Information Retrieval and Extraction
Assignment 1

REPORT

September 12, 2023

Ashna Dua
2021101072
ashna.dua@students.iiit.ac.in

Task 1: Preprocessing and Tokenisation

In this task, the text is prepared for retrieval. Punctuation, special characters, and numbers are removed using the regular expression [ˆ**a-zA-Z**]. This makes the text contain only alphabetic letters and spaces. Then, the text is separated into smaller units called tokens, which are essentially words, by splitting the text on the basis of a space. This process is applied to the first 15 articles from the NASA Corpus (alphabetically) to obtain raw data.

```python
# Clean and tokenize data
tokenized_text = []

for file_name in articles:
    file_path = os.path.join(corpus, file_name)
    with open(file_path, 'r', encoding="latin-1") as file:
        text = file.read()
        cleaned_text = re.sub(r'[^a-zA-Z\s]', ' ', text)
        tokens = cleaned_text.split()
        tokenized_text.append(tokens)
```

Task 2: Stemming and Visualization

Porter Stemming Algorithm is a widely used method in NLP to reduce words to their base or root form. It applies heuristic rules to remove common suffixes from words, simplifying them for analysis. This process involves tokenization, where words are split into tokens, and then stemming, where rules are applied to transform words into their root forms. For example, ”jumps” becomes ”jump” after stemming. In this task, the tokens were stemmed using the **PorterStemmer** from **nltk.stem**. This gives the root forms of the tokens.

```python
# Stemming of tokens
from nltk.stem import PorterStemmer
stemmer = PorterStemmer()
stemmed_tokenized = []

for tokens in tokenized_text:
    stems = []
    for token in tokens:
        st = stemmer.stem(token)
        stems.append(st)
    stemmed_tokenized.append(stems)
```

To visualize the frequency of words using **Tag Clouds**, we utilize the **WordCloud** library. The tokens are flattened into a list, and the top 50 most frequent words are used to generate tag clouds. In a tag cloud, the font size of each word is determined based on its frequency, using the formula:

$$s_i = f_{max} \cdot \frac{t_i - t_{min}}{t_{max} - t_{min}}$$

Here, s_i represents the font size of a word, f_{max} is the maximum font size, t_i is the word count, t_{min} is the minimum count, and t_{max} is the maximum count.

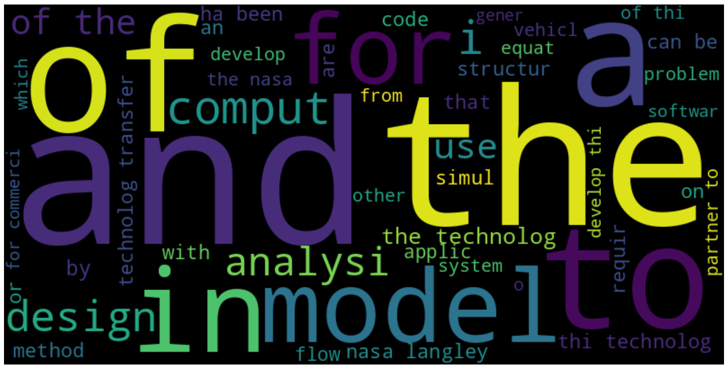


Figure 1: Tag Cloud of top 50 words (including stop words)

To visualize the frequency distribution of the 20 most frequent words in the first NASA article, the **FreqDist** class from **nltk** is utilized. The frequency distribution of words in the initial article presents the top 20 words on a bar chart.

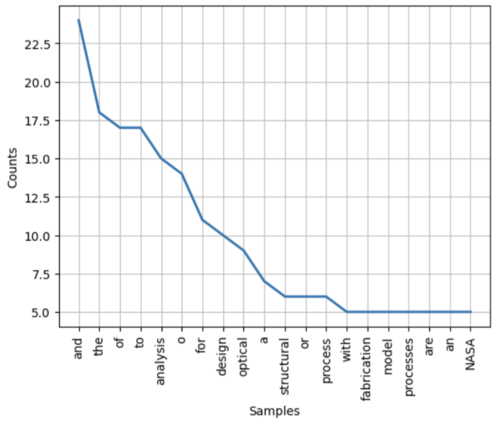


Figure 2: Frequency Distribution of first NASA article

```python
# visualizing the word cloud
flat_tokens = []
for tokens in stemmed_tokenized:
    for token in tokens:
        flat_tokens.append(token)
text = ' '.join(flat_tokens)
wordcloud = WordCloud(
    width=800, height=400, max_words=50, stopwords=[]
).generate(text)
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```

```python
# plot 20 most frequence words
from nltk import FreqDist

first_article_tokens = tokenized_text[0]
freq_dist = FreqDist(first_article_tokens)
freq_dist.plot(20, cumulative=False)
plt.show()
```

Task 3: Computation of tf and tf-idf

In this task, the Term Frequency (*tf*) and TF-IDF (*tf-idf*) are computed for each document. Subsequently, the top p stems in each document are compared based on both *tf* and *tf-idf*. These two metrics are essential in information retrieval and text mining, aiding in understanding the importance of terms within a document and across a collection.

Term Frequency (tf)

tf measures how often a term appears in a specific document. It is calculated as:

$$tf(t, d) = \frac{f_{t,d}}{\max(f_{t',d} : t' \in d)}$$

Where, $tf(t, d)$ is the **term frequency** of term t in document d and $f_{t,d}$ is the **number of occurrences** of term t in document d. The purpose of ***tf*** is to normalize the term frequency by dividing it by the maximum term frequency in the document. This normalization prevents longer documents from having an inherent advantage.

