Information Retrieval System for image search using the BM25 model

Danyal Quazi
danyal.quazi2@mail.dcu.ie
School of Computing, Dublin City University
Dublin, Ireland

ABSTRACT

A prototype of an Image Search Engine with all the necessary steps like data collection, crawling through websites, annotation, creation of a data frame, text pre-processing, creation of inverted indexing, text similarity model implementation(BM25), testing the system using queries, and the creation of a GUI. A python library called Beautiful soup is used to collect images from a website called Unsplash. A dataset of around 1000 images is created which contains 17 animal tags, approximately 60 images of each animal. Pandas data frames are used to create various structures required for the project like the image data frame with image information, the IDF, and inverted index. The images are annotated by extracting some extra information about images during the process of website crawling. The system can accept a query of multiple words and display various images related to that query. This is done by comparing the description of the images with the query.

CCS CONCEPTS

• **Computing methodologies** → **Information extraction**; • **Information systems** → **Query representation**; **Information extraction**; **Probabilistic retrieval models**; **Similarity measures**; **Relevance assessment**; **Presentation of retrieval results**; **Retrieval efficiency**; **Search interfaces**; **Image search**.

KEYWORDS

Information Retrieval,Web crawling, Beautiful Soup, Image search, Anvil, Text pre-processing, (TF), Inverse Document Frequency(IDF), TF-IDF, BM25, Indexing

ACM Reference Format:
Danyal Quazi. 2022. Information Retrieval System for image search using the BM25 model. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The job of an information retrieval system is to simply find relevant information from a large collection. The collection is usually in the form of unstructured or semistructured data which can be text, images, videos, etc. These systems are very important in an era where unimaginable amounts of data are generated every second. The rise of social media websites has boosted the generation of image data. In every industry, the need for an efficient system capable of finding relevant images for a given query keeps increasing. This project is a continuation of another project which was an attempt to implement an Information Retrieval system that used ranking models like Vector Space Model, BM25, and Multinomial Language Model. The models were compared to conclude that BM25 delivered the highest scores. The same BM25 model is employed in this project for text matching. In this project, a dataset is created containing about 1000 images with their tag and a description. The job of the system is to rank these images based on the text description of the images for a given query. Figure 1 shows the basic architecture of the system which is discussed in detail in further sections.

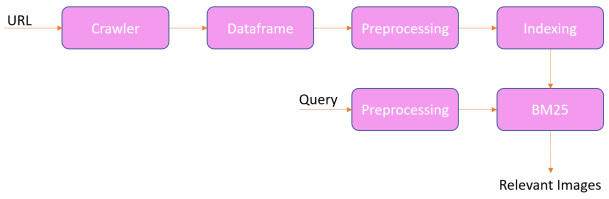

Figure 1: Architecture of the system.

As seen in the architecture, the system is divided into smaller steps. Which include: Collection of data using a web crawler, Annotation of images, Creation of data frame and preprocessing, Indexing, and finally ranking of the images using BM25 for a given query.The Jupyter notebook containing the code for the system will be directly connected to Anvil[1] where a simple UI is created where a user can enter a query and some images that are similar to the query according to the model will be displayed from the collection.

2 COLLECTION OF DATA USING A WEB CRAWLER

The very first step of the project is the collection of data. For this project, we have collected 1099 images of different animals (about 17 categories) the list of the animals is: animals = ['tiger', 'cat', 'dog', 'horse', 'elephant', 'rhino', 'monkey', 'deer', 'kangaroo', 'bear', 'giraffe', 'koala', 'Cow', 'frog', 'crocodile', 'penguin', 'squirrel']. A loop was iterated on this list to search for the list element which in this case is an animal. About 66 pictures are collected for each animal.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

The data is collected from a website called Unsplash. Unsplash grants you an irrevocable, nonexclusive, worldwide copyright license to download, copy, modify, distribute, perform, and use photos from Unsplash for free, including for commercial purposes, without permission from or attributing to the photographer or Unsplash.[4]

Once we have the complete URL by simply concatenating the name of the animal from the list, using Beautiful Soup we can extract the contents of the webpage. The most important element for us is the image source which contains the URL of the image, from which we can download the image. Some other information is also extracted from the webpage for labeling the image or its description which is discussed in a later section.

