Communications of the Association for Information Systems

Volume 41 Article 21

11-2017

A Guide to Text Analysis with Latent Semantic Analysis in R with Annotated Code: Studying Online Reviews and the Stack Exchange Community

David Gefen
Drexel University, gefend@drexel.edu

James E. Endicott
University of Colorado Boulder

Jorge E. Fresneda
Drexel University

Jacob Miller
Drexel University

Kai R. Larsen
University of Colorado Boulder

Recommended Citation

Gefen, David; Endicott, James E.; Fresneda, Jorge E.; Miller, Jacob; and Larsen, Kai R. (2017) "A Guide to Text Analysis with Latent Semantic Analysis in R with Annotated Code: Studying Online Reviews and the Stack Exchange Community," *Communications of the Association for Information Systems*: Vol. 41 , Article 21.
DOI: 10.17705/1CAIS.04121
Available at: https://aisel.aisnet.org/cais/vol41/iss1/21

Communications of the Association for Information Systems

Tutorial ISSN: 1529-3181

A Guide to Text Analysis with Latent Semantic Analysis in R with Annotated Code: Studying Online Reviews and the Stack Exchange Community

David Gefen
LeBow College of Business
Drexel University
gefend@drexel.edu

James E. Endicott
Leeds School of Business
University of Colorado Boulder

Jorge E. Fresneda
LeBow College of Business
Drexel University

Jacob Miller
LeBow College of Business
Drexel University

Kai R. Larsen[1]
Leeds School of Business
University of Colorado Boulder

Abstract:

In this guide, we introduce researchers in the behavioral sciences in general and MIS in particular to text analysis as done with latent semantic analysis (LSA). The guide contains hands-on annotated code samples in R that walk the reader through a typical process of acquiring relevant texts, creating a semantic space out of them, and then projecting words, phrase, or documents onto that semantic space to calculate their lexical similarities. R is an open source, popular programming language with extensive statistical libraries. We introduce LSA as a concept, discuss the process of preparing the data, and note its potential and limitations. We demonstrate this process through a sequence of annotated code examples: we start with a study of online reviews that extracts lexical insight about trust. That R code applies singular value decomposition (SVD). The guide next demonstrates a realistically large data analysis of *Stack Exchange*, a popular Q&A site for programmers. That R code applies an alternative sparse SVD method. All the code and data are available on github.com.

Keywords: Text Analysis, Latent Semantic Analysis (LSA), IS Research Methods, Measurement, Metrics, SVD, Sparse SVD.

This manuscript underwent editorial review. It was received 11/01/2016 and was with the authors for 5 months for 2 revisions. Oliver Müller served as Associate Editor.

[1] David Gefen was the lead author with Kai Larsen. The three PhD students contributed approximately the same to the tutorial. Their names appear alphabetically.

Disclaimer

In this annotated code guide, we provide readers with enough functional knowledge to be able to run and understand text analysis using LSA in R. This annotated code contains functions and the parameters to them that our teams at Drexel University and at Colorado University, Boulder, found most applicable. We encourage readers to refer to the CRAN R LSA Package[1] and to the CRAN R LSAfun Package[2] for additional functions and additional parameters to the functions the annotated code discusses. In the case of analyzing large semantic spaces, other packages may be necessary, including the basic framework for text analysis packages in R, *tm*, and *RSpectra* (a wrapper for the Spectra library that can perform truncated SVD as well as use sparse matrices).

Code and Data

Readers can find the code and corpora used in this guide at: https://github.com/jakemiller3/LSATutorial

1 LSA

1.1 Context and Opportunities

At the philosophical core underlying LSA (and, indeed, many other text-analysis methods) is that text embeds knowledge not only by conveying information explicitly through sentences but also implicitly through how words co-occur with each other. That implicit knowledge can be extracted and revealed, at least in part, through text analysis. That is, the very tendency of specific words to occur together or to occur in the context of another specific word may reveal some relationship such as a synonym or an idea between those words that the language enshrines. The words “mate” and “companion” are examples of the type of synonym one might discover through text analysis. But, there is more to text analysis than just revealing synonyms. Sometimes, the co-occurrence of words may also reveal ideas. As an example, think of the words “black” and “white”. These two words often occur together across documents. One of the many ideas embedded in that pair of words is one of a dichotomous distinction in a shared context. Another is racism, which demonstrates how ambiguous this implied knowledge is and how difficult and subjective its interpretation can be. We show another case, more in the MIS context, in the code snippet in Section 4.3 where we demonstrate, based on Q&A in *Stack Exchange*, that one might apply different programming languages to different coding contexts.

As Gefen and Larsen (Forthcoming) note, it is possible to partly replicate the technology acceptance model (TAM) (Davis, 1989) based on newspaper articles alone (even though none of the TAM items actually appeared in either of the two corpora that they examined) because ideas are embedded into language through word co-occurrences. Those co-occurrences are enough to statistically reconstruct the measurement model that factors together the perceived ease-of-use items into one factor, perceived usefulness into another, and use into a third. Moreover, apparently, the words associated with “usefulness” and “use” are so frequently tied together in English that even the expected path in TAM from the perceived usefulness of an IT to its acceptance or use becomes a matter of the English language and not only experience with a particular IT. (That being said, Gefen and Larsen show that surveys that relate to actual experience with an IT do provide significantly better fit indices and results. A key point in that paper was that the lexical closeness influence of words on questionnaire responses could be controlled for statistically.)

This paper proceeds as follows: in Sections 1 and 2 we discuss what LSA is, what is does mathematically, what has been and can be done with LSA. We also look at how LSA relates to other text analysis methods and review the LSA process. In Section 3, we introduce annotated code that walks the reader through the LSA process with three detailed examples. In Section 4, we discuss the potential and limitation of LSA, including issues of validity and reliability. Finally, in Section 5, we conclude the paper.

[1] https://cran.r-project.org/web/packages/lsa/lsa.pdf
[2] https://cran.r-project.org/web/packages/LSAfun/LSAfun.pdf

1.2 Text Analysis in LSA in a Nutshell

The underlying idea behind latent semantic analysis (LSA) is that co-occurrences of terms (e.g., words)[3] across many documents (e.g., book chapters or paragraphs) suggest that those terms are somehow related in that they are either synonymous or reflect a shared latent concept. Terms can be related to one another in LSA even if they do not co-occur in the same document as long as both terms co-occur with shared other terms. LSA represents terms internally as vectors of a given rank (number of dimensions) based on a transformation of the co-occurrence matrix. The co-occurrences of terms across the documents may also indicate that the documents too, and not only the terms within them, can be factored into groups based on those co-occurrences. Observing the co-occurrence of one set of terms in one group of documents and the co-occurrence of another set of terms in another group of documents may suggest the existence of two distinct groups of documents. Note that LSA applies a "bag-of-words" approach: it typically analyzes words regardless of their part of speech (such as noun, verb, adjective, adverb) or their position in the sentence[4]. As such, it fails to capture some of the text's meaning; nonetheless, what remains can still be very informative.

As an illustrative example consider the words "cat", "dog", "mouse", "green", "blue", and "red". If one were to compare the co-occurrences of these words in typical language usage in English as they appear in newspaper articles then it is likely that "cat", "dog", and "mouse" would co-occur highly with each other, and less so with "green", "blue", and "red", while "green", "blue", and "red" would co-occur highly with each other but less so with "cat", "dog", and "mouse". This could be interpreted as "cat", "dog", and "mouse" relating to one category of words, and "green", "blue", and "red" to another. One could then project one's own worldly knowledge to assume that "cat", "dog", and "mouse" are animals while "green", "blue", and "red" might be colors or political parties. Moreover, it is likely that also the documents in which those 6 words appear could be classified into two groups based on those where "cat", "dog", and "mouse" mostly co-occur and those in which "green", "blue", and "red" mostly co-occur. One could then project worldly knowledge to assume that the first group of documents deals perhaps with animals, and the second group of documents with colors or politics. Of course, things are not that simple. As an alternative example, think of the terms "red", "pink", "ruby", "wine", "burgundy", and "bordeaux". (Note that LSA treats uppercase and lowercase the same.) The six terms may be thought of as reflecting a shared latent concept of "hues of red", which they do. However, notice that "wine", "Burgundy", and "Bordeaux" may additionally have their own shared latent concept (i.e., "wines") and that "Burgundy" and "Bordeaux" might also co-occur as places in France.

As those examples imply, analyzing word co-occurrences can provide powerful insight—even if it requires adding outside worldviews to interpret. But LSA does more than that. It may be that two words (or terms) are related through a third word (or term) only. Consider the words "red" and "merlot". These words would likely appear together frequently in a corpus constructed from a series of wine blogs, and their common use in documents could then be used to identify that they are related. However, merely looking for words that appear together directly would never identify "Cheval Blanc" and "Franzia" as being related since they would not frequently appear together within the same document. LSA, on the other hand, could identify that the two terms are related through their frequent shared co-occurrence alongside other terms such as "producer" or "merlot". So, despite the fact that Château Cheval Blanc produced the most expensive merlot ever sold and that the Franzia brothers package wine in cardboard boxes, LSA could identify that they are related. Mathematically, this is done by running a singular value decomposition (SVD) on the *term-document* [frequency] *matrix* (TDM)[5]. This matrix contains the number of times any term of interest (or to be exact, any term not excluded) appears in any of the documents being analyzed. SVD is a two-mode data reduction analysis that transforms the TDM into three matrices: (1) terms, (2) documents, and (3) a matrix that multiplied by the other two will reconstruct the original TDM matrix. In terms perhaps better known in behavioral sciences, running an SVD is conceptually equivalent to the data reduction a principal component analysis (PCA) does in identifying underlying factors in a matrix. SVD and PCA are closely related mathematically except that a PCA creates one transformed matrix of interest while SVD creates two. In both SVD and PCA, underlying factors are assumed to carry some higher-level abstract

[3] Terms can also be abbreviated words such as "ru" for "run", "running", "ran", or combinations such as "information technology".

[4] The part of speech can be assigned to a word, allowing each different part-of-speech usage of a word to be treated as a unique term. .

[5] To be consistent with the R function **TermDocumentMatrix** and with Debortoli et al. (2016), this guide shall refer to this matrix as the *term-document matrix*. The reader should be aware that some applications, such as JMP, use the term DTM, for *document-term matrix*. See http://www.jmp.com/support/help/13/Latent_Semantic_Analysis_SVD.shtml

meaning that is common across the items that compose that factor. As in PCA, determining the number of factors and their meaning can be challenging and controversial, revealing different results and meanings depending on the number of factors. LSA allows the application of SVD also to combinations of terms. *Running SVD on the TDM is what defines LSA* and makes it more than mere word co-occurrences analysis.

1.3 How Does LSA Work?

SVD and PCA are closely related mathematically except that a PCA creates one transformed matrix of interest while SVD creates two. In both SVD and PCA underlying factors are assumed to carry some higher-level abstract meaning that is common across the items that compose that factor. As in PCA, determining the number of factors and their meaning can be challenging and controversial, revealing different results and meanings depending on the number of factors. LSA allows the application of SVD also to combinations of terms. *Running SVD on the TDM is what defines LSA* and makes it more than mere word co-occurrences analysis.

$$M = U \times \Sigma \times V^T \tag{1}$$

The original *M* matrix could, therefore, be reconstructed by multiplying the *U*, *Σ*, and *V* matrices. However, in LSA, a truncated SVD is used wherein only a portion of the *Σ* matrix is calculated or retained[6] and the remaining singular values are replaced with zeroes. If the matrices were multiplied back together, as in formula 2, it would create an approximation of the original matrix where the number of singular values used determines how close the approximation is. The reconstructed matrix is known as the rank *k* approximation, A_k, where *k* is the number of singular values used. That multiplying the reduced rank matrices only creates an approximation of the original matrix may seem to be a problem but is actually one of the most powerful features of LSA. Because SVD seeks to minimize error, it combines vectors that are closest to each another, thus preserving as much of the original information as possible in fewer dimensions. As a result, selecting an appropriate rank is critically important in LSA. If *k* is too small, then the result may be combining vectors that are not related conceptually but are just the most related among those remaining. If *k* is too large, vectors that are related conceptually may not be combined because the algorithm stopped at *k*. In the case of large text corpora, as in the code example in section 3.4, *k* is often set at 100, 200, or 300. There is no rule on how best to select *k* a priori. The SVD transformation creates a *semantic space* out of the TDM.

$$M \approx A_k = U_k \times \Sigma_k \times {V_k}^T \tag{2}$$

While an SVD can be run on the TDM of the raw text of the documents, performance is often improved when external knowledge is applied to the documents before the SVD is run. This pre-processing often transforms the original text by turning words into their base forms, excluding words that carry little meaning, and giving more importance to words that are uncommon. Details on some common implementations of these are included in the *Overview* in section 2. The advantage of applying these transformations is that the number of terms can be greatly reduced, which may also improve the quality of the results because LSA is designed to function on the semantic meanings of the words, not their particular usage.

1.4 What Can One Do with LSA?

Once the semantic space has been created, much can be done with the term and document matrices created within that space. One common analysis is to compare vectors of terms by applying cosine similarity. This kind of analysis can be applied to find which terms are related to one another by calculating the cosine similarity between vectors in the *Σ×U* matrix. Likewise, such an analysis can be applied to determine which documents are related to one another by calculating cosine similarity between vectors in the $\Sigma \times V^T$ matrix. Using this relatedness information, it is possible to run a cluster analysis or a PCA to organize terms or documents into groups. Additionally, because both terms and documents are in the same vector space, it is possible to create a list of terms that are most related to a given document,

[6] Older implementations of LSA often had to perform this entire process manually and then select the top *k* dimensions—which in effect replaced singular values after the k^{th} with zeroes in the *Σ* matrix. Modern implementations often use probabilistic approximations of SVD that can take advantage of parallelism and the fact that TDM are often extremely sparse (i.e., most words do not appear in most documents, so their respective TDM include many zeroes).

thereby providing document labels, and documents that are most related to a given term, thereby allowing the grouping of documents into clusters of interest.

One type of analysis that is often performed with LSA is projecting new content into an existing semantic space. This new content takes the form of pseudo-documents. These pseudo-documents can be as short as only a few words. The pseudo-documents are compared to terms or documents in the existing space or to each other. Pseudo-documents can also be applied to identify the most related documents to a set of terms[7]. Pseudo-documents can also be compared across contexts, such as comparing the meaning of the word “mail” as used in history books about the Middle Ages, where it relates to armor, as opposed to its current use, which often relates to post. LSA can also be a preliminary step for other algorithms, using its vectors as input to those steps. The applicability of LSA to add insight is demonstrated in brief in Sections 3.3 and 3.4 in the contexts of analyzing consumer complaints and in the context of analyzing software Q&A.

1.5 Examples of Current LSA Application

Researchers have applied LSA to identify synonyms based on word co-occurrences (Gomez, Boiy, & Moens, 2012, Islam, Milios, & Keselj, 2012, Valle-Lisboa & Mizraji, 2007). Apparently, LSA’s ability to identify word co-occurrences is such that some researchers claim that it can even simulate some aspect of human thought as evidenced by its ability to answer multiple choice questions in introduction to psychology exams and do as well as students do (Landauer, Foltz, & Laham, 1998). Likewise, LSA can score on TOFEL exams comparably to non-native speakers (Landauer & Dumais, 1997). Going beyond synonyms, one can apply LSA to analyze medical journal papers to identify expected co-occurrences of terms (Landauer, Laham, & Derr, 2004).

In the realm of IS, LSA has mostly been used as a tool to aid in text categorization. By 2012, according to Evangelopoulos, Zhang, and Prybutok (2012), IS research has mostly applied LSA to: 1) create quantitative analyses of literature reviews, 2) the analysis of textual data in computer-mediated communication, 3) the analysis of customer feedback, interviews, and free text surveys; and 4) the management of knowledge repositories. See details in the Appendix. LSA has been applied to the IS discipline to examine the conceptual scope of the discipline as in Sidorova et al. (2008), Larsen et al. (2008a), and Larsen et al. (2008b). In those papers, a semantic space was created from document vectors based on the text of papers in IS journals using either abstracts (Sidorova, Evangelopoulos, Valacich, & Ramakrishnan, 2008) or the full text of the paper (Larsen, Monarchi, Hovorka, & Bailey, 2008a Larsen, Nevo, & Rich, 2008b). Sidorova et al. (2008) analyzed this sematic space by rotating and examining the most significant eigenvectors. Larsen, instead, performed cluster analysis (Larsen et al., 2008a) and compared distances (Larsen et al., 2008b) on document vectors extracted from the sematic space. In addition to the applications identified by Evangelopoulos et al. (2012), recent papers in the Appendix applied LSA in pursuit of grounded research.

Another typical application of LSA in IS research, as elsewhere, is to “project” pseudo-documents onto an existing semantic space. Those pseudo-documents were often questionnaire items. Examples of that approach include questionnaire item analyses in Arnulf, Larsen, Martinsen, and Bong (2014) and Larsen and Bong (2016). Such analyses enable nearest neighbor word analysis, as used in thesaurus creation, and nearest neighbor document analysis, as used in k-nearest neighbor analysis in predictive analytics. As in related research, such as Landauer et al. (2004) who demonstrated lexical closeness of terms in medical data in PNAS paper, this method can allow researchers to derive insight from lexical closeness even if only keywords appear in the corpora (Landauer et al., 2004). That has been also shown by Larsen and Bong (2016) who used LSA to study the theoretical similarity of constructs, and by Cao, Duan, and Gan (2011) and by Ahmad and Laroche (2017) who showed what factors influence reviews. Demonstrating the potential of such methodology, Gefen and Larsen (Forthcoming) showed that the semantic distances among the original TAM (Davis, 1989) items could be constructed, and TAM replicated accordingly in covariance based structured equation modeling (CBSEM), even though the original TAM items never appeared in any of the newspaper corpora they entered into LSA (only scattered keywords did). The Appendix contains a summary of IS related research that applied LSA or equivalent methods.

[7] When LSA was created, this ability to use pseudo-documents as a query function was its primary use. That is why it is sometimes referred to as Latent Semantic Indexing or LSI.

1.6 How Does LSA Differ from other Text-analysis Methods?

Since its introduction, several other text analysis methods have attempted to improve on LSA. Chief among those alternatives include LSA evolutions such as the probabilistic LSA (pLSA) and the latent Dirichlet allocation (LDA) families of algorithms. Probabilistic LSA focuses on addressing the concern that LSA does not have a normalized probability distribution and that negative values may occur in the matrices. To address those issues, pLSA replaced the SVD step in LSA with an expectation-maximization (EM) method that calculates the weights of the vectors.

LDA was the next evolutionary step after pLSA. LDA applies an equivalent algorithm to pLSA except that it beings by calculating Dirichlet priors of term weights, which allows LDA to reduce over-fitting and to turn the analysis into a generative model. LDA emerged as an evolution of pLSA and uses different assumptions about the distribution of words in documents. Specifically, LDA creates probability distributions of the likelihood of a topic (associated with a collection of words), spontaneously emitting a word or a document. While similar to LSA, LDA is claimed by Debortoli, Müller, Junglas, and vom Brocke (2016) to be easier to interpret because terms (e.g., “color”) have words that clearly stand out as the most representative of that term (in the case of “color” this may be the three basic colors “red”, “green”, and “blue” that a human eye sees). Accordingly, one advantage of LDA, and to a lesser extent also pLSA, is that the vectors generated by those methods tend to be easier to interpret. For an excellent tutorial on topic modeling which focuses on LDA, see Debortoli et al. (2016).

Despite these advantages, there are some appealing aspects to LSA that are not present in those other algorithms. One major advantage is that LSA is significantly faster with modern code libraries. While trading speed for quality may not seem desirable, the increased speed means that it is easier to try LSA with a large number of different options in order to tweak it to perform at its best by making choices such as changing stop words, altering the number of dimensions, or changing documents in the corpus. A properly tuned version of a less powerful algorithm can show significantly improved results over a more powerful algorithm with less tuning. Another advantage of LSA over LDA is that there is consensus as to the appropriate function that should be used to compare words or documents, namely cosine similarity. Conversely, the literature on LDA is mixed with a number of different similarity algorithms—such as cosine similarity, Jaccard distance, Information Radius, and Jensen-Shannon Divergence—being proposed without a clear best similarity measure. While LDA is conceptually similar to LSA, there are some tasks in text analysis that neither LSA nor LDA are well suited for. Among those tasks is information extraction (IE). IE is the process of transforming unstructured data into a structured form. This includes comparatively simpler tasks such as named entity recognition (NER) that identifies multi-word expressions that constitute a single real world object, to more complex tasks, which include event and relationship extraction. Finally, it is worth noting that a new class of algorithm has received recent literature attention: neural network based algorithms such as word2vec and GloVe. Those algorithms share some underlying assumptions with LSA (they all include a conception of a vector space which translates words into numeric representations, for example) but depart in very significant ways. While these algorithms offer promise for the future, their newness means that rules of thumb and guidelines still need to be established and that the algorithms still need to be validated in specialized domains such as IS.

