PRIS at TREC 2011 Medical Record Track

Jiayue Zhang, Xueneng Lin, Yang Zou, Shuai Zhu, Jing Xiao,
Weiran Xu, Guang Chen, Jun Guo
School of Information and Communication Engineering,
Beijing University of Posts and Telecommunications
jyz0706@gmail.com

Abstract

Our method to accomplish the Medical Record Track is described in this paper. For ad hoc retrieval, Indri and Xapian are used for indexing, searching, and initial query expansion. The main query expansion is achieved using LSI. The evaluation results show the performance of our system is above the average.

1. Introduction

The goal of the Medical Records track is to foster research on providing content-based access to the free-text fields of electronic medical records. This year the task is to retrieve reports from the given test document collection, which is a set of de-identified medical records made available for research use through a university lab. The retrieval task for the track is an ad hoc search task as might be used to identify cohorts for comparative effectiveness research. The topics given specify a particular disease/condition set and a particular treatment/intervention set, and participants should return a list of visits ranked by decreasing likelihood that the visit satisfies the specification. To accomplish the task, Indri and Xapian are used for index building and query searching first, and query expansion is achieved by Indri, Xapian and LSI.

The remainder of the paper is organized as follows. Section 2 introduces the procedure of ad hoc search with Indri, and section 3 with Xapian. Section 4 describes the query expansion algorithm using LSI, and the evaluation results are given in section 5.

2. Ad hoc Retrieval with Indri

According to the feature of documents and the task's request, we build a system to do the retrieval jobs, which includes preprocessing, building index, term expansion, retrieving and ranking.

2.1 Preprocessing

The test document collection is more than one hundred thousand electronic medical reports. Each report has some "<tag>", but "visitid" is not contained. However, the task uses the "visited" (each report associating a "visitid", while most "visited" having more than one report) as the response unit. That is, our retrieval system must return visitreports_visited's. Since a simple ASCII table called the Report-to-Visit Mapping Key that specifies which reports belong to the same visit is provided, the preprocessing system add the tag "visitid" to each medical report. In order to do that, the set of reports is traversed to get each "check_sum", then a table called the Check_sum to ReportID is built. Finally, we add the tag "visitid" to each report

according to the two tables mentioned above.

As the retrieval task contains the request of ages' distinguish, while the reports don't have the tag "age", we use regular expression to extract patients' age information from each report. This tag will be used when building index. There is some useless information about patients' personal detail in the last part of each report, so we also use regular expression to get and delete them.

2.2 Index building

The indri query language provides a executive program called "buildindex.exe" to build index for given documents. So we use it to do the index building job for all of the test reports. Since the field "report_text" contains each report' main information such as secondary diagnoses, surgical procedure, hospital course and so on, while "chief_complaint" contains the sort of the patient's illness, additionally age is limited in the query condition, we build index for the context of the tag "age". Besides, the result is asked to be returned by "visitid", so an index is also built for the context of "visitid". As a result, fields that need to be built index on include "checksum", "chief_complaint", "report_text", "age", "visitid".

As to the ranking part, tag "age" and "visitid" is needed, and context of the two tags will be returned, so we add "metadata" for them.

2.3 Term expansion

The Indri provides the function of extracting term expansion from the returned documents. Figure 1 depicts the general procedure of our approach.