```python
# tf calculation
tf_values = []

for stemmed_doc in stemmed_all_files:
    max_term_freq = max(Counter(stemmed_doc).values())
    term_freq = {
    term: freq / max_term_freq for term, freq in Counter(stemmed_doc).items()
    }

    tf_values.append(term_freq)
```

Inverse Document Frequency (idf)

idf measures the importance of a term in the entire corpus. It is calculated as:

$$idf(t) = \log\left(\frac{N}{n_t}\right)$$

Where, ***idf(t)*** is the **Inverse Document Frequency** of term t, N is the **total number of documents** in the corpus and n_t is the **number of documents containing term** t. *idf* gives higher weight to terms that are rare across the entire corpus and lower weight to common terms.

```python
# tfidf calculation
tfidf_values = []

for doc in tf_values:
    tfidf_curr = {}

    for term, tf in doc.items():
        ni = sum(1 for d in tf_values if term in d)
        if ni > 0:
            idf = math.log(num / ni)
        else:
            idf = 0.0
        tfidf_val = tf * idf
        tfidf_curr[term] = tfidf_val

    tfidf_values.append(tfidf_curr)
```

Term Frequency-Inverse Document Frequency (tf-idf)

tf-idf combines *tf* and *idf* to assess the importance of a term within a document and the entire corpus. It is calculated as:

$$tfidf(t,d) = tf(t,d) \times idf(t)$$

The *tf-idf* value reflects how significant a term is within a document (*tf*) while considering its rarity across the entire corpus (*idf*). High *tf-idf* values are obtained for terms that are frequent within a document but relatively uncommon across the corpus.

Selecting Top p Stems

This task involved selecting the top p stems based on *tf* or *tf-idf* values for each document.

```
top_stems_tf = []

for doc in tf_values:
    top_terms = sorted(doc.items(), key=lambda x: x[1], reverse=True)[:p]
    top_stems_tf.append([term for term, _ in top_terms])

top_stems_tfidf = []

average_tfidf = {}
for doc in tfidf_values:
    for term, tfidf in doc.items():
        if term in average_tfidf:
            average_tfidf[term] += tfidf
        else:
            average_tfidf[term] = tfidf

for term, total_tfidf in average_tfidf.items():
    average_tfidf[term] = total_tfidf / num