Beautiful Soup is one of the easiest and fastest ways of crawling through websites, the syntax is straightforward and easy to understand. Beautiful Soup is a Python library that extracts, analyzes, and edits data from the DOM tree of webpages. It integrates with your preferred parser to offer fluent navigation, search, and modification of the parse tree. It saves hours or even days of effort for programmers. Beautiful Soup will examine any document that is supplied to it, including HTML/XML and plain text documents. Beautiful Soup will only construct a relevant DOM tree and make a sequence of API calls to acquire the specified data if the HTML and XML documents meet the standards. It also has a lot of cross-platform versatility.[2][3]

In the next figure, we can see the list element used to search which in this case is the tiger. When an image is inspected to see its code we can easily find the tag which contains the src attribute containing the image URL. All the URLs usually start with the same text and hence can be used to select a part of the soup generated by the Beautiful Soup to create a list of image URLs.

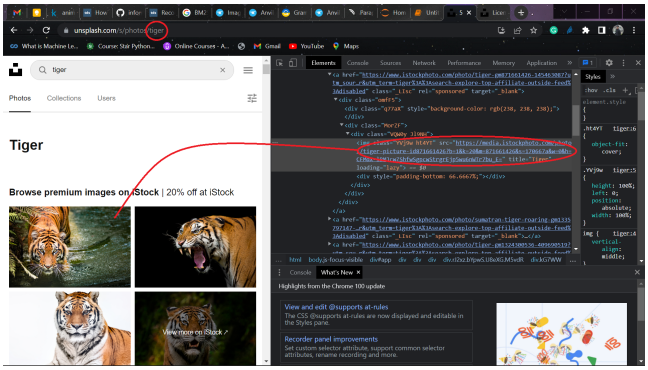

Figure 2: A screenshot of the website.

Once we have a list containing the source URLs of the images, we can simply run a loop and download the images using this simple code shown in the next figure.

```
In [ ]: os.mkdir('photos')

In [190]: for i in df.index:
              img_data = rq.get(df.loc[i]['links']).content
              with open(df.loc[i]['location'], 'wb+') as f:
                  f.write(img_data)
          f.close()
```

Figure 3: Code used to download images.

3 ANNOTATION OF IMAGES, CREATION OF DATA FRAME, AND PREPROCESSING

3.1 Annotation of images

There is no way we can compare a query directly to an image because these two data are completely different. In order to compare the image to the query, we need some sort of textual representation of the images. In order to get this extra information about images, we use the alt attribute from the HTML code extracted from the website. For the website we have crawled, a title or alt is present which contains a short description of the image as shown in the next figure.

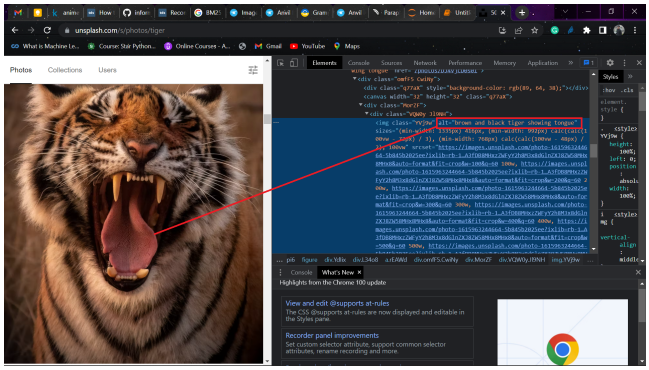

Figure 4: The description if image.

This alt attribute can be used as the representation of the images and can be compared to the query. The soup wasn't recognizing the alt hence was unable to extract directly like the image link. There are a few extra steps involved to extract the title of the image shown in Figure 4. A new list is created in which the soup file for each image is converted to a normal python string. Now the description or the alt part of the image is nothing but just a substring of this string which contains the entire HTML content. This code snippet shown in the next figure uses the regular expression's search function to extract the part we are interested in. The resulting string still contains an extra set of quotes which is easily removed using python string slicing.

```
In [117]: title = []

          for i in imgs:
              title.append(re.search('<img alt="(.+?)"',i))

In [118]: len(title)

Out[118]: 1099

In [119]: # title[2].group()
          for i in range (0,len(title)-1):
              if title[i] == None:
                  title[i] = none_type_clean(title[i])
                  continue
              title[i] = title[i].group()
              title[i] = title[i][10:-1]
```

Figure 5: Code for extracting the image description.

