2014-04-23

Termediator-II: Identification of Interdisciplinary Term Ambiguity Through Hierarchical Cluster Analysis

Owen G. Riley
Brigham Young University - Provo

Termediator-II: Identification of Interdisciplinary Term Ambiguity Through Hierarchical Cluster Analysis

Owen Riley

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Joseph J. Ekstrom, Chair
Dale C. Rowe
Kevin B. Tew

School of Technology

Brigham Young University

April 2014

ABSTRACT

Termediator-II: Identification of Interdisciplinary
Term Ambiguity Through Hierarchical
Cluster Analysis

Owen Riley
School of Technology, BYU
Master of Science

Technical disciplines are evolving rapidly leading to changes in their associated vocabularies. Confusion in interdisciplinary communication occurs due to this evolving terminology. Two causes of confusion are multiple definitions (overloaded terms) and synonymous terms. The formal names for these two problems are polysemy and synonymy. Termediator-I, a web application built on top of a collection of glossaries, uses definition count as a measure of term confusion. This tool was an attempt to identify confusing cross-disciplinary terms. As more glossaries were added to the collection, this measure became ineffective.

This thesis provides a measure of term polysemy. Term polysemy is effectively measured by semantically clustering the text concepts, or definitions, of each term and counting the number of resulting clusters. Hierarchical clustering uses a measure of proximity between the text concepts. Three such measures are evaluated: cosine similarity, latent semantic indexing, and latent Dirichlet allocation. Two linkage types, for determining cluster proximity during the hierarchical clustering process, are also evaluated: complete linkage and average linkage. Crowdsourcing through a web application was unsuccessfully attempted to obtain a viable clustering threshold by public consensus. An alternate metric of polysemy, convergence value, is identified and tested as a viable clustering threshold.

Six resulting lists of terms ranked by cluster count based on convergence values are generated, one for each similarity measure and linkage type combination. Each combination produces a competitive list, and no clear combination can be determined as superior. Semantic clustering successfully identifies polysemous terms, but each similarity measure and linkage type combination provides slightly different results.

Keywords: cosine similarity, LSI, LDA, text similarity, hierarchical clustering, polysemy

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

1.1 Background

Over the past 10 years there has been significant evolution in technical disciplines in Universities and professions around the world. Technical disciplines including computer science, information systems, computer engineering, software engineering, and information technology have emerged and evolved, each using different words and phrases to describe key concepts (Ekstrom, et al. 2010).

The individual definitions within a particular technical vocabulary change based upon academic, professional, and governmental influences. For example, before the term for the data structure now known as a "stack" was standardized, it was referred to as a LIFO (last in, first out), pushdown list, pile, or a reverse queue. It wasn't until the publication of *Fundamental Algorithms* by Donald Knuth that the term "stack" became standardized (Knuth 1968).

In 2010 charts were generated to illustrate topical differences between technical disciplines using academic bodies of knowledge (Ekstrom, et al. 2010). The charts were generated using the suggested topic hours which are clearly defined within the bodies of knowledge. The disciplines of information technology and Information Systems are illustrated in figure 1-1. Of similar origin, these two disciplines are clearly diverging in their emphases.

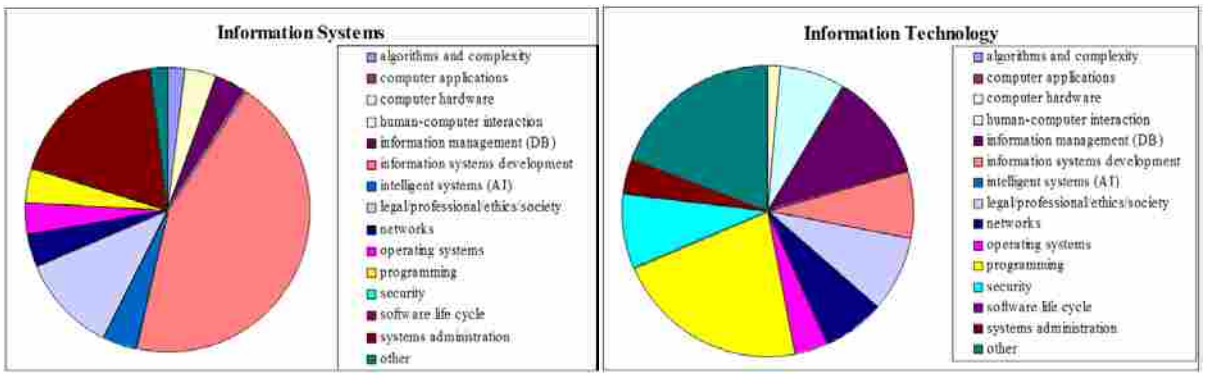

Figure 1-1: Differences between Two Technical Disciplines

Terminologies evolve as disciplines focus on different specialized areas within computing. For example, the word "tuple" could refer to a record of a database to a database engineer or a collection of values to a software developer. This discipline specific evolution leads to cross disciplinary communication problems.

1.2 Problem Statement

Multiple definitions and synonymous terms (polysemy and synonymy) cause terminological confusion between and within disciplines which often results in lost productivity. Constantly evolving professional terminology exacerbates the problem. This is especially true in rapidly evolving technology related disciplines including information technology, computer science, information systems, and software engineering (Ekstrom, et al. 2010).

1.3 Mitigating the Problem

A list of potentially confusing cross disciplinary terms could help mitigate this problem by providing teams with guidance as to how collaborating disciplines use the words. The

problem then becomes locating repositories of discipline specific terminology. Two potential sources of term data are dictionaries and glossaries.

Glossaries, like dictionaries, contain lists of terms associated with concepts (definitions). Dictionaries start with words and produce definitions applicable across all domains ranked by usage. On the other hand, glossaries start with concepts and produce terms (words or phrases) to accurately represent the ideas. Glossaries are better suited to resolve terminological confusion as they seek to facilitate understanding about concepts at a domain level while dictionaries seek to simply define words at a global level.

There is an existing data-set which contains 399 different domain specific glossaries compiled into a standardized XML format (Richards 2013). This aggregated glossary corpus contains 40,065 terms with a total of 71,199 text concepts (or definitions) from 15 domains. We refer to this corpus as "the compendium". Our semantic clustering process combines hierarchical agglomerative clustering algorithms with text similarity measures to determine term polysemy. We must first understand our underlying terminology model before we can address the specifics of this semantic clustering process.

1.4 **Terminology Model**

1.4.1 Text Concepts and Abstract Concepts

The source data for this research is a collection of technical glossaries, each consisting of terms and concepts. A term is an entry in a glossary which can be a word or phrase while a concept is the information defining that term. We needed to distinguish between the text used to describe a term and the idea that the text represents. We call the text element a "text concept" and the idea an "abstract concept". Text concepts are the text we see when we expand a term

within a glossary. For example, a text concept of the term “computer” is “a general-purpose tool for communication and control as well as computation”. An abstract concept is the semantic idea or “platonic ideal” that we are trying to represent with a text concept. Any attempt to define the abstract concept using text simply creates another text concept. Thus, an abstract concept is the set of text concepts that attempt to represent the same idea.

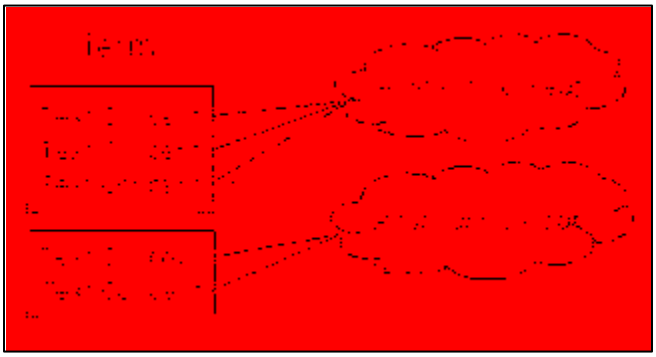

Figure 1-2: Relationship between Abstract and Text Concepts

We are interested in two sources of confusion that occur in communication: synonymy and polysemy. With these definitions of term, text concept, and abstract concept, we can now clearly define both sources of confusion.

1.4.2 Polysemy – One Term to Many Abstract Concepts

The first type of ambiguity is referred to as polysemy. A term is polysemous if it can mean two or more different ideas, or in our case if the term has text concepts belonging to different abstract concepts (Figure 1-3).

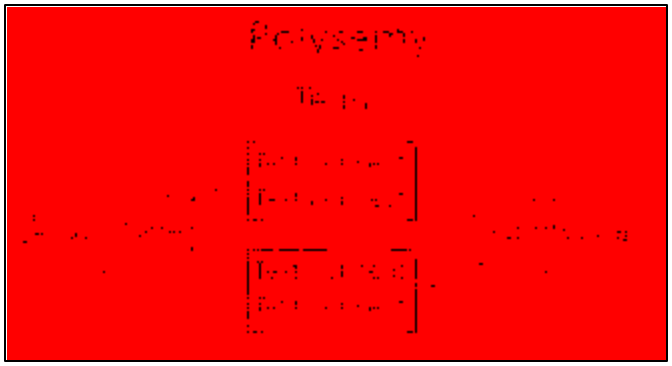

Figure 1-3: Polysemy Diagram

We can measure term polysemy using the text concepts found in glossaries. The polysemy of each term is quantified by semantically organizing the term's associated text concepts into clusters that represent abstract concepts. The degree of polysemy for each term is represented by its number of abstract concepts. Sorting the terms in the compendium by their number of abstract concepts in descending order provides a measure of the most polysemous or potentially confusing terms.

1.4.3 Synonymy – One Abstract Concept to Many Terms

The second type of ambiguity is called synonymy, where two or more terms have text concepts that belong to the same abstract concept.

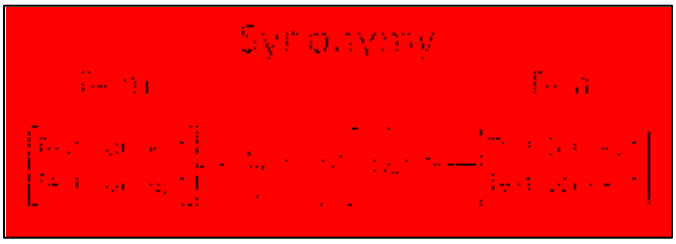

Figure 1-4: Synonymy Diagram

Unlike polysemy, synonymy is difficult to measure. Determining whether two terms are synonymous or simply highly similar is subjective and often domain specific. In the Termediator-I project, a basic measure (cosine similarity) was used to define distance between text concepts which attempted to identify synonyms. Termediator-I displayed a similarity score between the text concepts of potential synonyms and let the users judge whether they were actually synonyms or not.

Introducing synonymy analysis in addition to polysemy would introduce additional complexity: dealing with the combination of polysemy and synonymy. Some term A can be synonymous with one meaning of term B, and a different meaning of B can be synonymous with some term C. This relationship would not necessarily be transitive, meaning that A and C are not always synonymous given the fact that A is synonymous to B and B is synonymous to C.

Since Termediator-I has already targeted synonymy and to avoid issues stemming from synonymy subjectivity, our semantic clustering process in Termediator-II is targeted solely at measuring term polysemy.

1.5 **Research Questions**

Our Termediator-II analysis system addresses the following questions:

1. How well can semantic clustering of text concepts be used to identify abstract concepts?
2. How well can semantic clustering of text concepts reveal polysemous terms?
3. Which combination of semantic similarity measure and clustering algorithm is best at grouping semantically similar text concepts?
4. What terms are most likely to cause confusion among disciplines?

1.6 Delimitations

This research is bounded by the following parameters:

1. Similarity measures will be limited to cosine similarity, latent semantic indexing, and latent Dirichlet allocation. See Chapter 2 for a more complete explanation.
2. Hierarchical clustering will be used as the cluster method because the number of clusters to be formed cannot be predetermined.
3. Of the different linkage types associated with hierarchical clustering algorithms, only complete and average linkages will be analyzed.
4. All test applications will be built and processed using Python 2.7.

1.7 Glossary of Terms

Abstract Concept: The set of text concepts which all describe the same semantic idea.

Body of Knowledge: Typically a multi-tiered document outlining different topics for a particular discipline created by one or more professional organizations.

Compendium: The source data of this research, which is made up of many technical glossaries which were merged together.

Glossary: A collection of terms and text concepts from a single source.

Convergence Value: The smallest threshold value for a particular similarity measure and linkage type that produces a single cluster for a particular term.

Polysemy: Overloaded term where two or more of its text concepts belong to different abstract concepts.

Synonymy: Terms where each contain at least one concept which belong to the same abstract concept.

Term: An entry in a glossary which can be a word or phrase.

Termediator: An ongoing research project which created the compendium and performs basic text concept to term similarity measures.

Text Concept: One or more sentences explaining a specific term in a glossary.

Threshold: A hierarchical clustering distance parameter that determines when to stop combining clusters.

1.8 Thesis Statement

Term polysemy can be measured using hierarchical clustering algorithms with text similarity measures on a collection of glossaries.

To establish a background for Termediator-II, we will present a brief history of Termediator and its shortcomings in identifying polysemous terms. We will then describe how the hierarchical clustering process functions. An explanation of three different similarity measures used in the semantic clustering will be introduced and explained. Once all these pieces are understood, we can proceed to the methodology of this research.

2 BACKGROUND

2.1 Evolution of Disciplines

Relationships between computing and society have evolved over the past decade which led Brigham Young University IT professors, Ekstrom and Lunt, to research the evolution of technical disciplines (Ekstrom 2010). According to them, the "inevitable divergence of terminology and evolution of social structures around common interests has led to… less communication between closely related fields of study". The evolution of technical disciplines led to very specific computing majors emerging, such as bioinformatics and geographic information systems, each with their own nuanced vocabularies. Bodies of knowledge have been published in attempts to capture important information with regards to these fields, including relevant concepts, terms, and activities within a domain.

As evidence of the evolution of technical disciplines, a document entitled "Computing Curricula" by the ACM Council and the IEEE-CS Board of Governors was created in 2001. This document was intended to be a guide for developing various technical undergraduate degree programs and included related technical disciplines including computer science, information systems, and information technology. For each undergraduate program, the document included a short history of its development, a brief description of what it is, visual representations of the topics with the depth of each, and suggested career paths. Such a document was necessary to capture the differences between those related disciplines amidst their evolution.

The evolution of these technical disciplines inevitably leads to different specialized vocabularies, contributing to the communication problems that stem from polysemy and synonymy. Confusion caused solely by synonymy, when a person communicates a specific abstract concept using an unfamiliar term to a listener who attaches that same idea to another term, can often be recognized by that party and resolved. In contrast, a miscommunication caused by polysemy, when a person uses a familiar term but intends a different meaning than what is received, can initially go undetected making it much more serious. For example, an engineer is told to lay wire to create a "trunk" between two switches, which can either mean a single link that holds many signals or many links that hold one signal. If the wrong meaning is interpreted by the engineer, the mistake can quickly become very costly in both time and money.

Ekstrom's subsequent research into these communication problems laid the foundation for our work, starting with his aggregated glossary prototype (Ekstrom 2010).

