

Automatic Software Clustering via Latent Semantic Analysis¹

Jonathan I. Maletic, Naveen Valluri

The Department of Mathematical Sciences Division of Computer Science

The University of Memphis

Campus Box 526429 Memphis TN 38152

jmaletic@memphis.edu

Abstract

The paper describes the initial results of applying Latent Semantic Analysis (LSA) to program source code and associated documentation. Latent Semantic Analysis is a corpus-based statistical method for inducing and representing aspects of the meanings of words and passages (of natural language) reflective in their usage. This methodology is assessed for application to the domain of software components (i.e., source code and its accompanying documentation). The intent of applying Latent Semantic Analysis to software components is to automatically induce a specific semantic meaning of a given component. Here LSA is used as the basis to cluster software components. Results of applying this method to the LEDA library and MINIX operating system are given. Applying Latent Semantic Analysis to the domain of source code and internal documentation for the support of software reuse is a new application of this method and a departure from the normal application domain of natural language.

1. Introduction

This work describes some of the initial findings of applying Latent Semantic Analysis to software. Latent Semantic Analysis (LSA) [1, 8] is a corpus-based statistical method for inducing and representing aspects of the meanings of words and passages (of natural language) reflective in their usage. The method generates a real valued vector description for documents of text. This representation can be used to compare and index documents using a variety of similarity measures. By applying LSA to source code and its associated documentation candidate components or descriptions can be compared with respect to these similarity measures. Here the vector description is used to cluster components into semantically related groups.

Results have shown [1, 8] that LSA captures significant portions of the meaning not only of individual words but also of whole passages such as sentences, paragraphs and short essays. Basically the central concept of LSA is that the information about word contexts in which a particular

word appears or does not appear provides a set of mutual constraints that determines the similarity of meaning of sets of words to each other.

LSA relies on a Single Value Decomposition (SVD) [11, 10] of a matrix (word \times context) derived from a corpus of natural text that pertains to knowledge in the particular domain of interest. SVD is a form of factor analysis and acts as a method for reducing the dimensionality of a feature space without serious loss of specificity. Typically the word by context matrix is very large and (quite often) sparse. SVD reduces the number of dimensions without great loss of descriptiveness. Single value decomposition is the underlying operation in a number of applications including statistical principal component analysis [6], text retrieval [1, 4], pattern recognition and dimensionality reduction [3], and natural language understanding [7, 8].

2. The LSA Model

Latent Semantic Analysis is comprised of four steps [2, 8]:

1. A large body of text is represented as an occurrence matrix ($i \times j$) in which rows stand for individual word types, columns for meaning bearing passages such as sentence or paragraphs (granularity is based on problem or data), that is (word \times context). Each cell then contains the frequency with which a word occurs in a passage.

2. Cell entries $freq_{i,j}$ are transformed to:

$$\frac{\log(freq_{i,j} + 1)}{\sum_{1-j} \left(\left(\frac{freq_{i,j}}{\sum_{1-j} freq_{i,j}} \right) * \log \left(\frac{freq_{i,j}}{\sum_{1-j} freq_{i,j}} \right) \right)}$$

a measure of the first order association of a word and its context.

3. The matrix is then subject to Singular Value Decomposition (SVD) [4, 6, 10, 11]:

$$[ij] = [ik] [kk] [jk]'$$

¹ This paper appears in the 14th IEEE ASE'99, Cocoa Beach FL, Oct. 12-15th, pp. 251-254

where $[ij]$ is the occurrence matrix, $[ik]$ and $[jk]$ have orthonormal columns, $[kk]$ is a diagonal matrix of singular value where $k \leq \max(i,j)$. In SVD a rectangular matrix is decomposed into the product of three other matrices. One component matrix describes the original row entities as vectors of derived orthogonal factor values, another describes the original column entities in the same way. The third is a diagonal matrix containing scaling values such that when the three components are matrix multiplied, the original matrix is reconstructed.

4. Finally, all but the d largest singular values are set to zero. Pre-multiplication of the right-hand matrices produces a least squares best approximation to the original matrix given the number of dimensions, d , that are retained. The SVD with dimension reduction constitutes a constraint satisfaction induction process in that it predicts the original observations on the basis of linear relations among the abstracted representations of the data that it has retained.