Another list is generated containing the category of the animal present in the image which can be directly used as the label and can later be added one or more times to the text to increase the weight of that animal word.

3.2 Creation of a data frame

The previous assignment was all about creating an IR system without the use or help of any external python library and hence a data structure was generated with the help of python dictionaries. In this project, a pandas data frame[5] is used for the creation of a dataset that can be used for further processes.

While crawling through the website, a list is created which contains the links to the images. As mentioned earlier, another list is created which contains the descriptions for the images extracted from the alt part of the image tag. There is a third list that contains the label, tag, or category animal. A fourth list is created in which animal category is concatenated with a unique number this list can be treated as the unique id for each image. A fifth list contains the location of the image in the local system. All these lists are used as different columns to create a final data frame shown in the next Figure.

```
In [156]: df.head(10)
Out[156]:
```

	links	tag	text	location	id
0	https://images.unsplash.com/photo-1561731216-c...	tiger	Bengal tiger	photos//tiger1.jpg	tiger1
1	https://images.unsplash.com/photo-1551972251-1...	tiger	close-up photography of tiger	photos//tiger2.jpg	tiger2
2	https://images.unsplash.com/photo-161582499619...	tiger	tiger on brown tree log	photos//tiger3.jpg	tiger3
3	https://images.unsplash.com/photo-160249145363...	tiger		photos//tiger4.jpg	tiger4
4	https://images.unsplash.com/photo-159182443870...	tiger	brown tiger walking on brown sand during daytime	photos//tiger5.jpg	tiger5
5	https://images.unsplash.com/photo-153761693034...	tiger	tiger in forest	photos//tiger6.jpg	tiger6
6	https://images.unsplash.com/photo-160261263946...	tiger	tiger lying on ground near green leaves during...	photos//tiger7.jpg	tiger7
7	https://images.unsplash.com/photo-159076741306...	tiger	brown and white tiger on black background	photos//tiger8.jpg	tiger8
8	https://images.unsplash.com/photo-156760191666...	tiger	orange tiger	photos//tiger9.jpg	tiger9
9	https://images.unsplash.com/photo-160509267692...	tiger	brown and black tiger in close up photography	photos//tiger10.jpg	tiger10

Figure 6: Data frame.

3.3 pre-processing

Text pre-processing is one of the most crucial steps in making an information retrieval system. It can greatly influence the results and accuracy of the IR system. It converts the specified text into manageable text. Under preprocessing, all undesired and unnecessary info is removed from the text. Natural language processing (NLP) makes a bigger difference in text preparation.[6]

The pre-processing section or steps are the same as those done in the previous IR system. The pre-processing steps are as follows:

3.3.1 Converting the entire text to lower case. The documents contain lowercase as well as uppercase English letters. Many of the same words will be considered as different by the system if they are present in different forms. The text needs to be made consistent throughout the data frame. A simple python inbuilt function .lower() is used for the conversion of the text.

3.3.2 POS tagging and Tokenization. The process of POS tagging is labeling every single word in the text with the part of speech it belongs to. This information can later be used in Lemmatization. But the POS tags are created for each word in the text therefore the words need to be separated to form tokens. This process of splitting the words is called tokenization. Token creation is important as it can help us compare the words and use them as an element to see their frequency, importance, etc. To implement these two important steps we have used Natural Language Toolkit (NLTK). NLTK provides several functions which can make pre-processing much easier. nltk.word_tokenize(text) function is used for token creation. These tokens are then given to the get_wordnet_pos(w) function which will create tuples in this list of tokens. These tuples will contain the word or the token with its POS tag.

3.3.3 Lemmatization. Multiple words convey the same meaning and if these words are present in a query and an image description (even in their different forms), the chances of that document being relevant to the query are very high. But the system will consider both these words to be different resulting in the document being non-relevant. To handle such a problem, a method is needed to convert all the forms of a word to one form which can be consistent throughout the corpus and will also not affect the meaning or the context.

