

IT458 Assignment 1

NAME: SUYASH CHINTAWAR

ROLL NO.: 191IT109

TOPIC: INVERTED INDEX
CONSTRUCTION

Note:

1) The colab link has been attached below. After opening the link, if it opens in drive, click on “Open with Google Colaboratory” to view the complete code.

Colab notebook link:

<https://colab.research.google.com/drive/1DfZwdFWNt-bljeGt744p8LJQdFIHhK2>

Q. Construct the inverted index representation for the corpus of any 100 articles from the Incredible India website across different categories (use full text from each article as one document).

PART 1: Generating text corpus

For generating the inverted index, we will first need the text corpus. Web scraping has been used to extract 100 articles from the website (<https://www.incredibleindia.org>) across different categories like heritage, adventure, art, etc.

The requests library in python has been used to extract the html files of the provided URLs and BeautifulSoup has been used to extract the text of each article from the html extracted.

Different categories are present under the ‘Immersive Experience’ section for each city. Each category has multiple texts that have been extracted. A common dynamic URL has been used to extract the texts is as shown below,

`'https://www.incredibleindia.org/content/incredible-india-v2/en/destinations/<city_name>.html'`

The <city_name> field is replaced with different cities of India and the respective links to their categories are extracted.

```
[3] root = 'https://www.incredibleindia.org/'
    cities = ['delhi', 'bengaluru', 'kolkata', 'varanasi', 'hyderabad', 'jaipur', 'udaipur']
    links = []

    for city in cities:
        city_page_url = 'https://www.incredibleindia.org/content/incredible-india-v2/en/destinations/' + city + '.html'
        page = requests.get(city_page_url)
        soup = BeautifulSoup(page.content, 'html.parser')

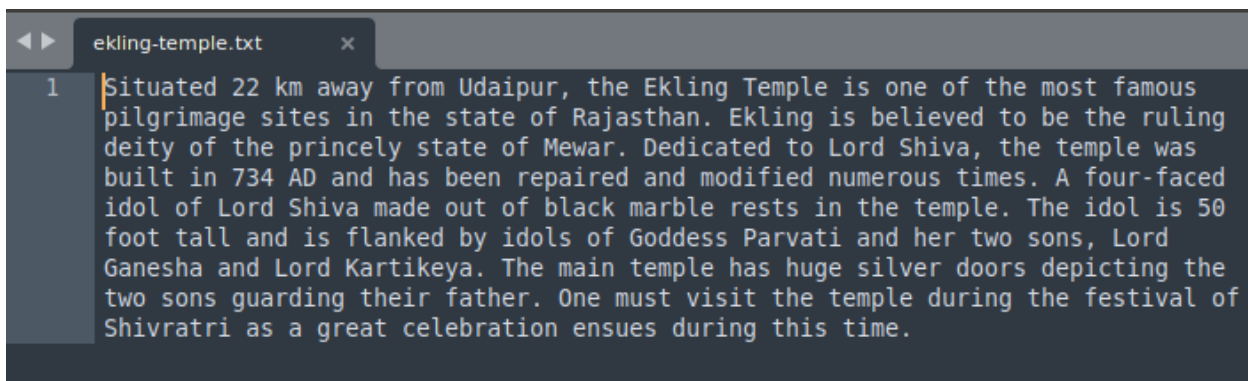
        s = soup.find_all('div', class_='immersive-experience-item')
        for category in s:
            link = category.find('a')['href']
            link = root + link
            links.append(link)
```

