PRACTICAL FULLTEXT SEARCH IN MEDICAL RECORDS

Vít Volšička[1, 2*], Milan Blaha[1, 2]
[1] *Institute of Biostatistics and Analyses, Faculty of Medicine, Masaryk University, Czech Republic*
[2] *Institute of Health Information and Statistics of the Czech Republic, Prague, Czech Republic*
[*] *Corresponding author: vit.volsicka@uzis.cz*

ARTICLE HISTORY
Received 28 May 2015
Revised 23 June 2015
Accepted 24 June 2015
Available online 7 July 2015

KEYWORDS
fulltext
apache solr
search
medical records
sql
information retrieval

ABSTRACT — *Performing a search through previously existing documents, including medical reports, is an integral part of acquiring new information and educational processes. Unfortunately, finding relevant information is not always easy, since many documents are saved in free text formats, thereby making it difficult to search through them. A full-text search is a viable solution for searching through documents. The full-text search makes it possible to efficiently search through large numbers of documents and to find those that contain specific search phrases in a short time. All leading database systems currently offer full-text search, but some do not support the complex morphology of the Czech language. Apache Solr provides full support options and some full-text libraries. This programme provides the good support of the Czech language in the basic installation, and a wide range of settings and options for its deployment over any platform. The library had been satisfactorily tested using real data from the hospitals. Solr provided useful, fast, and accurate searches. However, there is still a need to make adjustments in order to receive effective search results, particularly by correcting typographical errors made not only in the text, but also when entering words in the search box and creating a list of frequently used abbreviations and synonyms for more accurate results.*

BODY

Performing a search through text documents is an integral part of the educational process and acquiring new information. Unfortunately, finding relevant information is not always easy, since many documents are saved in free text formats, thereby making it difficult to search through them.

We face the same problem when attempting to scan medical reports, which contain relevant information about the health conditions and treatment of certain patients. These reports are most often saved in free text formats, without any inner structure and parametric values that could facilitate a search. There may be good reason to, at least partially, parameterise and categorise textual medical records. For example, this type of information would help to find relevant groups of patients for clinical studies, analyse treatment methods for certain groups of patients, obtain larger statistics, or find real cases for educational purposes.

FULL-TEXT INDEX

Medical records are usually stored in relational databases, such as unstructured text strings. The basic method, an alias for a naive approach to search in text string databases, means combing through each record, word by word and comparing it precisely with the query. It is not the fastest and best solution, especially when a language is as morphologically complicated as the Czech language. The performance of such a procedure, implemented onto the database, depends on the number of records. With their growing number, the time needed to scan the records increases linearly, which is not very effective. Scanning a large volume of data can easily exceed the time for which the user is willing to wait for the result. There is a need to choose a faster and more effective method. This method is called a full-text search, or an inverted index. It stores words into an index with links to the documents in which these words occur. It is much faster due to a smaller volume of scanned data. Inverted index is also a favourable possibility for inputting complex queries containing more words, with an option to add specifying logical operators.

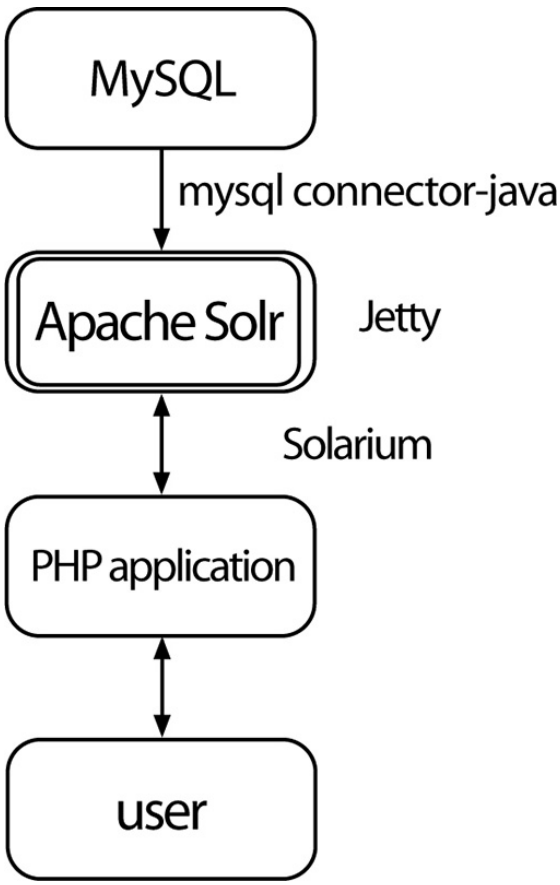

FIGURE 1 Scheme of connection

FIGURE 2 Frontpage

FIGURE 3 Results of search with highlighted keywords

MORPHOLOGY, NORMALISATION AND METHODS

We found out how to solve the performance problem, but not the problem with language morphology, which could be complex and tricky, especially with the Czech language. Words in the text might be found in various forms. Standard search looks for the same form of the word as if it was assigned in the query. If there is not a perfect match, the word is ignored. For example, when we insert the word "lékař" (doctor), we expect to receive records that contain additional words, such as "lékařův", "lékařovo", "nelékařský" etc., therefore finding all forms of word, including variants with various prefixes, suffixes, and plurals.

