Semantic Vectors: A Scalable Open Source Package and Online Technology Management Application

Dominic Widdows, Kathleen Ferraro

MAYA Design, University of Pittsburgh
dwiddows@gmail.com, kaf1@pitt.edu

Abstract

This paper describes the open source SemanticVectors package that efficiently creates semantic vectors for words and documents from a corpus of free text articles. We believe that this package can play an important role in furthering research in distributional semantics, and (perhaps more importantly) can help to significantly reduce the current gap that exists between good research results and valuable applications in production software. Two clear principles that have guided the creation of the package so far include ease-of-use and scalability. The basic package installs and runs easily on any Java-enabled platform, and depends only on Apache Lucene. Dimension reduction is performed using Random Projection, which enables the system to scale much more effectively than other algorithms used for the same purpose. This paper also describes a trial application in the Technology Management domain, which highlights some user-centred design challenges that we believe are also key to successful deployment of this technology.

1. Introduction

This paper describes the open source SemanticVectors software package, which can be freely downloaded from `http://semanticvectors.googlecode.com`.

The software can be used to easily create semantic vector models from a corpus of free text, and to search such models using a variety of mathematical operations including projections and algebraic product operations.

It is hoped that the availability of this software will be of benefit both to academic and commercial users, due to its simplicity, ease of use, and scalability. Instead of spending considerable time on the basic text processing and search operations, researchers and developers will be able to focus their efforts on new experiments that investigate the relationship between mathematical properties of the model and linguistic properties of the source texts, and on integrating semantic matching and search features into larger systems that support users with increasingly complex information needs.

The core idea behind semantic vector models is that words and concepts are represented by points in a mathematical space, and this representation is learned from text in such a way that concepts with similar or related meanings are near to one another in that space. This introduces a range of possible applications: the most immediate perhaps is the end-user 'semantic search engine'; semantic vector models can also be used in resource-building applications such as ontology learning and lexical acquisition, as part of data-gathering for decision-support systems, detailed research, etc. The SemanticVectors package itself was developed for a demonstration of such an application, to help Technology Management professionals at the University of Pittsburgh.

To systematically describe the package, this paper proceeds as follows. In Section 2., we review some of the basics of Semantic Vector models. Section 3. gives a summary of Random Projection, the dimension reduction technique used by the Semantic Vectors package, chosen particularly for its scalability and computational tractability. Section 4. describes the Semantic Vectors package itself, including a detailed description of its design, implementation, current and intended features. Section 5. describes the University of Pittsburgh's Technology Management Application, for which SemanticVectors was initially built, and discusses other potential applications of this technology.

2. Semantic Vector or WORDSPACE Models

Semantic vector models have received considerable attention from researchers in natural language processing over the past 15 years, though their invention can be traced at least to Salton's introduction of the Vector Space Model for information retrieval [Salton, 1971, Salton and McGill, 1983].

Semantic vector models include a family of related models for representing concepts with vectors in a high dimensional vector space, such as Latent Semantic Analysis [Landauer and Dumais, 1997], Hyperspace Analogue to Language [Lund and Burgess, 1996], and WORDSPACE [Schütze, 1998, Widdows, 2004, Sahlgren, 2006].

The main attractions of semantic vector models include:

- They can be built using entirely unsupervised distributional analysis of free text.
- While they involve some nontrivial mathematical machinery, they make very few language-specific assumptions (e.g., it is possible to build a semantic vector model provided only that you have reliably tokenized text).
- Similar techniques have been used in other areas, e.g., for image processing Bingham and Mannila [2001].
- The ease with which very simple distributed memory units can collaboratively learn and represent semantic vector models has been noted for its potential cognitive significance [Kanerva, 1988]. This has led to some overlap in interests between semantic vector researchers in computational linguistics, and compositional connectionist researchers in cognitive science.

- Being strongly distributional and associative in character, they have complementary strengths to some of the more traditional formalist and symbolic semantic techniques such as those based on propositional logic and lambda calculus (for more discussion of this angle, see Widdows [2008]).

