Text Based Information Retrieval Search Engine

Prakrati Gupta
IIIT Delhi
MT20014
prakrati20014@iiitd.ac.in

Vandana
IIIT Delhi
MT20106
vandana20106@iiitd.ac.in

Nidhi Allwani
IIIT Delhi
MT20034
nidhi20034@iiitd.ac.in

Chetan Sharaf
IIIT Delhi
MT20109
chetan20109@iiitd.ac.in

Avaneesh Kumar Patel
IIIT Delhi
MT20005
avaneesh20005@iiitd.ac.in

ABSTRACT

This document is regarding the project presentation on Text based search engine.In this we are analysing the challenge faced in retrieval of document , Methods used to make document retrieval more efficient and will try to create a hybrid model which will implement various approach and try to find best solution to retrieve the query more efficiently.

1 MOTIVATION AND PRECISE PROBLEM STATEMENT

According to the Pew Internet American Life Project, Web search continues its explosive growth there are over 107 million Web-search users in the United. States alone, in June 2004, and they did over 3.9 billion queries. A study at the beginning of 2005 says indexable web size is at least 11.5 billion pages. Thus search algorithmic efficiency is as important as ever: although processor speeds are increasing and hardware is getting less expensive every day, the size of the corpus and the number of searches is growing at an even faster pace
Sice 2016, IBM Marketing Cloud study 90 % of the data on the internet has been created. All People, businesses, and devices have become data factories that are pumping out incredible amounts of information to the web each day.
Over 3.5 Billion Google researches are attended worldwide every breath of each day. That is 2 trillion explorations per year worldwide. That is above 40,000 search questions per second With an increase in data available on the internet, getting difficult to retrieve a result from such large data that is becoming a challenge.
Here we will discuss Some features used to make retrieval of data more efficient from Big data.

A. TF-IDF Model Analysis and it's Variant for Document Retrieval[10]
. . The TF-IDF power is a mathematical measure used to estimate how important a word is to a group or corpus document.
It takes preprocessing Steps like tokenization, stemming/ lemmatization, normalization, stops word removing, etc. are applied on each document.
for each document in the collection, It generates weighted term vectors and the user query. Then the retrieval is based on the similarity between the query vector and document vectors. The output documents are ranked according to this similarity.

B. Index Compression Techniques [12].
In inverted index compression, our goal is to compress a sequence of integers, either a sequence of d-gaps obtained by taking the difference between each docID and the previous docID, or a sequence of frequency values. In addition, we always deduct one from each d-gap and frequency, so that the integers to be compressed are non-negative but do include 0 values

C. Information Retrieval for Swedish using Stemming[3]
Stemming is a technique to transform different inflections and derivations of the same word into one common "stem." Stemming can mean both prefix and suffix removal.
Our information retrieval system uses traditional information retrieval techniques extended with stemming techniques and normalization of both the query and text and other technologies such as caching, data compression, early termination, and massively parallel processing.

2 CHALLANGES

discussed unique problems at high level that appeared to search engines[7]

- **Storage in Computing power**: As many indices need to be stored, collecting a lot of data from the web via crawling will require a lot of storage.
  Resolved: We can resolve the storage by using big storage like cloud and computation power can be resolved by referring for computer which has high performance
- **Spam** Search Engine optimization and various techniques can be used for page rank manipulation, which might not give the best results.
  Resolved: Spam can be resolved by using different ranking algorithm
- **Quality Evaluation**:Determining the status of various ranking algorithms is a specially challenging query. Popular search engines benefit from massive quantities of user-behavior data they can employ to help estimate ranking. Users normally will not attempt to give exact feedback but give implicit feedback, such as the outcomes they agreed.
  Resolved: Joachims [8] suggested experimental technique which merge the result of two ranking algorithm in a single set of result. So comparisons between two algorithm is done in this way.Joachims uses number of clicks as quality metric and show under which condition which algorithm retrieve more relevant links than others.
- **Content Quality**: Quality of the content depends on the person; different people have a different perspective when searching for a query. E.g., "I saw a girl with Binoculars." has two different interpretations.
  Resolved: we can Resolve the content quality by link analysis and text-based judgement

3 IDEA OF PROJECT

This model is a hybrid model by the combination of various techniques (Such as tokenization, stopwords removal, stemming, inverted indexing) to overcome the limitations and create an optimized search engine.