The result is that each word is represented as a vector of length d . Performance depends strongly on the choice of the number of dimensions. The optimal number is typically around between 200 and 300 and may vary from corpus to corpus, domain to domain. The similarity of any two words, any two text passages, or any word and any text passage, are computed by measures on their vectors. Often the cosine of the contained angle between the vectors in d -space is used as the degree of qualitative similarity of meaning. The length of vectors is also useful as a measure.

One of the criticisms of this method, when applied to natural language texts is that it does not make use of word order, syntactic relations, or morphology. But very good representations and results are derived without this information [1]. This characteristic is very well suited to the domain of software, both source code and internal documentation, because much of the informal abstraction of the problem concept may be embodied in names of key operators and operands of the implementation, word ordering has little meaning. Source code is hardly English prose, but through the use of selective naming much of the high level meaning of the problem at hand is conveyed to the reader (programmer/developer). Internal documentation is also found to be commonly written in a subset of English [5] that may also lend itself to the methods utilized by LSA.

3. Clustering Software Assets

Experiments into how domain knowledge is embodied within software are being investigated in an empirical

manner. The work presented here focuses on using the vector representations to compare components and classify them into clusters of semantically similar concepts.

Two readily available software systems are being used as data for the experimentation: LEDA [9] (Library for Efficient Data structures and Algorithms) and MINIX [12] (Operating System). LEDA is a library of the data types and algorithms for combinatorial computing and provides a sizable collection of data types and algorithms in a form that allows them to be used by non-experts. LEDA is composed of over 140 C++ classes. MINIX a simple version of the UNIX operating system and widely used in university level computer science OS courses. It is written in C and consists of approximately 28,000 lines of code.

A semantic space was generated for each of these software systems using LSA software [1]. A dimensionality of 250 is used for both spaces. This gave a large reduction in dimensionality as both data sets have vocabulary counts of around a couple thousand. The granularity of the source code input to LSA is of interest at this point. In the applications of LSA to natural language corpuses, typically a paragraph or section is used as the granularity of a document. Sentences tend to be too small and chapters too large. In source code the analogous concepts are module or function. Obviously, statement granularity is too small and a file containing multiple functions too large. Given that LEDA is written in C++ using an object-oriented methodology the granularity chosen is that of the class LEDA has 144 source code documents. For MINIX the function level is used along with some whole files that are made up of data structure definitions. This resulted in 498 source code documents for MINIX.

For each of the systems a simple parsing of the source code is done to break the source into the proper granularity and remove any non-essential symbols. Comment delimiters and many syntactical tokens are removed as they add little or no semantic knowledge of the problem domain. Also the LSA method inherently will see such ubiquitous tokens such as a semi-colon as a totally non-discriminating feature between to source code components. That is, every meaningful C++ component contains a semi-colon. Therefore the variance of this feature is very low (most likely zero) thus if two components have a semi-colon then nothing can be said about their similarity.

A number of different types of experiments are being conducted with these semantic spaces including: clustering of components based on similarity; comparing

similarity between a component and external documentation; and component matching from natural language queries. The focus here is on the clustering or grouping of related software documents based on the similarity measure produced by LSA.

Cluster: Linear Lists
_d_array.h
_dictionary.h
_p_queue.h
_prio.h
_sortseq.h
dictionary.h
f_sortseq.h
p_dictionary.h
p_queue.h
prio.h
sortseq.h

Table 1. An Example Cluster in LEDA. This cluster gives grouping of all linear lists.

To cluster the source code documents they are partitioned based on similarity value λ with respect to the other documents in the semantic space. A document is added to a cluster if it is at least λ similar to any one of the other documents in the cluster. This strategy attempts to group as many documents together within the given similarity range. The similarity measures are computed by the cosine of the two vector representations of the source code documents. The

similarity value therefore has a domain of [-1, 1], with the value 1 being "exactly" similar.

Cluster: Delaunay's Method
float_delaunay.h
rat_delaunay.h

Table 2. An Example Cluster in LEDA. This cluster gives a grouping of Delaunay's triangulation method.

Tables 1 and 2 give two clusters generated by using a similarity measure of 0.7 (approximately 45 degrees) and above. This level of similarity is first thought to be a reasonable value, though actually quite low, and secondly found to be reasonable through

qualitative examination of the grouped classes.