Over several years, the strengths of these models have been examined and evaluated in the performance of several tasks of importance to natural language processing. Such applications of semantic vector models to date include:

- Information retrieval [Deerwester et al., 1990]. This was the original motivation for applying dimension reduction to a term-document matrix, in the hope of creating a more semantically aware search engine (e.g., a search engine that can locate documents based on synonyms and related terms as well as matching keywords).
- Lexical and ontology acquisition [Hearst and Schütze, 1993, Widdows, 2003b]. The core principle here is that knowledge of a few seed words and their relationships can help to infer analogous relationships for other similar words that are nearby in the semantic vector space.
- Word sense discrimination and disambiguation [Schütze, 1997, Schütze, 1998]. The core principle here is that the weighted sum of vectors for words found in a particular region of text (called context vectors) can be clustered, and the centroids of these clusters can be treated as word-senses: occurrences of an ambiguous word can then be mapped to one of these word-senses, with a confidence or probability derived from the similarity between the context vector for this occurrence and the nearest centroids.
- Document segmentation [Brants et al., 2002]. Given that we can compute context vectors for regions of text, one can detect document boundaries when context vectors leap from one part of the space to a completely different part of the space.
- Representing online communities and knowledge [McArthur and Bruza, 2003]. As the amount of information shared in online communities (Usenet groups, mailing list archives, web forums, etc.) grows, it is increasingly important for new users to be able to evaluate and distinguish important information. One way of doing this is to model the utterances observed in online communities using semantic vector representations.

While framing this discussion in terms of vector models suggests a strongly geometric account, it is worth noting that probabilistic interpretations have been applied to latent semantic models [Papadimitriou et al., 2000], and there are several related models built from distributional principles that are based on probabilistic rather than geometric insights (e.g., [Hofmann, 1999, Blei et al., 2003]). A thorough comparison of probabilistic and geometric points of view is beyond the scope of this paper: van Rijsbergen [2004] points out that quantum mechanics is already a clearly extant framework that combines both probabilistic and geometric insights, coordinates of vectors being related to probability amplitudes. It may therefore be a mistake to think of probabilistic and geometric methods as competing alternatives: they should perhaps rather be thought of as two compatible ways of looking at related conceptual structures.

In spite of the successes and benefits of semantic vector models, they have been underadopted outside of the research community. We believe that this is for at least three reasons:

1. Implementations of semantic vectors that satisfy industry needs for reliable and well integrated software have not been available.
2. Most algorithms for creating semantic vectors involve computationally expensive matrix factorization, which introduces scalability bounds.
3. There has been insufficient effort in exploring compelling deployments in end user applications.

The SemanticVectors package aims to bring about a new phase in the large scale deployment of semantic vector applications, by addressing the first two issues directly. We also hope that this will make it much easier for many researchers and developers to experiment in addressing the third issue.

3. Reducing Dimensions using Random Projection

Reducing dimensions is one of the key features that is used to uncover the underlying features — the so-called 'latent semantic dimensions' of a distribution. Random Projection has enjoyed increased attention in the past few years, Bingham and Mannila [2001], though the technique is older and was introduced to us by Kanerva [1988]. The main insight of Random Projection is that high dimensional vectors chosen at random are "nearly orthogonal", in a way that can be formally characterized. Thus it achieves a result that is for many purposes comparable to orthogonalization methods such as Singular Value Decomposition [see Sahlgren, 2005], but spends none of the computational resources.

The basic procedure for creating a random basic document vector is extremely terse and easy to implement. Consider two vectors which contain mainly zeros (i.e., they are sparse), and whose nonzero entries are comprised of an equal number of 1 and -1 entries. For example:

$$[0, 0, 0, 1, 0, -1, 0, 0, 0, -1, 0, 0, \ldots, 0, 0, 1, 0, 0]$$

and

$$[0, -1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, \ldots, 0, 0, -1, 0].$$

It is easy to see that the expected value of the scalar product of two such randomly generated vectors is zero. When we multiply each coordinate in one vector by the corresponding coordinate in the other vector, in most cases at least one of the multiplicands is zero. In the rare cases that both multiplicands are non-zero, half the time we expect the signs

of the multiplicands to be the same, contributing $+1$ to the scalar product, and the other half of the time we expect the signs of the multiplicands to differ, contributing -1. So on average, these contributions will cancel out. To judge orthogonality we actually calculate the angle between the vectors, so we normalize by the lengths of the vectors (i.e., cosine similarity rather than the raw scalar product), leaving any non-zero results even closer to zero.

In practice, the SemanticVectors software uses a sparse representation for these vectors (size proportional to the number of nonzero entries rather than the number of dimensions overall), which has enabled the package to process corpora consisting of far more documents. The authors would like to thank Prof. Trevor Cohen of Arizona State University for making this contribution.