2.2 Previous Glossary Aggregation

A glossary aggregation prototype was built which contained the text of ISO/IEC/IEEE 24765 (SEVocab). It was then parsed and normalized into a simple XML structure illustrated in figure 2-1.

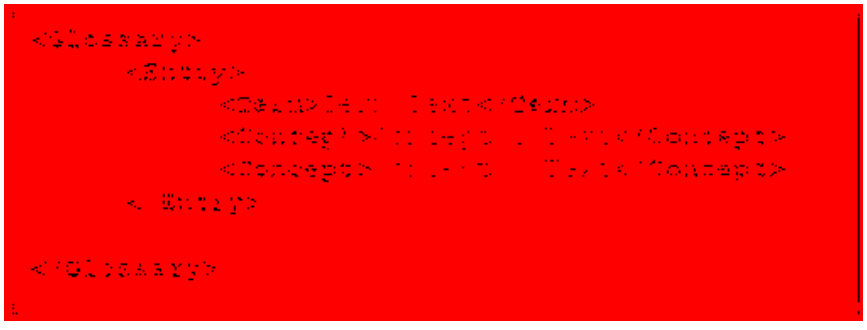

Figure 2-1: Simplified XML Format

The glossary aggregation prototype started with the SEVocab collection which contained 124 software and system engineering glossaries edited by the IEEE computer society. The editing process excluded terms and concepts that were "considered parochial to one group or organization" or "whose meanings could be inferred from the definitions of the component words" or "whose meaning in the IT field could be directly inferred from their common English meaning" (IEEE 2010). A web application was built which could access the data within SEVocab. The interface consisted of a sorted list of terms along with the number of associated text concepts which, when selected, would display the text of those concepts in a nearby frame.

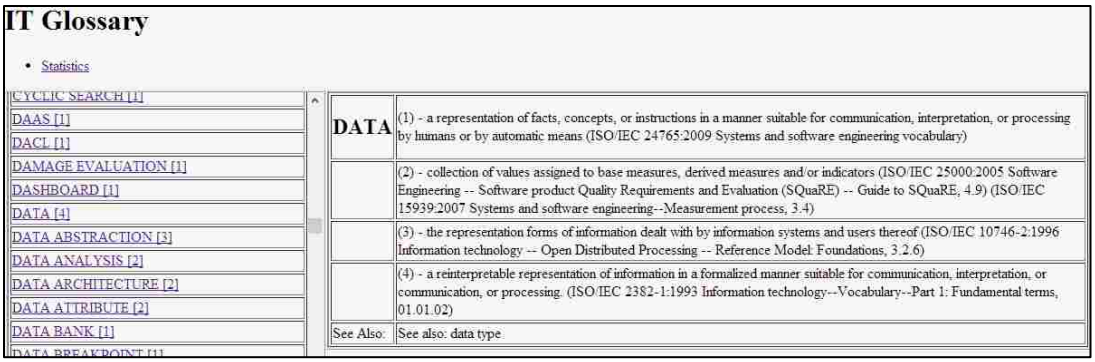

Figure 2-2: Initial Glossary Aggregation Prototype with the Term "DATA" Selected

Following the path laid out by the initial glossary aggregation prototype, an updated prototype called Termediator-I was created (Richards 2013). In addition to the edited glossaries used in the original aggregation prototype, Termediator-I contained dozens of additional technical glossaries. It also featured a slightly different interface which showed text concepts underneath terms instead of off to the side.

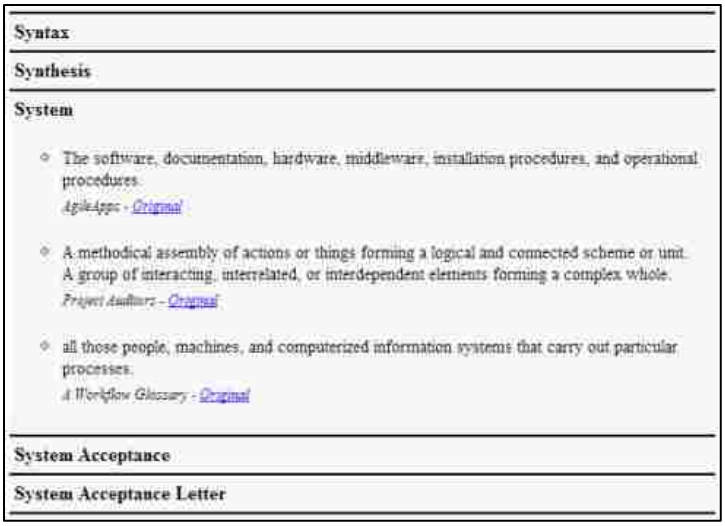

Figure 2-3: Basic Termediator Interface

In addition to presenting an interactive aggregated glossary, Termediator-I presents a list of terms with text concepts below them. Clicking on a concept then causes Termediator-I to identify the terms that contain concepts which are "closest" to the selected concept using cosine similarity.

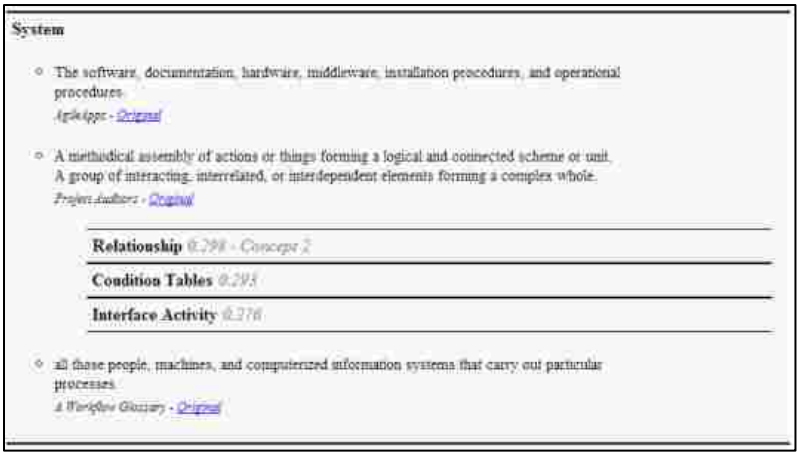

Figure 2-4: Termediator's Similarity Interface

The analysis of Termediator-I data and the data from the initial glossary aggregation prototype led to the conclusion that the number of text concepts associated with a term was a reasonable indicator of that term's polysemy. The initial glossary aggregation prototype produced a strong list of polysemous terms when sorted by the number of a term's associated text concepts. Once the Termediator-I compendium had grown to 160 glossaries, the terms were again sorted by their number of text concepts resulting in polysemous terms such as "constraint" and "activity" once again appearing near the top. However, some noise was also introduced with the addition of the new glossaries, such as the term "Gantt Chart" being ranked as the 9th most complex term in the entire compendium which began to challenge the naive hypothesis.

As the number of glossaries grew, the number of common terms used across disciplines also grew. With just under 400 glossaries aggregated together, the terms with the highest number of text concepts were not very polysemous at all. Instead of containing polysemous terms like "constraint", the list contained terms that have little variance in their meanings such as "HTML" and "Download". Table 2-1 shows the evolution of Termediator's top confusing terms using the text concept counts vs number of glossaries in the collection.

The new glossaries added into the SEVocab collection introduced a new problem: abstract concept duplication. Our initial metric of polysemy, text concept count, was premised on the idea that each text concept represented a different abstract concept. SEVocab's editing process by the IEEE produced a glossary collection that followed this premise by combining similar text concepts together. After the unedited glossaries were added, many terms had multiple text concepts communicating the exact same idea which rendered text concept counting ineffective. We needed metrics of polysemy that could address abstract concept duplication introduced by unedited glossaries.

Table 2-1: Evolution of the Top 10 Polysemous Terms by Text Concept Count

124 Glossaries		160 Glossaries		399 Glossaries	
Term	**# Concepts**	**Term**	**# Concepts**	**Term**	**# Concepts**
Constraint	14	Process	22	Bandwidth	54
Process	13	Activity	18	HTML	53
Entity	10	Task	17	Firewall	50
Measure	10	Baseline	15	Browser	49
Function	10	Constraint	14	Software	49
Baseline	9	Stakeholder	14	Internet	47
Implementation	9	Risk	13	URL	46
Input	9	C(A)	12	GIF	44
Activity	9	Gantt Chart	12	Download	43
System	9	Software	12	Virus	43

We address abstract concept duplication by combining semantically similar text concepts using hierarchical clustering. Each term has its associated text concepts semantically clustered so that each cluster represents an automatically derived approximation of an abstract concept, or collection of semantically similar text concepts. We will then count the clusters associated with terms exactly like we counted the filtered concepts in SEVocab with the same result: ranking by a measure of term polysemy. To accomplish that task, we must first understand what clustering is and how it can be used to semantically group text concepts together.

2.3 **Clustering Techniques**

The idea of clustering data has been around as early as the 1930's (Tryon 1939). The general idea is to group (or cluster) objects together such that objects within a cluster are more similar to each other than to objects from other clusters. Robert Tryon, one of the earliest

researchers to give clustering a practical application, referred to this area of study as cluster analysis.

Since the original idea of cluster analysis was published, many different clustering techniques were invented. Each of these variations have different notions when it comes to determining what a cluster is and how it is formed. As a result, clustering is often defined as exploratory data mining, where trial and error is often necessary to determine optimal results.

"K-nearest neighbors" is a popular clustering algorithm. This algorithm, given a set of pre-classified data points, determines which cluster each point belongs to based on the classification of its *k* nearest neighbors. The value chosen for k is often determined through experimentation. Another popular clustering algorithm is "K-means" which has a total of k centroids that are created and act as centroids to their own clusters. Through a series of iterations, these centroids slowly shift around until an optimal formation is discovered that encapsulates all the data. A third clustering algorithm, hierarchical clustering, takes a completely different approach than the previous methods discussed.

Hierarchical agglomerative clustering (HAC) initially places each text concept into its own cluster. Next HAC repeatedly combines the closest two clusters together using an associated proximity matrix until every text concept is a part of a single parent cluster. The proximity matrix stores the pairwise distances between the text concepts. HAC results are typically represented using a specific tree diagram called a dendrogram (figure 2-5).

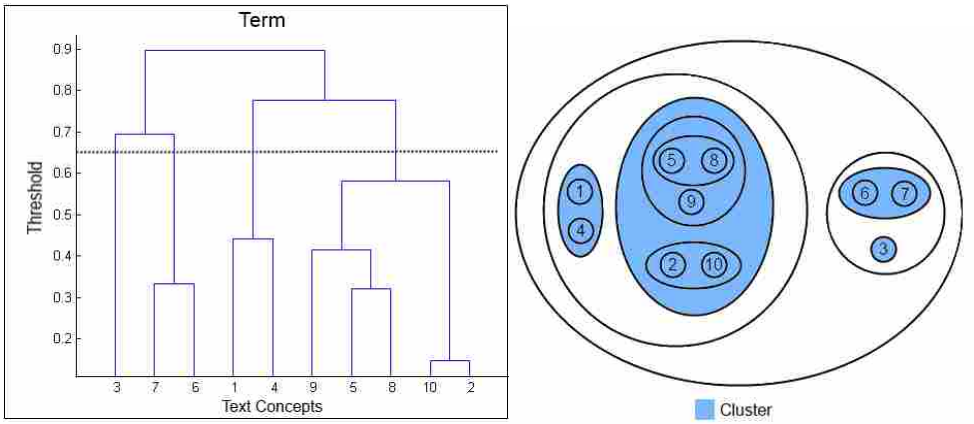

Figure 2-5: Sample Dendrogram (left) and Alternative Representation (right)

The data in the dendrogram can be reduced to a single metric of polysemy by establishing a threshold value. The threshold value is a distance measure that determines when to stop the clustering process. This value takes a horizontal slice of the dendrogram, producing a cluster for each line it intersects. The number of clusters produced ranges from a single cluster when drawn at the top of the dendrogram to the number of text concepts when drawn at the bottom. Constructing the proximity matrix for text concepts is the first step to performing hierarchical clustering.

2.4 Similarity Measures

2.4.1 Term Frequency, Inverse Document Frequency and Cosine Similarity

The vector space model for determining text similarity was originally proposed by Salton and his team from Cornell University (Salton 1975). They came up with the idea of representing texts as numerical vectors in order to use mathematics to compare them together. The vectors are composed of N dimensions where N is the number of distinct terms between them. Different weights for these dimensions were suggested such as a binary system (one if the term occurs at

least once, or zero otherwise), a raw term frequency, or a weighted term frequency. After converting two texts into vectors, Salton claims that "two documents with similar index terms are then represented by points that are very close together in the space". As an example, consider the following two simple phrases P_1 and P_2:

P_1 = the red cat

P_2 = the angry red dog

First, we take each distinct term and create vectors such that the vectors of P_1 and P_2 become what is shown in table 2-2.

Table 2-2: Text to Vector Illustration

	THE	RED	CAT	ANGRY	DOG
P_1	1	1	1	0	0
P_2	1	1	0	1	1

Thus, P_1 can be represented as the vector (1,1,1,0,0) and P_2 can be represented as the vector (1,1,0,1,1). Using this type of approach, we can convert the entire compendium into a collection of different vectors with N dimensions, where N is the number of distinct words found throughout the corpus.

When determining the optimal weight for the different dimensions in the vector, Salton's best solution was to multiply raw term frequency (TF) with the inverse document frequency (IDF). The IDF was originally defined in his work as the following:

$$(IDF)_k = \lceil \log_2 n \rceil - \lceil \log_2 d_k \rceil + 1 \quad (2\text{-}1)$$

Since then, it has become standardized as:

$$\text{idf}(t, D) = \log \frac{|D|}{|\{d \in D : t \in d\}|} \quad (2\text{-}2)$$

In this formula *t* is a term and *D* is the set of all documents comprising the corpus. It is the logarithm of the total number of documents divided by the number of documents that contain the term *t*. The base of the logarithm can be any value as it only changes the impact of the IDF portion and not the order of results. When IDF is used to help weigh a vector, it causes rarer terms to have more weight while also reducing the weight of very common terms. In Salton's experiments, using IDF with the raw term frequency gave an average precision and recall gain of 14 percent over raw term frequency alone (Salton 1975).

In addition to defining TF and IDF, Salton also suggested different methods for comparing the resulting vectors. One of these methods was to compare the angle between the vectors, stating that similarity is inversely related to that angle. If the angle is close to 0, that means the vectors are almost pointing in the exact same place in space which he inferred meant they would be semantically similar.

A simple and popular method for comparing the resulting vectors is known as cosine similarity. Using Salton's idea of comparing angles, one can take the cosine of the two vectors to get a distance metric between the two texts.

$$Similarity = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}} \quad (2\text{-}3)$$

A benefit of the cosine value is it converts all angular distances to a value between 0 and 1. The cosine of 0 degrees is 1 and the cosine of 90 degrees is 0, which means that very similar vectors will approach 1 while dissimilar vectors tend towards 0. While a true cosine ranges from -1 to 1, we use the absolute value so that it is bound between 0 and 1.

