2017

Reimagining the SSMinT Software Package

Kelly Cecil

Reimagining the SSMinT Software Package

Kelly Cecil

Thesis submitted to the
College of Engineering and Mineral Resources
at West Virginia University
in partial fulfillment of the requirements
for the degree of

Master of Science in Computer Science

Elaine Eschen, Ph.D., Chair
Alan Barnes, Ph.D.
Roy Nutter, Ph.D.

Lane Department of Computer Science and Electrical Engineering
Morgantown, West Virginia
2017

Keywords: text data mining contextual semantic document clustering index corpus

Abstract

Reimagining the SSMinT Software Package

Kelly Cecil

The SSMinT software package is a tool set created by Eschen, Barnes, Kiran, and Peddada in 2010 to explore a semantically-sensitive form of text mining. The text mining process finds relationships between keywords, taking the orientation and locality of the keywords into consideration. These are quantified to produce a *semantic signature*. Experiments found their text mining method to be successful, and additional research yielded automated keyword and semantic signature generation to assist in finding important concepts within documents. The tools created through this research demonstrated the concepts effectively, but are difficult to use on large data sets.

A new SSMinT library is presented that is designed and written for use as the basis of future experiments and applications leveraging SSMinT. Instructions are provided for using the included automated keyword and learner tools, and running the SSMinT applications in the cloud is discussed. An overview of the data structures and algorithms further outlines the SSMinT library, allowing a developer to write new applications with minimal friction.

We examine two proposed indexing algorithms taking advantage of the new SSMinT libraries. The two algorithms primarily differ in their selection of documents for learning. The batch indexing method selects some random number of documents for learning. The iterative indexing method uses a single randomly selected document to discover semantic signatures, which are then used to find additional related documents. The batch indexing method discovers one to three semantic signatures per document, resulting in poor clustering performance as evaluated by human cross-validation of clusters using the Adjusted Rand Index. The iterative indexing method discovers more semantic signatures per document, resulting in far better clustering performance using the same cross-validation method.

Our new tools enable faster development of new experiments, forensic applications, and more. The experiments show that SSMinT can provide effective indexing for text data such as e-mail or web pages. We conclude with areas of future research which may benefit from utilizing SSMinT.

Dedication

To my cocker spaniel Buddy Light.

Acknowledgments

To Mom, Dad and my entire Cecil and Rickman family, I could write endless acknowledgments and never fully convey the impact you've had on my life. I am humbled by the sacrifices you have made for me, and only because of you am I able to stand with the incredible people in my field. I sincerely hope that this work honors the sacrifices you have made for me. I love you all.

To Mr. Bill Clark, you introduced me to an entire world of wonder behind the computer screen. If I am half as influential to my students as you were to me, I'll consider myself an incredible teacher. Thank you for everything.

To my advisers Dr. Elaine Eschen and Dr. Alan Barnes, thank you for the input and insight you provided during this work. Thank you for being patient during my break from graduate school and working with me to finish this part of my journey. I sincerely hope this work was worth the wait.

Finally, to my wife, Mollie, you entered my life after my graduate school coursework and before the completion of this thesis. I'm still in awe of how quickly you were able to review my thesis. I'm fortunate to have married such a brilliant, beautiful woman. Thank you for the encouragement and love.

Contents

List of Figures

List of Tables

Symbols and Abbreviations

ARFF: Attribute Relationship File Format
SSD: Semantic Signature Descriptor
KDF: Keyboard Descriptor File

Chapter 1

Introduction

1.1 Motivation

The work on this thesis stems from a particular desire that tends to be very unique to software engineers.

A stable software package has built-in tests, such as unit and integration tests, to help verify the behavior and accuracy of the software. A mature package has code that is modular, easily readable, and has been designed to be relatively easy to extend. This readability and extensibility can significantly add to the life of a software package while drastically decreasing the time required for development iterations.

Why should research software care about reduced development iterations or testing? The answer is straight-forward. Research can be incredibly fast paced; ideas should be validated or disproven as quickly as possible. Stable, well-tested tools help to provide researchers and students with the confidence necessary to concentrate on their experiments without worrying whether their tools are functioning properly. The correct functioning of these tools is especially important in a group where researchers and students are working together and contributing to the same code base with the effort of making a high-quality, shared basis for experiments. A change in one code

location can lead to confusing results if the code is not shared across the team or can even affect the entire team if a poorly-tested change is committed to a repository and shared with others.

A part of thesis is the creation of a stable, well-tested code base for future students studying the SSMinT tools, to improve the code, and to build new tools on top of a modularized library of components. The original tools were created quickly as prototypes to demonstrate general concepts, but the original code has been frequently difficult to understand and to extend. The tools have been reimplemented to improve code quality, to encourage experiments with larger data sets, and to continue to push the research forward.

Our second objective is to build a brand new tool to demonstrate the effectiveness of the reimplemented tools. This newly implemented tool demonstrates leveraging the new code to construct and execute a larger experiment. Our choice is to implement an algorithm that performs unsupervised document indexing. The SSMinT tools have potential in finding variables and traits of data which can effectively divide a corpus of data, and the demonstration creates many possibilities for new research and partnerships.

1.2 Contributions

This thesis makes contributions in the implementation of the original tools as well as the introduction of a new document indexing algorithm using the new implementation of the original tools as a base.

The contributions of this thesis specifically are:

- New implementation of the original SSMinT tools in Java with the following attributes:
  - Modular components to maximize code reuse.
  - Unit testing to avoid regressions and help ensure that code is safe to refactor.
  - Easy to understand and extend.

  - Easy to build and deploy into a data center and cloud.
  - Built to take advantage of multi-core systems for higher throughput.
- Development and implementation of a document indexing algorithm.

1.3 Structure of Thesis

The structure of this thesis is as follows:

- Chapter 1 introduces the thesis including the contributions and motivations behind the work.
- Chapter 2 discusses important concepts and ideas that serve as a basis to understanding the work in this thesis.
- Chapter 3 introduces the new SSMinT software package and provides an overview to help introduce new users and developers to what is offered.
- Chapter 4 takes a look at a document indexing algorithm developed utilizing the new SSMinT software package.
- Chapter 5 provides an overview of several experiments conducted using the document indexing algorithm and observations of the results.
- Chapter 6 summarizes the findings of this thesis and suggests future work that may further advance the study.

Chapter 2

Background

An understanding of background material is crucial for a full understanding of the motivations behind the reworked tools.

2.1 Overview

The objective of the SSMinT tool package is to analyze text in a contextually sensitive way. The initial prototypes were designed and built by the Discrete Algorithms Research Team (DART) at West Virginia University, directed by Elaine Eschen and Alan Barnes, with member graduate students Uday Para Kiran [22] and Ramya Peddada [23]; later, improvements were made by graduate student Udaya Mallampati [18]. This chapter will provide background on the SSMinT tools, text mining concepts, and other topics to serve as a basis to introduce the new tools.

2.2 What is Text Mining?

Our modern world is largely data driven. Huge amounts of data is created daily. Consider social networking sites such as Facebook and Twitter; users contribute to a large textual data set with every status update. Look to your wallet or key chain, and you'll likely see one or more customer

loyalty cards. These cards enable retailers to track purchases as part of a larger data set. Every social media post, purchase, phone call, text message, and more contributes to the enormous amount of data our daily lives produce each and every day. What if one could take this data and find the interesting patterns within to help us learn more about our complicated, data-driven world?

This process is called data mining; data mining can be more formally defined as the practice of automatically searching large stores of data to discover patterns and trends that go beyond simple analysis [21]. Data mining employs mathematical algorithms to allow discovery of patterns and prediction of potential outcomes from data sets.

Data mining has made fantastic differences in how we understand our world. Trends from data can now be understood to maximize the desired effect. Data mining techniques can even identify trends before we realize them ourselves. The department store chain Target successfully used data mining techniques to identify a young, pregnant girl before her own father knew [10].

Text mining is a somewhat different problem. Data mining looks at numeric and discrete values to assess a pattern that likely will not be obvious to the casual observer without the aid of a mathematical algorithm. Text mining looks at textual data in which the information in the data is obvious; an author's objective is generally to state their opinions clearly and unambiguously. These differences in the data only make a difference to a human; the computer is still interested in finding the "actionable" information in the data [29].

2.3 Supervised vs. Unsupervised Learning

There are generally two types of learning in machine learning: *supervised learning* and *unsupervised learning*. Understanding the two types is helpful when understanding machine learning techniques, and we'll briefly describe the two learning techniques now.

Supervised learning forms predictions for data points based on previously observed data points. A metaphor in the book *Unsupervised Learning* [9] suggests that supervised learning is like the

relationship between a student and teacher. A student might suggest an answer in response to a teacher's question, and the teacher will respond with whether the answer is correct and why the answer is not correct if applicable; likewise, a data point and computed classification from a learning algorithm would be provided to a supervisor to be told whether the classification is correct and computed error associated with the classification.

Many algorithms are available for supervised learning. Two examples of supervised machine learning algorithms are Naive Bayes and HyperPipes; both algorithms are available for use in the WEKA machine learning tool set [25].

Unsupervised learning takes a different approach. Predictor variables for each data point are used, but the data points are unlabeled and thus do not use feedback from a supervisor. Unsupervised learning algorithms include clustering techniques such as K-Means or kd-trees, and blind signal separation techniques such as *principle component analysis* and *singular value decomposition*.

Our techniques described later will be unsupervised learning techniques with the explicit goal of finding variables to effectively divide space.

2.4 Relevant Problems in Text Mining

2.4.1 Text Classification

Much of the original research in text classification was performed to index documents using a *Boolean Information Retrieval System* [26]. This utilizes a finite set of keywords known as a *controlled dictionary* to describe the contents of a document. A controlled dictionary is a thematic thesaurus that differs between domains; for example, aerospace engineering and medicine would have different dictionaries to describe topics within that realm. The dictionaries would be created by trained humans and is thus an expensive task. This indexing easily becomes a text classification

problem if the keywords in the controlled dictionaries are replaced with categories [26].

2.4.2 Document Clustering

Document clustering is an unsupervised learning technique that attempts to group like documents together based on observed attributes in the individual documents. This allows documents to group themselves together instead of using pre-defined classes (in a supervised algorithm.)

Document clustering can be computationally expensive as compared to supervised learning algorithms. Documents can require hundreds to thousands of attributes in order to effectively cluster documents [29].

2.5 Real-World Applications for Text Mining

2.5.1 Application to Digital Forensics

SANS Institute author Craig Wright [30] believes that text mining has done little to change the field of computer forensics. Text documents such as e-mails and word processing documents are unstructured and have previously been difficult to evaluate in an automated fashion. Modern advancements in the field of text mining help to solve several problems inherent in digital forensic investigations.

One major topic at the time of this writing is the concept of the *plain view doctrine*. This doctrine allows an officer or investigator to seize evidence or contraband that is in plain view without a warrant. Current laws do not address what is considered *plain view doctrine* with respect to digital evidence, and there are several emerging viewpoints regarding how to amend the rules of investigation to address these concerns [8]. The application of text mining to forensic searches potentially bypasses concerns regarding the plain view doctrine, since the algorithms are only displaying text that is relevant to the query at hand.

The SSMinT package can allow forensic investigators to prepare a library of *semantic signatures* from prepared training sets or previous investigations that can assist in future investigations. This may also be useful to legal teams during the discovery phase of investigations. Legal teams may pour over hundreds of pages of documents given to them by the opposing side. Applying *semantic signatures* using the SSMinT package may provide interesting parts of the documents to investigate on a first pass saving a legal team's valuable time and resources.

2.5.2 Application to Health Care and Medicine

The applications of text mining in health care have grown substantially in recent years. We will focus on two possible applications: indexing of new medical research and searching of patient records.

New scientific literature in biology is published almost daily, and the literature represents an incredibly complex field with many subtleties. Text mining techniques would need to recognize entities such as interactions between drugs and proteins, adverse effects of chemical compounds, metabolic reaction relations, and more [14]. Contextually-sensitive text mining techniques stand to be very beneficial as to allow greater flexibility in understanding complex relationships between medical entities.

Physicians could also find huge benefits in applying text mining techniques to patient records. These patient "notes" are free-text fields that contain actions and medications prescribed by the physician making these a rich target for text mining. The task is not trivial; these notes can be written very differently depending on the physician that sees the patient. A contextually sensitive text mining algorithm could potentially filter patients to find prime candidates for drug testing or to search records and raise concerns when conflicting prescriptions are prescribed [27].

2.6 The Original SSMinT Software Package

We will begin with a brief history of the research efforts on *semantic signatures* and follow with a description of the primary tools in the SSMinT package. We will refer back to these sections later when describing our indexing algorithm.

2.6.1 A Brief History

The initial research into the concept of semantic signatures began with the work of Uday Kiran Para [22] and Sri Ramya Peddada [23] with Dr. Elaine Eschen and Dr. Alan Barnes of the Discrete Algorithms Research Team (DART) at West Virginia University. Traditional approaches of identifying document attributes such as bag-of-words were not found to capture the relationship between keywords in a document. Zhang et al. in [31] [32] derived a single high-dimensional vector to establish relationships between keywords in a document with the objective of classifying documents. DART successfully experimented with generating many vectors per document. These vectors form the basis for constructing *semantic signatures*. Rukmini Ravali Kota later developed an automated process of generating semantic signatures in a document corpus [13]. Additional code improvements and experiments on optimal keyword discovery were done by Udaya Mallampati [18].

2.6.2 Keyword Tool

The *Keyword Tool* is the first of three tools in the SSMinT package and is designed to help identify keywords that are semantically correlated and are frequent in the text. Human experts or investigators may select keywords to attempt to capture words of interest that represents concepts as anger, violent intent, paranoia, and other feelings.

The *Keyword Tool* counts the frequency of each word in the document, while filtering out common stop words to find good candidates for the first word in the set. The second and third

words are suggested based on the weight function in Figure 2.1. In our experiments the constant a is generally set to 5. The selected set of words are referred to as keywords and are stored in XML documents called a *Keyword Descriptor File (KDF)* to be used by the Learner tool.

$$w(x,a) = \sqrt{\frac{a^2}{a^2+x^2}}$$

Figure 2.1: Window weight function where a is a constant and x is the distance between words.