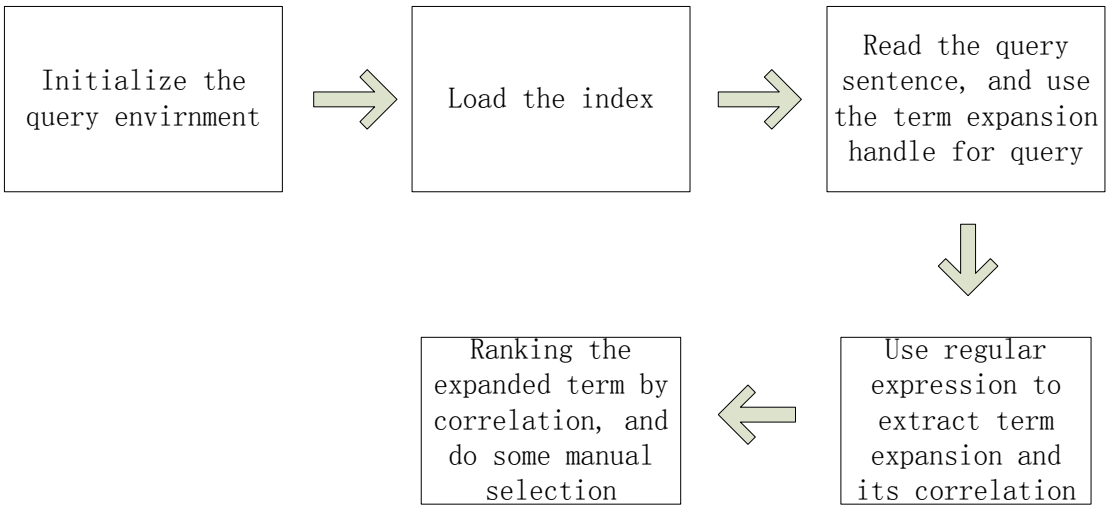

Figure 1. Procedure of extract term expansion

2.4 Designing search sentences

35 topics are given and they specify a particular disease or condition set and a particular treatment or intervention set. We extract the key words from each sentences, then form the search sentences by connecting key words on the regular "and", "or", "not". We design one search sentence for each topic and put them into a text file.

The search system runs two times:

(1) The first time is to extract term expansion for each topic. So this time we connect

the key words with “or” to form the search sentences.
(2) The second time is to return relative “visitid” and rank them. So we connect key words with their expanded term with “or”, then connect them with the result in the first run with “and”. After that we use this search sentences to run the second query.

2.5 Searching and ranking

No more than 1000 “visitid” is requested to submit for each topic, so we set the number of returned document to be 1000. The system run searching according to the search sentences which have been designed above. Relative medical reports are returned and then “visitid” is extracted from each of the reports. Since some different reports have a same visitid, which means this visitid is of high correlation with the topic, we add the score of the reports of this visitid when calculate the score of each visitid. Finally, we rank the visitid according to their score.

Figure 2 depicts the procedure of searching and ranking.