2.4.2 Latent Semantic Indexing

Another method of detecting semantic similarity between texts is called "Latent Semantic Indexing" (LSI), or sometimes "Latent Semantic Analysis" (LSA). Patented in 1989 by Scott Deerwester et al, LSI is considered a highly effective similarity measure. LSI is designed to "take advantage of implicit higher-order structure in the association of terms with documents" (Deerwester, 1990). The idea behind LSI is that words that are used in similar contexts with high frequency tend to have similar meanings. LSI is based on the statistical technique of correspondence analysis. To illustrate the weaknesses of straight keyword matching, an example is given in Table 2-3.

Table 2-3: Weaknesses of Keyword Matching

	Access	Document	Retrieval	Information	Theory	Database	Indexing	Computer	REL	MATCH
Doc 1	X	X	X			X	X		R	
Doc 2				X	X			X		M
Doc 3			X	X				X	R	M

**Query: "IDF in computer-based information lookup"*

The example in table 2-3 illustrates that even if query words match a document, it doesn't necessarily mean they are semantically similar. Document 2 shares the words "computer" and "information" with the query but would not address the query's intention. To a human observer, it is obvious that document 1 would match the query because both involve document retrieval, but a lack of shared terms causes them to be unmatched.

According to Deerwester, LSI needed to be able to "predict what terms 'really' are implied by a query or apply to a document". To do this, the first step is to create a matrix of terms by documents which is a common information retrieval technique. Using another technique called singular value decomposition, the matrix is then transformed into a latent semantic structure model. A more detailed analysis of this technique is beyond the scope of this thesis.

This technique was selected because of its inherent ability to deal with both polysemy and synonymy. The authors, while introducing the idea of LSI, expound on both of these linguistic hurdles specifically and discuss how this can overcome both when detecting similarity. Such a similarity technique could potentially be the best method for creating the semantic clusters defined earlier.

2.4.3 Latent Dirichlet Allocation

Researchers from Stanford University and the University of California invented a new type of similarity measure in 2003 based on probabilistic methods called latent Dirichlet allocation (LDA) (Blei 2003). It is different than the previous similarity methods because it uses topic modeling to represent a corpora. The researchers give a brief history of information retrieval, discussing the benefits of LSI in the process. Despite the improvement that came from using LSI over conventional TF*IDF measures, they felt the need to further dive into the

probabilistic nature of word occurrences. Their simple definition of LDA is "a generative probabilistic model of a corpus… [where] documents are represented as random mixtures over latent topics".

The mathematics used to derive LDA are extensive and far beyond the scope of this thesis. This method creates distributions over words as topics and then, based on the words of the small texts being compared, creates a vector with a dimension for every topic. The weight of each dimension is a measure of the applicability of the text to that particular topic. Similarity between texts is then simply the cosine of these two topic vectors. This method was chosen as a representative of the topic modeling approach to similarity in an attempt to determine if a probabilistic approach will outperform correspondence analysis (LSI) or the purely lexical approach (TF*IDF).

Cosine similarity, LSI, or LDA can all be used to create proximity matrices for the text concepts within a term. Once text concepts start being clustered together, the proximity matrix alone is insufficient because it doesn't outline the proximity of clusters of text concepts. To determine cluster proximity, we need to identify and use a parameter known as a linkage type.

2.5 **Linkage Types**

The proximity matrix alone is sufficient only when the clusters contain one text concept. Once clusters contain more than one text concept, linkage types are used to determining cluster proximity. One set of researchers from the University of Sheffield performed some analysis on a variety of linkage types of which we will only investigate three in particular (El-Hamdouchi 1987).

The first and most basic linkage type is referred to as single linkage. When two clusters are compared, single linkage defines their proximity as equal to the distance of the closest two

points between them, or in our case the two most similar text concepts. One issue that researchers have had in the past with this linkage type is what is called “chaining” which is when it “forms loosely bound clusters with little internal cohesion” due to the newest member continually chaining to another new member (El-Hamdouchi 1987). The research team from the University of Sheffield concluded that this method proved to be the worst of the linkage types in the document retrieval space for their particular corpus. We chose not to evaluate single linkage because additional research has proven that it “generally gives results that are far inferior to those obtainable when the other hierarchic agglomerative methods are used” (Willett 1988).

Where single linkage compares the closest two points, complete linkage compares the furthest two points, or the two most dissimilar members of the clusters. Complete linkage tends to form more compact clusters of approximately equal diameter. Unlike single linkage, complete linkage does not suffer from chaining.

The third type is average linkage, which does exactly what the name implies. When two clusters are compared, their distance is determined by the average distance between any point in the first and any points in the second. It is more computationally intensive than the first two types and tends to be a midpoint between the single and complete linkages.

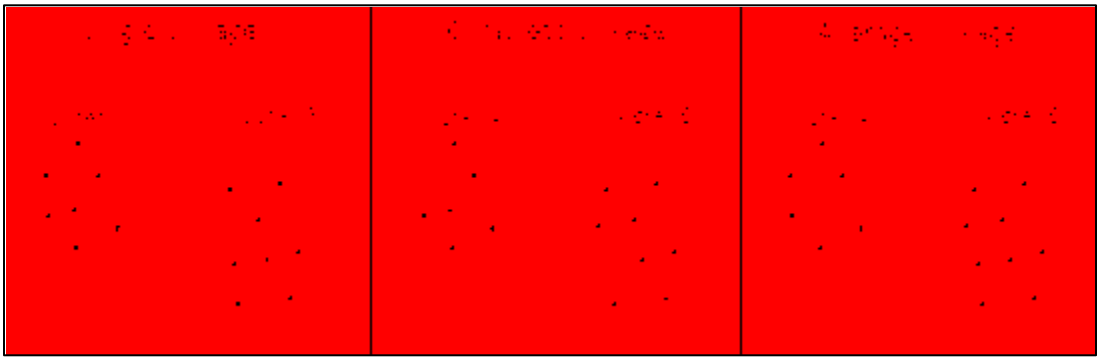

Figure 2-6: The Three Most Common Linkage Types

It is unknown which of these two latter linkage types will perform the best on the compendium. This thesis tests both so that their relative performance can be compared.

2.6 Other Work on Short Text Similarity

Anna Huang has performed similar work in regards to semantic clustering (Huang 2008). In her paper, she describes five different vector based similarity measures which are Euclidean, cosine, Jaccard coefficient, Pearson correlation coefficient, and averaged Kullback-Leibler divergence. Each is a slightly different application of the vector model. Huang's purpose was to determine which was the most effective at clustering text documents together.

Huang chose to use a standard K-means clustering algorithm which is fundamentally different than our hierarchical approach. Huang uses K-means clustering because she knows beforehand how many clusters she wishes to form. Her source data was manually pre-classified with a specific number of different categories which she tried to match with k-means clustering. Upon analyzing the results, it was determined that each similarity measure had comparable effectiveness with the exception of Euclidean, which underperformed the others.

The results of Huang's research encouraged our decision to use only one vector based similarity measure, cosine similarity. Testing additional vector based similarity measures would most likely replicate this prior research, reinforcing the idea that each of the vector based approaches are fairly equal. Using the cosine measure is also advantageous because of its relative simplicity in implementation compared to the other vector based similarity measures.

A variety of other papers have been published on detecting similarity specifically between short texts. One research team led by Donald Metzler focused on solving the vocabulary mismatch problem, which is the problem of using different words to describe the same idea (Metzler 2007). Their research looked first at a lexical approach, which included exact matching,

phrase matching, and subset matching. They also examined probabilistic methods, such as word co-occurrence, to deal specifically with vocabulary mismatching. Using a set of 363,822 MSN search queries from 2005, the researchers tested both methods independently and combined. What they discovered was that lexical matching was better at discovering extremely similar queries and that probabilistic matching was better at discovering moderately similar queries. The best method for determining similarity came from combining both methods, placing lexical matches first followed by the probabilistic ones.

We decided not to try and implement their specific similarity method into our tests. The source data for their experiments consisted of internet search queries which are fundamentally different than text concepts found in glossaries. Queries are concise requests for information while text concepts are explanations of terms.

One method they used to evaluate resulting matches was applicable to our research. A four point scale was established which consisted of a rank, a description, and an example (Table 2-4). A person than ranked a set of randomly selected matches from the experiments to determine their effectiveness. We used a variation of this approach when evaluating the quality of our semantic clustering.

Table 2-4: Four Point Rating Scale (Metzler 2007)

Judgment	Description	Examples (Query / Candidate)
Excellent	The candidate is *semantically equivalent* to the user query.	atlanta ga / atlanta georgia
Good	The candidate is related to (but not identical to) the query intent and it is likely the *user would be interested in the candidate.*	seattle mariners / seattle baseball tickets
Fair	The candidate is related to the query intent, but in an overly vague or specific manner that results in the *user having little, if any, interest in the candidate.*	hyundia azera / new york car show
Bad	The candidate is *unrelated* to the query intent	web visitor count / coin counter

Another research group led by Vasileios Hatzivassiloglou performed their own set of tests to determine short text similarity (Hatzivass 1999). Like previous research, their focus was on overcoming the vocabulary mismatch problem. Their research investigated a service called WordNet, which is a large lexical database for the English language funded by the National Science Foundation. One feature of WordNet is the ability to give a word and receive a list of potential synonyms.

By using the WordNet service, the researchers are able to find similarity between short texts that don't share any words. Such a scenario would fail in any vector-based approach automatically because they only rely on shared words to detect similarity. This research differs from our own in their specific definition of similarity. They define two texts as similar if they both "focus on a common concept, actor, object, or action". They further assert that the common actor or object must be subjected to the same action or description in both texts. Our definition of similar is not as restricted. This limits the applicability of their research to our problem.

They create their own method for detecting similarity which involves identifying basic and composite features that are shared between the two texts. Their method ultimately performs better on source data consisting of 16,000 news articles from Reuters, with each paragraph acting as a unit of text. Hatzivassiloglou claims that although they are using paragraphs as their text unit, sentences would work in the same way.

Once again, their methods could be implemented as a potential similarity measure to experiment with, but it was decided against. Their implementation requires the implementation of a classifier which was trained using a large number of manually tagged documents. The source data used for their research was pre-classified into topical categories so training was of little consequence to implement. The source data for our semantic clustering is only pre-tagged

with a domain which, due to the heavy overlap between disciplines, would be an ineffective measure.

2.7 **Methodology Decisions based on the Literature Review**

From the literature review, a few conclusions can be drawn. First, there is a growing need as disciplines evolve to disambiguate terminology. There must be some way to help these divergent disciplines communicate effectively despite the presence of polysemous and synonymous terms. The research we are performing will help to identify the most polysemous of these terms, and future work may perhaps use these identified terms to further aid in these efforts.

The second conclusion is that there is an adequate data set that was created to deal with terminological disambiguation. Starting in early 2012, the Termediator-I research team collected technical glossaries so that terminological disambiguation research could take place. Using this data set currently containing 399 glossaries, the Termediator-I research project implements basic similarity measures allowing users to find similar terms based solely on their text concepts. We will additionally use this dataset to identify the most polysemous terms present in these glossaries.

Third, hierarchical clustering appears to be the most efficient clustering algorithm for dealing with this problem. Using a threshold value that will be determined experimentally, hierarchical clustering can be used to generate an unknown number of clusters. Using clustering algorithms like k-means and k-nearest-neighbor would be ineffective since each term has an un-predetermined number of abstract concepts associated with it and we don't have pre-classified training data.

Fourth, there are a large variety of standard similarity measures that could be used to try and detect similarity. Many core similarity measures have a number of derivative techniques that slightly modify the core functionality. Testing each established derivative technique would not be efficient so only a few core similarity measures will be used for experimentation. Specifically, the cosine similarity measure will represent the vector based approach, the latent semantic indexing technique will represent the statistical approach, and the latent Dirichlet analysis technique will represent the probabilistic topic modeling approach. These three measures are each significantly different from each other and will offer a good generalization of each of the approaches.

Lastly, there are a number of different custom similarity measures that research teams have published recently to calculate semantic similarity of specific short texts. Each approach that was reviewed had positive results but was the result of a combination of existing techniques. After reviewing each and weighing their strengths and weaknesses, we determined that it would be best to leave the experimentation for our problem to the core similarity measures discussed above.

3 METHODOLOGY

3.1 Source Data - Termediator

The source data for this research was introduced and outlined in Chapter 2. It is a collection of 399 technical glossaries which have been merged together into one compendium. Each entry contains a term with one or more text concepts associated with it. In total there are 40,065 distinct terms and 71,199 different text concepts.

3.2 Automated Clustering

Our semantic clustering was performed using three main variables:

- Similarity Measure
- Hierarchical Linkage Type
- Threshold

Each of these variables will be discussed, including what role they play, the different permutations each has, and the estimated impact on the quality of the resulting clusters.

3.3 Similarity Measure

The function of the similarity measure in the hierarchical clustering process is to establish text concept proximity. The three similarity measures used to produce proximity matrices we evaluated were:

- Cosine Similarity
- Latent Semantic Indexing
- Latent Dirichlet Allocation

Each of the similarity measures has already been introduced and described in Chapter 2 in detail. To summarize, each was chosen to represent a different approach to similarity detection. The cosine similarity measure uses the basic vector model introduced by Salton back in the 70's (Salton 1975). Converting each text concept into a vector, this method measures the cosine between two vectors to determine the semantic similarity. Of the three measures, this was expected to perform the poorest due to the fact it depends on exact word matching.

Latent semantic analysis uses singular value decomposition to determine the similarity of two texts. The fundamental principle behind LSI is that if words co-occur often, they will be semantically similar. Even with short texts, this idea can be effective in successfully finding semantic similarity. We expected this measure to perform better than the cosine method in identifying abstract concepts.

The final measure, latent Dirichlet allocation, is based on the idea of topic modelling. A topic is defined as a distribution over words, so text concepts will be matched with specific topics and then subsequently compared for similarity. Of the three methods, it was expected that this would perform the best.

Similarity values needed to be converted to distance before they can be used as proximities in hierarchical clustering. All three of the similarity measures produce similarity values between zero and one with higher values indicating greater similarity. Taking the complement of the similarity values by subtracting them from one will convert similarity to distance, causing highly similar values to approach a distance of zero and dissimilar values to

approach a distance of one. Each term will have a different proximity matrix which is the collection of the distance values generated from the complemented similarity measures between each pair of the term's text concepts.

3.3.1 Stopword Removal and Stemming

Before we used any of the similarity measures, we performed two optimizations on every text concept found in the compendium. The first optimization was stop word removal. A majority of the text concepts contained words that are not useful in determining the text concept's semantic meaning. Some simple examples of such words are "the", "there", and "a". Removing these words "increases retrieval efficiency and generally improves retrieval effectiveness" (Croft 2010). The list of stop words used for this function is included in appendix A.

