2014

Enhanced Automated Discovery of Relevant Features in Text Mining

Udaya Mallampati

Follow this and additional works at: <https://researchrepository.wvu.edu/etd>

Recommended Citation

Mallampati, Udaya, "Enhanced Automated Discovery of Relevant Features in Text Mining" (2014).
Graduate Theses, Dissertations, and Problem Reports. 6148.
<https://researchrepository.wvu.edu/etd/6148>

This Thesis is protected by copyright and/or related rights. It has been brought to you by the The Research Repository @ WVU with permission from the rights-holder(s). You are free to use this Thesis in any way that is permitted by the copyright and related rights legislation that applies to your use. For other uses you must obtain permission from the rights-holder(s) directly, unless additional rights are indicated by a Creative Commons license in the record and/ or on the work itself. This Thesis has been accepted for inclusion in WVU Graduate Theses, Dissertations, and Problem Reports collection by an authorized administrator of The Research Repository @ WVU. For more information, please contact researchrepository@mail.wvu.edu.

Enhanced Automated Discovery of Relevant Features in Text Mining

Udaya Mallampati

Thesis submitted

to the Benjamin M. Statler 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

Approved by

Dr. Elaine Eschen, Ph.D., Chair

Dr. Alan Barnes, Ph.D.

Dr. Vinod Kulathumani, Ph.D.

Lane Department of Computer Science and Electrical Engineering

Morgantown, West Virginia

2014

Keywords: Text Mining, Document Retrieval, Keywords, Clustering

©2014 Udaya Mallampati

ABSTRACT

Enhanced Automated Discovery of Relevant Features in Text Mining

by

Udaya Mallampati

Semantic Signature Mining Tool (SSMinT) is a suite of software tools that aid a data analyst to develop semantic signatures that capture targeted content, and uses these semantic signatures to categorize text documents with unknown content or retrieve documents of a specific type or interest. This was developed by Barnes, Eschen, Para, and Peddada in 2010. These tools require expert input. An automated version of SSMinT software package was developed with the aim to reduce manual input and use machine learning techniques to discover semantic signatures. This was developed by Barnes, Eschen, and Kota in 2011. Key features of this include automated keyword group generation, automated generation of candidate semantic signatures, and methods to prune redundant relevant semantic signatures from the semantic signature set. Human input is required only at the time of choosing the training documents.

This thesis develops an enhanced version of the Automated Semantic Signature Mining Tool which increases the scope for capturing semantic content from the training documents. In particular, problems with analyzing very short documents are addressed. Improvements made in the tools minimize the unnecessary keyword groups in the early stages of the learning phase, and thereby maximizes the number of significant semantic signatures generated in the later stages of the learning phase. Thereby, a larger number of documents that are similar to the training documents are retrieved. The resulting fine-tuned semantic signatures also yield effective categorization of text documents into groups with closely related content. Tools are developed to automate the tedious process of measuring the document retrieval rates. A statistical method is also employed to estimate the precision of document retrieval.

Dedication

To My Parents

To My Advisor Dr. Elaine Eschen and Dr. Alan Barnes

Acknowledgements

I would like to thank my advisors Dr. Eschen and Dr. Barnes for their support and encouragement throughout the research. I would like to express my gratitude to Dr. Eschen for guiding me to program efficiently. Thanks to Dr. Barnes for the guidance and motivation that he provided throughout the project. I would like to thank Dr. Vinod for his time and consideration. It gives me immense pleasure to thank my parents. I couldn't have done this without their love and support. Love you daddy! You are the one who taught me how to accept challenges and solve them at ease. Thanks to my seniors Uday Para, Ramya Peddada, and Ravali Kota for providing me a strong base to work on.

