WSM Final Report: Project C

1st Long Shui
Department of Computer Science, SJTU
Shanghai, China
StuID: 119033910144

2st Yang Zi
Department of Computer Science, SJTU
Shanghai, China
StuID: 119033910160

Abstract—In this project, we have developed a search engine based on Wikipedia dump and implement the following functions: (1) Inverted Index generation, (2) Support at least five kinds of searching methods, and (3) A friendly search GUI. Next, we will explain how this search engine works in detail.

I. Search Engine Implementation

We will divide this section into two subsections to express in detail how the search engine works. They are: (1) Inverted Index Generation and (2) Search.

A. Inverted Index Generation

As we all know that to build a search engine, it is inevitable to build an Inverted Index for the pages so that we can search according to the Inverted Index. The posting list of our search engine is in the output file. Let me explain to you in detail the components of the output file.
As we can see from Wikipedia pages, each page includes five part roughly: title, body, infoBox, category and external links. In order to facilitate our more detailed cross-zone query, we also divide the index files into five different parts and denote them with t, b, i, c, e respectively.
numberOfFiles.txt: This is a text file that records how many pages we have indexed.
t0.txt, t1.txt, ...: These text files are the posting list of the title, which are organized follows: {word docid1 documentFrequency1 docid2 documentFrequency2 ...}. For instance: 'coffey 1455 2 1239 1 1020 1 937 1 765 1' which means word 'coffey' appears twice in document 1455 and appears only once in document 1239 and so on.
ot0.txt, ot1.txt, ...: These files correspond the above files. They are organized like this form '840 5'. This means the offset of word 'coffey' is 840 and there are at least five documents appears word 'coffey' in the title part. The main function of these files are to find the inverted index of a word(eg: b0.txt, ...) according to the offset.
Similarly, the b0.txt, ob0.txt, i0.txt, oi0.txt, c0.txt, oc0.txt, e0.txt, oe0.txt, ... are organized in the same way, we will not elaborate any more.
title.txt: As the name suggests, title.txt is a file which records the titles of all the pages. It is organized follows: 'titleId titileName', where titleId is a unique number to help our search engine find the title.
titleOffset.txt: This is a supplementary file of title.txt file to help the search engine find the title.
vocabularyList.txt: This is a file that contains all the vocabulary. For example: 'instat 3 3', where 'instat' is the word, '3' is the file number that instruct which file the search engine should find and the last '3' is how many times the word 'instat' appears in all the pages.
offset.txt: This is a text file that help the search engine to find the specific word and its detailed information in the vocabularyList.txt.
There are three functions that are relevant to the indexing part in our source code: textProcessing.py, fileProcessing.py and Indexing.py. The textProcessing.py and fileProcessing.py are modules that serve for Indexing.py. The roles of textProcessing.py are create stopwords list, tokenise, stemming and process the five main part(title, body, infoBox, category and external Link) of Wikipedia pages. The functions of fileProcessing.py are write the inverted index into file and merge files. The main function of indexing.py are create the inverted index. Since the Wikipedia dump xml files are huge(About 75G after decompression), it is impossible to read all the files into the memory to parse. We use a SAX parser, which is suitable for parsing large xml files. After run the indexing.py program, we will get the inverted index!

B. Search

After generating the inverted index, we can search now! When you type the query words, our search engine will find the file numbers of each query word first. Then, according to the file numbers, the search engine will find the posting list of each word. After that it will find all the satisfying documents and then rank these documents according to our ranking algorithms. At last, we choose the top ten articles and return their titles and scores. We have implemented at least five kinds of ranking algorithms, they are: tf-based ranking algorithm, tf-idf based ranking algorithm, a factor change algorithm, parametric search and ranked boolean retrieval algorithm. We will discuss the ranking algorithms in detail at the Results Show part.

II. Results Show

First, let me show you the simple GUI we have developed with a django framework. Next, I will show you the