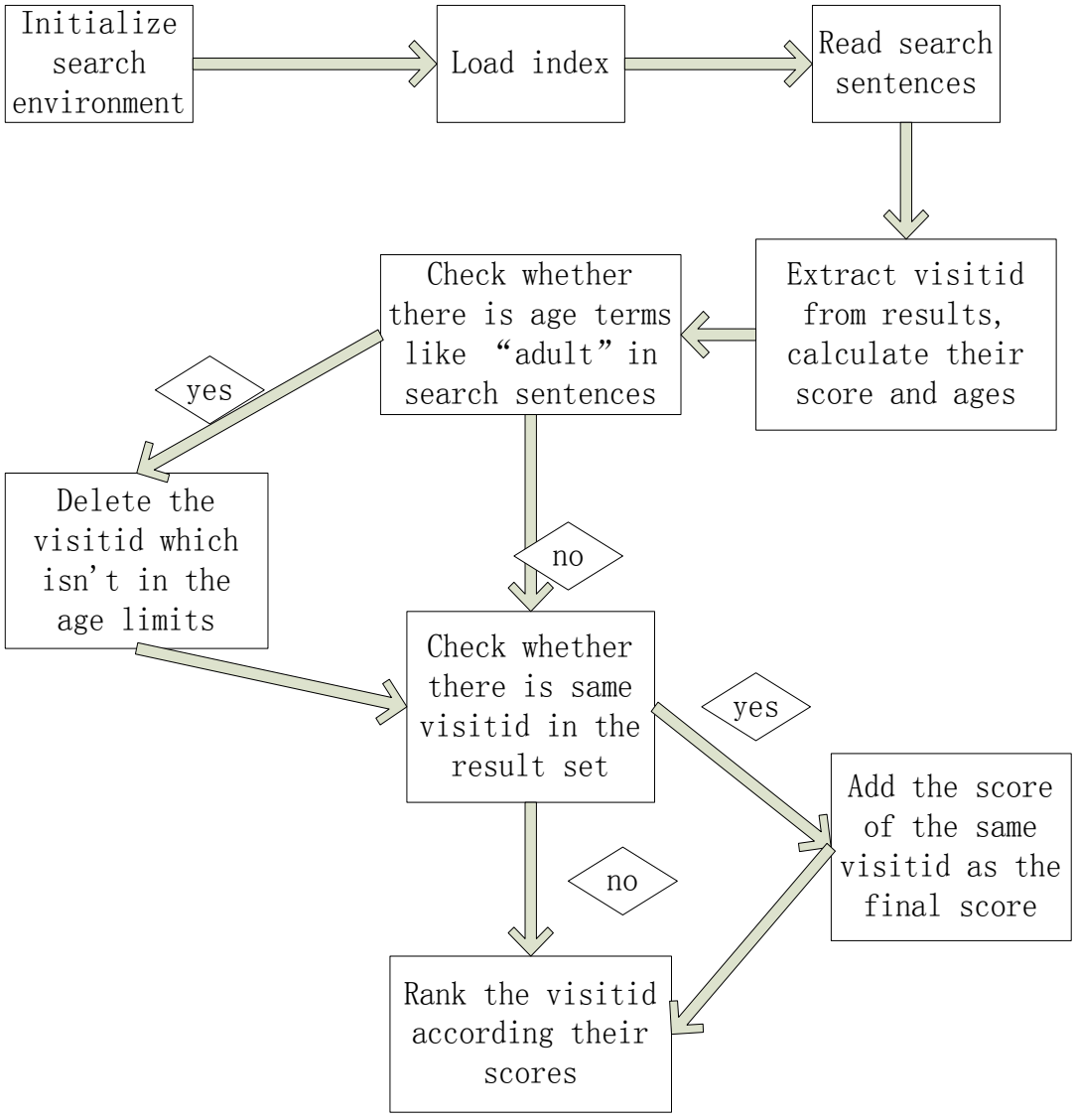

Figure 2. Framework of searching and ranking with Indri

3. Ad hoc Retrieval with Xapian

Indexing, initial query expansion and searching are accomplished on the platform of Xapian.

3.1 Building Index with Xapian

This approach will retrieve related medical reports by searching the test document collection twice. The first step is to build index for the given 92088 medical reports. The preprocessing is the same as the Indri approach, which is getting rid of the regular describe from each reports and keeping only useful message about patients' sickness. We use the Xapian toolkit to build index, which is a searching engine based on probabilistic search model. The default weight is set to be BM25.

The following contents are import when building index with Xapian.

1. "chief_complaint", is the most import key field. Terms in that tag have a relatively higher weight.
2. "report_text", gives main description of a patient, is treated as a normal field, terms in which have lower weight that the former field.
3. "age" in "report_text" is used as a value of documents when indexing by Xapian.

3.2 Preprocessing

This part begins with extracting the key content from topics, which can avoid the noise of useless words. According to the task, we need two kind of information, including symptom and therapy. For example, from "patients with complicated GERD who receive endoscopy", we extract "GERD" and "endoscopy" as key words. Since the test document collection comes from different medical staffs, who have their own writing styles, we added some manual procedure as following to improve the callback of first time search. 1) Add other forms of key words. 2) Expand each abbreviation, for example, "gastroesophageal reflux disease" for "GERD".

3.3 Query Expansion

Since medical text is somehow special with a same terminology may have different forms in medical field, we crawler the introduction "html" of each key word from WIKI, then extract the most frequent adjectives and nouns as the other description of the key word. In order to improve the extracting accuracy, we make up of a black list, which concludes the most frequent but vague words, such as "disease", "treatment", "hospital" and so on. These words don't give particular description on patient's disease, so they will not be treated as expansion terms.

Since each topic describes one kind of symptom or therapy, we connect the key words extracted from them with "or", which can make sure the first searching don't interference the next. For the same reason, other expansion terms are also connected with "or".

3.4 The first searching

We run the searching in the Xapian database with the query sentences designed above. If a topic gets a return set of more than 500, it's labeled with "nta" (need to

adjust), while those that get less than 500 returns are labeled with “nte” (need to expand). Reports returned in this run are of relatively higher accuracy, but if the number is less than 500, that is “nte” topics, need further expanding. Because not enough number means that the topics have many synonymies. As to “nta” topics, we’re also not sure the rank from this searching run is accurate, so the result needs to be adjusted by query expansion.

