Job Hunter Query System

Hide our individual information: powered by:Yao, Lu, Chen, Cao

Individual Contribution:

Yao: Data processing, algorithm optimization and UI development of whole query system

Lu: make slides of presentation, schedule our project steps

Chen: Crawl data, converse the data type and describe dataset

Cao: UI of login system, look for reference, introduction and related work

Abstract

For graduates, looking for a job is often a troublesome but have to do thing. This leads to the need to screen job recruitment. In this regard, information retrieval can play a great role. Our project function is similar to the search engine, but also has the function of learning the user's choice to help screen out the useful information for users. We will remove stop words with nltk for English and Harbin Institute of Technology stop word list for Chinese. The quicksort algorithm will be used to sort the results of cosine similarity because of its wonderful time efficiency on sorting. Besides we hope that in the existing program, we can improve the overall operation speed, and apply the content related to big data and machine learning to the project, so as to achieve the goal of making the program more intelligent as a whole.

1. Introduction
2. Definition of our problem

Our project is called jobhunter. obviously, users can use this application to hunt the job. In other words, this application is designed for users to find the suitable job. We will take full advantage of what we've learned in Information Retrieval and build a search engine by combining what we already know about Python usage. As a search engine, its function is similar to that of big search engines such as Baidu and Google. And our project more professionally meets the needs of job seekers.

1. Examples of our problem

By way of example, if someone who major in CS wants to be a JAVA engineer. What he needs to do is just inputting the keywords "Java engineer". Our program will query and display the data containing these two keywords in the database, so that he can easily and quickly obtain the job information he wants. In addition to Searching jobs directly. Maybe some people want to look for a job close to home, or some people want to work in another city. Candidates can also enter the area they want to work in our search engine, and we will provide some alternative job information in this city. Our project also adds a learning feature, which is designed to allow users to make judgments about the information being quested. If the information queried is not relevant to what the user wants, the user can click the irrelevant button. And in the further query work, the irrelevant information is excluded. Of course, the relevant buttons are also designed in our application. When the user finds the appropriate job information, he can click the relevant button. The content of this message is recorded and learned by the system, and on the next query, the system imposes weights on the information that has been recorded. Take an example, someone who queries the job information in Nanjing city and he clicks the related button about the Recruitment information which needs a Java engineer. When he queries for the second time, more information about the Java engineer will appear.

1. Motivation

In the traditional job application process, there are too few channels of communication between candidates and recruiters. Many times, candidates often spend a lot of time without finding the right job, and many companies don't hire the right people for this reason. So, it is the value of our project which saves candidates time in finding the right job opportunity.

1. Application

At present, some mature websites or applications also provide similar functions, and we have borrowed some of their designs in the design of our project

Liepin platform provides job information for middle and high-end talents. Many headhunters provide online job-hunting services for you. It covers many well-known enterprises in various industries and provides resume guidance, interview guidance and other job-hunting services.

1. Related Works

First of all, we looked up some papers on Jobhunter. In The article "The Job Hunter's Guide to The Library"[1], the author describes the Career Resources found in most public libraries. Includes Information on Reader indexes and specific references that are available. While in the article "Job Hunter's Sourcebook"[2], the author gives the opinions that what the employees need to find a job. These two articles give us fundamental realization about the realization of what employment leads and other job search resources, process of hunting a job and what the candidates consider when they are looking for a job[3][4].

Then, we looked up some information about information retrieval and search engines. By looking up these materials, we gained a deeper understanding of the relevant development process and learned some new ways to deal with data. For example, in the article "Job Search Engine and Methods of Use"[5], the author provides a method of accumulating, processing, and classifying the online job listings for searches by the users. The job listings are automatically, and more effectively, categorized, allowing users to more quickly and easily search job listings. And in the "Analysis of a very large web search engine query log"[6], they present an analysis of an AltaVista Search Engine query log consisting of approximately 1 billion entries for search requests over a period of six weeks. Their correlation analysis showed that the most highly correlated items are constituents of phrases. This result indicates it may be useful for search engines to consider search terms as parts of phrases even if the user did not explicitly specify them as such. These articles have been of great help to us.

1. Dataset Description
2. Method of data collection

We used a technology named web spider to crawl the data on the specific website. We divided this technology into five steps and implemented it with Python. Generally, these five steps are shown in the following part.

