Quesada, J. (in press) Creating your own LSA space.
In T. Landauer, D. McNamara, S. Dennis & W. Kintsch (Eds). Latent Semantic Anlysis: A road to meaning. Erlbaum.

Creating your own LSA space

Jose Quesada

Carnegie Mellon University

Abstract

This chapter is both a how-to and a review of the theoretical problems of computing SVDs to create LSA spaces. The chapter has two parts: The first one covers the software-specific issues. This chapter reviews some of the current alternatives for SVD-capable programs and environments. It also describes how to preprocess the text, perform the decomposition, and interpret possible errors using these programs. The second part focuses on software-independent issues, and discusses how to gather the corpus and select the appropriate weighting scheme and number of dimensions.

Introduction

Although many researchers use LSA today, not that may run their own SVDs to create their own spaces. There are several reasons to this: (1) There exists a public web site that is sufficient for most uses of LSA, and the group at the university of Colorado, Bouder has had programmers that could do some “extra jobs” when researchers requested features not available in the website. (2) In some occasions, it is difficult to collect a representative corpus of text. (3) Memory requirements. Current SVD methods place the entire matrix in memory to invert it (see Martin and Berry, this volume). That places a significant bottleneck in the size of the LSA analyses that one can run, and was the main reason why LSA analyses could not be run on personal computers. However, nowadays the memory bottleneck is no longer an issue, since a consumer-level PC can be configured with more than enough memory to run a large SVD. (4) Computer expertise required. Traditionally, computers with a memory large enough to run SVD ran UNIX. Also, the numerical computation support needed for large SVDs was available only under machines using UNIX. Not that many people in psychology had the expertise needed to run or desire to learn that operating system, not available on personal computers. Linux was available for PCs since early nineties but its user base was small. This situation has changed. On one hand, both apple and IBM-compatible PCs have reached a point where UNIX-based OSs are available and mainstream (MacOS-X and Linux, respectively). Most of the classical and current work on LSA takes place on UNIX machines. One reason is that the UNIX philosophy is text-oriented: the fact that UNIX relies on a command line interface where small utilities are chained together makes it ideal to process text. another reason is that in the past, machines with memories large enough to carry out LSA analyses were large mainframes. Nowadays, LSA analyses can be performed on personal computers under any operating system. However, learning UNIX will help any researcher that is interested in working with text. This chapter focuses on implementations of SVD programs for this OS. The researcher that still opts for staying within a Microsoft windows framework can still be served by all the commercial alternatives (Matlab, Mathematica, R, general programming languages) that will be described below.

This Chapter's main objective is to de-mystify the process of creating your own LSA space. As we will see, all that it takes is general knowledge of some programming language (described below) and an off-the-shelf computer system. The chapter is divided into Software-related issues and theoretical issues. The fist section will present the advantages and problems posed by all the different alternatives in the software world to run an SVD. The second section will focus on decisions regarding selecting the number of dimensions, the proper corpus, and weighting schemes (see Landauer, this volume for an introduction of these terms). These are software-independent. The third section concludes.

Software-related issues

There are three basic steps to create an LSA space and work with it: (1) parsing text, (2) computing an SVD, and (3) manipulating the vectors so one can compute similarities between the passages one is interested in. The ideal situation is that one can do the three of them with the same tool. However a combination of tools may prove to be better than a single one, because it affords more flexibility. For example, a programming environment A may be very well suited to parse text, but not for handling matrices and vectors. The opposite may be true for the programming environment B. In that situation, it might be advantageous to combine A and B. This section presents a review of some current possibilities to carry out the three steps.

Parsing text

Parsing is breaking the input text stream into useable tokens or units, usually words. There are many decisions and issues to address when breaking a text into tokens. To name a few: filtering (i.e. removing HTML tags), common words (i.e. remove or not remove common words), word length (i.e. How long do you want to allow words to be? Do you want to truncate or not the words that exceed the limit?), keep or not keep punctuation marks, numbers, etc., and document boundaries (i.e. What delimiter signifies a document boundary?). For example, we may decide that we are not interested in punctuation marks and numbers. Then, strings such as '18-hole" would be parsed as "hole" and strings such as "end," would be parsed as "end". Some natural language processing (NLP) libraries (e.g. GATE, http://gate.ac.uk/), provide functions to perform different types of parsing. Unfortunately, there is no universally accepted way to parse text, so different packages differ in the options they offer and the output they produce. This is important because the same corpus parsed with slightly different parser parameters will render different LSA spaces. It is highly encouraged that researchers compare their parsing with older ones when trying to replicate previous experiments.

