted_docs]]

# Example usage: retrieve documents containing the word "white" and ranking th
query = "white house"
result_df = score_documents(query, inverted_index, df)
print(result_df.head())
```

```
      Unnamed: 0                                              descr  \
1836        1836  house prices rebound says halifax  uk house pr...
1430        1430  white admits to balco drugs link  banned ameri...
1629        1629  uk house prices dip in november  uk house pric...
367          367  white prepared for battle  toughscrummaging pr...
859          859  no more concessions on terror  charles clarke ...

                                                   tags
1836  business, halifax, london, halifax, capital ec...
1430  san francisco chronicle, la times, oil, remy k...
1629  business, halifax, london, united kingdom, nor...
367   sports, zurich, cardiff, leicester, bristol, b...
859   politics, mps, bbc radio, bbc news, law lords,...
```

In the above result generated, we see that the documents retrieved are not very accurate for the query we specified as the results contain the token "white" or "house" and not both of them together. To solve this issue we use phrase queries, which we will see in the next sections.

In [25]:
```python
# Define a regular expression to tokenize the text
tokenizer = re.compile(r"\w+")

# Create an empty dictionary to store the inverted index
inverted_index = defaultdict(list)

# Loop over each document in the dataset
for i, row in df.iterrows():
    # Tokenize the text
    tokens = tokenizer.findall(row['descr'])
    # Count the frequency of each token
    term_freq = defaultdict(int)
    for token in tokens:
        term_freq[token] += 1
    # Add the document to the inverted index for each unique token
    for token, freq in term_freq.items():
        inverted_index[token].append((i, freq))

# Define a function to score documents based on a query
def score_documents(query, inverted_index, df):
    # Tokenize the query
    tokens = tokenizer.findall(query.lower())
    # Count the frequency of each token
    query_freq = defaultdict(int)
    for token in tokens:
        query_freq[token] += 1
    # Compute the score for each document
    scores = defaultdict(int)
    for token, freq in query_freq.items():
        if token in inverted_index:
            for doc_id, doc_freq in inverted_index[token]:
                scores[doc_id] += freq * doc_freq
    # Sort the documents by score
    sorted_docs = sorted(scores.items(), key=lambda x: x[1], reverse=True)
    # Return the top-scoring documents as a DataFrame
    return df.iloc[[doc_id for doc_id, _ in sorted_docs]]

# Example usage: retrieve documents containing the words "chelsea" and rank th
query = "chelsea"
result_df = score_documents(query, inverted_index, df)
print(result_df.head())
```

```
     Unnamed: 0                                              descr  \
0             0  chelsea sack mutu  chelsea have sacked adrian ...
12           12  desailly backs blues revenge trip  marcel desa...
139         139  desailly backs blues revenge trip  marcel desa...
155         155  chelsea hold arsenal  a gripping game between ...
358         358  chelsea clinch cup in extratime  after extrati...

                                                  tags
0    sports, stamford bridge, football association,...
12   sports, barcelona, milan, the chelsea, bbc, eu...
139  sports, barcelona, milan, the chelsea, bbc, eu...
155  sports, henry, thierry henry, robert pires, wi...
358  sports, barcelona, liverpool, newcastle, reds,...
```

The above query is accurate as it retrieves all documents containing the word "chelsea" in it

In [26]:
```
# Define a regular expression to tokenize the text
tokenizer = re.compile(r"\w+")

# Create an empty dictionary to store the inverted index
inverted_index = defaultdict(list)

# Loop over each document in the dataset
for i, row in df.iterrows():
    # Tokenize the text
    tokens = tokenizer.findall(row['descr'])
    # Count the frequency of each token
    term_freq = defaultdict(int)
    for token in tokens:
        term_freq[token] += 1
    # Add the document to the inverted index for each unique token
    for token, freq in term_freq.items():
        inverted_index[token].append((i, freq))

# Define a function to score documents based on a query
def score_documents(query, inverted_index, df):
    # Tokenize the query
    tokens = tokenizer.findall(query.lower())
    # Count the frequency of each token
    query_freq = defaultdict(int)
    for token in tokens:
        query_freq[token] += 1
    # Compute the score for each document
    scores = defaultdict(int)
    for token, freq in query_freq.items():
        if token in inverted_index:
            for doc_id, doc_freq in inverted_index[token]:
                scores[doc_id] += freq * doc_freq
    # Sort the documents by score
    sorted_docs = sorted(scores.items(), key=lambda x: x[1], reverse=True)
    # Return the top-scoring documents as a DataFrame
    return df.iloc[[doc_id for doc_id, _ in sorted_docs]]

# Example usage: retrieve documents containing the words "robot" and rank them
query = "robot"
result_df = score_documents(query, inverted_index, df)
print(result_df.head())
```

```
      Unnamed: 0                                              descr  \
711          711  robots learn robotiquette rules  robots are le...
426          426  humanoid robot learns how to run  carmaker hon...
780          780  hitachi unveils fastest robot  japanese electr...
1244        1244  humanoid robot learns how to run  carmaker hon...
721          721  gadget show heralds mp christmas  partners of ...