for doc in tfidf_values:
    top_terms = sorted(doc.items(), key=lambda x: x[1], reverse=True)[:p]
    top_stems_tfidf.append([term for term, _ in top_terms])
```

Comparing Top p Stems

Two comparisons are done to gain insights into the extracted keywords among all documents. The main objective is to evaluate the differences based on two distinct keyword extraction criteria: Term Frequency (*if*) and Term Frequency-Inverse Document Frequency (*tf-idf*).

Intersection of tf and tf-idf Stems

The sets of top stems obtained from *TF* and *TF-IDF* metrics are compared to identify shared and distinct significant terms. A Venn diagram illustrates their overlap, revealing common keyword selections.

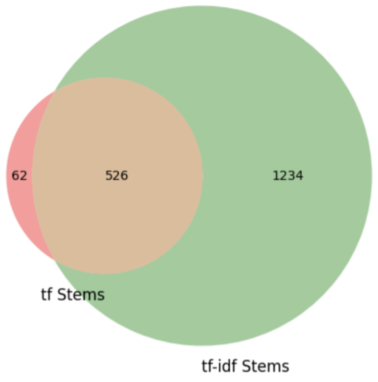


Figure 3: Intersection of tf and tf-idf Stems

Top Common Stems

The most common stems across all documents are extracted and visualized for both tf and tf-idf. The frequency of each stem and selecting the top results, the top common stems are displayed using bar charts.

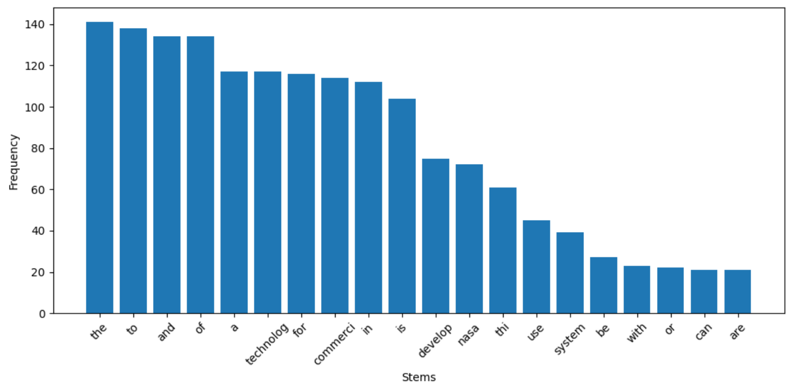


Figure 4: Top Common Stems - tf

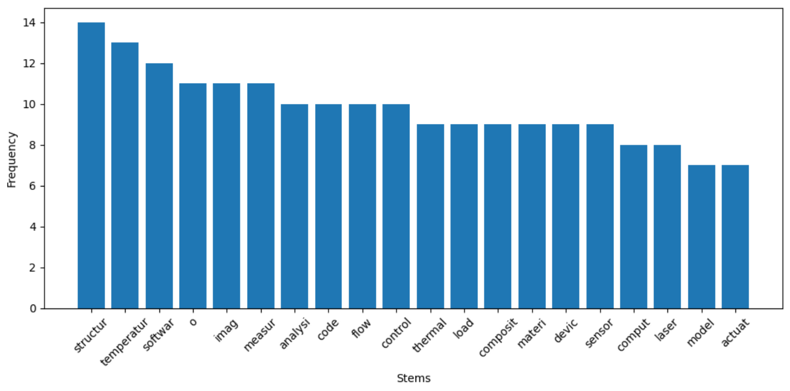


Figure 5: Top Common Stems - tf-idf

Importance of Extracting Top Stems

Selecting the top p stems based on *TF* or *TF-IDF* is essential for several reasons:

- **Information Retrieval:** When searching for documents, the extracted top stems can be used as query terms to retrieve relevant documents more efficiently.
- **Content Analysis:** Top stems provide insights into the key themes and concepts present in the entire collection, aiding content analysis and understanding of the corpus.

Differences Based on Keyword Criteria

When analyzing the extracted keywords using the Term Frequency (tf) and Term Frequency-Inverse Document Frequency (tf-idf) metrics, it becomes evident that these two methods exhibit distinct preferences in the selection of significant terms from the corpus. The intersection analysis, which identifies terms deemed important by both tf and tf-idf, reveals the presence of 526 common stems. These are the terms that both metrics concur upon as being noteworthy within the documents.

However, the true subtleties emerge when the top stems chosen by each metric individually are analyzed. There is no overlap between the terms selected as top 20 stems by tf and tf-idf. When we examine the tf-based top stems, we observe that a majority of these terms are stop words, reflecting their high frequency in the documents. This occurrence can be attributed to the inherent nature of tf, which tends to emphasize terms with higher occurrences in a document.

On the contrary, the top 20 stems chosen by tf-idf provide a more refined selection, offering a better representation of the truly significant terms within the corpus. These terms are less likely to be common stop words and, instead, tend to capture the essence of the documents more effectively.

However, the most intriguing observation arises when we consider the intersection of the two sets â the top 20 stems selected by tf and tf-idf. Remarkably, there are zero common words between these two sets. This phenomenon underscores the striking contrast in keyword selection preferences between the tf and tf-idf criteria, highlighting the essential role these methods play in capturing different facets of term importance within a corpus.

Boolean and Vector Models

In this task, the top p stems are incorporated into Boolean and Vector models to facilitate document retrieval and information retrieval tasks efficiently.

Boolean Model

In the Boolean Model, a Term-Document Matrix class is implemented with the goal of creating Boolean models based on the top p stems extracted from a corpus of text documents. The key components and steps involved in this process include:

- **Boolean Matrix Creation:** A Boolean matrix is constructed where each row corresponds to a unique term, and each column represents a document. In the Boolean matrix, '1' indicates the presence of a term in a document, while '0' signifies absence.

- **Boolean Query Representation:** The user's input query undergoes a series of transformations to create a representation suitable for Boolean querying. Firstly, the query is processed by tokenizing it into individual terms, and all terms are converted to lowercase for consistency in matching. Next, the Porter Stemmer Algorithm is applied to the query terms, reducing them to their root forms. This step aligns the query terms with the stemmed terms present in the term-document matrix, significantly enhancing the likelihood of precise matching. The final step in query representation involves the creation of Boolean Vectors. Each query term, except for the boolean operators ('and,' 'or,' and 'not'), is transformed into a Boolean Vector. In these Boolean Vectors, each position corresponds to a document in the corpus. A '1' in a particular position of the Boolean Vector indicates the presence of the query term in the corresponding document, while '0' signifies its absence. For example, consider the query "space and technology." This query is converted into a structured representation: [[Boolean Vector representing 'space'], 'and', [Boolean Vector representing 'technology']]. This representation allows for precise and efficient logical comparisons with the term-document matrix, ultimately facilitating accurate document retrieval based on the user's Boolean query expressions.

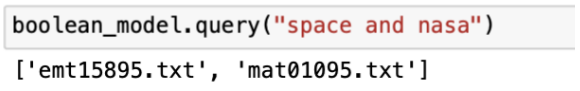
- **Boolean Query Comparison:** With the Boolean representation of the query and the TDM ready, the boolean operations ('and,' 'or,' and 'not') are carried out between the query terms and the corresponding Boolean Vectors in the TDM. This means that 'and' operation will result in '1' only if both the query terms are present in the document, 'or' operation will result in '1' if either or both of the query terms are present, and 'not' operation will result in '1' if the query term is absent in the document.

- **Result:** The result of these Boolean operations is a binary array of 1s and 0s. Each element in this array corresponds to a document in the corpus. A '1' indicates that the document satisfies the Boolean conditions set by the user's query, making it relevant, while '0' indicates that the document does not meet the criteria. The names of the relevant documents are extracted from this binary array based on the '1' values, and these document names are returned as the final result.

This Boolean Model provides a robust framework for document retrieval and information retrieval tasks, allowing users to effectively query the corpus based on the top p stems and logical expressions. The modular code allows users to change the parameters, making it easy to use and adapt to various scenarios. Whether it's searching for specific topics, filtering documents using logical operators, or customizing the number of top stems for relevance ranking, this model offers flexibility and control over the retrieval process. By transforming user queries into Boolean representations and comparing them with the Term Document Matrix, this model empowers users to retrieve precise and relevant documents, enhancing the efficiency of information retrieval tasks.

0.1 Query Display

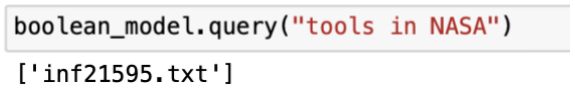
Query 1: space and nasa