At this point, it is important to introduce the distinction between types and tokens. A type is the class (e.g., word) that the parser uses as unit, and a token is each occurrence of that class. For example, in the sentence "I have one brother and one sister", the type "one" is reprensented by two tokens. The number of types in the corpus is usually the largest dimension in our word x context matrix, and thus it will determine the largest number of dimensions that we can ask in the SVD program (although it could be the number of documents; see Martin and Berry, this volume). An obvious first step in text analysis is to compute a frequency table (that is, how many tokens there are per type in the corpus). Frequency counts will depend on the parsing decisions. For example, words separated by a hyphen (e.g., high-quality) would be counted as one or two words depending on our decision to make the hyphen a valid character. Note that all the posterior processing (context-type matrix computation, weighting, SVD, etc) will be determined by the parsing. This is important since one of the most common problems when trying to replicate previous work is to reproduce the parsing exactly. This constitutes an exercise in reverse engineering unless we have the original parser used and a detailed list of the parameters.

The rest of this section explains several possible choices to parse text, commenting on the advantages and disadvantages of each.

Using a specialized language with regular expressions. A regular expression (abbreviated as regexp, regex or regxp) is a string that describes or matches a set of strings, according to certain syntax rules. For example, the regular expression ‘a*’ matches all strings that start with the letter ‘a’. A good option is to use a specialized language for the parsing. Perl (http://www.perlfoundation.org/) is probably the best at this job, although any language that implements regular expressions is suitable. The advantages of using a general language such as Perl are: (1) Perl has a Gigantic standard library. The Comprehensive Perl Archive Network (CPAN) contains thousands of libraries for text-related tasks, plus some other tasks such as talking to databases, crawling the web, etc. Crawling the web, parsing the text and talking to databases are three tasks that are very related to corpus compilation. Having an appropriate corpus is a prerequisite for creating an LSA semantic space. (2) Perl provides excellent support through mailing lists, books, and online tutorials. Perl is very popular in the NLP community, so chances are that somebody has solved the problem you face before you and has posted the solution. (3) Perl is multi-platform, available in mostly any known platform.

Perl was originally designed to parse text, although it has outgrown this purpose. The disadvantages of using Perl for parsing are hard to find. One of then is (1) speed (as it is an interpreted language). Some (2) Maintainability of the code. Perl scripts tend to be hard to maintain; in fact, Perl is the only language the author knows that has a book on the sole topic of increasing the maintainability of code (Scott, 2004). and (3) Perl is most efficient for small projects; coordinating different people to work on a large-scale application is normally better done on a language with strict typing and a more exigent implementation of Object-Oriented (OO) ideas. All these comments are also applicable to newer scripting languages such as Ruby and Python, which are newer and build from the ground-up as OO languages.

Using Telcordia tools: mkey. Telcordia is the company (formerly part of Bell labs) that originally developed Latent Semantic Indexing (LSI), and patented it for information retrieval. Telcordia owns the patent now, and they request researchers to fill a form where they acknowledge that this technology cannot be used for commercial purposes before they share this software. It can be obtained from http://lsi.research.telcordia.com/. The Telcordia tools are UNIX-only. They are designed with the UNIX philosophy in mind (small tools that do only one thing, very well, and can by concatenated together to do more complex tasks).

The best way to explain the basic mkey flags is to use an example. The following line is a possible use of the tool:

```
mkey –k10000 –c stop-list –l 2 –m 20 –M100000 docs.rawForm > docs.parsed
```

The “-k10000” allows 10000 unique words to be used; “-l 2” eliminates words of length less than 2; the “-m 20” truncates words of length greater than 20 and the “-M100000” allows a maximum line length of 100000 characters.

