Advances in Information Mining
ISSN: 0975–3265 & E-ISSN: 0975–9093, Vol. 3, Issue 1, 2011, pp-19-25
Available online at http://www.bioinfo.in/contents.php?id=32

AN SQL EXTENSION FOR LATENT SEMANTIC ANALYSIS

CRISTIAN BISCONTI [1], ANGELO CORALLO[1], HOSSAM FARIS [*], SALVATORE TOTARO[1]
[1]Incubatore Euro-Mediterraneo in "e-Business Management", University of Salento, Lecce, Italy
* Corresponding Author: 7ossam@gmail.com

Received: April 14, 2011; Accepted: May 12, 2011

Abstract- Latent Semantic Analysis (LSA) is a powerful statistical theory and method for indexing, retrieving and analyzing textual information. With the increased demand for more efficient databases and computing power, a lot of research and work has focused on the practical side of the LSA. In this paper we introduce a design and implementation of an SQL extension for LSA applications. The developed SQL extension for LSA is aimed to utilized as an easy and simple interface for basic LSA and cosine similarity functions provided for indexing, searching and retrieving large textual information stored in open source relational database management system like PostgresSQL.
Key words - Latent semantic indexing, SQL, Information retrieval

Introduction

Information retrieval and the integration of its tools in relational database management systems have been debated for a long time within the scientific community. Many researchers mentioned number of application fields for such an integration. Accordingly, many different approaches and techniques have been proposed to face these issues [1,2]. The increasing amount of available unstructured textual information documents motivated researchers to develop tools and design methodologies to improve the efficiency and to automate information retrieval systems [3, 4].

Almost all existing DBMS's have a full-text search system nowadays, but sometimes lexical matching methods can be inaccurate when they are used to match a user's query. When users search for required information, some documents which have the potential to be relevant might be missed and not included in the retrieved documents set only because the users are using different words with almost identical or similar meanings (synonyms). Naturally it would be better for users to retrieve information on the basis of a conceptual topic or meaning of a document rather than exact matching.

LSA technique [5] tries to overcome these problems by using a statistical based approach for indexing documents. LSA is a statistical information retrieval method that works with words that have similar meanings or what is called synonyms. LSI is capable of retrieving documents based on the concepts they contain by using statistically derived conceptual indices instead of individual words for retrieval. Thus querying using search and retrieval systems based on LSI techniques can return a set of documents as a result that are semantically similar to the search query even if the results don't share a specific words the search query.

LSA constructs a term-document matrix to identify the occurrence of unique terms within a collection of documents, in this term-document matrix, each term is represented by a row, and each document is represented by a column. Each matrix cell (intersection of row and column), holds the frequency of the appearance of a given term in a given document. LSI then applies local and global weight functions for terms based on term frequencies to reflect the relative importance of the term in each document in the collection, in another words, the term frequency indicates the weight of the term weight in a given document. In this way a document is represented by a set of terms with their frequencies ignoring the exact order of the terms in the document.

LSA then performs a particular mathematical technique called Singular Value Decomposition (SVD) on the matrix to identify the structure in term usage and recognize patterns in the relationships between the terms used across the documents. SVD is a mathematical mapping technique to reduce the number of dimensions for a document's vector in the term space of the matrix, into a low dimensional space to make it more useable and efficient.

As mentioned above, the use of LSA ensures the retrieving of the original semantic structure of the concept space. Accordingly it brings the advantages of improving the performance with respect to synonymy problem [6]. Indeed the traditional retrieval strategies are not able to face the challenge of discovering documents about the same topic that use a different vocabulary. In LSA, same concepts are all likely to be represented by a similar weighted combination of indexing variables. Moreover, while a large number of polysemous words in the query can reduce the precision of a search significantly. The use of a reduced representation in LSA makes it possible to remove some noise from the data. The applications of LSA range from the information retrieval to relevance feedback and information filtering.

LSA technique has been also applied to solve problems related to the disambiguation of the word sense [7, 8], the cross language retrieval [9,10] or to develop semiautomatic tool for document classification [11]. These considerations bring the opportunity of developing an integration of LSA in a DBMS [12] in order to provide an easy access and simple to use SQL extension for LSA applications over huge databases. This paper presents the early results of the implemented integration, specifically the attempt to integrate the main LSA functions as SQL extension in PostgreSQL. This kind of integration gives a chance of LSA technique application as an autonomous additional means for information retrieval through simple SQL commands. Since we already have the availability of the full-text search functionality in PostgreSQL DBMS, in the context of this experimentation we can use both of them together.