There are several other ways of reducing dimensions, including the following:

- Singular value decomposition (SVD) [Trefethen and Bau, 1997]. Taking a matrix A, an eigenvalue decomposition of the symmetric matrix AA^T yields a factorization $A = \hat{U}\Sigma V$, where U is an orthonormal matrix and Σ is a diagonal matrix with non-increasing leading values, called the *singular values*. Taking the first m of these only yields an approximation to the original matrix A, the projection onto the basis given by the first m columns of V. Singular value decomposition is the algorithm used in latent semantic indexing [Deerwester et al., 1990] or latent semantic analysis Landauer and Dumais [1997]. For an $m \times n$ matrix A, the full singular value decomposition usually takes time $O(mn^2 + m^2n + n^3)$ to compute Brand [2002]. This can be reduced in certain circumstances (e.g., if we declare in advance that only the first k eigenvalues should be computed), but certainly not below quadratic complexity.

- Probabilistic Latent Semantic Analysis (PLSA) [Hofmann, 1999]. PLSA uses a version of the Expectation Maximization algorithm (see e.g., [Manning and Schütze, 1999, Ch 14]) and reports some semantic improvements over geometric LSA. It is hard to evaluate the computational complexity of the family of EM algorithms generally, though they are often computationally demanding.

- Latent Dirichlet Allocation (LDA) [Blei et al., 2003]. Building on Hofmann's work, LDA provides a probabilistic generative model that accounts for documents as being probabilistic mixtures of underlying topics, these being the latent variables of the model. Blei et al. [2003] also use an EM algorithm for estimating the k topic parameters, and that estimating the model for each document is of order N^2k, where N is the number of words in the document.

Each of these approaches has been evaluated against at least one other, sometimes against several. Griffiths et al. [2007] also compare LDA-driven Topic Models against LSA, and Sahlgren [2005] demonstrates that random projection performs comparably well with LSA. However, we believe these results give at best an incomplete picture of the whole state-of-the-art, for at least three reasons:

1. It would be hard to organize the software necessary to compare all of the above algorithms on a common task.

2. Such a study might not yield positive results about new methods. which is often encouraged.

3. It would be extremely hard to choose any one task or even small collection of tasks in such a way that the choice of task did not introduce a major bias into the nature of the analysis. It is moreover probable that different methods are appropriate to different tasks.

Given this difficulty of comparing algorithms in terms of precision and recall, even on some fixed task using a static dataset, the decision to build the SemanticVectors package using the random projection technique was based on more physical engineering considerations.

- It is easily the *simplest* of the algorithms known to us. This means that the code for performing random projection was easy to write, easy to test, and easy to bundle with the main SemanticVectors package. It introduces no great complexity to the core package, and no external dependencies on other software components.

- However well implemented, matrix factorizations such as Singular Value Decomposition or probabilistic clustering methods that are quadratic or higher in the size of the data do not cope with the huge datasets we are seeing nowadays. Given the explosion in corpus size over the past decade, Moore's law will not "take care of it" for quadratic or higher algorithms — Moore's law instead ensures that *only* linear or better algorithms can keep pace with the growth in data.

- Because Random Projection uses nearly orthogonal basic vectors that are generated independently, parts of the model can be created independently of one another. This enables distributed model creation using techniques such as MapReduce [Dean and Ghemawat, 2004], and incremental addition of new terms and documents to models without rebuilding them from scratch. This is crucial for any large-scale industrial application.

- It would be easy to make improvements by extra rounds of training: this possibility will be described in the next section.

Moreover, in case Random Projection turns out to be a *poor* choice of algorithm, the SemanticVectors software has been designed so that several parts of the codebase are independent of this decision. For vectors created using a completely different method, it is still perfectly possible to use SemanticVectors as a search and exploration engine to analyse these vectors (as was done for some of the results presented in Widdows [2008]).

4. The Semantic Vectors Package Itself

Having described some of the principles behind semantic vector models and dimensionality reduction, this section (perhaps the most useful in this paper) gives practical details into how the software package is actually implemented. Our goal in this section, as with the package as a whole, is to make the package as accessible, stable, and useful as possible so as to reduce the bar of entry into this space.

4.1. Summary

To begin with, we reiterate that the SemanticVectors package is freely available under a no-cost, BSD license, and can be downloaded from `http://semanticvectors.googlecode.com/`. As well as providing free downloads of source and object code, this site is a portal to:

1. Wiki documentation pages, which currently include installation instructions, instructions for bilingual models, some discussion of and comparison of other software and algorithms, release notes for particular versions, etc.
2. Online discussion group and forum, used for a variety of topics including feature requests, suggested uses and experiments, and wider discussion of related research and engineering issues.
3. Issues and bug tracking features.
4. Detailed Javadoc pages. In general, the standard of Javadoc comments for the package has been quite good, so there is a wealth of information available on how to use the package.