The next challenge is to deal with diacritical marks found in some languages. Some users write words without diacritical marks, and the computer does not recognise that it is the same word. It is possible to remove diacritical marks entirely, but this could increase the number of irrelevant results.

As we mentioned above, the scanned text should contain various forms of words, which further complicate searches. The solution to this problem is to transform words into their basic forms. This process is called normalisation and we apply it both to the records and to the given query. Consequently, the number of successfully returned words will increase, while the size of the index will decrease due to a smaller amount of indexed words. We will introduce several ways of normalisation.

Lemmatisation

Lemmatisation is the process of transforming a work to the grammatically correct basic form, lemma. For example, the word "vyléčil" (cured) should be transformed to "léčit" (to cure). Homonyms are a weak point in this method, however. Homonyms are words with the same form but with different meanings. If we want to choose the correct meaning of the word, we would have to make a semantic analysis of the whole sentence. But the cost of this process is very high, so it is rarely used. Therefore, the correct approach is to return to all forms, which yields an even lower accuracy.

Stemming

Stemming is the process of eliminating prefixes and suffixes, so the word is reduced to its basic form, or the root. The stem might not necessarily be the same as the morphological root, and might have no meaning. This is the one difference from lemmatisation, which returns a valid word. For example, the root of word "léčit" is "léč." According to the complexity of

the language, conversion into the stem of the word can be algorithmised with a certain quality. This has an advantage over lemmatisation, which needs footing in the base form of a specific vocabulary in order to function. We face two kinds of mistakes when applying the stem algorithm. The first problem is the over-stemming that occurs when two different words are transferred to the same stem. This is considered a false positive error. The second problem is understemming, which occurs when two words that should have the same stem actually have different stems. This is called a false negative error [1]. It has been proved that aggressive stemming algorithms reduce false positive errors, but at the expense of an increase in false-negative errors. Aggressive algorithms behave conversely [2].

N-Grams

An n-gram is a contiguous sequence of n items from a given word extracted from a text. The principle of this method consists in the fact that similar words have a large number of the same sequences. The length of a sequence is typically chosen as either 2 (bigrams) or 3 (trigrams). The value of n is appropriate to test for each language, because it may significantly affect the quality of the result. If n is too small, each word is divided into many n-grams and the query yields a large number of irrelevant results. However, when we increase the value of n, the size of the index will grow exponentially [3]. For example, we could divide the word "nemocnice" (hospital) by bigram into the sequence *N, NE, EM, MO, OC, CN, NI, IC, CE, E* and by trigram: **N, *NE, NEM, EMO, MOC, OCN, CNI, NIC, ICE, CE*, E**, where the asterisk means an empty space. The search query is decomposed into the same size of n-grams and afterwards the number of identical parts is compared. The advantage of this technique is that it provides language independence of the text.

Combination of different approaches

We combine several approaches to receive the best quality outcome. One suitable combination for the Czech language is the cooperation of stemming algorithms, Brute Force and Suffix Stripping. The Brute Force method is based on a "look-up" table containing a construction of state forms of words and their roots. The algorithm enters a query into the table and, in conformity, yields the root of the word. The Brute Force approach is criticised for its lower speed of data processing, as well as for its inability to cover the entire language. Due to the number of word forms in the Czech language, it is unrealistic to expect that all of their forms could be captured. Therefore, it is difficult to design the Brute Force algorithm, because we would need to record a large number of words to achieve an acceptable level of accuracy. However, continuously adding words only improves the accuracy. The advantage of this method is that there is zero risk of over-stemming or under-stemming.

An appropriate complement to this purely vocabulary-based method is the Suffix Stripping algorithm, which removes the Brute Force method's criticized properties. Suffix Stripping works on an application's simple rules for removing prefixes and suffixes to obtain the root word [4]. The pitfall of this approach is that it yields unusual situations, such as, in some cases, changing the word root.