This work is divided into three parts: The first part introduces and presents suitable new data types designed in this work for the purpose of programming the LSA functions. The core implementation of LSA functions is described in details in part two. Finally, The third part discusses a real running example of using the implemented LSA functions within SQL queries processed over sampled data stored in an open source DBMS. The purpose of this example is to show the usefulness, powerful and the ease of use of these functions. Moreover, it shows how simply this work can apply the LSA techniques by using LSA functions introduced in this paper on textual information stored in DBMSs without the need of a third-party software.

The introduced extension was developed and implemented in PostgreSQL DBMS, chosen for its level of usage in the research and scientific communities as it supports several important programming languages as C.

Latent semantic data types

The integration of LSA technique in SQL language standard as an extension requires defining new data types different from the primitive data types already present in PostgreSQL. These data types are designed to store all the information related to a given document needed for the LSA. In particular, for each document, we need to know the number of lexemes, their position within the document and their length. In the presented case, this information is embedded in a structure written in C language. We store the lexemes in a memory block contiguous to its structure.

A. LSAVector data type

The new data type LSAVector is desgined so we can store all the information related to a given document that belongs to a specific Knowledge Base (KB). Basically, an LSAVector is an ordered map that associates a value, that is the number of occurrences of a specific word in a specific document, with a key, that is the word that we are considering. For example, if d1 is a document as shown in Figure 1 in which there are only the terms ”alfa”, ”beta”, ”gamma” and ”delta” the s1 LSAVector associated with d1 is a map with ”alfa”, ”beta”, ”gamma”, ”delta” keys and s1[”gamma”] = 3 indicates that in d1 the term ”gamma” appears tree times as shown in Fig. (1).

B. LSAInfoKB data type

By grouping LSAVectors that represent some specific documents, we obtain a KB. It is characterized as a set of vocabularies coming from the combination of all keys of the LSAVector’s involved. For example, consider that s2 is an LSAVector associated with a document d2 reporting only the terms ”alfa” and ”gamma” (s2[”beta”] = s2[”gamma”] = 0) and s1 and s2 are a KB with a vocabulary made by ”alfa”, ”beta”, ”gamma” and ”delta” terms. If we add an LSAVector s3 related to the document d3, that contain the word ”alfa”, ”beta”, ”gamma”, ”iota”, the KB that we obtain as result of combination has a vocabulary made by ”alfa”, ”beta”, ”gamma”, ”delta” and ”iota”. Each LSAVector is a KB, but what we want is to keep separating these two types of data.

KB is a type of data that has vocabulary information and a frequency matrix, named term-document matrix. It is designed as an array of arrays in which the term appearance frequency in each document is stored, but considering more dimensions. This means that if, for example, we create a KB by grouping s1,s2 and s3 LSAVector’s associated with d1,d2 and d3 documents, we obtain a KB K with a vocabulary and a frequency matrix as the one presented in Fig. (2).

In order to proceed the latent semantic analysis we need the U, S and V matrices, obtained from the SVD decomposition of the term-document matrix. The amount of information related to LSAInfoKB data type could be too expensive in term of memory allocation consider a LSAVector like a KB with only one document. Based on these considerations, we created two different data types: the one containing only the necessary information of a single document and the other containing the additional information related to the whole collection of documents also, that is the matrices of the SVD decomposition. For example considering a KB made by our d1, d2 and d3 documents, we obtain a LSAInfoKB type represented as shown in Fig. (3).

Implemented functions for latent semantic indexing

In order to operate with LSA analysis, we need specific functions related to specific tasks. Although Electronic document can exist in different formats, like doc, pdf, ppt or html, we are only interested in the textual information which is represented by the words stored in the files. For this reason we decided to operate on simple text field. In this section we introduce new four implemented functions for LSA applications.

A. LSAVector toLSAVector (text doc)

First of all, we need a function able to translate a simple text field into a LSAVector, that is the data type introduced above. The number of components of this vector is equal to the number of unique terms present in the document; each component is associated to an

integer number that represents the term appearance frequency in the document as shown in Fig. (4).

