Interactive Searching with Semantic Extraction

Mengyi Zhu

October 24, 2014

1 Abstract

In this project, I will try to focus on the problem of finding a better way to search. I try to design the system based on my understanding of how people would do it without computers. Since I believe the communication and the human searcher's own intelligence are critical to make the process of search efficient, I try to examine the possibility of building an interactive search engine which can "think" and "talk".

2 Introduction

There are more and more digitized content in the form of news, blogs, web pages, scientific articles on web today. Searching engines like Google can crawl millions of pages every minute and use some advanced strategies to rank the pages. However, even provided with a perfect ranking strategy, finding exactly what the users want is still hard. For example, when user searches the word "pizza", there is not enough information for the search engine to guess whether the user wants the recipe or a restaurant for pizza. In this case, ranking all the information without considering the semantic behind the keyword will return too much unwanted information.

However, understanding the semantic meaning of user's input is also very hard because in reality every person has different interpretation of things and no general rules can be applied to all the use cases. Therefore, when the users input more keywords to describe what they want, those keywords may not be efficient enough to help the search engine to narrow down the search range. Even worse, if the user specifies a wrong keyword, or the new keyword itself is strongly related to other things, the search result will be driven further away. For example, when the user thinks of "Where can I get pizza in America" and type "America" as addition to "pizza", the search result now may contain information about "American cheese" which is completely unrelated.

After all, the power of search engine is graceful because it can deal with a large amount of information; likewise, the power is also a waste due to information over-fitting. On the other hand, human beings can deal with this searching problem quite efficiently even with limited power. For example, Once professor

Alan Kay talked about how to find a book in the library in old days. At that time, one knowledgeable librarian worked like a search engine in the library, and people always talked to him in order to find books. Obviously, the term "talk" is the key here because during the talk, this person is able to understand what you are looking for and what book should be suitable for you. Inspired by this example, I think a search engine can be smarter as long as it can provide a communication interface and negotiate with the user to find the correct search result.

Another thing to consider is the intelligence of the search engine. As in the example of the library system, after the person figures out the goal of the customer, he tries to find matching books in his knowledge. This process requires the person to find books with related topics. However, current search engine only searches by word occurrence and ranks them based on a general level. Therefore, I think a more semantic-related strategy may be useful in finding the search result.

3 Related Work

In order to find the user's intent, search engines currently give suggestions to help the user decide what keywords to use to search the result. Suggestion systems can be very helpful to narrow down the search range as long as the user accepts one of the suggestions. In some sense, suggestion systems provide some intelligence in finding the right result, but they have some limitations as well. First of all, the intelligence of suggestion systems comes from history, that is, it records other user's behaviors. The reason that you can get suggestion like "Are you looking for pizza places?" when you type in "pizza" is because someone else has also searched for pizza places. In fact, the intelligence of suggestion systems is in how to organize and rank the suggestions based on history rather than come up with suggestions of what they really know.

One well-know problem therefore for the suggestion systems is called *cold start* [1]. It concerns the issue that the system cannot draw any inferences for users or items about which it has not yet gathered sufficient information. For example, if no one has searched for pizza places, the user will never get suggestions about pizza places. In addition, in order to make the suggestion system efficient, the number of suggestions are limited and the user may never find useful suggestions simply because what he/she is looking for is not majority. Another problem of suggestion system is between each suggestions, how to guarantee they are indeed suggesting different things. For example, "place to eat pizza" and "pizza restaurant" are suggesting very similar things.

After all, the suggestion system is helpful when the searching result is popular and it can quickly make suggestions based on user input. However, it is not enough to make searching accurate. In the case where the user is willing to sacrifice time to find accurate search result, suggestion system is not helpful enough to improve the searching quality. A tool is therefore required to look deeply into the information and try to extract useful knowledge for the user.

4 Approaches

The project can thus be divided into two parts, one handles the interaction with users and the other handles the intelligence of information extraction.

4.1 Communication Interface

The communication interface is relatively easy and it only needs to hold some key concepts of communications with users:

- The user should be able to input some words to indicate what he/she wants
- Some outcome should be presented to the user to ask the user to narrow down the semantic meaning of his/her input
- The user should be able to choose a presented outcome which he/she thinks is most related to the goal
- If the user cannot decide with given outcome, he can further update his/her searching keywords

These operations are simple yet they provide some functionality to make communication possible. Both the search engine and the user can negotiate to change their input or output to make things work. Unlike the suggestion system, the user has the chance to either accept or reject the output of the search engine and upon receiving the response from the user, the search engine can further refine the search results.

4.2 Information Extraction Intelligence

The information extraction intelligence depends on a back-end program that can take the user's input and project its knowledge database on the specified input. Moreover, in order to refine the search result, it needs to understand the search result. For example, if the search engine has a knowledge database of articles about pizza, then when searching related articles, it should have a way to tell whether one particular article is about pizza restaurant or pizza recipe. Obviously, this cannot be done simply by finding the word appearance, we need to look deeper into those articles and find out what these articles are about. Therefore, the back-end program will use some topic extraction skills to extract topics from these articles.