First, we need to find the given URL of the website we require. Second, we visit the website and check the HTML source code. Third, we parse the HTML code into JSON type with a tool website, which is convenient for us to find the data we want to attain. Fourth, we write Python code to simulate a real human being to send request to the website server for the purpose of fetching data. Finally, we save the data we crawled into csv file and converse it into utf-8 type.

We will talk about these steps in details with some images. Firstly, we get the dataset from the recruitment website called “lagou wang”. This website allows us to fetch the data without logging in. It is convenient for us to attain the URL of the website in Chrome. Then, we visit the home page of the website, and then find the Ajax location. In order to conveniently check the datatype, we parse it into JSON type. We check the source code of the website, and find response of our required positionAjax which is shown in Fig1, and then put the response code in the JSON analyze tool to get the datatype like the image shown in the following Fig2.

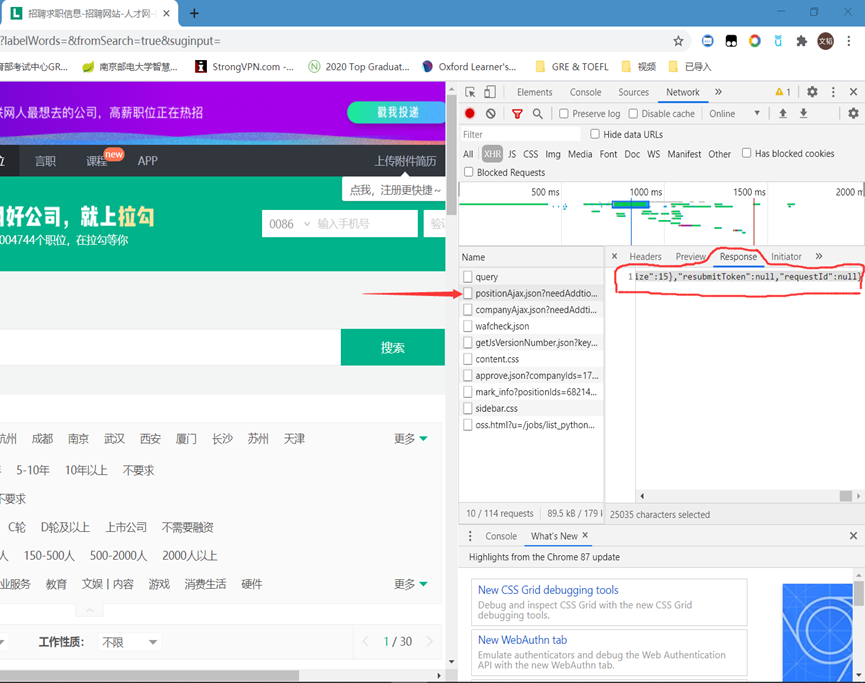


Fig1. Response of PositionAjax



Fig2. Results of datatype after JSON parsing

Subsequently, we create a session of request class to simulate user to send request to the server because the lagou wang uses asynchronous loading to load data. We need to copy the request URL of the response of positionAjax. Finally, we can get the data saved in the csv file.

1. Issues encountered during data collection

When we start to crawl the data, we ought to pay attention to the time interval of each crawling. We set the time interval a very small value in the beginning so that our request is denied by the website server. So, we set one second to crawl the data for sake of not being denied by frequently visiting the website server.

1. Data annotation

Because we set the datatype as utf-8, the csv file is full of messy code. To solve this problem, we use excel tool to change the utf-8 data into readable content.