4.2. Ease-of-use

Though from a research point of view, it may be tempting to think of algorithmic scalability and semantic evaluation as the most important aspects encouraging general community uptake of the software. However, we believe that this overlooks an important factor in producing successful open-source software: many projects do not see the success they deserve because they are *hard to use*. The fundamental requirement that SemanticVectors had to be as *easy* as possible was partly influenced by the first author's experiences maintaining the Stanford Infomap NLP package (available at `http://infomap-nlp.sourceforge.net`).

Over some four years of use by several members of the research community, we received comments from one user saying that the matrix decomposition algorithm (Singular Value Decomposition) was not computationally tractable for the amount of data they wished to process. Over the same period, we received several dozen messages saying that the software was missing dependencies that it could not find, that it claimed to have all the necessary dependencies but it would not compile, that it compiled but would not run, or that it compiled, ran, claimed to have finished, and produced no data. In spite of our best efforts, integration with matrix algorithm packages and database storage systems on different platforms was the main day-to-day problem with Infomap, and deterred or disabled many potential users. Most of these problems fell into two main categories (which are obviously correlated):

1. Differences between platforms.
2. Dependence on too many other components.

To minimize platform-dependent issues, the Semantic Vectors package is implemented entirely in Java. This means that as soon as a user has a basic Java Development Kit (JDK) installed, the only other things to do are to configure some path variables, collect some appropriate data, and run the software. (To make sure that initial data gathering is not an initial deterrent, the King James Bible is included on the main download page as a test dataset.)

To minimize dependence on external components, the core software depends only on core Java classes that ship with the JDK, and APIs that are part of Apache Lucene (`http://lucene.apache.org/`), a powerful and widely used piece of open source software. We use Lucene's tokenization and indexing packages to create a basic term document matrix. The Semantic Vectors package then uses the Lucene API to create a WORDSPACE model from this term document matrix, using Random Projection to perform on-the-fly dimensionality reduction.

In practice, this means that all a user has to do is download the software, make sure that the system `$PATH` variable includes the directories containing the `java` and `javac` binaries, and that the Java-specific `$CLASSPATH` variable contains the SemanticVectors classes and (for building indexes) the Apache Lucene classes. To compile from source, Apache Ant is also needed. Users who do not wish to compile the code for themselves can use the binary `jar` distribution, which works on all major platforms that run Java.

The whole installation and configuration process is of course described more fully in the online documentation. So far the design decisions above appear to have been a great success: over 500 copies of the software have been downloaded, and so far we have not received a single question or bug report regarding installation.

The package is extremely lightweight: the `.tar.gz` compression of the current source distribution (version 1.6 at the time of writing) ships at a somewhat astonishing 23KB!

4.3. Software Architecture

SemanticVectors has two main functions: i. building WORDSPACE models (indexing), and ii. searching through the vectors in such models (querying). Because these processes have such different computational requirements, we have tried to limit their common dependencies mainly to core mathematical functions. Classes responsible for building and searching models both use a common vector store file format: available options are at present: i. a simple pipe-delimited text format, and ii. an optimized Lucene format.

Building Models

The main utility is `BuildIndex`, which creates an object that reads some external data (e.g., a `TermVectorsFromLucene` object, which reads Lucene's term-document matrix and creates reduced

semantic vectors). Once initialized, the object implements the `VectorStore` interface, and `getAllVectors()` is called to serialize the learned vectors to disk, using one of the implemented file formats.

To build a model using Lucene, it is first necessary to build a Lucene index from suitably arranged files on some part of a filesystem. The Lucene documentation gives easy instructions on how to do this in simple cases, and some utils ship with the SemanticVectors package for building bilingual Lucene indexes for input. For examples tested so far, the SemanticVectors indexing process is at least two orders of magnitude faster than Lucene's (it is doing much less of the work!), so while incremental indexing is a long-term requirement for SemanticVectors, effort has not been put to this yet. The indexing process is often memory intensive, though large collections of several hundred thousand documents can be handled quickly on a standard computer if plenty of Java heap space is allocated.

As well as building term vectors, there are classes for subsequently building document vectors as a weighted sum of the term vectors of their constituent terms, as has been done since early vector model information retrieval systems [see e.g., Salton and McGill, 1983]. These derived document vectors should not be confused with the basic random vectors for each document that are used to create the term vectors in the first place.