Mkey can use a stop-list, which is a list of words that will not be included in the results. The file should contain one word per line. The Telcordia tools come with an example stop-list, but there are many other available on the web. One can specify function words (high-frequency words that carry little meaning) in that list. Most uses of LSA do not require a stop-list, since the weighting will take care of those high-frequency words. The “-c stop-list” flag indicated that stop-words should be taken from the file “stop-list”.

Note that some of the parsing options are designed to reduce the number of types (example, truncating long words or removing very short words). This is a vestigial influence of the design decisions made in the time mkey was created to optimize very smalls amounts of memory. None of these decisions has psychological, nor practical, relevance nowadays. The Telcordia tools write a file called RUN_SUMMARY with all the options used in the parsing, so replicating old parsings should be easy.

Using General Text Parser (GTP). GTP (Giles, Wo, & Berry, 2003) replicates the mkey parsing for backwards compatibility. GTP could be considered the reference program for LSA analyses since. It is a very large program. Contrary to what its name indicates, GTP is not only a parser: it actually can run an SVD at the end of the process. GTP is 100% C++ code[1].

GTP "has increased in the number of command line arguments to an almost unbearable point. Even expert users have a difficult time keeping all the options straight. A wordy dialogue about each option would be as unbearable as the number of options available." (Giles et al., 2003, p. 460). However, the chapter by Giles et al. documents very well all

[1] Although a new java version has been developed. The JAVA version includes a graphical user interface (GUI) that helps solving the problems with the multitude of command line options of previous versions

the options available, files written, etc. The reader is referred to that chapter for a more extensive description of the features available.

GTP has the following advantages: (1) it has compact source code, because the program is only one application that encompasses all the needed functionality to parse the text (and even run the SVD). (2) GTP provides good debugging and error handling. (3) GTP is actively maintained by the authors. (4) There is java version that comes with a graphical user interface.

However, GTP has some shortcomings. (1) It uses the operating system's sort command. Sort is a UNIX utility that takes input (e.g., text) and returns a sorted list. Thus it is exposed to any known problems of this tool. In some operating systems, sort may create large temporary files. (2) Although the error handling is better than the one in the Telcordia tools, there are still several situations in which interpreting GTP errors may be difficult to interpret.

An important warning should be issued against using 8-bit characters. Both the Telcordia tools and GTP are sensible to this problem. If your text contains accents or any other special characters, the Telcordia tools and GTP will not be able to use it. It is recommended to remove it during the parsing.

Using Matlab and the Text to Matrix Generator library (TMG, Zeimpekis & Gallopoulos, 2004). Matlab (MathWorks, 2004) is a common tool to manipulate matrices. However, Matlab's text processing capabilities are reduced. TMG is a Matlab toolbox that was designed to mimic the parsing that is standard in the information retrieval conferences such as TREC. This just happens to be the parsing described in the previous section. There are several advantages to using TMG: (1) all the steps (parsing, SVD, vector handling) can take place within the same environment: Matlab. (2) TMG has a graphical user interface. (3) Matlab's sparse matrix package makes all the matrix operations really easy. One can normalize, apply weights, etc with simple commands.

The disadvantages associated with TMG are the following: (1) It does not have as many options for manipulating text as some other parsers. (2) At this moment the software has not been tested with large corpora such as TASA, and several errors are possible.

Using R (R-foundation Team, 2004). R is an environment and programming language for statistical computation. There exist a set of Tools for Textual data Analysis (TTDA library, Muller, 2004) that specializes on corpus processing in R. The advantages of using R for parsing are many: (1) one has access to many recently-developed statistical techniques (e.g., support vector machines, Bayesian networks, etc.). This is because many of the statitstic departments where new ideas are developed use R, so the best way to use the most modern, up-to-date approach to statistics is to use R as well. (2) R also implements regular expressions, making it easy to process text. (3) There is an excellent mailing list on R, where expert answer problems and post their solutions.