In addition to removing the stop words in each concept, another technique called stemming was applied. Stemming is the process of "[capturing] the relationships between different variations of a word" (Croft 2010). With stemming, words are broken down to a root word so that these and other similar words can be compared together. To perform the stemming, we used a common algorithmic stemmer called the Porter stemmer. It follows a series of steps to transform an input word into a common stem and "has been shown to be effective in a number of… evaluations and search applications" (Van Rijsbergen 1980). For example, the words "computer", "computers", and "computing" would all become the stem "comput", causing them to be evaluated as the same word. The stemmer script we used is a Python version obtained from a web site maintained by Martin Porter, the author of the Porter stemmer (Porter 2001).

After performing stop word removal and stemming, each text concept became a list of semantically significant stems without stop words which were then suitable to be used with any of the three similarity measures mentioned.

3.3.2 Training Topics

The number of topics that the LSI and LDA training algorithms create has to be predetermined. Prior research into this dimensionality by Roger Bradford revealed that an ideal number of topics to create for any sufficiently large corpus is between 300 and 500, and that any value chosen outside this range results in "significant distortions" (Bradford 2008). Based on this data, we decided to create 400 topics when training both the LSI and LDA techniques.

3.3.3 Training Corpora

LSI and LDA both require training before they can be used to evaluate text similarity. Initially, the compendium was used as the training corpus with the assumption that the text concepts would contain enough information to generate effective models. An alternative source of training data would be bodies of knowledge (BOK). BOKs are official domain specific documents produced by academic and professional expert groups which outline topics of study. The BOKs generally follow a hierarchical structure with three main levels. The top level is a knowledge area, which is a word or short phrase which defines a large area within the discipline. An example of this in the Computer Science body of knowledge is "operating systems". The level below this is typically called a unit such as "concurrency" and "memory management" for the example knowledge area. The third level is comprised of topics, which are very specific words or phrases such as "paging and virtual memory" and "caching" for the memory management unit.

16 different bodies of knowledge from several domains were obtained and converted into a standard XML format to act as a potential training corpus. The complete collection is shown in table 3-1.

Table 3-1: Bodies of Knowledge in Our Collection

NAME	AUTHOR	YEAR
A Guide to the Project Management Body of Knowledge Fourth Edition	Project Management Institute	2008
Business Analysis Body of Knowledge	International Institute of Business Analysis	2009
Computer Science Curricula 2013	Association for Computing Machinery IEEE Computer Society	2012
Computer Science Curriculum 2008: An Interim Revision of CS 2001	Association for Computing Machinery IEEE Computer Society	2008
Computing Curricula 2001 Computer Science	Association for Computing Machinery IEEE Computer Society	2001
Curriculum Guidelines for Graduate Degree Programs in Software Engineering	Integrated Software & Systems Engineering Curriculum	2009
Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering	Association for Computing Machinery IEEE Computer Society	2004
Curriculum Guidelines for Undergraduate Degree Programs in Information Technology	Association for Computing Machinery IEEE Computer Society	2008
Curriculum Guidelines for Undergraduate Degree Programs in Information Systems	Association for Information Systems	2010
Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering	Association for Computing Machinery IEEE Computer Society	2004
Enterprise Architecture Body of Knowledge	Mitre	2004
Essential Body of Knowledge	US Department of Homeland Security	2008
Essential Body of Knowledge	US Department of Energy	2011
Guide to the Quality Body of Knowledge	American Society for Quality	2009
Human Resources Professionals in Canada Revised Body of Knowledge	Canadian Council of Human Resources Associations	2007
The Common Body of Knowledge for Computing and IT	Canada's Association of Information Technology Professionals	2012

There are some key differences between glossaries and bodies of knowledge which could affect the quality of a training corpus. The first relates to the amount of contextual information. Bodies of knowledge typically have descriptions for the various knowledge areas and units, and in some cases even the topics. Glossaries lack this type of context and are most often made up of short definitions. The second key difference is authorship. Bodies of knowledge are created by

professional and academic organizations such as the IEEE computer society which have vast expertise in their respective disciplines. As such, the BOKs are reviewed by many individuals and refined over time resulting in a higher quality product. Many glossaries do not undergo such a vigorous editing process. The overall quality of the glossaries information is potentially lower. We evaluated the effectiveness of training LSI and LDA with the compendium vs training with the bodies of knowledge to determine which training corpus was most effective.

One method of evaluating a training corpus is to look at the topics which are generated. Each topic is a collection of highly co-occurring words which should all be related in some way if the training corpus was effective. LSI topics are the most logical choice for this comparison because it generates topics sorted by prevalence. LDA's topics have no order whatsoever which prevents them from being able to be compared. We compared the top 5 LSI topics generated by the bodies of knowledge alone, the compendium alone, and with both combined.

Table 3-2: Top 5 LSI Topics

Rank	BOKs	Compendium	BOKs + Compendium
1	Perform, FTC, Mightily, Consult, Zoom	System, Software, Computer, Data, Information	System, Software, Computer, Data, Information
2	Mightily, FTC, Directory, AIF, Video	Software, IEC, ISO, IEEE, Engineering-Technology	Software, IEC, ISO, IEEE, Network
3	Perform, FTC, Zoom, AIF, Protocol	Network, Page, Web, File, Image	Network, Page, Web, File, Image
4	Directory, Fidelity, Expert, Perform, FTC	Computer, Program, Process, File, Information	Computer, Program, Process, File, Information
5	Expert, Perform, Network, AIF, Consult	Web, Page, Data, Internet, Computer	Web, Data, Page, Internet, Protocol

The poor topic quality of the topics from the bodies of knowledge alone was surprising considering all the points made previously. One reason for its inability to stand up as a training

corpus on its own is hypothesized to be due to its small size. Compared to the compendium, the body of knowledge collection has about ¼ the number of entries. The poor topic quality could also likely be due to the number of topics generated. For the size of the bodies of knowledge, our topic number of 400 may have been too high while being just right for the size of the compendium.

When evaluating the effectiveness of adding the bodies of knowledge to the compendium as a training corpus, the topics themselves did not give much indication of their effectiveness. Judging from the top 5 topics alone, it was difficult to tell if the compendium alone was the better training corpus than the compendium with the bodies of knowledge added. When both are used to perform semantic clustering and compared, they produce slightly different resulting lists.

Table 3-3: Top 5 Polysemous Terms with Different Training Corpora

RANK	WITHOUT BOKS	WITH BOKS
1	Data	Interface
2	Interface	Risk
3	**Workstation**	Object
4	Firewall	Function
5	Baseline	Firewall

Training with the compendium with the bodies of knowledge appeared to produce lists of more polysemous terms than training on the compendium alone. For example, the term "workstation" dropped in one of the LDA lists which allowed more polysemous terms such as "object" and "function" to rise to the top. Based on these results, we chose to train LSI and LDA on the compendium combined with the bodies of knowledge for our semantic clustering.

3.4 Linkage Type

The next variable was the linkage type, or the method for determining proximity between two clusters. There were two different types we chose to evaluate: complete and average linkages. The linkage type plays a very important role in determining the overall shape and density of the resulting clusters. Complete linkage compares the two furthest points of a cluster to determine distance which results in numerous dense clusters. Average linkage compares clusters using an average distance between each member. Intuitively both complete and average linkages seemed to be very effective and fair ways to evaluate cluster proximities, but it was unknown which would perform better.

3.5 Threshold

The final variable in these experiments was a tuning variable called the threshold. Normally, hierarchical clustering produces a complete dendrogram but our experiments require us to take a single slice of that dendrogram at a particular threshold value. The threshold value is a number between zero and one corresponding to a maximum distance. The clustering process repeatedly combines the two closest clusters until every cluster is more distant from each other than the threshold value. Higher threshold values will always produce fewer clusters as a result. We initially chose to obtain a candidate threshold value through what we call the crowdsourcing application.

3.5.1 Crowdsourcing Application

A candidate threshold would be the number where, after clustering is performed, most terms would have their text concepts semantically clustered “properly”. Since determining when text concepts are properly clustered is subjective, we initially chose to obtain a candidate

threshold through crowdsourcing. We constructed a web application which visually displayed clusters in a format similar to a dendrogram for a set of sample terms. The interface allowed users to manually adjust the threshold until the clusters appeared "right" to them. Collecting enough of these thresholds would enable us to use statistics to determine which threshold was optimal according to public consensus. This process began with choosing a suitable platform to build the web application.

3.5.2 Platform

The web application used for gathering data was built using the Google App Engine (GAE) platform. GAE is a service offered by Google which allows users to build web applications and subsequently host them on Google's servers. GAE was selected because the web application is hosted on their servers allowing users to use the web application from any device with an internet connection. GAE also allows developers to easily attach a database which was necessary in order to store the collected data.

3.5.3 Data Storage

The GAE platform gave us the freedom to choose from a variety of data stores. One was a standard SQL engine hosted by Google called Cloud SQL. A second was called Google Cloud Storage and is a file system hosted in the cloud. The third option was the GAE data store, which is a schema-less NoSQL database. The data being collected was very simple and required none of the benefits of relational SQL databases so we chose to use the GAE data store.

The data being stored consisted of two main parts: a survey portion and the clustering threshold portion. The answers to the survey questions were simply stored as text with no additional modification necessary. The clustering portion contained the current sample term and

six different thresholds that ranged from zero to one. A final piece of information that was stored was the completion date and time. Using the GAE data store, we were able to store each response as one row that contained all the information just described. Simple queries could be made after the data was gathered to try and find patterns much like regular SQL queries. The data store, like the application, was hosted by Google and could be accessed by any machine with an internet connection. With the platform and database established, we needed to determine which specific terms users would need to cluster to collect our sample thresholds.

3.5.4 Identifying Sample Terms

Due to the size of the corpus, experimental testing was performed on a subset of the compendium. The most efficient method for choosing a subset was to manually select a handful of terms. The semantic clustering can only be performed on terms with more than one text concept and of the 40,065 total terms, only about 25% had more than one associated text concept.

Sample terms needed to have a high number of concepts to precisely identify the optimal semantic clustering threshold. This is because terms with a low number of text concepts are less sensitive to changes in the threshold than concepts with high numbers. Aside from the number of associated text concepts, it was also important for sample terms to come from both sides of the polysemy scale to show that the threshold was globally applicable. There needed to be terms that were semantically simple with few different meanings as well as terms that were semantically complex with many different meanings. We subsequently selected 16 different terms to be part of the sample. We then designed and built the interface that would let users cluster the sample terms shown in table 3-4.

Table 3-4: Complete List of Sample Terms

ATM	Database	GIF	Software
Bandwidth	Download	HTML	Upload
Browser	Firewall	Internet	URL
Constraint	FTP	Process	Virus

3.5.5 Interface

When using the application, the first thing users saw was a series of simple questions. The purpose of these questions was to gain a little user background to add different dimensions to the resulting data. For example, gender-specific patterns could have played into how semantic clusters were evaluated. The data we chose to gather was gender, age range, and technical expertise.

Once the questions were answered, the user was presented with a term and six duplicate sets of its concepts. The six sets of concepts corresponded to the six combinations of similarity measures and linkage types. Below each set of concepts was a slider which corresponded to the threshold value identified and explained earlier in this thesis. For each set of text concepts, the user adjusted the threshold slider until the concepts were clustered the "best" according to their judgment. It was up to each user how to define this subjective measure. A couple additional features were added to assist the users in their clustering. The first was background coloring which ranged from green to red based on max intracluster distance. Each member within a cluster was compared with every other member within the same cluster, after which the most dissimilar members were identified and measured. Distances closer to 0 were greener indicating a tighter cluster whereas distances closer to 1 were redder indicating a looser cluster. The second feature utilized the domains from which each concept came from which were manually tagged when the glossaries were added into the compendium. Each concept was given a light dotted

border which had a color signifying the concept's particular domain that was displayed by hovering the cursor over the text concept.

After each set of concepts was a four point evaluation question which asked the user how well the clusters were grouped by meaning. This step was necessary because users were asked to find the point on the slider that was closest to how they would cluster the text concepts, but it did not measure how close that actually was. Some clusters could have perfectly aligned with the user's expectations while others at their best were still vastly different. This data would have been useful in identifying which similarity algorithm and linkage type combinations regularly aligned with users' expectations and which did not. The options were "barely", "somewhat", "mostly", and "exactly" corresponding to how well the clusters were semantically grouping the text concepts at the chosen threshold.

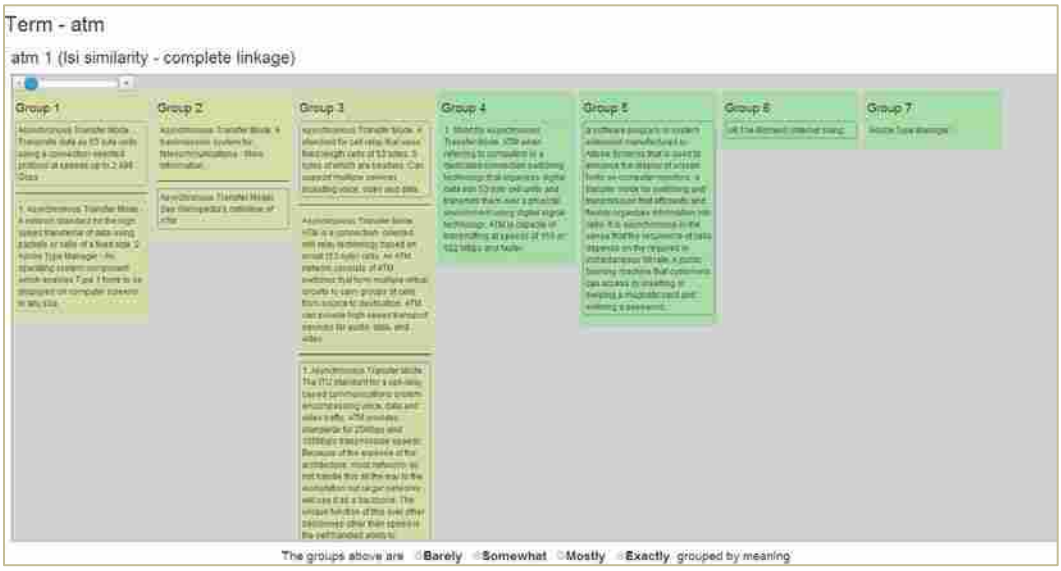

Figure 3-1: Semantic Clustering Interface

3.5.6 Interface Optimizations

While building the prototype data collection application, a few issues arose. It was clear that user fatigue was going to impede accurate data collection based on feedback from early testers. For each sample term, the tester would be required to adjust six different sliders while reading and organizing large amounts of text. Users would be adjusting 96 different bars in one sitting to evaluate all 16 of our sample terms. With so much required of users, data would rapidly begin losing value once users began getting mentally fatigued.

To address user fatigue, it was necessary to require less of each individual tester. Instead of asking each user to evaluate every sample term, we instead asked each user to evaluate a single one. Each of the sample terms needed to be evaluated multiple times for the results to be statistically significant so the tradeoff was more testers would be required. This updated approach required a tester to adjust only six bars instead of almost 100.