                                                 tags
711   technology, london, bbc, london's science muse...
426   sony, honda, czech republic, electronics, car ...
780   technology, washington dc, toyota, asimo, sony...
1244  spokesman, kelly holmes, king, europe, epo, ru...
721   entertainment, technology, london, sony, creat...
```

The above query is accurate as it retrieves all documents containing the word "robot" in it

In [27]:
```
# Define a regular expression to tokenize the text
tokenizer = re.compile(r"\w+")

# Create an empty dictionary to store the inverted index
inverted_index = defaultdict(list)

# Loop over each document in the dataset
for i, row in df.iterrows():
    # Tokenize the text
    tokens = tokenizer.findall(row['descr'])
    # Count the frequency of each token
    term_freq = defaultdict(int)
    for token in tokens:
        term_freq[token] += 1
    # Add the document to the inverted index for each unique token
    for token, freq in term_freq.items():
        inverted_index[token].append((i, freq))

# Define a function to score documents based on a query
def score_documents(query, inverted_index, df):
    # Tokenize the query
    tokens = tokenizer.findall(query.lower())
    # Count the frequency of each token
    query_freq = defaultdict(int)
    for token in tokens:
        query_freq[token] += 1
    # Compute the score for each document
    scores = defaultdict(int)
    for token, freq in query_freq.items():
        if token in inverted_index:
            for doc_id, doc_freq in inverted_index[token]:
                scores[doc_id] += freq * doc_freq
    # Sort the documents by score
    sorted_docs = sorted(scores.items(), key=lambda x: x[1], reverse=True)
    # Return the top-scoring documents as a DataFrame
    return df.iloc[[doc_id for doc_id, _ in sorted_docs]]

# Example usage: retrieve documents containing the words "hollywood" and rank
query = "hollywood"
result_df = score_documents(query, inverted_index, df)
print(result_df.head())
```

```
      Unnamed: 0                                              descr  \
1091        1091  godzilla gets hollywood fame star  movie monst...
2199        2199  keanu reeves given hollywood star  actor keanu...
754          754  games win for bluray dvd format  the nextgener...
762          762  games win for bluray dvd format  the nextgener...
469          469  movie body hits peertopeer nets  the movie ind...

                                                 tags
1091  entertainment, human interest, walk of fame, g...
2199  entertainment, human interest, beirut, volvo, ...
754   entertainment, technology, las vegas, dell, to...
762   entertainment, technology, las vegas, dell, to...
469   entertainment, technology, law, phoenix, bitto...
```

The above query is accurate as it retrieves all documents containing the word "hollywood" in it

In [28]:
```
# Define a regular expression to tokenize the text
tokenizer = re.compile(r"\w+")

# Create an empty dictionary to store the inverted index
inverted_index = defaultdict(list)

# Loop over each document in the dataset
for i, row in df.iterrows():
    # Tokenize the text
    tokens = tokenizer.findall(row['descr'])
    # Count the frequency of each token
    term_freq = defaultdict(int)
    for token in tokens:
        term_freq[token] += 1
    # Add the document to the inverted index for each unique token
    for token, freq in term_freq.items():
        inverted_index[token].append((i, freq))

# Define a function to score documents based on a query
def score_documents(query, inverted_index, df):
    # Tokenize the query
    tokens = tokenizer.findall(query.lower())
    # Count the frequency of each token
    query_freq = defaultdict(int)
    for token in tokens:
        query_freq[token] += 1
    # Compute the score for each document
    scores = defaultdict(int)
    for token, freq in query_freq.items():
        if token in inverted_index:
            for doc_id, doc_freq in inverted_index[token]:
                scores[doc_id] += freq * doc_freq
    # Sort the documents by score
    sorted_docs = sorted(scores.items(), key=lambda x: x[1], reverse=True)
    # Return the top-scoring documents as a DataFrame
    return df.iloc[[doc_id for doc_id, _ in sorted_docs]]