The disadvantages of parsing within R are: (1) R (as Matlab) is a matrix-oriented programming language and was not designed from the bottom-up to process text. For example, R places all objects in memory, and this might be a limitation when working with large amounts of text. (2) R has a steep learning curve. (3) The TTDA library is still in initial stages, so it might not be as reliable as a final version.

As a summary, the options to parse text for an LSA analyses are very diverse. Most of the work done in the past with LSA used proprietary unix tools, but currently many packages and programming languages offer the features needed to parse text. The tools used in the past determine the parsing that should be done today if the experimenter wants to compare her results with the ones published.

Computing an SVD

There are two ways of getting an SVD done once we have an adequate parsing of the corpus: (1) implementing your own, or (2) use the existing programs. Implementing a new program for sparse SVD is not recommended, as the testing would have to be really exhaustive.

A number of warnings should be issued about computing SVDs. They have been particularly relevant when working using GTP/pindex, but should be considered when using any other method.

Checking the existence of empty documents in the corpus. If the program finds a document that, after filtering, has no words in it, it will still add a column with all zero frequencies to the words-contexts matrix. This may change the SVD solution obtained. It is advised to remove all empty documents. Empty documents may not be obvious to detect. They may appear, for example, (1) when the string used as document separator is repeated accidentally, and (2) when a document contains no valid keywords after the parser traverses it (due to stop lists, character exclusion list, etc).

The number of dimensions that are requested is not necessarily the number of dimensions that the program returns. Some researchers may be surprised finding that their LSA spaces do not contain exactly the same number of dimensions that were requested. This is due to the nature of the Lanczos algorithm (see Martin and Berry, this volume). Corpora with more dimensions than the number requested can be easily fixed by removing the extra dimensions, so it is always better to generate spaces with more dimensions than needed, and then use only a smaller number.

Creating the space is normally an iterative process. Generating a new space is an elaborate, time-consuming process. Normally, the experimenter tries several combinations of parameters (such as number of dimensions, parsing options, and weighting). The final goal of explaining behavioral data that serve as a benchmark for the model may be obtained with different approaches, so one should be ready to run more than one SVD on the same dataset.

The next step for this section is to describe the possibilities for getting SVDs on sparse matrices.

Mathematica (Wolfram Research, 2004). After version 5 Mathematica has sparse arrays, and a singular value decomposition function that works on them. Mathematica is a mainly symbolic environment, which is not exactly what is needed, but its numerical capabilities are certainly getting better with newer versions and make it a solid alternative. As always, external libraries that run SVDs efficiently can be linked.

Matlab (MathWorks, 2004). After version 6 sparse matrices are supported. The implementation of sparse SVDs in Matlab does not use the Lanczos algorithm. Alternative packages based on the Lanczos method can be found (e.g., PROPACK, Larsen, 2000).

R (The R foundation, open source). It can be linked to LAPACK and other sparse matrices libraries. R provides a sparse library called sparseM (Koenker & Ng, 2004). However, it seems to be in state of development and no new updates have been seen in the last year. There seems to be a faction of R contributors that would want to write an alternative one, with different design decisions. At this moment, it seems to be advisable to just wait and see what happens with R and sparse matrices. Given that the R community is very active, there could be changes in the near future that make R a strong alternative. Linking external libraries is possible as well.

GTP and Pindex. (part of the Telcordia tools). Original software developed at Bell Labs to perform SVDs for information retrieval. It is based on the public domain LAPACK library, but has been refined over time. Pindex is operating-system specific, and it will only run under UNIX. Pindex uses system calls and pipes to bind together a large collection of smaller programs that do the parsing and SVD on an specified corpus. The program in charge of the parsing, mkey, has been described in the previous section. Pindex is not maintained and poorly documented. However, most of the current LSA work at the University of Colorado, Boulder is still performed using pindex. GTP is a C++ version of pindex. In the case of GTP, the sources are available, there is documentation and it is maintained. GTP can do the parsing and the SVD, or one can parse the corpus independently, create a sparse matrix, and use GTP to run an SVD on it. For details on its use and concrete options, see Giles, Wo and Berry (2003, table 27.2).