4 DATASET

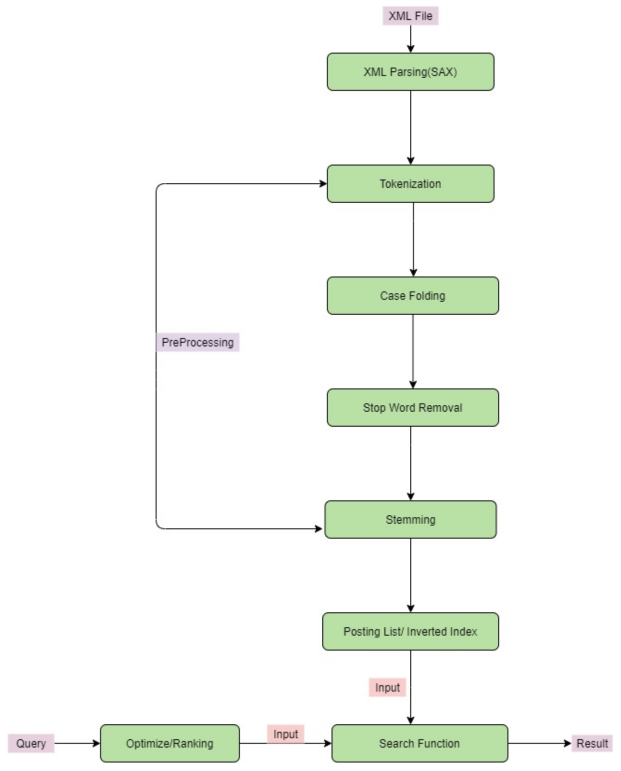
The dataset can be found here.
There has a XML file format of input data for this wiki search. It contains two major part in this data.

1. Title
2. Body

Title represents title of the document, which will be final result for any search and body part contains multiple things such as category, reference link, content of documents etc.

5 METHODOLOGY

Project Flow

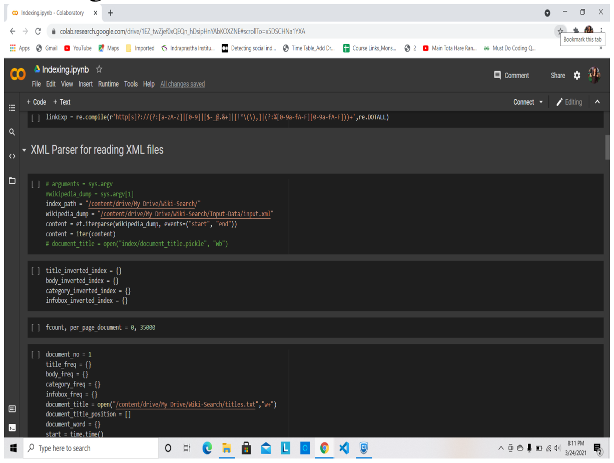


- XML parsing was done at XML data using SAX parser because it is most suitable for large datasets.
- Tokenization was applied at the parsed data.
- Case folding was applied at tokenized data.
- Stemming was applied at case folded data.
- Posting list/inverted index was created on the stemmed data.

6 RESULT

The Output file can be found here.

Parsing


Indexing.ipynb - Colaboratory
colab.research.google.com/drive/1EZ_twZjef0xQEQn_hDsipHnYAbKOXZNE#scrollTo=x5DSCHNa1YXA
Bookmark this tab
Indexing.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
Connect Editing
Type here to search
8:11 PM 3/24/2021

```
linkExp = re.compile(r'http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[!*\(\),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+',re.DOTALL)
```

XML Parser for reading XML files

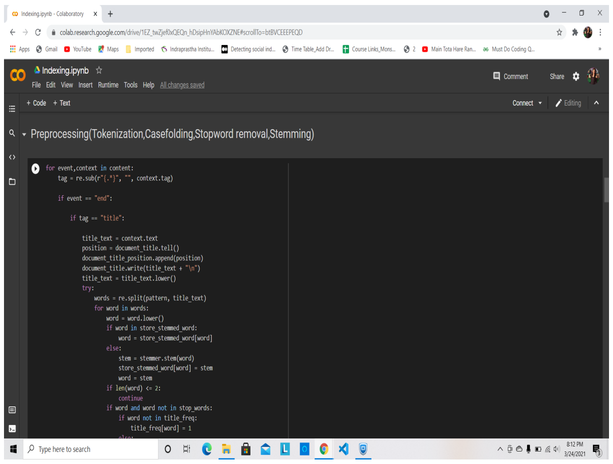
```
# arguments = sys.argv
#wikipedia_dump = sys.argv[1]
index_path = "/content/drive/My Drive/Wiki-Search/"
wikipedia_dump = "/content/drive/My Drive/Wiki-Search/Input-Data/input.xml"
content = et.iterparse(wikipedia_dump, events=("start", "end"))
content = iter(content)
# document_title = open("index/document_title.pickle", "wb")
```

```
title_inverted_index = {}
body_inverted_index = {}
category_inverted_index = {}
infobox_inverted_index = {}
```

```
fcount, per_page_document = 0, 35000
```

