E6998 Search Engine Technology Final Project

Hang Qian hq2124@columbia.edu

Nov 30,2012

1 Abstract

In this project, we implemented a search engine(api) of Chinese wikipedia pages. Using the zh.wikipedia.org dumped xml file of October 2012, we provide a set of api and command line interface to generate relevant pages based on the input query.

To evaluate our program, we test it on a datasef of over 1 million pages. We test it with several common/uncommon query and gets the satifying results though there's more work to do on effiency and accuracy. Part of the results are in results.txt.

2 Motivation

Wikipedia can be viewed as a huge collection of entities. It is based on objects like "America", "Operating Systems" etc. Sometimes our query could be "the composer who wrote Canon in D". It's better that search engine can "guess" what we want and give back a list of entities rather than a general search result, in which case we need more clicks to find out the real information we want.

Notice that Google has implement this feature in their search page. However it give you direct information if only there is one certain result(like "capital of Italy" but not "capitals in europe"). And unfortunately it's not available in Chinese.

I want to create a web interface based on several apis, to implement the likely features, in Chinese.

The project is expected to work like IBM' Watson did in Jeopardy: take a description and give an answer(list). Unlike a common Q&A system, my project is expected to give a list of objects that fits the description of the query, rather than a general answer. It will not be able to answer a question like "What's the weather tomorrow?" but expected to answer "Vice president of USA".

3 Tools and Dataset

The project is implemented in pure Python. The majority of the project is completely built from scratch, including parsing xml, database created, and query handling. We use jieba library to handle Chinese segmentation problem (since the main concern is not NLP).

We use MySQL-5.6 as the database server. Though not tested, we believe it's also should work on version greater than 5.0 as long as it supports InnoDB.

We use Chinese wikipedia pages as our dataset. The wikipedia pages is reasonable to choose for this project since it contains mostly of accurate and well described knowledge. For evaluation part, we use over 1 million wikipedia pages (nearly half of the whole data).

4 System Description

The system consists 4 parts.

4.1 Interface/API

We provide two interfaces to demonstrate the functionality. Before running the code, one need to change the variable `username` and `password` in `iris/db.qy` to your mysql server information. 1. Search. It asks user to input a query(in Chinese) and return a list of page titles(wiki page titles can be viewed as URI on the Internet) in order of their relevance. One can shoose how many results to return (default by 10). If result contains index 0, it means the query itself is one page's title. It is considered to be the most relevant one. 2. Create Database. If one want to test the code from scratch, he needs to create the database. We assume you installed MySQL server standardly on your machine. The database is named as "wiki", so you may ensure there's no database "wiki" created before or you may change the database name in code

4.2 XML Parsing

This part is in `iris/parse.py`. To handle XML structure, we use lxml library. Because the original dataset is huge(single file over 5Gb), we parse it iteratively. By go through the whole tree structure of the file, we store the information we need into two tables. Table `docs` contains the page information including page id, page title and the links to other pages it contains; Table `inverted_index` contains the inverted index we generated. For one certain word, we store the page id and frequency.

4.3 Database Operations

This part is in `iris/db.py`. It contains every operation that interact with database, including open,close,search,insert and update.

4.4 Query Handling

This part is in `iris/query.py`. Once we got a query, there are two things we need to do:

1. Search the database using the inverted index.
2. Rank the result based on their relevance.

The first part is considered trivial since we have taken time creating the index. Notice before we search the index, we need to check if the query itself is some page's title. If so, we return it first with highest rank. To rank the result, we consider two parts of the pages we got.

1. The sum of tf-idf. Assume query = , page p1 is in the return list of w1 and w2, then the importance of p1 is `TF-IDF(w1)+TF-IDF(w2)`. Thus the more important word contributes more to the final score.
2. Coverage. If a page is in most of the query words, it should be consider more relevant to the query. Thus we multiply the score we got by the square of the number that one page appears in return list of index search.

5 Evaluation

The system shows satisfying results on test case.

For instance, by query "诺贝尔和平奖 中国人"(Nobel Peace Prize, Chinese), it returns (top 10)

index	title
1	刘晓波(+++)
2	2010年诺贝尔和平奖(++)
3	中華人民共和國新聞自由(+)
4	2009年诺贝尔和平奖(++)
5	贝拉克·奥巴马(++)
6	红十字国际委员会(+)
7	翁山蘇姬(++)
8	無國界醫生(+)
9	艾默理大學(-)
10	西奥多·罗斯福(+)

We see that with mark (+)is actually relevant to the search query. And the top one is exactly what we want.

By query “自由软件基金会，Emacs作者”(Free Software Foundation, The author of Emacs), it returns (top 10)

index	title
1	理查德·斯托曼(Richard Stallman)(+++)
2	Emacs(++)
3	自由软件基金会(FSF)(++)
4	Python(-)
5	GNU通用公共许可证(GPL)(+)
6	GNU通用公共许可证(GPL)(+)
7	GNU宽通用公共许可证(LGPL)(+)
8	Copyleft(+)
9	Linux历史(Linux History)(+)
10	LISP(+)

We can see that this result is nearly perfectly close to what we want.

5.1 Potential Improvement

There are at least two aspects that our system can be improved. 1. Though we store the out links, due to the huge amount it need to compute, we didn’t actually use it. We believe if we can generate the gragh of the links between pages, the result can be more accurate and intelligent by using the idea of page-rank. 2. It can be improved by introducing the semantic of the query. For example, if the query is “The author of Emacs”, it clearly refers to a human. If we are able to retrieve this information and put more weight on pages that describes human, we will gain better results.

5.2 Demo Output

Several demo output can be found at result.txt.