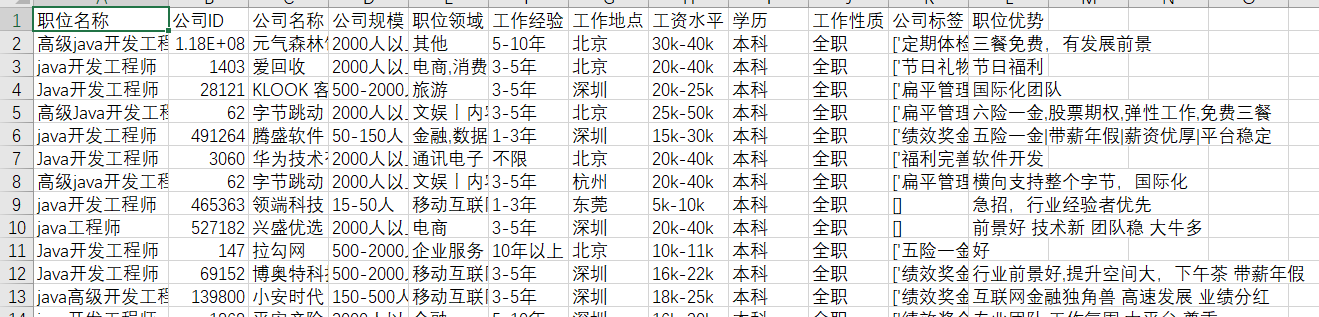
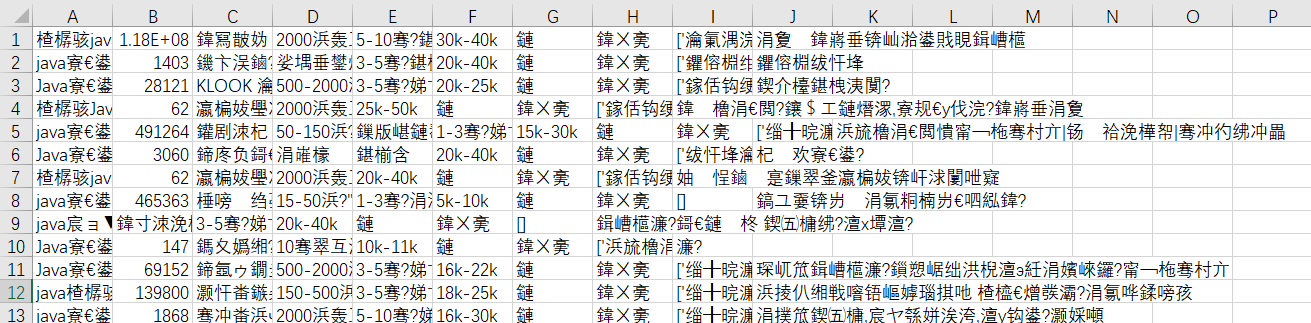


Fig3. Original data and converted data

1. Approach Description

We followed the basic pipeline and implemented ranking search engine with some classical algorithms that we have studied.

Firstly, we combine elements in each record into strings, and take these strings as documents. Then we tokenize them. Chinese text needs to obtain individual words through word segmentation, so we import jieba library to tokenize Chinese data we get from the results of crawled data. Jieba library is an excellent third-party library for Chinese word segmentation. The principle of word segmentation in Jieba library: use a Chinese thesaurus to determine the correlation probability between Chinese characters, and the words with high probability between Chinese characters form phrases to form word segmentation results.

Jieba library provides three modes of segmentation: 1) Precise Mode, returns a list-type participle result with no redundancy. 2) Full mode, returns a list type word segmentation result with redundancy. 3) Search engine mode returns a list-type word segmentation result with redundancy that subdivides longer words. Here we choose the precise mode. Besides, we also use nltk to tokenize english data if there is any. Then we remove stopwords with nltk for english and Harbin Institute of Technology stopword list for chinese. To combine words with the same meaning, for english, we use porterstemmer in nltk; for chinese, HIT IR-Lab Tongyici Cilin (Extended) can be employed in the future, but here we do not do chinese synonym mergence. Then we count the frequency of words in each document into a frequency matrix. Next, we calculate tfidf matrix with normalization. Tfidf matrix is a multiplication result of tf matrix and idf matrix. Tf matrix is calculated as , where f is the frequency matrix. IDF is about the whole corpus, calculated as, where N is the number of all documents, the denominator is the number of documents that contain this term.

We store the tfidf matrix for the database and read that every time our search engine is started, so we do not have to calculate it besides the first time. The calculation takes time. IDF is also stored for follow-up calculation.

The queries and the records marked by users as relevant or irrelevant are firstly taken the same as records, calculating tfidf, but with the stored IDF for the whole corpus. Then the query and marked record list are processed by standard rocchio formula: , generating a new query with weight. If we choose simple search mode without history, this step is skipped, and the query frequency vector is directly used.