Searching Models

Vectors are read from a file by an object that implements the `VectorStore` interface. This object enables the caller to get a vector with a particular name (for building queries), and to iterate through all the vectors (for searching). We have as yet implemented no way to perform a search without iterating through all vectors in the store: how best to do this for high-dimensional indexes is a fascinating and persistent challenge in computer science [Chávez et al., 2001].

Scanning and ranking of these vectors is performed by `VectorSearcher`, an abstract class whose implementations must also implement a `getScore()` method which returns a score for each test vector. The highest scores are recorded and the ranked list returned by the `getNearestNeighbors` method which is shared by all implementations of `VectorSearcher`. Training the searcher (e.g., looking up simple query vectors, or creating more complex query expressions such as entangled tensor products) is done during the initialization of each `VectorSearcher`.

The benefit of this design is that it is very easy for new scoring functions to be easily implemented. Whatever input data these scoring functions require, if they can be reduced to a mapping from the vector space to the real numbers, then the rest of the code in the `VectorSearcher` class enables the scanning and ranking to be done automatically.

A standard procedure is to build a query using one `VectorStore` and run the search using another: provided they use a common basis, meaningful results will be obtained. The same principle is used both for using term vectors to search for documents, and for bilingual searches where the query terms and the results are in different languages.

Common Mathematical Functions

There are several simple mathematical functions coded into the `VectorUtils` class. These include scalar products and cosine similarity, normalization, tensor operations (inner and outer product, sum, normalization), convolution products, and orthogonalization routines for vector negation and disjunction. Background on these mathematical operations and their linguistic uses can be found in [Plate, 2003, Widdows, 2003a, 2004, 2008]

Vector File Formats

SemanticVectors currently supports two different file formats for vector storage and I/O. The default is an optimized binary format that uses Lucene's very fast serialization and deserialization operations for floating point numbers. There is also a flat text file format, which represents one vector per line using the simple format

$$\texttt{name}|\texttt{a}_1|\texttt{a}_2|\ldots|\texttt{a}_n$$

where `name` is a string identifier (e.g., the word for a word vector or the path and filename for a document vector) and the following numbers are the coordinates of the corresponding vector.

As one would expect, the text format is larger (up to 3 times) and slower (up to 10 times), but it is valuable for interoperability with other software, e.g., for exporting a model to be manipulated by another analysis tool such as Matlab. There is a translater utility that easily enables users to map between the two formats.

For either format, the number of dimensions is encoded as metadata at the beginning of the file by newer versions of the software: indexes created by older versions of the software lack this metadata, in which case newer versions estimate the number of dimensions by parsing the first vector.

Vectors and Numbers

It should be noted that the package presently assumes that vector spaces are over the real numbers, and that real numbers are approximated by (4-byte) floats. (An exception is the basic random index vectors, which are represented in a sparse short integer (2-byte) format.) Simple floats are used in preference to 8-byte double-precision numbers: at first glance, it may seem strange in this day and age to save resources by accepting less precise arithmetic representations; but it appears empirically to be more effective to use these resources to enable more independent dimensions to be represented.

Quite where to balance resources between real number precision and more expressive mathematical structures (e.g., more dimensions) is an interesting question. A related opportunity in 2-dimensions is whether to use complex numbers (pairs of interrelated real numbers) or more precise real numbers [see van Rijsbergen, 2004, p. 25]. The mathematical properties we used to motivate the use of basic random vectors whose non-zero values are taken from the set $\{-1, 1\}$ would also apply if these values are taken randomly from the group of unit complex numbers. An adaptation of SemanticVectors to use vectors in complex Hilbert spaces would be a logical next step, and proposes interesting experiments.

Documentation and Testing

The package's documentation is reasonably thorough and well-organized: all major classes and interfaces have API documentation, and a range of use cases and example scripts are described in the project Wiki documentation. The Javadoc-generated `html` files are checked in to the main SVN repository, and so are automatically available online. These files are not shipped with source distributions, but fresh documentation is generated locally as a standard part of compilation.

The package's tests are in much poorer shape. There is a framework for building unit tests in a parallel part of the source tree, and running these automatically using JUnit (`junit.org`). Very few have been added so far, and as features are added, this is becoming an increasingly pressing issue.

Maintenance and Support

Most of the development time for SemanticVectors has so far been devoted to the development and documentation of new functionality. However, there have been several support requests, which have been posted to and answered through the Google Group associated with the project. Often these have led to the introduction of a new feature or extra flexibility in the software.