2.6.3 Learner Tool

The *Learner Tool* is the second of three tools in the SSMinT package and is intended to help users and investigators select good semantic signatures from training text. A user can use the *Learner Tool* to construct a library of semantic signatures from sample documents that may be applied to a corpus such as an collection of an organization's e-mail to find relevant information in a more context-sensitive manner than using a simple bag-of-words approach.

The *Learner Tool* uses the keywords stored in the Keyword Descriptor Files created by the Keyword tool to find combinations of these selected interesting keywords. The tool scans documents to find an occurrence of any of the keywords. Finding a word begins a search window (typically of size 20 words) in which other keywords may occur. Weighted distances are found between all keywords within this window and used to construct a term matrix. This term matrix is flattened in row major order to create a vector. An example document, term matrix, and vector are shown in Figures 2.2, 2.3, and 2.4. Vectors generated from a keyword set are clustered using K-means. The cluster centroids and radii are calculated for the selected clusters. The cluster centroid and radius plus keyword set are called a *semantic signature.*

How useful is using **curl** over **netcat**? Using **curl** can potentially provide benefits in **scripting** and ease of use, but anyone who already has the muscle memory to hammer out something familiar to the netcat line above might not be as easily persuaded by the convenience.

Figure 2.2: An example sentence taken from [7] with keywords *curl*, *scripting*, and *netcat*.

	curl	scripting	netcat
curl	0.61	0.1	0.86
scripting	0.0	0.0	0.0
netcat	0.86	0.28	0.0

Figure 2.3: An example term matrix derived from Figure 2.2

2.6.4 Data Analysis Tool

The *Data Analysis Tool* is the third of three tools in the SSMinT package. An investigator can apply semantic signatures collected through the investigation of training documents to an interesting corpus. The application allows the user to save the *term document matrix* as an ARFF file for analysis in WEKA [25] and supports K-means clustering on the *document analysis matrix* from within the application.

2.7 Important Concepts

The research in this thesis builds strongly on prerequisite work. Understanding this work is very important to a proper understanding of the methods used in this research.

2.7.1 K-means Clustering Algorithm

The K-means clustering algorithm is a popular clustering algorithm in data mining that aims to partition the data into k clusters based on their natural centroids. The algorithm was first applied to clustering in 1967 in a paper by James MacQueen [17] and is still a popular algorithm for data mining today.

<0.61, 0.1, 0.86, 0.0, 0.0, 0.0, 0.86, 0.28, 0.0>

Figure 2.4: A vector formed from the term matrix in Figure 2.3

Input: D - Data points to be clustered, n - number of data points in D, k - number of clusters
Initialize: A list of cluster centroids $C_{1..k}$ to random initial data points, m representing a list of cluster memberships
for each data point $d_i \in D$ **do**
  $m(p_i) \leftarrow \underset{k \in \{1..n\}}{argmin}$ distance(p_i, c_k)
end for
while cluster membership m has changed **do**
  for each $c_i \in C$ **do**
    $c_i \leftarrow \frac{1}{|m_i|} \sum_{x_j \in m_i} x_j$
  end for
  for each data point $d_i \in D$ **do**
    $m(p_i) \leftarrow \underset{k \in \{1ton\}}{argmin}$ distance(p_i, c_k)
  end for
end while

Figure 2.5: Pseudo-code for K-means clustering algorithm.

The algorithm has two stages: the *initial stage* and the *update stage*. The algorithm begins with the *initial stage* by picking k random points to act as initial centroids. The algorithm proceeds to the *update stage*. The distance between each cluster centroid and point is calculated, and points are assigned to the cluster whose centroid is the shortest distance. The centroid locations are then recalculated to the mean point of the points currently assigned to the cluster. This update stage is repeated until none of the points change their cluster membership after the update phase. Figure 2.5 provides high-level pseudo-code for the algorithm [17].

One pitfall of the K-means clustering algorithm is that the random selection of clusters may lead the algorithms to produce low quality clusters. Several sampling algorithms have been proposed to improve the selection of the initial clusters. K-means++ is one of the most popular sampling methods for initial clusters and consistently provides initial clusters that have a smaller average of sums than the standard K-means algorithm [2].

2.7.2 Cluster Analysis

A concern that must be addressed when comparing clustering methods such as K-means is how to compare the clusters produced using different methods. Clustering methods may be evaluated using questions outlined by William Rand [24].

The first question is whether the methods find "natural" clusterings. Unsupervised document clustering will produce some sort of structure in the documents even in the absence of an obvious structure in the data. Does the clustering method in question find the obvious structure in the data, or is it incapable of finding the structure?

The second question is, given two clustering methods, whether the methods yield the same clustering when applied to the same data. The ability to evaluate clusterings from different methods opens the analysis to other interesting questions. Suppose that we have two clustering methods. One method may be more computationally expensive than the other, but does the more simple method provide a suitable approximation in cases where a trade off between computation time and accuracy is acceptable? Suppose that we have a clustering generated from a clustering method and a clustering provided by one or more humans. We can compare the two clusterings to see how the humans and the clustering method agree (or disagree.)

A popular method for computing the similarity between clusters is called the Rand Index [24]. The Rand Index is calculated for two clusters Y and Y' containing the same N points. One must arbitrarily number the clusters and let n_{ij} be the number of points that are simultaneously in the i-th cluster of Y and the j-th cluster of Y'. The similarity between Y and Y' is then calculated by: $c(Y,Y') = [\binom{n}{2} - [\frac{1}{2}\{\Sigma_i(\Sigma_i n_{ij})^2 + \Sigma_j(\Sigma_i n_{ij})^2\} - \Sigma\Sigma n_{ij}^2]]/\binom{n}{2}$.

The quantity (Y, Y') will be a value between 0 and 1, with 0 indicating clusters that have nothing in common and 1 indicating the clusters are identical.

The Rand Index is a good start for comparing clusterings, but the measure has a slight flaw. There remains the possibility that our clustering algorithms may produce some similar clusters by random chance instead of agreeing through some reasoned method. We may use the Adjusted

Y/Y'	Y'_1	Y'_2	...	Y'_s	Sums
Y_1	n_{11}	n_{12}	...	n_{1s}	a_1
Y_2	n_{21}	n_{22}	...	n_{2s}	a_2
...	...	...	...	...	...
Y_r	n_{r1}	n_{r2}	...	n_{rs}	a_r
Sums	b_1	b_2	...	b_s	

Figure 2.6: Contingency Table for Adjusted Rand Index.

Rand Index [11] to produce a cluster similarity measure that is adjusted to compensate for cluster matches that may be random.

Our first step when calculating the Adjusted Rand Index is to produce a contingency table demonstrated in Figure 2.6. Our contingency table is made by using n elements in clustering algorithms to produce two groupings $Y = Y_1, Y_2, ..., Y_r$ and $Y' = Y'_1, Y'_2, ..., Y'_s$. The Adjusted Rand Indexing (ARI) is then calculated as $ARI = \frac{\Sigma_{ij}\binom{n_{ij}}{2} - [\Sigma_i \binom{a_i}{2} \Sigma_j \binom{b_j}{2}]/\binom{n}{2}}{\frac{1}{2}[\Sigma_i \binom{a_i}{2} + \Sigma_j \binom{b_j}{2}] - \Sigma_i \binom{a_i}{2} \Sigma_j \binom{b_j}{2}]/\binom{n}{2}}$ [11]. The result of the ARI is similar to the standard Rand Index with an exception: A negative value is possible due to the correction for clusters with random chance, and indicates the clusterings have nothing in common.

Additional methods and computations for comparing clusters are available [20], but the Adjusted Rand Index has seen wide adoption and alternate methods of cluster comparison are often compared to it. Our experiments will use the Adjusted Rand Index, but other measures can and should be evaluated further.

2.7.3 Term Frequency/Inverse Document Frequency

The art of selecting the correct keywords from a document is important when performing unsupervised learning from textual data. One must take care to find words that provide the document with the most impact regarding topic or statement. We turn to a statistical method to aid in finding the most important keywords for a document that might set it apart from the rest of the corpus. This method is called Term Frequency/Inverse Document Frequency, or TF-IDF as it will be referred to

from this point forward.

The *term frequency* originates from the need to distinguish documents from one another. Suppose that we are searching for documents containing "apple cider". We can initially sort documents by whether they contain the words "apple" and "cider." H.P. Luhn [15] observed that documents can be further sorted by observing the number of times the words appear in documents. Luhn specifically states [15] that the "weight of a term that occurs in a document is simply proportional to the term frequency".

There are several methods of computing the term frequency, but the simplest method is known as the binary (or Boolean) method. A value of 1 is used when a term is used in a document, and a value of zero is used if the word is not used in the document.

Using *term frequency* alone is not sufficient. Words such as "the" occur frequently in many documents, but they have very little use in information retrieval. A method of finding words that are meaningful to a specific document as opposed to other documents is necessary. Karen Jones [12] made the observation that the "specificity of a term can be quantified as an inverse function of the number of documents in which it occurs". This observation leads to the *inverse document frequency* calculation.

There are several methods for calculating the inverse document frequency, but the most simple measure is to take the natural logarithm of the number of documents divided by the number of documents containing the term. Let D be the set of documents and $N = |D|$, then

$$idf(t,D) = \log \frac{N}{|\{d \in D | t \in d\}|}$$

Figure 2.7: The standard IDF formula

The final TF-IDF weight is found by multiplying the term frequency and the inverse document frequency.

	assets_liabilities	inventory_loss	net_sales_increase	assets_impairment
Circuit City	6	13	1	7
Best Buy	9	25	9	5

Figure 2.8: An example term-document matrix containing two companies and five semantic signatures.

2.7.4 Term-Document Matrix

A *term-document matrix* is a convenient arrangement to analyze the frequency of terms (or *semantic signatures* in our case). The matrix holds terms (or semantic signatures) along one axis and documents along the other. The individual cells represent the frequencies that semantic signatures occur in each document.

We see an example term-document matrix in Figure 2.8 that one might see from financial reports. We can see that the *semantic signature* using keywords “assets" and “liabilities" is found six times in the Circuit City report and found nine times in the Best Buy report.

2.7.5 Singular Value Decomposition

An important mathematical concept in our algorithm is known as Singular Value Decomposition. Suppose we have a matrix A that is m by n. The Singular Value Decomposition of A can be defined as $A = U \sum V^T$ [4]. The matrices U and V represent the breakdown of the relationships of the data into linearly-independent vectors [5].

The Singular Value Decomposition is useful in text mining in processes such as Latent Semantic Indexing. The objective in Latent Semantic Indexing is to use the Singular Value Decomposition of a term-document matrix to model the relationship between terms and documents. The component matrices U and V of the decomposition represent the document space and term space respectively, and let us observe the relationships between terms and documents [5].

2.7.6 Comparison of GUI and Console Application

Dealing with large amounts of streaming data can be a somewhat difficult venture if not considered properly.

GUI Application

The most significant advantage of an application featuring a guided user interface (GUI) is the ease-of-use such an interface provides. The application can provide active feedback regarding the use of the application including the ability to select keywords or vectors to export, context of the selected keywords, or progress on the current processing. GUI applications are the perfect choice when interacting with an analyst in real time.

A well-executed GUI application is also a somewhat difficult item to build. A GUI application must be written with the data processing executed on a thread to avoid blocking the user interface from processing events. Failure to allow UI events to be processed can cause the operating system to misidentify the application as a hung process and attempt to halt the process erroneously.

Console Application

Console applications can have varying levels of interactivity depending on their design. Many console applications intend for the user to provide configuration options through flags and switches passed through the console. Providing configuration through the console does not have the same level of user-friendliness that a GUI application provides, but this style of configuration provides the ability to chain applications together such that data or other processing tasks can be accomplished by chaining multiple applications together. Constructing programs in this way is often referred to as the UNIX philosophy, and this design allows for a knowledgeable analyst to accomplish more with less.

Application to Data and Text Mining

A well-constructed data mining application is very important to the usability of the SSMinT software package. The process of manually discovering semantic signatures and creating a library that could be used in an investigation is a task that benefits from a guided user interface to assist users through the process of finding keywords and learning from documents. Automated keyword and semantic signature learning, as well as the process of finding semantic signature hits, can generally be hindered by a guided user interface as headless operation allows for easier use of pipes between tools and ease of deploying to cloud-based servers.

Chapter 3

The Redesigned SSMinT Software Package

This section will discuss the tools in the SSMinT package and how the tools have been restructured to facilitate better chaining and streaming.

3.1 Motivation

The original SSMinT package demonstrated the value of using semantic signatures and the methodology behind contextually sensitive text mining. The package continued to evolve and improved quickly during the development of the initial prototypes. Keeping up with the speed of discovery meant that implementing new ideas in a clean, concise manner was a concern of the researchers.

The question at hand is what can be gained from reimplementing the tools in both technical terms and in terms of programmer and researcher productivity. The common wisdom in software is that the problem is not well understood until the first implementation, and lessons from the previous tools can be used to build a more robust framework for experiments.

3.2 Easier Automation

The world of text mining has changed in recent years with the rise of cloud computing services. Researchers previously needed to invest significant amounts of money into hardware to power large experiments. Providers such as Amazon through Amazon Web Services and Google through Google Cloud Engine provide virtual machines that can be used for hosting applications, performing desktop work, and any number of additional applications. Users are only charged for time during which the virtual machine is active. The "pay-as-you-go" nature of these virtual machines often means that use of cloud services can be much cheaper than obtaining and maintaining equivalent hardware. Researchers have taken advantage of these cloud providers to perform larger and more parallelized experiments; providers such as Amazon and Google have embraced this trend by providing products geared towards business intelligence and data mining.

This elasticity afforded to us by cloud providers means that one must consider how to engineer our experiments. These virtual machines are often headless and configured by pulling packages from a repository to the machine during the virtual machine configuration. Our experimental harnesses must allow us to easily distribute our code as a package or library and be easily invoked programatically.

Reimplementing the SSMinT package to be a set of libraries allows one to create any interface we need for the task. A command line interface was easily created for use in the cloud, and the same libraries providing the functions for our command line interface can power a GUI for interactive use.

3.3 Better Scalability

The original SSMinT package was created to allow an investigator to interactively use the tools to search for documents in a corpus that may be of interest. These original tools are acceptable for small data sets that may be used for testing or experimental purposes, but the normal case will

consist of gigabytes of data. Extreme cases will consist of terabytes or more of data. Scalability is very important to the ability to process large amounts of data, and the new tools have been built with large amounts of data in mind. We analyzed the SSMinT software to design a process that allows distributed processing at different stages.