# Example usage: retrieve documents containing the words "apple" and rank them
query = "appl"
result_df = score_documents(query, inverted_index, df)
print(result_df.head())
```

```
Empty DataFrame
Columns: [Unnamed: 0, descr, tags]
Index: []
```

We see that for the query "apple", we get the above reults which are accurate. We can cross verify this by seeing the type of news this is in the tags fields, it says technology for the retrieved data.

In [29]:
```
# Define a regular expression to tokenize the text
tokenizer = re.compile(r"\w+")

# Create an empty dictionary to store the inverted index
inverted_index = defaultdict(list)

# Loop over each document in the dataset
for i, row in df.iterrows():
    # Tokenize the text
    tokens = tokenizer.findall(row['descr'])
    # Count the frequency of each token
    term_freq = defaultdict(int)
    for token in tokens:
        term_freq[token] += 1
    # Add the document to the inverted index for each unique token
    for token, freq in term_freq.items():
        inverted_index[token].append((i, freq))

# Define a function to score documents based on a query
def score_documents(query, inverted_index, df):
    # Tokenize the query
    tokens = tokenizer.findall(query.lower())
    # Count the frequency of each token
    query_freq = defaultdict(int)
    for token in tokens:
        query_freq[token] += 1
    # Compute the score for each document
    scores = defaultdict(int)
    for token, freq in query_freq.items():
        if token in inverted_index:
            for doc_id, doc_freq in inverted_index[token]:
                scores[doc_id] += freq * doc_freq
    # Sort the documents by score
    sorted_docs = sorted(scores.items(), key=lambda x: x[1], reverse=True)
    # Return the top-scoring documents as a DataFrame
    return df.iloc[[doc_id for doc_id, _ in sorted_docs]]

# Example usage: retrieve documents containing the words "artificial" and "int
query = "blockchain"
result_df = score_documents(query, inverted_index, df)
print(result_df.head())
```

```
Empty DataFrame
Columns: [Unnamed: 0, descr, tags]
Index: []
```

For this particular query, we do not have any document in the dataset which has the word "blockchain". Hence the retrieved dataframe is empty output for the query and would be an empty result set. That is, the query would return no documents, since there are no documents that match the query.

In [30]:
```
# Define a regular expression to tokenize the text
tokenizer = re.compile(r"\w+")

# Create an empty dictionary to store the inverted index
inverted_index = defaultdict(list)

# Loop over each document in the dataset
for i, row in df.iterrows():
    # Tokenize the text
    tokens = tokenizer.findall(row['descr'])
    # Count the frequency of each token
    term_freq = defaultdict(int)
    for token in tokens:
        term_freq[token] += 1
    # Add the document to the inverted index for each unique token
    for token, freq in term_freq.items():
        inverted_index[token].append((i, freq))

# Define a function to score documents based on a query
def score_documents(query, inverted_index, df):
    # Tokenize the query
    tokens = tokenizer.findall(query.lower())
    # Count the frequency of each token
    query_freq = defaultdict(int)
    for token in tokens:
        query_freq[token] += 1
    # Compute the score for each document
    scores = defaultdict(int)
    for token, freq in query_freq.items():
        if token in inverted_index:
            for doc_id, doc_freq in inverted_index[token]:
                scores[doc_id] += freq * doc_freq
    # Sort the documents by score
    sorted_docs = sorted(scores.items(), key=lambda x: x[1], reverse=True)
    # Return the top-scoring documents as a DataFrame
    return df.iloc[[doc_id for doc_id, _ in sorted_docs]]

# Example usage: retrieve documents containing the words "artificial" and "int
query = "pear"
result_df = score_documents(query, inverted_index, df)
print(result_df.head())
```

```
Empty DataFrame
Columns: [Unnamed: 0, descr, tags]
Index: []
```

When we searched for the query "apple" we got documents which matched with it. But when we tried it with a different fruit, "pear" we have no matching documents in the dataset

Section 4.3

Result of Wild Card queries

In this section, we have used the asterisks symbol at multiple places in the query. The description of each is given below the output of the code snippets

In [31]:
```
# read in the dataset
df = pd.read_csv("/content/BBCNews.csv")

# define the wildcard search function
def wildcard_search(query, df):
    # replace the wildcard character with a regular expression
    query = query.replace("*", ".*")
    query = query.replace("?", ".")