B. LSAVectorConcat and toLSAVector oveloading

When we group a huge amount of documents, this means many LSAVectors, we obtain a KB. This KB provides us the related vocabulary, that is the set of words present at least one time in one document of the KB, that enables the LSA analysis. An LSAVector may have many representations based on the choosed vocabulary. Then we need to overload the toLSAVector function adding a new argument containing the vocabulary associated to KB that we are considering. The argument vocabulary is essentially another LSAVector data type generated by the KB, since all the information related to vocabulary is already contained within its structure.

• LSAVector toLSAVector(text simpletext, LSAVector vocabulary): this function builds an LSAVector by adding terms in simple text.

• LSAVector toLSAVectorConcat(LSAVector v, LSAVector vocabulary): This function builds a new LSAVector with terms that are in vocabulary.

C. LSAInfoKB toLSAInfo(LSAVector v) and LSAInfoKB toLSAIKB(LSAVector v))

In order to perform latent semantic analysis, we need to get information that must be combined to the KB through the application of a Singular Value Decomposition of the term-document matrix. The term-document matrix is the result of the grouping LSAVector elements based on the same vocabulary. In order to compute this information we created an aggregator function.

An aggregator function in SQL has the same data type for argument op and for results res. This means that, to obtain an LSAInfoKB as result of the aggregator function, we need to aggregate LSAInfoKB. For this reason we need a function to convert a LSAVector in LSAInfoKnowlegdeBase. This function is LSAInfoKB toLSAIKB(LSAVector v).

Now we can use the aggregator LSAInfoKB with a LSAVector like argument casted before using toLSAKB function. The general operation of this aggregator is only an LSAVectorConcat function, while the final operation is to calculate the three matrices needed for the LSA Analisys. The U, S and V matrix allow us to make similarity estimation in SQL statement.

D. double simCos(LSAVector di, LSAVector q, LSAInfo KB)

One important operation in LSA analysis is represented by the estimation of the similarity of a document di, that belongs to KB, according to a given query q. To perform this operation, we created the simCos function. This function takes the lsavectors q and di and translates them into the concept space using the following equations:

$$q_p = S^{-1}U^T q$$
$$d_{ip} = S^{-1}U^T d_i$$

the simCos function uses the S and U matrices stored in the KB LSAinfoMatrix (its third argument). The last step to obtain the real similarity value is to compute:

$$q_p d_{ip}^T$$

In the same way we can estimate the similarity of a document that does not belong to the KB. It is important to point that this document might contain some terms that are not present in the KB, so we could estimate the portion of similarity due to the term (and concept) appearing in our KB.

Power of LSA use in SQL environment: an example

Here we give an example with some applications showing the simplicity of using the implementation of the SQL extension we proposed in this research. Considering the chance to use this implementation to index and search, by natural language process, the same kind of documents whose source could be web site, ftp server, filesystem or samba server, we suppose that a crawler takes .doc, .ppt, .pdf, .html files from these sources, extract the textual information from the structure of the documents and put all the terms within a simple text field database. Moreover, we suppose we have a simple table made as shown in Fig. (5):

• ID: is a numeric value used like primary key
• corpus: is the result of the textual information extraction task accomplished by the crawler during documents catching.
• fileName: stores the name and the path of the document
• fileType: stores the type of the document catched by the crawler
• sourceName: stores the name of the source where the document has been catched by the crawler.

This simple table suggests us that we have more several ways to perform a search on this document. Naturally we can perform a full-text search on corpus field or on fileName field. We can apply a filter on file type. But we can also make a search based on semantic correlations. We can use more different groups of documents to make our KBs. There is no reason that suggest us to use all records in the ”documents” table. By using LSA in SQL environment we have more flexibility in our analysis, we can consider different sets of documents as KB specifies a WHERE condition in our SELECT statement. This means that for all the considered subsets of documents we can estimate similarity between a query and one or more documents. For example we can consider a KB made only of documents that are present on ”myUSBPen” resource, and then we can use SQL-query like:

```
SELECT
LSAInfoKB(toLSAInfo(toLSAVector(corpus)))
FROM documents
WHERE sourceName= 'myUSBPen'
```

Choosing a KB, we can store it in a table for further similarity querying or we can make direct similarity query

by using nested query. Suppose we have a query captured by a simple text field from a client application connected to our database. Obviously a query is exactly a document that can be converted into a LSA vector and used to make similarity estimation. In order to perform a similarity query without a ”materialized” KB we can submit a query like this to our database.

```
SELECT
  simCos ( toLSAVector( corpus ),
         toLSAVector('TextOfTheQuery'),
         KB)
FROM
  documents ,
 (SELECT LSAInfoMATRIX(toLSAVector(corpus))
  FROM docs ) AS KB
```