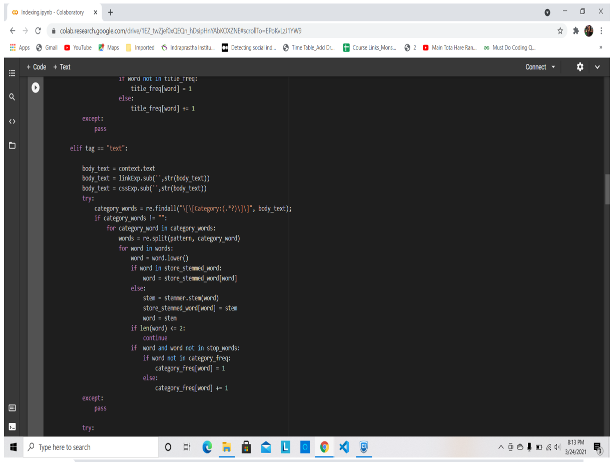
```
document_no = 1
title_freq = {}
body_freq = {}
category_freq = {}
infobox_freq = {}
document_title = open("/content/drive/My Drive/Wiki-Search/titles.txt","w+")
document_title_position = []
document_word = {}
start = time.time()
```

Preprocessing

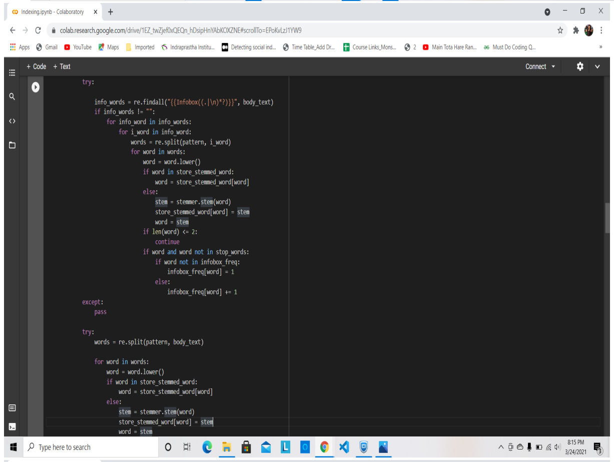

Indexing.ipynb - Colaboratory
colab.research.google.com/drive/1EZ_twZjef0xQEQn_hDsipHnYAbKOXZNE#scrollTo=btBVCEEEPEQD
Indexing.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
Connect Editing
Type here to search
8:12 PM 3/24/2021

Preprocessing(Tokenization,Casefolding,Stopword removal,Stemming)

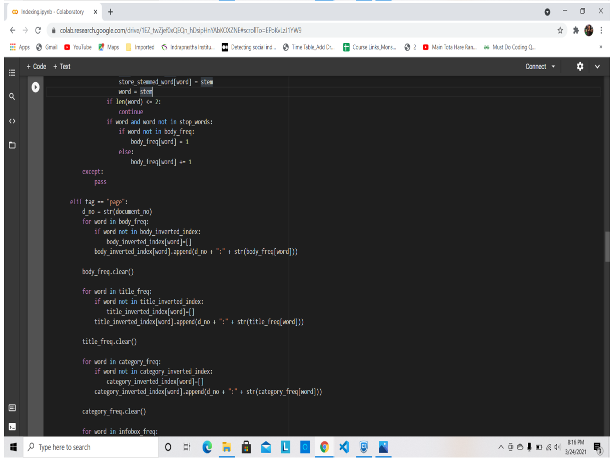
```
for event,context in content:
    tag = re.sub(r"{.*}", "", context.tag)

    if event == "end":

        if tag == "title":

            title_text = context.text
            position = document_title.tell()
            document_title_position.append(position)
            document_title.write(title_text + "\n")
            title_text = title_text.lower()
            try:
                words = re.split(pattern, title_text)
                for word in words:
                    word = word.lower()
                    if word in store_stemmed_word:
                        word = store_stemmed_word[word]
                    else:
                        stem = stemmer.stem(word)
                        store_stemmed_word[word] = stem
                        word = stem
                    if len(word) <= 2:
                        continue
                    if word and word not in stop_words:
                        if word not in title_freq:
                            title_freq[word] = 1
```


