KoalaGo: A Simple Search Engine

Zhuohao Yu

July 22, 2020

Abstract

KoalaGo is a simple search engine practice program built for course *Summer Programming Practice 2020*. This essay will explore the main features and technical challenges in the project.

1 Introduction

KoalaGo mainly features 4 functioning parts: the webpage crawler, the HTML Parser, the Search Engine itself and the frontend.

The project is implemented in Python with Flask serving as the backend framework and Bootstrap4 as frontend framework.

For word segmentation, I came up with a simple approach with the help of pkuseg, jieba, HanLP and baiduLAC to improve accuracy as well as efficiency.

2 Details and Challenges

2.1 The Webpage Crawler

In this project, our goal is to collect every page from website: http://info.ruc.edu.cn/

In order to improve efficiency, I implemented a multi-thread Python crawler. Accoring to tests performed on my local envionment, the crawler could retrieve all the pages within 62 seconds with 16 threads.

A simple trick on improving efficiency is to setup a URL blacklist, so that the crawler could skip any page with non-HTML format files.

However, as Python does not natively support atomic ++ operator, I have to set a thread lock when the threads try to retrieve a unique file ID. Otherwise there will be duplicate IDs thus causing confusion.

2.2 The HTML Parser

The HTML Parser reads all the retrieved pages on the disk and convert the HTML pages into plain text.

This part is especially challenging because we prefer a universal way to solve the problem, instead of writing rules that only applies to this specific project.

Since it's quite hard to determine which part of text is actually the main content, we'll take a different approach.

My HTML Parser is based on BeautifulSoup. It filters anything that's not in HTML format, then try to remove all the external links, scripts and comments on the page. If the page authors followed the basic rules of HTML when creating the website, this method will work in the intended way. Glad that http://info.ruc.edu.cn/ have proper format and followed the rules, this approach seems to be working very vell.

In order to improve accuracy in word segmentation, I first generated a large dictionary containing all the unique results by pkuseg and jieba. Then, I took advantage of the entity tagging feature of HanLP, which is based on BERT and thus it's running quite slow on my local environment. So only applying entity tagging in a preprocessed dictionary is necessary to reduce processing time. If a word in the roughly made dictionary contains an entity, the word will then be saved in the "accurate" dictionary file. At last, I have baiduLAC to load the user dictionary and redo the preprocessing on all pages. When a user input any query, the query will first be cut into words

with the same method. According to my tests, this optimization on word segmentation is very effective since TF-IDF algorithm relies heavily on word frequency.

2.3 The Search Engine

The search engine itself and the indexing parts are all written in python. As we all know, Python is not the most efficient programming language, but it's the most convenient one with many useful libraries and native support for UTF-8 characters. With some trivial optimization on choosing the data structures, the TF-IDF algorithm works quite efficiently on Python.

To achieve better accuracy for results, I assign more weight to words that appear in both the title and the content by applying the following formula:

$$W_{term,doc} = 1 + \log_{10}(tf_{term,doc} \times length(term) \times factor)$$

Where $factor$ is a constant relying on the type of the document with a simple rule.

This method ensures that longer words from user query could lead to more accurate result with the help of proper word segmentation method mentioned above.

2.4 The Frontend

Since the program is written purely in Python, Flask is the best lightweight framework for user interface.

Bootstrap4 is used to make the responsive interface user-friendly and mobile compatible.

3 Conclusion

From this project, I've learned more aspects to real-world programming other than emphasizing on speeds. Being able to try new things like Python, Flask and Bootstrap is actually a fun process. This course brought me a chance to take a glance at real-world projects, and I really enjoyed doing all these work.