1.7 Possible Directions of Interest in Applying LSA in IS

Apart from the examples about current LSA applications we discuss in Section 1.5, LSA could potentially open the door to a host of other types of avenues of research, some of which could provide new insight and maybe even redirect IS research into new pastures. We briefly discuss some of these new avenues. By no means is the list comprehensive.

1. **Identifying ontologies**: Larsen et al. (2016) investigated whether they could use LSA to create ontogenies that could unify behavioral medicine research in the interest of creating a unified body of knowledge by aggregating results across studies—even when they use different terminologies. Mixed labels, which Larsen et al. use to refer to different terminologies, is arguably not uncommon in IS research either. Creating a unified ontology of IS research terms could greatly benefit also IS research projects by allowing researchers to compare results across studies. One such example that Larsen et al. give is self-efficacy and perceived behavioral control. Ontologies might provide a partial solution to such terminology overlaps by creating a “knowledge base” to provide a standardized set of keywords for commonly used terms. As Larsen et al. point out, integrating the results of many studies through a shared unifying ontology could also suggest new hypotheses and create insights.

2. **Ontologies need not apply only to key terms that define theories' key constructs**: ontologies, or "semantic neighborhoods" as Kintsch (2001, p. 177) refers to them, could also define groupings of words of various degrees of overlapping meaning based on context across user groups. Ontologies could provide a relative standardized automated way to identify different ways in which IT are used and how IT, such as email and social networks, are used differently by different groups. A case in mind is Gefen and Straub (1997) who showed that female employees in the airline industry perceived more social presence in their email usage than male employees did. Gefen and Straub built their hypotheses on previous qualitative research into how men and women communicate differently (e.g., Tannen, 1994; Tannen, 1995), but they collected surveys data to support their hypotheses. Extending that research by looking into the actual ontologies of words used by men and of words used by women could add important insight into how exactly men and women communicate differently, and verify if this depends on the type of IT usage. As gender does affect IT usage (Gefen & Straub, 1997), understanding in depth how the specific words are used differently by men and by women could provide valuable insight into further developing theories of IT adoption. Such better understanding of constructs of interest could, as Coussement, Benoit, and Antioco (2015) showed, also improve experts' analysis of consumer reviews.
3. **Scale building**: is another domain where LSA and methods like it could come in handy. LSA could allow researchers to build research scales by consulting context specific synonyms and then verify those scale items by projecting them back onto those and other semantic spaces. LSA is a powerful tool for identifying alternative terms to use in scale building and learning hidden context-specific meanings (Kintsch, 2001). As Gefen and Larsen (Forthcoming) recently suggested, LSA could to some extent even replace the need for pretesting surveys through Q sorts and interviews to verify how people understand and aggregate key terms[8].
4. Moreover, LSA can provide a method for **pretesting surveys** by projecting questionnaire items on a semantic space of interest. As Gefen and Larsen (Forthcoming) show, the results of such a projection can as of itself support the expected research model of TAM even if real data do so better.
5. Such an exploratory projection of questionnaire items could also provide a powerful **theory-building tool** by allowing researchers to "inquire" how a corpus (and, by implication, maybe also how the people who wrote its content) aggregates questionnaire items or just key terms of interest. Researchers could use such aggregation, given by cosine and other lexical closeness measures, to initially test instruments and theory. Possibly, such questionnaire item aggregation could even serve to partly replace the need to interview experts.

2 Overview of the LSA Process

Figure 1 overviews the text-analysis process in LSA. It has four major steps: preparing documents, creating a semantic space, projecting pseudo-documents onto that semantic space, and comparing vectors.

[8] As Kintsch (2001, pp. 192-193) note, assessing how similar one word is to another through LSA has limitations in that the cosines are context (i.e., corpus) dependent. Moreover, these measures may not be symmetric, which means that the order of the terms in the sentence may make a difference if the vector lengths of the terms are grossly mismatched. For example, Kintsch (p. 193) compares "Korea is like China" to "China is like Korea". In the former comparison, "Korea" is the argument term being assessed, while "China" is the predicate. In that case, the projection compares "Korea" (about which their corpus contains little information as its shorter vector length indicates) to the context of "China" (about which the vector is much longer). In the latter comparison, "China" as the argument is compared to "Korea" as the predicate context. The cosines of those two sentences were .98 and .77, respectively, which suggests that the greater amount of knowledge (vector lengths) about China modifies what is known about Korea more than the other way around.

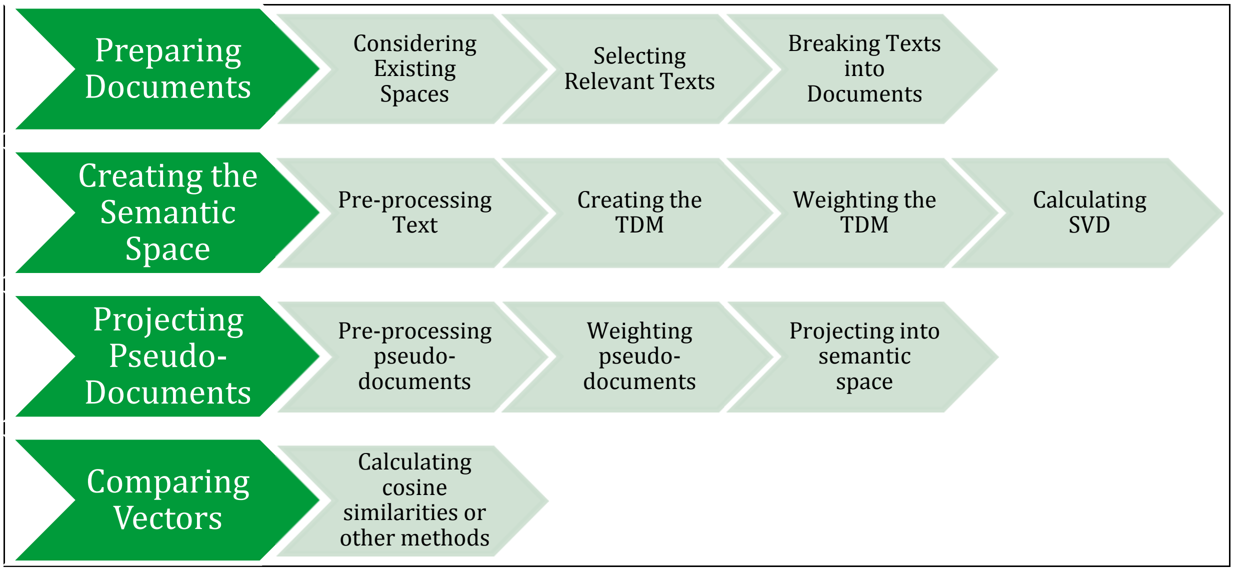

Preparing Documents
Considering Existing Spaces
Selecting Relevant Texts
Breaking Texts into Documents
Creating the Semantic Space
Pre-processing Text
Creating the TDM
Weighting the TDM
Calculating SVD
Projecting Pseudo-Documents
Pre-processing pseudo-documents
Weighting pseudo-documents
Projecting into semantic space
Comparing Vectors
Calculating cosine similarities or other methods

Figure 1. Typical Steps Taken in LSA Text Analysis

2.1 Preparing Documents

In the preparation stage, one first collects a corpus of relevant documents. When selecting documents to include in the corpus, one must consider the context in which the phenomena of interest reside. The adage “garbage in, garbage out” applies here, too. Even including a large collection of high-quality documents could fail if the context of those documents is not aligned with the phenomena of interest. This context must match factors such as the intended audience (e.g., CEO vs. floor manager vs. reporter), professionalism (e.g., formal report vs. personal emails), domain (e.g., advertising vs. human resources), and other features that may impact language (e.g., language proficiency or use of abbreviations). However, one does not always have to create a novel semantic space for each task: re-using standard corpora such as those from *TASA*[9] or the *Business News Corpus* (Larsen & Bong, 2016) might also enable comparisons. Furthermore, even in narrowly focused domains, jargon represents a relatively small portion of the language people use in documents, which means the jargon terms will likely stand out and have a greater impact if one uses global weighting. Also, for LSA to function well, one needs to use a relatively large corpus (on the order of thousands of documents at the lower end). A large corpus is necessary in order to create a representative sample and to increase the probability that any word that may be compared to that corpus does appear in it. If new documents are compared to an existing corpus and those documents contain words that do not appear in the corpus, then the analysis will exclude those words, skewing the results. As a result, it may be better to use a standardized corpus or to cast a wider net when collecting documents because of the importance of size. Like in all research, one must consider the trade-off between quality and availability.

Another consideration is what exactly is meant by “document”. A document in the context of LSA is ideally a portion of text that relates to a single topic. In many cases this is more akin to how a paragraph is used (in English at least) rather than to a whole document that is composed of many paragraphs. As a result, segments of text that are a paragraph long might be a better choice for a “document” in LSA. However, this is certainly not always the case; some documents are very short and narrowly focused, such as press releases. In such cases, breaking the document into paragraphs may create unwarranted distances between terms if each paragraph is considered independently.

Another consideration when selecting documents for a corpus is the accessibility of the documents. There are many available sources: classical literature is available online through sources such as *Project Gutenberg*, open source projects such as *Wikipedia* and *Wiktionary* are available through *Wiki Source*, and archives of public websites are available on the *Internet Archive*. There are also tools available to automate downloading content from more recent websites, but one needs to be careful about terms of

[9] TASA is perhaps better known as the “General Reading up to 1st Year of College” at http://lsa.colorado.edu. In our experience, it has withstood the test of time for evaluating general concepts but has limited value for special-purpose problem solving.

service. Accessing those texts may require an extensive investment in time and money to access or to buy the rights to the documents. In some cases we have been involved in, accessing and analyzing the documents, such as medical records, also required IRB approval and oversight. Akin to typical data collection in behavioral research, building the sample could be the most expensive and perplexing part in the research.

2.2 Creating the Semantic Space

Once the corpus has been created, its documents are often pre-processed. This pre-processing is where words are normalized into a form that, depending on research objectives, could be better suited for grouping terms. Such transformations reduce the risk that words with context equivalent meaning, such as “run” and “running”, may be misinterpreted as not carrying equivalent contextual meaning. Common transformations are:

1. **Stemming**: involves creating a single representation of the word regardless of its tense (past, present, present continuous, future, singular, plural, etc.). Stemming is language dependent. In this guide we shall rely on the default English stemming functionality in R. There are existing functions in R that stem data in other languages, too. Stemming turns words into a base form such as turning “cats” into “cat” and “jumped” into “jump”.
2. **Removing stop words**: involves discarding words that are common in a language but do not carry significant semantic meaning such as “a”, “the”, and “and”. There are standard lists of stop words in English and other languages.
3. **Orthographic transformations**: involves removing accents, expanding contractions or handling possessives, casting the text into lower case, and standardizing formatting.
4. **Stripping punctuation**: involves replacing punctuation signs with spaces.
5. **Identifying named entities**: involves combining words that commonly appear together and likely represent a single concept into one word, such as replacing “New York” with “New_York”.
6. **Lemmatization**: involves replacing words with their base form as it appears in a special dictionary. (Stemming does an equivalent operation based on predefined rules. [10].)
7. **Substitution**: involves replacing words with a string that indicates their class, such as replacing names of people in a document with [NAME].

A word of caution. Adding or removing words from consideration will as a matter of math change the SVD results and, through it, how other words and documents might relate to each other even beyond the words that were removed. As an example, deciding to remove all the words that are not in the Queen’s English may avoid slang and misspelled words from being included in the analysis, which might reduce the risk of introducing bias into the SVD results, and so may plausibly be a desired outcome in some circumstances. However, doing so may also exclude portions of text of interest, such as text written by or describing minorities with their own unique words and spelling—a plausibly undesired outcome depending on the objective of the study. Likewise, stemming. Automatically dropping “e”, “ed”, and “ing” from the end of a word might arguably correctly treat “walked” and “walk” as the same term because they are referencing the same physical activity. This might be desired because, plausibly, in some cases, the researcher may not wish to differentiate between the temporal tenses as they appear in those two words. However, stemming may also treat “university”, “universal”, and “universe” as the same term. That is arguably undesirable. Another case in mind is whether to differentiate between American and British spelling. Doing so would make no sense in many cases—unless the objective is something like comparing American to British writers. That having been said, consistency is of even greater importance than selecting the optimal transformations. All documents within the corpus must have the same transformations applied in the same order using the same rules. All pseudo-documents projected onto the corpus must also use the same transformations; not doing so might produce meaningless results.

Once the words/terms have been transformed, the next step in semantic space creation is creating the TDM. The TDM is a matrix that contains the terms in the corpus as rows, and the documents as columns.

[10] “**Lemmatisation** (or **lemmatization**) in linguistics is the process of grouping together the different inflected forms of a word so they can be analysed as a single item… In many languages, words appear in several inflected forms. For example, in English, the verb 'to walk' may appear as 'walk', 'walked', 'walks', 'walking'. The base form, 'walk', that one might look up in a dictionary, is called the lemma for the word. The combination of the base form with the part of speech is often called the lexeme of the word.” (Lemmitisation, n.d.).

The cells contain the number of times a term appears within a document, and is known as the raw count. Next, a second set of transformations is applied based on the distribution of terms within the corpus. This process is known as weighting, and comes in two forms: local weighting and global weighting. In local weighting, additional importance is given to terms that appear more times within a single document. In global weighting, less importance is given to terms that appear in a greater number of documents within the corpus. These local weights are used in order to account for the diminishing increase in importance of additional appearances of a term in a document. Global weights account for how often a term appears in other documents based on the notion that terms that appear in many documents are less important—this is the same logic as that behind stop word removal. The most common types of global weights are inverse document frequency (known as IDF) and entropy ratio (usually referred to as entropy, even though it is not Shannon's entropy function). The TF-IDF weighting is the most common in IS literature. Larsen and Bong (2016) recommend log-entropy, which is more common in other disciplines. Table 1 provides the formulae for these transformations.

Table 1. Common Weighting Applied in LSA

Local weight for word *i* in document *j*		Global weight for word *i*	
Raw	$tf_{i,j}$	None	1
Binary	$\begin{cases} tf_{i,j} \geq 1: 1 \\ tf_{i,j} = 0: 0 \end{cases}$	IDF	$1 + \log_2\left(\frac{n}{df(i)}\right)$
Log	$\log_2\left(tf_{i,j} + 1\right)$	Entropy	$1 + \sum_{j=document}^{corpus} \frac{\frac{tf_{i,j}}{gf_i} \times \log_2\left(\frac{tf_{i,j}}{gf_i}\right)}{\log_2(n)}$
$tf_{i,j}$	Term frequency: number of times word i appears in the document j		
df_i	Document frequency: Number of documents word i appears in at least once in the corpus		
gf_i	Global frequency: Number of times word i appears across the entire corpus		
n	Number of documents in the corpus		
$Final\ weight = local\ weight \times global\ weight$			

With the TDM adjusted for weights, the final step in creating the semantic space is to perform SVD. When performing SVD, it is necessary to select an appropriate number of dimensions, also known as rank, for the reduced matrices. Once performed, SVD will produce three matrices *U*, *Σ*, and *V*. These three matrices together with the rules for pre-processing and information about which rows and columns correspond to which terms and documents, comprise the semantic space. It may be beneficial to create and store the *Σ*×*U* and the *Σ*×V^T matrices for use in similarity calculations of words and documents, respectively. The *Short Introductory Example* in Section 3.1 applies an *Σ*×*U* matrix to compare words.

2.3 Projecting Pseudo-documents onto the Semantic Space

Projecting pseudo-documents is the process of creating vectors for texts that were not already present as documents in the corpus. If the objective of the analysis is to examine the relationships between documents within the corpus, then it is unnecessary to project pseudo-documents onto the semantic space because there is already access to the vectors for each document. Projecting pseudo-documents is necessary when comparing, for example, the cosines of two new sentences as they are projected onto the semantic space. See examples in Sections 3.3 and 3.4. The process starts by pre-processing the text using the same transformations and in the same order that were applied when creating the semantic space, followed by counting the occurrences of each transformed term and applying the same weighting functions as applied to create the semantic space. This will produce a set of terms and a weighted count of how many times each term appears. With the weighted counts for each term ready, the next stage is to create the pseudo-document vector by summing the term vector from the *Σ*×*U* matrix multiplied by the weighted count of that term in the pseudo-document. The vector created from this pseudo-document represents what the document vector would have been for this document if this document had been in the semantic space. It can then be used just like any other document vector in the *Σ*×V^T matrix for purposes of analysis.

2.4 Comparing Vectors

Vectors are typically compared to each other with cosine similarity. These similarities are used to find which vectors are most similar to each other and which documents have a similarity above a specified threshold. Cosine similarity is the dot product of the vectors over the product of their magnitude (see Equation 3).

$$\cos(\vec{u},\vec{v}) = \frac{\vec{u}\cdot\vec{v}}{\|\vec{u}\|_2 \cdot \|\vec{v}\|_2} = \frac{\sqrt{\sum_i(\vec{u}_i \times \vec{v}_i)}}{\sqrt{\sum_i({\vec{u}_i}^2)} \times \sqrt{\sum_i({\vec{v}_i}^2)}} \quad (3)$$

The cosine similarity between a vector and itself is equal to 1.0. This means that the closer the cosine of two vectors is to 1.0, the more similar they are. However, caution is advised when interpreting low cosine similarities. A similarity near 0.0 may indicate that terms have opposite meanings, but it may also indicate that they are unrelated. If the words overlapping in those documents are only words that are relatively common in the corpus, even the co-occurrences of frequently co-occurring words may not drive their similarities up. The way cosine similarity is calculated also cancels out the magnitudes of the vectors. This is important because it enables terms, documents, and pseudo-documents of different lengths to be compared to one another. We show how meaning may be drawn from the comparison of vectors in the next sections.

3 Annotated Code Examples

In this section, we present annotated code in R that walks the reader through how to perform LSA. We begin with an overly simplistic case with small data so the reader can see all the data and how to transform it through the process. The code in Section 3.1 (example step 1) represents typical steps researchers take to create the semantic space (see second row in Figure 1). Readers can access a corpus of four text files of financial services complaints in a dedicated directory. The code example might make the process seem easy, but one must make many crucial decisions prior to running the code. As the code shows, many of those decisions are parameters that one passes to the software that does that process. The result is a relevant semantic space.

In Section 3.2 (example step 2), we pick up after having created the semantic space. Typical steps taken in this stage include projecting terms and vectors of terms onto the terms portion of the semantic space (the terms matrix). Likewise, one could project documents onto the documents portion of the semantic space (the documents matrix). One could do both these types of projection to calculate how close the terms or documents are to each other. Additional analyses that one can do in this step include running PCA, factor analysis, and multidimensional scaling to discover groupings of terms and groupings of documents. One can add graphics to visualize the story as well. Separating between step 1 in Section 3.1 and step 2 in Section 3.2 emphasizes that, in many cases, researchers are often more interested in comparing terms and phrases as they are commonly used in a specific corpus than in creating the corpora. Some past IS research includes both steps, such as Sidorova et al. (2008) who identified patterns in IS research based on papers in the top MIS journals. The corpus of those journals formed the basis for the semantic space. Other research emphasizes step 2, such as Larsen and Bong (2016) who used text analysis to identify construct identity. Larsen and Bong created a semantic space out of a large number of newspaper articles and then projected words of interest on that semantic space. In such cases, research focuses more on specific terms rather than on the documents themselves.

In the two example steps, we run LSA on a very small sample for pedagogical reasons. We follow the steps with two larger applications: in Section 3.3, we rebuild the semantic space of the short example with a more appropriately sized corpus of 2391 financial complaints. We apply the same SVD analysis as in the short example to derive possible insight about the term “trust”. SVD analysis, however, can be cumbersome when dealing with exceptionally large data. Analyzing such large data requires sparse SVD. Accordingly, the code example in Section 3.4 shows how to run a sparse SVD. Unlike the **lsa** function as applied in the first code sections, running sparse SVD requires extensive preceding data preparation. We demonstrate sparse SVD in analyzing a large corpus of Q&A posts on *Stack Exchange*, a popular Q&A programming site. We add background information about R as footnotes where necessary.

3.1 A Short Introductory Example: Step 1: Building the Semantic Space

In this section, we provide the reader a grounded hands-on experience with using LSA. This example is artificially extremely small so one can see all the data and map all the interconnections. For brevity, we create a directory of only four short documents so that one can trace the outcomes. We emphasize that one should not run LSA on such a small sample: we do here only so one can see all the data we run the LSA on. The reader can copy the code in this example and paste it as is into an R console or editor. If the computer supports it, we recommend that one open the 64-bit version or R instead of the 32-bit one. One can save the R script and rerun it[11]. We intend this example for users who use LSA and R for the first time. We show more realistic semantic spaces and more complex R code in Sections 3.3 and 3.4.