Finally, the cosine similarity between the weighted query and TFIDF matrix of the database is calculated. Then a quick sort is executed. Quick sort mainly takes advantage of the mind of divide-and-conquer. In other words, firstly, the original problem is divided or reduced into smaller sub-problems, then the sub-problems are solved recursively or iteratively, and finally the sub-problems are synthesized to obtain the solution of the original problem. In the end, we get the ranked list with document indexes and their similarity scores. Then we go back, find the documents by the index and show them.

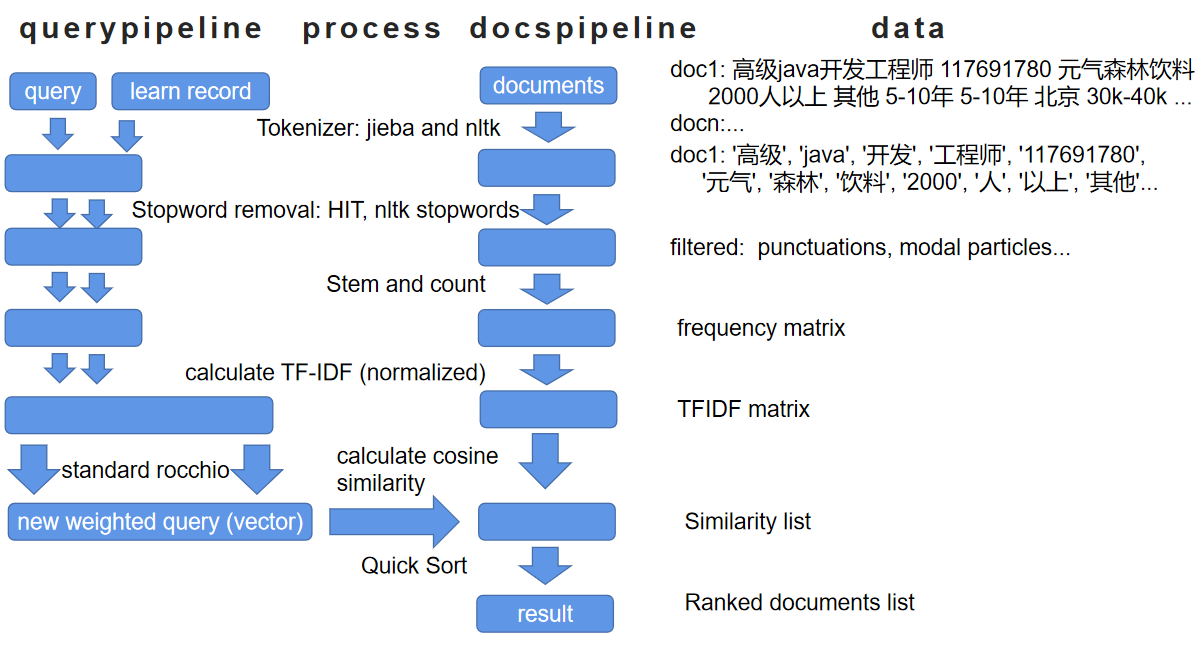


Fig4. Pipeline of our system

1. Experiments and Results

We use tkinter in python to build our user interface. We use pickle to serialize user information. We made a full function of a standard login system, we have login and register functions, we have verification, handle of any invalid manipulation.

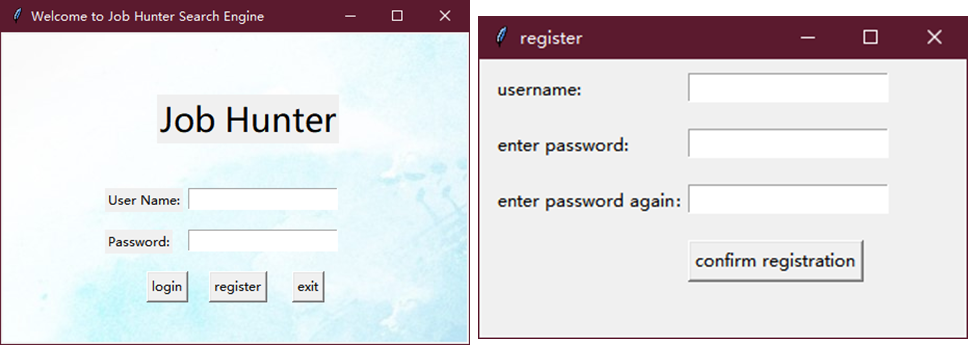


Fig5.6. Login windows

On the top of main window are input boxes and buttons, left are buttons to record user marks, right is a table to show the result. On the left bottom we also provide page swift to show more results.