Table of Contents

1. Introduction.....	1
1.1. Motivation.....	2
1.2. Contributions.....	2
1.3. Flow of the Document	3
2. Overview.....	3
2.1. Review of Automated SSMiT.....	3
2.1.1. Automated Keyword Tool.....	3
2.1.2. Automated Learner Tool	11
2.1.3. HAG Tool.....	15
2.1.4. SSR Tool	18
3. Corrections.....	22
3.1. Pruning Keyword Sets in ALT:	22
3.2. Increasing Weight of Keywords with Infinite IDF Value	23
3.3. Removing documents that do not generate hits from the final clustering results.....	23
3.4. Testing the Hits Array before sending it as input to Semantic Signature Refinement Algorithm.....	24
4. Enhancements	24
4.1. Eliminating keyword sets based on the proximity measure used in the Keyword Tool....	24
4.1.1. Data Structures	24
4.1.2. Proximity Measure	27
4.2. Proximity Measure used in the Hits Array Generator Tool.....	28
4.2.1. First Condition.....	28
4.2.2. Second Condition	29
4.3. Handling the case where the number of documents is less than the number of Semantic Signatures in Semantic Signature Refinement Tool	29
5. Supporting Tools.....	30
5.1. Automated Text File Extractor	31
5.1.1. Input for the Automated Text File Extractor Tool	31
5.1.2. Working of Automated Text File Extractor Tool.....	32
5.2. Automated Group Inserter	34

5.2.1. Input for the Automated Group Inserter Tool	34
5.2.2. Working of the Automated Group Inserter Tool	34
5.2.3. Output of the Automated Group Inserter Tool	35
5.3. Automated Value Extractor Tool	36
5.3.1. Working of the Automated Value Extractor Tool	36
5.3.2. Algorithm for Automated Value Extractor Tool	37
6. Experimental Setup	40
6.1. Experiments for Evaluating the Performance of Enhanced Tools in the Package	40
6.1.1. Objective	40
6.1.2. Corpus	40
6.1.3. Notation	40
6.1.4. Procedure	41
6.2. Multi-category Document Retrieval	42
6.2.1. Objective	42
6.2.2. Corpus	43
6.2.3. Modifications	43
6.2.4. Procedure	44
6.3. Statistical Method for Estimating Precision of Document Retrieval: True Positive and False Positive Rates	46
6.3.1. Objective	46
6.3.2. Assumptions and Theory Behind the Experiment	46
6.3.3. Procedure	48
7. Experimental Results	50
7.1. Experimental Results for Evaluating the Performance of Enhanced Tools	50
7.1.1. Results from Previous Version of Automated SSMiT	50
7.1.2. Results from Enhanced Version of Automated SSMiT	53
7.1.3. Results Obtained by Imposing the First Condition on HAG Tool	56
7.1.4. Results Obtained by Imposing Second Condition on HAG Tool	57
7.1.5. Results obtained by increasing the number of keywords considered for the first list	59
7.2. Multi-Category Document Retrieval Experiment	64
7.3. Statistical Method for Estimating Precision of Document Retrieval Experiment	64

7.3.1. Results obtained by setting ' a ' to 5	64
7.3.2. Results obtained by setting ' a ' to 10	65
7.3.3. Results obtained by changing window-size to 40	67
7.3.4. Results obtained by changing window-size to 50	68
8. Conclusions and Future Work	68
8.1. Conclusions	68
8.2. Future Work	69
Bibliography	74

List of Figures

Figure 2-1 Flow Chart for AKT	5
Figure 2-2: Flow Chart for ALT	11
Figure 2-3: Flow Chart for HAG Tool.....	16
Figure 2-4: Structure of Hits Array.....	18
Figure 2-5: Flow Chart for SVD.....	19
Figure 2-6: Flow Chart for Cluster Generation.....	20
Figure 4-1: Hash Table for Frequency.....	25
Figure 4-2: Hash Table for Start Location.....	25
Figure 4-3: Hash Table for storing next location.....	26
Figure 4-4: Array Storing Locations.....	27
Figure 5-1Structure of Input XML file	32
Figure 5-2Working of Automated Text File Extractor	33
Figure 5-3: Structure of Semantic Signature File	34
Figure 5-4: Working of Automated Group Inserter Tool	35
Figure 5-5: Structure of Enhanced Semantic Signature File	36
Figure 5-6: Working of Automated Value Extractor.....	37
Figure 5-7: Hits Array.....	38
Figure 6-1: Flow chart for the Experimental Procedure	42
Figure 6-2: Procedure for Document Retrieval Experiment.....	45
Figure 6-3: Procedure for Chi-Square Test.....	49
Figure 8-1: Diagonal Matrix	70
Figure 8-2: Modified Diagonal Matrix	70