Before we begin, we need to load required code libraries, known in R as “packages”. Because R serves many purposes, much of the code resides in libraries such that each library contains a collection of functions relevant to a specific topic[12]. Because R has a strong Unix heritage, R is case sensitive. Also, crucially, be aware that R expects quotes as vertical symbols (i.e., **"),** not the default curly ones (i.e., **“ ”**) that Microsoft Word creates as a default. R does not recognize curly quotes. The pound sign in R creates a comment. The **library** command loads the required packages **LSAfun** (analytic functions) and **lsa** (simple package to run LSA). The data analyzed in this snippet and the next two sections is financial-complaint data that one can download from https://catalog.data.gov/dataset/consumer-complaint-database.

```
library(LSAfun)
library(lsa)
```

After downloading the zipfile that contains the text files, extract the directories “MiniComplaints” and “FinancialComplaints”. Those directories contain the text for Sections 3.1-3.3. In relation to Figure 1, this section of code corresponds to the first row “preparing documents”.

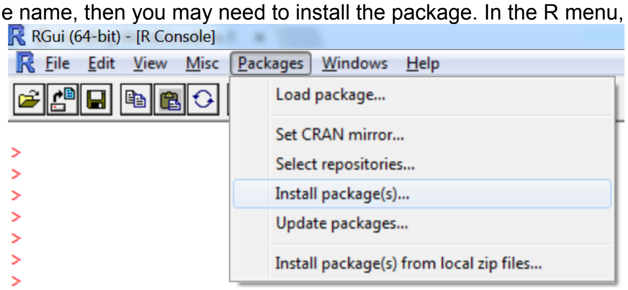
```
# Replace the [...] with the path in which the directory MiniComplaints was created.
source_dir = '[...]'
# source_dir = 'C:/Users/username/Desktop/MiniComplaints' # Windows example
# source_dir = '~/Desktop/MiniComplaints' # Mac/Unix example
```

We will now build the TDM. First, as a standard functionality, we will import the list of default stop words in English from the **LSA** package. This list is called **stopwords_en**. The **print** command prints the file in parentheses. In relation to Figure 1, this section of coded corresponds to the “pre-processing text” arrow.

```
data(stopwords_en)
```

[11] To rerun a saved script, type:
source*("path…/your_script_file_name")*

[12] In all the code examples, if the **library** commands says “Error in library(LSAfun): there is no package called ‘LSAfun’” or any other package name, then you may need to install the package. In the R menu, choose packages and then follow the dialog.



After the package is installed, rerun the **library** command in R. In Section 4.4, we show how to avoid this issue.

print(stopwords_en) # The complete list of stop words can be shown by using Print.

# or just entering

stopwords_en

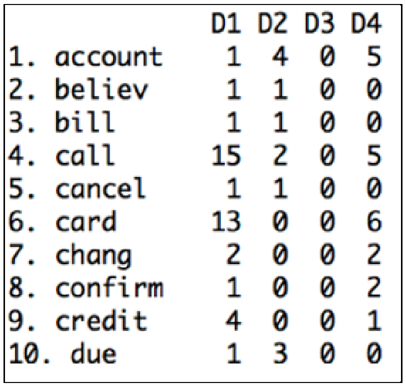
Next, we will create the TDM from the MiniComplaint text files. In doing so, we will apply the stop words list **stopwords_en** that we just imported. Since we know that our text files include two additional terms used to anonymize data, “xx” and “xxxx”, we will add these to our stop word list. We do so with the function **textmatrix**. Notice that TDM is a frequency matrix and that all the words have been transformed to lower case. The function **textmatrix** changes upper case to lower case by default and removes apostrophes and special characters. In relation to Figure 1, this step corresponds to the arrow “creating the TDM”.

TDM <- textmatrix(source_dir, stopwords=c(stopwords_en, "xx", "xxxx"), stemming=TRUE, removeNumber=F, minGlobFreq=2)

# Optionally, we can show the content of the matrix TDM by just typing the dataset name

TDM

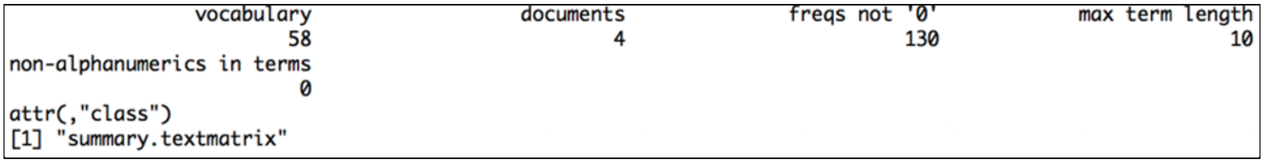
Notice that the terms appear as the row IDs and the documents as the column IDs. The numbers in the cells are the frequency of each term in each document.



```
             D1 D2 D3 D4
1. account    1  4  0  5
2. believ     1  1  0  0
3. bill       1  1  0  0
4. call      15  2  0  5
5. cancel     1  1  0  0
6. card      13  0  0  6
7. chang      2  0  0  2
8. confirm    1  0  0  2
9. credit     4  0  0  1
10. due       1  3  0  0
```

One shows descriptive statistics about a dataset with the **summary.textmatrix** command.

summary.textmatrix(TDM)

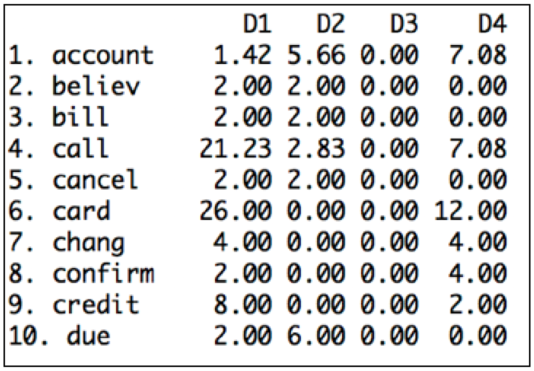


```
               vocabulary                  documents              freqs not '0'            max term length
                       58                          4                        130                         10
non-alphanumerics in terms
                        0
attr(,"class")
[1] "summary.textmatrix"
```

Before running an SVD, we will create a weighted matrix TDM2 out of the original TDM. TDM2 is the term frequency times its inverse document frequency. This method is a standard one. Again, because we are using a very small dataset, it is meaningful to show the content of the matrix TDM2. One can do so by typing the dataset name—in this case, **TDM2**. Notice how the matrix now contains weights rather than frequencies. In relation to Figure 1, this step corresponds to the “weighting the TDM” arrow.

TDM2 <- lw_tf(TDM) * gw_idf(TDM)

TDM2



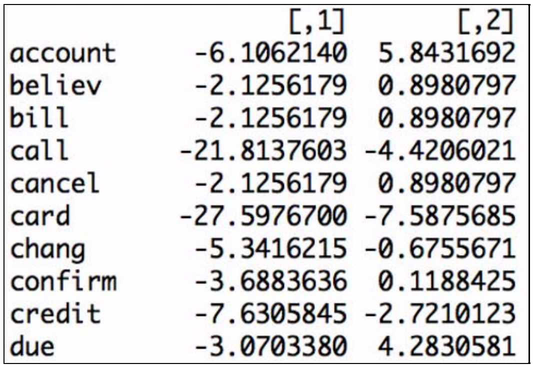
```
                D1   D2   D3    D4
1. account    1.42 5.66 0.00  7.08
2. believ     2.00 2.00 0.00  0.00
3. bill       2.00 2.00 0.00  0.00
4. call      21.23 2.83 0.00  7.08
5. cancel     2.00 2.00 0.00  0.00
6. card      26.00 0.00 0.00 12.00
7. chang      4.00 0.00 0.00  4.00
8. confirm    2.00 0.00 0.00  4.00
9. credit     8.00 0.00 0.00  2.00
10. due       2.00 6.00 0.00  0.00
```

We can now run an LSA on the weighted matrix TDM2. The code in this case chooses the number of dimensions by default using the option **dimcalc_share().** To specify any positive integer number of dimensions, replace **dimcalc_share()** with a number (e.g., 3). Notice how the LSA-transformed matrix now contains cross-loadings that did not appear in the original matrix TDM2. This additional information comes from the SVD. To view the matrix, we will first transform it into a textmatrix data type with the **as.textmatrix** command. Typing **as.textmatrix(miniLSAspace)** will print the **miniLSAspace** as a text matrix. In relation to Figure 1, this part corresponds to the arrow "calculating SVD".

```
miniLSAspace <- lsa(TDM2, dims=dimcalc_share())
as.textmatrix(miniLSAspace)
```

In the code above, the function **lsa** transformed TDM2 into three matrices and placed all three into the **miniLSAspace** object. We show these three matrices below. The suffixes **$tk, $dk**, and **$sk** represent the three matrices: the term matrix, the document matrix, and the singular value matrix, respectively. Adding those suffixes allows one to view each of the matrices in **miniLSAspace** separately. As in a PCA, one can interpret groupings of item loadings as revealing a higher abstract dimension, although they are unrotated.

```
# This command will show the value-weighted matrix of Terms
tk2 = t(miniLSAspace$sk * t(miniLSAspace$tk))
tk2
```

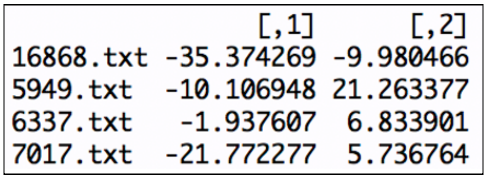


```
               [,1]       [,2]
account  -6.1062140  5.8431692
believ   -2.1256179  0.8980797
bill     -2.1256179  0.8980797
call    -21.8137603 -4.4206021
cancel   -2.1256179  0.8980797
card    -27.5976700 -7.5875685
chang    -5.3416215 -0.6755671
confirm  -3.6883636  0.1188425
credit   -7.6305845 -2.7210123
due      -3.0703380  4.2830581
```

In the **tk2** matrix, we see that "believ", "bill", and "cancel" have identical values, which means that they co-occur exactly the same across the four documents. We also see that "card" and "call" are very different in the first dimension compared to the other terms shown. The terms "credit" and "account" seem close in the first dimension but are separated in the second dimension.

# This will show the matrix of Documents

miniLSAspace$dk

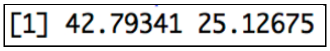


```
                [,1]      [,2]
16868.txt -35.374269 -9.980466
5949.txt  -10.106948 21.263377
6337.txt   -1.937607  6.833901
7017.txt  -21.772277  5.736764
```

The **dk** matrix shows the factoring of the documents. The first dimension separates the documents along its axis, but the last two are relatively close in the second dimension. We will graphically plot these relationships later. The document *7017.txt*, a complaint about a customer service experience with a credit card, falls between document 16868.txt with its multifaceted credit card complaint and document *5949.txt* that deals with a mortgage complaint that focuses on payment terms rather than service. The LSA space reflects this dimensionality. Notice that **$sk** is a matrix of singular values that connects **$tk** and **$dk** matrices to reproduce the original TDM2.

# Because the $sk matrix only has values on the diagonal, R stores it as a numeric vector.

miniLSAspace$sk



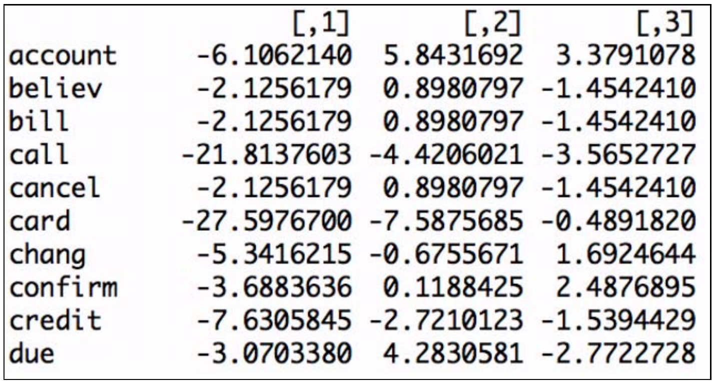
```
[1] 42.79341 25.12675
```

As a footnote to the above analysis, had we forced three factors on the SVD by specifying **dims=3** rather than the default, then the **$tk** matrix could have been more revealing.

miniLSAspace3 <- lsa(TDM2, dims=3)

tk3 = t(miniLSAspace3$sk * t(miniLSAspace3$tk))

tk3



```
              [,1]       [,2]       [,3]
account  -6.1062140  5.8431692  3.3791078
believ   -2.1256179  0.8980797 -1.4542410
bill     -2.1256179  0.8980797 -1.4542410
call    -21.8137603 -4.4206021 -3.5652727
cancel   -2.1256179  0.8980797 -1.4542410
card    -27.5976700 -7.5875685 -0.4891820
chang    -5.3416215 -0.6755671  1.6924644
confirm  -3.6883636  0.1188425  2.4876895
credit   -7.6305845 -2.7210123 -1.5394429
due      -3.0703380  4.2830581 -2.7722728
```

By considering an additional dimension, the relationships between terms add nuance. In two dimensions, the terms “account” and “due” are relatively similar, but, in the third dimension, they are separated. Their distances in a three-dimensional space are farther apart than in a two-dimensional space. Conversely, while the second dimension separated the terms “call” and “due”, the third dimension brings them closer. Because we are using a deliberately small dataset, presenting it in a diagram could be rather helpful. Figure 2 shows the result of the next section of code.

# The two lines of code must be run together. The first line of code creates a plot of the first two

```
# dimensions of $tk, marking the dots as red dots. The second line superimposes term names.
plot(tk2[,1], y= tk2[,2], col="red", cex=.50, main="TK Plot")
text(tk2[,1], y= tk2[,2], labels=rownames(tk2) , cex=.70)
# This can be done with the documents too. The added parameter cex determines text size.
plot(dk2[,1], y= dk2[,2], col="blue", pch="+", main="DK Plot")
text(dk2[,1], y= dk2[,2], labels=rownames(dk2), cex=.70)
```

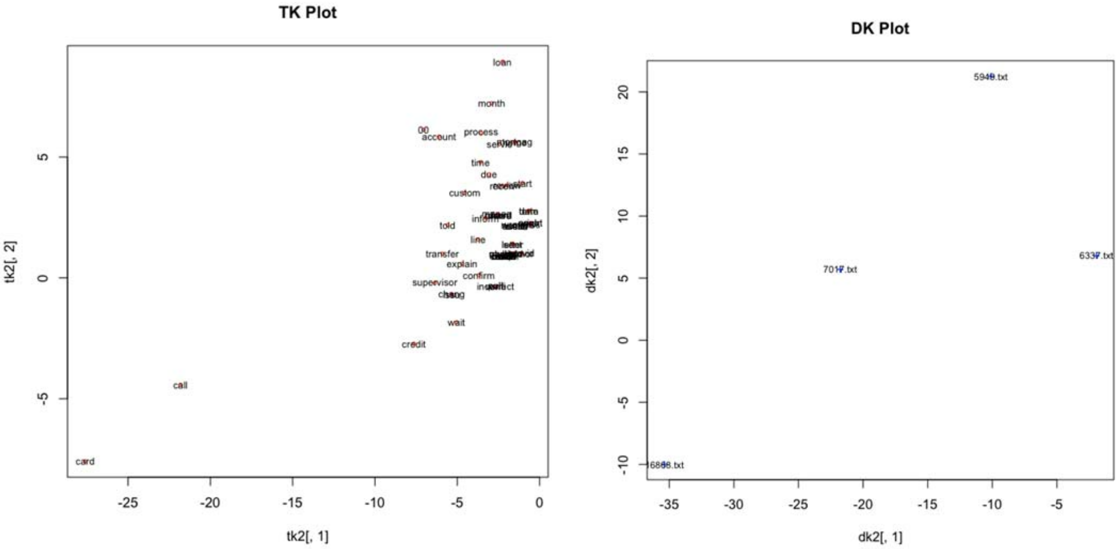


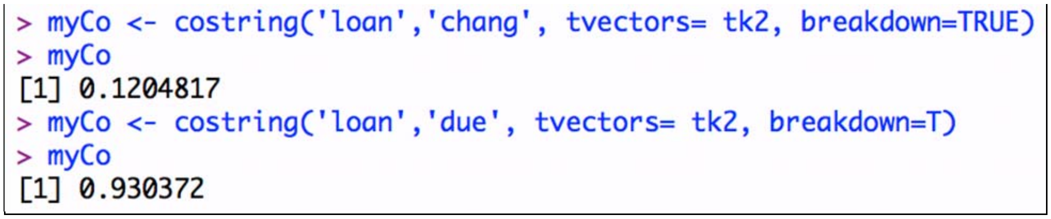
Figure 2. Terms (Left) and Documents (Right) Mapping in the MiniComplaints Dataset

3.2 A Short Introductory Example: Step 2: Analyzing the Semantic Space

The next section of the code starts at the point where the semantic space already exists. We will assume that the code in step 1 has already run so that the semantic space **miniLSAspace** exists. With a semantic space available, it is possible to calculate the cosines of terms in it to check how closely they relate to each other in that semantic space. In this case, we will calculate the cosine similarity between the terms “loan” and “chang”[13]. No two documents in this set use both these words, so the distance is not that close, but they are related through other words. The sample code will also calculate the distance between the terms “loan” and “due”, which are closely related. The parameter **tvectors** identifies the semantic space matrix of the terms. The parameter **breakdown=TRUE** forces the data into lower case, replaces umlauts with ae, removes accents, and replaces β with ss. As standard in R, the Boolean **TRUE** must be in uppercase or abbreviated as capital **T.** In relation to Figure 1, this step and the next ones correspond to the “projecting into semantic space” and “calculating cosine similarities or other methods” arrows.

```
# Create a cosine similarity between two Terms
myCo <- costring('loan','chang', tvectors= tk2, breakdown=TRUE)
myCo    # Typing the name of an object prints its value
myCo <- costring('loan','due', tvectors= miniLSAspace$tk, breakdown=T)
myCo
```

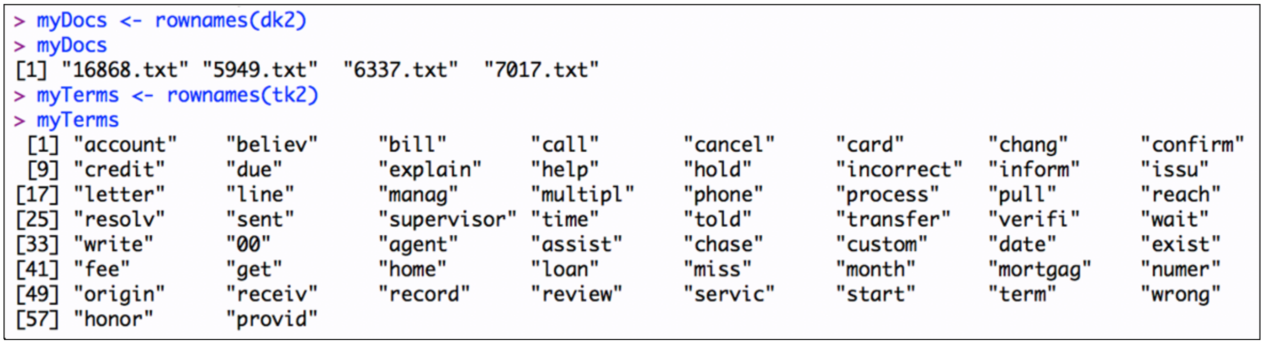
[13] “**Cosine similarity** is a measure of similarity between two non-zero vectors of an inner product space that measures the cosine of the angle between them. The cosine of 0° is 1, and it is less than 1 for any other angle. It is thus a judgment of orientation and not magnitude: two vectors with the same orientation have a cosine similarity of 1, two vectors at 90° have a similarity of 0, and two vectors diametrically opposed have a similarity of -1, independent of their magnitude.” (Cosine similarity, n.d.).



```
> myCo <- costring('loan','chang', tvectors= tk2, breakdown=TRUE)
> myCo
[1] 0.1204817
> myCo <- costring('loan','due', tvectors= tk2, breakdown=T)
> myCo
[1] 0.930372
```

We can run the code on all the documents or on all the terms in the semantic space. To retrieve the list of documents and the list of terms, we can run this code:

```
myDocs <- rownames(dk2)
myDocs
myTerms <- rownames(tk2)
myTerms
```

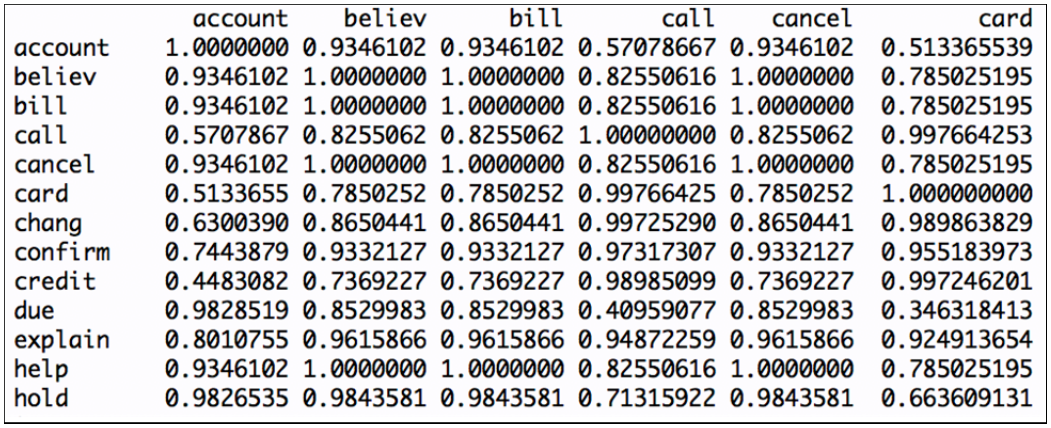


```
> myDocs <- rownames(dk2)
> myDocs
[1] "16868.txt" "5949.txt"  "6337.txt"  "7017.txt"
> myTerms <- rownames(tk2)
> myTerms
 [1] "account"    "believ"     "bill"       "call"       "cancel"     "card"       "chang"      "confirm"
 [9] "credit"     "due"        "explain"    "help"       "hold"       "incorrect"  "inform"     "issu"
[17] "letter"     "line"       "manag"      "multipl"    "phone"      "process"    "pull"       "reach"
[25] "resolv"     "sent"       "supervisor" "time"       "told"       "transfer"   "verifi"     "wait"
[33] "write"      "00"         "agent"      "assist"     "chase"      "custom"     "date"       "exist"
[41] "fee"        "get"        "home"       "loan"       "miss"       "month"      "mortgag"    "numer"
[49] "origin"     "receiv"     "record"     "review"     "servic"     "start"      "term"       "wrong"
[57] "honor"      "provid"
```

We can then run a cosine similarity among all the words in the matrix.

```
myTerms2 <- rownames(tk2)
myCosineSpace2 <- multicos(myTerms2, tvectors=tk2, breakdown=TRUE)
myCosineSpace2
```

A partial listing of the cosine matrix appears below.