Another method to reduce the amount of user fatigue was to initialize the threshold sliders. If the thresholds started at a point which would be close to where most users would place it, they would then either agree with the groupings or make small adjustments in either direction. One option was to simply initialize every slider with some value predetermined through heuristic testing. Another was to take a mathematical approach and attempt to find some cluster optimization which would place the slider in roughly the same area as humans would. To create a more academically sound experiment, the latter option was investigated.

Each threshold produces some number of clusters which each have different characteristics, such as density or max width. It was theorized that these values could be measured and related to each threshold such that an optimization exists. The first cluster characteristic measured was max width, which was determined for each cluster by identifying

max distance between its members. Another cluster characteristic that was measured was a normalized max width which was generated by taking the number of cluster members divided by the max width of the cluster. These lists were then reduced to a variety of single values using different measures including sum, mean, root mean square, and median.

Using a python library called matplotlib, we were able to create graphs which had thresholds along the X axis and one of the calculated values we just discussed along the Y axis (Figure 3-2). Thresholds shown have been multiplied by 100 to avoid dealing with decimals so the thresholds range from 0 to 100. The goal was that some combination of these variables would create a curve which would have either a maximum or minimum in the last quartile of the possible threshold values, where we heuristically determined the candidate threshold would reside. All attempts at determining a mathematical optimization were initially unsuccessful as each attempt generated graphs that were either always increasing or always decreasing like the example shown in figure 3-2.

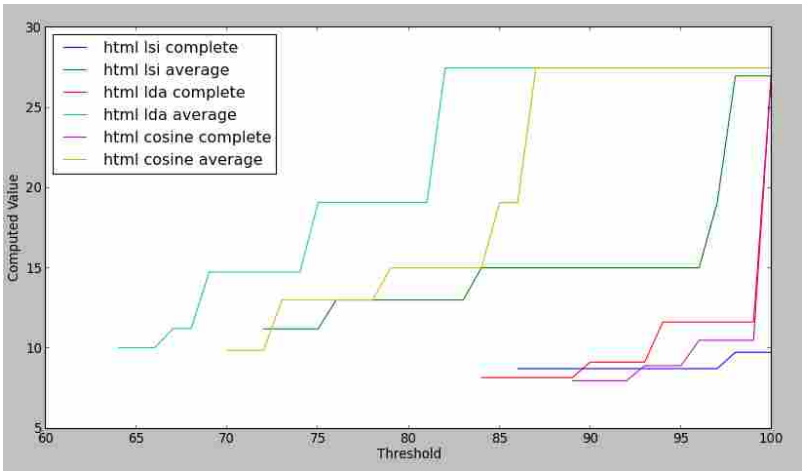

Figure 3-2: Sample Graph for Finding Cluster Optimizations

3.5.7 Crowdsourcing Outcomes

The first attempt at collecting data was to crowd-source, using the public to try and find ideal thresholds. As described in the previous section, a web application was constructed and subsequently published which gave anyone the opportunity to perform their own clustering. The goal was to obtain at least 200 responses which, when averaged together, would reveal a candidate threshold for each similarity algorithm and linkage type combination obtained through consensus.

We anxiously awaited for the data to start flooding in after broadcasting the URL for the web application throughout our social circles. After three weeks only 14 responses had been recorded, most of which originated with very close family members. The GAE platform had analytics information available which revealed that less than 10% of the users who opened the application actually completed it. In consultation with user experience experts, it became clear that the problem was too complex for this type of analysis (Helps 2014).

The core problem with the tool, we were informed, was it came off as overwhelming to the average user. The instructions that were required to give the user a clear picture of their task took up over half a page. If we further include the text that needed to be read in each of the six interactive windows, users were simply overwhelmed and ultimately decided against going further and completing the task. After further discussion, we determined that there would be no realistic way to refine the tool to mitigate this problem. Adding an incentive for users to use the application would increase the quantity of the results, but the complexity of the task would cause the quality of those results to deteriorate.

Despite this failure, the crowdsourcing tool's visual representation of the clusters illuminated a new potential measure of polysemy. Without valid crowdsourcing data, we had to find a replacement data source to determine our clustering threshold to proceed.

3.5.8 Convergence Values

Adjusting the thresholds of various terms in the tool revealed that terms converged into a single cluster at different thresholds. Terms we considered to be polysemous converged at higher thresholds than terms we considered to be less polysemous. This should have been obvious because more polysemous terms have more abstract concepts and would therefore require a higher threshold to converge. Looking back at the graphs we created when we attempted to initialize a starting threshold for the crowdsourcing tool, we once again noticed this pattern. The graphs shown in figure 3-3 show the general difference between less polysemous terms like "HTML" and more polysemous terms like "process" in relation to convergence. For each similarity measure and linkage type, we collected the threshold value when the last two clusters converged for every term, calling these numbers convergence values.

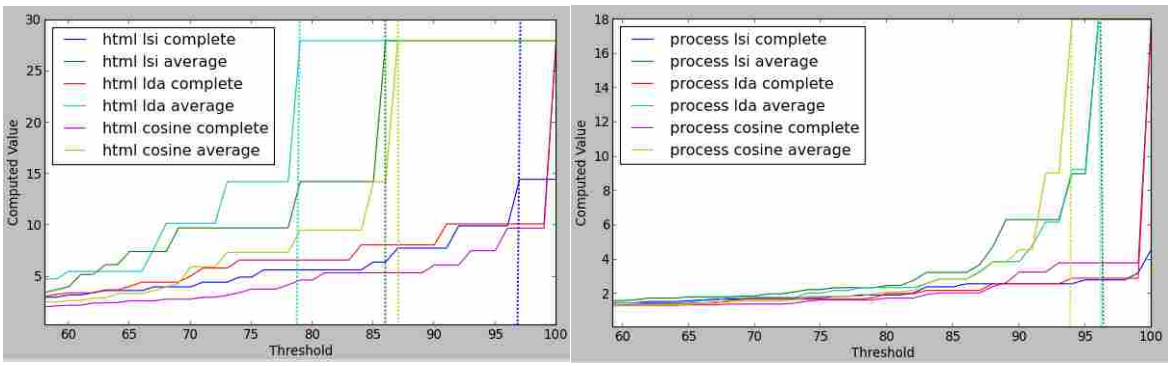

Figure 3-3: Convergences for HTML (left) and Process (right)

Graphing the convergence values for each combination of similarity measure and linkage type give us the graph in figure 3-4.

Regardless of the similarity measure or linkage type used, the overall pattern of the convergence values remains the same. Convergence values alone cannot be used to identify the most polysemous terms because the majority of the terms converge at the max threshold. The mean of the convergence values on the other hand is informative.

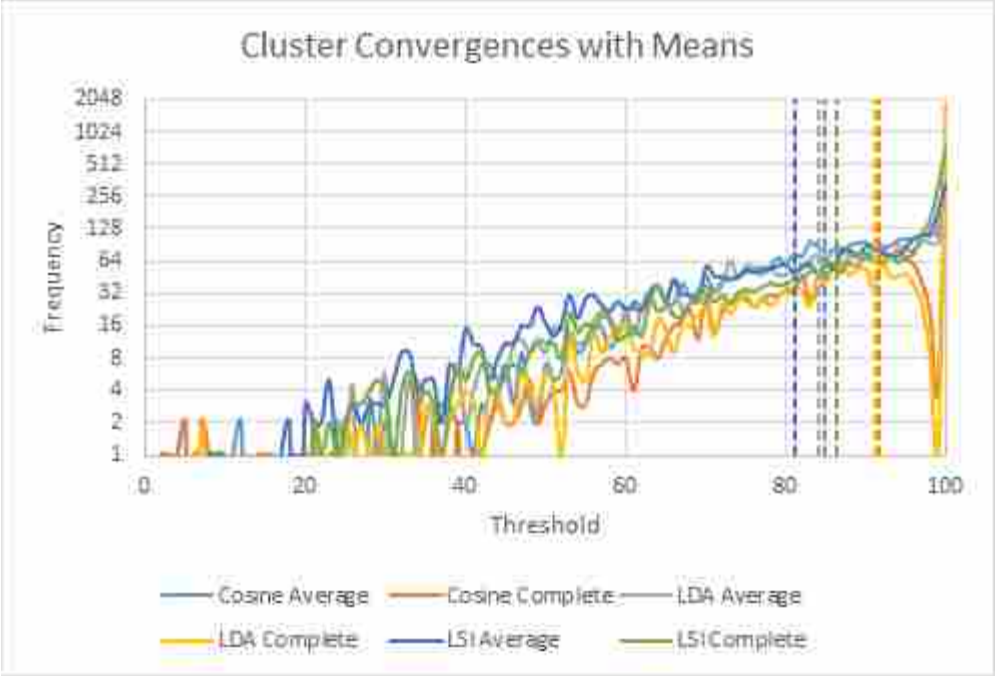

Figure 3-4: Convergence Values across the Compendium with Means

The mean convergence values identified in figure 3-4 all fall in the last quartile of possible thresholds, which was where we heuristically determined the candidate threshold would reside. To test the validity of the mean convergence value, we took our 16 sample terms from the crowd sourcing tool and performed semantic clustering to see if they would be sorted by polysemy.

Table 3-5: Top 10 Polysemous Sample Terms Lists

LDA		LSI		Cosine	
Average	**Complete**	**Average**	**Complete**	**Average**	**Complete**
Database	Bandwidth	Process	Software	Firewall	Firewall
Virus	Software	Virus	Firewall	Constraint	Bandwidth
Software	Process	Software	Process	Process	Virus
Firewall	Firewall	HTML	Virus	Virus	Software
Process	Database	Firewall	Database	Database	Process
GIF	Virus	Download	Bandwidth	GIF	Database
FTP	Download	GIF	Download	FTP	Constraint
Download	Constraint	FTP	HTML	Download	Browser
Constraint	Browser	Database	Browser	Bandwidth	Internet
Bandwidth	URL	Constraint	Internet	ATM	HTML

The different similarity measures and linkage types produced slightly different results, but in general it can be seen that the less polysemous sample terms like HTML and GIF tended to be ranked lower than the more polysemous terms like database and constraint. With enough evidence of success, we generated an ideal threshold for each similarity measure and linkage type combination by using the convergence values in place of the crowdsourcing data.

After calculating the arithmetic mean for each collection of convergences, we identified the following candidate thresholds based on the data found in the compendium shown in table 3-6.

Table 3-6: Mean Convergence Values

Similarity Measure / Linkage Type	*Mean Convergence Value*	*Rounded Convergence Value*
Cosine / Average	84.92	.85
Cosine / Complete	91.66	.92
LSI / Average	86.55	.87
LSI / Complete	87.71	.88
LDA / Average	84.43	.84
LDA / Complete	91.54	.92

Using the mean convergence values as the thresholds, we were subsequently able to perform semantic clustering for each similarity algorithm and linkage type combination to discover the most polysemous terms in the compendium.

4 CONTRIBUTION

4.1 Summary of Previous Findings

The process of producing a high quality list of polysemous terms led us to some valuable findings that we wish to reiterate.

When we researched similarity measures we identified two, LSI and LDA, which required a training corpus to establish models to compare texts. We looked at two main texts as potential training corpora: the compendium and BOKs. Both corpora were evaluated by looking at the top topics generated. Unexpectedly, topics generated by the compendium training corpus were far superior to those generated by the BOKs. Additionally, we analyzed the topics when both corpora were added together to determine if the BOKs could enhance the compendium during training. The resulting topics were too similar to the topics generated by the compendium alone to determine which corpus was superior. We subsequently performed semantic clustering using the compendium and then the compendium with the BOKs and compared the term lists. The compendium with the bodies of knowledge appeared to produce a slightly higher quality list according to our judgment. We chose to combine the bodies of knowledge with the compendium when training LSI and LDA. Our next findings were discovered when we tried to obtain a candidate threshold value required for hierarchical clustering.

We initially attempted to obtain, by a consensus, a candidate clustering threshold. Our crowdsourcing application was too complex for users, resulting in a completion rate of less than

10%. Despite this failure, the tool gave us a new perspective on cluster visualization, leading to our next finding. The tool, along with our attempt to find an initialized threshold to aid users, illuminated a new generic measure of term polysemy we call the convergence value. We observed that less polysemous terms tended to converge into a single cluster at lower thresholds than more polysemous terms. By using convergence values in place of the crowdsourcing thresholds, we could generate a single candidate threshold using the mean. Experimentation additionally showed that the mean convergence values for the compendium consistently fell in the last quartile of possible thresholds. The vast majority of the threshold values collected by the crowdsourcing tool also fell in the last quartile further indicating that the mean convergence values were an adequate substitute for the crowdsourcing data. With a candidate threshold identified, we were able to perform our semantic clustering process.

4.2 **Compendium Results**

Using the cosine, LSI, and LDA similarity measures along with the complete and average linkage types with their associated mean convergence value, we performed hierarchical clustering on each term in the compendium and sorted the results by cluster count. This produced six different lists of terms of which we chose to limit to the top 30 from each (table 4-1).

Evaluating the effectiveness of each combination of the variables is not an easy task due to the subjective nature of determining term polysemy. Each individual has different backgrounds and experiences meaning that each individual would have a slightly different order if asked to order the same terms by polysemy.