```
boolean_model.query("space and nasa")
['emt15895.txt', 'mat01095.txt']
```

Figure 6: Boolean Model: space and nasa

Query 2: tools in NASA



```
boolean_model.query("tools in NASA")
['inf21595.txt']
```

Figure 7: Boolean Model: tools in NASA

Vector Model

In the Vector Model, a Term-Document Matrix class is implemented with the goal of creating Vector models based on the top p stems extracted from a corpus of text documents. The key components and steps involved in this process include:

- **Vector Matrix Creation:** A Vector matrix is constructed where each row corresponds to a unique term, and each column represents a document. In the Vector Matrix, each cell represents the $tf - idf$ value of the term, with respect to the document. These values, denoting term significance within specific documents and across the entire corpus, lend a nuanced and context-aware representation.

- **Vector Query Representation:** The user's input query undergoes a series of transformations to create a representation suitable for Vector querying. Firstly, the query is processed by tokenizing it into individual terms, and all terms are converted to lowercase for consistency in matching. Next, the Porter Stemmer Algorithm is applied to the query terms, reducing them to their root forms. This step aligns the query terms with the stemmed terms present in the term-document matrix, significantly enhancing the likelihood of precise matching. The final step in query representation involves the creation of a vector. The vector is created with a length equal to the number of unique terms in the documents. Each element in this vector is initially set to zero. The Term Frequency (tf) of each term in the query is calculated, along with the Document Frequency (df). Using the df values, idf is calculated for each term and the vector stores the tf-idf value at the corresponding position. This vector can be used for comparing the query to the documents to find the the most relevant ones.

- **Vector Query Comparison:** With the Vector representation of the query and the TDM ready, the similarity between the user's query vector and each document's vector in the TDM is calculated using cosine similarity. The formula for cosine similarity is given by:

  $$\text{Cosine Similarity} = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}$$

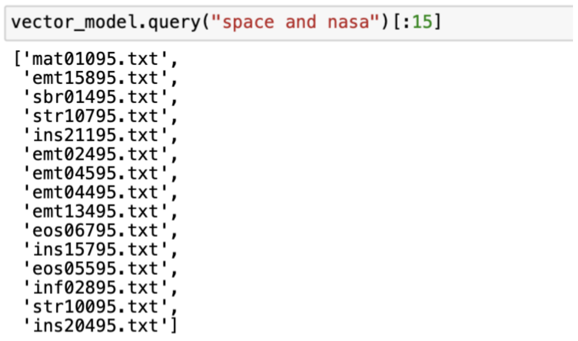
  It assesses how closely the query aligns with each document in terms of the terms they contain and their importance.

- **Result:** The the top 'n' documents that exhibit the highest cosine similarity with the user's query are identified. These documents are considered the most relevant matches to the query. If a document has a similarity score of zero, it is typically excluded from the results, as it does not match the query's content. The document names along with their similarity measures are returned as the final result.

The Vector Model leverages vector representations of the query and documents, along with cosine similarity, to find and rank the most relevant documents in the corpus that match the user's information needs. This approach enhances the efficiency and effectiveness of document retrieval and information retrieval tasks by considering the semantic similarity between the query and documents.

0.2 Query Display

Query 1: space and nasa

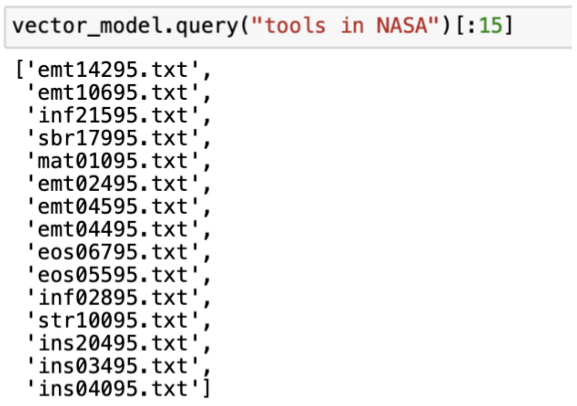


```
vector_model.query("space and nasa")[:15]
```

```
['mat01095.txt',
 'emt15895.txt',
 'sbr01495.txt',
 'str10795.txt',
 'ins21195.txt',
 'emt02495.txt',
 'emt04595.txt',
 'emt04495.txt',
 'emt13495.txt',
 'eos06795.txt',
 'ins15795.txt',
 'eos05595.txt',
 'inf02895.txt',
 'str10095.txt',
 'ins20495.txt']
```

Figure 8: Vector Model: space and nasa

Query 2: tools in NASA



