An Introduction to Search Engines

Travis Hite
Matt Glazebrook
Andy Poore
Russ McCoy

Abstract

The size and diversity of the internet has made information retrieval one of the key fields in data representation. Websites like Google index as much of the internet as possible providing a simple way for users to input keywords in order to efficiently retrieve a list of results organized by importance. The need to give users the ability to find information quickly and efficiently will only become more challenging as the data available on the internet becomes increasingly larger and more unwieldy.

As the field becomes more and more diverse, the need to comprehend the simple components vital in creating a search engine will increase. Understanding the complexities that must be surmounted is important in broaching this subject. As the field expands, the importance in understanding the basics inherent in creating a search engine becomes vastly more important.

1. Introduction

There are many ways to begin understanding the functionality of search engines. There have been many books written on the subject matter, but the concepts presented without an overarching theme may feel abstract to the reader. We present our own search engine, from start to finish, along with information regarding the process which we used to create it, as an instructional tool for the targeted user.

1.1 Concept

An important thing to remember is not to reinvent the wheel. There are already many search engines that do the same thing. We feel that creating a search engine with a slightly modified goal is the best solution.

As such, we have created a search engine that will solely crawl powerpoint files from educational websites. We feel this is enough of a departure to make some minor difference between the underlying basic functionality and to display the problems inherent in creating a search engine.

1.2 Functionality

We begin by creating a general overview of the functionality of our search engine. Later, we will break this down into unique segments in order to instill a greater understanding of the functionality of this basic search engine to the reader.

The first step is to crawl the internet for important files. The internet is a large, heterogeneous creation, and must be examined on a large scale. We must begin by actively seeking new locations, and extracting relevant documents.

We then store the files locally so they can be properly dissected for relevant information. Until these files have been properly processed, we have not yet gained anything useful from them.

Following this, the information must be indexed. Useful text is extracted from the powerpoint slides and converted to a format useful to the algorithm. The terms

used in a document are noted, as well as the frequency with which they are used.

At this point, the user will access the front end and submit a string representing the information for which he is searching. We are using a simple weighted boolean search method, which uses the terms "and", "or", and "not" to sort the documents efficiently.

The search is finally evaluated against all documents that have been indexed. A comparison of the terms used in the search is made with those which were indexed. The results are listed in the order of highest to lowest ranking of importance, and then returned to the user.

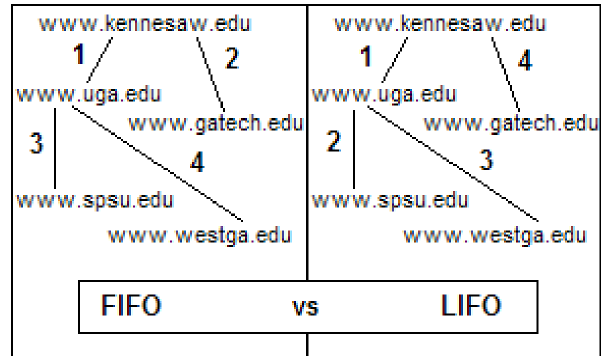
3. Crawling

The very first step is to crawl the websites; i.e., we scan every website we can find. Any information pertaining to our search criteria is noted.

We begin with a list of starting URLs, also called nodes. This primary listing of URLs is called a seed. We begin at our first node. Next, we examine the web page, specifically looking for links. If we find a URL that links to a .edu domain, we add this to the end of the list. If it links to a .ppt file, it is converted to a text file and stored.

The page is parsed for URLs by looking for the <a href=> tag. After the first page is scanned, the URLs are checked against the alreadySearched file. If a page is found that already has been searched, it is not saved. If it has not been searched, it is added to the end of the toBeSearched file.

There are two methods for continuing. Initially, there is the breadth first method, also known as first in first out (FIFO). Consider the hierarchy of URLs to be searched as a tree, branching as it goes down. Using FIFO, the top URL in the list



Graph 1

is searched, ensuring pages are searched level by level.

Also, there is also the depth first method, also known as last in first out (LIFO). In this method, the last URL added is the first one to be searched. The tree diagram is still applicable under this method. However, instead of ordering by level, it searches as far as it can down a specific path.

The powerpoint to text conversion proved to be the most difficult step. A powerpoint file, seen as a normal text file, is garbage to the human eye. Extracting relevant information, in an efficient manner, proved difficult. A third party solution was vital. Several vendors provided solutions, however the vast majority demanded some form of compensation for their product.