Table 4-1: Top 30 Potentially Confusing Terms

COSINE						LSI						LDA					
AVERAGE			COMPLETE			AVERAGE			COMPLETE			AVERAGE			COMPLETE		
Term	# Clusters	# Concepts	Term	# Clusters	# Concepts	Term	# Clusters	# Concepts	Term	# Clusters	# Concepts	Term	# Clusters	# Concepts	Term	# Clusters	# Concepts
Function	11	25	Interface	10	42	Interface	13	42	Interface	14	42	Interface	10	42	Interface	12	42
User	8	34	Firewall	10	67	Function	10	25	Function	11	25	Risk	9	30	Risk	10	30
Risk	8	30	User	9	34	Template	9	28	Bandwidth	11	64	Function	9	25	Object	10	32
Resource	8	12	Spam	9	53	Signature	9	31	User	10	34	Signature	8	31	Function	10	25
Process	8	36	Risk	9	30	Object	9	32	Risk	10	30	Scope	8	17	Firewall	10	67
Object	8	32	Function	9	25	Unit	8	13	Object	10	32	Policy	8	13	Template	9	28
Interface	8	42	Design	9	23	Stakeholder	8	19	Firewall	10	67	Object	8	32	Encryption	9	46
Entity	8	18	Data	9	41	Scope	8	17	Signature	9	31	Design	8	23	Baseline	9	42
Cc	8	13	Baseline	9	42	Process	8	36	Node	9	31	CC	8	13	Authentication	9	36
Case	8	15	Bandwidth	9	64	Feedback	8	15	Network	9	47	Baseline	8	42	User	8	34
Unit	7	13	Signature	8	31	Design	8	23	Header	9	24	AI	8	14	Signature	8	31
Template	7	28	Object	8	32	Constraint	8	21	Domain	9	32	User	7	34	Scope	8	17
Task	7	29	Node	8	31	CC	8	13	Design	9	23	Template	7	28	Path	8	20
State	7	12	Encryption	8	46	Baseline	8	42	Constraint	9	21	Standard	7	13	Design	8	23
Spoofing	7	20	Domain	8	32	Spoofing	7	20	Bot	9	21	Padding	7	10	Database	8	44
Signature	7	31	Cc	8	13	Risk	7	30	Baseline	9	42	Lol	7	11	Class	8	23
Set	7	10	Bot	8	21	Resource	7	12	Authentication	9	36	Firewall	7	67	CC	8	13
Scope	7	17	Virus	7	61	Queue	7	18	Stakeholder	8	19	Class	7	23	AI	8	14
Robot	7	15	Terminal	7	25	Pi	7	8	Scope	8	17	Worm	6	43	Worm	7	43
Project	7	16	Template	7	28	Parameter	7	13	Protocol	8	48	Tos	6	13	Terminal	7	25
Policy	7	13	Task	7	29	Lol	7	11	Process	8	36	Testing	6	17	Task	7	29
Non-Repudiation	7	15	System	7	20	Link	7	23	Gateway	8	30	Task	6	29	System	7	20
Feedback	7	15	Phishing	7	35	Header	7	24	Flash	8	30	System	6	20	Standard	7	13
Domain	7	32	Link	7	23	Error	7	15	Encryption	8	46	State	6	12	Spoofing	7	20
Design	7	23	Input	7	23	Entity	7	18	Data	8	41	Spoofing	6	20	Node	7	31
Data	7	41	Header	7	24	Domain	7	32	Cut	8	11	Simulation	6	21	Measure	7	17
Constraint	7	21	Feedback	7	15	Data	7	41	Client	8	37	Resource	6	12	Lol	7	11
Class	7	23	Entity	7	18	Cut	7	11	CC	8	13	Queue	6	18	Link	7	23
Bot	7	21	Database	7	44	Case	7	15	Thread	7	18	Process	6	36	Kilobyte	7	25
Authentication	7	36	Client	7	37	Bot	7	21	Terminal	7	25	Paradigm	6	7	Header	7	24

If we look further at the results, we can see that there are 10 terms which exist in all six lists. These 10 terms are all highly polysemous and are evidence of the success of using semantic clustering to identify term polysemy. Those particular terms are especially significant because, in a compendium containing over 40,000 terms, they made it into the top 30 most polysemous terms regardless of the similarity measure or linkage type used. Table 4-2 compares our initial metric of polysemy, simple concept count, to our new metric of cluster count using the terms found in every list.

Table 4-2: Top 10 Polysemous Terms Alphabetized Before and After

Old Metric Concept Count	New Metric Semantic Clustering
Bandwidth	Design
Browser	Function
Download	Interface
Firewall	Object
GIF	Risk
HTML	Signature
Internet	System
Software	Task
URL	Template
Virus	User

Our new metric using semantic clustering has clearly produced a list of highly polysemous terms. The less polysemous terms that previously rose to the top of our polysemy lists from their high text concept count, like HTML and GIF, dropped dramatically. The clustering process was able to successfully combine semantically duplicate text concepts so that a more accurate count of each term's different abstract concepts could be obtained.

5 CONCLUSIONS AND FUTURE WORK

5.1 Overview

Vocabularies within and between disciplines have been rapidly evolving, making identifying potentially confusing words and phrases an important task. We wanted to identify the most polysemous terms being used today using a collection of 399 glossaries. Polysemy of a term was measured by semantically clustering its associated text concepts and counting the resulting number of clusters. Terms with one text concept were not included. As part of the semantic clustering process, we compared three similarity measures consisting of cosine similarity, latent semantic indexing, and latent Dirichlet allocation. Two hierarchical clustering linkage types were also compared: complete and average linkages. A third clustering variable, threshold, was calculated using the mean convergence value, or the mean value at which a term's text concepts are clustered into a single cluster across the entire compendium. We produced a list of terms sorted by polysemy for each of the six combinations of similarity measures and linkage types.

5.2 Evaluation of Results

Each list that was produced using our semantic clustering was effective. The results are so effective that no combination of similarity measure and linkage type can clearly be deemed the best. It is our conclusion that, in the absence of clear evidence supporting which similarity

measure and clustering combination performed the best, our best list of the most polysemous terms was created by finding the terms present in every one of the six lists. We created that final list of polysemous terms that can be found earlier in table 4-2. In a compendium of over 40,000 terms, it is highly significant that the 10 terms in our final list appeared as highly polysemous regardless of the similarity measure or linkage type used.

We consider the semantic clustering successful based on the intuitive sense of polysemy in the terms that were displayed in the lists. Every method produced few clusters for the common, less polysemous terms like "HTML" and produced more clusters for the more polysemous terms like "object".

Another result of our research was the realization that there is no ideal clustering threshold that accurately clusters text concepts semantically for every term. The number of clusters depicted in table 4-1 do not directly indicate the number of real meanings the associated term contains. The thresholds used for the semantic clustering are designed to create cluster quantities that, when compared with other terms in the same experiment, will give the terms a sense of rank. The crowdsourcing tool demonstrated that each term has a different ideal threshold that accurately clusters its text concepts by meaning. This means that while we can identify the terms that are polysemous, we do not have a clear method of automatically determining the exact number of different meanings (abstract concepts) associated with each term.

5.3 **Future Work**

Many additional text similarity measures exist and could be evaluated for their effectiveness in semantic clustering. Additional testing is also necessary to determine the effect of using different thresholds than the mean convergence value when clustering the compendium.

It is currently unknown exactly how the threshold value effects the resulting lists of polysemous terms.

A stronger method for evaluating the results of semantic clustering is necessary to accurately identify the best similarity algorithm and linkage type combination. The subjective nature of linguistic problems like term polysemy makes it difficult to objectively rank the different results. Creating some method for doing so would allow us to evaluate each similarity measure and linkage type and conclusively determine which combination produces the most accurate results.

5.4 **Closing**

The idea behind semantic clustering being used to evaluate term polysemy has solid ground. We have shown that semantic clustering can be used on glossaries to measure the degree of polysemy in terms with high accuracy. We generated six lists of highly polysemous terms and were able to create a single list showing the shared polysemous terms between them all. We were successful in identifying terms that people would deem as highly polysemous which would have a very high potential of causing confusion in interdisciplinary communication.

REFERENCES

Blei, D, A Ng, and M Jordan. "Latent Dirichlet Allocation." the Journal of machine Learning research (2003). http://dl.acm.org/citation.cfm?id=944937 (accessed November 22, 2013).

Bradford, R.B. "An Empirical Study of Required Dimensionality for Large-Scale Latent Semantic Indexing Applications." *Proceeding of the 17th ACM conference on Information and knowledge mining - CIKM '08* (2008): 153. http://portal.acm.org/citation.cfm?doid=1458082.1458105.

Croft, W, D Metzler, and T Strohman. Search engines: Information retrieval in practice. Reading: Addison-Wesley, 2010.

Deerwester, S, S Dumais, and T Landauer. "Indexing by Latent Semantic Analysis." JASIS (1990). http://www.cob.unt.edu/itds/faculty/evangelopoulos/dsci5910/LSA_Deerwester1990.pdf (accessed November 20, 2013).

Ekstrom, J. "Experience with a Cross-Disciplinary Aggregated Glossary of Technical Terms." Proceedings of the 13th annual conference on … (2012). http://dl.acm.org/citation.cfm?id=2380562 (accessed November 12, 2013).

Ekstrom, J, and B Lunt. "Academic IT and Adjacent Disciplines 2010." Proceedings of the 2010 ACM conference on … (2010). http://dl.acm.org/citation.cfm?id=1867653 (accessed March 25, 2013).

El-Hamdouchi, A, and P Willett. "Comparison of Hierarchic Agglomerative Clustering Methods for Document Retrieval." The Computer Journal (1989). http://medcontent.metapress.com/index/A65RM03P4874243N.pdf (accessed December 2, 2013).

Hatzivassiloglou, V, J Klavans, and E Eskin. "Detecting Text Similarity over Short Passages: Exploring Linguistic Feature Combinations via Machine Learning." Proceedings of the 1999 joint … (1999). http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.32.9798&rep=rep1&type=pdf (accessed October 1, 2013).

Helps, R. Interview with Owen Riley. Personal interview. BYU, February 26th, 2014.

Huang, A. “Similarity Measures for Text Document Clustering.” Proceedings of the Sixth New Zealand Computer …, no. April (2008). http://favi.com.vn/wp-content/uploads/2012/05/pg049_Similarity_Measures_for_Text_Document_Clustering.pdf (accessed November 20, 2013).

IEEE. “Using SEVOCAB” SEVocab. http://pascal.computer.org/sev_display/help.jsp (accessed February 10, 2014) (2010).

Knuth, D. “The Art of Computer Programming: Fundamental Algorithms, Vol. 1.” (1968)

Metzler, D, S Dumais, and C Meek. “Similarity Measures for Short Segments of Text.” Advances in Information Retrieval (2007). http://link.springer.com/chapter/10.1007/978-3-540-71496-5_5 (accessed October 1, 2013).

Porter, M. *The Porter Stemming Algorithm.* 2001. http://tartarus.org/martin/PorterStemmer/index-old.html.

Richards, J, O Riley, J Ekstrom, and K Tew. “Termediator: Early Studies in Terminological Mediation Between Disciplines.” Proceedings of the 2013 ACM conference on Information technology research (2013).

Salton, G, A Wong, and C Yang. “A Vector Space Model for Automatic Indexing.” Communications of the ACM 18, no. 11 (1975). http://dl.acm.org/citation.cfm?id=361220 (accessed November 22, 2013).

Tryon, R. *Cluster Analysis*. Ann Arbor: Edwards Brothers, Inc, 1939. (accessed November 12, 2013).

Van Rijsbergen, C., S Robertson, and M Porter.*New models in probabilistic information retrieval*. Computer Laboratory, University of Cambridge, 1980.

Willett, P. “Recent Trends in Hierarchic Document Clustering: A Critical Review.” *Information Processing & Management* 24, no. 5 (January 1988): 577–597. Accessed March 12, 2014. http://linkinghub.elsevier.com/retrieve/pii/0306457388900271.

APPENDIX A. LIST OF STOPWORDS

When transforming the text during the glossary aggregation, the following words are removed as they generally detract from the semantic meaning of the text:

a	alone	anent
about	along	another
above	alongside	any
across	already	anybody
afore	also	anyone
aforesaid	although	anything
after	always	are
again	am	aren't
against	american	around
agin	amid	as
ago	amidst	aslant
aint	among	astride
albeit	amongst	at
all	an	athwart
almost	and	away

b
back
bar
barring
be
because
been
before
behind
being
below
beneath
beside
besides
best
better
between
betwixt
beyond
both
but
by
c
can
cannot
can't
certain
circa
close
concerning
considering
cos
could
couldn't
couldst
d
dare
dared
daren't
dares
daring
despite
did
didn't
different
directly
do
does
doesn't
doing
done
don't
dost
doth
down
during
durst
e
each
early
either
em
english
enough
ere
even
ever
every
everybody

everyone
everything
except
excepting
f
failing
far
few
first
five
following
for
four
from
g
gonna
gotta
h
had
hadn't
hard
has
hasn't
hast
hath
have
haven't
having
he
he'd
he'll
her
here
here's
hers
herself
he's
high
him
himself
his
home
how
howbeit
however
how's
i
id
if
ill
i'm
immediately
important
in
inside
instantly
into
is
isn't
it
it'll
it's
its
itself
i've
j
just
k
l

large
last
later
least
left
less
lest
let's
like
likewise
little
living
long
m
many
may
mayn't
me
mid
midst
might
mightn't
mine
minus
more
most
much
must
mustn't
my
myself
n
near
'neath
need
needed
needing
needn't
needs
neither
never
nevertheless
new
next
nigh
nigher
nighest
nisi
no
no-one
nobody
none
nor
not
nothing
notwithstanding
now
o
o'er
of
off
often
on
once
one
oneself
only
onto
open

or
other
otherwise
ought
oughtn't
our
ours
ourselves
out
outside
over
own
p
past
pending
per
perhaps
plus
possible
present
probably
provided
providing
public
q
qua
quite
r
rather
re
real
really
respecting
right
round
s
same
sans
save
saving
second
several
shall
shalt
shan't
she
shed
shell
she's
short
should
shouldn't
since
six
small
so
some
somebody
someone
something
sometimes
soon
special
still
such
summat
supposing
sure
t

than
that
that'd
that'll
that's
the
thee
their
theirs
their's
them
themselves
then
there
there's
these
they
they'd
they'll
they're
they've
thine
this
tho
those
thou
though
three
thro'
through
throughout
thru
thyself
till
to
today
together
too
touching
toward
towards
true
'twas
'tween
'twere
'twill
'twixt
two
'twould
u
under
underneath
unless
unlike
until
unto
up
upon
us
used
usually
v
versus
very
via
vice
vis-a-vis
w
wanna

wanting
was
wasn't
way
we
we'd
well
were
weren't
wert
we've
what
whatever
what'll
what's
when
whencesoever
whenever
when's
whereas
where's
whether
which
whichever
whichsoever
while
whilst
who
who'd
whoever
whole
who'll
whom
whore
who's
whose
whoso
whosoever
will
with
within
without
wont
would
wouldn't
wouldst
x
y
ye
yet
you
you'd
you'll
your
you're
yours
yourself
yourselves
you've
z