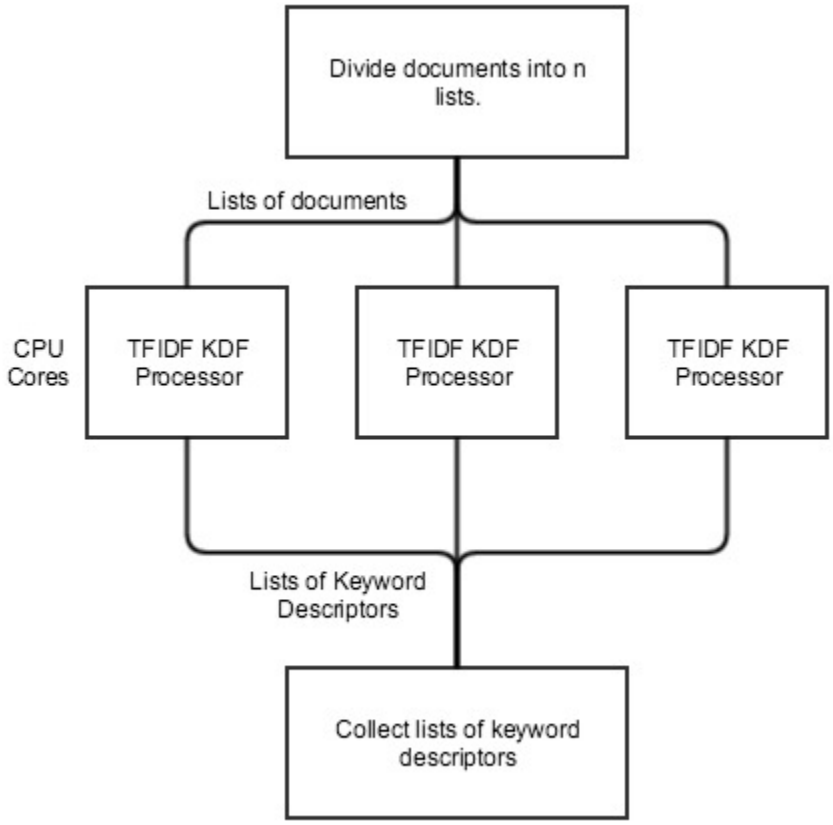

Figure 3.1: Splitting of KDF processing across cores.

Scalability in some stages is straight-forward. Threading the search for keywords in documents is a simple divide-and-conquer technique. The list of documents is divided evenly and distributed to multiple worker processes executed on separate CPUs. The individual workers process the documents provided and return lists of keyword descriptors. The new implementation of K-means also uses a similarly simple divide-and-conquer strategy; the computation of distance of vectors to centroids is divided among the available CPU cores minus one.

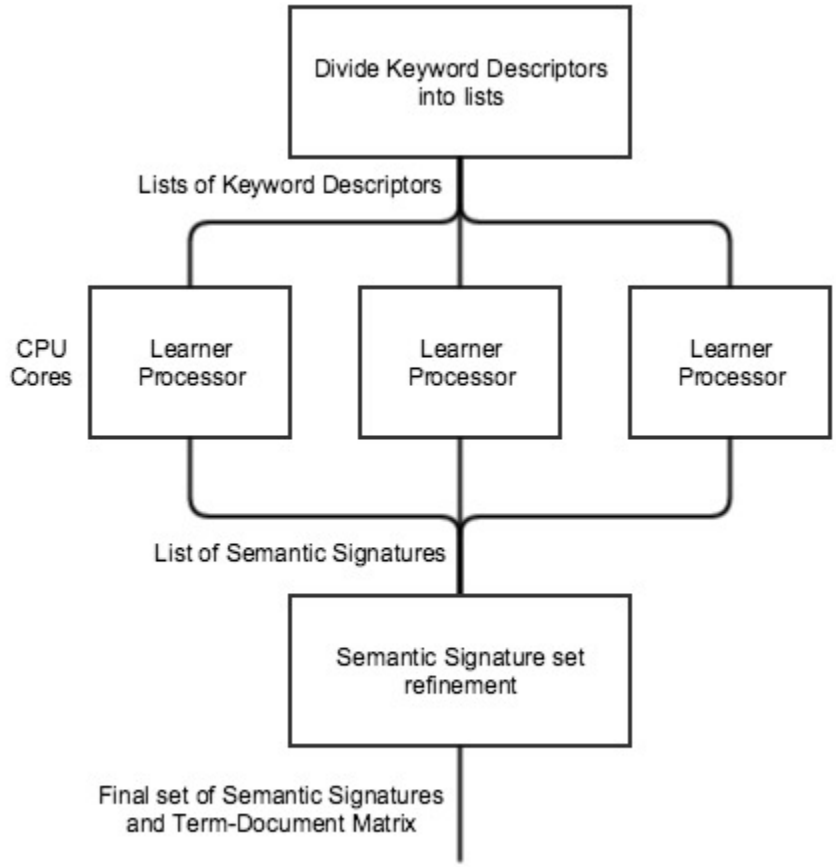

Figure 3.2: Multi-threading of Learning process.

Scalability in other stages, particularly the learning stage, requires more thought. The objective is for vectors from learning to be collected from all documents. We also want to cluster the vectors when finished to find sets of vectors that provide different concepts. Scaling using multiple processes of any kind adds processing overhead to maintaining the additional processes, and care is required to ensure that the processing overhead is minimized. We can cautiously use a divide-and-conquer approach to scaling our process. The work for the learning stage is divided evenly by separating the sets of keywords into lists and distributing these lists to each process.

The approaches to threading allow these processes to take advantage of multiple CPU cores. This could also be applied to distributing the work across a network to take advantage of multiple

computers in a data center.

3.4 Modularized Components

Special attention has been paid to ensure that components in the SSMinT package are modularized according to established best practices. The two followed practices are briefly discussed: the Single Responsibility Principle and DRY (Don't Repeat Yourself) components.

The Single Responsibility Principle [19] is a best practice termed by Robert C. Martin that states that a class or module should only ever have one reason to change. A class or module that can change for multiple reasons is believed to be concerned with too many responsibilities and may be easily broken by changes. Consider an application where a class containing GUI code also contains an implementation of K-Means. The Single Responsibility Principle is violated since our class could change due to a user interface change or a modification to our K-means algorithm. Such a change in the GUI could have side effects in our K-means implementation.

The nature of the Single Responsibility Principle promotes class reuse, and this helps to build DRY components. Consider K-Means as an example. SSMinT extensively uses the K-Means clustering algorithms, but the original tools duplicated the code for each tool. Bugs in one implementation in one tool may not be fixed in the other implementations leading to tools that may behave inconsistently. Reuse of the same code through our refactor helps to assure fixes and improvements in K-means are in all SSMinT tools.

3.5 Component Testing

Software testing is an important task to ensure the proper function of software components. Our testing for SSMinT is focused on the correct behavior of components that we have written, so our testing is implemented using *unit tests*. Unit tests are intended to verify the behavior of code

by verifying a certain output given some input. Some algorithms such as K-means uses a preset random seed to account for randomness during the tests. An example unit test from the SSMinT tools can be found in Figure 3.5.

The new SSMinT package implements many unit tests to validate that the behavior of classes and methods are correct for most cases. These tests help to identify regressions during code refactor and assurance that code is behaving as intended.

3.6 The New Tools

This section introduces the new tools for users who want to use the existing applications to perform experiments. Following sections introduces programmers to the algorithms and structures in the library so they can write their own experiments and applications.

3.6.1 Hardware Requirements

The SSMinT toolset has been reimplemented in the Java programming language. The JVM (Java Virtual Machine) is available on Windows, Mac OS, Linux, and more operating systems, and is available for most CPU architectures. SSMinT is thus able to run on most platforms with minimal hassle. Users MUST use Java 8 as the code makes frequent use of functional programming idioms only available in that version.

Processor Requirements

Machines should have at least one CPU that is reasonably powerful. The new SSMinT toolset have example applications that are both single-threaded (fantastic for running small experiments) and multi-threaded (great for large data sets). More powerful CPUs are recommended to speed the processing of files, but are not necessary.

RAM Requirements

Experiments require a machine with RAM sufficient for the size of the data set. The required memory has been reduced where the balance of memory usage and performance was found to be acceptable. The SSMinT toolset uses a lightweight implementation for Lists and a matrix math library supporting sparse matrices. Additional steps could be taken to flush data to the hard disk to continue to free up RAM, but was not required for our experiments.

Recommended Environment

Indexing experiments were performed on an Ubuntu Linux-based VPS cloud instance using 18 CPU cores and roughly 21GB of RAM. This instance was chosen simply to allow the new SSMinT tools to take advantage of the additional cores and caching of the file reads in the data sets. This cloud setup is highly-recommended for indexing experiments that require multiple passes of data. Additional experiments with smaller data sets (usually 500 short documents or less) and data analysis tools were performed on a 2013 MacBook Pro running Mac OS X utilizing an Intel Core i7 processor with 8 GB of RAM.

3.6.2 Building the Toolset

Much care has been taken to ensure the new SSMinT tools are easy to build and deploy for new developers. The Gradle build tool has been adopted and configured to assist with downloading dependencies, configuring either Eclipse or IntelliJ development environments, and building an artifact for easy deployment. Users should install both Java 8 (available from Oracle's website) and Gradle (included in many operating system package managers for Linux distributions, available through Homebrew in Mac OS X, or downloadable from http://gradle.org).

Users create "tasks" in Gradle to accomplish objectives such as building the application and creating a distributable binary. SSMinT includes a build task that downloads and installs depen-

dencies, builds the applications, and produces JAR files that includes all Java code and dependencies (often referred to as a fat JAR). The *build* task is invoked by executing the command *"gradle build"* in the working directory of the SSMinT project. Task name *build* can be replaced with other task names. For example, a developer may generate an Eclipse project by using the *eclipse* task. The *build.gradle* file contains SSMinT's configuration and is the canonical reference for tasks.

3.6.3 Running the Tools

The Gradle *build* task produces executable JAR files for automated keyword finding and learning on documents using found keywords to produce semantic signatures, and calculating a matrix of semantic hits using semantic signatures. This provides similar functionality to the automated SSMinT tools. Executable JAR files are also created for performing both proposed indexing methods proposed in this thesis. These JAR files contain all dependencies and application code required for distributing and executing the tools.

An example of executing the Indexing tool in the bash terminal on Mac OS X, Linux, and Windows 10 (with bash installed) is included in Figure 3.3. Users can easily pass parameters to configure details of the indexing process; for example, this design allows experiments such as finding optimal parameters to be easily automated. Some JVM parameters for memory configuration and program execution may be required: -Xms for minimum heap usage, -Xmx for maximum heap usage, and -jar for specifying the JAR for execution should be kept in mind. Users are recommended to reference the JVM documentation for more details on these settings.

```
$ java -Xms20G -Xmx60G -jar ThreadedIndexing-0.1.jar
--threads 18 --percent 5 --signature-clusters 100
--number-of-ssds 2 --sources $(pwd)/data/
--output-folder $(pwd)/output/
```

Figure 3.3: Executing the threaded Indexing tool in the bash shell.

3.7 Brief Tour of the Source Code

The source is divided at a high level between the application source code and the unit tests. Efforts to divide up parts of the code into reasonable Java namespaces have been made. We begin with the application source code looking at important Java namespaces and follow with an example of the unit tests in the application.

3.7.1 Data Structures

SSMinT data structures are used throughout the code and are introduced first to aid understanding of the algorithms later.

Data Sources (org.kelcecil.thesis.data.datasources)

Text data is found in a wide array of sources such as plain text files, databases, or HTML fetched from the internet. The details of obtaining the data should be hidden from code that is only concerned with processing text. Many sources provide some structure or additional information with the text. Parsing or transforming the data will often be necessary to remove structure or metadata that provide no value to our tools.

Obtaining text data is encapsulated into a concept called a *Data Source*. DataSource is an abstract Java class that represents a source such as a document or remote URL to be fetched. The developer can extend this class to obtain the data using any implementation required. Two implementations have been provided in the *org.kelcecil.thesis.data.datasources.impl* namespace: a plain text data source called *FileDataSource* and a source that parses content from a Reuters' NewsML file called *ReutersXMLDataSource*. Possible future implementations could include a DataSource that fetched the content of a website, a DataSource that parses and understands Word documents, and more. Extending DataSource for new implementations of obtaining text data is highly recommended to ensure easy integration with existing tools and algorithms.

Keyword Descriptor and Semantic Signature (org.kelcecil.thesis.data)

There are two primary data structures for holding information regarding keywords and vectors: *KeywordDescriptors* and *semantic signatures*. Both of these ideas are represented as data structures in namespaces in the *org.kelcecil.thesis.data* namespace. These classes both implement the *Serializable* interface and can be easily written to a file using the methods in *JsonUtils* or any other method supporting *Serializable*.

Stop Words (org.kelcecil.thesis.data)

The *StopWords* class is provided to find if a candidate word is a stop word. A static method isAStopword(String) provides a simple Boolean answer to finding if a word is a stop word. Words are loaded from a simple text file at "/resources/stopwords/default-words" and can easily be added just by inserting a word into the file.

Word Weight (org.kelcecil.thesis.data)

The WordWeight structure exists to associate a computed weight with a particular word in a document and to make sorting words with weights attached easier by implementing the *Comparable* interface to allow sorting using Java's standard libraries.

DocumentSSD Matrix (org.kelcecil.thesis.data.common)

The *DocumentSSDMatrix* data structure is intended to provide an easy-to-use structure for recording semantic hits and easily retrieving the columns or rows of the matrix. The structure can be initialized with lists of documents and Semantic Descriptors or initialized to be empty. The dimensions of the structure are increased at will, so the dimensions do not need to be known prior to initialization.

The *DocumentSSDMatrix* class provides several options for retrieving information from the

data structure. Single values may be obtained given a document and semantic signature name. A vector of semantic hits given a document or semantic signature name (using the getVectorForKey(String) method) can be obtained for clustering data. Data sources (or documents) that have had zero hits (getSourcesWithNoHits()) can be obtained to provide insight for an incremental learning algorithm.