Besides the ability of extracting topics from the knowledge database, the back-end program should also have a way to understand the relationship between each topics. For example, we can have different but related topics with keyword pizza such as recipe, flavor, restaurant and so on. Some times, the user may have difficulties to choose one topic over another. For instance, it may be hard for the user to choose either recipe or flavor. Therefore, the back-end program should be able to group topics into different groups so that it is easier

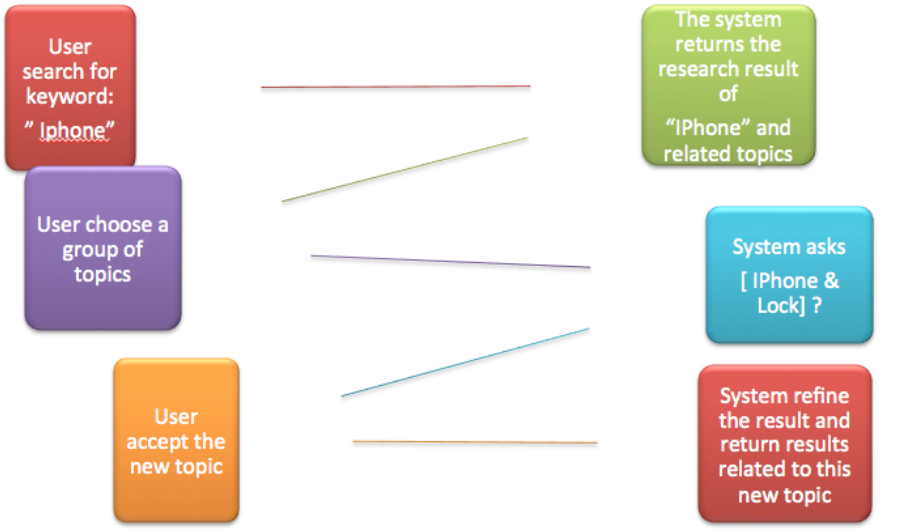

Figure 1: An Example of use cases

for the user to choose one group over another. By recursively grouping the topics, the system expects the user would get closer and closer to his/her goal and eventually stop at certain point.

4.3 Putting It Together

As a result, the system projects its knowledge database on the user's input first and tries to extract different semantic meanings. After converting semantic meanings into topics, it will group the topics into two groups and output the groups to the user. Then, the user can either choose one group or further specify additional keywords to clarify what he/she wants. By iteratively regrouping or adding new words, eventually the user will accept the result or the system will run out of results.

5 Implementation

In order to build and examine the system, we not only need the communication interface and the back-end intelligence but also need a knowledge database to present the things the search engine knows. In this section, I will go into details of the actual implementation of the system part by part.

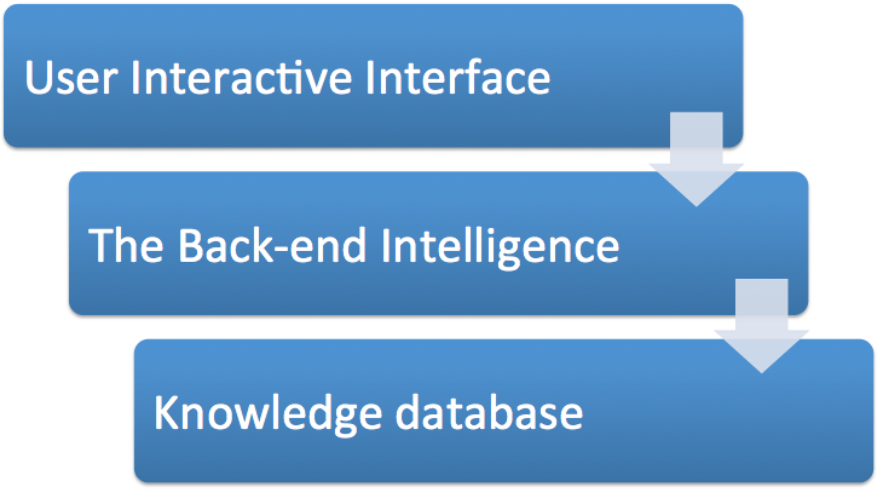

Figure 2: System Diagram

5.1 Knowledge Database Management

Because the knowledge database projection and topic extraction depend on the user input, it requires the database management to be handled in runtime as much as possible. In other words, if our database consists of articles, we need to support fast retrieving and searching method to get the articles. In this project, I used the MySQL database to manage my database and the database I am using are archives from StackExchange forums. Therefore, the information I stored in the database is about online conversations. I built the schema and wrote script to store information about threads, comments and authors. The index I built are based on the body of comments, posts and labels. In other words, when searching a keyword, threads contains this keyword in its posts, comments or labels will all be retrieved.

5.2 Communication Interface

There are two communication interfaces for this project. The better one is built in python for testing purpose. It takes a set of keywords to search and will output results found in number of posts. Once the results are found, it waits for the user to decide what to do with the search result. It can either take a new input keyword or it can let user choose one group of topics to further narrow down the results. It can also output all the results as long as the user wants to. Similarly, the web browser based interface has all these functionalities as well. Intentionally, the interaction on the web browser based interface should have more graphic effect to make the interaction easier and clearer. Nevertheless, it

is not well developed at this time.