    # search the text column of the dataframe using the regular expression
    results = df[df["descr"].str.contains(query, case=False)].index

    return results

# test the wildcard search function
query = "robo*"
results = wildcard_search(query, df)
print(results)
```

```
Int64Index([ 124,  426,  492,  500,  541,  544,  579,  588,  711,  721,  728,
             734,  780,  920, 1025, 1232, 1244, 1310, 1318, 1359, 1362, 1397,
            1406, 2071, 2098, 2118, 2250],
           dtype='int64')
```

The above function will return a list of document indices that contain words starting with "robo", such as "robot" and "robotics".

In [32]:
```
# read in the dataset
df = pd.read_csv("/content/BBCNews.csv")

# define the wildcard search function
def wildcard_search(query, df):
    # replace the wildcard character with a regular expression
    query = query.replace("*", ".*")
    query = query.replace("?", ".")

    # search the text column of the dataframe using the regular expression
    results = df[df["descr"].str.contains(query, case=False)].index

    return results

# test the wildcard search function
query = "*tech"
results = wildcard_search(query, df)
print(results)
```

```
Int64Index([  37,   38,   84,   92,   95,  117,  152,  204,  238,  256,
            ...
            2293, 2299, 2317, 2320, 2339, 2349, 2358, 2366, 2369, 2398],
           dtype='int64', length=420)
```

The above query will match all documents that end with the word "tech". This could be useful for finding articles about different types of technology or tech-related news.

In [33]:
```
# read in the dataset
df = pd.read_csv("/content/BBCNews.csv")

# define the wildcard search function
def wildcard_search(query, df):
    # replace the wildcard character with a regular expression
    query = query.replace("*", ".*")
    query = query.replace("?", ".")

    # search the text column of the dataframe using the regular expression
    results = df[df["descr"].str.contains(query, case=False)].index

    return results

# test the wildcard search function
query = "cyber*attack"
results = wildcard_search(query, df)
print(results)
```

```
Int64Index([474, 482, 516, 525, 554, 786, 1292, 1300, 1334, 1343, 1372], dtyp
e='int64')
```

The above query will match all documents that contain words starting with "cyber" and ending with "attack". This could be useful for finding articles about cybersecurity, cybercrime, or cyber attacks.

In [34]:
```
# read in the dataset
df = pd.read_csv("/content/BBCNews.csv")

# define the wildcard search function
def wildcard_search(query, df):
    # replace the wildcard character with a regular expression
    query = query.replace("*", ".*")
    query = query.replace("?", ".")

    # search the text column of the dataframe using the regular expression
    results = df[df["descr"].str.contains(query, case=False)].index

    return results

# test the wildcard search function
query = "mark*crash"
results = wildcard_search(query, df)
print(results)
```

```
Int64Index([22, 92, 152, 181, 495, 573, 807, 822, 1313, 1391, 1516, 1580, 179
0,
            1969],
           dtype='int64')
```

In [35]:
```
# read in the dataset
df = pd.read_csv("/content/BBCNews.csv")

# define the wildcard search function
def wildcard_search(query, df):
    # replace the wildcard character with a regular expression
    query = query.replace("*", ".*")
    query = query.replace("?", ".")

    # search the text column of the dataframe using the regular expression
    results = df[df["descr"].str.contains(query, case=False)].index

    return results

# test the wildcard search function
query = "*energy*"
results = wildcard_search(query, df)
print(results)
```

```
Int64Index([  21,   57,  161,  162,  191,  212,  221,  238,  253,  308,  345,
             351,  359,  360,  420,  487,  571,  572,  689,  723,  741,  746,
             783,  813,  862,  973, 1216, 1305, 1389, 1390, 1433, 1434, 1439,
            1472, 1516, 1575, 1607, 1634, 1678, 1685, 1695, 1705, 1712, 1731,
            1738, 1745, 1770, 1788, 1792, 1812, 1837, 1847, 1853, 1867, 1876,
            1895, 1910, 1914, 1927, 1929, 1931, 1934, 2007, 2211, 2227, 2232,
            2233, 2235, 2244, 2278, 2291, 2312, 2343, 2362, 2372, 2389, 240
8],
           dtype='int64')
```

The above query will match all documents that contain the word "energy" or any word that starts or ends with "energy". This could be useful for finding articles about renewable energy, fossil fuels, or energy policy.

Section 4.4

Result of Phrase queries:

In free text queries we saw that if either of the words were present in the query, we got a result. In phrase queries, unlike free txt queries the query given has to be in the same format to find a match.

In [36]:
```python
# Define a phrase to search for
phrase = 'climate change'

# Find all documents that contain the phrase
matches = df[df['descr'].str.contains(phrase)]

# Print the number of documents that match the query
print(f"Found {len(matches)} documents containing the phrase '{phrase}':")

