ASSIGNMENT 1

Giridhar S(MT21026) *Alka Kumari(MT21006)*

UNIGRAM INVERTED INDEX

- Implementation of a unigram inverted index after first pre-processing the files provided.
- Finding the number of documents matching the phase query provided by the user as input and number of comparison required to do so.

Dataset:

The above task is performed on the "Humor,Hist,Media,Food" dataset provided. The number of files present in the dataset is 1132.

Pre-processing Steps:

This is the first step which cleans the data in the dataset. This process includes operations to remove unnecessary data, handle missing data, normalizing the data and also to remove redundancy in the dataset. We are using NLTK library for data pre-processing.

1. Firstly, all the useful libraries were imported.
2. Then, the non-ASCII characters were removed from the data.
3. The text was converted into lowercase.
4. All the punctuations were removed from the text.
5. All the stopwords were removed from the text.
6. The text was tokenized using word_tokenizer returning a list of tokens.

All these preprocessing steps are done in both the dataset and the input query.

Methodology:

1. All the files were read at first.
2. All the file names were stored in a dataframe **df.**
3. All the files were then **preprocessed** and then iterating over each document, a Posting list was created for each unique term.
4. The inverted indexing is created in the form of a dictionary which is stored in **d.**
5. All the posting list corresponding to the term was made **unique and sorted** in ascending order of docId.
6. The lengths of the posting lists were stored in a list **fq**.
7. Then, the query and operations were taken as input and preprocessed.
8. The list **s1** contains the list of terms in the query and **s3** contains the list of operations.
9. For each term present in s1, we perform the operation in s3 and then copy back the result to that and we count the number of documents and comparisons.
10. The variable **c** stores the number of comparison, **l** stores the number of documents matched and **temp** stores the list of matched documents.

There are four Functions, one for each operation(OR, AND, AND NOT, OR NOT).

Assumptions:

- All the Non-ASCII characters were removed during preprocessing steps from the text as well as query.
- Words were converted into their base form.

POSITIONAL INDEX

- Pre-processing the files provided.
- Implementation of a positional index.
- Finding the number of documents matching the phase query provided by the user as input and number of comparison required to do so.

Dataset:

The above task is performed on the “Humor,Hist,Media,Food” dataset provided. The number of files present in the dataset is 1132.

Pre-processing Steps:

This is the first step which cleans the data in the dataset. This process includes operations to remove unnecessary data, handle missing data, normalizing the data and also to remove redundancy in the dataset. We are using NLTK library for data pre-processing.

1. Firstly, all the useful libraries were imported.
2. The text was converted into lowercase.
3. All the punctuations were removed from the text.
4. All the stopwords were removed from the text.
5. The text was tokenized using word_tokenizer returning a list of tokens.
6. All the blank space tokens were removed.

All these preprocessing steps are done in both the dataset and the input query.

Methodology:

1. All the files were read at first.

2. All the file names were stored in a list **doc10.**

3. All the files were then **preprocessed** and then iterating over each document, a Posting list was created for each unique term containing docIds and corresponding to which positions of those terms in the respective documents.
4. The positonal indexing is created in the form of a dictionary of list of lists which is stored in **d1.**

5. All the posting list corresponding to the term was already **sorted** in ascending order of docId as we iterated over docIds in ascending order.

6. Then, the query and operations were taken as input and preprocessed.

7. The list **s1** contains the list of terms in the phrase query.

8. Now, we find the documents where the phrase query is present by maintain the relative positions of each term in the query.

9. The list of documents retrieved is stored in name and is printed and the number of documents is also printed.

Assumptions:

- All the punctuations and stopwords and blank space tokens were removed during preprocessing steps from the text as well as query.
- Words were converted into their base form.