Table 1 is a group of classes dealing with linear lists. Each of the classes has much in common with underlying semantics and general domain. This is also one of the bigger clusters generated for the LEDA data set. Table 2 gives the classes in a smaller, more typically sized cluster that relates to a particular method. This grouping is also much more obvious from the naming of the classes/files.

Singleton clusters made up most of the clusters uncovered. Figure 1 gives a break down of how many

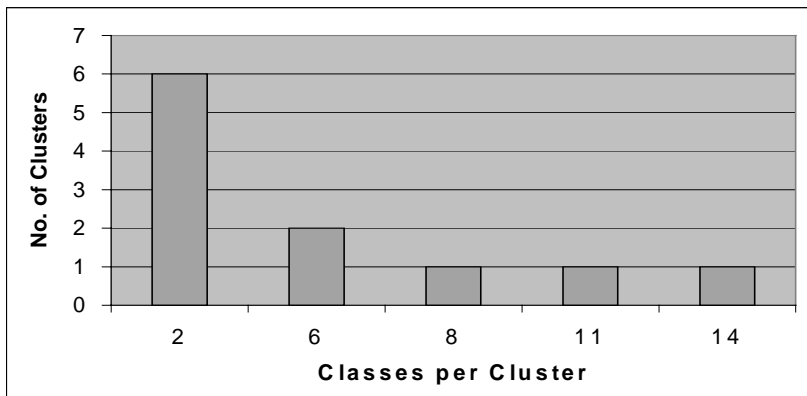
Figure 1. LEDA Cluster distribution. Number of clusters with size greater than one: 11. Number of singleton clusters: 87 (not shown).

clusters of each size in the data set. There are 87 classes that are not within the similarity range to any other class thus resulting in a cluster of size one. But 11 clusters occurred with size greater than one.

The overview for the MINIX semantic space is given in Figure 2. Here the number of none singleton clusters is 47 with many of them fairly small (size 2). It is interesting to note that there are a couple of fairly large clusters with sizes of 16, 20, and 25. A complete analysis of the data generated for MINIX is still under way and more qualitative analysis is needed to validate this work.

4. Discussion of Results

The clusters produced by LSA represent an abstraction of the source code based on a semantic similarity. The grouping produced in this automated fashion reflect the reality. Pieces of source code that had large amounts of semantic similarity were grouped together and modules with no relation to others remained apart. The clusters in the LEDA library seem to reflect class categories, that is groups of related classes the function on similar concepts or solve common types of problems. By grouping similar classes (or components) together a broader understanding of the software system may be achieved. Understanding of one of the components in the cluster implies some basic understanding of the others. In the MINIX system the clusters are quite different due to the different methodology and programming language utilized. In this case the clusters seem to represent sets of documents that represent a class or abstract data type. Basically, the larger clusters are typically composed of one or two data structure definitions and a number of functions that utilize these data structures.



A large number of the software documents in each of the systems are grouped by themselves as singleton clusters.

Eighty-seven of the 144 documents in LEDA are singletons and 311 out of 498 in MINIX. These represent stand-alone components that have little semantic relationship with the rest of the system. Given the fact that both of these systems are products of good design and programming practices the high percentage of stand-alone documents is not surprising.

The next step in the research will be to expand the sets of software systems being examined. It will most likely be prudent to select some very orthogonal domains and some closely inter related domains to assess the application of LSA. Each domain must have a number of example components with varying degrees of internal and external documentation, which will give a good spectrum of the particular domain and result in a valid representation of the domain knowledge. Assessing the relative quality and validity of the constructed semantic spaces is the main goal of this research. The semantic spaces will be used to locate and classify components in the selected problem domains. This will support searching large pieces of software for components that match to any of the known domain types in the knowledge base. Utilizing this similarity measure in some way to produce a useful metric for cohesion is also under investigation.

LSA is a powerful tool to assist in supporting many of the activities of the software reuse process. The reuse activities that can be directly supported by LSA are the identification and locating reusable components; classification and storage of components; retrieval and indexing; and understanding. Research is underway to assess LSA ability to support these activities.