The structure also includes methods to assist in pruning semantic signatures. The highest scoring semantic signatures for various contexts can be obtained using the getHighestSSDsFromClusters(int, int) method. This method performs singular value decomposition on the matrix and finds the sum of squares for the semantic signature component matrix. The columns (semantic signatures) of the DocumentSSD Matrix are clustered to attempt to find signatures that capture similar concepts.

Vector (org.kelcecil.thesis.data.common)

The *Vector* class is a collection of floating point numbers (Java's *double* type) as well as an identification tag to provide clues as to the origin of the instance. Lists of vectors are used for passing data into K-Means for clustering.

There are several other additional Vector utility classes in addition to the data structure. The *Vectors* class provides utilities to calculate the average of a list of *Vectors* as well as generating random vectors. *VectorFilter* provides methods for filtering undesirable signatures as identified in Kota's work in the automated learning tools.

WeightMatrix (org.kelcecil.thesis.data.common)

The *WeightMatrix* data structure accepts a *SemanticSignature* and a document to create a matrix containing the weight calculations between the words in the semantic signature keyword set. This structure is primarily for use for learning and generally not user-accessible.

3.7.2 Distance Measures (org.kelcecil.thesis.distance)

Two distance measures are implemented in the new toolset: Euclidean distance and cosine distance. Cosine distance is the preferred distance measurement in most cases, but Euclidean distance is provided for completeness.

3.7.3 KMeans (org.kelcecil.thesis.clustering.kmeans)

SSMinT includes an implementation of the K-Means clustering algorithm. The algorithm accepts a list of vectors and the number of clusters desired.

The algorithm returns a list of instances, the *KMeansCluster* class. An instance of *KMeansCluster* includes a list of vectors representing the members of the cluster as well as an instance of a *Centroid*, a special *Vector* with additional information about the cluster.

```
List<Vector> conceptVectors = getVectors();
KMeans clusterMethod = new KMeans(conceptVectors);
clusterMethod.setNumberOfClusters(numberOfClusters);
List<KMeansCluster> clusters = clusterMethod.run();
```

Figure 3.4: Example invocation of the KMeans clustering algorithm.

3.7.4 Preprocessors (org.kelcecil.thesis.preprocessor.text)

An interface is provided for creating new text preprocessors as well as two example preprocessors: a suffix stemming preprocessor and a synonym preprocessor. New preprocessors can easily created by implementing the *TextPreprocessor* interface in a class.

3.7.5 Weight

The following classes are used for calculating weights in SSMinT. Extra precautions have been made during development to provide code that is easy to understand, test, and debug.

TF-IDF (org.kelcecil.thesis.weight.tfidf)

SSMinT implements the Term Frequency-Inverse Document Frequency (TF-IDF) to be utilized in the automatic keyword selection. There are three methods of computing the term frequency for TF-IDF: Boolean computation, logarithmic computation, and an augmented frequency intended to avoid a bias toward larger documents in a corpus. All three are implemented in SSMinT.

Learner Word Weight (org.kelcecil.thesis.weight)

The *LearnerWordWeight* class implements the logic for computing keyword vectors found in documents. The class is initialized with a SemanticSignature Java object, the document name, and a section of the document to be evaluated. The weight matrix is processed from the document snippet provided, and the output is a *Vector* representing the semantic signature.

Word Distance Weight (org.kelcecil.thesis.weight)

The *WordDistanceWeight* class calculates weights between a specified word and other words that occur within a document window prior to and after the specified word. This structure is used in the TF-IDF computations and unlikely to be used directly.

3.7.6 Utility Methods

The *org.kelcecil.thesis.util* namespace contains several utility classes for various data structures used in SSMinT. These classes aid in merging multiple *DocumentSSDMatrix* objects, perform mathematical operations on data structures used by dependencies, help to simplify generating JSON for writing files, and more.

3.7.7 Processors

Part of the objective in this rewrite is to create a library that can be used to build a wide array of potential applications: web-based investigation tools, advanced guided user interfaces, or experiments in using distributed computing tools to process large amounts of data. Reuse of the higher process requires a separation between the process code and the user interface code that initializes the process.

The classes in the *org.kelcecil.thesis.process* namespace implements the automated keyword tool, automated learning tool, hits processor, and experimental indexers.

3.7.8 Main Classes

Main executable classes that serve as the entry points for the processors can be found in the *org.kelcecil.thesis.exec* class. These classes should strictly exist as some sort of interface to the user; for example, this tool set provides command line interface tools to parse options passed by the user to the program.

3.8 Test Code

An objective in the rewrite of these tools is to design tools that are high quality and are tested as best they can be. Efforts have been made to test parts of the code by writing *unittests*. Unit tests are methods that set up and invoke a method in order to verify the correctness of the output. We look at an example from the LearnerWordWeightTest to demonstrate.

Figure 3.5 shows a unit test that verifies the output of a simple input. Lines 3 through 9 configure the required state to pass to the class initializer. Lines 11 and 12 process the input and retrieve the result. Lines 13 through 15 verify that the output is what was expected.

The benefits of unit tests are worth repeating. These unit tests help to bring situations to a

```
@Test
public void testPopulateWordCacheSimply() {
    String snippet = DocumentUtils.
       filterPunctuationFromDocument("Apples␣make␣for␣
       excellent␣ale.");
    List<String> listSnippet = Arrays.asList(snippet.split("␣
       "));
    KeywordDescriptor descriptor = new KeywordDescriptor();
    descriptor.addWord("apples");
    descriptor.addWord("excellent");
    descriptor.addWord("ale");
    SemanticSignature signature = SemanticSignature.
       getSemanticSignatureFromKDF(descriptor);

    LearnerWordWeight weight = new LearnerWordWeight(
       signature, "", listSnippet);
    Map<Integer, String> cache = weight.getCache();
    assertThat(cache.get(0), is("apples"));
    assertThat(cache.get(3), is("excellent"));
    assertThat(cache.get(4), is("ale"));
}
```

Figure 3.5: Example of a unit test verifying the behavior of the calculation of the word weight.

developer's attention that a code change may have lead to a regression.

3.9 Future Improvements

These new tools have many improvements over the previous generation of tools, moreover there are many new possibilities opened by the new tools. There is also additional work from which the new tools would benefit. Following is a list of a few possibilities:

- New GUI tools that utilize the new back end Java code.
- Implementing the multi-threading as a distributed network application to fully take advantage of a cloud computing environment.

- Increased unit test coverage for improved regression protection.

Chapter 4

The Bulk Indexing Method

4.1 Overview

We now introduce a possible method for indexing documents. This methodology borrows from previous work on the SSMinT project and looks for observations that will help to improve the process.

4.2 Method Overview

4.2.1 Objective

The first method of indexing was conceived with certain objectives. Sorting through large numbers of documents can be difficult for a human. The process is prone to bias in the human's interpretation of the document, and a person's mind can easily wander off task when performing this tedious sort of work. An ideal indexing algorithm can sort documents into groups based solely on the document text, while avoiding intensive computation on the entire data set. An algorithm might save time if we were able to learn from a subset of the data and maximize the effectiveness of indexing by selecting the best semantic signatures learned from the set.

We propose an indexing method that attempts to satisfy these objectives. The "bulk indexing" method is named in reference to the algorithm's learning data selection process. Several experiments will be described and analyzed to determine the effectiveness of the algorithm.

4.2.2 Algorithm Overview

We briefly describe the process of indexing a data set using the bulk indexing method. The algorithm will be described in greater detail following the overview.

A training set is randomly selected from the documents in the data set. These documents are provided to the automated keyword selector to find sets of keywords using the TF-IDF algorithm. The learner scans the training set for the keywords and constructs semantic signatures from the files. These semantic signatures are then used to scan the entire corpus and construct a matrix with documents and semantic signatures

The document/semantic signature matrix is decomposed using singular value decomposition in order to find the component matrix representing the semantic signatures. We compute a score for each semantic signature to evaluate the strength of the semantic signature. The rows representing the semantic signatures are clustered to find signatures that capture similar semantic content. The top signatures based on the computed score are taken from each cluster in an effort to select an assortment of semantic signatures capturing different topics.

The document hits for these top signatures are kept in an aggregate document/semantic signature matrix. The aggregate matrix is inspected to find a new set of random documents that have no hits using the high performing semantic signatures that have been kept. We continue to repeat the process until each document has at least one hit in the aggregate document/semantic signature matrix.

4.2.3 The Algorithm

We will now examine the algorithm in a more in-depth fashion.

Finding Keyword Sets

Finding sets of keywords that describe concepts within documents well is extremely important to an unsupervised learning algorithm. Without proper keyword sets, our algorithm would have no reliable way to find concepts in documents that would allow it to distinguish documents with differing content from each other.

To address this, we use the TF-IDF algorithm as described in Section 2.7.3 to find highly-ranked keyword sets that are prominent in a selected document, and distinguish it from other documents. Kota's [13] work with the automated semantic signature discovery tools found that the number of top keywords selected should be sufficiently large in order to find words that describe general concepts instead of words that describe more specific proper nouns. The indexing algorithm calculates the TF-IDF weight for every word that is not a stop word in the document and selects n words with the largest TF-IDF weights. Five words are used in our experiments. These words are used as the first words of individual keyword sets. Second and third words for each set are individually selected using the window weight function in Figure 2.1. The words with highest weights relative to the first word of the set are used. The value of constant a in the window weight function is 2.

Scanning Documents for Semantic Signatures

The keyword sets found using the technique described in the previous section are then used to scan our randomly selected training documents to look for occurrences of keywords that can be used to create semantic signatures. We scan through each word in each document and check to see if the word is one of the keywords in a Keyword Descriptor. If a keyword belonging to a

Keyword Descriptor is found, we scan the learning window to find other keywords in the Keyword Descriptor. Distance measures are computed between the keywords found within the window. These distance measures are stored in a term matrix, which is flattened into a vector after searching to the end of the learning window is complete. The vector is kept with other vectors generated from a Keyword Descriptor until all training documents have been scanned.

All vectors found for each Keyword Descriptor are clustered into three clusters using K-means to separate vectors representing different concepts. The radius of each of these clusters is calculated using the cosine distance measurement, and used when looking for semantic "hits". Vectors, the centroid, and the radius from each of these clusters are used to create semantic signatures that will be used in the next section.

Further details of this step may be found in Para's [22] foundational work on the original SSMinT Learner Tool.

Finding Semantic Hits

We use the semantic signatures generated using the method described in the previous section to search all of our data in the corpus for semantic "hits". This is similar to finding a word in a document when using the Bag-of-Words approach to data mining.

A semantic signature is selected, and a document is scanned for the keywords in the signature to produce a vector. The distance is measured between the found vector and the centroid of the selected semantic signature. This distance between the vector and the centroid of the semantic signature is compared to the radius of the selected signature. It is a hit if the distance is *less* than the radius when using the Euclidean distance measurement and a hit if the distance is *greater* than the radius when using cosine distance measurement. Documents with semantic signature hits are incremented in a term-document matrix (described in Section 2.7.4).

We repeat this process for every semantic signature discovered using every document in our data set.

Finding the Best Signatures

The hits found using our learned semantic signatures provides the ability to reduce our semantic signatures to the best-performing, newly-discovered signatures allowing us to perform dimensionality reduction through term-space reduction [26].

We must first associate a weight with each semantic signature to evaluate its effectiveness in describing our document. We accomplish this by computing the singular value decomposition of the term-document matrix. The singular value decomposition produces two component matrices: one component matrix represents the trend in our documents and the other transposed matrix represents the trend in our semantic signatures. The component representing semantic signatures contains a column for each signature. We derive weights for our semantic signatures from the singular value decomposition.

Kota [13] found during her research that the weight from the component of the singular value decomposition had a significant shortcoming. Similar semantic signatures often had the highest weights and would cause redundant signatures to be selected. Kota overcame this problem by using K-Means to cluster the semantic signature columns of the term-document matrix to find signatures representing similar concepts. The best n signatures based on the weights calculated from the signature component matrix are selected from each cluster. The remaining semantic signatures are discarded.

Determining New Training Documents

Our refined set of semantic signatures are evaluated against all documents in our corpus. Documents that have had no semantic hits are found, and the new training documents are randomly selected from these documents. The process repeats itself until all documents have at least one semantic hit.

4.3 Experiment 1: 15 Days of Consecutive Documents

4.3.1 Objective

The objective in our maiden experiment is to perform our indexing process as described in Chapter 4 on the 15 days of documents in the Reuters corpus. We will compare our clustering to the categories provided with the Reuters data set as a point of comparison.

4.3.2 Experimental Setup

We began with 15 days of Reuters documents (totaling 30,303 documents), and randomly selected five percent of the documents for learning. In the algorithm, the number of documents taken for learning each round is the minimum of the calculated five percent of the initial number of documents in the corpus and the number of documents left with no training sets.

The training documents are then run through the Automated Keyword tool. TF-IDF is performed for each document in the training document set using the entire corpus for the inverse document frequency. Weights are calculated for each word in the document. The weights are sorted in descending order and the ten highest weighted words are selected as the first word in new keyword sets. The second and third words for a keyword set are selected using weights relative to the first word in the keyword set, but only the two highest weighted words are kept to control the number of keyword sets generated.

These keyword sets are used to scan all of the training documents for document vectors. The collected vectors are clustered using k-means with three clusters to separate potential semantic contexts. The vectors from these clusters are used to create separate semantic signatures. No semantic signatures are filtered at this step.

We use the semantic signatures found in the previous step to scan the entire corpus for hits. The hits are tallied in a term-document matrix. We then evaluate the set of semantic signatures. Our

experiments use 100 clusters and take the two semantic signatures with the highest quality score.

The refined set of semantic signatures are evaluated against the entire corpus to documents with hits and those that still have no hits. Five percent of the initial number of documents is selected from the list of documents with no hits. This process repeats itself until there are no documents remaining without at least one hit.

The output of the experiment is a CSV file representing a term-document matrix. The documents are the Reuters articles used in the experiment, and the semantic signatures are the signatures that are discovered iteratively during the experiment.

4.3.3 Results and Observations

We begin by clustering the term-document matrix rows using k-means clustering with k = 50 to find similar documents. Figure 4.3 shows representation of Reuters classifications in our 50 k-means clusters using a heatmap. The scale is white to yellow to red: White means that no documents for a category are in a cluster while red means that most if not all of the documents for a category are in a cluster.