Fig7. simple search with “阿里巴巴杭州2000人以上”

Our users can mark documents as relevant and irrelevant through clicking buttons on the left, and our system can learn from record. For example, if we choose the 11th record as relevant, the result when we search with learning is showed in Figure 8. Notice that our learned search does not take current input into consideration.



Fig8. Learned search example

Attention points:

The number, for example, 5 to 10 years, the little lines will be eliminated as stop words and two numbers will become together. So, we have to make sure there is at least one space between numbers, I basically replace these with a space before tokenize.

The repetitiveness of the identification information may influence our result. For example, every company have a unique name and a unique company ID, so when our system learns from the marked documents, it will learn more from this, which means the repeated identification will have a higher weight when learn, the same company will more likely to go up the rank than the same information from other angles.

Evaluation:

We lack the goad standard. When we tried to use some example queries both in our system and Lagou website and make comparison to evaluate our result, since our dataset is relatively small to the whole Lagou database, it shows problem, many relevant records show up which are not in our database. We may do meaningful evaluation in the future when our database is close to Lagou database. But we still show some methods of evaluation here.

Let Positive and Negative be the relative and irrelative result our system classify, True and False be the accuracy of our classification, which is a comparison between our result and real class, the same is true, vice versa.

|  |  |  |
| --- | --- | --- |
|  | classify right | classify wrong |
| classify as positive | TP | FP |
| classify as negative | TN | FN |

Table 1: evaluation base

Some clasical evaluations:















ROC curve: Roc curve is a curve of FPR and TPR pairs of all possible thresholds, FPR on x axis and TPR for y axis. The bigger the area below the curve, the better the system is.

1. Conclusions and Future Work

Our goal is to obtain a large number of data accurately and quickly to prepare for subsequent analysis, which is relatively smooth to complete crawling data and analyze it as JSON and create a session of request class to simulate user to send request to the server. Due to the attention to the time interval, the problem of possible denial of access has also been resolved.

In the actual writing, it is found that it can't guarantee the complete correspondence on the basis of long word partition like search engine, which is solved after installing Jieba library. For the search results, we found that the input of repeated keywords will affect the search results, and then through the analysis of TF IDF to solve this problem. In addition, the program can learn the user's choice through the array, so as to know whether the search results are relevant or not. We store TFIDF matrix for database and solve the problem of redundant calculation. In addition, the program can learn the user's choice through the array, so as to know whether the search results are relevant or not.

In the process of the experiment, for user login and document relevance selection, our writing is more successful, and there are no problems in the experiment. But it was found that there would be errors in the processing of numbers. So, we chose to separate the numbers with spaces.

At the same time, in the experiment we found that it is a bit slow of our system each time we calculate. It is because that we take queries as documents and calculate them with documents together, in every time we search. This part can be separated out of the user systems. To improve the system, the tfidf results can be calculated and stored in advance. Then we may only calculate standard rocchio and similarity, which are very fast. The result should show up immediately then. Furthermore, repeated information may affect the weight, although there is no actual impact, but still hope to be solved in the follow-up work.

We also hope that on the basis of this system, we can combine the search content and view frequency of users for machine learning in the future, so as to recommend data to users. This improvement combined with big data can ensure that the whole system is more intelligent.

REFERENCES

1. Martin, G. M. . (1980). The job hunter's guide to the library. Occupational Outlook Quarterly, 24, 6-11.
2. Bianco, D. , & Maki, K. E. . (2015). Job hunter's sourcebook: where to find employment leads and other job search resources.
3. Frakes, W. B. , & Baezayates, R. . (2004). Information retrieval: data structures and algorithms. Prentice Hall, 15(5), 1211 - 1214.
4. Rocchio, J. J. . (1971). Relevance feedback information retrieval. The Smart Retrieval System-Experiments in Automatic Document Processing.
5. Luo, T. , Weck, P. , Sequeira, A. , Tendulkar, N. , Bentov, S. , & Levine, J. . (2008). Job Search Engine and Methods of Use. US.
6. Silverstein, C. , Marais, H. , Henzinger, M. , & Moricz, M. . (1999). Analysis of a very large web search engine query log. Acm Sigir Forum, 33(1), 6-12.