Indexing.ipynb - Colaboratory
colab.research.google.com/drive/1EZ_twZjef0xQEQn_hDsipHnYAbKOXZNE#scrollTo=EPoKvLzJ1YW9
+ Code + Text
Connect
Type here to search
8:13 PM 3/24/2021

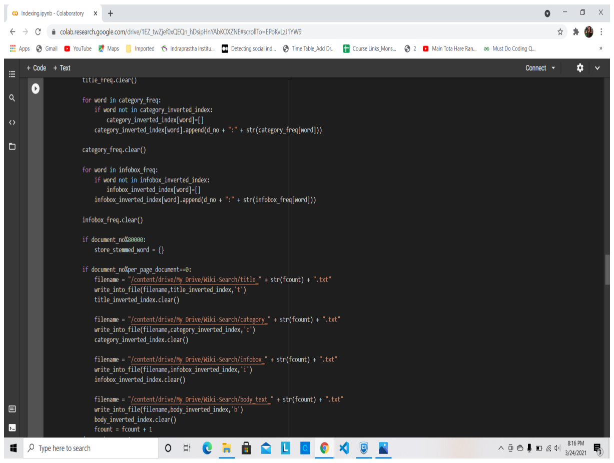
```
                        if word not in title_freq:
                            title_freq[word] = 1
                        else:
                            title_freq[word] += 1
            except:
                pass

        elif tag == "text":

            body_text = context.text
            body_text = linkExp.sub('',str(body_text))
            body_text = cssExp.sub('',str(body_text))
            try:
                category_words = re.findall("\[\[Category:(.*?)\]\]", body_text);
                if category_words != "":
                    for category_word in category_words:
                        words = re.split(pattern, category_word)
                        for word in words:
                            word = word.lower()
                            if word in store_stemmed_word:
                                word = store_stemmed_word[word]
                            else:
                                stem = stemmer.stem(word)
                                store_stemmed_word[word] = stem
                                word = stem
                            if len(word) <= 2:
                                continue
                            if  word and word not in stop_words:
                                if word not in category_freq:
                                    category_freq[word] = 1
                                else:
                                    category_freq[word] += 1
            except:
                pass

            try:
```


Indexing.ipynb - Colaboratory
colab.research.google.com/drive/1EZ_twZjef0xQEQn_hDsipHnYAbKOXZNE#scrollTo=EPoKvLzJ1YW9
+ Code + Text
Connect
Type here to search
8:15 PM 3/24/2021

```
            try:

                info_words = re.findall("{{Infobox((.|\n)*?)}}", body_text)
                if info_words != "":
                    for info_word in info_words:
                        for i_word in info_word:
                            words = re.split(pattern, i_word)
                            for word in words:
                                word = word.lower()
                                if word in store_stemmed_word:
                                    word = store_stemmed_word[word]
                                else:
                                    stem = stemmer.stem(word)
                                    store_stemmed_word[word] = stem
                                    word = stem
                                if len(word) <= 2:
                                    continue
                                if word and word not in stop_words:
                                    if word not in infobox_freq:
                                        infobox_freq[word] = 1
                                    else:
                                        infobox_freq[word] += 1
            except:
                pass

            try:
                words = re.split(pattern, body_text)

                for word in words:
                    word = word.lower()
                    if word in store_stemmed_word:
                        word = store_stemmed_word[word]
                    else:
                        stem = stemmer.stem(word)
                        store_stemmed_word[word] = stem
                        word = stem
```