We first look to Figure 4.2 to see how many documents are classified over iterations of our indexing algorithm. We can see that our algorithm found semantic signatures that were hit against a larger number of documents in the first iterations of the algorithm. semantic signatures found later in the algorithm hit on fewer documents and are probably specific to particular documents.

Table 4.1 lists clusters and Reuters classifications that have strong representation in the same clusters. The categories in the clusters are largely topics that work well together. This can be somewhat attributed to Reuters documents possibly having multiple classifications assigned to individual documents.

To validate the algorithm, it is desirable to present the documents to human reviewers to cluster for comparison. Evaluating over 30,000 documents within clusters is not feasible by humans, so a different validation technique was chosen. To accommodate this, we cluster the term-document

Cluster	Reuters Classification
X29	Equity Markets
X17	Trade/Reserves Government Finance Economic Performance Monetary/Economic
X37	Legal/Judicial Ownership Changes Management
X33	Insolvency Liquidity Economic Performance Inflation/Prices Consumer Finance Housing Starts
X18	Output/Capacity Leading Indicators
X40	Production/Services
X8	Share Listings

Table 4.1: Table of samples periods from the Reuters data set over three months

	Indexer	Person 1	Person 2	Person 3
Indexer	1	0.064	-0.008	-0.08
Person 1	0.064	1	0.208	0.016
Person 2	-0.008	0.208	1	0.088
Person 3	-0.08	0.016	0.088	1

Table 4.2: Rand Index from Human Cross Validation with no documents removed

matrix using the K-means clustering algorithm with k=50. The term-document matrix generated using the indexing algorithm is clustered row-wise to cluster like documents together. Five clusters are chosen at random and five documents are randomly chosen from these selected clusters. The selected 25 documents are presented in Table 4.3. These documents were provided to three volunteers to cluster into five groups of five documents. The volunteers were purposely selected to have differing backgrounds in tech, medicine, and finance. The human-generated clusters were compared using the Adjusted Rand Index (Section 2.7.2).

A matrix representing the Adjusted Rand Index between the indexing algorithm and each of the three humans can be seen in Table 4.2. Person 1 and Person 2 appear to agree with each other the most: The Adjusted Rand Index of their clusterings is a 0.208. Person 1 and Person 2 both created a cluster from documents that contained outlier documents involving foreign affairs from several of the clusters. There is one situation in which a human agrees more with the indexer than another human, but generally, the humans agree more with one another than they agree with the indexing algorithm.

The sample documents from the clusters and the poor performance during our human validation demonstrates poor performance of our algorithm. Previous successes using the semantic signature methodology suggested that our algorithm needed additional investigation.

4.4 Experiment 2: Term Reduction

4.4.1 Objective

Why might the algorithm be having such a difficult time clustering the documents? Recall that document clustering algorithms require a large number of attributes for documents to be effective [29]. We can compare attributes to individual semantic signatures. All documents have at least one semantic signature. Only 9,899 documents have two or more semantic signatures, while 2,173

File Name	Title
Cluster 2	
27110	UKRAINE: Ukraine launches new hryvna but distribution slow.
110709	MALAYSIA: Malaysia physical palm oil prices - Oct 11.
127309	SA: NetManage Inc Q3 net tumbles, shr $0.01.
143252	RUSSIA: Kremlin aides have limited access to sick Yeltsin.
191597	USA: Medicore Inc Q3 full results.
Cluster 12	
54768	UK: BoE's George warns of dispute with Germany, France.
83237	MALAYSIA: Malaysia July rubber output up 13.6 pct.
111857	USA: U.S. September retail sales up on autos surge.
127128	USA: NYSE closing averages.
212551	INDIA: Indian foodgrains prices - Delhi - Nov 23.
Cluster 30	
126588	PERU: European Union to donate $42.5 million to Peru.
126795	USA: U.S. urges Americans in Pakistan to mind security.
126835	USA: Albuquerque, N.M., deal rated A1 by Moody's.
160082	USA: Pacifica Loan notes, Calif., at Baa - Moody's.
209985	USA: New Castle, N.Y., $2.92 mln deal Aa1 by Moody's.
Cluster 34	
83354	FRANCE: Lebanon says Israel does nothing to end unrest.
145857	AFGHANISTAN: Dostum warplanes hit Taleban positions - spokesman.
159250	USA: FORECAST-Fed seen refraining from Nov 13 action.
209704	ARGENTINA: Bulgaria buys 300,000 tonnes Argentine wheat - trade.
225823	SWEDEN: S&P assigns claims-paying rating to Norway's Vesta.
Cluster 48	
68668	UK: Key stock and currency market movements, Sept 20.
98461	UK: Senior Conservative defects to UK Referendum Party.
143358	MEXICO: Mexico to sign new pact Saturday - gov't.
144997	PHILIPPINES: Belle sees 1996 net at 870 mln pesos.
226461	SWEDEN: Sydkraft 1996 profit seen 2.4 bln skr.

Table 4.3: Documents selected for human cross validation with no documents removed

documents have three or more semantic signatures. Figure 4.4 demonstrates how quickly the number of documents with increasing number of semantic signatures drops off with our scheme. Does increasing the number of semantic signatures for documents improves the performance of our indexing algorithm?

4.4.2 Experimental Setup

We have already found semantic signature hits in the previous experiment, so we will use the term-document matrix found in the last experiment. We began by removing all documents that have less than three attributes (individual semantic signatures) from the data set. We performed dimensionality reduction by removing semantic signatures that have no hits in the remaining documents. The remaining documents are clustered using k-means similar to the last experiment.

4.4.3 Results and Observations

We began by clustering the documents using the k-means algorithm. The number of clusters in the last experiment was selected based on the number of categories in the Reuters corpus, but this left many sparsely populated clusters. We calculated the value of the Dunn Index for numbers of clusters between 20 and 60 clusters. The Dunn Index for 33 clusters has the highest score, and thus is used in our clustering.

We again performed human validation between the indexing algorithm and two people (as the third person was unavailable to participate). The clusterings of the algorithm and people were again compared using the Adjusted Rand Index. The results are shown in Table 4.5. The largest score is 0.16 between the indexing algorithm and Person 1. The worst score is between Person 1 (medical background) and Person 2 (finance background); both people agreed more with the indexing algorithm than they did with each other. Several of the documents are simply tables of numbers, and some are very short news briefs; documents such as these are subject to human

File Name	Title
Cluster 1	
795562	Cosmetic Group sees Q2 loss
795711	USA: Windmere-Durable sees sales, earnings accelerating.
780559	JAPAN: Saudi lifts Asia Sept Super Light formula 10 cts.
799786	USA: Aviva plan seeks to buy $5 mln US property
782507	USA: Vectra is seeking buyer for company.
Cluster 4	
796568	USA: U.S./LATAM SPOT CRUDE TRADES - AUG 13 1711 EDT
793247	UK: EUROPE RAW COTTON AWAITS DIRECTION
808139	RUSSIA: Mir crew change crashed computer part, do tests
792646	USA: U.S./LATAM spot crude trades - Aug 12 1557 EDT.
778948	USA: U.S./LATAM spot crude trades - Aug 05 1537 EDT.
Cluster 18	
783609	INDONESIA: Astra to seek national car privileges.
798478	LITHUANIA: PRESS DIGEST - LITHUANIA - AUG 14
790843	UK: EXCISE DUTIES BOOST UK PPI, UPSTREAM INFLATION WEAK.
794535	GERMANY: German BHW drops plans for Postbank stake.
804150	MALAYSIA: Tenaga sees good results this year.
Cluster 23	
792834	USA: TABLE- N-Viro International Corp Q2 shr up.
809500	MOROCCO: Morocco's bourse closes flat.
795414	USA: TABLE - Bradley Pharmaceuticals Q2.
777774	USA: TABLE - Zonagen Inc Q2 shr loss widens
795483	USA: TABLE - PPT Vision Inc Q3 earnings fall.
Cluster 29	
801776	USA: Eastwind Group Inc Q2 net loss.
780881	USA: TABLE - Strategic Distribution Q2 loss.
800124	USA: TABLE - Atchison Casting Corp Q4 net rises.
785766	LEBANON: Five killed in south Lebanon violence.
799145	USA: TABLE - Ezcony Q2 oper net loss.

Table 4.4: Documents selected for Human Cross Validation having more than 3 semantic signature hits

	Indexer	Person 1	Person 2
Indexer	1	0.16	0.088
Person 1	0.16	1	-0.008
Person 2	0.088	-0.008	1

Table 4.5: Rand Index from Human Cross Validation with a reduced set of documents

interpretation and can be very difficult to sort based on the limited information available.

4.5 Conclusions

Our results in both experiments for our bulk indexing method demonstrate that our method does a poor job of clustering documents. Furthermore, the algorithm takes a great deal of time to find over 17,000 semantic signatures only to provide such a poor clustering. Our method is unsuccessful, but the experiments do uncover some interesting observations.

The most surprising discovery when testing our indexing algorithm is the very low number of documents with multiple semantic signature hits. Our bulk indexing method generates over 17,000 semantic signatures that are the result of term reduction using singular value decomposition, but the number of documents with more than one semantic signature hit is surprisingly low. This fact makes clustering our documents impossible. Our documents need multiple semantic signature hits in order to provide the information that a clustering algorithm can use to group documents in a manner that makes sense. The first improvement we must make is for our indexing to ensure more semantic signature hits for each document.

What about the documents that did have multiple semantic signature hits? Recall that our indexing method performs term (semantic signature) reduction on each iteration of our algorithm to find the best semantic signatures for a given iteration of learning. The objective for term reduction is to only represent the semantic signatures that provide the most value to identification given a set of terms, but mixing terms that represent different subjects and contexts does not allow us to do a

fair comparison between terms. Our objective should not be to find the "best" terms that describe all documents, but should be to find the best terms *for a particular subject*.

In chapter 5 we introduce a second indexing method that builds on these observations to create an improved indexing method that is suitable for further experimentation.