We can achieve this by replacing or converting the words to their root form and this process of converting the words to their root form is called Lemmatization. For example, the word normalising will be converted to its root form normalize.The text is already converted into a list of tokens with their POS tag. These tuples can be used to lemmatize these words using an NLTK function called wordnet_lemmatizer.lemmatize(). This function will also remove the POS tags after the lemmatization as we do not need them further. The previous image shows the words converted to their root form.

3.3.4 Stop Words removal. There are still some words present that do not contribute a lot to the context or meaning of a document. For instance, words like "a," "who" and "be" will be less significant than the words like "tiger" and "nature". It is better to remove these words from the corpus. A collection of stop words, nltk.corpus.stopwords.words("english") from NLTK is used to compare and remove these words.

The following figure shows the text after cleaning:

```
In [232]: df.head()
Out[232]:
```

id	links	tag	text	location
tiger1	https://images.unsplash.com/photo-1561731216-c...	tiger	[bengal, tiger]	photos//tiger1.jpg
tiger2	https://images.unsplash.com/photo-1551972251-1...	tiger	[close-up, photography, tiger]	photos//tiger2.jpg
tiger3	https://images.unsplash.com/photo-161582499619...	tiger	[tiger, brown, tree, log]	photos//tiger3.jpg
tiger4	https://images.unsplash.com/photo-160249145363...	tiger	[]	photos//tiger4.jpg
tiger5	https://images.unsplash.com/photo-159182443870...	tiger	[brown, tiger, walk, brown, sand, daytime]	photos//tiger5.jpg

Figure 7: Text after pre-processing.

4 INDEXING

An inverted index is a data structure that contains a word (or atomic search item) with the set of documents (or indexed units) that include that term and the postings. A posting is simply a binary value of the occurrence of that word in a document or list of documents or in our case, image descriptions, or it may contain a piece of additional information indicating the frequency of that term in that particular document. An inverted index is accessed using only one key, i.e. the word. Efficient access usually implies that the index is sorted or organized as a hash table[7].

This section shows the creation of an inverted index. As we are using the same IR system built for the last assignment as the foundation of this image search engine, a similar inverted index is created. But in this project, we have the freedom of using pandas, we can make the process easier and less time-consuming compared

to the dictionary implementation which took almost 6 hrs for the execution.

```
In [128]: inverted_index.head()
```

Out[128]:

	word	doc	frequency
0	bengal	[tiger1, tiger21, tiger47]	[1, 1, 1]
1	tiger	[tiger1, tiger2, tiger3, tiger4, tiger5, tiger...	[1, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, ...
2	close-up	[tiger2, tiger33, tiger50, squirrel1026, squir...	[1, 1, 1, 1, 1, 1]
3	photography	[tiger2, tiger10, tiger12, tiger13, tiger25, t...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
4	brown	[tiger3, tiger5, tiger8, tiger10, tiger11, tig...	[1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...

Figure 8: The inverted index.

As seen in the figure 8, there are three columns in the data frame, The word, The list of documents or images in which the word is present, and The frequency of that word in that document.

5 SEARCH AND RANKING

This is the section of the project which will be comparing the queries with the image descriptions. The job of the created model is to compare the query to the descriptions of all the images and assign a score for all the images depending on the relevance of that image to the query. In this project, we have implemented the same BM25 system created in the first assignment using nothing but pure python code without any external text similarity tool.

5.1 BM25 Model

BM25 stands for best match 25 is a very popular ranking model. The formula for BM25 contains two constant variables "b" and "k". The range of k is between 1.2 - 2 and for b is 0.5 - 0.8. BM25 uses the concepts like TF which we have already calculated and IDF which is calculated using the code shown in figure 9. An additional value is the document length normalization in which the constant k will be regulating the effect of tf on the score. The last bit we need for the calculation is the average length of a document. The figure 10 shows a simple code written to find the average length. The average length of a description in the corpus is 7 words.

```
In [133]: import numpy as np

          idf=pd.DataFrame()
          i = []

          for word in vocab:
              i.append(np.log2(len(df)/len(inverted_index.loc[word]['doc'])))

In [134]: idf['word'] = vocab
          idf['idf'] = i

In [135]: idf.head()
```

Out[135]:

	word	idf
0	bengal	8.517013
1	tiger	4.057582
2	close-up	7.517013
3	photography	3.709658
4	brown	1.234697

Figure 9: Code for the IDF.

```
In [138]: addition = 0
          for doc in df['text']:
              addition += len(doc)
          avg_l = addition/len(df)

In [139]: avg_l
```

Out[139]: 7.110100090991811

Figure 10: Code for the average length.