The code snippet above shows the links extracted for each available category for a particular city. The links extracted are shown below,

```
✓ [4] links
0s
[ 'https://www.incredibleindia.org/content/incredible-india-v2/en/destinations/delhi/heritage.html',
  'https://www.incredibleindia.org/content/incredible-india-v2/en/destinations/delhi/spiritual.html',
  'https://www.incredibleindia.org/content/incredible-india-v2/en/destinations/delhi/food-and-cuisine.html',
  'https://www.incredibleindia.org/content/incredible-india-v2/en/destinations/bengaluru/nature.html',
  'https://www.incredibleindia.org/content/incredible-india-v2/en/destinations/bengaluru/food-and-cuisine.html',
  'https://www.incredibleindia.org/content/incredible-india-v2/en/destinations/bengaluru/art-and-crafts.html',
  'https://www.incredibleindia.org/content/incredible-india-v2/en/destinations/kolkata/food-and-cuisine.html',
  'https://www.incredibleindia.org/content/incredible-india-v2/en/destinations/kolkata/nature.html',
  'https://www.incredibleindia.org/content/incredible-india-v2/en/destinations/kolkata/arts-and-crafts.html',
  'https://www.incredibleindia.org/content/incredible-india-v2/en/destinations/varanasi/food-and-cuisine.html',
  'https://www.incredibleindia.org/content/incredible-india-v2/en/destinations/varanasi/listicle/ghats-of-varanasi.html',
  'https://www.incredibleindia.org/content/incredible-india-v2/en/destinations/varanasi/spiritual.html',
  'https://www.incredibleindia.org/content/incredible-india-v2/en/destinations/hyderabad/food-and-cuisine.html',
  'https://www.incredibleindia.org/content/incredible-india-v2/en/destinations/hyderabad/art-and-crafts.html',
  'https://www.incredibleindia.org/content/incredible-india-v2/en/destinations/hyderabad/heritage.html',
  'https://www.incredibleindia.org/content/incredible-india-v2/en/destinations/jaipur/food-and-cuisine.html',
  'https://www.incredibleindia.org/content/incredible-india-v2/en/destinations/jaipur/art-and-crafts.html',
  'https://www.incredibleindia.org/content/incredible-india-v2/en/destinations/jaipur/heritage.html',
  'https://www.incredibleindia.org/content/incredible-india-v2/en/destinations/udaipur/spiritual.html',
  'https://www.incredibleindia.org/content/incredible-india-v2/en/destinations/udaipur/heritage.html',
  'https://www.incredibleindia.org/content/incredible-india-v2/en/destinations/udaipur/food-and-cuisine.html' ]
```

Each of the links in the above image consist of multiple documents. A set of 151 documents was extracted from the set of the links in the image. Of these 100 were chosen for the further preparation of inverted index as well as for preprocessing.

A sample text document extracted is shown below,



```
ekling-temple.txt
1 Situated 22 km away from Udaipur, the Ekling Temple is one of the most famous pilgrimage sites in the state of Rajasthan. Ekling is believed to be the ruling deity of the princely state of Mewar. Dedicated to Lord Shiva, the temple was built in 734 AD and has been repaired and modified numerous times. A four-faced idol of Lord Shiva made out of black marble rests in the temple. The idol is 50 foot tall and is flanked by idols of Goddess Parvati and her two sons, Lord Ganesha and Lord Kartikeya. The main temple has huge silver doors depicting the two sons guarding their father. One must visit the temple during the festival of Shivratri as a great celebration ensues during this time.
```

Its corresponding text on the website is shown below to ensure that all of the available text of the document has been scraped successfully.

[Home](#) / [Destinations](#) / [Udaipur](#) / Ekling Temple

Situated 22 km away from Udaipur, the Ekling Temple is one of the most famous pilgrimage sites in the state of Rajasthan. Ekling is believed to be the ruling deity of the princely state of Mewar. Dedicated to Lord Shiva, the temple was built in 734 AD and has been repaired and modified numerous times. A four-faced idol of Lord Shiva made out of black marble rests in the temple. The idol is 50 foot tall and is flanked by idols of Goddess Parvati and her two sons, Lord Ganesha and Lord Kartikeya. The main temple has huge silver doors depicting the two sons guarding their father. One must visit the temple during the festival of Shivratri as a great celebration ensues during this time.

(Source:

<https://www.incredibleindia.org/content/incredible-india-v2/en/destinations/udaipur/ekling-temple.html>)

Q. Observe and report the effect of different preprocessing techniques applied to the corpus and the changes to the vocabulary size w.r.t the final index terms, when compared to the number of initial tokens.