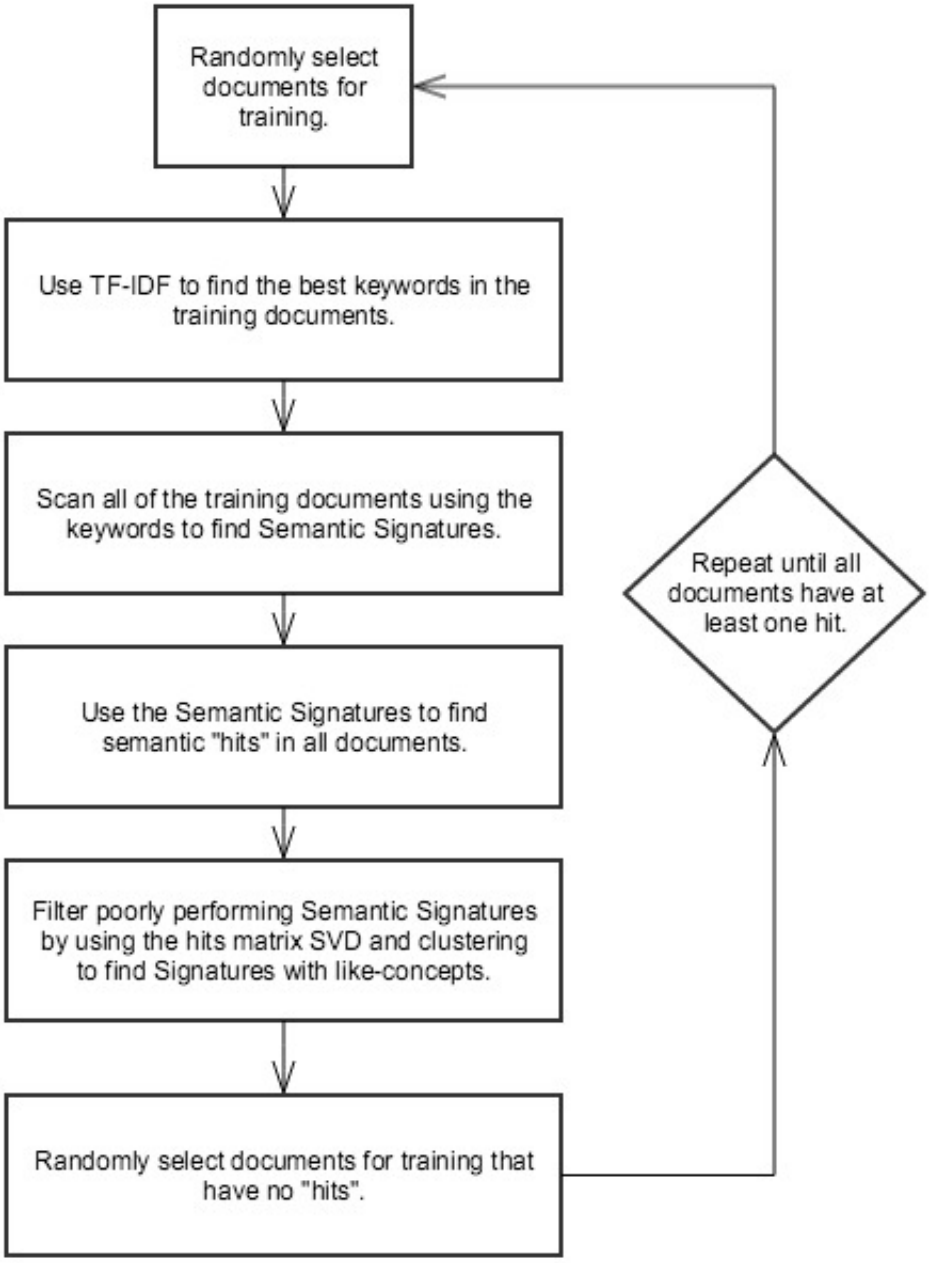

Figure 4.1: An overview of the bulk indexing algorithm

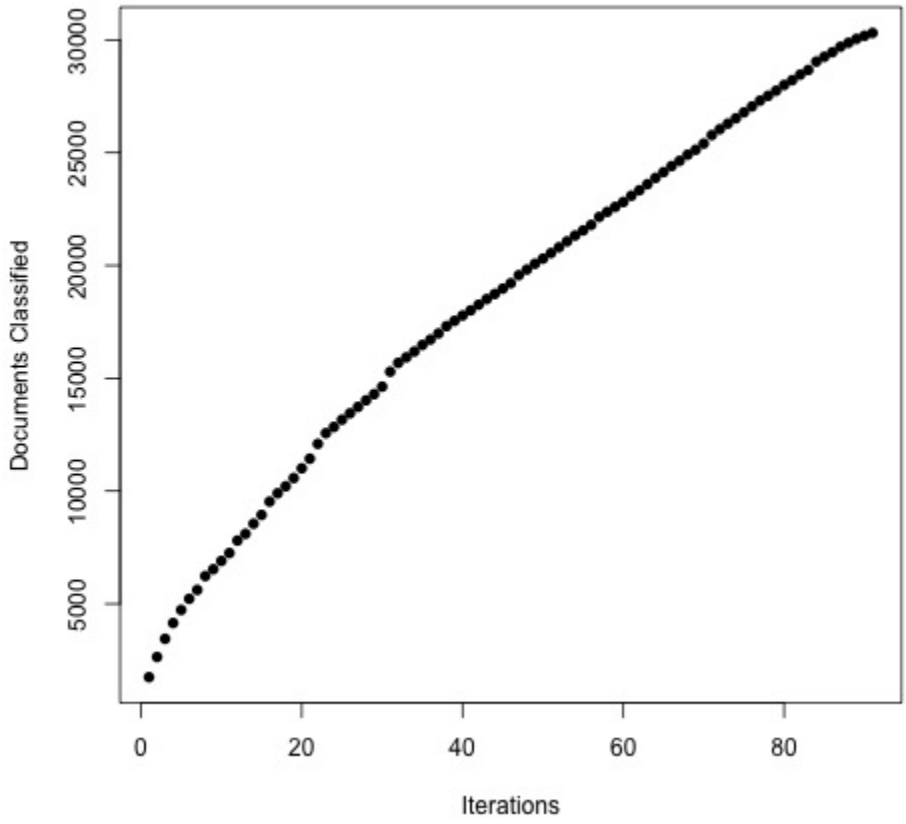

Figure 4.2: Documents classified per iteration of indexing

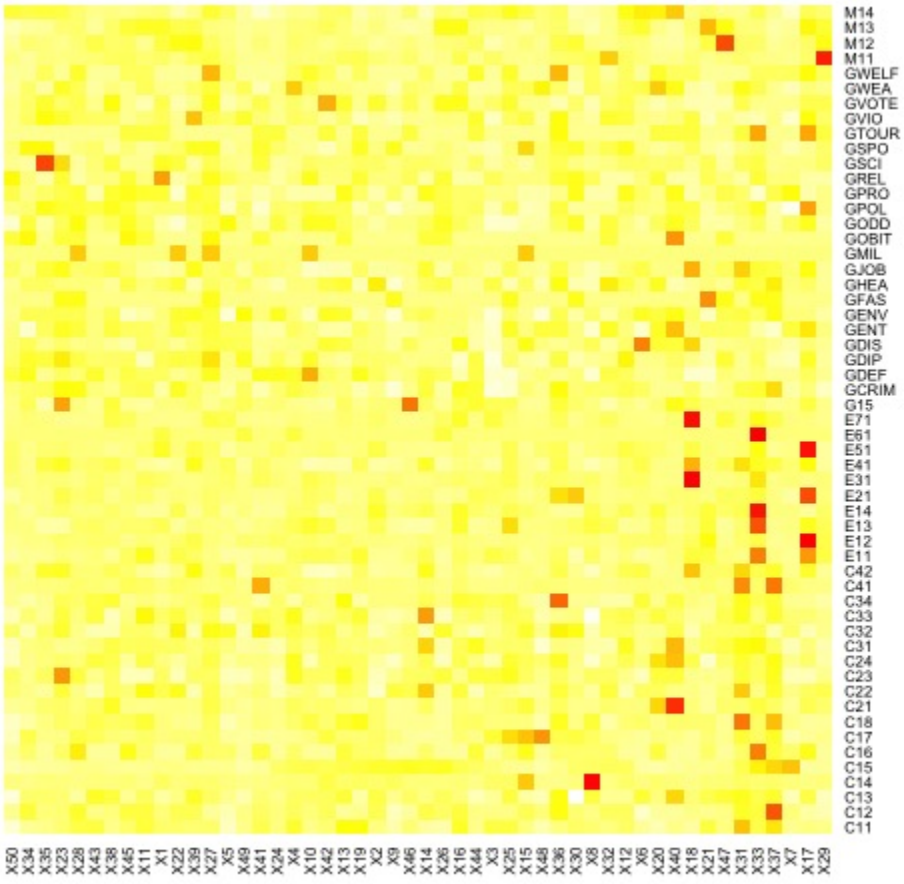

Figure 4.3: A heatmap for representing 50 clusters for documents

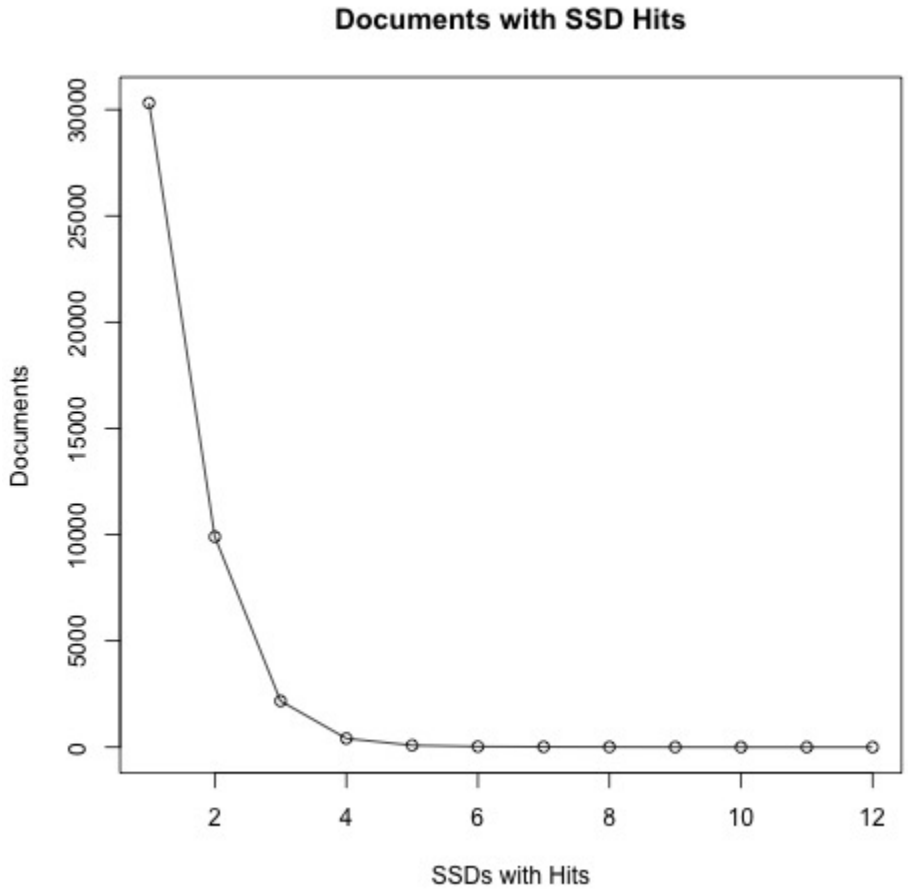

Figure 4.4: Semantic signature hits for number of documents

Chapter 5

The Iterative Indexing Method

5.1 Objective

Our experiments in the previous chapter demonstrate flaws when using our bulk indexing method for clustering documents, but we believe that performance can be improved. A method that focuses on finding and comparing semantic signatures for documents that are related to each other and finds overlapping terms to provide more context for our clustering algorithms could provide a more meaningful clustering.

We now present a new method for indexing documents that is designed to find a series of related documents for learning semantic signatures that could be reduced to provide a refined set of signatures that describe a particular topic.

5.2 Method Details

5.2.1 Overview

Our experiments on the bulk index method demonstrated several issues with the method that would lead to a better indexing algorithm. This improved algorithm would have more semantic signature

hits per document and term reduction by comparing terms from related topics. An easy method of finding documents with related topics is simply to pick a document at random, learn from the document, and use what we've learned to discover other documents in our data set that may be related. We named our new algorithm the iterative indexing method.

5.2.2 Algorithm Overview

We begin by selecting a document at random from the data. We search this document for keyword sets using TF-IDF. The documents are scanned for semantic hits using these keyword sets, and we make note of documents that have semantic hits that have previously not had any.

We repeat the process of finding keyword sets using TF-IDF and finding semantic signatures using the documents that had the new hits. Semantic signatures for each iteration are kept. The process repeats until no new documents have new hits. We take the semantic signatures accumulated for these iterations and compute a term-document matrix for the data set. We calculate the singular value decomposition to find the component matrix for each of the semantic signatures and refine the set of semantic signatures.

5.2.3 The Algorithm

Finding Important Keywords

We begin the algorithm by selecting a single document from our data set at random. We use a TF-IDF algorithm with logarithmic term frequency as described in Section 2.7.3 to find important words in the selected document. We use the selected document for the term frequency and the entire data set for the inverse document frequency. The n highest scoring words using TF-IDF are selected as the first words in sets of keywords, with the second and third words selected by using the window function score described in Figure 2.1.

Documents that are used for initial keyword discovery are added to a list of used documents to

prevent these documents from being selected for learning after the initial discovery.

Searching Documents for Semantic Signatures

Our first step is to search documents for semantic signatures. This process is the same as described in Section 4.2.3.

Generating the Term-Document Matrix for Semantic Signatures

Our next step is to find documents that may be related using the newly discovered semantic signatures.

Finding and Reducing Semantic Signatures

Following the preceding steps, we have created a term-document matrix. The term-document matrix for the current iteration is compared to the term-document matrix of the previous iteration to find documents that did not have hits previously, but now have new semantic hits. These documents are now candidates for learning. We take these newly hit documents that are not in the list of used documents and repeat the steps of the three preceding sections to find new keywords and semantic signatures. Semantic signatures for each iteration are accumulated, and the steps are repeated until unused documents have new semantic hits.

The semantic signatures that have been accumulated are used to generate a hits array, and we perform dimensionality reduction similarly to the method described in Section 4.2.3. This is performed to keep the strongest semantic signatures in our discovery. The rationale is that discovering new documents through the semantic signatures of another document will allow us to learn from a strand of related documents. The strongest semantic signatures are kept in a list while the rest of the semantic signatures are discarded.

Iterating Until Finished

The processes described in the previous sections are repeated to find sets of semantic signatures that classify additional sets of documents. Documents that are used for learning keywords are marked to avoid using the documents again. Failure to do so can result in cyclic file usage when searching for new documents as described in Section 5.2.3. Our process is repeated until every document has at least one semantic signature hit in our set of refined semantic signatures.

5.3 Experiment: Large Documents

5.3.1 Objective

Our first indexing method did not provide promising results, but the work did provide observations that helped to drive to our new indexing method. The objective for this experiment is to demonstrate that the new indexing method is a promising lead to future research.

5.3.2 Experimental Setup

This experiment follows the indexing algorithm precisely as described in Section 5.2.3. We observed in our last experiments that both humans and our algorithm struggled to find differences in short documents and news briefs that consists only of tables of numbers. Our experiment avoids this problem by using 4,973 documents that contains 500 or more words in the document. These documents were selected at random from the Reuters corpus and do not represent a particular range of time.

The rows of the term-document matrix representing semantic signature hits on a document were clustered using k-means in the same manner as our previous experiments to group documents with similar semantic signature hits together. This ideally provides self-organized documents that are similar to other members of the cluster. We used k-means clustering with 200 clusters, and we

Document Name	Document Title
292272newsML.xml	SOUTH KOREA: Seoul says North Korean apology not good enough.
288653newsML.xml	USA: U.S. Treasury okays Cargill food export to N.Korea.
289733newsML.xml	SOUTH KOREA: S.Korean president to detail policies on Tuesday.</title>
282646newsML.xml	SOUTH KOREA: After Pyongyang's apology, a hard slog lies ahead.
282031newsML.xml	SOUTH KOREA: Asian traders wary on Cargill barter with N.Korea.
285631newsML.xml	SOUTH KOREA: Seoul says N.Korea nuclear project "back on track".
280550newsML.xml	USA: U.S. mulls aid to N. Korea, key talks due soon.
281022newsML.xml	USA: U.S. mulls aid to N. Korea, key talks due soon.
287625newsML.xml	USA: Police continue probe in murder of Colorado child.
283089newsML.xml	USA: YEAREND - Imperilled deals await new U.S. foreign policy team.
273862newsML.xml	USA: YEAREND - Imperilled deals await new U.S. foreign policy team.

Table 5.1: Filenames and documents from Cluster 23

analyze this grouping to compare to our previous experiments.

5.3.3 Results and Observations

We can easily begin with a visual inspection of our clusters. Our large number of clusters (k = 200) makes inspecting each cluster somewhat difficult, but a subset of the clusters shows promise. Tables 5.1, 5.2, and 5.3 contain example clusterings from our experiment.

Table 5.1 contains mostly news briefs regarding foreign affairs of North Korea. We note that this cluster contains two sets of duplicate news briefs. This fact was unknown to us before this experiment and demonstrates clustering of identical documents.

One news brief regarding the murder of a Colorado child is an outlier in our cluster. A check of our data revealed that only one other news brief in our data set mentions this murder. This second news brief describes the child's parents returning to their home aboard their private jet and was included with other documents regarding Boeing and jets. We believe this outlier is a result of too few related documents.