# Loop over the matching documents and print their IDs, titles, and text
for i, row in matches.iterrows():
    print(f"\nDocument ID: {i}")
    print(f"Article: {row['descr']}")
    print(f"Tags: {row['tags']}")
```

```
Found 11 documents containing the phrase 'climate change':

Document ID: 221
Article: uk set to cut back on embassies  nine overseas embassies and high
commissions will close in an effort to save money uk foreign secretary jac
k straw has announced  the bahamas east timor madagascar and swaziland are
among the areas affected by the biggest shakeup for the diplomatic service
for years other diplomatic posts are being turned over to local staff mr s
traw said the move would save m a year to free up cash for priorities such
as fighting terrorism  honorary consuls will be appointed in some of the a
reas affected by the embassy closures nine consulates or consulates genera
l will also be closed mostly in europe and america  they include dallas in
the us bordeaux in france and oporto in portugal with local staff replacin
g uk representation in another  the changes are due to be put in place bef
ore the end of  with most savings made from cutting staff and running cost
s some of the money will have to be used to fund redundancy payments in a
written statement mr straw said the savings made will help to underpin hig
her priority work in line with the foreign and commonwealth offices strate
gic priorities including counter proliferation counterterrorism energy and
```

In [37]:

```
# Define a phrase to search for
phrase = 'gender pay gap'

# Find all documents that contain the phrase
matches = df[df['descr'].str.contains(phrase)]

# Print the number of documents that match the query
print(f"Found {len(matches)} documents containing the phrase '{phrase}':")

# Loop over the matching documents and print their IDs, titles, and text
for i, row in matches.iterrows():
    print(f"\nDocument ID: {i}")
    print(f"Article: {row['descr']}")
    print(f"Tags: {row['tags']}")
```

Found 1 documents containing the phrase 'gender pay gap':

Document ID: 1161
Article: hewitt decries career sexism plans to extend paid maternity leave b
eyond six months should be prominent in labours election manifesto the trade
and industry secretary has said patricia hewitt said the cost of the proposa
ls was being evaluated but it was an increasingly high priority and a shared
goal across government ms hewitt was speaking at a gender and productivity se
minar organised by the equal opportunities commission eoc mothers can current
ly take up to six months paid leave and six unpaid ms hewitt told the semina
r clearly one of the things we need to do in the future is to extend the peri
od of payment for maternity leave beyond the first six months into the second
six months we are looking at how quickly we can do that because obviously the
re are cost implications because the taxpayer reimburses the employers for th
e cost of that ms hewitt also announced a new drive to help women who want t
o work in male dominated sectors saying sexism at work was still preventing w
omen reaching their full potential plans include funding for universities to
help female science and engineering graduates find jobs and taster courses fo
r men and women in nontraditional jobs women in fulltime work earn less than
men according to the equal opportunities commission eoc the minister told de
legates that getting rid of career sexism was vital to closing the gender pay
gap career sexism limits opportunities for women of all ages and prevents th
em from achieving their full potential it is simply wrong to assume someone c
annot do a job on the grounds of their sex she said earlier she told bbc radi
o s today programme what we are talking about here is the fact that about six
out of women work in jobs that are lowpaid and typically dominated by women
so we have got very segregated employment unfortunately in some cases this re
flects very oldfashioned and stereotypical ideas about the appropriate jobs f
or women or indeed for men career sexism is about saying that engineering for
instance where only of employees are women is really a maledominated industr
y construction is even worse but it is also about saying childcare jobs are r
eally there for women and not suitable for men career sexism goes both ways
she added that while progress had been made there was still a gap in pay figu
res the average woman working fulltime is being paid about p for every pound
a man is earning for women working parttime it is p the department for trade
and industry will also provide funding to help a new pay experts panel run by
the tuc it has been set up to advise hundreds of companies on equal wage pol
icies research conducted by the eoc last year revealed that many britons beli
eve the pay gap between men and women is the result of natural differences be
tween the sexes women hold less than of the top positions in ftse companies
the police the judiciary and trade unions according to their figures and reti
red women have just over half the income of their male counterparts on averag
e
Tags: labor, bbc radio, ftse 100, equal opportunities commission, department
for trade and industry, patricia hewitt, industry secretary, minister, bbc ra
dio 4, gender equality, family law, discrimination, sexism, family, politics,
chauvinism, human behavior, patricia hewitt, parental leave, maternity leave
in the united states, gender pay gap in russia

In [38]:
```
# Define a phrase to search for
phrase = 'global economic'