List of Tables

Table 6-1: Abbreviations for Professor's Names	41
Table 6-2 Filter Count Determination.....	46
Table 7-1: Inputs and Outputs for the training tools of SSMinT.....	51
Table 7-2: Semantic feature vector clustering results.....	52
Table 7-3: Inputs and Outputs for learning tools	53
Table 7-4: Inputs and Outputs for Testing Tool	54
Table 7-5: Semantic feature vector clustering results.....	54
Table 7-6: Inputs and Outputs for Testing Tool	56
Table 7-7: Semantic feature vector clustering results.....	57
Table 7-8: Inputs and Outputs for Testing Tool	58
Table 7-9: Semantic feature vector clustering results.....	58
Table 7-10: Inputs and Outputs for learning tools	60
Table 7-11: Inputs and Outputs for the Testing Tools.....	60
Table 7-12: Semantic feature vector clustering results.....	61
Table 7-13: Inputs and Outputs for Testing Tools.....	62
Table 7-14: Semantic feature vector clustering results.....	63
Table 7-15 Known values obtained by setting 'a' to 5.....	64
Table 7-16: Parameters obtained by setting 'a' to 5	65
Table 7-17 Known values obtained by setting 'a' to 10	65
Table 7-18 Parameters obtained by setting 'a' to 10.....	65
Table 7-19 Known Values obtained by setting 'a' to 15	66
Table 7-20 Parameters obtained by setting 'a' to 5.....	66
Table 7-21: Known values obtained by changing window-size to 40.....	67
Table 7-22: Parameters obtained by changing window-size to 40	67
Table 7-23: Known Values obtained by changing window-size to 50	68
Table 7-24 Parameters obtained by changing window-size to 50	68
Table 8-1 Difference in Hits Array	73

List of Symbols

1. DF- Document Frequency
2. IDF- Inverse Document Frequency
3. KGG – Keyword Group Generation
4. KDF – Keyword Descriptor File
5. SSDF – Semantic Signature Descriptor File
6. HADF – Hits Array Descriptor File
7. ARFF – Attribute Relation File Format
8. AKT – Automated Keyword Tool
9. ALT – Automated Learner Tool
10. HAGT – Hits Array Generator Tool
11. AFET- Automated File Extractor Tool
12. AGIT- Automated Group Inserter Tool
13. AVET- Automated Value Extractor Tool
14. SSRT – Semantic Signature Refinement Tool
15. CD– Cluster Definition
16. HTML – Hyper Text Markup Language
17. XML – Extensible Markup Language

1. Introduction

Text mining refers to the process of deriving high-quality information from text [1]. Text mining is different from data mining which tries to find interesting patterns in large databases. In text mining, the patterns are extracted from the natural language text rather than the structured database of facts [2]. Information to be extracted is explicitly stated in the document. But it is a tedious task to read the entire text and extract required information. This process is made simple by training software tools to make knowledge driven decisions.

Our basic approach to mining text data aims at capturing the semantic structures in the text. Semantic structure depends on the correlations between keywords and locality of keyword groups. The traditional bag-of-words or keyword frequency approaches fall short of modeling these attributes. Our approach models not only keyword frequency, but also the distance between keywords and their relative ordering in the text. To this end, we derive high-dimensional vectors that store quantified relationships between keywords in a text document. In order to capture the locality of semantic structures, we generate many vectors per document. The content of these vectors is similar to the document vector (one per document) used by Zhang et al. in [3] [4]. However, unlike Zhang et al., we do not use these vectors directly to classify documents. Vectors generated from known content (learning) documents are used to develop semantic signatures that model the semantic structure of the target content. Multiple semantic signatures can be used to model various nuances of single target content. Semantic signatures drawn from a library are then used to classify documents of unknown content. Our new approach has proven to be a remarkably sensitive tool for differentiating semantic content in text data.