The first author's time commitment to the project has been supported by Google as a "20% project", through which Google encourages its engineers to devote time to new research and development beyond their main projects.

Support and commitment have gradually been emerging as some users of the software become developers, which is particularly important in the development of community-backed open source software.

This concludes our presentation of the main features of the software. The reader is reminded that the most up-to-date versions of documentation should be found online, and if these come to differ from anything stated in this paper, the new versions take preference.

5. Technology Matching at the University of Pittsburgh

The use case for which the SemanticVectors package was specifically created was for the University of Pittsburgh's Office of Technology Management. The Technology Management domain illustrates some key aspects of information management today.

Matching technologies to prospective licensees is crucial to translating scientific knowledge generated through university research into useful products. Searching and matching capabilities incorporating concept mapping tools promise to significantly increase and enhance our ability to match available technologies to prospective licensees' needs, as well as to perform targeted marketing of technologies.

The Technology Matching Project investigates some specific ways in which semantic vector technologies may be used to help answer questions related to technology transfer such as:

- Given a technology concept, what other technology concepts are related?
- What patents have been applied for or issued for a particular technology?
- What other patents are relevant?
- What groups of technologies are available that satisfy similar or complementary needs?
- What other materials — refereed articles, product descriptions, press releases — can be used to learn about a technology?
- How can we find out when problems faced in one domain may be solved by techniques developed in another area?
- How can we tell which companies may be interested in licensing a particular technology or group of technologies?

For many of these question types, experts in the field expressed frustration with search technologies focussed on pure keyword-matching. It is rare for a technology disclosure to contain exactly the same keywords as a description of a product on a company's website, but there are many more cases where a relatively modest amount of induction and inference can lead a user to join the dots between technologies and companies and at least make a useful hypothesis. The missing ingredient using keywords only is still some representation of the semantic connection between different terms, and this led us to create a demonstration online technology matching application, for which the Semantic Vectors package forms the main "semantic glue."

The initial demonstration is aimed at a user who has access to technology disclosures and wants to match these to potential licensees, as shown in Figure 1. Information about 834 companies was collected from their websites by taking names from the US Federal Drug Administration database, matching these to websites, and gathering documents from these websites. (This was a reasonably accurate and low-cost way of gathering enough data to create a useful system, though it raises important questions of its own and improving this "Web-as-corpus" part of the system may become a whole new project.) These documents were indexed by Lucene, and SemanticVector indexes created from the Lucene indexes. The user can create a query by typing as usual, or (more importantly) can select one of the existing disclosures and use that as a query to find related documents. The user can then eyeball those documents to see if they are good matches, and in exceptionally good situations, this could lead the user to contact the company in question.

The demonstration has so far been received with enthusiasm by technology managers and analysts, though as a thought-provoking proof-of-concept, rather than a complete application. Some excellent matches have been suggested that would not have been found with keywords alone: however, data quality is a serious issue. The coverage and accuracy of the web crawls could be curated