```
vector_model.query("tools in NASA")[:15]
```

```
['emt14295.txt',
 'emt10695.txt',
 'inf21595.txt',
 'sbr17995.txt',
 'mat01095.txt',
 'emt02495.txt',
 'emt04595.txt',
 'emt04495.txt',
 'eos06795.txt',
 'eos05595.txt',
 'inf02895.txt',
 'str10095.txt',
 'ins20495.txt',
 'ins03495.txt',
 'ins04095.txt']
```

Figure 9: Vector Model: tools in NASA

Ranking Comparisons

The top 20 rankings produced by Vector and Boolean Models are compared using two different comparisons: Venn Diagrams and Bar Plots. Venn Diagram signifies the intersection of results of both the models and Bar Plot signifies the frequency of the results of both the models.

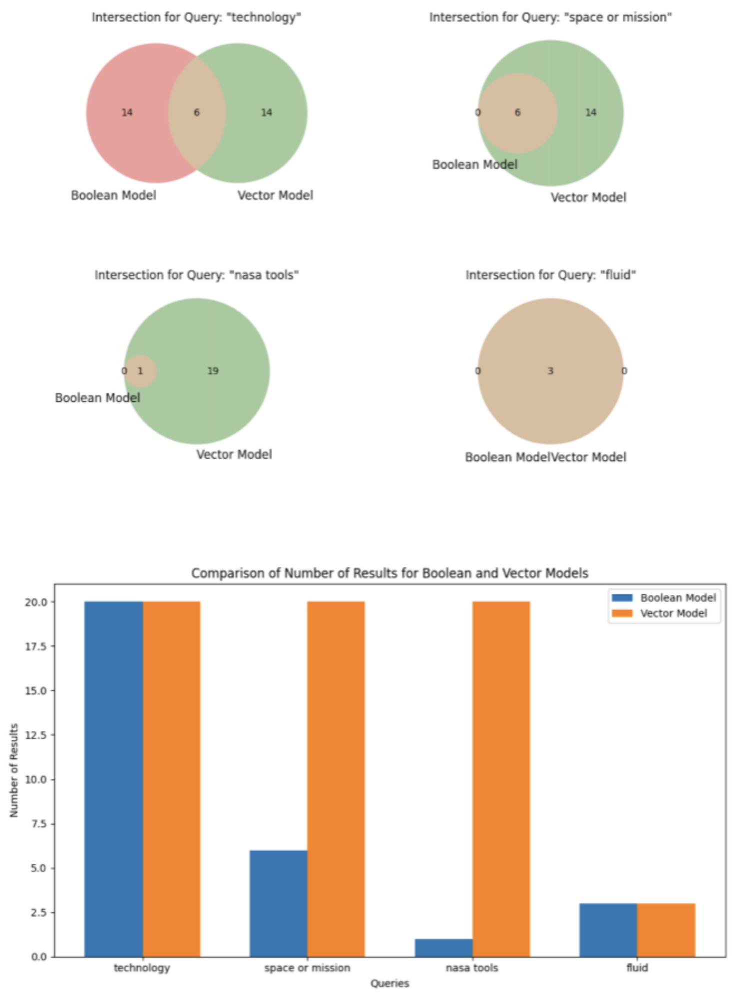


Figure 10: Ranking Comparison

Task 5: Stop Words Removal

In this task, the stop words mentioned in **english.stop** are used to remove stopwords from the text corpus. The removal of stopwords is a crucial step in the preprocessing of text data. Stop words usually refer to the most common words in a language and they do not contain important meaning and are usually removed from texts. Stemming is performed on the non-stop words which were converted into tokens. The tokens are flattened into a list, and the top 50 most frequent words are used to generate tag clouds.

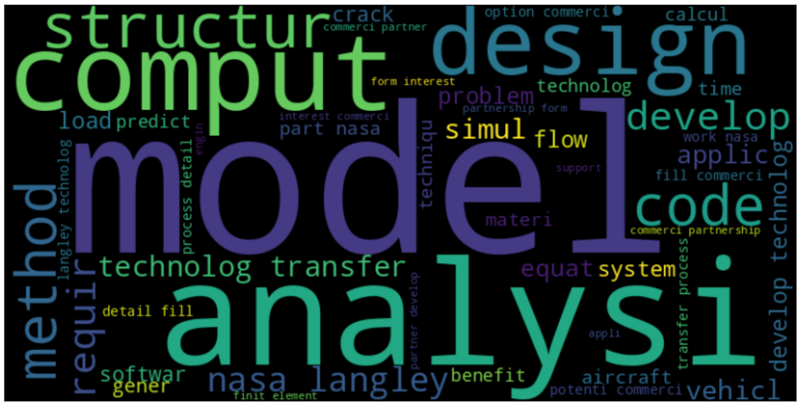


Figure 11: Tag Cloud of top 50 words

```python
stop_file = "english.stop"
stop_words = set()

with open(stop_file, 'r', encoding="latin-1") as file:
    stop_words = set(file.read().split())

filtered_tokens = []
for tokens in tokenize_all_files:
    filter_mid = []
    for token in tokens:
        if token.lower() not in stop_words:
            filter_mid.append(token)
    filtered_tokens.append(filter_mid)