Obviously, without stored KB, the query is more time-expensive than a query based on a materialized KB. In other words if we store our KB in a table defined like:

```
CREATE TABLE AS
  SELECT
   LSAInfoMATRIX(
         toLSAInfo(toLSAVector(corpus) )
FROM  documents
```

we can evaluate the similarity level with a query like this:

```
SELECT
   simCos ( toLSAVector(corpus),
          toLSAVector('TextOfTheQuery'),
          KB. kb )
FROM documents , KB
```

In order to appreciate the potential of SQL-LSA fusion we can consider the following example. Suppose that we found a query made by a natural language that satisfies a specific knowledge needed for our work. We can constantly update the documents simply using a view based on a semantic query as:

```
CREATE VIEW
  myPersonalKnowledgeCollectionOnElectric
AS simCos( toLSAVector(corpus),
         toLSAVector('TextOfTheQuery'),
         KB. kb )
FROM documents ,
(SELECT LSAInfoMATRIX(toLSAVector(corpus))
 FROM docs ) AS KB
```

The semantic similarity obtained from the LSA analysis can be used within a SQL command as DELETE or INSERT, including a WHERE clause, in order to manage our KB. Suppose we want to prune our KB, deleting or moving the documents that have a level of semantic similarity less than 0.2 with respect to a specific query. This task can be done using the following SQL statement:

```
DELETE FROM documents
WHERE id in
 SELECT
  id,
  simCos(toLSAVector(corpus),
        toLSAVector('TextOfTheQuery'),
        KB,
        kb) < 0 ,2
 FROM
   documents ,
  ( select
     LSAInfoMATRIX(toLSAVector(corpus))
    from docs )as KB
```

Notice that in the query above, the deletion is based on a LSA analysis executed on the fly.The potential showed above is possible because our implementation is considered as an extension of the SQL language.

Future work

Scalability is one of the most important issues in designing software. A significant degradation in system's performance must not happen when the database becomes larger. In the scope of this research, a further work is aimed to be done about the reliability and efficiency of using the implemented SQL extension for real and huge database of textual information, hopefully, with maintained overall system performance.

Conclusion

In this research we proposed a design and implementation of an SQL extension for latent semantic analysis applications. We introduced the data types and functions used to provide the basic LSA operations. Furthermore some real examples were given in order to show the simplicity of using the extension in SQL queries.

References

[1] McHugh J., Abiteboul S., Goldman R., Quass D., Widom J. (1997) *Lore: A database management system for semistructured data. Technical report, Stanford University*.

[2] Manber U., Wu S. (1993) *GLIMPSE: A tool to search through entire file systems. Technical Report TR 93-34, Department of Computer Science, University of AZ, Tuscon, Arizona, http://webglimpse.org*.

[3] Paris L.A.H., Tibbo H.R. (1998) *Information Processing and Management* , 34(23):175190.

[4] Belkin N.J., Croft W.B. (1987) *Annual Review of Information Science and Technology*, 22(9), 110145.

[5] Furnas G.W., Deerwester S., Dumais S.T., Landauer T.K., Harshman R.A., Streeter L.A., Lochbaum K.E. (1988) *Information retrieval using a singular value decomposition model of latent semantic structure. International Conference on Research and Development in IR, New York*.

[6] Deerwester S., Dumais S.T., Furnas G.W., Landauer T.K. and Harshman R.A., (1990) *Journal of the American Society for Information Science*, 41, 391-407.
[7] Gallant I. (1991) *Neural Computation*, 3.
[8] Schutze H. (1992) *Dimensions of meaning, in Proceedings of Supercomputing*.
[9] Landauer T.K. and Littman M.L. (1990) *Fully automatic cross-language document retrieval using latent semantic indexing, proceedings of the Sixth Annual Conference of the UW Centre for the New Oxford English Dictionary and Text Research, UWCentre for the New OED and Text Research, Waterloo Ontario.*
[10] Young P.G. (1994) *Cross-Language Information Retrieval Using Latent Semantic Indexing, Master's thesis, The University of Knoxville, TN.*
[11] Ceglowski M., Coburn, A.,Cuadrado J. (2003) *Proceedings of the IEEE/WIC International Conference on Web Intelligence* (WI03),0-7695-1932-6/03.
[12] Kumaran A. and Haritsa J.R. (2005) *SemEQUAL: Multilingual Semantic Matching in Relational Systems, DASFAA*.
[13] Rosario Barbara (2000) *Latent Semantic Indexing: An overview - INFOSYS 240 Spring 2000*.
[14] April Kontostathis (2007) *International Conference on System Sciences, 2007. HICSS 2007.* 73.