PART 2: Preprocessing texts

Four different types of preprocessing techniques have been used in this case. The four techniques are,

- a) Tokenization: Here, the raw text of each document is converted into a set of tokens. This is achieved using simple splitting of the raw text with respect to white spaces. The code used to fulfill this purpose is shown below,

```
[9] def tokenize(text):  
    return text.split()
```

- b) Normalization: It is the process in which all the terms inside the text are brought to the same form.

```
[10] def normalize(text):  
    new_tokens = []  
    for token in text:  
        token = re.sub('[^a-zA-Z0-9]', '', token)  
        token = token.lower()  
        new_tokens.append(token)  
    return new_tokens
```

This is done by converting all the tokens to lower case letters and removing all non-word, non-numeric characters from the texts. Regular expressions have been used to replace the unwanted characters from the text.

- c) Lemmatization: Lemmatization is a technique in which the words are converted into their base forms. The WordNet lemmatizer is used to achieve this purpose.

```

✓ [11] from nltk.stem import WordNetLemmatizer
0s

def lemmatize(text):
    lemmatized_text = []
    lemmatizer = WordNetLemmatizer()
    for token in text:
        lemmatized_token = lemmatizer.lemmatize(token)
        lemmatized_text.append(lemmatized_token)
    return lemmatized_text

```

- d) Stopword removal: Finally the last step consists of removing the stop words present in the text. The NLTK stopwords corpus is used in this context.

```

✓ [12] from nltk.corpus import stopwords
0s

def remove_stopwords(text):
    stopwords_list = stopwords.words('english')
    new_tokens = []
    for token in text:
        if token not in stopwords_list:
            new_tokens.append(token)
    return new_tokens

```

The effect of these preprocessing techniques on the set of indexed terms in the corpus is shown below,

```

Tokenized texts:
[['Situaded', '22', 'km', 'away', 'from', 'Udaipur', 'the', 'Ekling', 'Temple', 'is', 'one', 'of', 'the', 'most', 'famous', 'pilgrimage', 'sites', 'in', 'the', 'state',
Number of tokens after tokenization: 3844

Normalized texts:
[['situated', '22', 'km', 'away', 'from', 'udaipur', 'the', 'ekling', 'temple', 'is', 'one', 'of', 'the', 'most', 'famous', 'pilgrimage', 'sites', 'in', 'the', 'state',
Number of tokens after normalization: 3062

Lemmatized texts:
[['situated', '22', 'km', 'away', 'from', 'udaipur', 'the', 'ekling', 'temple', 'is', 'one', 'of', 'the', 'most', 'famous', 'pilgrimage', 'site', 'in', 'the', 'state',
Number of tokens after lemmatization: 2840

Texts after stopword removal:
[['situated', '22', 'km', 'away', 'udaipur', 'ekling', 'temple', 'one', 'famous', 'pilgrimage', 'site', 'state', 'rajasthan', 'ekling', 'believed', 'ruling', 'deity', 'p
Number of tokens after lemmatization: 2744

```

We can see that the number of indexed terms after each step decreases as,

- Tokenization - 3844
- Normalization - 3062
- Lemmatization - 2840
- Stopword removal - 2744

The corresponding changes in the tokens can also be seen in the image above.

PART 3: Inverted index construction

Two types of data structures have been implemented in the construction of inverted index. One is a hashmap of sets to store the postings of each index term and the other is a hashmap of list postings.

To generate an inverted index using sets, the occurrence of each token in each document is recorded and stored in a set. The generated postings for some of the index terms is shown below,

```
'festival': {0, 5, 28, 73, 80, 94, 105, 126},  
'shivratni': {0},  
'great': {0, 21, 23, 28, 29, 36, 40, 76, 88, 101, 126},  
'celebration': {0, 141},  
'ensues': {0},  
'1906': {1},  
'national': {1, 22, 25, 69, 176},
```

Every index term has a set associated to it consisting of the document ID it's present in.

Q. What type of data structure might be most optimal for storing the index? Compare and provide a detailed analysis w.r.t different data structure choices and give the cost analysis from storage and retrieval/insertion/update/deletion perspectives.

The most optimal data structure for storing the inverted index is a hashmap of sets. Hashmap to store the index terms and sets to store the corresponding documents the index terms appear in.