Today, the reference SVD package is GTP. The spaces that are accessible to the general public have all been created with the University of Tennessee implementation of the Lanczos SVD algorithm in GTP/pindex (see below). For a complete description of the Lanczos method of running an SVD on sparse matrices, see Parlett (1998) and Martin and Berry (this volume). For For reference on the options available and general overview of its use, see Giles, Wo and Berry (2003).

Operating with vectors

Most of the time the experimenter has to handle vectors and do operations with them. Several design decisions can be made depending on the needs. For example, one can place all the vectors in memory if the operations concern the entire space (e.g., nearest neighbor operations). In other circumstances, it will be best to leave the vectors on disk and read them only when they are needed (e.g., when we are interested in comparing a few pairs only). These different approaches can save a lot of computation time.

The 'Telcordia' suite. This suite contains several utilities to handle vectors. For example, *Syn* compares sets of vectors using cosines or Euclidean distances, *tplus* combines several vectors into one (what is called a *pseudodoc)* by averaging or summing over them, and *getkeydb* converts from words to unique ids (rows in the matrix). The reader is referred to the manual pages for additional information on these tools' functionality.

Matlab, Mathematica, and R. Matlab, Mathematica, and R are all matrix-oriented programming languages. Once we have our matrices in Matlab format, all the vector operations are trivial. The standard is to place all vectors in memory, although it is always possible to read them sequentially from disk or use the available interfaces to relational databases.

Software-independent (theoretical) issues

This second section will review the most common issues on the creation of LSA spaces: the selection of the corpus, weighting and dimension optimization.

Size, properties of the corpus

In a very strict sense, a corpus is part of the hypothesis that the experimenter test. Implicitly, it is assumed that the contents of the average participant's memory must be some reflection of that corpus. Because that, corpus selection is a very important part of the process of creating an LSA space. There are two common errors that are easily committed when creating LSA spaces:

(1) *To use a very small corpus*. One can use general corpus statistics to determine what is an appropriate corpus size. According to Zipf's law (Zipf, 1949) the frequency of use of

the nth-most-frequently-used word in any natural language is approximately inversely proportional to n. If this is true, there must be an asymptote in the distribution. We could use this fact as an additional rule to determine corpus size: one has a corpus big enough when, for any new learning experience added, the probability of adding a new type is so low that is negligible. In other words, we can interpret any new utterance as a function of our collection of previously seen types, without having to add new ones.

(2) *To use a non-representative corpus*. As an extreme example, an experimenter could imagine to create a corpus of nineteenth-century literature to represent the knowledge of his undergraduate participants. However, given the age difference, this corpus is probably not very representative of what they have read.

An interesting property of corpus-based theories of cognition (such as LSA) is that they cannot be tested independently of the corpus. Imagine that we collect a corpus, run and SVD on it, and use the resulting space to predict human similarity judgments between certain words. Imagine that the model does not explain the data very well. Is it that the model's processes are unrealistic, or is it that the corpus is not very representative? In this situation, those two factors are confounded. A possible solution is to test the same model with different corpora and different tasks. If the models explains the judgments' variance across different situations, we have more convincing evidence of the psychological reality of the model.

Weighting

LSA uses a transformation of the raw frequency of word-context coocurrence (see e.g., Landauer & Dumais, 1997) that is called weighting. The objective is to diminish the influence of words that are extremely frequent and that do not carry much meaning. These are called function words (e.g., “the”, “a”). Different weighting schemes have been tried with different success. The most common weighting scheme is the log of the local term frequency and the inverse of the entropy of the term in the corpus. This weighting scheme gets substantial improvements in both information retrieval applications (e.g., Dumais, 1991, this volume; Harman, 1986) and cognitive modeling (e.g., Landauer & Dumais, 1997).

$$P_{ij} = \frac{keyLocalFrequency}{keyGlobalFrequency} \qquad (1)$$

$$localWeight = \log_2(keyLocalFrequency + 1) \qquad (2)$$

$$globalWeight = 1 + \frac{\sum_{j}^{ncontexts} P_{ij} * \log_2 P_{ij}}{\log ncontexts} \qquad (3)$$

The expression above (inverse entropy) gives more weight to words that convey more information. It takes values between 0 and 1, being smaller for higher entropy terms. The inverse entropy measure estimates the degree to which observing the occurrence of a component specifies what context it is in; the larger the entropy of a key is, the less information its observation transmits about the places it has occurred, so the less usage-defined "meaning" it has.