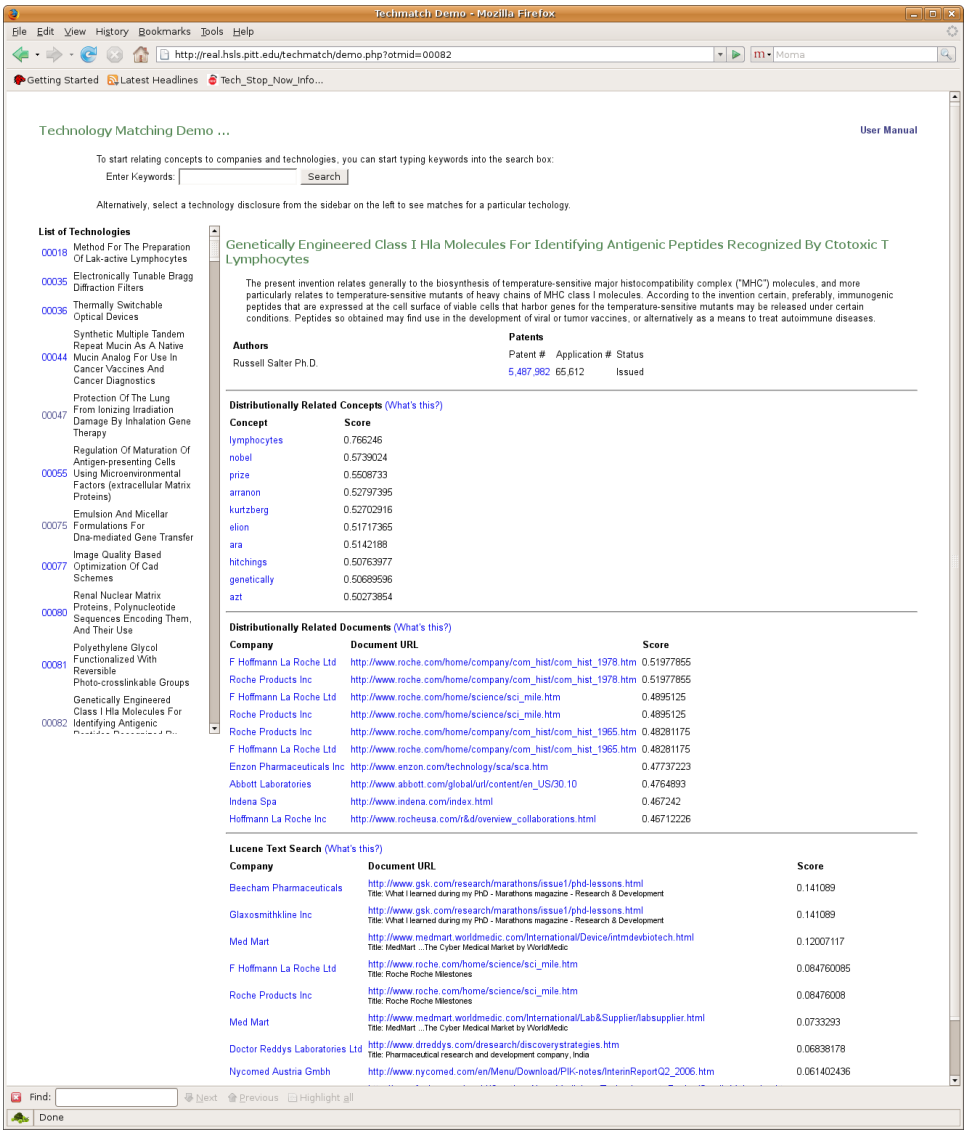

Figure 1: Screenshot of the Technology Matching Demo showing a Technology Disclosure, Related Concepts, and Documents found using Semantic Vectors and Apache Lucene.

more effectively, and in many cases the difference in textual genre between company websites and university technology disclosures makes matching difficult (even for a human judge). Accurate results are not yet produced reliably enough for the demonstration to warrant its own niche as a separate application in an already complex environment of Google searches, MedTrack, databases of biological disclosures, inventor contacts, CRM systems, etc. Integration as a component in one or more of these tools is a long term prospect, within the more general problem of designing better discovery-rich applications for knowledge professionals, without increasing complexity ever further with yet more places for yet more information. Potential ways to explore data offered by semantic vector models include clustering and visualization [Widdows, 2004, Ch 6], and there is some excitement about using these technologies to enable the linking and bundling of complementary technologies.

6. Conclusion

It is by now clear that a variety of distributional models can be used to give interesting information about ideas that are related in meaning. At the end of Section 4., we raised three challenges in moving these techniques beyond the research arena: providing stable reusable software; deploying algorithms that scale with today's expectations; and finding compelling ways to use these techniques to help information professionals.

We have described how the SemanticVectors project has approached the first two problems we raised. We have created and publicly released reliable, sustainable software, which has attracted several core users and some new active contributors. The current system performs and scales well compared with alternatives, and we believe that it has laid the right groundwork to be able to scale much further.

This leaves us to discuss the third challenge: finding convincing user applications. One possible use case that we have been investigating is the use of semantic vectors in information exploration in technology management. Initial

responses have been enthusiastic, though many questions about information quality have been raised.

7. Acknowledgments

The initial work on the SemanticVectors package was carried out in a collaboration between the University of Pittsburgh and MAYA Design, funded by a Keystone Innovation grant from the State of Pennsylvania.
The authors would also like to thank members of the research community, particularly Trevor Cohen, for their contributions to the project's codebase, and Google, Inc. for hosting the SemanticVectors package, and for supporting the first author's ongoing commitment to this project.
This paper differs slightly from the official version that appeared in the LREC proceedings. This is because of comments and corrections posted by members of the SemanticVectors Google Group, who the authors would also like to thank.

References

Ella Bingham and Heikki Mannila. Random projection in dimensionality reduction: applications to image and text data. In *KDD '01: Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 245–250, New York, NY, USA, 2001. ACM. ISBN 1-58113-391-X. doi: http://doi.acm.org/10.1145/502512.502546.