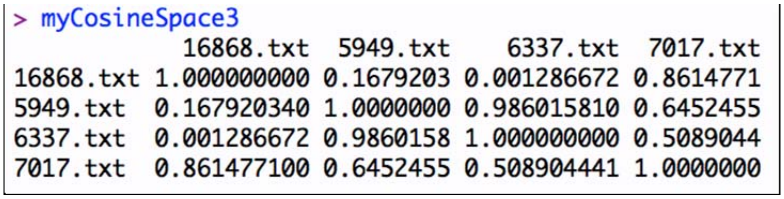
```
          account    believ      bill       call     cancel        card
account 1.0000000 0.9346102 0.9346102 0.57078667 0.9346102 0.513365539
believ  0.9346102 1.0000000 1.0000000 0.82550616 1.0000000 0.785025195
bill    0.9346102 1.0000000 1.0000000 0.82550616 1.0000000 0.785025195
call    0.5707867 0.8255062 0.8255062 1.00000000 0.8255062 0.997664253
cancel  0.9346102 1.0000000 1.0000000 0.82550616 1.0000000 0.785025195
card    0.5133655 0.7850252 0.7850252 0.99766425 0.7850252 1.000000000
chang   0.6300390 0.8650441 0.8650441 0.99725290 0.8650441 0.989863829
confirm 0.7443879 0.9332127 0.9332127 0.97317307 0.9332127 0.955183973
credit  0.4483082 0.7369227 0.7369227 0.98985099 0.7369227 0.997246201
due     0.9828519 0.8529983 0.8529983 0.40959077 0.8529983 0.346318413
explain 0.8010755 0.9615866 0.9615866 0.94872259 0.9615866 0.924913654
help    0.9346102 1.0000000 1.0000000 0.82550616 1.0000000 0.785025195
hold    0.9826535 0.9843581 0.9843581 0.71315922 0.9843581 0.663609131
```

In an LSA of a corpus of appropriate size, terms should rarely co-occur identically and, thus, not show cosines of 1 as in this example. One can export this matrix for analysis in other statistical languages.

```
# Save the cosine space (the user should define the path within file=”…”)
write.csv(myCosineSpace2, file="C:/Users/…/CosineResults.csv")
```

One can perform the same process for documents, too, which shows that complaint 6337 is closer to complaint 5949 and that complaint 7017 is closer to complaint 5949 than to complaint 6337.

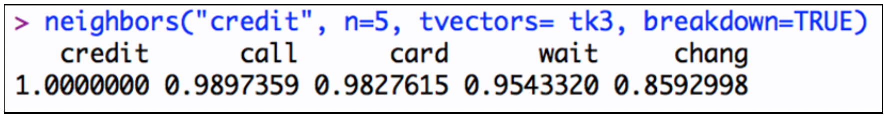
```
# This provides us with a similarity matrix between documents
myCosineSpace3 <- multicos(myDocs, tvectors=dk2, breakdown=F)
myCosineSpace3
```



```
> myCosineSpace3
            16868.txt  5949.txt    6337.txt  7017.txt
16868.txt 1.000000000 0.1679203 0.001286672 0.8614771
5949.txt  0.167920340 1.0000000 0.986015810 0.6452455
6337.txt  0.001286672 0.9860158 1.000000000 0.5089044
7017.txt  0.861477100 0.6452455 0.508904441 1.0000000
```

Another method for examining how close terms or documents are to each other is the function **neighbors**. This function returns the n nearest words in meaning to the term in the first parameter—in this case, the term “credit”. We use the term matrix from the three-dimensional space.

```
neighbors("credit", n=5, tvectors=tk3, breakdown=TRUE)
```



```
> neighbors("credit", n=5, tvectors= tk3, breakdown=TRUE)
   credit      call      card      wait     chang
1.0000000 0.9897359 0.9827615 0.9543320 0.8592998
```

One can plot these distances, too. The n below specifies how many of the closest neighbors to include in the diagram. The results are telling. The plot shows that “credit” is closely related to “card” and to “call” but less so to “chang”. Figure 3 shows the plot.

```
plot_neighbors("credit", n=20, tvectors= tk3)
```

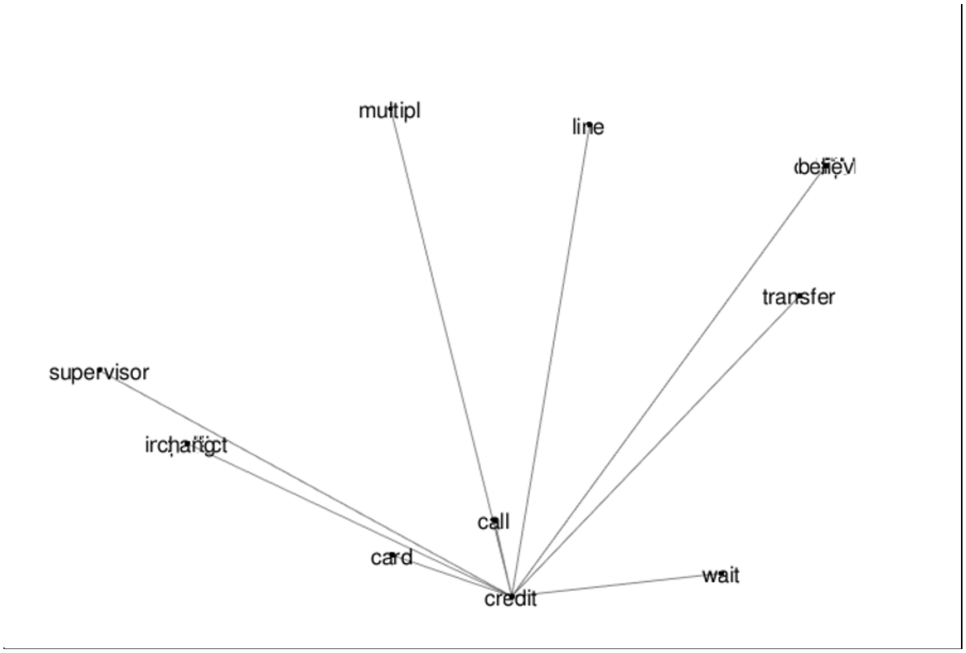


Figure 3. Mapping of Terms Related to “Credit” in the MiniComplaints Dataset

By default, **plot_neighbors** runs a PCA on the full vectors of the nearest neighboring terms and uses the first three components to plot them in three dimensions. One can then rotate the plot to investigate further. One can also plot a list of words in the same way. In this case, one does so into a two-dimensional space. While the above two-dimensional term plot used two dimensions of the semantic space, this function takes the full vector of each specified term and computes a PCA (by default) or an MDS from the vectors of the terms selected. In this example, we plot two components derived from three semantic dimensions. Figure 4 shows the plot.

```
words <- c("credit","card", "time", "supervisor")
plot_wordlist(words,tvectors=tk3,dims=2)
```

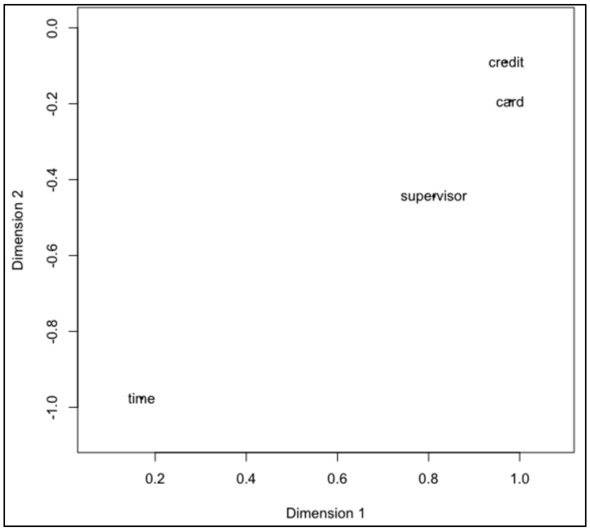


Figure 4. Mapping of “Credit”, “Card”, “Time”, and “Supervisor” into a Two-dimensional PCA Space

To make the interpretation easier, we can add connecting lines and explicitly specify that the space will be created using PCA. The results are now easier to interpret. In this case, one does so into a three-dimensional space. Figure 5 shows the plot.

```
plot_wordlist(words,tvectors= tk3, method="PCA", dims=3,connect.lines="all")
```

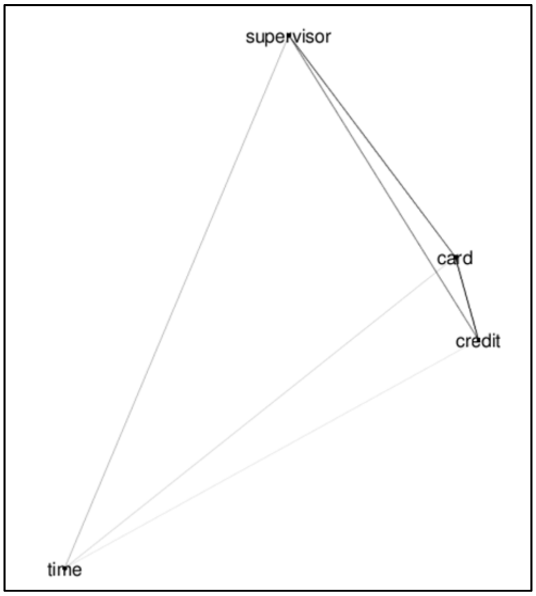
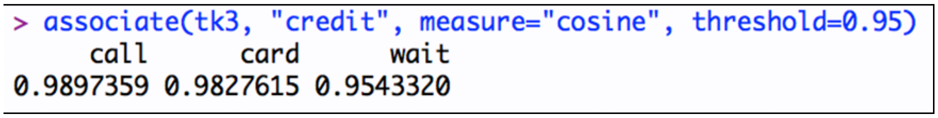


Figure 5. Mapping of “Credit”, “Card”, “Time”, and “Supervisor” into a Two-dimensional PCA Space with Connecting Lines

Another function that returns terms that are close to a given term is **associate**. While **neighbors** returns the nearest n terms, **associate** returns whichever terms are within a particular cutoff. And, while **neighbors** uses cosine, **associate** can also calculate Pearson and Spearman measures. (The parameters **pearson** and **spearman** in the parameter list must be in lower case.) The four parameters are the term matrix **tk3**, the term whose closest terms being sought, the closeness measure, and the threshold above which the terms will be selected. As is standard in R, be aware to keep lower case as lower case even if Microsoft Word changes the leading letter of the first word to upper case.

```
associate(tk3, "credit", measure="cosine", threshold=0.95)
```



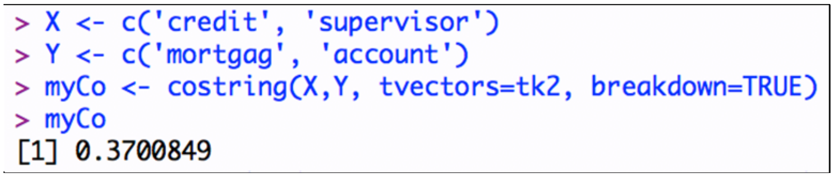
```
> associate(tk3, "credit", measure="cosine", threshold=0.95)
     call      card      wait 
0.9897359 0.9827615 0.9543320 
```

To compare vectors of terms in a sematic space—which could be convenient when comparing measurement items that are composed of many terms—all that needs to be done is to create the vectors of the combination of terms that compose each measurement item. In this case the code calculates how close the combination of the terms “credit” and “account” are to the combination of the terms “mortgag” and “account”. The measure is given in cosine distance. Note that **costring** does not stem the terms, so the exact terms should be used, and that if a term does not appear in the **tvectors** matrix, the analysis will omit the term without a warning. As a footnote, notice that single quotes and double quotes are interchangeable in the code. To demonstrate that, the next section of code uses single quotes.

```
X <- c('credit', 'supervisor')

Y <- c('mortgag', 'account')
```

myCo <- costring(X,Y, tvectors=miniLSAspace$tk, breakdown=TRUE)

myCo

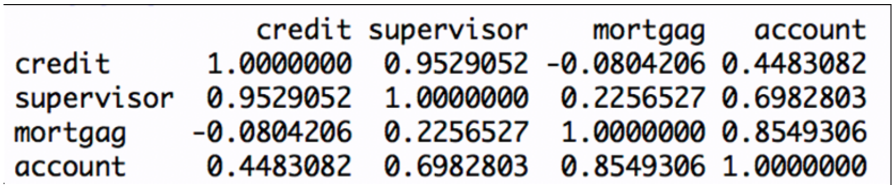


```
> X <- c('credit', 'supervisor')
> Y <- c('mortgag', 'account')
> myCo <- costring(X,Y, tvectors=tk2, breakdown=TRUE)
> myCo
[1] 0.3700849
```

To calculate the cosine of a series of terms, use the **multicos** function. The same function can be run to calculate the cosine between documents, too—all that needs to be changed to do so is that the **dk** matrix should be used in that case.

mcTerms <- multicos(c('credit', 'supervisor', 'mortgag', 'account'), tvectors= miniLSAspace$tk, breakdown=F)

mcTerms



```
               credit supervisor    mortgag   account
credit      1.0000000  0.9529052 -0.0804206 0.4483082
supervisor  0.9529052  1.0000000  0.2256527 0.6982803
mortgag    -0.0804206  0.2256527  1.0000000 0.8549306
account     0.4483082  0.6982803  0.8549306 1.0000000
```

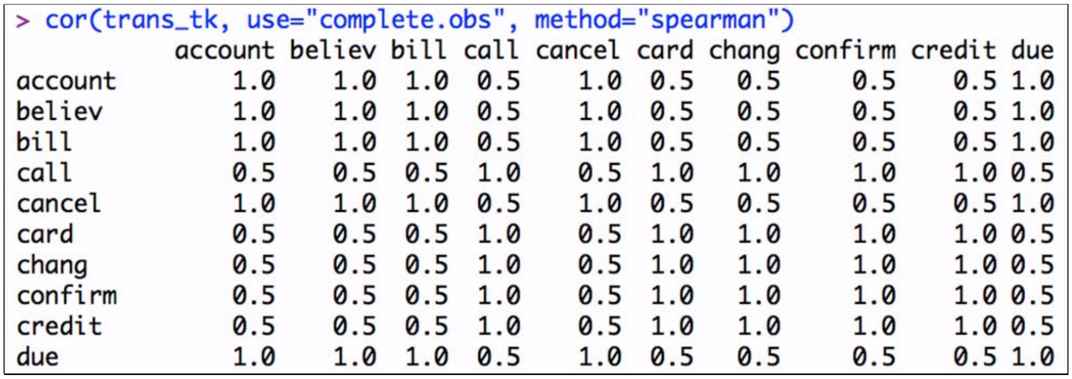
R has additional packages that one can run on semantic spaces. A convenient method of eyeballing what the data may indicate is to run a correlation on the terms matrix or on the documents matrix. Because of the structure of the **tk** and **dk** matrices, one needs to transpose them first. The function **t** does that. The correlation function is called **cor**. By default, **cor** runs a Pearson correlation. In the example below, we force it to run a Spearman and then a Kendall correlation just to show that it one can do it. The function **cor** can also be set to treat missing values as a listwise ("**complete.obs**") or as a pairwise deletion ("**pairwise.complete.obs**")[14]. These correlations provide insight on how the data might be interrelated. In this case, there is a clear grouping into two sets of terms and two sets of documents.

trans_tk <- t(as.matrix(tk3))

trans_dk <- t(as.matrix(dk3))

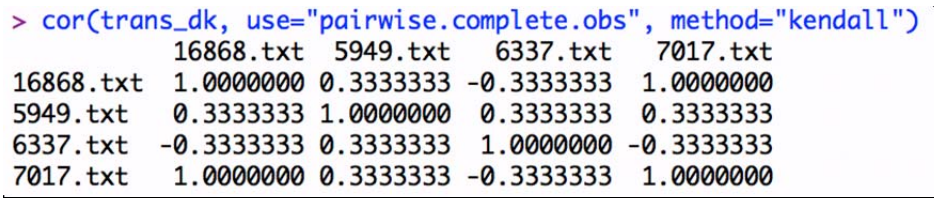
cor(trans_tk, use="complete.obs", method="spearman")

cor(trans_dk, use="pairwise.complete.obs", method="kendall")



```
> cor(trans_tk, use="complete.obs", method="spearman")
          account believ bill call cancel card chang confirm credit due
account       1.0    1.0  1.0  0.5    1.0  0.5   0.5     0.5    0.5 1.0
believ        1.0    1.0  1.0  0.5    1.0  0.5   0.5     0.5    0.5 1.0
bill          1.0    1.0  1.0  0.5    1.0  0.5   0.5     0.5    0.5 1.0
call          0.5    0.5  0.5  1.0    0.5  1.0   1.0     1.0    1.0 0.5
cancel        1.0    1.0  1.0  0.5    1.0  0.5   0.5     0.5    0.5 1.0
card          0.5    0.5  0.5  1.0    0.5  1.0   1.0     1.0    1.0 0.5
chang         0.5    0.5  0.5  1.0    0.5  1.0   1.0     1.0    1.0 0.5
confirm       0.5    0.5  0.5  1.0    0.5  1.0   1.0     1.0    1.0 0.5
credit        0.5    0.5  0.5  1.0    0.5  1.0   1.0     1.0    1.0 0.5
due           1.0    1.0  1.0  0.5    1.0  0.5   0.5     0.5    0.5 1.0
```

[14] A good source for simple stats and plots is http://www.gardenersown.co.uk/education/lectures/r/correl.htm.



```
> cor(trans_dk, use="pairwise.complete.obs", method="kendall")
           16868.txt  5949.txt   6337.txt   7017.txt
16868.txt  1.0000000 0.3333333 -0.3333333  1.0000000
5949.txt   0.3333333 1.0000000  0.3333333  0.3333333
6337.txt  -0.3333333 0.3333333  1.0000000 -0.3333333
7017.txt   1.0000000 0.3333333 -0.3333333  1.0000000
```

The **lsa** and **LSAfun** packages provide other useful functions to assist analysis. The function **choose.target** returns randomly selected terms within a particular range of distances from a focal term or phrase. One can assess contextual differences with **conSIM**. Paragraphs can be analyzed to assess the coherence of the sentences or to create a "typical sentence" of that paragraph based on its distance measures with other sentences in that paragraph with **genericSummary**. In R, there are help pages for functions. One can quickly access these help pages with a leading question mark.

```
?coherence
```

R also provides a set of functions to visualize text. One such tool is a word cloud. Word clouds can be very informative. To create a word cloud, we need to load a new package **wordcloud**. We will install it and its associated package **RcolorBrewer** beforehand. Associated packages are loaded through the **dependencies = TRUE** parameter.

```
install.packages("wordcloud", dependencies = TRUE)
library(wordcloud)
```

Recall that we previously created **TDM** and **myTerms**. We will now plot the frequency of the words in **myTerms** as they appear in TDM. The function **apply** returns the results of a function—in this case, **sum** with a parameter 1 on the rows of the matrix **TDM**[15]. Doing so will create a term count vector, which adds 1 to the sum of the occurrences of each term. After the matrix is transposed (i.e., turned 90 degrees) with the function **t**, we can provide it as a parameter to the function **wordcloud** that does exactly as its name implies. In this case, we added to **wordcloud** parameters that limit the diagram to only terms with a frequency of at least 1 (though, admittedly, doing so is superfluous). The parameter **random.order** specified not to apply a random order to the plot[16]. Adding colors helps one to interpret the word cloud: one can do so through the **color** parameter. The resulting plot is informative: it shows that, in our weighted matrix, "card" has a disproportionate weight in the space. Figure 6 shows the resulting wordcloud.

```
Term_count <-apply(TDM2,1,sum)
TCT <- t(Term_count)
wordcloud(myTerms, TCT, min.freq=1, random.order=FALSE, color=brewer.pal(8, "Dark2"))
```

[15] Details about **apply** appear at https://stat.ethz.ch/R-manual/R-devel/library/base/html/apply.html.