APPENDIX B. CROWDSOURCING TOOL CODE

app.yaml

```
application: glass-radar-428
version: 1
runtime: python27
api_version: 1
threadsafe: true

libraries:
- name: numpy
  version: "latest"

handlers:
- url: /static
  static_dir: static

- url: /.*
  script: thesisapp.application
```

build_terms.py

```
""" Execute this script to build cache.json, the cache file used
for the semantic clustering application """

import scipy
import os
import porter
import re
import numpy
from lxml import etree
from gensim import corpora, models, similarities
import json
from scipy.cluster.hierarchy import linkage, inconsistent,
fcluster, maxdists, dendrogram, cophenet
```

```
import sem_cluster
import itertools
from math import ceil, sqrt
import matplotlib.pyplot as plt
import csv

terms = sem_cluster.terms
sim_algorithms = sem_cluster.sim_algorithms
linkage_types = sem_cluster.linkage_types

def get_sim_matrix(sim_algorithm):
    """ Takes a similarity algorithm and returns a condensed
similarity matrix with every term """

    # Check if the matrix already exists. If so, load it.
Otherwise build a new one
    try:
        matrix =
similarities.MatrixSimilarity.load(sim_algorithm + ".index")
    except IOError:
        stoplist = [w.strip() for w in open('../stopwords.txt',
'r').readlines()]
        splitter = re.compile ( "[a-z\-']+", re.I )
        stemmer = porter.PorterStemmer()
        glossary = etree.parse("../glossary.xml")
        source = []
        glossary_source = []
        for subdir, dirs, files in os.walk("../DatabaseFiles"):
            for file in files:
                filename = subdir+'/'+file
                bok = etree.parse(filename)
                for node in bok.iter():
                    if node.text and (node.tag == "name" or
node.tag == "text" or node.tag == "learningOutcome"):

source.append(node.text.encode('ascii','ignore').strip().lower()
)
        for entry in glossary.findall("Entry"):
            for concept in entry.findall("Concept"):

glossary_source.append(concept.text.encode('ascii','ignore').str
ip().lower())

        texts = [[stemmer.stem(word, 0, len(word)-1) for word in
splitter.findall(document) if word not in stoplist]
                 for document in source]
        clean_texts = [text for text in texts if text]
```

```
        dictionary = corpora.Dictionary(clean_texts)
        corpus = [dictionary.doc2bow(text) for text in
clean_texts]

        glossary_texts = [[stemmer.stem(word, 0, len(word)-1)
for word in splitter.findall(document) if word not in stoplist]
                  for document in glossary_source]
        dictionary = corpora.Dictionary(glossary_texts)
        glossary_corpus = [dictionary.doc2bow(text) for text in
glossary_texts]

        if sim_algorithm == "lsi":
            tfidf = models.TfidfModel(glossary_corpus + corpus)
            corpus_tfidf = tfidf[glossary_corpus + corpus]
            lsi = models.LsiModel(corpus_tfidf,
id2word=dictionary, num_topics=400)
            lsi.save('lsi.model')
            matrix =
similarities.MatrixSimilarity(lsi[glossary_corpus])
            matrix.save('lsi.index')
        elif sim_algorithm == "lda":
            lda = models.LdaModel(corpus + glossary_corpus,
id2word=dictionary, num_topics=400)
            lda.save('lda.model')
            matrix =
similarities.MatrixSimilarity(lda[glossary_corpus])
            matrix.save('lda.index')
        else:
            raise

    return matrix

def get_term_sim_list(term, sim_algorithm):
    """ Returns a condensed proximity matrix specific for the
term and sim_algorithn """

    # Load the term_sim_list cache if it exists
    try:
        with open(sim_algorithm + "_term_dict.json") as f:
            term_dict = json.load(f)
    except:
        term_dict = {}

    # If the term_sim_list is in the cache, load it. Otherwise
build a new term_sim_list
    if term in term_dict:
        term_sim_list = term_dict[term]
```

```
    else:
        glossary = etree.parse("../glossary.xml")
        if sim_algorithm == "cosine":
            stoplist = [w.strip() for w in
open('../stopwords.txt', 'r').readlines()]
            splitter = re.compile ( "[a-z\-']+", re.I )
            stemmer = porter.PorterStemmer()

            source = []
            for entry_obj in glossary.findall("Entry"):
                cur_term = entry_obj.find("Term").text.lower()
                if cur_term == term:
                    for concept_obj in
entry_obj.findall("Concept"):
                        source.append(concept_obj.text)
            texts = [[stemmer.stem(word.lower(), 0, len(word)-1)
for word in splitter.findall(document) if word not in stoplist]
                     for document in source]
            dictionary = corpora.Dictionary(texts)
            corpus = [dictionary.doc2bow(text) for text in
texts]
            cosine = models.TfidfModel(corpus)
            matrix = similarities.MatrixSimilarity(corpus)
        else:
            matrix = get_sim_matrix(sim_algorithm)
        valid_ids = []
        term_sim_list = []
        total_num = 0
        source = []
        term_concept_dict = {}
        try:
            with open("term_concept_dict.json") as f:
                term_concept_dict = json.load(f)
        except:
            term_concept_dict = {}
        for num, entry_obj in
enumerate(glossary.findall("Entry")):
            term_name = entry_obj.find("Term").text.lower()
            for concept_num, concept_obj in
enumerate(entry_obj.findall("Concept")):
                if term == term_name:
                    valid_ids.append(total_num)
                    source.append(concept_obj.text)
                total_num += 1
        term_concept_dict[term] = source
        with open("term_concept_dict.json", "w") as f:
            json.dump(term_concept_dict, f)
```

```
        id_len = len(valid_ids) - 1
        result_num = 0
        while id_len > 0:
            result_num += id_len
            id_len -= 1
        if sim_algorithm == "cosine":
            valid_ids = range(len(valid_ids))
        min_id = min(valid_ids)
        max_id = max(valid_ids)
        for id_num in valid_ids:
            if id_num == max_id:
                break
            vec = matrix.index[id_num]
            for x in matrix[vec][id_num+1:max_id+1]:
                val = 1-x
                if val < 0:
                    val = 0
                term_sim_list.append(val)
        term_dict[term] = term_sim_list

        # Save the newly created term_sim_list in the cache
        with open(sim_algorithm + "_term_dict.json", "w") as f:
            json.dump(term_dict, f)

    return term_sim_list


def cluster(X, t, method):
    """ Slightly modifies the default SciPy hierarchical
clustering function """
    Z = linkage(X, method=method)
    R = inconsistent(Z, d=2)
    T = fcluster(Z, criterion="distance", depth=2, R=R, t=t)

    return T

def getClusterDistances(clusterList, term_sim_list):
    """ Takes a list of clusters along with the proximity matrix
and returns the max intracluster distance for each cluster as a
list """
    distances = []
    val = 0
    n = 0
    while val != len(term_sim_list):
        n += 1
        val = n*(n-1)/2
    combo = list(itertools.combinations(range(n), 2))
```

```
    for x in range(max(clusterList)):
        indices = [i for i, cluster in enumerate(clusterList) if
cluster == x+1]
        tempDistances = [0]
        for y in list(itertools.combinations(indices, 2)):
            for index, z in enumerate(combo):
                if y == z:
                    tempDistances.append(term_sim_list[index])
                    break
        num = max(tempDistances)
        dist = ceil(num * 1000) / 1000.0
        if dist > 1:
            dist = 1
        distances.append(dist)

    return distances

def printTopNConfusingTerms(cache, n):
    """ Evaluates the cache and produces a list of the top n
polysemous terms for each similarity measure and linkage type
"""

    data_store = {}
    for term in cache:
        for sim_algorithm in sim_algorithms:
            sim_store = data_store.setdefault(sim_algorithm, {})
            for linkage_type in linkage_types:
                link_store = sim_store.setdefault(linkage_type,
[])
                cool =
cache[term]["data"][sim_algorithm][linkage_type]
                temp_thres = 0
                temp_dia = []
                for threshold in range(100):
                    cur_thres = str(threshold)
                    cur_dia = cool[cur_thres][0]
                    if cur_dia != temp_dia:
                        temp_dia = cur_dia
                        temp_thres = cur_thres
                link_store.append(temp_thres)
    glossary = etree.parse("../glossary.xml")
    term_concept_count = {}
    for entry in glossary.findall("Entry"):
        term = entry.find("Term").text.lower()
        term_concept_count[term] = len(entry.findall("Concept"))
    for sim in data_store:
        for link in data_store[sim]:
```

```
            threshold = str(int(round(numpy.mean(map(int,
data_store[sim][link])))))
            results = []
            for term in cache:
                concept_count = term_concept_count[term]

results.append((max(cache[term]["data"][sim][link][threshold][0]
), term, concept_count))
            results = sorted(results, reverse=True)
            print sim, link
            counter = 0
            for result in results:
                print '\t',result[1],'-', result[0],'clusters -
', result[2],'concepts'
                counter += 1
                if counter == n:
                    break
            print '\n'


def experimentalData(cache, show=False):
    """ Test function for building graphs when seeking to
initialize threshold sliders in the crowdsourcing application
"""

    datapointStringList = ["MaxClusterWidths",
"NormalizedClusterDensity"]
    valueStringList = ["Median","Mean","Sum","Root-mean-
squared","Variance"]

    test = {}
    for sim_algorithm in sim_algorithms:
        test[sim_algorithm] = {}
        for linkage_type in linkage_types:
            test[sim_algorithm][linkage_type] = [[] for i in
range(len(datapointStringList)*len(valueStringList))]

    for term in cache:
        for sim_algorithm in sim_algorithms:
            for linkage_type in linkage_types:
                vals = []
                thres = []
                data = [[] for i in
range(len(datapointStringList)*len(valueStringList))]
                for threshold in range(0, 101, 1):
```

```
                    clusterList =
cache[term]["data"][sim_algorithm][linkage_type][str(threshold)]
[0]
                    clusterDiameters =
cache[term]["data"][sim_algorithm][linkage_type][str(threshold)]
[1]

                    datapointsList = []

                    #This method uses a list of max cluster
widths
                    datapointsList.append(clusterDiameters)

                    #This method takes the number of cluster
members / max width of cluster + 1

datapointsList.append([clusterList.count(x+1)/((clusterDiameters
[x] + 1)) for x in range(max(clusterList))])

                    for ind, datapoints in
enumerate(datapointsList):
                        #This method takes the median
                        data[len(valueStringList)*ind +
0].append(numpy.median(datapoints))

                        #This method takes the mean
                        data[len(valueStringList)*ind +
1].append(numpy.mean(datapoints))

                        #This method takes the sum
                        data[len(valueStringList)*ind +
2].append(sum(datapoints))

                        #This method takes the root mean squared
                        data[len(valueStringList)*ind +
3].append(sqrt(sum(result ** 2 for result in
datapoints)/len(datapoints)))

                        #This method takes the variance
                        data[len(valueStringList)*ind +
4].append(numpy.var(datapoints))

                    thres.append(threshold)
                for index, values in enumerate(data):
                    zippedList = zip(values, thres)
```

```
                    if len(zippedList) > 0:
test[sim_algorithm][linkage_type][index].append(max(zippedList,k
ey=lambda item:item[0])[1])
                    dataindex, valueindex = (index /
len(valueStringList), index % len(valueStringList))
                    if datapointStringList[dataindex] ==
"NormalizedClusterDensity" and valueStringList[valueindex] ==
"Mean":
                        plt.plot(thres, values,'',
label=""+term+" " +sim_algorithm+" "+linkage_type)
#                 plt.plot(thres, data[0],'', label=""+term+" "
+sim_algorithm+" "+linkage_type)
        plt.ylabel("Computed Value")
        plt.xlabel("Threshold")
        plt.legend(loc=2)
        if show:
            plt.show()

    for sim_algorithm in test:
        print sim_algorithm
        for linkage_type in test[sim_algorithm]:
            print '\t',linkage_type
            for index, values in
enumerate(test[sim_algorithm][linkage_type]):
                dataindex, valueindex = (index /
len(valueStringList), index % len(valueStringList))
                mean = numpy.mean(values)
                stdev = numpy.std(values)
                if stdev <= 10 and mean >= 60 and mean <= 95:
                    print '\t\t', mean, stdev,
datapointStringList[dataindex], valueStringList[valueindex]
                    print '\t\t\t', values

if __name__ == '__main__':
    """ Builds the cache file required for clustering
application """
    from pprint import pprint
    stoplist = [w.strip() for w in open('../stopwords.txt',
'r').readlines()]
    splitter = re.compile ( "[a-z\-']+", re.I )
    stemmer = porter.PorterStemmer()

    try:
        with open("cache.json") as f:
            cache = json.load(f)
    except:
        cache = {}
```

```
    source = []

    glossary = etree.parse("../glossary.xml")

    terms = []
    for entry in glossary.findall("Entry"):
        if len(entry.findall("Concept")) > 3:
            terms.append(entry.find("Term").text.lower())

    issue = False
    data_store = {}
    for term in terms:
        if issue:
            print "\tIssue Detected"
            issue = False
        print "Starting", term, "..."
        current_entry = ''
        for entry in glossary.findall("Entry"):
            if term == entry.find("Term").text.lower():
                current_entry = entry
                break
        domains = []
        concepts = []
        for concept in current_entry.findall("Concept"):
            concepts.append(concept.text)
            source = concept.find("ConceptAnnotation").text

domains.append(glossary.find("GlossaryRef[@id='"+source+"']").fi
nd("OriginDomain").text.replace(" ",""))
        for sim_algorithm in sim_algorithms:
            sim_store = data_store.setdefault(sim_algorithm, {})
            term_sim_list = get_term_sim_list(term.lower(),
sim_algorithm)
            for linkage_type in linkage_types:
                link_store = sim_store.setdefault(linkage_type,
[])
                temp_dia = []
                temp_thres = 0
                for threshold in range(0, 101, 1):
                    try:
                        clusters = cluster(term_sim_list,
threshold/float(100), linkage_type)
                        clusterList = clusters.tolist()
                    except UnboundLocalError:
                        issue = True
                        clusters = [1]
                        clusterList = clusters
```

```
                    term_cache = cache.setdefault(term, {})
                    domain_cache =
term_cache.setdefault("domains", domains)
                    concept_cache =
term_cache.setdefault("concepts", concepts)
                    data_cache = term_cache.setdefault("data",
{})
                    sim_algorithm_cache =
data_cache.setdefault(sim_algorithm, {})
                    linkage_type_cache =
sim_algorithm_cache.setdefault(linkage_type, {})
                    clusterDiameters =
getClusterDistances(clusterList, term_sim_list)
                    cur_thres = str(threshold)
                    if clusterDiameters != temp_dia:
                        temp_dia = clusterDiameters
                        temp_thres = cur_thres

                    linkage_type_cache[threshold] =
(clusterList, clusterDiameters)

#                    Z = linkage(term_sim_list,
method=linkage_type)
#                    dendrogram(Z)
#                   plt.show()
#                if temp_thres != "0" and temp_thres != "1":
#                    link_store.append(temp_thres)
#    for sim in data_store:
#        print sim
#        for lin in data_store[sim]:
#            data = map(int, data_store[sim][lin])
#            print '\t',lin
#            print '\t\t', numpy.mean(data)
#            with open(sim + '_' + lin + '.csv', 'w') as f:
#                writer = csv.writer(f)
#                writer.writerows([data])
    with open("cache.json", "w") as f:
        json.dump(cache, f)
```

sem_cluster.py

```
""" Contains helper functions for thesisapp.py to use in AJAX
calls """

import json
import numpy
```

```
import math

terms = ["html", "constraint", "process", "database", "ftp",
"atm", "virus", "bandwidth", "firewall", "browser", "software",
"internet", "url", "gif", "download", "upload"]
sim_algorithms = ["lsi", "lda", "cosine"]
linkage_types = ["complete", "average"]

def get_clusters(term, sim_algorithm, linkage_type, threshold):
    text = ''
    with open("term_concept_dict.json") as f:
        term_concept_dict = json.load(f)
    source = term_concept_dict[term]
    with open("cache.json") as f:
        cache = json.load(f)
    clusters =
cache[term]["data"][sim_algorithm][linkage_type][str(int(float(t
hreshold)))][0]
    domains = cache[term]["domains"]
    clusterWidths =
cache[term]["data"][sim_algorithm][linkage_type][str(int(float(t
hreshold)))][1]
    clusterColors = get_cluster_colors(clusterWidths)
    clusters = numpy.array(clusters)
    for x in range(max(clusters)):
        clusterFrequency = clusters.tolist().count(x+1)
        clusterWidth = clusterWidths[x]
        text += "<div class='cluster' style='background-
color:rgba(" + clusterColors[x] + ",.2)'><h4>Group " + str(x+1)
+ '</h4>'
        for y in range(clusterFrequency):
            domain =
domains[numpy.flatnonzero(clusters==x+1)[y]]
            text += "<p class='"+ domain + "' title='" + domain
+ "'>" + source[numpy.flatnonzero(clusters==x+1)[y]] +
'</p><hr>'
        #remove trailing horizontal rule
        text = text[:-4]
        text += '</div>'
    text += '<div class="clear"></div>'

    return text


if __name__ == '__main__':
    get_clusters("html", "lsi", "complete", float(50)/100)
```

