Search Engine based on VSM and word2vec for *info.ruc.edu.cn*

Luo Bei

July 22, 2020

Abstract

In this work, A tiny search engine for *info.ruc.edu.cn* is maded. Firstly, I crawl and store the webpages from *info.ruc.edu.cn*. Then I use *cpp-jieba* to split the contents to words and convert the webpages to vectors(VSM). To train *word2vec*, I download over 360000 documents from *zh.wikipedia.org* and split the sentences into words. After doing this, for every query, every webpage will get a total score which combines the VSM score with the word2vec score, to evaluate the similarity between the query sentence and the webpage content. Finally, I use *cpp-httplib* to build a web server and show the titles and bodys of ten webpages which get the top 10 scores and are most similar to the query.

1 Function of the Search Engine

The search engine including a web server and a search API. Users can search from the web page or get the result urls from API.

The search engine do well on :

1. Search the correct name of person or structure.
2. Precise search short or long text.
3. Search the text which is not from the webpages but related to the webpage contents.

The search engine doesn't well on

1. Search words that not appear in any webpages.
2. Search long text that contains so many words which appear in many webpages.

2 Design of the Search Engine

2.1 Crawl and Store

To crawl all the webpages from *info.ruc.edu.cn*, *BFS(Breath First Search)* is used. From the root page, I use the system command *wget* to get the htmls. Then the program read downloaded htmls and find new urls from *info.ruc.edu.cn* till there are no new urls found. To store the pages, I change the filenames of the webpages into their urls. Since character '/' is not allowed to appear in the filenames, I replace all charactor '/' with charactor '.' .

2.2 Parser

2.2.1 Split Title and Body

Spliting titles and bodys are mainly through parsering all the html tags.

It's easy to get title by tag '<title>'. But it's hard to get body. I parser the '<p>', '<h1>', '<h2>', '<h3>', '<td>', '<li>', ' ' tags in order. But some webpages don't have this tags. For these webpages, I parser the tag ' ' and then delete all characters which are not Chinese words or character ' '. For most of webpages, this works. But for less than 100 pages, this don't work. These webpages have no content but a flash object so I can't get the content.

2.2.2 Inverted Index & Chinese Word Segmentation: *cpp-jieba*

cpp-jieba is a efficient Chinese Word Segmentation project which is writen by c++. *cpp-jieba* provides dictionary of over 300000 words and over 1000 stop words. *cpp-jieba* can split the sentence into words very quickly. By spliting the titles and bodys of the webpages, it's easy to create an inverted index for every word which records the IDs of the websites which contain the word. It become easy to find the ducuments which the word in query appear in by inverted index, since the query is often short while the contents are long;

2.3 Vector and Score

It's slow and hard to do things with strings. So I have to vectorize the contents and query.Two models, *VSM* and *word2vec*, are used to vectorize the strings.

2.3.1 TF-IDF, VSM & Scoring

In information retrieval, tf–idf or TFIDF, short for term frequency–inverse document frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus.

cpp-jieba provide a method to calculate tf-idf. It calculate the tf-idf by the equation:

$$w_{word,doc} = tf_{word,doc} \times idf_{word}$$

The idf_{word} comes from the idf dictionary of over 250000 words, which provided by *cpp-jieba*.

The equation is not working well as some of the words from documents can't be found in the idf dictionary and the equation is not good enough. So I write the program to calculate tf-idf by the equation:

$$w_{word,doc} = (1 + \log tf_{word,doc}) \times \log \frac{N}{df_{word,docs}} \ (N = the\ number\ of\ docs)$$

As $\log \frac{N}{df_{word,documents}}$ won't change, I update the idf dictionary by $\log \frac{N}{df_{word,documents}}$. When the word in query appear in the crawled websites, its idf equal to $\log \frac{N}{df_{word,documents}}$. When the word in query don't appear in the crawled websites but appear in the dictionary which is provided by *cpp-jieba*, its idf equal to idf_{word}. When the word in query don't appear both in the crawled websites and *cpp-jieba*, its idf equal to the average idf of all the words.