To exemplify this fact, imagine a corpus with 1000 passages. Imagine the very extreme case of a key K that appears once in each passage. Then, knowing that we have the key K in a new passage, A, in which we are interested, the presence of key A does not help us much in telling apart the meaning of this passage. This fact is reflected in the weighing that K gets 0.

$$P_{ij} = \frac{1}{1000} = .001 \qquad (4)$$

$$GlobalWeight = 1 + \frac{\sum_{1}^{1000} .001 * \log_2 .001}{\log_2 1000} = 1 + \frac{1000 * \frac{1}{1000} * (\log_2 1 - \log_2 1000)}{\log_2 1000} = 1 - 1$$

The final weight for each key is the multiplication of the local weight and the global weight. For an empirical comparison of different weighting functions, see Harman (1986)..

Number of dimensions

Determining the number of dimensions of a statistical models is a crucial problem. In Multidimensional Scaling (MDS) there is a measure of “goodness of fit” called stress. If one plots number of dimensions and stress, there is usually a decreasing function that can be used as a guide to select the optimum dimensionality: this is called the elbow rule, because one looks for the ‘elbow’ in the curve (Cox & Cox, 2001). However, Several researchers show that there is an optimum dimensionality for LSA spaces (e.g., Dumais, 2003, figure 2; Landauer & Dumais, 1997, figure 3).LSA does not seem to show a monotonic decreasing curve as MDS. The peaked profile that LSA usually shows is common with other statistical models where dimensionality is a free parameter, with the optimal model being detailed enough to capture the information available in the dataset, but not so complex as to be fitting noise.

There is currently no method for automatically selecting the optimum dimensionality of LSA spaces. However, in related algorithms some advances have been made. Model selection has been applied to MDS by Lee (2001). Model selection has also been applied to principal component analysis (PCA, Minka, 2001). Note that in both the MDS and the

PCA cases, probabilistic formulations of the algorithms were created to apply the model complexity ideas. A probabilistic formulation of LSA (e.g., pLSI, Blei, Ng, & Jordan, 2001; Hofmann, 1999) produced the family of topics models (Steyvers and Griffiths, this volume). In the topics models, Griffiths and Steyvers (2004) proposed a method to automatically select the number of topics, which are the equivalent to LSA dimensions.

A standard practice to select the dimensionality is to use some external validation criterion, such as performance of humans in an certain task. Researchers manipulate systematically the number of dimensions and validate the 'behavior' of the resulting space against human data. For example, Landauer and Dumais (1997) found that a dimensionality of around 300 was the best for solving the TOEFL test, and plenty of other experiments have confirmed that there must be is some interval (around 100 – 1500, e.g., Landauer, 2002) where most tasks are performed optimally. However that is not necessarily the case. For example, Laham (personal communication) showed that in an LSA space for the Myers corpus (using the textbook by Myers, 1995) the accuracy solving a psychology test increased with dimensions.

There are at least three reasons to argue against general procedures to determine the optimal dimensionality. (1) The optimal dimensionality is task dependent. That is, an space may perform best at task A with dimensionality *n*, but the optimal dimensionality for task B may be different from *n*. (2) The optimal dimensionality is content-dependent.

Remember that LSA postulates that the collection contains a representative sample of the experience of a single human. The collection can be biased towards a particular type of knowledge. Then, two corpora representative of two people may be radically different in contents, and there is no reason to believe that the dimensionality of the spaces that best describe them is similar. (3) The optimal dimensionality is size-dependent: big (encyclopedia-like) versus small (small-heart) spaces have completely different properties. In Figure 1, it can seen that in general higher dimensionality implies smaller mean cosine, and smaller variance too. The graph shows large corpora, such as TASA (3 M. words) and MetaMetrics (2 Billion words), and small corpora of a few thousand words: Mesoamerican, Energy, and smallHeart (see the LSA website for descriptions). Small corpora are more sensible to isolated groups of objects that are similar only to their selves, and not to the rest of the corpus. For example, in the smallHeart corpus there is a spike of cosines around zero. They correspond to a small set of documents that are not about the heart, but about the Manati (south American aquatic mammal) that were probably placed in that corpus by accident. These documents form a cluster of their own, and the cosines between all other documents and these are really small.