5.2 Ranking and UI

Once the scores are assigned, the list of the image ids is sorted in a reverse order to get the images with the highest scores which according to the BM25 are the most relevant images to the given query. As mentioned earlier, more weight can be added to the image category by simply appending the animal category to the description text this will also help with the images with no description. A function is created which will return a list that contains the source links of the most relevant images for a query.

The UI of the system is designed using Anvil which is a very useful tool for converting a python code or a jupyter notebook into a responsive web app. In the simplest terms, the front-end of the website can be directly designed using Anvil's simple drag and drop tools. Using Anvil, the jupyter notebook can be considered as a server or back-end of the website. The query entered in the text box will be given as the parameter to a function on the jupyter notebook side which will return a list containing the links for the relevant images for the query. Once we have the source links for the images, we can easily present the images using a grid tool. The URL for the anvil app can also be shared as long as the jupyter notebook kernel is running.

```
def button_1_click(self, **event_args):
  """This method is called when the button is clicked"""
  query = self.text_box.text
  link_list = []
  link_list = anvil.server.call("get_sem", query)
  #self.label_1.text = link_list

  #self.image_4.source, self.image_9.source, self.image_7.source, self.image_8.sour
  for i in link_list:
    m = URLMedia(i)
    self.grid_panel_1.add_component(Image(source=m),col_xs=0, width_xs=4,row='A')
```

Figure 11: The anvil side of the code calling a function defined in the jupyter notebook.

```
In [193]: import anvil.media

          @anvil.server.callable
          def get_sem(text):
              return sim(text)
```

Figure 12: Function in jupyter notebook which can be called from anvil.

6 EVALUATION AND RESULTS

The system can search images for all 17 categories easily with high accuracy. The system can also search the queries with some additional information like 'Tiger tongue', 'horse running', etc. Multiple queries were tested multiple times and the results were good. There are many repeated images, the reason seems to be the dataset or repeated images on the website which was crawled. As the length of the query and the image description is very less(hardly

a sentence), the scores for the relevant images from BM25 are almost the same. The reason is, that when the stopwords are removed from the text or the query, the remaining text or the set of tokens is a very small vocabulary with actual meaningful words. The increase in the size of the data used can improve this problem.

Three queries that returned good results are:

"Close up picture of a tiger": Returned a bunch of close up images of tigers from the collection.

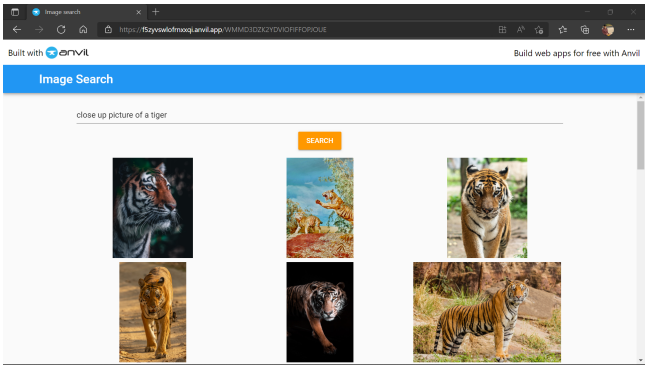

Figure 13: Results for the query 'Close up picture of a tiger'.

"Two Polar bears": This query surprisingly returned a picture of two polar bears and a picture with two bears. The query performed well as there is no such animal tag as 'Polar bear'. The system was able to get this image from the bear category by adding two additional parameters 'polar' and 'two' and this image was given the highest preference.