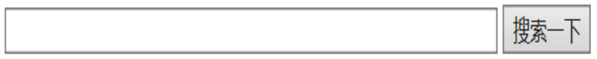
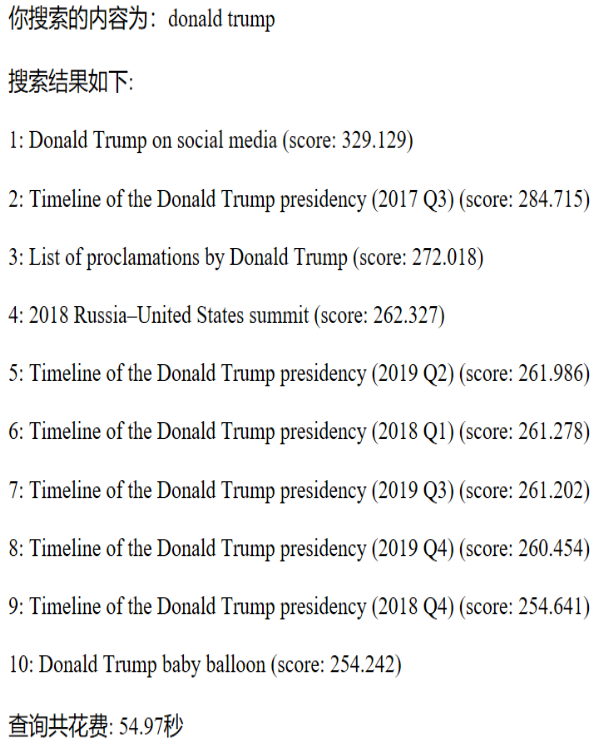


Fig. 1. Search GUI

interface to return results. When we type 'donald trump' we get the following results: The first line of the figure is



你搜索的内容为：donald trump

搜索结果如下:

1: Donald Trump on social media (score: 329.129)

2: Timeline of the Donald Trump presidency (2017 Q3) (score: 284.715)

3: List of proclamations by Donald Trump (score: 272.018)

4: 2018 Russia–United States summit (score: 262.327)

5: Timeline of the Donald Trump presidency (2019 Q2) (score: 261.986)

6: Timeline of the Donald Trump presidency (2018 Q1) (score: 261.278)

7: Timeline of the Donald Trump presidency (2019 Q3) (score: 261.202)

8: Timeline of the Donald Trump presidency (2019 Q4) (score: 260.454)

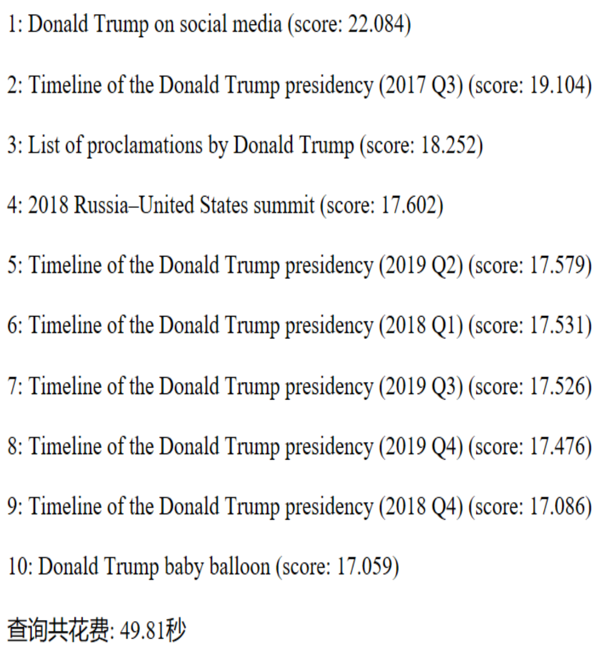
9: Timeline of the Donald Trump presidency (2018 Q4) (score: 254.641)

10: Donald Trump baby balloon (score: 254.242)

查询共花费: 54.97秒

Fig. 2. Interface of the Returned Result(tf-idf based)

the content you type in, and the next eleven lines shows the title of the returned result and their scores. The last line is the total time of this query.
Now let me show you the effectiveness of different algorithms. For, instance, we type 'donald trump' in the search box and change the algorithms and finally compare the results and search time of these algorithms. Note that we only compare the GUI version we offered to you, in fact, search time for console version are far less than the GUI version except the first time query(about 2 seconds), because it has read most of the inverted index into the memory the first time search. The next time you search, the engine will find most of the indexes in the memory but not the hard disk. However, due to lack of time, we have not improve the GUI version largely. Therefore, the average search time of console is much faster than the GUI version.