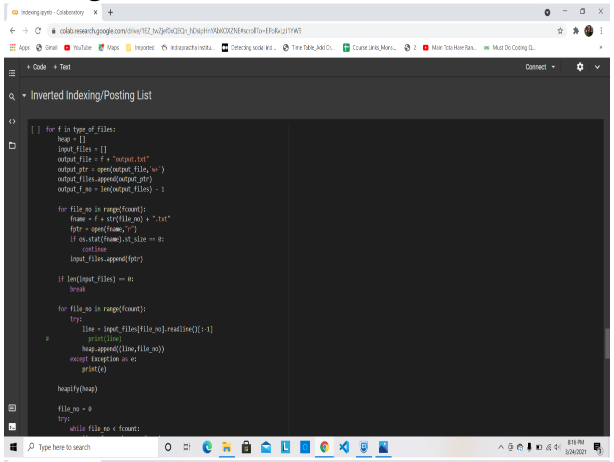

Indexing.ipynb - Colaboratory
colab.research.google.com/drive/1EZ_twZjef0xQEQn_hDsipHnYAbKOXZNE#scrollTo=EPoKvLzJ1YW9
+ Code + Text
Connect
Type here to search
8:16 PM 3/24/2021

```
                        store_stemmed_word[word] = stem
                        word = stem
                    if len(word) <= 2:
                        continue
                    if word and word not in stop_words:
                        if word not in body_freq:
                            body_freq[word] = 1
                        else:
                            body_freq[word] += 1
            except:
                pass

        elif tag == "page":
            d_no = str(document_no)
            for word in body_freq:
                if word not in body_inverted_index:
                    body_inverted_index[word]=[]
                body_inverted_index[word].append(d_no + ":" + str(body_freq[word]))

            body_freq.clear()

            for word in title_freq:
                if word not in title_inverted_index:
                    title_inverted_index[word]=[]
                title_inverted_index[word].append(d_no + ":" + str(title_freq[word]))

            title_freq.clear()

            for word in category_freq:
                if word not in category_inverted_index:
                    category_inverted_index[word]=[]
                category_inverted_index[word].append(d_no + ":" + str(category_freq[word]))

            category_freq.clear()

            for word in infobox_freq:
```



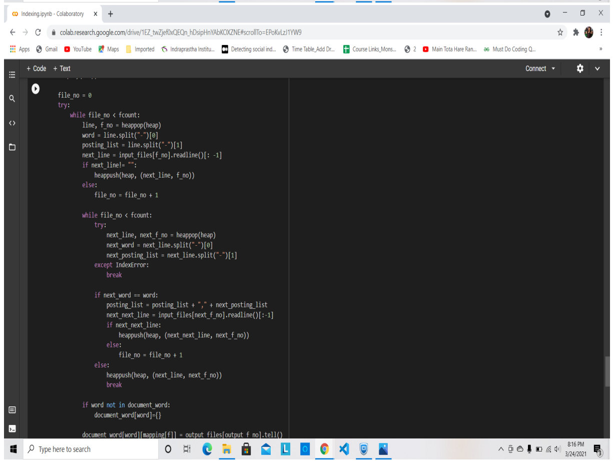
```
title_freq.clear()

for word in category_freq:
    if word not in category_inverted_index:
        category_inverted_index[word]=[]
    category_inverted_index[word].append(d_no + ":" + str(category_freq[word]))

category_freq.clear()

for word in infobox_freq:
    if word not in infobox_inverted_index:
        infobox_inverted_index[word]=[]
    infobox_inverted_index[word].append(d_no + ":" + str(infobox_freq[word]))

infobox_freq.clear()

if document_no%80000:
    store_stemmed_word = {}

if document_no%per_page_document==0:
    filename = "/content/drive/My Drive/Wiki-Search/title_" + str(fcount) + ".txt"
    write_into_file(filename,title_inverted_index,'t')
    title_inverted_index.clear()

    filename = "/content/drive/My Drive/Wiki-Search/category_" + str(fcount) + ".txt"
    write_into_file(filename,category_inverted_index,'c')
    category_inverted_index.clear()

    filename = "/content/drive/My Drive/Wiki-Search/infobox_" + str(fcount) + ".txt"
    write_into_file(filename,infobox_inverted_index,'i')
    infobox_inverted_index.clear()

    filename = "/content/drive/My Drive/Wiki-Search/body_text_" + str(fcount) + ".txt"
    write_into_file(filename,body_inverted_index,'b')
    body_inverted_index.clear()
    fcount = fcount + 1
```