Table 5.2 contains articles regarding the value of multiple countries' currency and some securities. Many clusters have news briefs regarding financial topics, but this cluster is a fine example

Document Name	Document Title
289701newsML.xml	JAPAN: Japan financial markets make gloomy start.
277971newsML.xml	USA: Dlr/yen ends near flat in US after move above 116.
290280newsML.xml	USA: Ford chief plays down impact of dlr-yen rise.
279786newsML.xml	USA: Dollar ends firmer in quiet U.S., above 116 yen.
290034newsML.xml	JAPAN: Japan financial markets make gloomy start.
277934newsML.xml	USA: Dlr/yen comes off US highs to end little changed.
285663newsML.xml	USA: Dollar gets boost from strong stocks and bonds.
282160newsML.xml	USA: US Treasuries remain tightly range bound at midday.
299225newsML.xml	USA: Dollar rallies after strong jobs report.
281109newsML.xml	AUSTRALIA: RTRS-Australian dlr regains ground as yen retreats.
280601newsML.xml	USA: Dollar climbs as Japanese officials remain silent.
299224newsML.xml	USA: Dlr ends strong in U.S. as stocks surge to record.
299313newsML.xml	USA: Strong jobs data pushes dlr broadly higher in U.S..

Table 5.2: Filenames and document titles from Cluster 35

of a cluster that brings together similar subtopics.

Table 5.3 contains almost entirely articles regarding Boris Yeltsin and how health issues in 1996-1997 were preventing him from leading Russia. This cluster does an impressive job capturing news briefs that cover his absence, however these are not the only news briefs that discuss him. In the cluster listed in Table 5.4, we again see mentions of Yeltsin. This cluster mostly contains news briefs relating to German Chancellor Helmut Kohl, but the items of interest are his work regarding Boris Yeltsin and NATO reform. We can see that many of these news briefs specifically describe Yeltsin in the context of working with Kohl and NATO, but only one document involves Yeltsin's health. This is a promising example of contextually sensitive text mining.

As before, human validation of these results is necessary. We tested this by again randomly selecting five documents from five random clusters, and we asked two people to form clusters to judge the effectiveness of our algorithm compared to humans. We compared the clustering of the algorithm to the clustering of both humans using the Adjusted Rand Index described in Section 2.7.2.

We found that both Adjusted Rand Index between our humans and our iterative indexing algo-

Document Name	Document Title
277725newsML.xml	RUSSIA: Lebed says Yeltsin still sick, should resign.
295667newsML.xml	RUSSIA: Yeltsin suffers new setback in struggle for health.
295669newsML.xml	Russia: Yeltsin battles with pneumonia in hospital.
298284newsML.xml	RUSSIA: Yeltsin said "satisfactory" despite pneumonia.
298830newsML.xml	RUSSIA: Gloom grows as no breakthrough in Yeltsin health.
295835newsML.xml	RUSSIA: Russia's Yeltsin in hospital again.
274717newsML.xml	RUSSIA: Yeltsin to meet PM, trains guns on tax-dodgers.
280457newsML.xml	NEW ZEALAND: PRESS DIGEST - New Zealand newspapers - Dec 31.
294135newsML.xml	RUSSIA: Yeltsin taken to hospital with signs of pneumonia.
298285newsML.xml	RUSSIA: Yeltsin illness renews concern about stability.
276547newsML.xml	RUSSIA: Russia's Yeltsin to meet Chinese premier.
275160newsML.xml	USA: Niche essential for professional practices, too.
295836newsML.xml	RUSSIA: Yeltsin illness renews concern about stability.
298286newsML.xml	RUSSIA: Yeltsin illness renews concern about stability.
298825newsML.xml	RUSSIA: Yeltsin faces more than three weeks out of Kremlin.
295817newsML.xml	RUSSIA: Yeltsin said "satisfactory" despite pneumonia.
295678newsML.xml	RUSSIA: Yeltsin stable in clinic, political outlook cloudy.
293178newsML.xml	RUSSIA: Yeltsin taken to hospital with signs of pneumonia.
295675newsML.xml	RUSSIA: Yeltsin in hospital with signs of pneumonia.
272818newsML.xml	RUSSIA: Yeltsin back in Kremlin, "ready for battle".
274101newsML.xml	Australia: Aboriginal boys out of jail after attack on MP.
272813newsML.xml	RUSSIA: Yeltsin "ready for battle" as returns to Kremlin.
295676newsML.xml	RUSSIA: Rival Lebed sees Yeltsin health troubles critical.
298631newsML.xml	RUSSIA: RIVAL LEBED SEES YELTSIN HEALTH TROUBLES CRITICAL.
272827newsML.xml	RUSSIA: Yeltsin to meet PM, trains guns on tax-dodgers.
274662newsML.xml	RUSSIA: Yeltsin meets PM, Moscow mayor, busy agenda ahead.
297771newsML.xml	UK: Surgeon believes Yeltsin has transient condition.
298961newsML.xml	RUSSIA: Russian newspapers ask - "Who's in charge?".
298816newsML.xml	RUSSIA: Yeltsin pneumonia unchanged, ordered to rest.
276587newsML.xml	RUSSIA: FEATURE - Russia still has rocky road to economic growth.
299264newsML.xml	USA: Morris says Clinton thought Powell could beat him.
282907newsML.xml	RUSSIA: FEATURE - Yeltsin ends 1996 on high note but problems abound.
295659newsML.xml	RUSSIA: Yeltsin in hospital, Kremlin says condition stable.
298838newsML.xml	RUSSIA: Yeltsin stable, talks to leaders by phone.

Table 5.3: Filenames and document titles from Cluster 56

Document Name	Document Title
284534newsML.xml	GERMANY: Kohl visits Yeltsin to further NATO reform.
285238newsML.xml	RUSSIA: Yeltsin moves to hunting lodge before Kohl talks.
286939newsML.xml	RUSSIA: Yeltsin, Kohl end talks on NATO.
286940newsML.xml	RUSSIA: Yeltsin welcomes Germany's Kohl for talks on NATO.
300766newsML.xml	GERMANY: Germany rebuffs stars' accusations over Scientology.
300160newsML.xml	GERMANY: Germany rebuffs U.S. stars' Scientology charges.
272930newsML.xml	RUSSIA: Main events in Yeltsin's political career.
296754newsML.xml	GERMANY: Kohl laments German unemployed rejecting jobs.
288356newsML.xml	RUSSIA: Russian army quits Chechnya but peace still shaky.
300410newsML.xml	UK: PRESS DIGEST - British business press - Jan 10.
289135newsML.xml	UK: Rocky road, little time for NATO and Russia.
286941newsML.xml	RUSSIA: Germany's Kohl arrives in Moscow to meet Yeltsin.
297452newsML.xml	GERMANY: Kohl keeps Germans guessing about his future.
286942newsML.xml	RUSSIA: Kohl sees NATO deal with Yeltsin this year.
297681newsML.xml	Germany: Kohl dismisses Hollywood stars' rebuke of Germany.
288327newsML.xml	RUSSIA: Kremlin slams report on return of Tsar's heir.
295837newsML.xml	RUSSIA: Yeltsin's health fails him again.
286964newsML.xml	RUSSIA: Germany's Kohl set to meet Yeltsin in Russia.
300192newsML.xml	GERMANY: New row erupts over Berlin Holocaust memorial.

Table 5.4: Filenames and document titles from Cluster 9

	Indexer	Person 1	Person 2
Indexer	1	0.568	0.496
Person 1	0.568	1	0.52
Person 2	0.495	0.52	1

Table 5.5: Rand Index from human cross validation using the iterative index method

rithm are much improved from our bulk indexing algorithm. Table 5.6 shows the documents used in the cross validation as clustered by the indexing algorithm. The clusters formed by our human cross-validators had Adjusted Rand Indexes of 0.568 and 0.496, respectively, when compared with the original clusters from the iterative indexing algorithm.

Both humans successfully identified the contents of Cluster 19 containing news briefs regarding Boris Yeltsen and cluster 94 regarding bond market prices in the UK. The remaining clusters can be described as United States business news (Cluster 66), French business news (Cluster 135), and news briefs regarding European banking (Cluster 102.) The humans agreed with 4 and 3 of the 5 documents in Cluster 135 respectively. The humans diverged from the algorithm for the remaining documents. One human observed a trend in articles involving technology, while the other human (having a background in finance) proceeded to group remaining documents by types of investments. The comparison of groups between humans and the indexing algorithm is interesting. Both the humans and algorithm agree on documents that have obvious groupings but, at least in this case, diverge when the categories of the news briefs are more subjective.

Recall from our previous experiments our hypothesis regarding the number of semantic signature hits for documents. The majority of our documents were classified with a single semantic signature and thus contained no overlapping information that might help us identify contextually related information. This is corrected in our new iterative index algorithm. Figure 5.2 shows distributions for the number of documents that had a particular number of semantic signatures when the respective indexing algorithm finished. The distribution of our bulk indexing method is the left chart, and our iterative indexing method is on the right.

File Name	Title
Cluster 19	
299935	UK: West more confident about Yeltsin this time around.
295775	RUSSIA: This time, Russians believe the Kremlin on health.
272820	RUSSIA: Yeltsin back in Kremlin on Monday.
275936	RUSSIA: Yeltsin vows to fight wage delays, abuse of power.
277683	RUSSIA: Moscow to stage Russian-Chinese summit next April.
Cluster 66	
289086	USA: FULL TEXT - Raytheon to buy TI defense unit.
291189	USA: Pittsburgh struggles to shake blue-collar legacy.
279941	USA: FULL TEXT -Tandy closing computer stores.
277126	USA: Fund returns depend on taxes.
283915	USA: Fund investors can't ignore dividends.
Cluster 94	
272733	UK: Late bond market prices.
274548	UK: Late bond market prices.
283560	UK: Late bond market prices.
285106	UK: Late bond market prices.
298690	UK: Late bond market prices.
Cluster 102	
298103	HONG KONG: Moody's rates four Chinese commercial banks.
273394	BULGARIA: Bulgaria c.bank liquidity drain helps battered lev.
282902	POLAND: Poles part with their millions after money change.
288361	BULGARIA: Bulgaria small banks must merge to survive - banker.
286512	UK: UK pension funds thrived in '96 - WM Company survey.
Cluster 135	
273394	FRANCE: French bond sales seen up 20 percent in 1997.
294710	FRANCE: Ariane seeks 30 pct cost cut in new rocket order.
295638	COTE D'IVOIRE: AT&T partner out Ivorian phone bid - official.
280911	FRANCE: GAN has no comment on new state cash call.
286130	UK: WORLD BONDS - Supply to weigh down Germany, France.

Table 5.6: Randomly selected documents from indexed clusters

The majority of documents (20,409) finish the bulk index method with a single semantic signature with under 10,000 documents finishing the algorithm with two or more hits from different semantic signatures. Compare this to our iterative indexing method. Our distribution for number of semantic signature hits peak at between 5 to 7 semantic signatures, and this is accomplished without changing the rule for our indexing algorithm to end. Earlier research into document clustering suggested that documents may require many attributes for an effective clustering [29], but our method provides promising results with a lower number of semantic signatures. This is a promising discovery that should be evaluated further.

5.4 Conclusions

The results from our iterative indexing method show promise. The cross-validations with our human participants appear to match quite well for documents where the topic is obvious, and the groupings of both the algorithm and humans can be explained where there is disagreement. The larger number of semantic signature hits per document has greatly expanded our clusters' ability to sort documents in an unsupervised manner.