1: Donald Trump on social media (score: 22.084)

2: Timeline of the Donald Trump presidency (2017 Q3) (score: 19.104)

3: List of proclamations by Donald Trump (score: 18.252)

4: 2018 Russia–United States summit (score: 17.602)

5: Timeline of the Donald Trump presidency (2019 Q2) (score: 17.579)

6: Timeline of the Donald Trump presidency (2018 Q1) (score: 17.531)

7: Timeline of the Donald Trump presidency (2019 Q3) (score: 17.526)

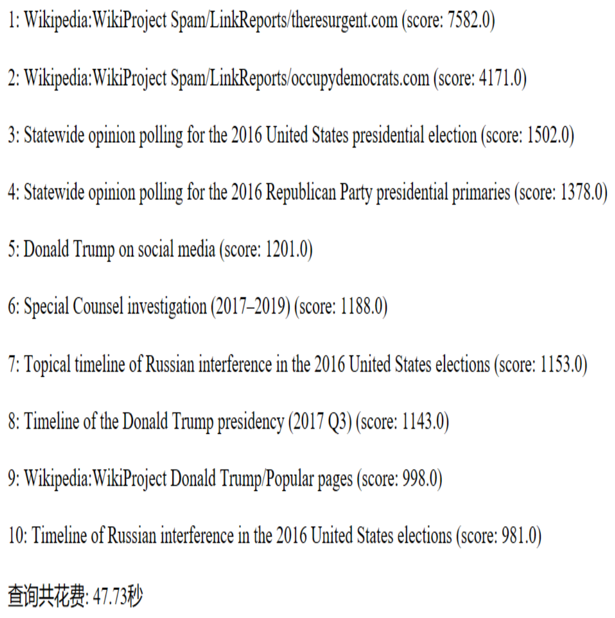
8: Timeline of the Donald Trump presidency (2019 Q4) (score: 17.476)

9: Timeline of the Donald Trump presidency (2018 Q4) (score: 17.086)

10: Donald Trump baby balloon (score: 17.059)

查询共花费: 49.81秒

Fig. 3. tf based with field weight {1,1,1,1,1}



1: Wikipedia:WikiProject Spam/LinkReports/theresurgent.com (score: 7582.0)

2: Wikipedia:WikiProject Spam/LinkReports/occupydemocrats.com (score: 4171.0)

3: Statewide opinion polling for the 2016 United States presidential election (score: 1502.0)

4: Statewide opinion polling for the 2016 Republican Party presidential primaries (score: 1378.0)

5: Donald Trump on social media (score: 1201.0)

6: Special Counsel investigation (2017–2019) (score: 1188.0)

7: Topical timeline of Russian interference in the 2016 United States elections (score: 1153.0)

8: Timeline of the Donald Trump presidency (2017 Q3) (score: 1143.0)

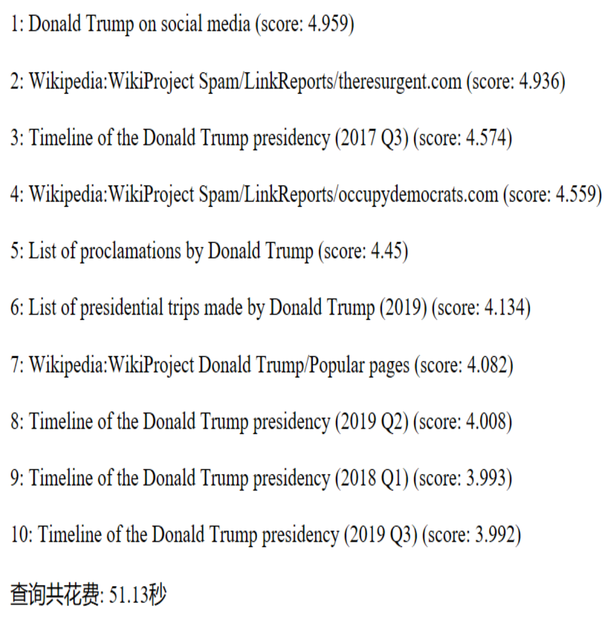
9: Wikipedia:WikiProject Donald Trump/Popular pages (score: 998.0)

10: Timeline of Russian interference in the 2016 United States elections (score: 981.0)

查询共花费: 47.73秒

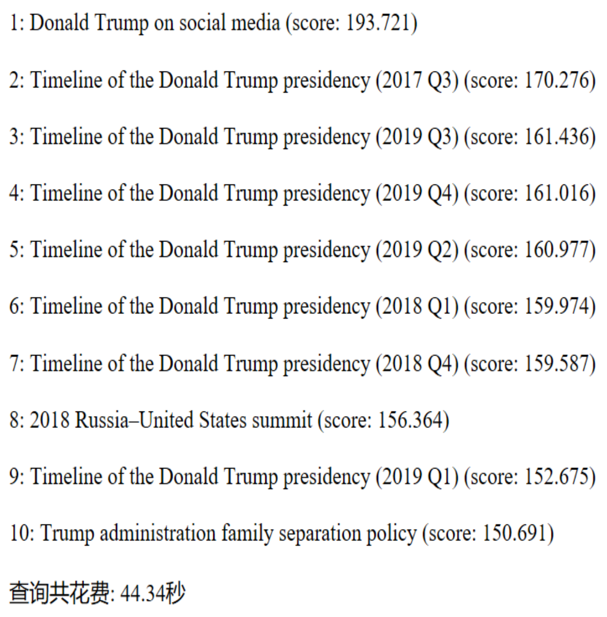
Fig. 4. tf based with field weight {0.3,0.3,0.2,0.1,0.1}