[16] More details on **wordcloud** appear at https://cran.r-project.org/web/packages/wordcloud/wordcloud.pdf.

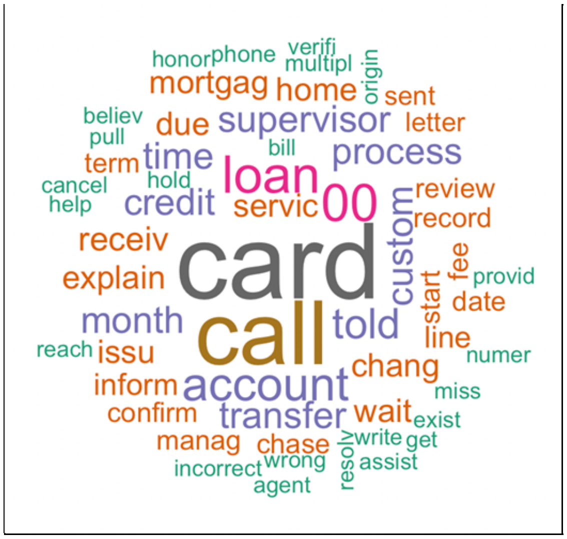


Figure 6. Wordcloud Depicting the Most Commonly Used Terms in the MiniComplaints Dataset

3.3 Research Application: A Realistic Analysis of the Complaint Data Corpus

Having run LSA on a very small set of documents in Sections 3.1 and 3.2, in this section, we describe what one can more realistically do with LSA. This demonstration shows one type of application that behavioral scientists can use text analysis for. Behavioral scientists often turn to interviews with informed people or resort to reading about their experiences and opinions as a way to gain insight into organizational processes and their related social, behavioral, and organizational and psychological issues. In this section, the code demonstrates using text analysis to extract possible insight about the meaning of trust as it is used in real-life complaints. The demonstration then shows that one could gain some plausible insight from the semantic space by projecting sentences onto it. The data deal with consumer financial complaints. Showing the applicability of consumer complaints to IS research, Coussement et al. (2015) studied related data dealing with consumer reviews.

In this section, we use the text manipulation package **tm**.

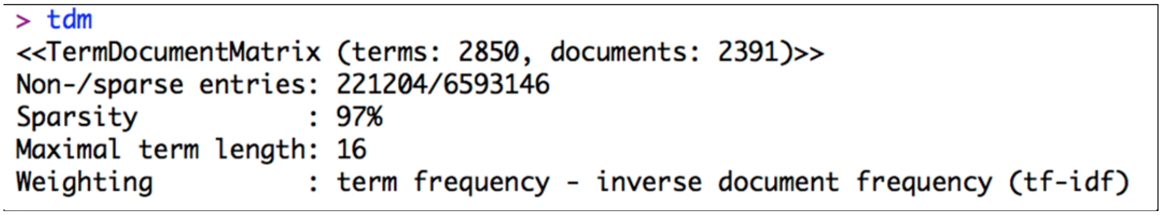
```
# Load required code libraries
library(cluster)
library(tm)
library(LSAfun)
```

To create the text matrix, we now use **tm**’s **DirSource** function, which imports text faster than **textmatrix** does. In relation to Figure 1, this step corresponds to the row of arrows in “preparing documents”.

```
# Replace the [...] with the path in to the FinancialComplaints directory.
source_dir = '[...]'
# source_dir = 'C:/Users/username/Desktop/FinancialComplaints' # Windows example
# source_dir = '~/Desktop/FinancialComplaints' # Mac/Unix example
# We shall now create a corpus in memory
raw_corpus <- VCorpus(doc_source, readerControl=list(language='en'))
```

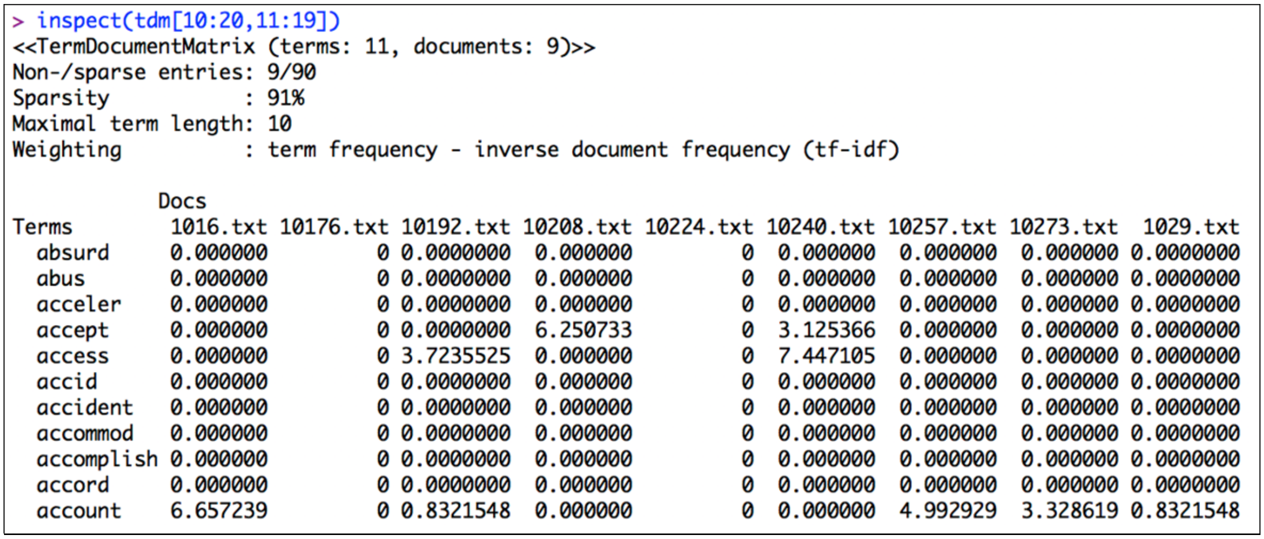
We will next create a TDM out of that corpus. In this case, we will use an alternative function **TermDocumentMatrix**. This function too appears in the **tm** package. One can instruct this function to already include the stop words and the weighting. Because we know that these documents contain meaningless blackout characters, we will first add those to the **tm** stop word list. Those instructions are specified in the **control = list()**. Note that we create a weighted TDM in one step. In relation to Figure 1, this step corresponds to the arrows "pre-processing text", "creating the TDM", and "preparing documents".

```
stoplist <- c(stopwords("en"), "xx", "xxxx","xx/xx/xxxx","xxxx/xxxx/", "xxxxxxxxxxxx","xxxxxxxx")
tdm <- TermDocumentMatrix(raw_corpus,
  control=list(removePunctuation = TRUE,
            removeNumbers = TRUE,
            tolower = TRUE,
            stopwords = stoplist,
            stemming = TRUE, # snowball stemmer
            weighting = function(x) weightTfIdf(x, normalize = FALSE), # Weight with tf-idf
            bounds=list(global=c(5,Inf)))) # Keep only 5 or more appearances, to accelerate
                    # space creation for purposes of this guide
# The tdm matrix is very sparse
tdm
```



```
> tdm
<<TermDocumentMatrix (terms: 2850, documents: 2391)>>
Non-/sparse entries: 221204/6593146
Sparsity           : 97%
Maximal term length: 16
Weighting          : term frequency - inverse document frequency (tf-idf)
```

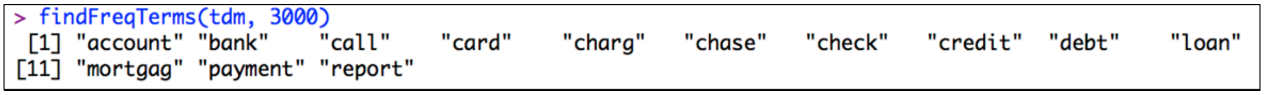
```
# Still, it may be very sparse, but inspecting it we can show the occasional non-zero value
inspect(tdm[10:20,11:19])
```



```
> inspect(tdm[10:20,11:19])
<<TermDocumentMatrix (terms: 11, documents: 9)>>
Non-/sparse entries: 9/90
Sparsity           : 91%
Maximal term length: 10
Weighting          : term frequency - inverse document frequency (tf-idf)

            Docs
Terms        1016.txt 10176.txt 10192.txt 10208.txt 10224.txt 10240.txt 10257.txt 10273.txt  1029.txt
  absurd     0.000000         0 0.0000000  0.000000         0  0.000000  0.000000  0.000000 0.0000000
  abus       0.000000         0 0.0000000  0.000000         0  0.000000  0.000000  0.000000 0.0000000
  acceler    0.000000         0 0.0000000  0.000000         0  0.000000  0.000000  0.000000 0.0000000
  accept     0.000000         0 0.0000000  6.250733         0  3.125366  0.000000  0.000000 0.0000000
  access     0.000000         0 3.7235525  0.000000         0  7.447105  0.000000  0.000000 0.0000000
  accid      0.000000         0 0.0000000  0.000000         0  0.000000  0.000000  0.000000 0.0000000
  accident   0.000000         0 0.0000000  0.000000         0  0.000000  0.000000  0.000000 0.0000000
  accommod   0.000000         0 0.0000000  0.000000         0  0.000000  0.000000  0.000000 0.0000000
  accomplish 0.000000         0 0.0000000  0.000000         0  0.000000  0.000000  0.000000 0.0000000
  accord     0.000000         0 0.0000000  0.000000         0  0.000000  0.000000  0.000000 0.0000000
  account    6.657239         0 0.8321548  0.000000         0  0.000000  4.992929  3.328619 0.8321548
```

Next, to identify the most frequent terms in the matrix (in this case, those with a frequency of at least 3000), we will run the **findFreqTerms** command.

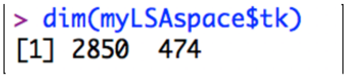
findFreqTerms(tdm, 3000)



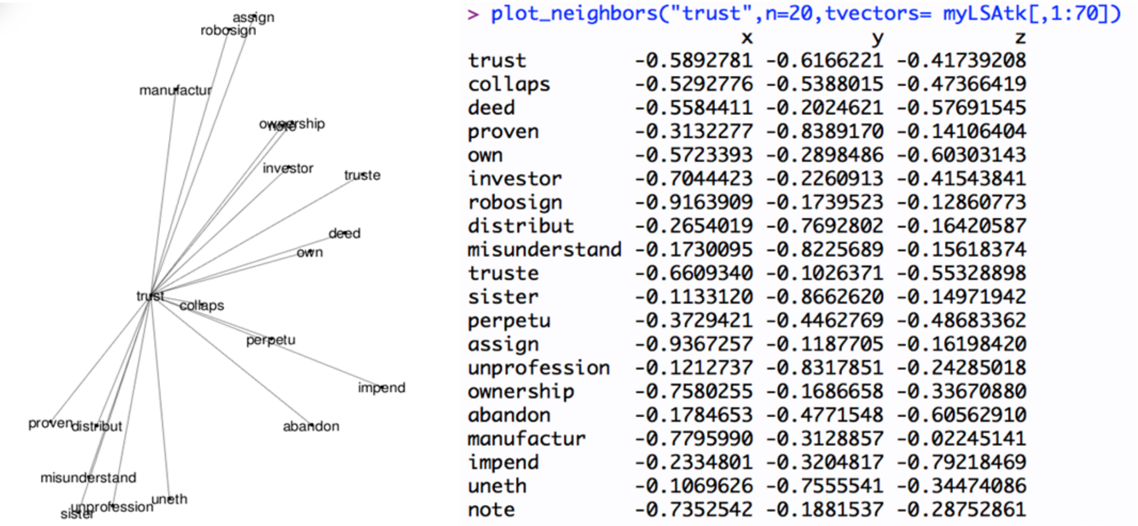
```
> findFreqTerms(tdm, 3000)
 [1] "account" "bank"    "call"    "card"    "charg"   "chase"   "check"   "credit"  "debt"    "loan"
[11] "mortgag" "payment" "report"
```

To create a semantic space out of the tdm matrix in the code, we can run the **lsa** command. Doing so will run an SVD. The next snippet of code shows how to do that and how many rows/columns the **$tk** matrix has before printing the 20 closest neighbors of the term “trust”. In relation to Figure 1, this step corresponds to the “calculating SVD” arrow. Figure 7 shows the result of the **plot_neighbors** command on those data with the 20 nearest neighbors of the word/term “trust” in that sematic space.

```
myLSAspace <- lsa(tdm, dims=dimcalc_share());
dim(myLSAspace$tk)  # Check how many rows/columns the tk matrix has
myLSAtk = t(myLSAspace$sk * t(myLSAspace$tk))
plot_neighbors("trust",n=20,tvectors= myLSAtk[,1:70])  # Use only the first 70 dimensions
```



```
> dim(myLSAspace$tk)
[1] 2850  474
```



```
> plot_neighbors("trust",n=20,tvectors= myLSAtk[,1:70])
                       x          y           z
trust         -0.5892781 -0.6166221 -0.41739208
collaps       -0.5292776 -0.5388015 -0.47366419
deed          -0.5584411 -0.2024621 -0.57691545
proven        -0.3132277 -0.8389170 -0.14106404
own           -0.5723393 -0.2898486 -0.60303143
investor      -0.7044423 -0.2260913 -0.41543841
robosign      -0.9163909 -0.1739523 -0.12860773
distribut     -0.2654019 -0.7692802 -0.16420587
misunderstand -0.1730095 -0.8225689 -0.15618374
truste        -0.6609340 -0.1026371 -0.55328898
sister        -0.1133120 -0.8662620 -0.14971942
perpetu       -0.3729421 -0.4462769 -0.48683362
assign        -0.9367257 -0.1187705 -0.16198420
unprofession  -0.1212737 -0.8317851 -0.24285018
ownership     -0.7580255 -0.1686658 -0.33670880
abandon       -0.1784653 -0.4771548 -0.60562910
manufactur    -0.7795990 -0.3128857 -0.02245141
impend        -0.2334801 -0.3204817 -0.79218469
uneth         -0.1069626 -0.7555541 -0.34474086
note          -0.7352542 -0.1881537 -0.28752861
```

Figure 7. The 20 Nearest Neighbors of “Trust” in the FinancialComplaints Dataset

Showing those words in a heat map adds a compelling visualization[17]. To do so, we will first install the **gplot** package and its dependent packages. Figure 8 shows the resulting heatmap.

```
install.packages("gplots", dependencies = TRUE)
library(gplots)


# Extract the closest words to “trust” (a list of their distances as a named vector).
```

[17] More on heatmaps can be found at http://sebastianraschka.com/Articles/heatmaps_in_r.html

```
words<-neighbors("trust",n=20,tvectors= myLSAtk[,1:70])

# Extract the actual words, and find the distances in the space.
myCosineSpace2 <- multicos(names(words), tvectors= myLSAtk[,1:70], breakdown=TRUE)
heatmap.2(myCosineSpace2)
```

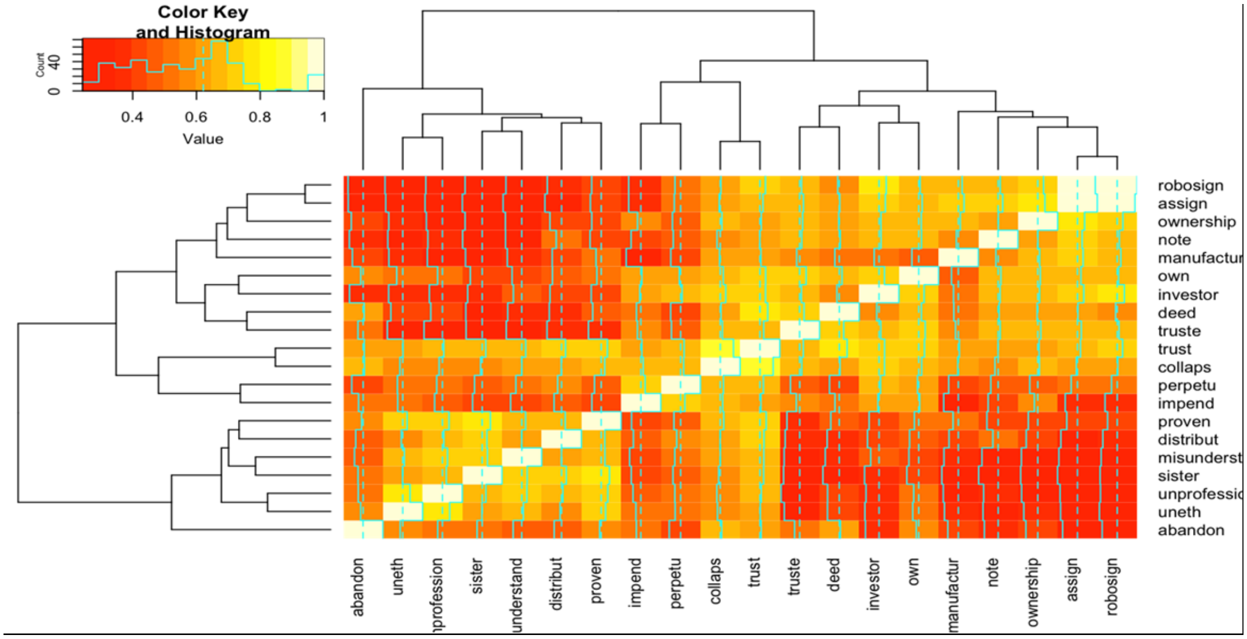
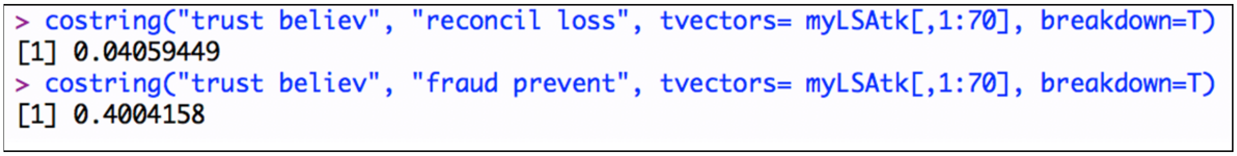


Figure 8. A Heatmap of the Most Frequent Words in Stack Exchange

The heatmap shows two clusters that emerge from among the 20 closest terms to “trust” in this corpus. Referring to the list of terms below the heatmap (the same list as on its right), the left-hand side terms seem to relate to interpersonal trust and the lack of it. That list contains terms such as “power”, “abandon”, “incid” (incident), “uneth” (unethical), and “unprofession” (unprofessional). On the bottom right-hand side list of terms, the terms seem to relate more to the financial structure of “a trust” with “own”, “deed”, and, tellingly, the stemmed form of “trustee”. The clustering of these terms in the heatmap suggests that “trust” indeed has two distinct meanings in our corpus. The two clusters overlap, and the terms “trust” and “collaps” (collapse) straddle the two clusters. That trust and collapse join in this way and that the two jointly straddle both clusters of meaning is not surprising coming from a corpus about complaints.

Having created a semantic space, we can now project sentences onto it to calculate how similar those vectors of words are. For example, does interpersonal trust in a banking context relate more strongly to gaining restitution after being wronged or to preventing the wrong in the first place? We can compare “trust” plus “believ” to other terms to compare their distances. The **costring** does that. In relation to Figure 1, this step corresponds to the “projecting into semantic space” and “calculating cosine similarities or other methods” arrows.

```
costring("trust believ", "reconcil loss", tvectors= myLSAspace$tk[,1:75], breakdown=T)
costring("trust believ", "fraud prevent", tvectors= myLSAspace$tk[,1:75], breakdown=T)
```



```
> costring("trust believ", "reconcil loss", tvectors= myLSAtk[,1:70], breakdown=T)
[1] 0.04059449
> costring("trust believ", "fraud prevent", tvectors= myLSAtk[,1:70], breakdown=T)
[1] 0.4004158
```

The results show that “fraud prevent” is considerably closer to the “trust believ” string than “reconcil loss”. Conceptually, one might extract plausible insight from that projection: it might suggest that, among those people whose postings appear in this corpus, preventing fraud is more important to trust beliefs (i.e., semantically closer) than reconciling loss.