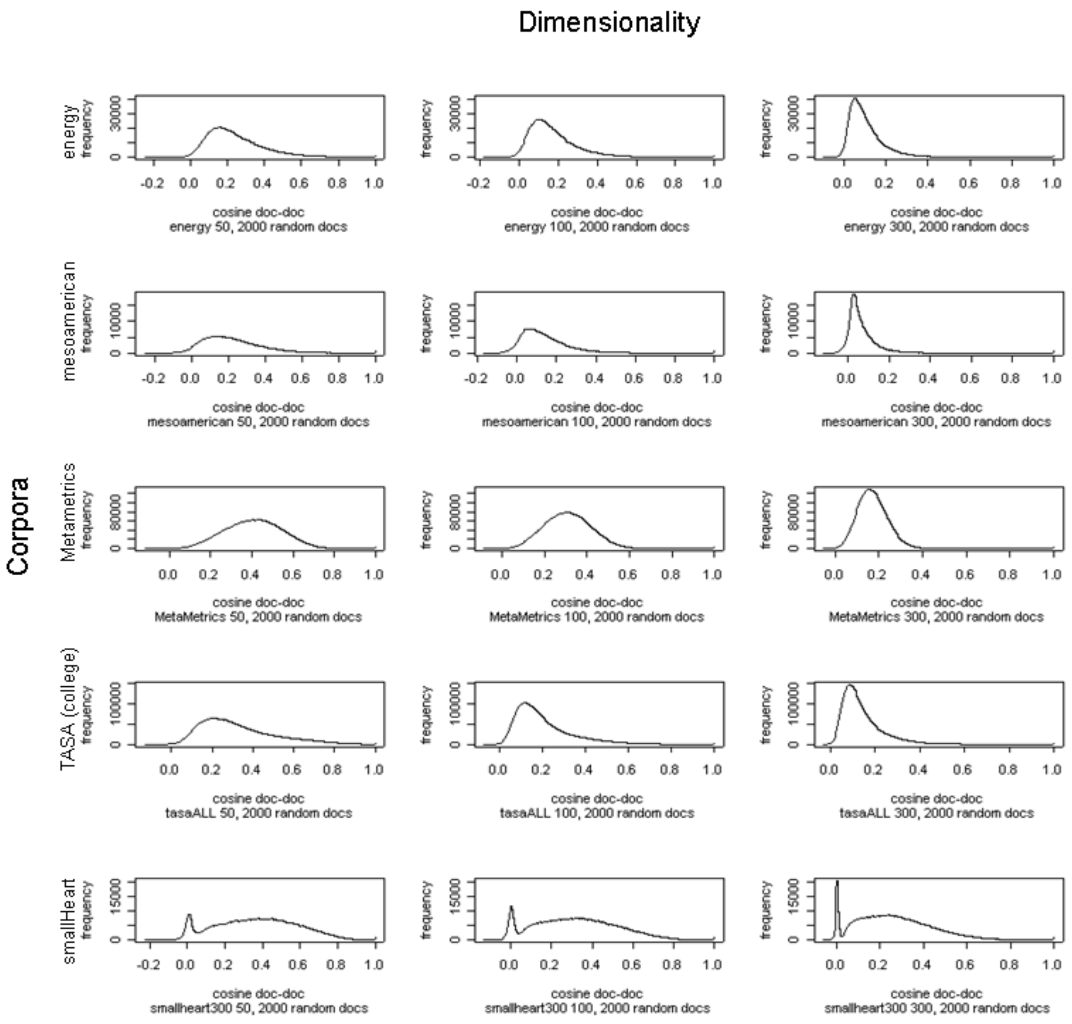

Figure 1: Distribution of cosines in the document to document space at dimensions 50, 100, 300 for different corpora.