Posting list

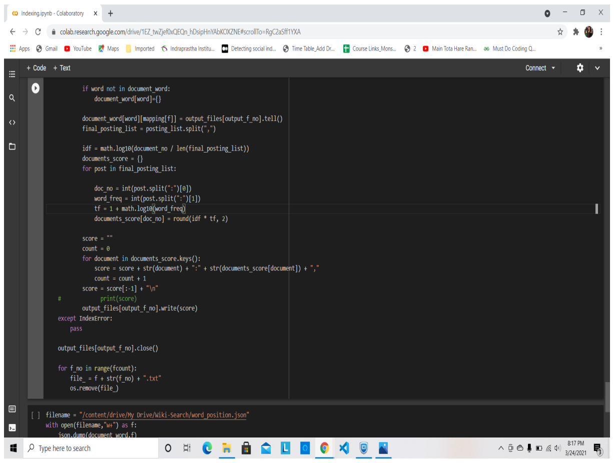


Inverted Indexing/Posting List

```
for f in type_of_files:
    heap = []
    input_files = []
    output_file = f + "output.txt"
    output_ptr = open(output_file,'w+')
    output_files.append(output_ptr)
    output_f_no = len(output_files) - 1

    for file_no in range(fcount):
        fname = f + str(file_no) + ".txt"
        fptr = open(fname,"r")
        if os.stat(fname).st_size == 0:
            continue
        input_files.append(fptr)

    if len(input_files) == 0:
        break

    for file_no in range(fcount):
        try:
            line = input_files[file_no].readline()[:-1]
#             print(line)
            heap.append((line,file_no))
        except Exception as e:
            print(e)

    heapify(heap)

    file_no = 0
    try:
        while file_no < fcount:
```



```
file_no = 0
try:
    while file_no < fcount:
        line, f_no = heappop(heap)
        word = line.split("-")[0]
        posting_list = line.split("-")[1]
        next_line = input_files[f_no].readline()[: -1]
        if next_line!= "":
            heappush(heap, (next_line, f_no))
        else:
            file_no = file_no + 1

        while file_no < fcount:
            try:
                next_line, next_f_no = heappop(heap)
                next_word = next_line.split("-")[0]
                next_posting_list = next_line.split("-")[1]
            except IndexError:
                break

            if next_word == word:
                posting_list = posting_list + "," + next_posting_list
                next_next_line = input_files[next_f_no].readline()[:-1]
                if next_next_line:
                    heappush(heap, (next_next_line, next_f_no))
                else:
                    file_no = file_no + 1
            else:
                heappush(heap, (next_line, next_f_no))
                break

        if word not in document_word:
            document_word[word]={}

        document_word[word][mapping[f]] = output_files[output_f_no].tell()
```



```
        if word not in document_word:
            document_word[word]={}

        document_word[word][mapping[f]] = output_files[output_f_no].tell()
        final_posting_list = posting_list.split(",")

        idf = math.log10(document_no / len(final_posting_list))
        documents_score = {}
        for post in final_posting_list:

            doc_no = int(post.split(":")[0])
            word_freq = int(post.split(":")[1])
            tf = 1 + math.log10(word_freq)
            documents_score[doc_no] = round(idf * tf, 2)

        score = ""
        count = 0
        for document in documents_score.keys():
            score = score + str(document) + ":" + str(documents_score[document]) + ","
            count = count + 1
        score = score[:-1] + "\n"
#             print(score)
        output_files[output_f_no].write(score)
except IndexError:
    pass

output_files[output_f_no].close()

for f_no in range(fcount):
    file_ = f + str(f_no) + ".txt"
    os.remove(file_)
```

```
filename = "/content/drive/My Drive/Wiki-Search/word_position.json"
with open(filename,"w+") as f:
    json.dump(document_word,f)
```

7 FUTURE WORKS

- Query Normalisation.
  - Tokenization of query.
  - Removal of stop words.
  - Case Folding of stop-words removed query.
  - Stemming of case folded.
  - Determine which of the logic operations should be used for getting appropriate results.
- Searching for documents when posting list /inverted index and optimized query will be passed as input in searching algorithm and it will give most related documents related to query using TF-IDF and word count.[4]