5.3 Back-end Intelligence

Once the system gets the initial search results from the database, the back-end intelligence will write them into files and use natural language process to omit stop words and change words to word stems. This step is for improving the quality of topic extraction later on. I accomplished this by providing a very general list of stop words and used the nltk library. Then, the system will combine all the posts into one corpus and use LDA to extract topics from the corpus[3]. Since the LDA can extract user-specified number of topics and each topic is represented by a set of words, I chose two topics with twenty words each for the purpose of performance and accuracy. Because LDA is not a perfect topic extraction tool, the topics may not be independent enough to tell each other apart. Therefore, it is possible that when user chooses one group of topics over another, he/she may be misled by the words and choose the wrong group. Furthermore, the more groups there are, the more likely the user will make the wrong decision. Therefore, I kept the minimal number of topics which is two to decrease the chance of making the wrong choice.

When the user specifies a new keyword, it is also the back-end intelligence's duty to further prune the result based on the additional keyword. This process is quite intuitive that we just need to keep all the posts which also contains this new keyword or contains this keyword in it's topic and omit the rest. Because after using LDA, we obtained a model of all the posts, we can simply treat the additional keyword as a topic and query whether or not the posts contains this topic. However, it is not possible for LDA model to query a never-existed topic, so we have to make sure the additional keyword is actually a topic in the LDA model. To simplify this, I have limited the range of the additional keyword so that only words shown in the output of the system can be additional keywords.

Another possibility is that the user may choose one group of topics over another. In this case, posts related to any topic or words within this group should be kept. On the other hand, posts that have no relationship with any of the topic in that group will be omitted. After that, the selected group of topics will be further divided into two groups and returned to the user again. In order to do so, the system needs a way to decide which topics are related to each other and it also needs a fast grouping algorithm to make it work in runtime. To answer the first question of how to decide relationships between topics, I made a simple assumption that topics in a single article or post are more related to each other. In other words, if we take the intersection of the posts of different topics, the larger the intersection is, the more related the two topics are. Therefore, I built a correlation matrix based on the posts and topics and used this matrix as a reference to group them. As the answer to performance of the grouping algorithm, I chose to use k-means since the dimension of the matrix is not large. So far, building the matrix and using k-means are both fast enough to return the grouping result in runtime.

6 Evaluation

The evaluation goal of this project is to examine the possibility of such a system. The major concern is always about performance and accuracy. In order to test the performance, I built this system on a moderate laptop. The knowledge database contains posts from three forums and a total size of 3GB data. I randomly come up with some searching goal and keywords to see if the system can interact with me in runtime.

As a result, the performance bottle neck is at the topic extraction step. Due to the design, I have some file I/Os in the system that affect the performance and can be removed to boost up the performance. As an approximation, when the search result is as big as 1GB, it takes around 5 minutes to finish topic extraction and is beyond the runtime acceptance.

Besides, another evaluation goal is based on accuracy. Since LDA is not a perfect topic modeling tool and a lot of research interest are still in this area, it is possible to improve the result of extracted topics for our system. So far, it seems that the topics generated are very closely related to each other. The difference of topics can only be told apart by one or two particular words out of twenty. Therefore, it is hard for the user to choose one group of topics over another. I think in order to improve the system, a better topic extraction tool is more desired than the performance.

7 Discussion

The goal of this project [2] is to take a different approach in improving the quality of searching. Unlike the suggestion systems in current search engine, the interactive system I built is not aiming for an immediate solution. In fact, this system is expected to have performance and accuracy issues even before it was built. However, I believe with careful engineering and improving skills in topic extraction, such a system may become a solution of the problem in the future.

Moreover, we can make this system more scalable than the suggestion system. Imagine that some small service providers have small search engines with our back-end intelligence. Upon receiving the request, they are able to deal with the interactive searching in runtime as long as the database is small enough. In addition, based on the service they provide, they can generate information to indicate what kind of request each of them can handle. Then they can upload those information to a parent search engine as part of its knowledge database. By doing so, when a searching request is sent to the parent search engine, it is able to first interact with the user to find out which child search engines are most suitable and redirect the user to those small search engines. With such structured searching process, we may guarantee runtime interaction at each step.

As a conclusion, in this project I tried to explore the possibility of adding interactions and intelligence to search engines to improve the quality of searching.

Apparently, there are still many problems to make it practical. In addition, it is expected to see more challenges to come up in the future. However, without considering the performance and accuracy too much, adding interactions and intelligence is very helpful to find out what users really want. This may be a research direction to look at and I expect to see more research and engineer effort to be done in this field in the future.

References

[1] How google instants autocomplete suggestions work. `http://searchengineland.com/how-google-instant-autocomplete-suggestions-work-62592`. Accessed: 2014-04-30.

[2] My code for this project. `https://andy0727@bitbucket.org/andy0727/interactive-search-engine.git`. Accessed: 2014-06-12.

[3] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022, 2003.