filtered_stems = []
for tokens in filtered_tokens:
    mid = []
    for token in tokens:
        mid.append(stemmer.stem(token))
    filtered_stems.append(mid)
```

Task 6: Recomputation of tf and tf-idf for Vector and Boolean Models

In this task, Term Frequency *tf* and TF-IDF *tf-idf* are recomputed for each document after the removal of stop words. Subsequently, the top p stems in each document are conpared on both *tf* and *tf-idf*.

Comparing Top p Stems

After the removal of stop words, the comparisons are done again to gain insights into the extracted keywords. The main objective is to evaluate the differences based on two distinct keyword extraction criteria: Term Frequency *tf* and Term Frequency-Inverse Document Frequency *tf-idf*.

Intersection of tf and tf-idf Stems

The sets of top stems (after removal of stop words) obtained from *TF* and *TF-IDF* metrics are compared to identify shared and distinct significant terms. A Venn diagram illustrates their overlap, revealing common keyword selections.

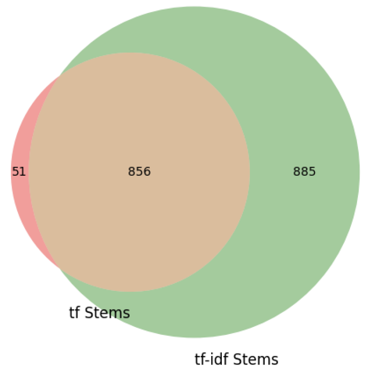


Figure 12: Intersection of tf and tf-idf Stems

Top Common Stems

The most common stems after the removal of stop words across all documents are extracted and visualized for both tf and tf-idf. The frequency of each stem and selecting the top results, the top common stems are displayed using bar charts.

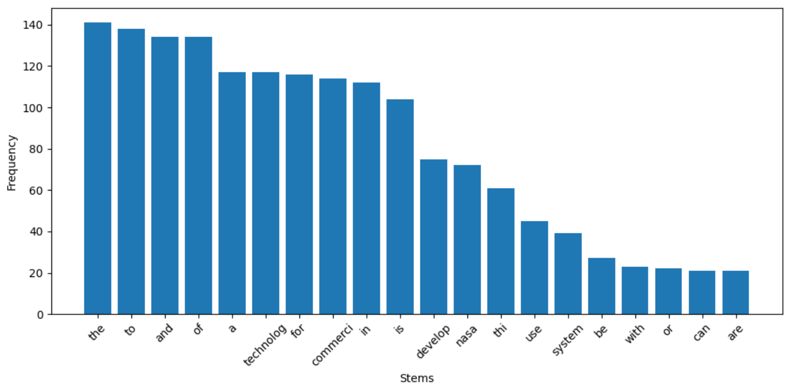


Figure 13: Top Common Stems - tf

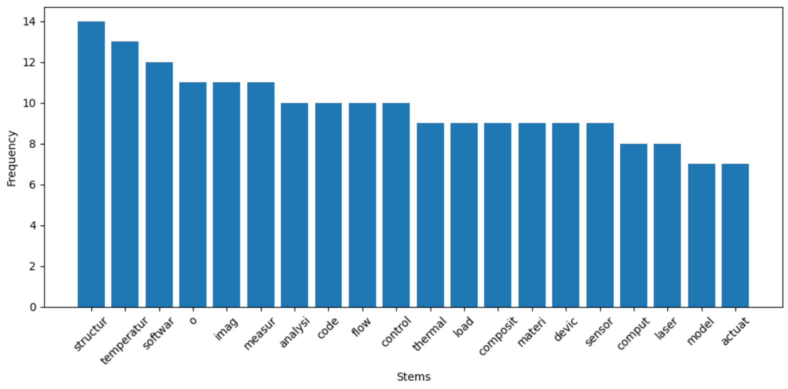


Figure 14: Top Common Stems - tf-idf

The top p stems after the removal are used to build Boolean and Vector Models, using the same methodology as described in **Task 4**. The comparison of rankings of these models after removal of stop words from the text, can be observed as follows.

Ranking Comparisons

The top 20 rankings produced by Vector and Boolean Models after removal of stop words from the top p stems are compared using two different comparisons: Venn Diagrams and Bar Plots. Venn Diagram signifies the intersection of results of both the models and Bar Plot signifies the frequency of the results of both the models.

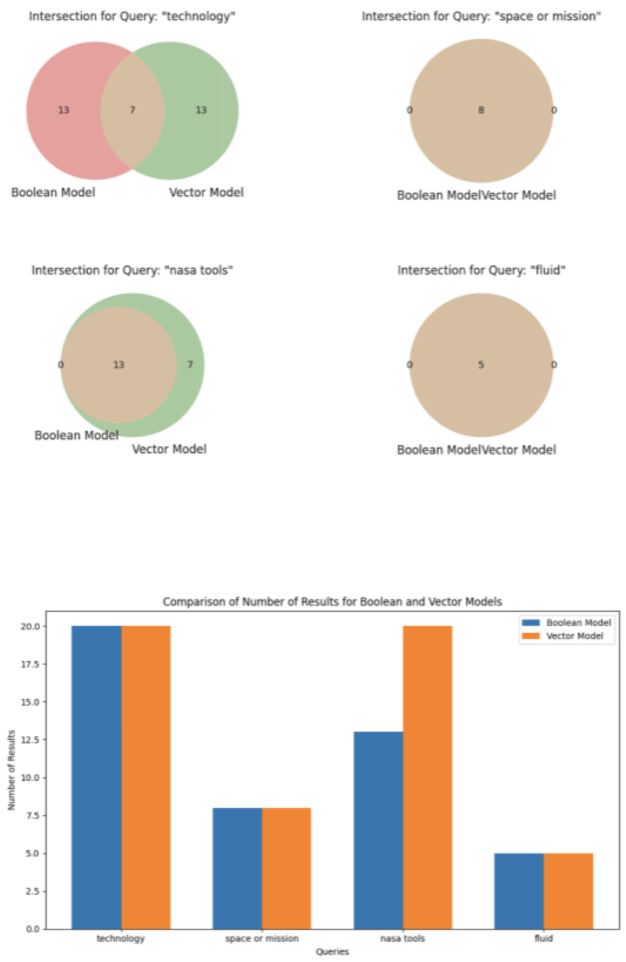
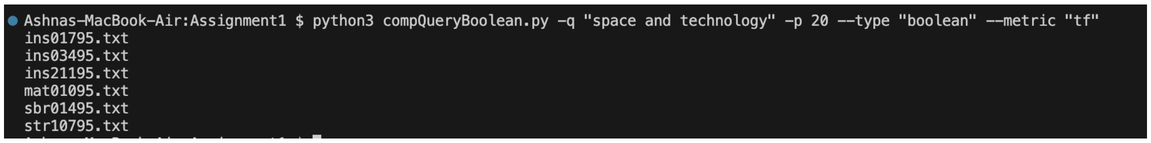


Figure 15: Ranking Comparison

Query Results

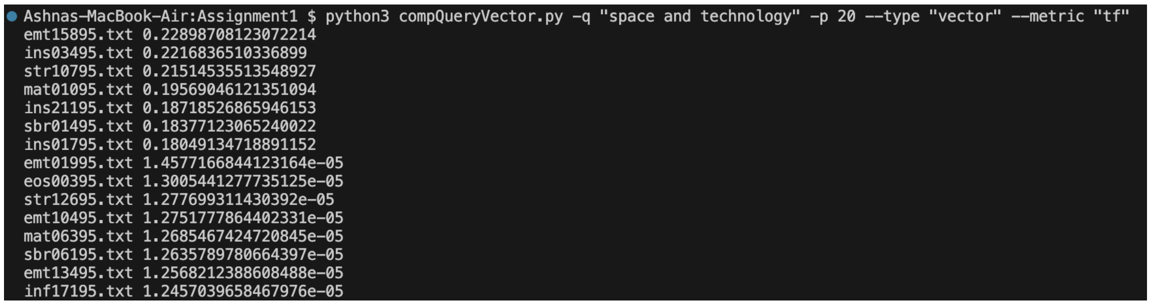
In this section, the query results of both the models is analyzed.

Query 1: space and technology