4. Query Expansion with LSI

There are a lot of professional words in medical reports, and many of them have the phenomenon of polysemy and synonymy, which make the traditional approach of relative term searching based on term co-occurrence not satisfying. So we use LSI (Latent Semantic Indexing) to detecting the inner relation of professional vocabulary.

4.1 LSI

LSI is an information retrieval method which attempts to capture this hidden structure using techniques from linear algebra. It maps the terms and documents to a latent semantic space of low dimension, by which noise in the original vector space is removed. LSI’s core step is singular value decomposition (SVD), which is an import method of extracting matrix’s feature. It’s computed as:

$$A = U \sum V^T \qquad (1)$$

Suppose that A is a N*M matrix of terms and documents, then U is an N*N matrix, Σ is an N*M matrix (diagonal matrix, and its diagonal item is the singular value), V^T is an N*N matrix.

Under normal situation, the total of the first 10%, even 1% singular value of Σ makes up of 99% of all the singular value. That is, we can use the first R of them to describe the A. So, a low rank matrix of A is built to approximate Ar by SVD:

$$A_{m*n} = U_{m*r} \sum_{r*r} V^T_{r*n} \qquad (2)$$

U_{m*r} is SVD term matrix，V^T_{r*n} is SVD document matrix，we use U_{m*r} in this task。

The input for LSI is the returned reports from the first time of searching. The procedure of LSI is as follows:

1. Build the TF-IDF matrix
2. singular value decomposition
3. make use of the matrix from the second step to extract the relative terms for key words

The preprocessing is the same as building index with Xapian. After that we use “genia tagger” to do the pos tagging. Only these tagged to be noun is used to build the TF-IDF matrix.

The GNU Scientific Library, which provides the function of random number, integral computing, and matrix decomposition and so on, is used to do the singular value decomposition. In the result of singular value decomposition, we choose the first 100 to approximate the original term document.

The dealt SVD term matrix U_{m*r} is used to compute the expansion of a key word. Each row of U_{m*r} is the description of one term in a low dimension space. For

example, we find the row of “GERD”, and then compute its cosine value with every other row. By ranking the result, we get the first two as the relative terms, as to this task, they are “regurgitation” and “dysphagia”. Finally, the expansion terms from LSI is connected to the original query sentence with “or”.

4.2 The second searching

Since further expansion terms are obtained from above, we simply use them to search the Xapian database. Now that we have two results from two searching runs, a weight score is needed to find out a final result. It goes as follows:

nta： score(d) = s1*0.7 + s2 * 0.3
nte： score(d) = s1*0.5 + s2 * 0.5 s1 > 0 (3)
nte： score(d) = s1*0.5 + s2 * 1 s1 = 0

“nta” is based on the regular of fine adjustment, which gives more weight to the result from first searching, because it get enough returns from first time ,so the accuracy is relatively higher. “nte” is based on the regular of supplement, which average the result of two searching runs. If a report returned by second searching is not appeared in the first time, s1is set to be 0, while s2 set to be 1.

The whole procedure is shown in Figure 1.