# Find all documents that contain the phrase
matches = df[df['descr'].str.contains(phrase)]

# Print the number of documents that match the query
print(f"Found {len(matches)} documents containing the phrase '{phrase}':")

# Loop over the matching documents and print their IDs, titles, and text
for i, row in matches.iterrows():
    print(f"\nDocument ID: {i}")
    print(f"Article: {row['descr']}")
    print(f"Tags: {row['tags']}")
```

```
Found 7 documents containing the phrase 'global economic':

Document ID: 1690
Article: newest eu members underpin growth  the european unions newest mem
bers will bolster europes economic growth in  according to a new report  t
he eight central european states which joined the eu last year will see  g
rowth the united nations economic commission for europe unece said in cont
rast the  euro zone countries will put in a lacklustre performance generat
ing growth of only  the global economy will slow in  the unece forecasts d
ue to widespread weakness in consumer demand  it warned that growth could
also be threatened by attempts to reduce the united states huge current ac
count deficit which in turn might lead to significant volatility in exchan
ge rates  unece is forecasting average economic growth of  across the euro
pean union in  however total output across the euro zone is forecast to fa
ll in  from  to  this is due largely to the faltering german economy which
shrank  in the last quarter of  on monday germanys bdb private banks assoc
iation said the german economy would struggle to meet its  growth target i
n  separately the bundesbank warned that germanys efforts to reduce its b
udget deficit below  of gdp presented huge risks given that headline econo
```

In [39]:
```
# Define a phrase to search for
phrase = 'food waste reduction initiatives'

# Find all documents that contain the phrase
matches = df[df['descr'].str.contains(phrase)]

# Print the number of documents that match the query
print(f"Found {len(matches)} documents containing the phrase '{phrase}':")

# Loop over the matching documents and print their IDs, titles, and text
for i, row in matches.iterrows():
    print(f"\nDocument ID: {i}")
    print(f"Article: {row['descr']}")
    print(f"Tags: {row['tags']}")
```

```
Found 0 documents containing the phrase 'food waste reduction initiatives':
```

The above phrase has no documents found as no document in the dataset has the above phrase in it

In [40]:
```
# Define a phrase to search for
phrase = 'human rights'

# Find all documents that contain the phrase
matches = df[df['descr'].str.contains(phrase)]

# Print the number of documents that match the query
print(f"Found {len(matches)} documents containing the phrase '{phrase}':")

# Loop over the matching documents and print their IDs, titles, and text
for i, row in matches.iterrows():
    print(f"\nDocument ID: {i}")
    print(f"Article: {row['descr']}")
    print(f"Tags: {row['tags']}")
```

```
Found 55 documents containing the phrase 'human rights':

Document ID: 216
Article: custody death rate shocks mps  deaths in custody have reached sho
cking levels a committee of mps and peers has warned  the joint committee
on human rights found those committing suicide were mainly the most vulner
able with mental health drugs or alcohol problems members urged the govern
ment to set up a task force to tackle deaths in prisons police cells deten
tion centres and special hospitals there was one prison suicide every four
days between  and  mps said the report which followed a yearlong inquiry b
y the committee found the high death rate amounts to a serious failure to
protect the right to life of a highly vulnerable group  many of those who
ended up taking their own lives had presented themselves to the authoritie
s with these problems before they even offended the report said it questio
ned whether prison was the most appropriate place for them to be kept and
whether earlier intervention would have meant custody could have been avoi
ded  increased resources and a reduction in the use of imprisonment was ne
eded to address the issue in the longer term the report said committee cha
irman labour mp jean corston said each and every death in custody is a dea
```

The above output is very large. We have used the below code to retrieve the 10 most relevant documents containing "human rights" in them. We have used TF-IDF and cosine similarity to get this output

In [41]:
```
# Load the dataset into a pandas dataframe
df = pd.read_csv("/content/BBCNews.csv")

# Create a document-term matrix using TfidfVectorizer
vectorizer = TfidfVectorizer()
doc_term_matrix = vectorizer.fit_transform(df['descr'])

# Define the phrase query and perform the search
query = "human rights"
query_vec = vectorizer.transform([query])
doc_scores = cosine_similarity(query_vec, doc_term_matrix)[0]
indices = np.argsort(doc_scores)[::-1]