```
Ashnas-MacBook-Air:Assignment1 $ python3 compQueryBoolean.py -q "space and technology" -p 20 --type "boolean" --metric "tf"
ins01795.txt
ins03495.txt
ins21195.txt
mat01095.txt
sbr01495.txt
str10795.txt
```

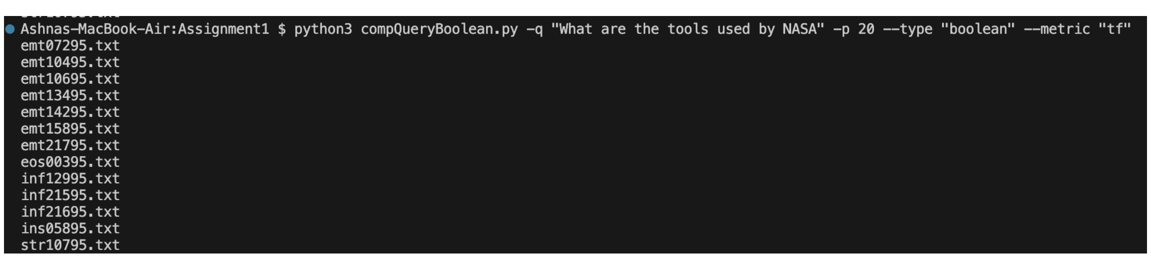
Figure 16: Boolean Model: space and technology



```
Ashnas-MacBook-Air:Assignment1 $ python3 compQueryVector.py -q "space and technology" -p 20 --type "vector" --metric "tf"
emt15895.txt 0.22898708123072214
ins03495.txt 0.2216836510336899
str10795.txt 0.21514535513548927
mat01095.txt 0.19569046121351094
ins21195.txt 0.18718526865946153
sbr01495.txt 0.18377123065240022
ins01795.txt 0.18049134718891152
emt01995.txt 1.4577166844123164e-05
eos00395.txt 1.3005441277735125e-05
str12695.txt 1.277699311430392e-05
emt10495.txt 1.2751777864402331e-05
mat06395.txt 1.2685467424720845e-05
sbr06195.txt 1.2635789780664397e-05
emt13495.txt 1.2568212388608488e-05
inf17195.txt 1.2457039658467976e-05
```

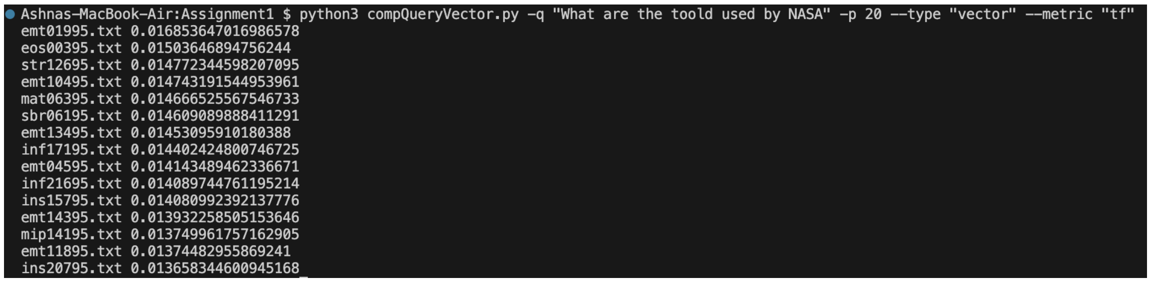
Figure 17: Vector Model: space and technology

Query 2: What are the tools used by NASA