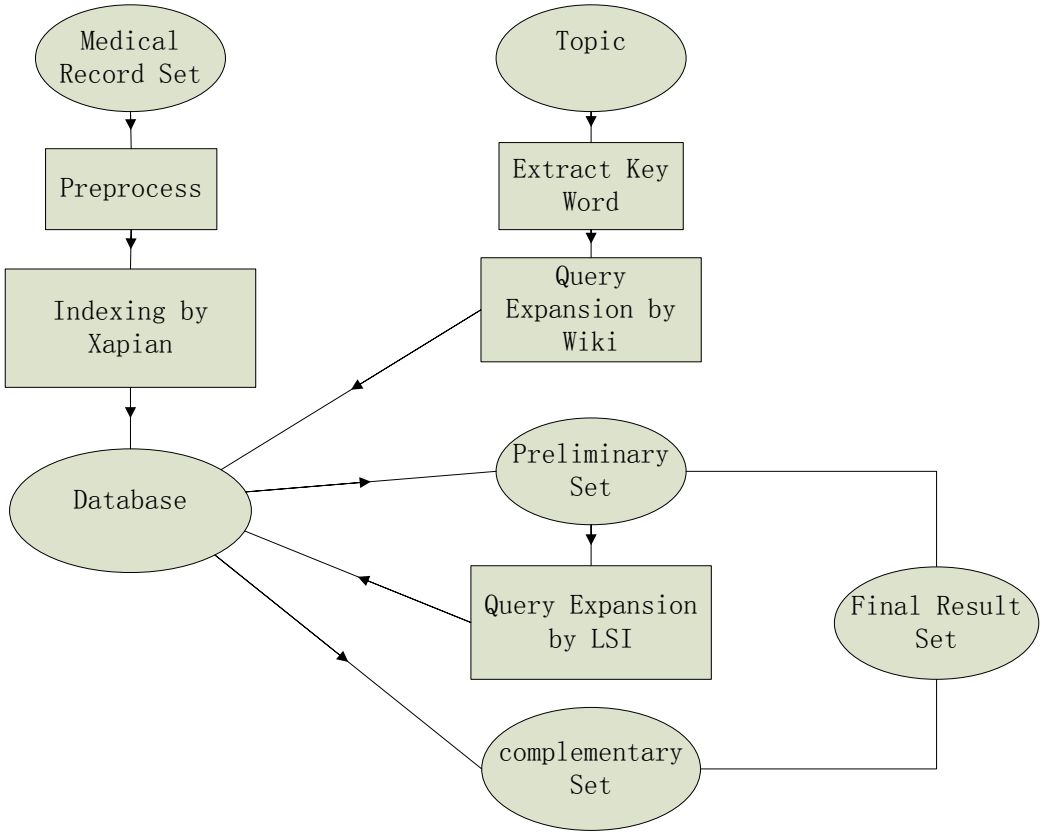

Figure 3. Framework of searching and ranking with Xapian

5. Evaluation Results

The results of our system, the median and the best by three measurements are listed

in table 1. The numbers of topics whose score are higher than the median are shown in table 2. From the evaluation results in table 1 and 2, a conclusion can be reached that our results are between median and best in bpref and P@10, but in R-prec, our result is lower than median. In general, the performance of our system is above the average.

	bpref	*R-prec*	*P@10*
pris	0.474	0.3422	0.5471
Median	0.308718	0.411529	0.476471
Best	0.609471	0.760738	0.876471

Table 1. Evaluation Results

	bpref	*R-prec*	*P@10*
>Median	22	17	17
=Median	3	8	9
<Median	9	9	8

Table 2. Relation between our system results and Median

References

[1] F. Farfan, V. Jrostodos, A. Ranganathan, M. Weiner, MD. XOntoRank: Ontology-Aware Search of Electronic Medical Records.

[2] Thomas Hofmann. Probabilistic Latent Semantic Indexing. In Proceedings of the Twenty-Second Annual International SIGIR Conference on Research and Development in Information Retrieval (SIGIR-99), 1999.

[3] Blei, David M, Andrew Y, Jordan, Michael I. Latent Dirichlet allocation. Journal of Machine Learning Research, 2003.3.

[4] Deerwester, Dumais, S. T., Landauer, T. K., Furnas, G. W. and Harshman, R. A. Indexing by latent semantic analysis." Journal of the Society for Information Science, 41(6), 391-407.

[5] Chen, C., Stoffel, N., Post, N., Basu, C., Bassu, D. and Behrens, C. Telcordia LSI Engine: Implementation and Scalability Issues. In Proceedings of the 11th Int. Workshop on Research Issues in Data Engineering (RIDE 2001): Document Management for Data Intensive Business and Scientific Applications, Heidelberg, Germany, Apr. 1-2, 2001.

[6] Dumais, S. T. "Using LSI for Information Retrieval, Information Filtering, and Other Things". Talk at Cognitive Technology Workshop, April 4-5, 1997.

[7] http://www.lemurproject.org/lemur/IndriQueryLanguage.php

[8] http://www.xapian.org

[9] http://www.lemurproject.org/lemur/indexing.php#IndriBuildIndex

[10] http://ciir.cs.umass.edu/~metzler/indriretmodel.html