# Print the top 10 documents with matching phrase and their document ids
print("Top 10 documents with matching phrase:")
for i in range(10):
    doc_id = indices[i]
    doc_text = df.iloc[doc_id]['descr']
    doc_score = doc_scores[doc_id]
    if "human rights" in doc_text:
        print(f"Document ID: {doc_id}, Score: {doc_score}")
        print(f'Article: {doc_text}')
        print("----------")
```

```
Top 10 documents with matching phrase:
Document ID: 830, Score: 0.6321156770132687
Article: amnesty chief laments war failure  the lack of public outrage abo
ut the war on terror is a powerful indictment of the failure of human righ
ts groups amnesty internationals chief has said  in a lecture at the londo
n school of economics irene khan said human rights had been flouted in the
name of security since  september  she said the human rights movement had
to use simpler language both to prevent scepticism and spread a moral mess
age and it had to fight poverty not just focus on political rights for eli
tes  ms khan highlighted detentions without trial including those at the u
s camp at guantanamo bay in cuba and the abuse of prisoners as evidence of
increasing human rights problems whats a new challenge is the way in which
this ageold debate on security and human rights has been translated into t
he language of war she said by using the language of war human rights are
being sidelined because we know human rights do not apply in times of war
ms khan said such breaches were infectious and were now seen in almost ver
y major country in the world the human rights movement faces a crisis of f
aith in the value of human rights she said that was accompanied by a crisi
s of governance where the united nations system did not seem able to hold
```

In [42]:
```python
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

# Load the dataset into a pandas dataframe
#df = pd.read_csv("BBC News Dataset.csv")

# Create a document-term matrix using TfidfVectorizer
vectorizer = TfidfVectorizer()
doc_term_matrix = vectorizer.fit_transform(df['descr'])

# Define the phrase query and perform the search
query = "economic growth"
query_vec = vectorizer.transform([query])
doc_scores = cosine_similarity(query_vec, doc_term_matrix)[0]
indices = np.argsort(doc_scores)[::-1]

# Print the top 10 documents with matching phrase and their document ids
print("Top 10 documents with matching phrase:")
for i in range(10):
    doc_id = indices[i]
    doc_text = df.iloc[doc_id]['descr']
    doc_score = doc_scores[doc_id]
    if "economic growth" in doc_text:
        print(f"Document ID: {doc_id}, Score: {doc_score}")
        print(f'Article: {doc_text}')
        print("----------")
```

```
Top 10 documents with matching phrase:
Document ID: 1690, Score: 0.4048869826479714
Article: newest eu members underpin growth  the european unions newest mem
bers will bolster europes economic growth in  according to a new report  t
he eight central european states which joined the eu last year will see  g
rowth the united nations economic commission for europe unece said in cont
rast the  euro zone countries will put in a lacklustre performance generat
ing growth of only  the global economy will slow in  the unece forecasts d
ue to widespread weakness in consumer demand  it warned that growth could
also be threatened by attempts to reduce the united states huge current ac
count deficit which in turn might lead to significant volatility in exchan
ge rates  unece is forecasting average economic growth of  across the euro
pean union in  however total output across the euro zone is forecast to fa
ll in  from  to  this is due largely to the faltering german economy which
shrank  in the last quarter of  on monday germanys bdb private banks assoc
iation said the german economy would struggle to meet its  growth target i
n  separately the bundesbank warned that germanys efforts to reduce its b
udget deficit below  of gdp presented huge risks given that headline econo
mic growth was set to fall below  this year publishing its  economic surve
```

The code defines the phrase query "climate change action" and performs the search using cosine similarity between the query vector and the document-term matrix. The code prints the top 10 documents with matching phrase and their document IDs, ranked by the cosine similarity score.

Section 4.5

How the evaluator can test, an arbitrary text query relevant to your corpus?

Ans: For an evaluator to test an arbitary text query, we can use any of the above mentioned methods, free text, wild card or phrase queries. One method to do this is by taking an input from the user and performing the above code snippets from section 4.1, 4.2, 4.3 and 4.4. The other way is we manually plug in the input query given by the user and run the code. Both ways, we can figure out whether the query is relevant to our corpus or not. It acts as a serch engine for a specified dataset. If the query is relevant, the ouput is a set of documents containing the query. We can further enhance the relevance of the query by using cosine similarity, TF-IDF, ranking and other parameters to retrieve the most relevant documents. If the output is empty, we know that the query provided is irrelevent to the corpus.

In greater detail, we can follow the below steps to evaluate an arbitary text query.

To evaluate the relevance of an arbitrary text query to the corpus, you can use a common evaluation metric in information retrieval called "precision at k" (P@k).

Here's how it works:

- Choose a value for k, which represents the number of documents to consider for each query.
- For each query, retrieve the top k documents from the corpus that match the query. You can use any information retrieval technique such as TF-IDF, BM25, or neural models for this.
- Evaluate the relevance of each retrieved document using some relevance criteria such as relevance judgments made by human assessors.
- Compute the precision at k by dividing the number of relevant documents retrieved by the total number of documents retrieved up to k.
- Repeat the above steps for multiple queries to obtain an average P@k score.

By using precision at k, you can evaluate how well a retrieval system is performing for a given query. The higher the P@k score, the more relevant the retrieved documents are for the query.

The choice of k depends on the specific application and the available resources. A larger k would give more comprehensive results but would also increase the computational cost.

Section 4.6

Any one additional functionality: Relevance feedback, Semantic matching, re-ranking of results, and finding out query intention.

Semantic matching is a technique that involves identifying the underlying meaning of the query and matching it with the meaning of the documents in the corpus. This can be done using various natural language processing (NLP) techniques such as semantic analysis, named entity recognition, and dependency parsing.

The steps below can be followed to check:

- Load the dataset: You can load the BBC News dataset into a pandas dataframe using the following code:

In [43]:
```
import pandas as pd