```
Ashnas-MacBook-Air:Assignment1 $ python3 compQueryBoolean.py -q "What are the tools used by NASA" -p 20 --type "boolean" --metric "tf"
emt07295.txt
emt10495.txt
emt10695.txt
emt13495.txt
emt14295.txt
emt15895.txt
emt21795.txt
eos00395.txt
inf12995.txt
inf21595.txt
inf21695.txt
ins05895.txt
str10795.txt
```

Figure 18: Boolean Model: tools used by NASA



```
Ashnas-MacBook-Air:Assignment1 $ python3 compQueryVector.py -q "What are the toold used by NASA" -p 20 --type "vector" --metric "tf"
emt01995.txt 0.016853647016986578
eos00395.txt 0.01503646894756244
str12695.txt 0.014772344598207095
emt10495.txt 0.014743191544953961
mat06395.txt 0.014666525567546733
sbr06195.txt 0.014609089888411291
emt13495.txt 0.01453095910180388
inf17195.txt 0.014402424800746725
emt04595.txt 0.014143489462336671
inf21695.txt 0.014089744761195214
ins15795.txt 0.014080992392137776
emt14395.txt 0.013932258505153646
mip14195.txt 0.013749961757162905
emt11895.txt 0.01374482955869241
ins20795.txt 0.013658344600945168
```

Figure 19: Vector Model: tools used by NASA

Task 7: Clustering

In this task, similar NASA articles are grouped together based on the features extracted (*tf-idf matrix* to find structure within them. The clustering is done using Sklearn's KMeans Clustering using number of clusters = 15.

The first plot is a histogram that shows the distribution of documents across the clusters.

The second plot is a scatter plot created using t-SNE (t-Distributed Stochastic Neighbor Embedding), a technique for dimensionality reduction that is particularly well suited for the visualization of high-dimensional datasets. The x and y axes represent these two dimensions, and each point on the plot represents a document. The color of the points indicates which cluster they belong to.

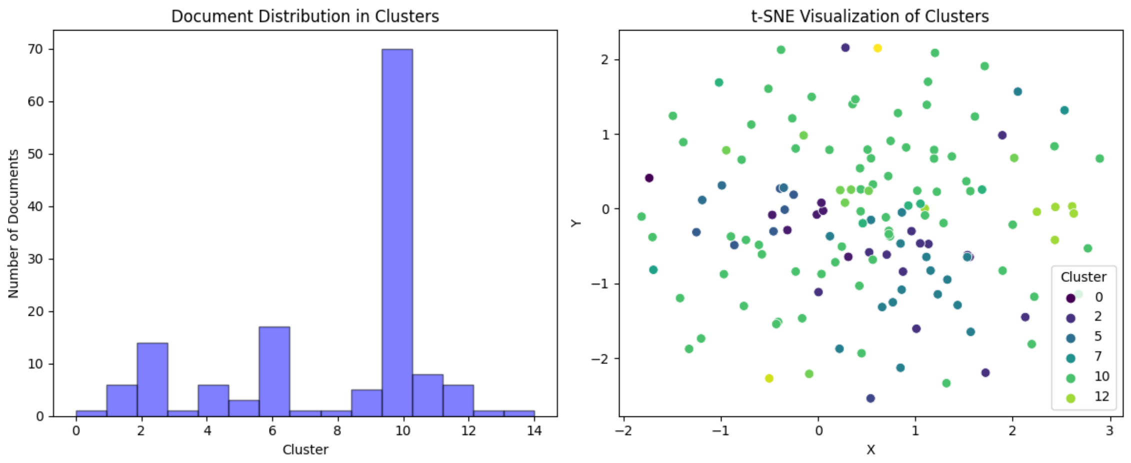


Figure 20: Clustering Plots

```python
n_clusters = 15
kmeans = KMeans(n_clusters=n_clusters, random_state=42)
kmeans.fit(vm.matrix.T)
cluster_labels = kmeans.labels_

plt.figure(figsize=(12, 5))
plt.subplot(121)  # 1 row, 2 columns, subplot 1
plt.hist(cluster_labels, bins=n_clusters, alpha=0.5, color='blue', edgecolor='black')
plt.xlabel('Cluster')
plt.ylabel('Number of Documents')
plt.title('Document Distribution in Clusters')

tsne = TSNE(n_components=2, random_state=42)
embedded_docs = tsne.fit_transform(vm.matrix.T)
embedded_df = pd.DataFrame({
    'X': embedded_docs[:, 0], 'Y': embedded_docs[:, 1],
    'Cluster': cluster_labels})

plt.subplot(122)
sns.scatterplot(data=embedded_df, x='X', y='Y',
                hue='Cluster', palette='viridis', s=50)
plt.title('t-SNE Visualization of Clusters')

plt.tight_layout()
plt.show()
```

Task 8: Conclusion

The Vector Model, which is constructed using the top stems after removing stop words, stands out as the superior approach for document retrieval tasks. It excels in accurately categorizing both relevant and non-relevant documents and goes a step further by ranking them based on their cosine similarity scores. This ranking capability allows users to identify the most relevant documents with greater precision, enhancing the overall effectiveness of the retrieval process.

On the other hand, the Boolean Model, while also employing the top stems with stop words filtered out, offers correct results up to a certain extent. However, it falls short in the crucial aspect of ranking these documents effectively. In this model, most of the relevant documents receive identical similarity scores, making it challenging to distinguish between them based solely on relevance. This limitation can potentially hinder the user's ability to prioritize and access the most pertinent information within the search results.