David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

Matthew Brand. Incremental singular value decomposition of uncertain data with missing values. In *Proceedings of the European Conference on Computer Vision (ECCV)*, May 2002.

Thorsten Brants, Francine Chen, and Ioannis Tsochantaridis. Topic-based document segmentation with probabilistic latent semantic analysis. In *Conference on Information and Knowledge Management (CIKM)*, pages 211–218, 2002.

Edgar Chávez, Gonzalo Navarro, Ricardo Baeza-Yates, and José Luis Marroquín. Searching in metric spaces. *ACM Comput. Surv.*, 33(3):273–321, 2001. ISSN 0360-0300. doi: http://doi.acm.org/10.1145/502807.502808.

Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI'04: Sixth Symposium on Operating System Design and Implementation*, San Francisco, December 2004.

Scott Deerwester, Susan Dumais, George Furnas, Thomas Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407, 1990.

Thomas L. Griffiths, Mark Steyvers, and Joshua B. Tenenbaum. Topics in semantic representation. *Psychological Review*, 114(2):211–244, 2007.

Marti Hearst and Hinrich Schütze. Customizing a lexicon to better suit a computational task. In *ACL SIGLEX Workshop*, Columbus, Ohio, 1993.

Thomas Hofmann. Probabilistic latent semantic analysis. In *Uncertainty in Artificial Intelligence (UAI'99)*, Stockholm, Sweden, 1999.

Pentti Kanerva. *Sparse Distributed Memory*. MIT Press, 1988.

Thomas Landauer and Susan Dumais. A solution to Plato's problem: The latent semantic analysis theory of acquisition. *Psychological Review*, 104(2):211–240, 1997.

K Lund and C Burgess. Producing high-dimensional semantic spaces from lexical co-occurrence. *Behavior research methods, instruments and computers*, 28(22): 203–208, 1996.

Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. The MIT Press, Cambridge, Massachusetts, 1999.

Robert McArthur and Peter Bruza. *Dimensional Representations of Knowledge in Online Community*, pages 98–114. Advanced Information Processing. Springer-Verlag, 2003. URL `citeseer.ist.psu.edu/mcarthur03dimensional.html`.

Christos H. Papadimitriou, Hisao Tamaki, Prabhakar Raghavan, and Santosh Vempala. Latent semantic indexing: A probabilistic analysis. *J. Comput. Syst. Sci.*, 61(2):217–235, 2000.

Tony Plate. *Holographic Reduced Representations: Distributed Representation for Cognitive Structures*. CSLI Publications, 2003.

Magnus Sahlgren. An introduction to random indexing. In *Proceedings of the Methods and Applications of Semantic Indexing Workshop at the 7th International Conference on Terminology and Knowledge Engineering (TKE)*, Copenhagen, Denmark, 2005. SICS, Swedish Institute of Computer Science. URL `http://www.sics.se/~mange/papers/RI_intro.pdf`.

Magnus Sahlgren. *The Word-Space Model: Using distributional analysis to represent syntagmatic and paradigmatic relations between words in high-dimensional vector spaces*. PhD thesis, Department of Linguistics, Stockholm University, 2006.

Gerard Salton, editor. *The Smart Retrieval System – Experiments in Automatic Document Processing*. Prentice-Hall, Englewood Cliffs, NJ, 1971.

Gerard Salton and Michael McGill. *Introduction to modern information retrieval*. McGraw-Hill, New York, NY, 1983.

Hinrich Schütze. *Ambiguity resolution in language learning*. CSLI Publications, Stanford CA, 1997.

Hinrich Schütze. Automatic word sense discrimination. *Computational Linguistics*, 24(1):97–124, 1998. URL `citeseer.nj.nec.com/schutze98automatic.html`.

Lloyd N. Trefethen and David Bau. *Numerical Linear Algebra*. S.I.A.M., 1997.

C.J. van Rijsbergen. *The Geometry of Information Retrieval*. Cambridge University Press, 2004.

Dominic Widdows. *Geometry and Meaning*. CSLI publications, Stanford, California, 2004.

Dominic Widdows. Orthogonal negation in vector spaces for modelling word-meanings and document retrieval. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL)*, Sapporo, Japan, 2003a.

Dominic Widdows. Unsupervised methods for developing taxonomies by combining syntactic and statistical information. In *Proceedings of Human Langauge Technology / North American Chapter of the Association for Computational Linguistics*, Edmonton, Canada, 2003b.

Dominic Widdows. Semantic vector products: Some initial investigations. In *Quantum Interaction: Papers from the Second International Symposium*, Oxford, 2008.