8 CITATIONS AND BIBLIOGRAPHIES

Sampling search-engine results[1],In this sampling is done on small part of data instead of full data.Analysis of tf-idf model and its variant for document retrieval authors worked with numerical as well as textual data and tried to find out which word, phrase or set of numbers can be considered more important relating to a particular query using different variation of vectorization techniques focusing more on TF-IDF.Improving precision in information retrieval for swedish using stemming ,Stemming make the words in its root form,defines how stemming improves both precision and recall for the swedish text.Inverted index compression and query processing with optimized document Ordering discussed how using previously optimised ordering , they developed a method in the compression technique that gives a higher throughput than most of the algorithms already present elsewhere Information retrieval on the world wide web[6] discussed navigation on web, how documents on web is represented, and models for retrieving information. Intelligent Search Engine algorithms on indexing and searching of text documents using text representation[9] describes and compares various indexing and searching technique with their complexities and which algorithm can be used for increasing indexing and searching speed in various circumstances. The authors of the paper,Document clustering: TF-IDF approach[2] provide a new overview of clustering the document where the similar documents can be clustered together, which makes the process of retrieving data much faster and efficient. CiteSeerX: AI in a Digital Library Search Engine,[11] article mentions how various machine learning and artificial intelligence techniques can be used for fast information retrieval, by using metadata , tables , subsections as features in the training model. The authors in the paper,[5] describe a basic procedure to use nificantly boost the page ranking process. The paper uses distributed format to compute the page rank.

REFERENCES

[1] Aris Anagnostopoulos, Andrei Z. Broder, and David Carmel. 2006. Sampling Search-Engine Results. *World Wide Web* 9, 4 (Dec. 2006), 397–429. https://doi.org/10.1007/s11280-006-0222-z

[2] P. Bafna, D. Pramod, and A. Vaidya. 2016. Document clustering: TF-IDF approach. In *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*. 61–66. https://doi.org/10.1109/ICEEOT.2016.7754750

[3] Johan Carlberger, Hercules Dalianis, Martin Hassel, and Ola Knutsson. 2001. Improving Precision in Information Retrieval for Swedish Using Stemming. (2001), 5.

[4] J. Shane Culpepper, Gonzalo Navarro, Simon J. Puglisi, and Andrew Turpin. 2010. Top-k Ranked Document Search in General Text Databases. In *Algorithms – ESA 2010*, David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Doug Tygar, Moshe Y. Vardi, Gerhard Weikum, Mark de Berg, and Ulrich Meyer (Eds.). Vol. 6347. Springer Berlin Heidelberg, Berlin, Heidelberg, 194–205. https://doi.org/10.1007/978-3-642-15781-3_17

[5] Atish Das Sarma, Anisur Rahaman Molla, Gopal Pandurangan, and Eli Upfal. 2015. Fast distributed PageRank computation. *Theoretical Computer Science* 561 (2015), 113–121. https://doi.org/10.1016/j.tcs.2014.04.003 Special Issue on Distributed Computing and Networking.

[6] Venkat N Gudivada, Vijay V Raghavan, William I Grosky, and Rajesh Kasanagottu. [n.d.]. RETRIEVAL ON THE WORLD WIDE WEB. *IEEE INTERNET COMPUTING* ([n. d.]), 11.

[7] Monika R Henzinger, Rajeev Motwani, and Craig Silverstein. 2002. Challenges in web search engines. In *ACM SIGIR Forum*, Vol. 36. ACM New York, NY, USA, 11–22.

[8] Thorsten Joachims. 2002. Optimizing Search Engines Using Clickthrough Data. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '02)*. Association for Computing Machinery, New York, NY, USA, 133–142. https://doi.org/10.1145/775047.775067

[9] D. Minnie and S. Srinivasan. 2011. Intelligent Search Engine algorithms on indexing and searching of text documents using text representation. In *2011 International Conference on Recent Trends in Information Systems*. 121–125. https://doi.org/10.1109/ReTIS.2011.6146852

[10] Apra Mishra and Santosh Vishwakarma. 2015. Analysis of tf-idf model and its variant for document retrieval. In *2015 international conference on computational intelligence and communication networks (cicn)*. IEEE, 772–776.

[11] Jian Wu, Kyle Mark Williams, Hung-Hsuan Chen, Madian Khabsa, Cornelia Caragea, Suppawong Tuarob, Alexander G. Ororbia, Douglas Jordan, Prasenjit Mitra, and C. Lee Giles. 2015. CiteSeerX: AI in a Digital Library Search Engine. *AI Magazine* 36, 3 (Sep. 2015), 35–48. https://doi.org/10.1609/aimag.v36i3.2601

[12] Hao Yan, Shuai Ding, and Torsten Suel. 2009. Inverted Index Compression and Query Processing with Optimized Document Ordering. (2009), 10.