Concluding remarks

This chapter has reviewed the current software options to perform three steps to create LSA spaces: (1) parsing the text, (2) computing an SVD, and (3) operating with vectors. A number of problems and general observations have been commented on the processes. There are other dimensionality reduction techniques, reviewed in the chapter by Hu et al. (this volume). Some of the problems described in this chapter are common in those other techniques. For example, the parsing issues are important for the topics model (Steyvers and Griffiths, this volume).

This chapter tried to demystify the process of creating an LSA space. If more researchers can create and use LSA spaces, chances are that novel statistical theories can be put forward and compared with LSA using the same corpora.

Blei, D., Ng, A. Y., & Jordan, M. I. (2001). Latent Dirichlet Allocation. *Advances in Neural Information Processing Systems 13*.

Cox, T. F., & Cox, M. A. A. (2001). *Multidimensional Scaling*. London: Chapman & Hall.

Dumais, S. (1991). Improving the retrieval of information from external sources. *Behavior Research Methods, Instruments and Computers, 23*(2), 229-236.

Dumais, S. (2003). Data-driven approaches to information access. *Cognitive Science, 27*(3), 491-524.

Giles, J. T., Wo, L., & Berry, M. W. (2003). GTP (General Text Parser) Software for Text Mining. In H. Bozdogan (Ed.), *Statistical Data Mining and Knowledge Discovery* (pp. 455-471). Boca Raton: CRC press.

Griffiths, T. L., & Steyvers, M. (2004). Finding scientific topics. *Proceedings of the National Academy of Sciences*.

Harman, D. (1986). An experimental study of the factors important in document ranking. *In Association for Computing Machinery Conference on Research and Development in Information Retrieval, New York: Association for Computing Machinery*.

Hofmann, T. (1999). Probabilistic Latent Semantic Indexing. *Proceedings of ACM SIGIR'99*, 50-57.

Koenker, R., & Ng, P. (2004). SparseM (R-package) (Version 0.54): The R foundation.http://cran.r-project.org/src/contrib/Descriptions/SparseM.html

Landauer, T. K. (2002). On the computational basis of learning and cognition: Arguments from LSA. In N. Ross (Ed.), *The psychology of learning and motivation* (Vol. 41, pp. 43 - 84): Academic Press.

Landauer, T. K., & Dumais, S. T. (1997). A solution to Plato's problem: The Latent Semantic Analysis theory of the acquisition, induction, and representation of knowledge. *Psychological Review, 104*, 211-240.

Larsen, R. M. (2000). Lanczos bidiagonalization with partial reorthogonalization (PROPACK).http://soi.stanford.edu/~rmunk/PROPACK/

Lee, M. D. (2001). Determining the dimensionality of multidimensional scaling representations for cognitive modeling. *Journal of Mathematical Psychology, 45*(1), 149-166.

MathWorks. (2004). Matlab. Natick, MA: The MathWorks.http://www.mathworks.com/

Minka, T. P. (2001). Automatic choice of dimensionality for PCA. *Advances in Neural Information Processing Systems, 15*.

Muller, J.-P. (2004). tools for textual data anaylsis (R-package).http://wwwpeople.unil.ch/jean-pierre.mueller/

Myers, D. G. (1995). *psychology, 5th ed.* New York: Worth Publishers, Inc.

Parlett, B. N. (1998). *the symmetric eigenvalue problem* (2nd ed.): SIAM.

Scott, P. (2004). *Perl Medic: Transforming Legacy Code*: Addison-Wesley.

Team, R. D. C. (2004). R: A language and environment for statistical computing (Version 2.0). Vienna, Austria: R Foundation for Statistical Computing

Wolfram Research, I. (2004). Mathematica (Version 5.0). Champaign, Illinois: Wolfram Research, Inc.

Zeimpekis, D., & Gallopoulos, E. (2004). Text to Matrix Generator.http://scgroup.hpclab.ceid.upatras.gr/scgroup/Projects/TMG

Zipf, G. K. (1949). *Human Behaviour and the Principle of Least-Effort*. Cambridge MA: Addison-Wesley.