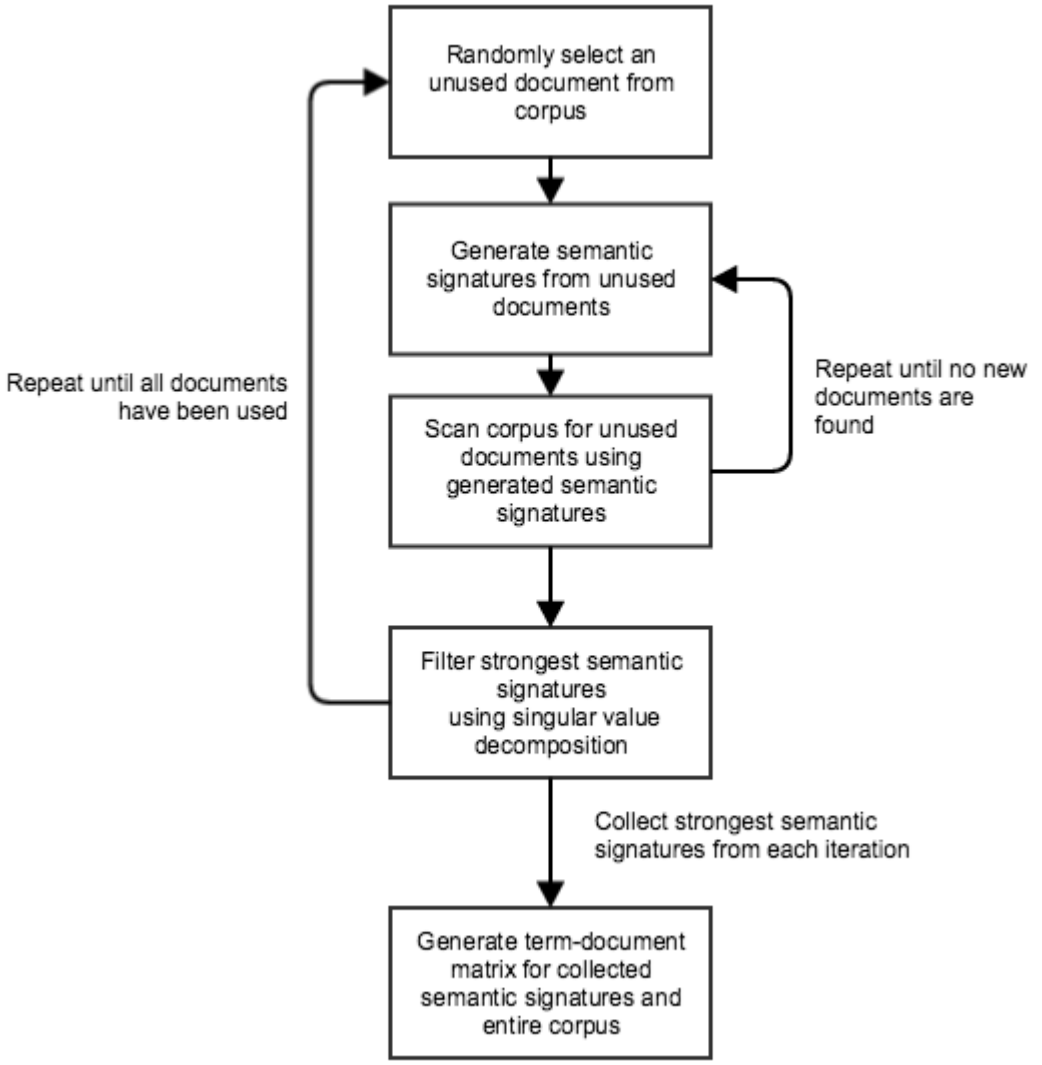

Figure 5.1: An overview of the iterative indexing algorithm

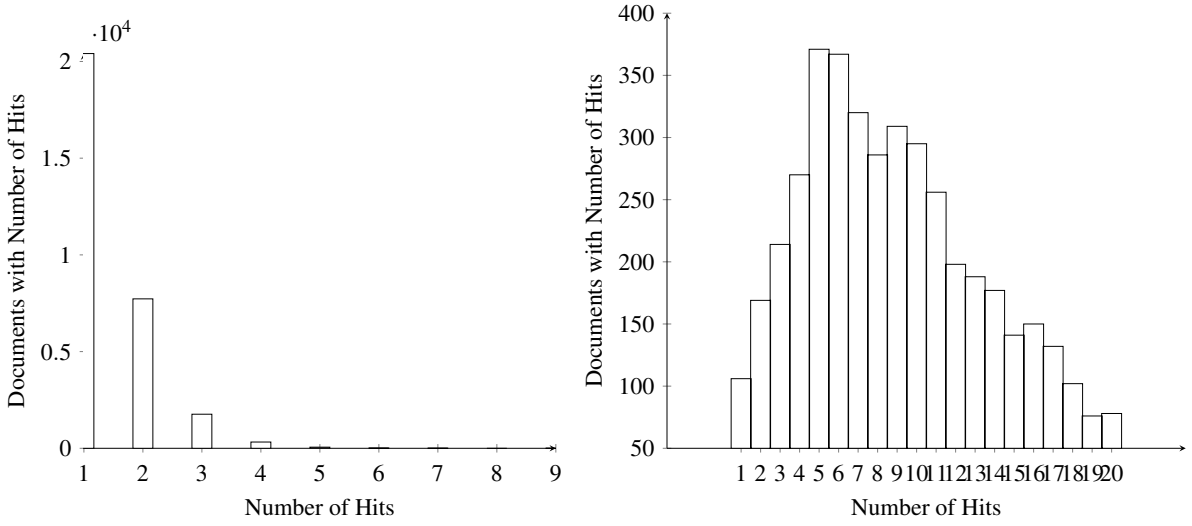

Figure 5.2: Histograms representing the number of semantic signatures hits in documents for the bulk indexing method (left) and the indexing method (right)

Chapter 6

Conclusions

6.1 Conclusions

The task of rewriting the SSMinT tools has been a fruitful endeavor. Our complete reimplementation of the SSMinT software package has demonstrated that the algorithms and ideas of the SSMinT research are successful across implementations, and verifies that the science behind the SSMinT research is sound. The unit tests of the new implementation of tools will help future developers to make changes to the code base and have some degree of confidence that more complicated sections of the code are still functioning as intended. This drastically helps to avoid the concern of programming errors and focus more on performing experiments.Embracing well established software engineering practices such as easily repeatable builds and well formed modules will allow future researchers and developers to quickly iterate experiments.

We proposed our first indexing algorithm the "bulk index method". Our experiments using this method produced poor quality clusters that have no observable theme to most groups of documents as suggested by our human cross-validators. We observed that our process provided too few semantic signatures for most documents to be clusters in any meaningful way, and we theorized that adjusting our indexing algorithm to collect and perform term reduction with semantic signatures

that are related by topic would drastically improve the performance of the document clustering.

We introduced our second indexing algorithm, the iterative indexing algorithm, to address this concern. Our second algorithm strives to find semantic signatures by finding documents that are related in some way. Our clusters are for the most part drastically improved in the quality of the groupings. Our human cross-validators agree with the grouping of our algorithm considerably as evidenced by using the Adjusted Rand Index values comparing between the first experiment and our second. Situations where the humans did not agree with the algorithm's groupings were able to be explained by grouping the documents using another criteria. A comparison of the number of semantic signatures per document between the bulk indexing method and the iterative indexing method shows that the latter method selected between 5 and 7 semantic signatures for most documents with very few documents having less than 3 semantic signatures.

Our methods are ready to be evaluated against other data sets, and would benefit from a real-world trial in the area of forensics and law enforcement. Every effort was made to create an experimental software package to enable researchers to construct and perform future experiments with minimal effort. We sincerely hope to hear that we've accomplished this goal and look forward to the discoveries to come.

6.2 Future Work

This work in this thesis scratches the surface of what can be achieved using indexing methods and the SSMinT software package, and this work leads us to many new questions. The reworked tools were built with the idea that researchers would be able to take parts of the software package and create new experiments with minimal hassle.

Below are some potential areas where this work could be expanded.

6.2.1 Future Areas of Algorithmic Investigation

Considerable work has been presented in this thesis, but there is more work that can be done to improve the speed and efficiency of the progress. The code was often written with the readability of the code a priority with memory and CPU efficiency a secondary concern. This would be worth revisiting to find a better balance between well-written code and performance of the code.

Finding Optimal Semantic Signatures for Documents

Recall in Chapter 5 our histogram comparing the number of semantic signature hits per document between the two indexing methods. We found that more semantic signatures aided our clustering algorithm, but continually adding more semantic signatures will eventually add little more than noise to the clustering. A study aimed at finding numbers of semantic signatures that well summarize various sizes of documents could help design algorithms that seek out some target number of semantic signatures per document.

Focusing on Topics for Semantic Signatures

We observed during the cross-validation of the iterative indexing algorithm detailed in Chapter 5 that our humans and algorithm defined certain documents differently from one another. The humans were able to identify additional topics such as technology and finance to separate the documents, while the algorithm only considered what was discovered through the use of TF-IDF. These are general terms. We may be able to expand our keyword selection by using variants of TF-IDF or even other keyword selection algorithms to create an ensemble method of keyword selections that can capture various points of views similar to humans.

Investigation of Alternate Clustering Algorithms

Significant work can be done in investigating clustering algorithms to act in place of K-means. Bisecting K-means is a possible replacement algorithm that may perform better for document clustering [28] [6]. KD-trees are also an interesting possibility for clustering that would remove the hassle of selecting the number of clusters up front [3].

Investigation of SSMinT as an Ensemble Learning Component

Recent years have seen the rise of a concept in machine learning called ensemble learning in which multiple algorithms are individually trained and used in classification [16]. Ensemble learning became of particular interest when teams competing in the Netflix challenge began to merge their algorithms together into ensemble techniques. Ensemble learning in text mining has been gaining traction in recent years [1]. SSMinT has noted shortcomings for documents that may be too short, and a demonstration of the automated process's effectiveness as an algorithm in an ensemble learning technique may increase the attractiveness of the process.

6.2.2 Future Areas of Possible Applications of SSMinT

A significant amount of research has been performed on the algorithms in the SSMinT text mining process, but very little work has been performed for using SSMinT in a production-like capacity. A demonstration of how SSMinT could solve a real-world problem would be an incredible demonstration of SSMinT's utility.

Discovery of Files on a Confiscated Device

SSMinT has a potential use in the realm of Computer Forensics. Investigators of seized computers and mobile devices often find themselves in a situation where the device must be searched for information relevant to the investigation. This can be a time intensive task which can be prone

to human error. SSMinT and the automatic indexing algorithm could potentially be applied to a device to find and separate text data of interest to an investigator. The rewritten SSMinT tools includes an abstraction (the DataSource) for different kinds of files that can be easily extended to read documents in whatever format may be required.

Mining Patient Data in Medical Records

SSMinT also has potential applications in mining medical records. Medical records often consist of free-form text that will vary greatly depending on the physician authoring the record entry. The ability to mine these records to find patients that may be at high risk of complications, to find suitable patients for medical trials, or to perform health care quality monitoring is often limited. Optimizing SSMinT to make these processes easier to perform could drastically improve many areas of of health record data mining.

A Commercial SSMinT Suite

A core concept in the original creation of the SSMinT tools was that an analyst could construct a library of semantic signatures that could then be used to investigate documents. A possible idea for a student of Software Engineering would be to create an investigation tool to demonstrate an implementation of this shared library. Investigators could upload documents and find keywords through a web interface. Semantic signatures could be learned automatically and asynchronously from uploaded learning documents. These semantic signatures could be shared between investigators/users to encourage collaborative investigation efforts. Files might be uploaded in bulk for hits to be calculated asynchronously while the investigator and their computer is freed up for other tasks. This client-server application would potentially be an excellent demonstration of the SSMinT tools as a product, and the work performed in this thesis rewriting the SSMinT tools can serve as a base to start.

6.3 Conclusion

In conclusion, we believe that the SSMinT tools will enable researchers and developers to build experiments faster than before. Two algorithms were proposed to index a corpus of documents: the bulk indexing method and the iterative indexing method. The bulk indexing method suffered from finding too few semantic signatures per document and poor coverage of semantic signatures for topics in the corpus. The iterative indexing method addresses issues from our first method, and cross-validation shows promising performance when compared with humans. SSMinT and our new indexing algorithms may potentially be applied in numerous problems, but ongoing work will be required to answer additional questions raised by this research.

Bibliography

[1] ADEVA, J. J. G., BERESI, U. C., AND CALVO, R. A. Accuracy and diversity in ensembles of text categorisers. *CLEI Electron. J. 8* (2005).

[2] ARTHUR, D., AND VASSILVITSKII, S. k-means++: the advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms* (Philadelphia, PA, USA, 2007), SODA '07, Society for Industrial and Applied Mathematics, pp. 1027–1035.

[3] BENTLEY, J. L. Multidimensional binary search trees used for associative searching. *Commun. ACM 18*, 9 (Sept. 1975), 509–517.

[4] BERRY, M. W. Large scale sparse singular value computations. *International Journal of Supercomputer Applications 6* (1992), 13–49.

[5] BERRY, M. W., DUMAIS, S., AND O'BRIEN, G. Using linear algebra for intelligent information retrieval. *SIAM REVIEW 37* (1995), 573–595.

[6] BOLEY, D. Principal direction divisive partitioning. *Data Min. Knowl. Discov. 2*, 4 (Dec. 1998), 325–344.

[7] CECIL, K. Querying unix sockets with curl. `http://kelcecil.com/curl/2015/12/07/query-unix-socket-with-curl.html`. Accessed: 2016-04-04.

[8] DANIEL, L. E. Plain view doctrine in digital evidence cases - a common sense approach. `http://www.dfinews.com/article/plain-view-doctrine-digital-evidence-cases%E2%80%94-common-sense-approach`.

[9] HASTIE, T., TIBSHIRANI, R., AND FRIEDMAN, J. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer New York, New York, NY, 2009, ch. Unsupervised Learning, pp. 485–585.

[10] HILL, K. How target figured out a teen girl was pregnant before her father did. `http://www.forbes.com/sites/kashmirhill/2012/02/16/how-target-figured-out-a-teen-girl-was-pregnant-before-her-father-did/`. Accessed: 2016-04-10.

[11] HUBERT, L., AND ARABIE, P. Comparing partitions. *Journal of Classification 2*, 1 (1985), 193–218.

[12] JONES, K. S. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation 28*, 1 (1972), 11–21.

[13] KOTA, R. R. Automated discovery of relevant features for text mining. Master's thesis, West Virginia University, May 2011.

[14] KRALLINGER, M., LEITNER, F., RABAL, O., VAZQUEZ, M., OYARZABAL, J., AND VALENCIA, A. Overview of the chemical compound and drug name recognition (chemdner) task. *BC IV - CHEMDNER Proceedings 2* (2013), 2–33.

[15] LUHN, H. A statistical approach to mechanized encoding and searching of literary information. *IBM Journal of Research and Development 1* (1957), 309–317.

[16] MACLIN, R., AND OPITZ, D. W. Popular ensemble methods: An empirical study. *CoRR abs/1106.0257* (2011).

[17] MACQUEEN, J. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics* (Berkeley, Calif., 1967), University of California Press, pp. 281–297.

[18] MALLAMPATI, U. Enhanced automated discovery of relevant features in text mining. Master's thesis, West Virginia University, May 2014.

[19] MARTIN, R. C. *Clean Code: A Handbook of Agile Software Craftsmanship*, 1 ed. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2008.

[20] MEILĂ, M. Comparing clusterings - an information based distance. *Journal of Multivariate Analysis 98*, 5 (2007), 873 – 895.

[21] ORACLE. What is data mining? `https://docs.oracle.com/cd/B28359_01/datamine.111/b28129/process.htm`. Accessed: 2016-04-10.

[22] PARA, U. K. Computer-aided semantic signature identification and document classification via semantic signatures. Master's thesis, West Virginia University, May 2010.

[23] PEDDADA, S. R. Sensitivity of semantic signatures in text mining. Master's thesis, West Virginia University, May 2010.

[24] RAND, W. M. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association 66*, 336 (1971), 846–850.

[25] REUTEMANN, M. H. E. F. G. H. B. P. P., AND WITTEN, I. H. The weka data mining software: an update. *SIGKDD Explor. Newsl. 11*, 1 (2009), 10–18.

[26] SEBASTIANI, F. Machine learning in automated text categorization. *ACM Comput. Surv. 34*, 1 (Mar. 2002), 1–47.

[27] SONG, M. Opinion: Text mining in the clinic. http://www.the-scientist.com/?articles.view/articleNo/34820/title/Opinion--Text-Mining-in-the-Clinic/. Accessed: 2016-04-05.

[28] STEINBACH, M., KARYPIS, G., AND KUMAR, V. A comparison of document clustering techniques. In *KDD Workshop on Text Mining* (2000).

[29] WITTEN, I. H., DON, K. J., DEWSNIP, M., AND TABLAN, V. Text mining in a digital library. *International Journal on Digital Libraries 4*, 1 (2004), 56–59.

[30] WRIGHT, C. Forensic mining. http://computer-forensics.sans.org/blog/2008/12/05/forensic-mining.

[31] WU, Q., FULLER, E., AND ZHANG, C. Q. Text document classification and pattern recognition. In *2009 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)* (July 2009), pp. 405–410.

[32] WU, Q., FULLER, E., AND ZHANG, C. Q. Graph model for pattern recognition in text. In *Studies in Computational Intelligence*, H. J. W. I. H. Ting and T. H. Ho, Eds., vol. 288. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 1–20.