RESULTS – FULL-TEXT SEARCH TOOL IMPLEMENTATION IN APACHE SOLR

The aforementioned theoretical solutions are implemented in a platform called Apache Solr, which was primarily intended for efficient full-text searches. It is an open-source application written in Java. This makes Solr a cross-platform, so it can be deployed on most systems. Other advantages include support in choosing the standardisation of words, removing diacritical marks, speed, and a broad range of settings. Solr needs to run its own server, though; fortunately, "virtual" servers (servlet container) – such as Jetty, for example – are an option.

From the above-mentioned methods, Suffix Stripping was chosen as the most appropriate method for the standardisation of Czech words [5]. It has already been implemented into Solr.

Firstly, for proper functioning we must connect Solr to a database or other repository, where a medical centre has stored its medical records. It is advisable to have the records stored in UTF-8 coding for a proper display and to retrieve characters with diacritical marks; however, Solr works with most databases. Interconnection is based on downloading the appropriate file *-connector-java.jar and entering access data into the configuration file.

The second part is to index records from the database. The appropriate configuration file will specify which database field would be indexed, and how it would be processed and normalised. Based on tests on which algorithm achieves the best results when processing a text written in Czech, we found that Built Stemmer was the best candidate [5]. Solr is natively able to process more than 33 languages. In case of the Czech language, it is worth to add a feature for the removal of diacritical marks. When we want to add new entries to the index, we might not start the whole process over again, but run a process called Delta import.

The last part of the process is to install a library called Solarium. Solarium is not absolutely necessary, but makes typing commands in PHP for communicating with Solr faster and easier. The PHP programming language is the most widely used in the web

scripting language, and as Solr, is not dependent on the platform. We use it to create a web application which will provide a user-friendly interface for searching through the indexed records. The appearance of the web application depends on specific requirements. Solr makes calculations based on proprietary algorithm relevance for each document of the given query, and determines the order of results accordingly. The way of ordering can be changed. Found words can also be highlighted in the text.

CONCLUSION

Performing a search in the given text documents, especially in medical reports, it is not a trivial matter. We encounter several issues that need to be resolved, depending on the morphological complexity of the given language. However, these problems can be solved, and the solutions have been already implemented in applications primarily designed for text searching. One of these applications is the Apache Solr, which includes a built-in support for more than 33 languages, including the Czech language. Additional languages can be added by using the library Hunspell, which provides support for 99 languages (http://hunspell.sourceforge.net/). Apache Solr thus offers a suitable solution for searching through medical records. Thanks to its cross-platform nature, the application can be deployed on existing solutions in medical facilities without the need for major interventions to their systems.

The tool has been implemented and successfully used for searching through medical records stored in a MySQL database. Records themselves might be located in different data stores and, due to the support, might be written in many languages. The application has been pilot-tested and used to scan 54,486 written records in Czech. Solr has dealt with the normalisation of words and with the size of databases very well. Solr provides useful, fast, and accurate searches. An example of a successful search is shown in Figure 3. Solr offers a wide range of options and customisations, so it can be easily modified to suit specific requirements.

The described solution for full-text searches supposes that there are syntactically correct medical records in the database, and that the query is correct. However, typographical errors are very common. Likewise, a mixture of Czech and Latin words and expressions, or common use of abbreviations is often used. This semantic part is rather problematic within computer text analysis, but very important for returning relevant records.

One possible solution to correcting typographical errors is to implement an algorithm called the Damerau–Levenshtein Distance. This algorithm counts the number of steps (insertion, deletion, or substitution of a single character, or a transposition of two adjacent characters) that are needed to retrieve another word. Damerau stated that the operations correspond to more than 80% of all human misspellings [6].

A solution for abbreviations and Latin words consists of creating a dictionary of synonyms and translations. Solr already offers this function. For our future work, it will be necessary to build such functionalities and implement the correction of typographical errors.

Vít Volšička

REFERENCES

[1] Jivani AG. A Comparative Study of Stemming Algorithms. Int J Comp Technol Appl 2011; 2(6): 1930-1938.
[2] Paice C. An evaluation method for stemming algorithms. Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval. Springer-Verlag: New York 1994: 43–50. ISBN 0-387-19889-X.
[3] Coetzee D. TinyLex: Static N-Gram Index Pruning with Perfect Recall. Proceedings of the 17th ACM Conference on Information and Knowledge Management. Association of Computation Machinery: New York 2008: 409–418. ISBN 978-1-59593-991-3.
[4] Kumar D, Rana P. Stemming of punjabi words by using brute force technique. Int J Eng Sci Technol 2011; 3(2): 1351–1358.
[5] Sikora R.. Vyhledávání v českých dokumentech pomocí Apache Solr. Masaryk University: Brno 2012. MSc thesis.
[6] Damerau FJ. A technique for computer detection and correction of spelling errors. Commun ACM 1964; 7 (3): 171–176.