3.4 Applying Sparse SVD: A Demonstration on Stack Exchange

In this example, we apply LSA to an even more realistically sized dataset and, in the process, add slightly more complex R code. Large data, especially if extremely sparse, make applying SVD as we do above with the **lsa** function less practical. The code in this section introduces an alternative sparse SVD method. That method is part of the **RSpectra** package. Running a sparse SVD requires preparing the data in the TDM beforehand as opposed to the previous code snippet where the **lsa** function prepared the data through specified parameters. In the case of the sparse SVD algorithm in this section, that code will be explicitly run prior to running the sparse SVD function. We downloaded the data we analyze in this section from *Stack Exchange*. The code used to create the data appears in the footnote[18].

3.4.1 Creating the Semantic Space

The code uses the **tm**, **RSpectra**, and **lsaFun** packages. The additional **RSpectra** package is a wrapper around a C++ library that calculates SVD on sparse matrices efficiently. For readers who start the code at this point, we will also install the previously installed R packages[19] together with R packages that those packages depend on. Because readers may have already installed these packages in the previous sections of this guide, this time the code verifies if there one needs to install the package beforehand.

```
if (!require("tm")) {
  install.packages("tm", dependencies = TRUE)
  library(tm)
  }
if (!require("RSpectra")) {
  install.packages("RSpectra", dependencies = TRUE)
  library(RSpectra)
  }
if (!require("LSAfun")) {
  install.packages("LSAfun", dependencies = TRUE)
  library(LSAfun)
   }
```

[18]
```
### Query from Stack Exchange:
### https://data.stackexchange.com
### (Query Specification: SELECT TOP 10000 Text, PostId,
### UserDisplayName, CreationDate, Score FROM Comments)
### Data downloaded as .csv file

# Sets the path to the working directory
setwd("path")

# Reads the .csv file downloaded (notice headers are used and stringAsFactors=FALSE
# avoids character vectors to be converted into factors)
File <- read.csv(file="QueryResults.csv", header=TRUE, stringsAsFactors=FALSE)

# Sets the path to the file folder as the new working directory
setwd("path/StackExchange")

# Loop to store each line in the .csv file as an independent .txt file
for (i in 1:10000)  {  cat(File$Text[i],file=paste0(i,".txt"))  }
```

[19] https://cran.r-project.org/web/packages/RSpectra/RSpectra.pdf

```
if (!require("gplots")) {
  install.packages("gplots", dependencies = TRUE)
  library(gplots) }
```

With the packages loaded, we next need to load the individual txt documents into a TDM. We will do so using the **tm DirSource** function. This function loads all the text files in a directory into a corpus. The code specifies that the documents reside in a directory source (meaning that the directory contains documents as files) and that the folder to look in is *StackExchange.* The parameter **recursive = True** indicates that we will build the corpus also from files in subdirectories. In relation to Figure 1, this step corresponds to the arrows in the row “preparing documents”.

```
doc_source <- DirSource('C:/…/StackExchange')
```

In the next line we create a corpus from these documents by reading each file into R. This function has several variants. In this case, the code uses **VCorpus** to load the entire content of all the documents into memory to enable faster processing. If the computer the code is running on has insufficient memory, then it is advisable to apply **DCorpus** instead to store the data on the hard drive. The **readerControl** argument specifies additional pre-processing options. Here, those additional options include only that the documents read in are in English.

```
raw_corpus <- VCorpus(doc_source, readerControl=list(language='en'))
```

At this point, we have loaded 10,000 posts into the corpus. The code will now create a function that replaces all characters that are not letters, spaces, or hyphens with the space character. We will apply this function to every document in the corpus. Such a function exists in other libraries. We show it here as a template for pedagogical purposes. In relation to Figure 1, this step and the next one correspond to the “pre-processing text” arrow.

```
remove_nonletter <- function(text) { return(gsub('[^a-z\\s\\-]+', ' ', text))}
```

We can now pre-process the corpus. We will run the **tm_map** function with the wrapper function **content_transformer**. This wrapper will cause the transformations to apply to the text of each document, not to its metadata. The parameter passed to **content_transformer** specifies what action to perform. The transformations cast the text to lowercase and remove standard English stop words and non-letter characters. Finally, in the last line of code in the snippet, the **tm_map** function stems the words in each document. Notice that the transformed output of each command is the input to the next command.

```
p_corpus <- tm_map(raw_corpus, content_transformer(tolower))
p_corpus <- tm_map(p_corpus, content_transformer(removeWords), tm::stopwords('en'))
p_corpus <- tm_map(p_corpus, content_transformer(remove_nonletter))
p_corpus <- tm_map(p_corpus, stemDocument)
```

The next command creates a TDM from the **p_corpus** output by applying the **TermDocumentMatrix** function. **TermDocumentMatrix** receives as parameters a corpus and a list of parameters that it delegates to functions it calls to create the matrix. The only argument passed in the example below is **bounds**. This parameter creates a limit that discards words that appear too frequently or too infrequently. In this case, the function will discard words that appear fewer than 10 times in the corpus. We run this function because terms that are extremely infrequent will take up a significant amount of space but will add little value to the semantic space. The exact cutoff will depend on the goals of the analysis and the nature of the corpus. In relation to Figure 1, this step corresponds to the "creating the TDM" arrow.

```
tdm <- TermDocumentMatrix(p_corpus, control = list(bounds = list(global = c(10, Inf))))
```

The **tdm** matrix is now transformed with **sparseMatrix** into a sparse matrix before it is passed to an SVD in the **RSpectra** package. While unnecessary for small corpora, this step is recommended in large corpora because it can increase speed and reduce memory by orders of magnitude[20].

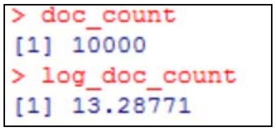
```
sparse_tdm <- Matrix::sparseMatrix(i = tdm$i, j = tdm$j, x = tdm$v, dims = c(tdm$nrow,
tdm$ncol))
```

The next command assigns the names of the rows and columns of the newly created **sparseMatrix** to be the same as the names of the rows and columns of the **TermDocumentMatrix**, which will make looking things up easier in subsequent steps.

```
dimnames(sparse_tdm) <- dimnames(tdm)
```

Another step that needs to be taken before performing SVD on this matrix is weighting. The code will apply log-entropy weighting to this corpus. To do so, the code will retrieve the number of items in the second dimension of the **sparse_tdm** matrix (i.e., the number of documents) and calculate the log base 2 of that count. There should be 10,000 documents and the log base 2 of that 10,000 should be about 13.29. In relation to Figure 1, this part and the next snippets correspond to the arrows "weighting the TDM" and "calculating SVD".

```
doc_count <- dim(sparse_tdm)[[2]]
log_doc_count <- log2(doc_count)
```



```
> doc_count
[1] 10000
> log_doc_count
[1] 13.28771
```

Next, the code will create a copy of **tdm** and name it **weighted_tdm**. The code will then apply the log weighting function to each non-zero entry in that matrix. The code does this with the **vapply** function. That function receives as parameters a vector, a function, and the data type of the result. The **sparseMatrix** format stores rows as triples that can be mapped to the index for the row and column and a value but not storing anything in cells with the value 0. The values for each cell are stored in the attribute *x* which is a vector, so instead of applying the log transformation to every cell in the matrix, the code will only apply it to each value in the attribute *x*.[21] Because **log2** of a zero is negative infinity, the value of each cell shall be incremented by .00001.

[20] This section of code also illustrates the **::** syntax in R. This operator allows one to invoke a function from a package without loading the entire package. The **Matrix** package is bundled with all distributions of R, so one does not need to install it. As a rule, if one needs only a few functions in a package, then one should apply the **::** operator.

[21] The code will create an anonymous function that will only exist for this one instruction. This is similar to how functions are normally defined except that it is not assigned to a name and it is unnecessary to surround its returned value with the function return.

```
weighted_tdm <- sparse_tdm
weighted_tdm@x <- vapply(sparse_tdm@x, function(x) log2(x+.00001), numeric(1))
```

The code will now create a count of global frequencies of each word. We will use this count in the entropy function. To do so, the code will call the **rowSums** function from the **Matrix** package and run it on the **sparse_tdm** matrix we previously created, which will sum all the cells in each row. The code will then assign the names of these sums to be the same as the term names from **tdm** in order to make future indexing easier.

```
gf <- Matrix::rowSums(sparse_tdm)

names(gf) <- dimnames(sparse_tdm)$Terms
```

The last step before calculating the entropy for each word is to create a helper function that will be applied to each row in **sparse_tdm** to calculate its entropy. This function will be named **partial_entropy** because the entropy is the sum of all of these parts plus .00001.

```
partial_entropy <- function(tf, gf) {

                                p <- tf/gf

                                return((p*log2(p))/log_doc_count)
                            }
```

The code will next calculate the entropy of each word by creating a vector to store the entropy values, assigning names to the vector, and iterating through the rows in **sparse_tdm** to calculate the entropy for each row. After creating the vector and naming it, the code will iterate through a for-loop. The loop will run the code inside the curly brackets multiple times controlled by **i**. The code assigns the value of **i** to run between 1 and the number of terms in sparse_tdm. The code selects the rows that have a frequency greater than 0 by writing a comparison that produces a vector of the same shape as **word_row**. This vector contains a value of either **TRUE** or **FALSE** depending on whether the condition is true. The code selects only those rows whose value is **TRUE**. The code then uses those indexes as the list of columns to be extracted from **word_row**. This provides a list of frequencies greater than zero in **word_row**. The last line adds 1 to the sum of the result of the **mapply** function. The **mapply** function applies a function to a combination of lists and values. The code calls the **partial_entropy** function created earlier. This function is applied to each cell in the **non_zero_frequencies** vector. The code also specifies that regardless of which value is used from the **non_zero_frequencies** vector, the value of **gf** that is passed to the **partial_entropy** function should be the i^{th} entry in **gf** (i.e., the global frequency of the word calculated previously).

```
word_entropy <- numeric(dim(sparse_tdm)[[1]])

names(word_entropy) <- dimnames(sparse_tdm)$Terms


for(i in 1:dim(sparse_tdm)[[1]]){

   word_row <- sparse_tdm[i,]

   non_zero_frequencies <- word_row[which(word_row>0)]

   word_entropy[i] <- 1.0 + sum(mapply(partial_entropy, non_zero_frequencies, gf=gf[i]))

}
```

Now that there are entropy values for each word in the corpus, the code can apply weighting to the corpus by multiplying the **weighted_tdm** matrix by the entropy value of each word. This is done with the **sweep** function that iterates through a matrix along a specified axis and applies a function to it with a provided additional argument. In this case, the code is applying the multiplication function along the first axis of the **weighted_tdm** matrix using a second argument from the word_entropy vector.

```
weighted_tdm <- sweep(weighted_tdm, 1, word_entropy, '*')
```

With the weighted TDM ready, the code can now run an SVD. The code will use the **svds** function from the **RSpectra** package. This function serves the same objective as **svd** except that it is designed to work on sparse matrices and only calculate the top *k* rows instead of calculating all the rows and then selecting the *k* best. This function significantly improves performance and, thus, enables one to analyze very large corpora. The code will create two matrices for future use as the result of performing SVD with 300 dimensions and name them **su_mat** and **sv_mat**. When naming the dimensions of these two matrices, the code applies the term and document names for one dimension and labels the other with the numbers 1 through 300 (since they are reduced matrices). Note that **svt_matrix** requires that the **v** matrix be transposed. One can do so with the **t** function from the **Matrix** package. The transpose is necessary in order to maintain the correspondence between columns and documents.

```
space <- svds(weighted_tdm, 300)
su_mat <- space$d * space$u
svt_mat <- space$d * Matrix::t(space$v)

#Assign names
dimnames(su_mat) <- list(dimnames(weighted_tdm)[[1]], 1:300)
dimnames(svt_mat) <- list(1:300, dimnames(weighted_tdm)[[2]])
```

The data are now ready for analysis. As exemplars, the code will compare the closest neighbor terms in *Stack Exchange* to three popular programming languages. The results are rather informative, perhaps suggesting the obvious that different programming languages are applied to different types of problems. This is a good example of deriving meaning from the “semantic neighborhood” [Kintsch, 2001, p. 177] of words in question: Terms that semantically closer to a given word help define its meaning in the specific context of the corpus. In this case the analysis shows the kinds of contexts, identified by the closest terms, in which each of the programming languages are used in this corpus. In relation to Figure 1, this part corresponds to the arrows “projecting into semantic space” and “calculating cosine similarities or other methods”. Figure 9 shows the results of the **plot_neighbors** functions.

```
plot_neighbors("python",n=20,tvectors= su_mat)
plot_neighbors("java",n=20,tvectors= su_mat)
plot_neighbors("javascript",n=20,tvectors= su_mat)
```

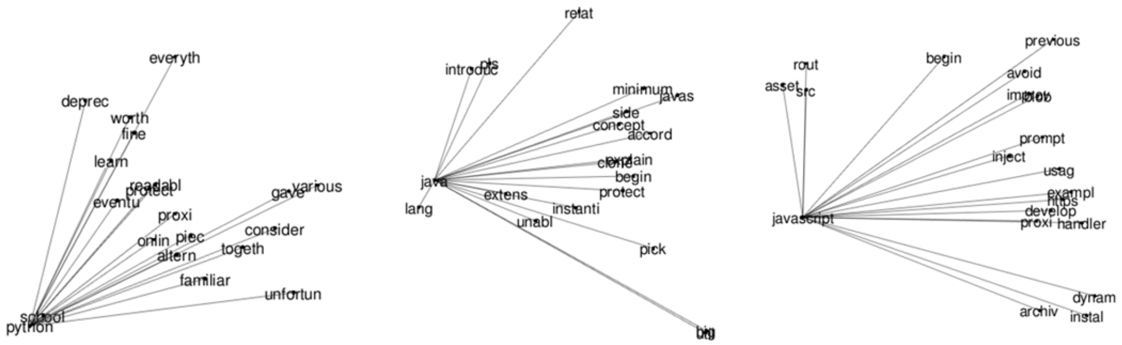


Figure 9. The Closest Neighbor Terms in Stack Exchange to Three Popular Programming Languages

We can also apply code to identify the most frequent words in *Stack Exchange*. In this case, they are, as one might expect, “package” and “answer”. Figure 10 shows the results of the **wordcloud** functions.

```
if (!require("wordcloud")) {
  install.packages("wordcloud", dependencies = TRUE)
  library(wordcloud)
  }
Term_count <-apply(su_mat,1,sum)
TCT <- t(Term_count)
myTerms <- rownames(su_mat)
wordcloud(myTerms, TCT, min.freq=1, random.order=FALSE, color=brewer.pal(8, "Dark2"))
```

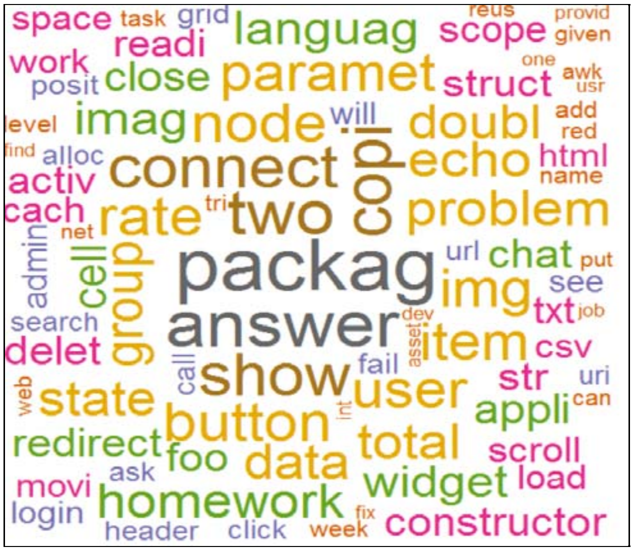
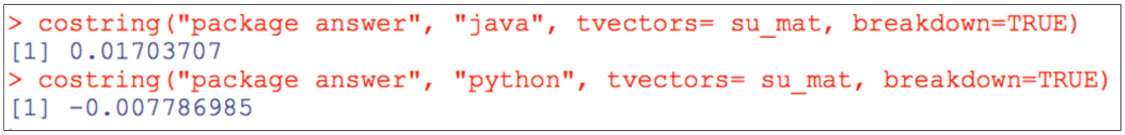


Figure 10. A Wordcloud of the Most Frequent Words in Stack Exchange

Additionally, pseudo-documents can be projected onto the sematic space to gleam possible insight about its content and maybe what associations the people whose posting are included in this corpus made. In this case, we will ask whether the combination “package answer”, combining the two most frequent words in the corpus, is more related Java or to Python. Interestingly, these two most frequent words are more closely related to Java rather than Python. Possibly, answers about packages are requested more in the context of Java than Python. This demonstration shows the power of LSA to allow researchers a glimpse at *indirect* information in the corpora, information a researcher reading the tens of thousands of Q&A might have skipped because acquiring such insight requires reading beyond the texts. Projecting arbitrary sentences that may not even occur as is in the texts and then comparing their distances thus may allow an overview of how information that may not even be directly recorded in the text can be acquired. Such inference is an example of the kind of “family resemblances” (Wittgenstein, 1953) that Kintsch (2001) discusses as a key advantage of LSA. In relation to Figure 1, this step corresponds to the arrows in “projecting pseudo-documents”.

```
costring("package answer", "JAVA", tvectors= su_mat, breakdown=TRUE)

costring("package answer", "Python", tvectors= su_mat, breakdown=TRUE)
```



```
> costring("package answer", "java", tvectors= su_mat, breakdown=TRUE)
[1] 0.01703707
> costring("package answer", "python", tvectors= su_mat, breakdown=TRUE)
[1] -0.007786985
```

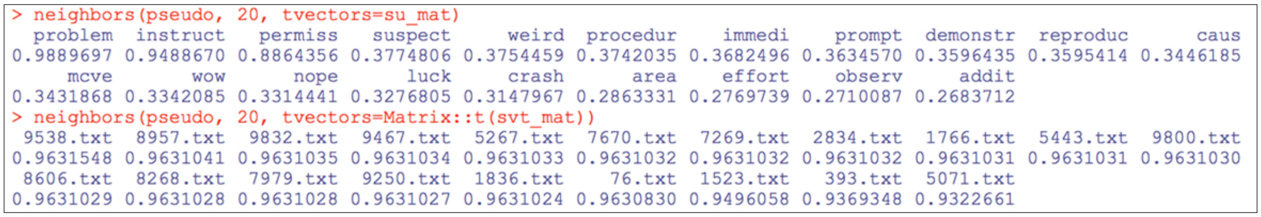
Another application of projecting new documents is presented in the next code snippet. A complete sentence in English, rather than terms that we know already appear in the corpus, will be created. Then, the previous pre-processing steps that were applied to the corpus when the semantic space was created will be applied to it, followed by weighting, and projecting it onto the semantic space to identify the terms most associated with it. In contrast to the previous snippet where relationships between known terms were compared, in this case the code will identify possible “answers” to that complete sentence.