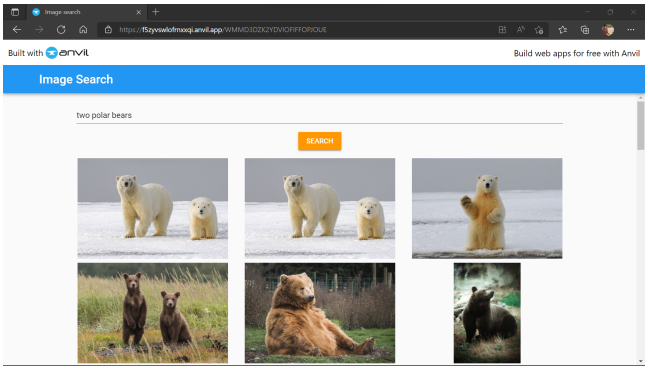

Figure 14: Results for the query 'Two Polar bears'

'frog on a leaf or water': This query performed well as it returned some images of a frog on a leaf and some with frogs in the water. Another reason this query performed well could be, that most of the images of frogs will have these two elements. But still, the system presented these images on the top shows that it is performing well.

7 CONCLUSION

An Information retrieval system was implemented for image search using the BM25 model created in the previous project. The system accepts a query related to animals(17 animal tags) and presents a

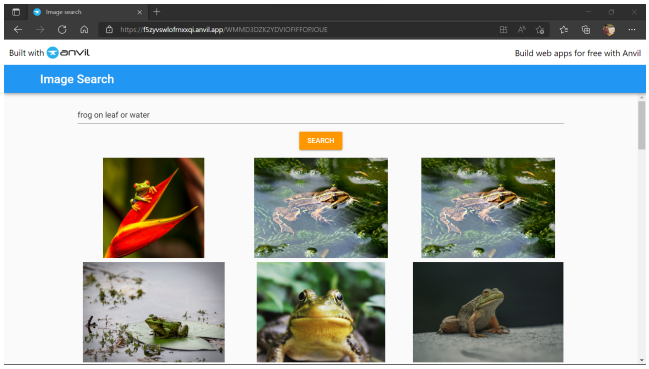

Figure 15: Results for the query 'frog on a leaf or water'

bunch of related images. The system can accept a query of more than one word and still return decent results. The challenging part was the extraction of additional information for the images from the alt attribute of an image tag. The soup file generated by beautiful soup could not identify the alt part which contained the information of the images which is the most important part for the text similarity. The description was hence extracted manually using regular expression. A significant improvement is seen in execution time when compared to the previous assignment. The reasons could be: that pandas data frames are more efficient compared to basic python dictionaries. The collection size is significantly smaller.

The collection size can be increased to improve the accuracy and vocabulary size. A more careful information crawling process can reduce the issue of repeated images. The UI can be greatly improved by adding multiple functionalities and better error or exception handling capabilities. A better preprocessing can improve the results for queries such as "close up" as the stop word removing function removes the word up. The boolean model can be used to implement NOT to eliminate the undesired results.

The system is far away from being a robust one. There is a lot of space for improvement. The greatest achievement is a better understanding of the IR concepts and knowledge of factors that can influence the working of an image search engine.

REFERENCES

[1] https://anvil.works/docs/overview
[2] Beautiful Soup Documentation, Release 4.4.0, Leonard Richardson, https://buildmedia.readthedocs.org/media/pdf/beautiful-soup-4/latest/beautiful-soup-4.pdf
[3] A Study of Web Information Extraction Technology Based on Beautiful Soup, Chunmei Zheng1 , Guomei He1, Zuojie Peng2 1 School of Information Engineering, China University of Geosciences, Beijing, 100083, China. 2 Tencent, Inc., Beijing, 100083, China. http://www.jcomputers.us/vol10/jcp1006-03.pdf
[4] Unsplash License https://unsplash.com/license
[5] Pandas Data frame Documentation, https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html https://github.com/pandas-dev/pandas/blob/v1.4.2/pandas/core/frame.py#L459-L10970
[6] R. S. Dudhabaware and M. S. Madankar, "Review on natural language processing tasks for text documents," 2014 IEEE International Conference on Computational Intelligence and Computing Research, 2014, pp. 1-5, doi: 10.1109/ICCIC.2014.7238427.
[7] Optimizations for Dynamic Inverted Index Maintenance, Doug Cutting and Jan Pedersen, Xerox Palo Alto Research Center http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=86C188D06D5634BC2C86C914BE8FAA31?doi=10.1.1.56.5130&rep=rep1&type=pdf