data = pd.read_csv('/content/BBCNews.csv')
```

- Preprocess the data: Before performing any retrieval or semantic matching, you need to preprocess the data. This involves converting the text data into a format that can be used for retrieval, such as tokenizing the text, removing stop words, and stemming the words.

In [44]:
```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem.porter import PorterStemmer

nltk.download('stopwords')
nltk.download('punkt')

stemmer = PorterStemmer()
stop_words = set(stopwords.words('english'))

def preprocess_text(text):
    tokens = word_tokenize(text.lower())
    tokens = [stemmer.stem(token) for token in tokens if token not in stop_wor
    return ' '.join(tokens)

data['processed_text'] = data['descr'].apply(preprocess_text)
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

- Extract semantic features: To perform semantic matching, you need to extract semantic features from the text data. One way to do this is to use pre-trained word embeddings, such as Word2Vec or GloVe.

In [45]:
```python
import gensim.downloader as api
import numpy as np

word_vectors = api.load("glove-wiki-gigaword-100")

def extract_semantic_features(text):
    tokens = word_tokenize(text.lower())
    vectors = []
    for token in tokens:
        if token in word_vectors:
            vectors.append(word_vectors[token])
    if vectors:
        return np.mean(vectors, axis=0)
    else:
        return np.zeros(100)

data['semantic_features'] = data['processed_text'].apply(extract_semantic_feat
```

- Retrieve documents using a query: You can use semantic matching to retrieve documents based on a query by computing the cosine similarity between the semantic features of the query and the semantic features of the documents. For example, you can use the following code to retrieve the top 10 documents related to the query "technology":

In [46]:
```
import numpy as np

query = 'technology'
query_features = extract_semantic_features(query)

scores = data['semantic_features'].apply(lambda x: np.dot(x, query_features)).
ranked_documents = sorted(enumerate(scores), key=lambda x: x[1], reverse=True)

for document_index, score in ranked_documents:
    print(data.iloc[document_index]['descr'])
```

```
electronics firms eye plasma deal  consumer electronics giants hitachi and
matshushita electric are joining forces to share and develop technology fo
r flat screen televisions  the tieup comes as the worlds top producers are
having to contend with falling prices and intense competition the two japa
nese companies will collaborate in research  development production market
ing and licensing they said the agreement would enable the two companies t
o expand the plasma display tv market globally  plasma display panels are
used for large thin tvs which are replacing oldstyle televisions the displ
ay market for highdefinition televisions is split between models using pla
sma display panels and others  manufactured by the likes of sony and samsu
ng  using liquidcrystal displays lcds the deal will enable hitachi and mat
sushita which makes panasonic brand products to develop new technology and
improve their competitiveness hitachi recently announced a deal to buy pla
sma display technology from rival fujitsu in an effort to strengthen its p
resence in the market  separately fujitsu announced on monday that it is q
uitting the lcd panel market by transferring its operations in the area to
japanese manufacturer sharp sharp will inherit staff manufacturing facilit
ies and intellectual property from fujitsu the plasma panel market has see
n rapid consolidation in recent months as the price of consumer electronic
```

- Improve the performance: You can improve the performance of the retrieval system by using various techniques such as relevance feedback or query expansion. You can also try different pre-trained word embeddings or fine-tune your own word embeddings using the dataset.