The first line of this snippet creates the sentence. The next lines perform the same pre-processing steps that we performed to create the semantic space. The last two lines are the only ones that differ significantly. The first of these two lines multiplies the log weights by the values from the **word_entropy** table. When creating the semantic space, the entropy of each word in the corpus was calculated. That weighting is the one that was used for a global weight. However, in this case, we are only interested in the weights of words from the pseudo-document. This is where naming the indices earlier comes in handy. By using the same names as used earlier, it is possible to look up their entropy weights by the words themselves. As a result, this function only needs to multiply matches by one another. The last line then takes the column sums of the log-entropy weights (currently stored in **pseudo**) and the **su_mat** matrix. The same trick is used to pull only the relevant rows from **su_mat** that were applied to extract entropy from the **word_entropy** table. Notice that a comma follows the list of names because the code wants each column from that table instead of just a single value. Once the code sums across each of the columns, the result is a vector that represents the pseudo-document and can be used just like the vector of any term or document in the **neighbor** function (among others). In relation to Figure 1, this step corresponds to the “pre-processing pseudo-documents” and “weighting pseudo-documents” arrows.

```
pseudo <- 'Tell me about overflow problems'

pseudo <- tolower(pseudo)

pseudo <- removeWords(pseudo, tm::stopwords('en'))

pseudo <- remove_nonletter(pseudo)

pseudo <- stemDocument(PlainTextDocument(pseudo))

pseudo <- termFreq(pseudo)

pseudo <- vapply(pseudo, function(x) log2(x+.00001), numeric(1))

pseudo <- mapply(function(x, y) x*y, pseudo, word_entropy[names(pseudo)])

pseudo <- colSums(pseudo * su_mat[names(pseudo),])
```

For illustration, the next snippet identifies the terms most close to the sentence entered in the first line in the snippet above, and lists the documents where one may wish to start looking for content close to that sentence. The results suggest that overflow problems as the posts on *Stack Exchange* discussed are associated mostly with terms such as instructions, permission, reproduce, and crash and that, sometimes, one might describe the results as weird. These problems appear in a wide range of documents.

```
neighbors(pseudo, 20, tvectors=su_mat)

neighbors(pseudo, 20, tvectors=Matrix::t(svt_mat))
```



```
> neighbors(pseudo, 20, tvectors=su_mat)
  problem  instruct   permiss   suspect     weird  procedur    immedi    prompt  demonstr  reproduc      caus 
0.9889697 0.9488670 0.8864356 0.3774806 0.3754459 0.3742035 0.3682496 0.3634570 0.3596435 0.3595414 0.3446185 
     mcve       wow      nope      luck     crash      area    effort    observ     addit 
0.3431868 0.3342085 0.3314441 0.3276805 0.3147967 0.2863331 0.2769739 0.2710087 0.2683712 
> neighbors(pseudo, 20, tvectors=Matrix::t(svt_mat))
 9538.txt  8957.txt  9832.txt  9467.txt  5267.txt  7670.txt  7269.txt  2834.txt  1766.txt  5443.txt  9800.txt 
0.9631548 0.9631041 0.9631035 0.9631034 0.9631033 0.9631032 0.9631032 0.9631032 0.9631031 0.9631031 0.9631030 
 8606.txt  8268.txt  7979.txt  9250.txt  1836.txt    76.txt  1523.txt   393.txt  5071.txt 
0.9631029 0.9631028 0.9631028 0.9631027 0.9631024 0.9630830 0.9496058 0.9369348 0.9322661 
```

And, since R runs in memory, after one has run the code, we recommend that one remove objects that are not currently in use from memory. It is recommended to apply functions such as **rm()** (removes an object), **rm(list = ls())** (removes all objects in memory), or **ls()** (list all the objects in memory) as necessary.

```
ls()

rm(doc_source)
```

4 The Potential and Limitations of LSA Modeling

To better understand how to use a methodology, one needs to think of it in the context of its epistemology and philosophy of science. This section accordingly briefly discusses what can be prudently done methodologically with LSA and discusses some key epistemological and methodological specialties and limitations that need to be considered. We begin with the technical aspects of LSA, which researchers have discussed previously, and then discuss issues of validity and reliability informed by previous IS guidelines (Boudreau, Gefen, & Straub, 2001; Straub, Boudreau, & Gefen, 2004).

4.1 The Technical Side

Before addressing the reliability and validity issues of LSA, we review some analysis and reporting issues. Evangelopoulos et al. (2012) summarizes these issues well. They recommend that researchers should:

1. Consider their research's objective before deciding what analysis to perform on the matrices that LSA produces. Evangelopoulos et al. discuss and demonstrate how to classify documents, cluster them, and conduct factor analyses on the items. This recommendation of course extends to other types of analysis than one can perform on LSA-derived matrices such as PCA and CBSEM in which case one should also consider whether to derive correlations or cosines (Gefen & Larsen, Forthcoming).
2. Researchers should also be cautious in applying rules of thumb threshold values borrowed from other research epistemologies (such as choosing items loadings to be above a certain value when running factor analyses). Evangelopoulos et al. (2012) argue that, based on their experience, the meaning of the term-factors derived in a factor analysis may be lost if too rigorous rules of thumb are applied. Statistically too that recommendation is correct: it cannot be assumed that the cosines (or any other measure) derived from the LSA matrices necessarily have a normal or any other assumed distribution. Hence, applying rules of thumb that assume a specific distribution is unavoidably introducing misinterpretations of the results.. Further, as we discuss in Section 4.2, because LSA deals with language, it will inevitably produce many cross loadings of terms on factors. That is, terms will likely load on many factors (representing the shared meaning across those terms) because of the inherent nature of words to display polysemy (multi-meaning) and polysemousness (ambiguity), to be heterosemous

(different meanings depending on other words in the sentence), and to be used as metonyms (a word or expression used as a substitute for something else with which has a close association, such as referring to the U.S. Government as the White House). Assuming a clean factor loading on a large sample of terms derived from real-life language usage corpora ignores the nature of language.

3. Likewise, researchers should not hide crucial information that others may need to understand the exact parameters they applied. In the case of LSA that includes inter alia whether stop words and stemming were applied. That recommendation is not unique to LSA. Researchers have suggested equivalent recommendations for PLS and CBSEM in the *MIS Quarterly* guidelines (Gefen, Rigdon, & Straub, 2011).
4. Researchers should also try more than one transformation on the data, such as TF-IDF and log-entropy, and choose the transformation that best applies to the research question and data. Of course, there are many other transformations, including Standard Boolean, TF weights, latent Dirichlet allocation, latent Dirichlet allocation multicore, and others. (See https://cran.r-project.org/web/packages/lsa/lsa.pdf for a detailed listing of available transformations.). However, we need to clarify this recommendation. Like data transformation in other contexts such as CBSEM, researchers should look into their data when making such transformations rather than fishing for the best-fitting model (Muthén & Muthén, 2010).
5. And, as in much previous research on LSA (e.g., Landauer et al., 2004), researchers should investigate alternative LSA dimensionalities before they choose a specific level. This advice applies in principle also to PCA, and is crucial for the data we present in this paper (Hair, Black, Babin, & Anderson, 2005). If the code analyzing the *complaint data* in Section 3.2 is run on the entire 42,000 documents rather than on the much smaller sample, then, with 20 dimensions, the terms “trust” and “security” are quite similar with a cosine of 0.72, but, if one chooses 200 dimensions, then that cosine decreases to a nearly orthogonal 0.09.

To a large extent, these recommendations apply not only to LSA but also in principle to many other statistical methods. It is always important to choose the model carefully based on the research objectives and the type of data and to compare alternative models and alternative dimensionality (Hair et al., 2005). Indeed, CBSEM philosophy has long included the tenet that one should try alternative models with alternative factor patterns (e.g., Bollen, 1989). Likewise, researchers should not blindly apply rules of thumb (as the American Statistical Association also demands in the case of p-values (Wasserstein & Lazar, 2016)). And, it is a matter of integrity that researchers do not hide crucial information that others need to interpret their results (Gefen et al., 2011).

Putting those recommendations into context, the Appendix summarizes IS and related research that has applied LSA or equivalent methods. It also includes some technical aspects of that research. The Appendix shows that almost all the papers applied stemming, that almost all reported whether they used stop words (and almost all did), but that not all reported whether they rotated the semantic space or how many dimensions and terms they retained. The Appendix also demonstrates the power of LSA and related methods to do more than just classify documents and terms.

4.2 Reliability and Validity Considerations

The above technical aspects of LSA implementation are important: the mathematics, and specifically the SVD transformation and the data preparation before the SVD is run, will dictate different results based on the parameters applied to them. After all, LSA is at its core a mathematical transformation. Some technical aspects of that research are also included in the Appendix. To demonstrate that point, we will compare LSA to CBSEM-type survey research.

In a typical setting of CBSEM survey research, a predefined survey is given to a supposedly random and reasonably large enough representative sample of the population of interest. The methodology typically assumes an implicit *positivist* approach (at least to the extent of giving meaning to the p-value of the X^2 statistic) and should be *top down* at least in the confirmatory measurement model part of the analysis. The measurement model defines how the measurement items either load on (in reflective scales) or form into (in formative scales) the factors (often also known as latent variables). That being said, researchers do occasionally add ad hoc analyses in which they add or remove paths based on modification indices and other fit statistics that the CBSEM software provides, but those modifications should be only minor adjustments (and one should report them). The ad hoc additions should not be the driving force in the modeling: the theory should do that (Bollen, 1989). Because theory should drive CBSEM survey research, CBSEM analysis is *confirmatory.* In that context, the p-value takes on the meaning of the *probability* that the pattern in the sample is not random noise, but, rather, reflects what is happening in the population of interest. The statistics that CBSEM produces both at the path level and at the overall model level are based on the sample but are taken to *also refer to the population* that one draws the sample from. If a path or a model is significant in the sample, it is treated as if it probably applies also in the population.

In contrast, at its core, LSA is a *bottom-up*, *data-driven exploratory* method. Moreover, while one can run LSA on a very large sample of documents, no published method we know about provides a systematic approach to creating a representative random sample out of a population of interest. As such, because the sample does not claim to be a random representative sample of a population of interest, the p-values produced by analyses on LSA matrices are harder to interpret. This presents several threats to validity. As Straub (1989) and by Straub et al. (2004) note, there are three kinds of validity researchers should pay special attention to: instrumentation validity, internal validity, and external validity. Instrumentation validity is about the instrument constructs representing their real-world namesakes and that they measure what they are supposed to be measuring. In the case of survey research that means that the items reflecting or forming a scale actually measure that construct. Internal validity deals with the ability to rule out alternative hypotheses and dimensionality. External validity deals with the probability that the results obtained from a sample apply to the population of interest too, as well as to other populations and situations.

Applying these types of validity—developed with surveys and experiments in mind—to LSA analyses shows how relatively problematic drawing conclusions from an LSA sample can be and that one might need to add appropriate limitations to papers that apply LSA and equivalent methodologies. Comparing LSA to survey research demonstrates these limitations.

1. As the data cannot be assumed to come from a random sample, p-values produced by subsequent analysis on the matrices produced by LSA, such as a factor analysis or CBSEM run as in Gefen and Larsen (Forthcoming), relate to an unknown distribution. The p-value is, therefore, merely a function of the t, F, X^2, or any other statistic it refers to as the math indicates. Interpreting the p-value as probability that the sample statistics may apply to the population is conjecture, meaning that concluding from the corpora of sample documents to the world at large is at best tentative. In other words, unless the entire population of interest is studied, as was done when analyzing all the abstracts in a set of journals in range of dates (e.g., Sidorova et al., 2008), then the *external validity* of the results cannot be established.. The results of a convenient sample may be indicative, but they are not definitive. Having said that, however, the same is true of CBSEM survey research too unless the sample there is random and large. (If one includes the entire population of interest in the analysis, then, by definition, the p-value is no more that its mathematical function implies. The p-value in that case does not mean the probability that what applies to the sample may apply to the population.) Thus, *researchers should frame their conclusions accordingly.*
2. Additionally about the p-value produced by subsequent analyses and expanding on the previous point: if the corpora being analyzed in LSA cannot be convincingly shown to be a non-random sample then the distribution of the means in the sample cannot be assumed to be a normal distribution. That means that the interpretation of a p-value less than .05 as being significant is mere speculation. That poses a threat to *statistical validity*. That is another reason why the rules of thumb as applied to survey research do not readily apply to LSA data.
3. Internal validity may be at stake, too. Analyzing LSA data creates a threat to internal validity because no established rules define the best dimensionality of the LSA. (Contrast that with the eigenvalue greater or equal to 1 recommendation in PCA and factor analysis (Hair et al., 2005).) It is a matter of trial and error, with different dimensionality choices producing different result patterns in the LSA-derived matrices (Landauer et al., 2004). Inevitability, changing the number of dimensions produced by LSA will change the resulting results of subsequent factor analysis, clustering, or PCA. That constitutes a threat to internal validity and means that also *statistical reliability*, at least in its traditional meaning of obtaining equivalent results in a test-retest process or as an alpha coefficient across respondents (corpora in the case of LSA) cannot be established. Here too, researchers should be cautious to frame their conclusions with this limitation in mind. *A possible way to partially address the issue of reliability is to split the corpora into several datasets and then run exactly the same analysis on all those datasets*. Standard reliability tests could then be run across those dataset results, including Cohen's Kappa (Cohen, 1960).

4. Having alternative dimensionality also poses a threat to *instrumentation validity*. If alternative factoring of the same terms can be done, and, thus, also alternative meaning produced, then one could doubt conclusions that those factors represent specific constructs in the real world and not others. Instrumentation validity threats exist in CBSEM too (Straub et al., 2004), but in CBSEM there is a methodology for determining the optimal number of factors and establishing their reliability and factorial validity (e.g., Bollen, 1989). Establishing factorial validity In LSA is more complicated because the number of dimensions is orders of magnitude larger and because words, in stark contrast to survey items, are expected to carry more than one meaning and consequently to load on more than one factor. Assigning a construct name to a factor with LSA data is therefore bound to be more ambiguous than with CBSEM survey data. Some consequences of this are discussed in section 4.3
5. This lack of an established methodology for choosing the appropriate number of dimensions (Landauer et al., 2004) opens the floodgates to *fishing* by researchers. Fishing constitutes threats to both internal and external validity. There is nothing in the methodology to guide researchers on dimensionality, and therefore nothing to prevent them from choosing the dimensionality that best fits their objectives—and doing so even in the face of alternative dimensionality that produces different results. This makes fishing inevitable. That being said, fishing is not unique to LSA. It applies even to CBSEM, it is just that there are guidelines in CBSEM (e.g., Bollen, 1989). Here too, a possible way to partially address the issue is to split the corpora into several datasets and then run exactly the same analysis on all those datasets. Validation by splitting the original dataset was applied also by Coussement et al. (2015). If the results appear stable across corpora samples then presumably the results are reliable (at least in the context of test-retest reliability).
6. An often consequence of fishing is that researchers *over fit the model* to the data. Over fitting means that researches adjust their model to what the data analysis suggests will improve the fit indices. This is a problem in CBSEM too (Bollen, 1989). The inevitable consequence of over fitting is that the conclusions relate to the sample, but cannot be extended to the population that that the sample represents. CBSEM methodology suggests researchers should disclose their actions when they do so (Gefen et al., 2011). The same should be applied to LSA, too. Splitting the corpora into several datasets and running exactly the same analysis on all those datasets may provide a partial solution to those inevitable consequences. .
7. As a footnote and going outside the realm of LSA (and LDA), research has looked at methods to address some issues of the *interpretability* of the results of text analysis. This is an issue of *external validity.* One of the most important papers to address this issue is Chang et al.'s (2009) "Reading Tea Leaves: How Humans Interpret Topic Models". Chang et al. proposed verifying the interpretability of the results by explicitly adding unrelated words and topics into the results of text analysis and verifying that human subjects can identify those "intruders". They suggested two types of intruders: words and topics. In a *word intrusion*, a human subject is presented with several words that relate to a random latent topic derived from a semantic space. An additional unrelated (intruder) random word is then added to that list, and the subject is asked to identify that intruder word. Run many times on many random latent topics, that test provides a measure of the correctness of the text analysis factoring of terms by comparing it with human cognition. In *topic intrusion*, an equivalent experiment is run on human subjects, but this time with the intention of verifying that the human subject can identify an intruder document (rather than an intruder word). The intruder document is added to a list of documents that is composed of a factor of documents that the text analysis process produced.

4.3 More on Validation through Splitting the Data

Splitting the data and then comparing the results of running a machine learning algorithm on each split alone as a way to assess the reliability of the results is common (e.g., Coussement et al., 2015). Adapting the reliability assessment methods of test-retest and split-half from survey research may suggest a nuanced adaptation of those established statistical methods.

In survey research, assessing reliability through test-retest is about having the same person take the same survey twice. Test-retest reliability is calculated by comparing the results across the two or more times the survey was taken. Adapting this approach to LSA would entail splitting the semantic space into two or more spaces based on a time criterion, such as when the documents were created. If the LSA results of the first time unit resemble those of the subsequent time units, then test-retest reliability can be established. Alternatively, postings (e.g., in social media) by a “respondent” over several periods of time could be compared to a “gold standard” of texts from the same kind of media about the same topic. If that respondent’s postings retain their levels of close similarity to the gold standard texts over time, this too could be interpreted as a type of test-retest reliability.

Another reliability method commonly used in survey research is split-half reliability. In this approach, the data are split randomly into two or more datasets, the analysis is then performed separately on each dataset, and the results compared. Reliability is measured as a function of how close the results are across the datasets. In the context of LSA split-half reliability could be established by randomly splitting the corpora into several datasets, creating a semantic space out of each dataset with LSA, and then comparing the subsequent analyses. Reliability is shown through the degree to which the results across datasets are close to each other. In contrast to test-retest, the splitting in this case is done randomly, rather than by a time criterion. If the algorithm randomly selects a large set of documents from the corpora to create each semantic space, then, because this method conceptually resembles the idea of bootstrapping, statistical confidence levels can be also established around the mean values.

4.4 Cognitive Complexity

Exacerbating threats to interpretability is the enormous size of the data often analyzed with these methods, and the number of dimensions needed to represent it (Landauer et al., 2004). And, by extension, this threat means that if people cannot understand what the data mean then it cannot be ruled out that there are also threats to instrumental, statistical, internal, and external validities. Landauer et al. suggested that 250 to 400 might be the correct dimensionality needed in some applications of LSA, such as answering TOFEL multiple choice questions, and then went on to lament that such dimensionality cannot be visualized or otherwise analyzed easily by people. That being said, Landauer et al. also added that “the 300-dimension optimum is not a universal law, nor is there a theory to explain it” (p. 5125). The dimensionality Landauer et al. discussed related to a unique corpora universe, but as the Appendix shows, having the number of dimensions in the hundreds is not rare in IS research either. Interestingly, earlier research on LSA also adopted a 300 dimension approach, claiming that “a (roughly) 300-dimensional space usually compares best with human performance” (Kintsch, 2001, p. 176).

Such large dimensionality poses a clear problem for interpretation because it is not the visualization or data presentation as such that scientists look for but rather the meaning that can be projected on that matrix or visual pattern (Landauer et al., 2004). Telling a story that encompasses 300 dimensions is clearly overwhelmingly complex. The task is even more overwhelmingly complex because, empirically, it seems that even as few as 75 to 125 word paragraphs could be enough for LSA to create a semantic space out of (Landauer et al., 2004). That so many dimensions often come from such very small snippets of text makes deriving meaning even more challenging. That challenge comes on top of the complexity introduced by terms’ cross-loading extensively on their factors in the LSA-derived data because of polysemy, polysemousness, heterosemousness, and metonymsness. The comparison to CBSEM is startling. In CBSEM survey research, models seldom have more than 10 latent constructs; that is, factors, and those are composed of only a few items each.

Adding further to the complexity of assigning meaning to LSA results is that LSA is mostly applied as an exploratory method (see Appendix), meaning that the actual pattern in the data is initially mostly unknown. (And, because of polysemy, polysemousness, heterosemousness, and metonymsness there may be more than just one pattern.) In contrast, CBSEM in essence takes a positivist approach to research, and so CBSEM models typically compare the data to an expected pattern. In short, mapping of LSA discovered patterns back to theory might at times be too complex for human cognition. That said, LSA can be run applying a positivist approach, such as in verifying that TAM can be supported based on projecting its items’ keywords onto newspaper corpora (Gefen and Larsen, Forthcoming), making the interpretation of the results straightforward and defendable.

4.5 Visualization

Due to the relative simplicity of LDA outputs, researchers have successfully created visualization packages such as the LDAvis package (Sievert & Shirley, 2014). As in other statistical analyses, visualization is one possible method to reduce cognitive complexity. Unfortunately, LSA’s data structure is quite complex compared to LDA. LSA’s data structure allows one to calculate similarities between terms and documents that may or may not have been part of the original semantic space. Because LSA represents each term and document as a high-dimensional vector, only by reducing these dimensions can visualization become useful. Some approaches to visualizing LSA can be found in Fortuna, Grobelnik, and Mladenič (2005) and Teplovs (2008). Both approaches begin by constructing a similarity matrix from each document to each other document in a corpus and then use that similarity matrix to construct a two-dimensional representation. In Fortuna et al.'s approach, an energy minimization function is applied based on multi-dimensional scaling. In Teplovs's approach, documents are plotted on a graph where edge lengths correspond to the similarity scores between nodes with scores below a threshold being discarded. However, neither these nor other approaches have gained much traction.

5 Conclusion

A tremendous amount of data is stored as text, and that text data is growing in leaps and bounds and making analysis possible in contexts that were not easily accessible in the past. It used to be that acquiring information out of text would require an informed person to actually carefully read the texts, compare them, and then draw subjective conclusions. Making that manual option even less practical with large corpora, methodologically, preferably, that kind of manual text analysis should be done independently by several informed people, not just one, so that their conclusions can be compared. That may still be the best way to understand text, but it is not practical nor is it applicable when tens of thousands of documents need to analyzed within a short period of time.

One alternative to a person actually reading large corpora of text and drawing reasonable information out of it, even if clearly not at the level of a human reader, is to run text analysis with LSA. LSA is by no means perfect, but it does hold the promise of at least partly acquiring some insight from text analysis in a semi-automatic manner from large corpora. To date, researchers have presented several rather convincing applications of LSA, including in the context of IS. For example, Gefen and Larsen (Forthcoming) show that the most cited model in the discipline, TAM, is correct also because of the sematic closeness of the keywords in its scale items.