We can see from these figures that the average search time of different ranking algorithms, the search time gaps are not that big. In fact, the ranking algorithms do not affect the total time complexity bigly. Most of these algorithms are just change the calculation method but not affect the complexity. In particular, the parametric search method(specific zone search) spends less time than other methods. Because of that this method search only a specific area index instead of the entire index. Therefore, the search scale is small and the time complexity is



```
1: Donald Trump on social media (score: 4.959)

2: Wikipedia:WikiProject Spam/LinkReports/theresurgent.com (score: 4.936)

3: Timeline of the Donald Trump presidency (2017 Q3) (score: 4.574)

4: Wikipedia:WikiProject Spam/LinkReports/occupydemocrats.com (score: 4.559)

5: List of proclamations by Donald Trump (score: 4.45)

6: List of presidential trips made by Donald Trump (2019) (score: 4.134)

7: Wikipedia:WikiProject Donald Trump/Popular pages (score: 4.082)

8: Timeline of the Donald Trump presidency (2019 Q2) (score: 4.008)

9: Timeline of the Donald Trump presidency (2018 Q1) (score: 3.993)

10: Timeline of the Donald Trump presidency (2019 Q3) (score: 3.992)

查询共花费: 51.13秒
```

Fig. 5. ranked boolean retrieval



```
1: Donald Trump on social media (score: 193.721)

2: Timeline of the Donald Trump presidency (2017 Q3) (score: 170.276)

3: Timeline of the Donald Trump presidency (2019 Q3) (score: 161.436)

4: Timeline of the Donald Trump presidency (2019 Q4) (score: 161.016)

5: Timeline of the Donald Trump presidency (2019 Q2) (score: 160.977)

6: Timeline of the Donald Trump presidency (2018 Q1) (score: 159.974)

7: Timeline of the Donald Trump presidency (2018 Q4) (score: 159.587)

8: 2018 Russia–United States summit (score: 156.364)

9: Timeline of the Donald Trump presidency (2019 Q1) (score: 152.675)

10: Trump administration family separation policy (score: 150.691)

查询共花费: 44.34秒
```

Fig. 6. parametric search

relatively low.
Then let's take a look at the search quality of different algorithms. In fact, the quality of the contents searched by different algorithms is not large. Due to the lack of a uniform scoring standard, its hard to tell which one is more excellent. Till now, we find most of the ranking algorithms are acceptable.

III. Usage Guide

We offer you with two versions of code, the console version and the GUI version. And we test them on pycharm with python 3.7 version. Both versions need to install nltk and Pystemmer package first. For the GUI version, you should install a django package in the pycharm. Both of versions you should first run the Indexing.py with command 'python Indexing.py sourcexmlFile outputPath' to get the posting list where sourcexmlFile is the Wikipedia xml file and the outputPath is the path you want to save the posting list.(This step would take a very long time to generate the inverted index, hence we offer you with a mini posting list in the output file). For the search step, two versions are slightly different. For the GUI version, you should first run the manage.py(type 'python manage.py runserver' in the terminal), then open your browser, type '127.0.0.1:8000/search-form', then a search GUI will occurs. Then you can type content you want to search.
For the console version, you just need do type 'python search.py outputFilePath' where the outputFilePath is the place where the inverted index stored and then you will type the querywords.
As we have stated before, we have implement several search methods, how do we change different search methods? For the GUI version, you need to change the code of search.py in line 183 to the ranking method you want to use. For the console version you should change the code of search.py in line 197 to the ranking method you want to use. Our default algorithm is tf-idf based algorithm. For each algorithm we have implement multi-zone query. For example, if you want to query 'donald trump' only in the tile, you just need to type 't:donald trump' where t means title as we have stated before.

IV. Summary

We have implemented a search engine based on Wikipedia dump. However, this is just a prototype with lot of defects. For instance, most of the offset files are not necessarily needed, if we cut this part, the time complexity can reduce a lot. Due to a lack of time, its hard to do all the work. But by doing this project I have a deeper understanding of how a search engine works and have a better knowledge of the course. Thanks for Dr. Kenny Zhu's lecture and TA's effort and I have learned a lot from them.