To vectorize the documents, every document has its own vector:

$$t_{doc} = \begin{bmatrix} w_{words[0],doc} & w_{words[1],doc} & \cdots & w_{words[n],doc} \end{bmatrix}^T$$

$$v_{doc} = \frac{t_{doc}}{||t_{doc}||}$$

And the score between query and document is the cosine of the vectors times 10-based log of the number of the words which appear in both query and document. The equation is

$$\begin{aligned} sco_{qry,doc} &= \lambda_{qry,doc} \times v_{qry} \cdot v_{doc} \\ &= \frac{\lambda_{qry,doc}}{||t_{qry}|| \times ||t_{doc}||} \sum_{word} w_{word,qry} \times w_{word,doc} \\ \lambda_{qry,doc} &= \log_{10} |\{word | word \in qry \cap doc\}| \end{aligned}$$

Since not so many words appear in one document, most position of the vector is 0. So saving the vectors is not necessary. Just saving the tf-idf value $w_{word,doc}$ in the inverted index and $||t_{doc}||$ is enough.

2.4 *word2vec* & Scoring

word2vec is a group of related models that are used to produce word embeddings. These models are shallow, two-layer neural networks that are trained to reconstruct linguistic contexts of words. *word2vec* takes as its input a large corpus of text and produces a vector space, typically of several hundred dimensions, with each unique word in the corpus being assigned a corresponding vector in the space. Word vectors are positioned in the vector space such that words that share common contexts in the corpus are located close to one another in the space.

2.4.1 Train model & Vectorizing

To train *word2vec*, I download all of the articles from *zh.wikipedia.org* while wikipedia provides the download url. There are over 360000 articles. After word segmentation, they are trained by *word2vec*

The command to train the model is

./word2vec -train wiki.zh.text.jian.seg.utf-8 -output vectors.model -size 200 -window 8 -negative 25 -hs 0 -sample 1e-4 -threads 20 -binary 0 -iter 15

After training, 200-dimension vectors of the word are saved in *vectors.model*. When scoring, I load the vectors directly from *vectors.model* without training again.

2.4.2 Scoring

After getting all the vectors of words, the vector of a document or a string is calculated through the equation:

$$t_{doc} = \sum_{word \in doc} vector_{word}$$

$$v_{doc} = \frac{t_{doc}}{||t_{doc}||}$$

And the score between query and document is also the cosine of the vectors. The equation is

$$\begin{aligned} sco_{qry,doc} &= v_{qry} \cdot v_{doc} \\ &= \frac{1}{||t_{qry}||} \sum_{i=1}^{200} t_{qry,i} \times v_{doc,i} \end{aligned}$$

The v_{doc} can be calculated when parsering and analysing the webpages, and loaded before query. For every query, calculating v_{qry} and $sco_{qry,doc}$ is enough and quick.

2.5 Web Server

I use cpp-httplib to build a web server. The pages is crawled from Google search engine. And all the words which match the words in query are marked.

3 Help & Deploy

3.1 How to Use

After Deploy, run the *server* and then visit 'localhost:1234'. You will find the search page.

3.2 How to Deploy

My operating system is Ubuntu 20.04 LTS. It may have unexpected error on the other OS.

3.2.1 Before Deploy

Before Crawling,you need to train *word2vec*. There are some models which is already trained. You can download them to 'dict/vectors.model'. Attention: vectors.model must be a normal file instead of a binary file.

If you choose to train by yourself. I recommand articles from wikipedia. You can download them from download.wikipedia.org easily.

3.2.2 Crawling

Compile and run pc.cpp in './urls'. And then you will get over 7000+ webpages. You need to delete some files such as .doc .xls .css and so on. Files whose size are less than 1kb are need also to be deleted. Files with url 'download.php' and 'userfiles' are also need to be delete.

3.2.3 Parsering

Compile and run parser.cpp. And you will get 'dict/dict.utf-8' and 'dict/htmls.utf-8'.

3.2.4 Run

Compile and run server.cpp. And it's able to use.