It is reasonable to expect that as LSA and related methods mature, and their coding becomes easier and their philosophy of science more standardized, that more researchers will turn to LSA and related methods as a means of both augmenting existing research and opening new avenues. The text data are there and readily available, so it would be a pity not to take advantage of such powerful text analysis methods. LSA is a powerful tool, but applying it correctly requires both understanding what it does, and hence its epistemology and strengths and weakness, and how to run it. We hope this guide with its practical code snippets will serve that purpose.

References

Ahmad, S. N., & Laroche, M. (2017). Analyzing electronic word of mouth: A social commerce construct. *International Journal of Information Management*.

Arnulf, J. K., Larsen, K. R., Martinsen, Ø. L., & Bong, C. H. (2014). Predicting survey responses: How and Why semantics shape survey statistics on organizational behaviour. *PLoS ONE, 9*(9),

Bollen, K. A. (1989). *Structural equations with latent variables*. New York: John Wiley and Sons.

Boudreau, M. C., Gefen, D., & Straub, D. W. (2001). Validation in IS research: A state of the art assessment. *MIS Quarterly,* 25(3), 1-16.

Cao, Q., Duan, W., & Gan, Q. (2011). Exploring determinants of voting for the "helpfulness" of online user reviews: A text mining approach. *Decision Support Systems, 50*(2), 511-521.

Chang, J., Boyd-Graber, J., Gerrish, S., Wang, C., & Blei, D. M. (2009). Reading tea leaves: How humans interpret topic models. In *Proceedings of the 23rd Annual Conference on Neural Information Processing Systems* (pp. 288-296).

Chen, L.-C. (2012). Building a term suggestion and ranking system based on a probabilistic analysis model and a semantic analysis graph. *Decision Support Systems*, *53*(1), 257-266.

Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational and Psychological Measurement, 20*, 37-46.

Cosine similarity. (n.d.). In *Wikipedia*. Retrieved from https://en.wikipedia.org/wiki/Cosine_similarity

Coussement, K., Benoit, D. F., & Antioco, M. (2015). A Bayesian approach for incorporating expert opinions into decision support systems: A case study of online consumer-satisfaction detection. *Decision Support Systems, 79*, 24-32.

Coussement, K., & Poel, D. V. d. (2008). Improving customer complaint management by automatic email classification using linguistic style features as predictors. *Decision Support Systems, 44*(4), 870-882.

Davis, F. D. (1989). Perceived usefulness, perceived ease of use and user acceptance of information technology. *MIS Quarterly, 13*(3), 319-340.

D'Haen, J., Poel, D. V. d., Thorleuchter, D., & Benoit, D. F. (2016). Integrating expert knowledge and multilingual Web crawling data in a lead qualification system. *Decision Support Systems, 82*, 69-78.

Debortoli, S., Müller, O., Junglas, I., & vom Brocke, J. (2016). Text mining for information systems researchers: An annotated topic modeling tutorial. *Communications of the Association for Information Systems*, 39, 110-135.

Eliashberg, J., Hui, S. K., & Zhang, Z. J. (2007). From story line to box office: A new approach for green-lighting movie scripts. *Management Science, 53*(6), 881-893.

Evangelopoulos, N., Zhang, X., & Prybutok, V. R. (2012). Latent semantic analysis: Five methodological recommendations. *European Journal of Information Systems, 21*(1), 70-86.

Evangelopoulos, N. (2016). Thematic orientation of the ISJ within a semantic space of IS research. *Information Systems Journal, 26*(1), 39-46.

Fortuna, B., Grobelnik, M., & Mladenič, D. (2005). Visualization of text document corpus. *Informatica, 29*(4), 497-502.

García-Crespo, Á., Colomo-Palacios, R., Gómez-Berbís, J. M., & Ruiz-Mezcua, B. (2010). SEMO: A framework for customer social networks analysis based on semantics. *Journal of Information Technology*, *25*(2), 178-188.

Gefen, D., & Larsen, K. (Forthcoming). Controlling for lexical closeness in survey research: A demonstration on the technology acceptance model. *Journal of the Association for Information Systems*

Gefen, D., Rigdon, E., & Straub, D. W. (2011). An update and extension to SEM guidelines for administrative and social science research. *MIS Quarterly*, *35*(2), III-XIV.

Gefen, D., & Straub, D. W. (1997). Gender differences in perception and adoption of e-mail: An extension to the technology acceptance model. *MIS Quarterly*, *21*(4), 389-400.

Gomez, J. C., Boiy, E., & Moens, M.-F. (2012). Highly discriminative statistical features for email classification. *Knowlege Information Systems*, *31*(1), 23-53.

Hair, J. F., Black, B., Babin, B., & Anderson, R. E. (2005). *Multivariate data analysis* (6th ed.). Upper Saddle River, NJ: Prentice Hall.

Hao, J., Yan, Y., Gong, L., Wang, G., & Lin, J. (2014). Knowledge map-based method for domain knowledge browsing. *Decision Support Systems*, 61, 106-114.

Indulska, M., Hovorka, D. S., & Recker, J. (2012). Quantitative approaches to content analysis: Identifying conceptual drift across publication outlets. *European Journal of Information Systems*, *21*(1), 49-69.

Islam, A., Milios, E., & Keselj, V. (2012). Text similarity using Google tri-grams. In *Proceedings of* the *25th Canadian Conference on Artificial Intelligence* (pp. 312-317).

Kintsch, W. (2001). Predication. *Cognitive Science*, *25*(2), 173-202.

Landauer, T. K., & Dumais, S. T. (1997). A solution to Plato’s problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge. *Psychological Review*, *104*(2), 211-240.

Landauer, T. K., Foltz, P. W., & Laham, D. (1998). An introduction to latent semantic analysis. *Discourse Processes*, *25*(2&3), 259-284.

Landauer, T. K., Laham, D., & Derr, M. (2004). From paragraph to graph: Latent semantic analysis for information visualization. *PNAS, 101*, 5214-5219.

Larsen, K. R., Michie, S., Hekler, E. B., Gibson, B., Spruijt-Metz, D., Ahern, D., Cole-Lewis, H., Ellis, R. J. B., Hesse, B., Moser, R. P., & Yi, J. (2016). Behavior change interventions: The potential of ontologies for advancing science and practice. *Journal of Behavioral Medicine*, *40*(1), 6-22.

Larsen, K. R., & Bong, C. H. (2016). A tool for addressing construct identity in literature reviews and meta-analyses. *MIS Quarterly*, *40* 3), 529-551,

Larsen, K. R., Monarchi, D. E., Hovorka, D. S., & Bailey, C. (2008a). Analyzing unstructured text data: Using latent categorization to identify intellectual communities in information systems. *Decision Support Systems*, *18*(1), 23-43.

Larsen, K. R., Nevo, D., & Rich, E. (2008b). Exploring the semantic validity of questionnaire scales. In *Proceedings of the Hawaii International Conference on System Sciences* (pp. 1-10).

Lemmitisation. (n.d.). In *Wikipedia.* Retrieved from https://en.wikipedia.org/wiki/Lemmatisation

Love, J., & Hirschheim, R. (2016). Reflections on *Information Systems Journal's* thematic composition. *Information Systems Journal, 26*(1), 21-38.

Meire, M., Ballings, M., & Poel D. V. d. (2016). The added value of auxiliary data in sentiment analysis of Facebook posts. *Decision Support Systems, 89*, 98-112.

Mendoza, M., Alegría, E., Maca, M., Cobos, C., & León, E. (2015). Multidimensional analysis model for a document warehouse that includes textual measures. *Decision Support Systems*, *72*, 44-59.

Muthén, L. K., & Muthén, B. O. (2010). *MPlus user’s* guide (6th ed.). Los Angeles, CA.

Sidorova, A., Evangelopoulos, N., Valacich, J. S., & Ramakrishnan, T. (2008). Uncovering the intellectual core of the information systems discipline. *MIS Quarterly, 32*(3), 467-482.

Sidorova, A., & Isik, O. (2010). Business process research: A cross-disciplinary review. *Business Process Management Journal, 16*(4), 566-597.

Sievert, C., & Shirley, K. E. (2014). LDAvis: A method for visualizing and interpreting topics. In Proceedings of the *Workshop on Interactive Language Learning, Visualization, and Interfaces* (pp. 63-70).

Straub, D. W. (1989). Validating instruments in MIS research. *MIS Quarterly, 13*(2), 147-169.

Straub, D. W., Boudreau, M. C., & Gefen, D. (2004). Validation guidelines for IS positivist research. *Communications of the Association for Informations Systems*, *14*, 380-426.

Tannen, D. (1994). *You just don't understand women and men in conversation*. New York, NY: Ballantine Books.

Tannen, D. (1995). The power of talk: Who gets heard and why. *Harvard Business Review, 73*(5), 138-148.

Teplovs, C. (2008). The knowledge space visualizer: A tool for visualizing online discourse. In *Proceedings of the International Conference of the Learning Sciences* (pp. 1-12).

Valle-Lisboa, J. C., & Mizraji, E. (2007). The uncovering of hidden structures by latent semantic analysis. *Information Sciences*, *177*(19), 4122-4147.

Visinescu, L. L., & Evangelopoulos, N. (2014). Orthogonal rotations in latent semantic analysis: An empirical study. *Decision Support Systems*, *62*, 131-143.

Wasserstein, R. L., & Lazar, N. A. (2016). The ASA's statement on p-values: Context, process, and purpose. *The American Statistician*, *70*(2), 129-133.

Wittgenstein, L. (1953). *Philosophical investigations*. New York: Macmillan.

Wei, C.-P., Yang, C. C., & Lin, C.-M. (2008). A latent semantic indexing-based approach to multilingual document clustering. *Decision Support Systems*, *45*(3), 606-620.

Yu, L.-C., & Chien, W.-N. (2013). Independent component analysis for near-synonym choice. *Decision Support Systems*, *55*(1), 146-155.

Appendix

Table A1. List of IS Studies that Applied LSA or Equivalent Method

Study	Questions answered / hypotheses tested
Ahmad, S. N., & Laroche, M. (2017). Analyzing electronic word of mouth: A social commerce construct. *International Journal of Information Management*.	1) To explore the topics of positive and negative reviews 2) to identify which topics are seen as helpful by potential customers.
Cao, Q., Duan, W., & Gan, Q. (2011). Exploring determinants of voting for the “helpfulness” of online user reviews: A text mining approach. *Decision Support Systems, 50*(2), 511-521.	What influences helpfulness of reviews? (Used "SVD factors" as variables.)
Chen, L.-C. (2012). Building a term suggestion and ranking system based on a probabilistic analysis model and a semantic analysis graph. *Decision Support Systems*, *53*(1), 257-266.	Application. Build a system to recommend cheaper adwords,
Coussement, K., Benoit, D. F., & Antioco, M. (2015). A Bayesian approach for incorporating expert opinions into decision support systems: A case study of online consumer-satisfaction detection. *Decision Support Systems, 79*, 24-32.	Application. Review categorization as part of expert system,
Coussement, K., & Poel, D. V. d. (2008). Improving customer complaint management by automatic email classification using linguistic style features as predictors. *Decision Support Systems, 44*(4), 870-882.	Application. Email classification system using LSI and linguistic style from LIWC.
D’Haen, J., Poel, D. V. d., Thorleuchter, D., & Benoit, D. F. (2016). Integrating expert knowledge and multilingual Web crawling data in a lead qualification system. *Decision Support Systems, 82*, 69-78.	Application. Use LSA to identify better prospects and improve sales call efficiency via web data
Eliashberg, J., Hui, S. K., & Zhang, Z. J. (2007). From story line to box office: A new approach for green-lighting movie scripts. *Management Science, 53*(6), 881-893.	Method to improve movie investment decisions.
Evangelopoulos, N. (2016). Thematic orientation of the ISJ within a semantic space of IS research. *Information Systems Journal, 26*(1), 39-46.	Literature review.
Evangelopoulos, N., Zhang, X., & Prybutok, V. R. (2012). Latent semantic analysis: Five methodological recommendations. *European Journal of Information Systems, 21*(1), 70-86.	Methods. Identify best practices for LSA.
García-Crespo, Á., Colomo-Palacios, R., Gómez-Berbís, J. M., & Ruiz-Mezcua, B. (2010). SEMO: A framework for customer social networks analysis based on semantics. *Journal of Information Technology*, *25*(2), 178-188.	
Hao, J., Yan, Y., Gong, L., Wang, G., & Lin, J. (2014). Knowledge map-based method for domain knowledge browsing. *Decision Support Systems*, 61, 106-114.	
Indulska, M., Hovorka, D. S., & Recker, J. (2012). Quantitative approaches to content analysis: Identifying conceptual drift across publication outlets. *European Journal of Information Systems*, *21*(1), 49-69.	
Larsen, K. R., & Bong, C. H. (2016). A tool for addressing construct identity in literature reviews and meta-analyses. *MIS Quarterly*, *40* 3), 529-551,	Methods to identify similar theoretical constructs.
Larsen, K. R., Monarchi, D. E., Hovorka, D. S., & Bailey, C. (2008). Analyzing unstructured text data: Using latent categorization to identify intellectual communities in information systems. *Decision Support Systems*, *18*(1), 23-43.	
Lin et al. (2017). Social commerce research: Definition, research themes and the trends. *International Journal of Information Management*, 37(3), 190-201.	Literature review. What are the themes of “social commerce” research?
Love, J., & Hirschheim, R. (2016). Reflections on *Information Systems Journal's* thematic composition. *Information Systems Journal, 26*(1), 21-38.	

Table A1. List of IS Studies that Applied LSA or Equivalent Method

Meire, M., Ballings, M., & Poel D. V. d. (2016). The added value of auxiliary data in sentiment analysis of Facebook posts. *Decision Support Systems, 89*, 98-112.	What information can help identify sentiment in FB posts, what predictors are most important, and what is relationship between predictors and sentiment?
Mendoza, M., Alegría, E., Maca, M., Cobos, C., & León, E. (2015). Multidimensional analysis model for a document warehouse that includes textual measures. *Decision Support Systems*, *72*, 44-59.	Application. OLAP data warehouse for storing/sorting text.
Sidorova, A., & Isik, O. (2010). Business process research: A cross-disciplinary review. *Business Process Management Journal, 16*(4), 566-597.	
Sidorova, A., Evangelopoulos, N., Valacich, J. S., & Ramakrishnan, T. (2008). Uncovering the intellectual core of the information systems discipline. *MIS Quarterly, 32*(3), 467-482.	
Visinescu, L. L., & Evangelopoulos, N. (2014). Orthogonal rotations in latent semantic analysis: An empirical study. *Decision Support Systems*, *62*, 131-143.	
Wei, C.-P., Yang, C. C., & Lin, C.-M. (2008). A latent semantic indexing-based approach to multilingual document clustering. *Decision Support Systems*, *45*(3), 606-620.	Application. LSI-based technique for multilingual document clustering.
Yu, L.-C., & Chien, W.-N. (2013). Independent component analysis for near-synonym choice. *Decision Support Systems*, *55*(1), 146-155.	
Gefen, D., & Larsen, K. (Forthcoming). Controlling for lexical closeness in survey research: A demonstration on the technology acceptance model. *Journal of the Association for Information Systems*	Shows that the paper that initially introduced the technology acceptance model (TAM), the most cited paper in IS, is correct also because of the way the keywords in the survey relate to each other in English.

Table A2. List of IS Studies that Applied LSA or Equivalent Method

Authors	Number of documents	Number of terms	Number of dimensions	Document types	Stemming	Stop word lists	Was the semantic space rotated?
Ahmad & Laroche (2017)	2 sets: 147, 258.	1046, 1190	10	Amazon online reviews	Yes	Yes	No
Cao et al. (2011)	3460	3457		CNET Download.com online reviews	Yes	Yes	No
Chen (2012)							
Coussement et al. (2015)	1014			Consumer reviews	Yes	Yes, and rare terms	No
Coussement & Poel (2008)	5196		By 10s to 200	Customer service emails	Dictionary-based	Yes	No
D'Haen et al. (2016)	unspecified		1-100	Crawled corporate web pages	Yes	Yes, and rare terms	No
Eliashberg et al. (2007)	200	100	2	Movie spoiler reviews	Yes	Dropped words apart from "important words"	No
Evangelopoulos (2016)	4827	1225	5	IS Abstracts	Yes	Yes, also low-meaning words	Yes
Evangelopoulos et al. (2012)	498, 498, 498, 22	1873, 1873, 230, 261	100, 30, 12, 7 & 3	*EJIS* paper abstracts	Yes	563-word list, 563, 230, unspecified	No, yes, yes, yes
García-Crespo et al. (2010)							
Hao et al. (2014)	844	153 - autogen list, then expert-edited		844 documents about CNC technology		Go-list	No
Indulska et al. (2012)	8544			Abstracts from IS, Management, Accounting			
Larsen & Bong (2016)			300	Paragraphs from IS articles: MISQ/ISR	Yes		No
Larsen et al. (2008)	14510			IS Abstracts	Yes	Yes	Yes
Lin et al. (2016)	418 academic abstracts	1741	3	Article abstracts	Yes	Yes	
Love & Hirschheim (2016)	4745	7619	100	IS Abstracts + keywords + titles, Basket of 8 1991-2013	Yes	Yes	Yes
Meire et al. (2016)	17,697		100	Facebook posts	Yes	Yes	
Mendoza et al. (2015)	200, 400, 600			Scientific articles	Yes	Yes	
Sidorova & Isik (2010)	2701	1176	10, 20, 30	Academic abstracts on business processes	Yes	Yes, also low-meaning words	Yes
Sidorova et al. (2008)	1615	1318	5 and 100	IS Abstracts	Yes	Yes, also low-meaning words	Yes
Visinescu & Evangelopoulos (2014)	902; 1481; 1110			3, 5, 7, 9	SMS messages	Yes	
Wei et al. (2008)	2949		5 to 200 by 5	English and Chinese language abstracts from MIS theses/dissertations	Implied- only nouns and noun phrases		No

Table A2. List of IS Studies that Applied LSA or Equivalent Method

Yu & Chien (2013)		20,000 5-grams	100, 300, 500, 700, 1000, 1500, 2000, 2500, 3000	sentences from news articles			
Gefen & Larsen (Forthcoming)	JAIS	500	500	Two newspaper corpora	Yes	No	No

About the Authors

David Gefen is a Professor of MIS at Drexel University and Provost Distinguished Research Professor, Philadelphia, US, where he teaches IT Outsourcing, Strategic Management of IT, Database Analysis and Design, Data Mining in SAS, and research methodology. He received his PhD in CIS from Georgia State University and a Master of Sciences in MIS from Tel Aviv University. His research focuses on trust and culture as they apply to the psychological and rational processes involved in ERP, CMC, and ecommerce implementation management, and to outsourcing. His wide interests in IT adoption stem from his twelve years of experience in developing and managing large information systems. His research findings have been published in *MISQ, ISR, IEEE TEM, JMIS, JSIS, Omega*, J*ournal of the Association for Information Systems*, *Communications of the Association for Information Systems*, and elsewhere. He is an author of a textbook on VB.NET programming and a book on the art of IS outsourcing. He was a Senior Editor *MISQ* and is currently on the Editorial Board of *JMIS*.

James E. Ellicott is a PhD Candidate in Information Management at the Leeds School of Businesses, University of Colorado at Boulder. They received a Master of Science in Information Science and Technology at Claremont Graduate University in California. Their research focuses on using natural language processing techniques in order to increase the accessibility of texts to both experts in other disciplines through work like the Inter-nomological Network (INN) and non-experts through text simplification.

Jorge E. Fresneda is a PhD Candidate in the Marketing Department at the LeBow College of Business, Drexel University, Philadelphia, PA, US. He received a Master of Sciences in Applied Statistics from UNED and a Master of Arts in Marketing and Sales Management from EAE Business School. His research focuses on analyzing the influence of textual information in online consumers on their purchase decision processes. He teaches courses on marketing research and consumer analytics. Jorge has nine years of industry marketing experience as a product manager at a large Business-to-Business corporation.

Jacob Miller is a Ph.D. candidate with a concentration in Organization and Strategy at the LeBow College of Business, Drexel University, Philadelphia, PA, US. He received his MBA at York College of Pennsylvania. His research interests are focused in information technology and cognition through the use of large scale text analytics. He teaches courses on entrepreneurship and corporate strategy. Professionally, he developed and maintained information management systems in the software and information industries.

Kai R. Larsen is an associate professor of Information Systems at the Leeds School of Business, University of Colorado at Boulder. He holds a courtesy appointment in the Information Science Department, College of Media, Communication and Information. As director of the Human Behavior Project, he is conducting research to create a transdisciplinary “backbone” for theoretical research. He applies text mining technologies to create an integrating framework for predictors of human behavior. The research has implications for our understanding of human behaviors, including technology utilization, investor decisions, and cancer-prevention behaviors.