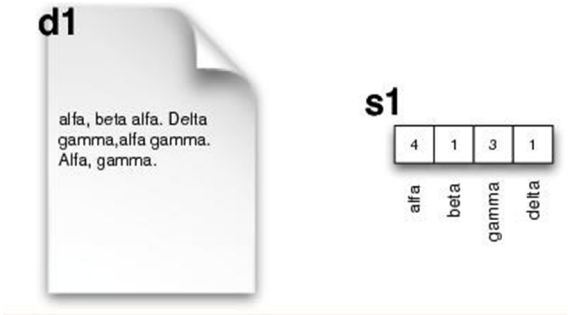

Fig. 1- d1 document and related LSAVector s1

	alfa	beta	gamma	delta	iota
s1	4	1	3	1	0
s2	1	0	0	3	0
s3	5	1	5	2	3

Fig. 2- vocabulary and term-document matrix for the KB made by s1,s2 and s3 LSAVectors

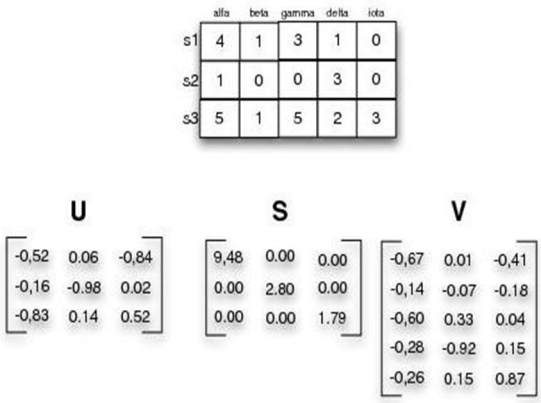

Fig. 3- term-document matrix and U,S and V matrix for the Knowlesdge Base made by s1,s2 and s3 LSAVectors

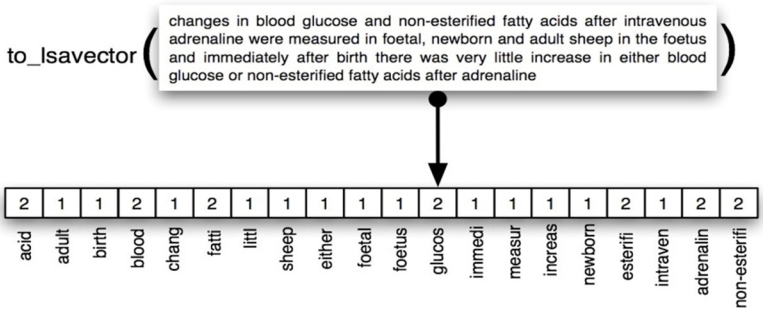

Fig. 4- function tolsavector applied to a simple text document

ID	Corpora
1	correlation between maternal and fetal plasma levels of glucose and free fatty acids . correlation coefficients have been determined between the levels of glucose and ffa in maternal and fetal plasma collected at delivery. significant correlations were obtained between the maternal and fetal glucose levels and the maternal and fetal ffa levels .
2	changes of the nucleic acid and phospholipid levels of the livers in the course of fetal and postnatal development . we have followed the evolution of dna, rna and pl in the livers of rat foeti removed between the fifteenth and the twenty-first day of gestation and of young rats newly-born or at weaning . we can observe the following
3	surfactant in fetal lamb tracheal fluid . lambs delivered by cesarean section with intact fetal circulation have a fluid filling the trachea . analysis revealed that this fluid contained material high in surface activity in lambs delivered near term, but less surface activity in premature lambs administration of 10 per cent oxygen to the ewe for 1 hour prior to
4	placental and cord blood lipids.. comparison in a set of double ovum twins, a stillborn and a live-born . 1. determinations of phospholipid, total and free cholesterol, triglyceride and nefa have been made on placental tissue and cord blood n a set of double ovum twins, one stillborn and one live-born .

Fig. 5- Table format