The time and space complexities of each of the two implemented methods is documented below,

1) Using hashmap of sets:

Assume that there are 'n' posting lists and the maximum size of the posting list can be 'm'. Then the space and time complexities are as follows,

Space Complexity: $O(n*m)$ to store all posting lists of all index terms.

Time Complexities:

a) Retrieval: The time required to retrieve a posting list given an index term is $O(1)$ as hashmap access takes $O(1)$ time.

b) Insertion: The time required to insert 'k' already existing index terms of a document is $O(k)$ as it takes $O(1)$ time to insert an ID into one posting list of an index term.

c) Deletion: If we delete a document having 'k' index terms, the document ID in all the posting lists of the index terms must be deleted. Hence, it takes $O(k)$ time to do so as searching for the index term and deleting one element from its posting list takes $O(1)$ time.

d) Updation: $O(k)$ time complexity as updation is a mix of insert and delete operations.

2) Using hashmap of lists:

Space Complexity: $O(n*m)$ to store all posting lists of all index terms.

Time Complexities:

a) Retrieval: The time required to retrieve a posting list given an index term is $O(1)$ as hashmap access takes $O(1)$ time.

b) Insertion: The time required to insert 'k' already existing index terms of a document is $O(k*m)$ as it takes $O(m)$ time to insert an ID into one posting list of an index term.

c) Deletion: If we delete a document having 'k' index terms, the document ID in all the posting lists of the index terms must be deleted. Hence, it takes $O(k*m)$ time to do so as searching for the index term and deleting one element from its posting list takes $O(1)$ time.

d) Updation: $O(k*m)$ time complexity as updation is a mix of insert and delete operations.

Q. Using the constructed inverted index, perform some sample boolean queries of the pattern shown below. What is the time complexity of finding the result assuming that there are 'n' postings lists in the inverted index. Analyze and explain in detail.

Assumption: The most optimal data structure has been considered here for storing the inverted index, i.e. hashmap of sets.

1) term1 AND term2 AND term3

Time complexity: $O(m)$ where m is the size of the longest posting list.

Explanation: Fetching the posting lists of the three terms will take $3 \cdot O(1)$ time and doing the 'AND' operation will take $O(m)$ time assuming ' m ' is the size of the longest postings list.

Examples:

```
Query: life AND country AND war
Query result:
Set of document IDs: {24, 26}
Name of corresponding documents: ['national-war-memorial.txt', 'india-gate.txt']
```

```
Query: saffron AND word AND mean
Query result:
Set of document IDs: {25}
Name of corresponding documents: ['sheer-korma.txt']
```

The results can be verified from their postings lists,

```
'life': {24, 26, 51, 62, 97},
'country': {24, 26, 29, 41, 48, 62, 64, 68, 92},
'war': {24, 26},
```

and

```
'saffron': {25, 44, 68},
'word': {25, 75, 79, 82, 97},
'mean': {25, 31, 79, 97},
```

2) term1 OR term2 AND NOT term3

Time complexity: $O(m)$ where m is the size of the longest posting list.

Explanation: Fetching the posting lists of the three terms will take $3 \cdot O(1)$ time and doing the 'AND' operation will take $O(m)$ time assuming ' m ' is the size of the longest postings list. Moreover, the 'NOT' operation can take at most $O(m)$ time if the postings list is almost empty (1 element).

Examples:

Query: life OR country AND NOT war

Query result:

Set of document IDs: {64, 97, 68, 41, 48, 51, 24, 26, 92, 29, 62}

Name of corresponding documents: ['jute-decor.txt', 'kantha.txt', 'hyderabadi-biryani.txt', 'cubbon-park.txt', 'terracotta.txt', 'golconda-fort.txt', 'national-war-memori

Query: saffron OR word AND NOT mean

Query result:

Set of document IDs: {82, 68, 25, 75, 44}

Name of corresponding documents: ['nathdwara.txt', 'hyderabadi-biryani.txt', 'sheer-korma.txt', 'sankat-mochan-mandir.txt', 'mutton-biryani.txt']

The results can again be verified from the postings list of the same index terms given above.

Note: Please refer to the attached code in moodle or the link the colab notebook is also provided in the beginning of this document.

THANK YOU