5. References

[1] Berry, M.W., Dumais, S.T., O'Brien, G.W., "Using Linear Algebra for Intelligent Information Retrieval", *SIAM: Review*, 37(4), 1995, pp. 573-595.
 [2] Deerwester, S., Dumais, S.T., Furnas, G.W., Landauer, T.K., Harshman, R., "Indexing by Latent Semantic Analysis", *Journal of the American Society for Information Science*, 41, 1990, pp. 391-407.

[3] Duda, R.O., Hart, P.E., *Pattern Classification and Scene Analysis*, Wiley, 1973.
 [4] Dumais, S.T., "Latent Semantic Indexing (LSI) and TREC-2" in Proceedings for The Second Text Retrieval Conference (TREC-2), Harman, D.K. (Eds), March 1994, pp. 105-115.
 [5] Etzkorn, Letha H., Davis, Carl G., "Automatically Identifying Reusable OO Legacy Code", *IEEE Computer*, 30(10), October 1997, pp.66-72.
 [6] Jolliffe, I.T., *Principal Component Analysis*, Springer Verlag, 1986
 [7] Landauer, T.K., Dumais, S. T. "A Solution to Plato's Problem: The Latent Semantic Analysis Theory of the Acquisition, Induction, and Representation of Knowledge", *Psychological Review*, 104(2), 1997, pp. 211-240.
 [8] Landauer, T.K., Laham, D., Rehder, B., Shreiner, M.E., How Well Can Passage meaning Be Derived without Using Word Order? A Comparison of Latent Semantic Analysis and Humans", Proceedings of the Nineteenth Annual Conference of the Cognitive Science Society, 1997, pp. 412--417.
 [9] The LEDA Manual Version R-3.7, <http://www.mpi-sb.mpg.de/LEDA/index.html>, Last accessed: 4/29/1999.
 [10] Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P., *Numerical Recipes in C, The Art of Scientific Computing*, Cambridge University Press, 1996.
 [11] Strang, G., *Linear Algebra and its Applications 2nd Edition*, Academic Press, 1980.
 [12] Tanenbaum, A., Woodhull, A., *Operating Systems Design and Implementation*, Prentice Hall, 1997.

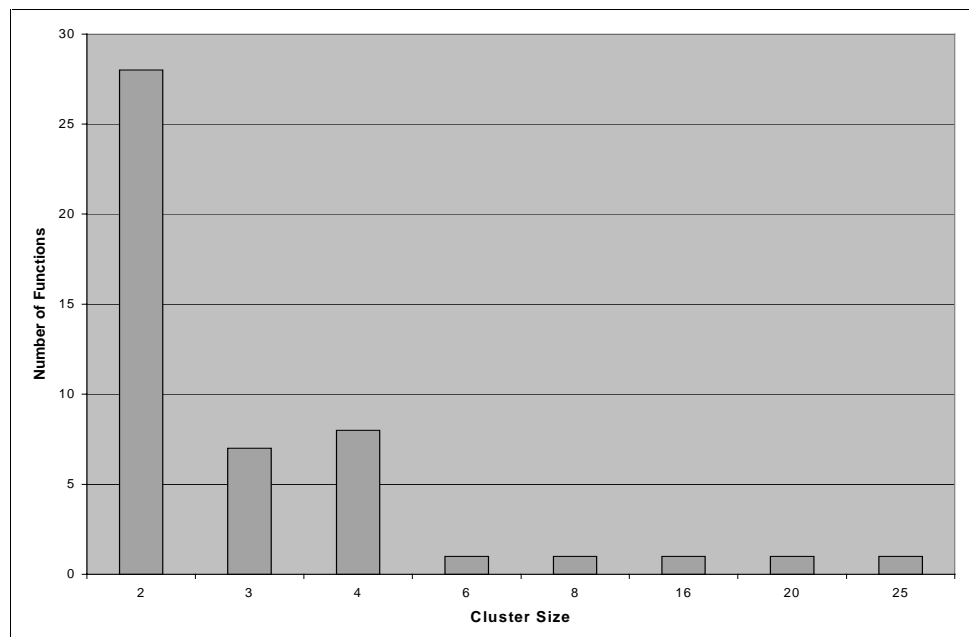


Figure 2. Cluster distribution for MINIX.
 Number of clusters with size greater than one: 47. Number of Singleton clusters: 311 (not shown).