thesisapp.py

```
""" The main script for the Crowdsourcing Application """
import webapp2
import sem_cluster
import json
from google.appengine.ext import ndb

class ClustererResponse(ndb.Model):
    """Models a clusterer's response with content and date."""
    gender = ndb.StringProperty()
    age = ndb.StringProperty()
    techy = ndb.StringProperty()
    term = ndb.StringProperty()
    lsi_complete = ndb.IntegerProperty()
    lsi_complete_rating = ndb.IntegerProperty()
    lsi_average = ndb.IntegerProperty()
    lsi_average_rating = ndb.IntegerProperty()
    lda_complete = ndb.IntegerProperty()
    lda_complete_rating = ndb.IntegerProperty()
    lda_average = ndb.IntegerProperty()
    lda_average_rating = ndb.IntegerProperty()
    cosine_complete = ndb.IntegerProperty()
    cosine_complete_rating = ndb.IntegerProperty()
    cosine_average = ndb.IntegerProperty()
    cosine_average_rating = ndb.IntegerProperty()
    date = ndb.DateTimeProperty(auto_now_add=True)

class MainPage(webapp2.RequestHandler):

    def get(self):
        html = ""
        html += '<script src="static/jquery.js"></script>'
        html += '<script src="static/script.js"></script>'
        html += '<script src="static/bootstrap-
slider.js"></script>'
        html += '<link rel="stylesheet"
href="static/style.css">'
        html += '<link rel="stylesheet"
href="static/bootstrap.css">'
        html += '<link rel="stylesheet"
href="static/slider.css">'
        html += '<h1><b>Instructions:</b></h1>'
        html += '<p class="instructionParagraph">The first
section is a simple survey designed to gather general data about
your background. No personally identifiable information will be
collected so please answer as accurately as you can.'
```

```
        html += '<p class="instructionParagraph">The following
section consists of first choosing a term. Six different
interactive windows will then appear beneath. Each term has a
number of different definitions which have been found and
gathered from all over the internet. For each interactive
window, your task is to drag the slider (or click the + and -
buttons) found in the top left until the definitions are grouped
together by <em>meaning</em> as best as you can. As the slider
moves from left to right, the definitions will begin to group
together more and more until eventually (when the slider is all
the way to the right) they will be in a single group. The total
number of definitions never changes, only their groupings with
eachother. It may be easier to start the slider all the way to
the right and then slowly move it left until an ideal set of
groups is formed. This will need to be done six times for the
chosen term in order to evaluate different test variables in the
experiment. The groupings will most likely not be perfect, so
try to find the spot on the slider where it is the CLOSEST to
how you would group them if you were told to group the
definitions manually by <em>meaning</em>. After each slider you
will find a simple question asking you to rate the resulting
groups by how well they group concepts by meaning.</p>'
        html += '<p class="instructionParagraph">The background
color of each group will change from green to red according to
definition density. In other words, greener groups are more
closely related (according to the computer) while redder groups
are less closely related. While it is intended to be an aid to
your task, you may ignore it when finding your ideal groups.
Each definition also has a dotted colored border which
identifies the domain from which the definition was retrieved.
Hovering over a definition will reveal this domain if so
desired.</p>'
        html += '<p class="instructionParagraph">After you
complete the exercise, click the submit button at the bottom of
the page. If you want, you can then hit the back button on your
browser and select a different term and repeat the exercise.
Either way, thank you so much for your help with this and have
fun!'

        html += '<h3 style="float:right"> - Owen</h3><br><br>'
        html += '<h1>Part 1 - Survey</h1>'
        html += '<form action="/storeData" method="post">'
        html += '<h4>My gender is:</h4>'
        html += '<input type="radio" name="gender" value="m"
required><span class="survey">Male</span><br>'
        html += '<input type="radio" name="gender"
value="f"><span class="survey">Female</span><br>'
```

```
        html += '<h4>My age range is:</h4>'
        html += '<input type="radio" name="age" value="10-19"
required><span class="survey">10-19</span><br>'
        html += '<input type="radio" name="age" value="20-
29"><span class="survey">20-29</span><br>'
        html += '<input type="radio" name="age" value="30-
39"><span class="survey">30-39</span><br>'
        html += '<input type="radio" name="age" value="40-
49"><span class="survey">40-49</span><br>'
        html += '<input type="radio" name="age" value="50-
59"><span class="survey">50-59</span><br>'
        html += '<input type="radio" name="age" value="60"><span
class="survey">60+</span><br>'
        html += '<h4>I would consider myself as being computer
savvy or "techy"</h4>'
        html += '<input type="radio" name="techy" value="true"
required><span class="survey">True</span><br>'
        html += '<input type="radio" name="techy"
value="false"><span class="survey">False</span><br>'

        html += '<h1>Part 2 - Grouping</h1>'
        html += '<label>Term : </label><select
id="termSelect" name="termSelect"><option> - Select a Term -
</option>'
        for term in sem_cluster.terms:
            html += '<option value="' + term + '">' + term +
'</option>'
        html += '</select>'
        html += '<div id="termDiv"></div>'
        html += '</form>'

        self.response.write(html)

class getCache(webapp2.RequestHandler):

    def get(self):
        cache = []
        with open("cache.json") as f:
            cache = json.load(f)
        jsonObject = json.dumps(cache)
        self.response.write(jsonObject)

class getTerm(webapp2.RequestHandler):

    def get(self):
        html = ''
        term = self.request.get('term')
```

```
        html += "<h2>Term - " + term + "</h2>"
        count = 0
        for sim_algorithm in sem_cluster.sim_algorithms:
            for linkage_type in sem_cluster.linkage_types:
                count += 1
                html += "<h3>" + term + " " + str(count) + " ("
+ sim_algorithm + " similarity - " + linkage_type + "
linkage)</h3>"
                html += """<div class='clusterWrapper'>
                            <input type='button' value='-'
class='adjuster'><input type='text' class='termSlider' value=''
data-slider-min='50' data-slider-max='100' data-slider-step='1'
data-slider-tooltip='hide' data-slider-value='50' data-term='"""
+ term + """' data-sim_algorithm='""" + sim_algorithm + """'
data-linkage_type='""" + linkage_type + """'/> <input
type='button' value='+' class='adjuster'/>
                            <div class="clusterContents"></div>
                            <input type='hidden'
class='clusterslider' name='clusterslider' id='""" + term + "_"
+ sim_algorithm + "_" + linkage_type + """' value='0'>
                          </div>"""
                html += "<center><h4>The groups above are"
                html += "<input type='radio'
id='1"+sim_algorithm+linkage_type+"' name='termRating" +
str(count) + "' value='1' required><label
for='1"+sim_algorithm+linkage_type+"'>Barely</label>"
                html += "<input type='radio'
id='2"+sim_algorithm+linkage_type+"' name='termRating" +
str(count) + "' value='2'><label
for='2"+sim_algorithm+linkage_type+"'>Somewhat</label>"
                html += "<input type='radio'
id='3"+sim_algorithm+linkage_type+"' name='termRating" +
str(count) + "' value='3'><label
for='3"+sim_algorithm+linkage_type+"'>Mostly</label>"
                html += "<input type='radio'
id='4"+sim_algorithm+linkage_type+"' name='termRating" +
str(count) + "' value='4'><label
for='4"+sim_algorithm+linkage_type+"'>Exactly</label>"
                html += "  grouped by
meaning</h4></center>"

        html += '<br><br><center><input type="submit"
value="Submit"></center>'
        self.response.write(html)

class getClusters(webapp2.RequestHandler):
```

```
    def get(self):
        self.response.headers['Content-Type'] = 'text/plain'

        term = self.request.get('term')
        sim_algorithm = self.request.get('sim_algorithm')
        linkage_type = self.request.get('linkage_type')
        threshold = self.request.get('threshold')
        text = sem_cluster.get_clusters(term, sim_algorithm,
linkage_type, threshold)
        self.response.write(text)

class storeData(webapp2.RequestHandler):

    def post(self):
        sliders = self.request.get_all('clusterslider')
        rating1 = self.request.get('termRating1')
        rating2 = self.request.get('termRating2')
        rating3 = self.request.get('termRating3')
        rating4 = self.request.get('termRating4')
        rating5 = self.request.get('termRating5')
        rating6 = self.request.get('termRating6')
        term = self.request.get('termSelect')
        clustererResponse = ClustererResponse(gender =
self.request.get('gender'),
                                              age =
self.request.get('age'),
                                              techy =
self.request.get('techy'),
                                              term =
self.request.get('termSelect'),
                                              lsi_complete =
int(sliders.pop(0)),
                                             lsi_complete_rating
= int(rating1),
                                              lsi_average =
int(sliders.pop(0)),
                                              lsi_average_rating
= int(rating2),
                                              lda_complete =
int(sliders.pop(0)),
                                             lda_complete_rating
= int(rating3),
                                              lda_average =
int(sliders.pop(0)),
                                              lda_average_rating
= int(rating4),
                                              cosine_complete =
```

```
int(sliders.pop(0)),
                                                cosine_complete_rating
= int(rating5),
                                                    cosine_average =
int(sliders.pop(0)),

cosine_average_rating = int(rating6))
        clustererResponse.put()
        html = "<h2>Your responses have been stored. Thanks so
much for your help and have a wonderful day!</h2>"
        html += "<h2>If you wish to evaluate another term,
please hit your back button and choose a different one.</h2>"
        html += "<h3>Otherwise, you may now close your
browser</h3>"
        self.response.write(html)

application = webapp2.WSGIApplication([
    ('/', MainPage),
    ('/getTerm', getTerm),
    ('/getClusters', getClusters),
    ('/storeData', storeData),
    ('/cache', getCache),
], debug=True)
```

script.js

```
// Contains all custom JS needed for clustering app

$(document).ready(function() {
    var cache;
    $.getJSON("/cache", function(data) {
        cache = data;
    });

    $("#termSelect").change(function() {
        var term = $(this).val();
        $.get("/getTerm", { term:term }, function(data) {
            $("#termDiv").html(data);
            $(".adjuster").unbind().click(function() {
                value = $(this).val();
                val = parseInt($(".clusterslider",
$(this).parent()).val());
                if (value == "+") {
                    val += 1;
                }
                else {
```

```
                    val -= 1;
                }
                $(".clusterslider", $(this).parent()).val(val);
                $(".termSlider",
$(this).parent()).slider('setValue', val).trigger("slide");

            });
            $(".termSlider").slider()
                .on('slide', {cool:$(this).attr("data-term")},
function(ev) {
                    var text = "";
                    var term = $(this).attr("data-term");
                    var sim_algorithm = $(this).attr("data-
sim_algorithm");
                    var linkage_type = $(this).attr("data-
linkage_type");
                    var threshold =
$(this).data('slider').getValue();
                    var clusters =
cache[term].data[sim_algorithm][linkage_type][threshold][0];
                    var domains = cache[term].domains;
                    var concepts = cache[term].concepts;
                    var widths =
cache[term].data[sim_algorithm][linkage_type][threshold][1];
                    var clusterColors =
get_cluster_colors(widths);
                    var temp = []
                    for (var i = 0; i < clusters.length; i++) {
                        value = clusters[i] - 1;
                        if (temp[value] instanceof Array) {
                            temp[value].push(i);
                        }
                        else {
                            temp[value] = [i];
                        }
                    }
                    for (var i = 0; i < temp.length; i++) {
                        text += "<div class='cluster'
style='background-color:rgba(" + clusterColors[i] +
",.2)'><h4>Group " + (i+1) + '</h4>'
                        for (var j = 0; j < temp[i].length; j++)
{
                            originIndex = temp[i][j];
                            originText = concepts[originIndex]
                            domain = domains[originIndex]
                            text += "<p class='"+ domain + "'
title='" + domain + "'>" + originText + '</p><hr>';
```

```
                        }
//                        text = text[:-4]
                        text = text.substring(0, text.length -
4);

                        text += '</div>'
                    }
                    text += '<div class="clear"></div>'
                    $(this).parent().next().next().html(text);

$("#"+term+"_"+sim_algorithm+"_"+linkage_type).val(threshold);
                });
            $(".termSlider").trigger("slide");
        });
    });

    function get_cluster_colors(clusterWidths) {
        colors = [];
        for (var i = 0; i < clusterWidths.length; i++) {
            width = clusterWidths[i];
            if (width >= 1) {
                width = .99;
            }
            width *= 511;

            redValue = 0;
            greenValue = 0;
            if (width < 255) {
                greenValue = 255;
                redValue = Math.round(Math.sqrt(width) * 16);
            }
            else {
                redValue = 255;
                width = width - 255;
                greenValue = 256 - Math.round(width * width /
255);
            }
            colors.push(redValue + "," + greenValue + "," +
"0");
        }

        return colors
    }
});
```

style.css

```
body {
    font-size: 12px
!important;
    margin:10px !important;
}

.clear {
    clear:both;
}

.cluster {
    display:inline-table;
    width:200px;
    margin-right:5px;
    white-space:normal;
    padding:5px;
}

.clusterContents {
    height:650px;
    overflow:auto;
}

.clusterWrapper {
    white-space:nowrap;
    margin-left:25px;
    border: solid 1px gray;
    background-
color:lightgray;
}

h2 {
    margin-left:15px;
}

h3 {
    margin-left:25px;
    margin-bottom:0px;
}

input[type="radio"] {
    margin-left:15px
!important;
}

.instructionParagraph {
    font-size:18px;
}
p {
    margin-top:0px;
}

.cluster p {
    padding:3px;
    border-style: dotted;
    border-width: 1px;
    border-radius: 5px;
}

hr {
    border-top-color:black
!important;
    margin-top:5px
!important;
    margin-bottom:10px
!important;
}

.survey {
    font-size:15px;
    margin-left:5px;
    position:relative;
    top:-2px;
}

.GraphicDesign {
    border-color: red;
}

.InformationTechnology {
    border-color: blue;
}

.Telecommunications {
    border-color: green;
}

.UserExperienceDesign {
    border-color: purple;
}

.ComputerScience {
    border-color: orange;
}
```

```
.SoftwareEngineering {
    border-color: yellow;
}

.Workflow {
    border-color: cyan;
}

.SoftwareandSystemEngineering
{
    border-color: white;
}

.RequirementsEngineering {
    border-color: fuchsia;
}

.BusinessProcessManagement {
    border-color: peru;
}

.InformationSecurity {
    border-color: teal;
}

.InformationSystems {
    border-color: chartreuse;
}
```