We finally decided on using Aspose, which is a free utility. However it is specifically a file management product, and has a hefty API. Our solution was to loop through all slides. Each slide contains a certain number of "shapes", or specifically, fields for information to be held on a slide. The type of shape is then checked. If it is a type that contains regular text, it is stored to a text file. Any other form of information is dumped.

This process continues indefinitely, as it is impossible for one person to crawl the entire internet. Also, websites change over time, and a date would need to be set where the nodes, which have already been searched, can be searched again.

4. Indexing

After crawling we are given a collection of text files containing purely the text of the powerpoint files. However, in their present state, they still mean nothing, until we have parsed the text files, and indexed them for relevant information. At the end of this process, we will obtain an array list that our algorithm can use.

To begin, each text file is checked for stop words. Stop words are words that are used commonly, but frequently, or have no meaning in context to the remaining text. These words need to be removed. For example, you might want to filter out articles or adverbs. Examples of stop words would be "of" "the" "you" or "and".

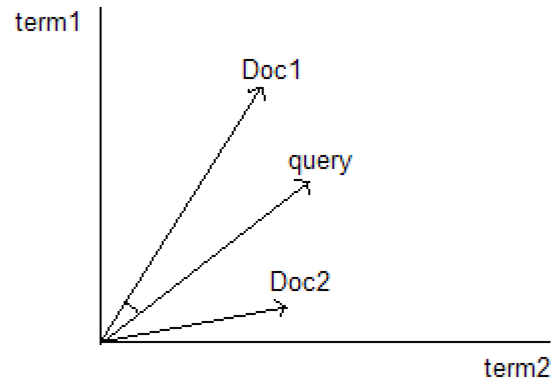
Following this, the information needs to be serialized; i.e., it should be processed in such a way that is more efficient than having the program read a straight text document. The text is parsed, to remove spaces and punctuation, and saved as an array list.

This array is then scanned, word by word, and if a term is read that has not been read before, it is added to the list of terms. If the term already has been found within the file, the total number of times that term has been found, or, the term frequency, is noted.

The terms, and their term frequency, are then stored. Finally, the array is deserialized and stored permanently.

4. The Algorithm and Presentation

It is extremely important to keep the interface simple to the user. As stated previously, we have decided to use a boolean search method for simplicity's sake.



Example Vector Space Graph

Graph 2

In boolean search, there are three fields of importance. If it is entered in the "and" field of the query, it must be in the document returned. If it is in the "or" field, it is preferred to be in the document, but it is not vital. Finally, if it is in the "not" field, it should not be in the document returned at all.

Rather than implement such harsh search restrictions, we have decided to assign a weight to each word. This keeps from totally eliminating entries that may have been valuable to the user, but may have contained a word, in a different context, than the one they had in mind.

In comparison, there are several other ways to compare the documents against the user's input. Consider for example, if each word is a dimension in a graph, and the term frequency is a vector within that dimension, such as one would find in a vector-space graph. One could construct a vector for each document and the query. Then, consider the difference of these vectors, to find the most relevant term. However, this would raise some subtle problems, as to how many dimensions should be considered and how best to model the vector for the user's query. In the end, it seemed as if the boolean model would be the best fit for our scenario.

Each document is looped through, with the user's query used as input. For

each term in the document that matches the user's input in the "and" field, 1 is added to that document's weight. For each term that corresponds to the "or" field, 0.5 is added. Finally, for each term that corresponds to the "not" field, 0.5 is subtracted. These values determine the total weight of each document. The documents are then sorted, according to highest total weight, and only those above a specified threshold are returned to the user. This keeps the user from receiving a plethora of useless documents, with low rating.

Finally, these documents are presented to the user from highest relevance to lowest, at a rate of 10 documents at a time. The user may sift through these documents at his own will.

5. Conclusions

From here, a lot of things could be done to make the search engine better. There are obvious things, such as using a more dynamic algorithm, or retrieving additional file types such as word documents and various other file types. However, this basic information should be sufficient in creating a basis for a dynamic search engine.

There is a wide range of uses for such an engine. It is important to learn how to tailor the engine to your specific needs. No one approach is better than another, and there are varying applications for specific user needs. With enough flexibility and attention to design detail, making a new, useful search engine is simply a matter of creativity and approach.