An automated technique that allows grouping documents of similar content by semantic signature discovery or retrieving documents of a targeted content from an arbitrary corpus with minimal input from the analyst was developed by Barnes, Eschen, and Kota [5].

This thesis presents measures that enhance these automated tools. These enhancements increase the efficiency of the tools and assist to dig deep into the keyword frequency list. Thus, the training tools gain greater knowledge about the semantic structures of the documents. This knowledge can be used by the testing tools to efficiently classify or retrieve documents with

targeted content. Furthermore, this thesis automates the process of measuring document retrieval rates. A statistical method for estimating precision of document retrieval is also developed.

1.1. Motivation

Automated Semantic Signature Mining Tool consists of four tools: Automated Keyword Tool (AKT), Automated Learner Tool (ALT), Hits Array Generator Tool (HAGT), and Semantic Signature Refinement Tool (SSRT). These tools operate in two phases: Training and Testing. AKT, and ALT are training tools and HAGT and SSRT are testing tools. Tools are trained with the documents similar to the ones that are required to be grouped or retrieved. The gained information is stored in the form of Semantic Signatures. The generated Semantic Signatures are used in the testing phase to perform the intended task. This works fine except that they generate numerous Keyword Sets and Semantic Signatures in the training phase. This seriously limits the number of frequent words considered in the frequent word list. So there is a chance of ignoring important semantic content from the training documents.

The motivation to modify and enhance these tools is to decrease the workload on the training tools, increase the amount of semantic content retrieved from the training documents, and also increase the sensitivity of testing tools. These improvements are used to train the tools effectively in the learning phase, and thereby, make the testing tools that categorize documents or retrieve documents of interest efficient.

1.2. Contributions

The contributions of this thesis can be summarized as follows:

- Modified the learning tools to decrease the number of Keyword Sets and Semantic Signatures generated.
- Changed the data structures being used in the Automated Keyword Tool in order to increase efficiency.
- Increased the sensitivity of the Hits Array Generator Tool.
- Enabled the Semantic Signature Refinement Tool to handle all possible cases.
- Analyzed the ability of the automated tools in retrieving documents specific to a category.
- Built tools to automate the process of measuring document retrieval rates.

1.3. Flow of the Document

Chapter 2 introduces the suite of automated tools developed by Barnes, Eschen, and Kota. The algorithms used by these tools are also explained in this chapter.

Chapter 3 deals with the modifications made in the algorithms and assumptions used by the automated tools. The need to make these modifications is also clearly explained.

Chapter 4 details the enhancements made in the automated tools to increase efficiency.

Chapter 5 introduces the tools required to automate the tedious process of measuring document retrieval rates.

Chapter 6 describes the experiments conducted to measure the efficiency of the enhancements, and to analyze the ability of the automated tools using a statistical measure. It also describes the data-set used for conducting the experiment.

Chapter 7 gives the results obtained by conducting the experiments mentioned in Chapter 6. Two experiments were presented in this chapter. These experiments contain sub-experiments. These are used to analyze the capability of enhancements made in the automated tools.

Chapter 8 presents an overview of the conclusions derived from this thesis and leaves a scope for future work. This chapter also stresses on the areas that require concentration in order to further refine the results.

2. Overview

2.1. Review of Automated SSMinT

The Automated Semantic Signature Mining Tool (Automated SSMinT) consists of four tools: Automated Keyword Tool (AKT), Automated Learner Tool (ALT), Hits Array Generator Tool (HAGT), and Semantic Signature Refinement Tool (SSRT). These tools discover semantic signatures in text data with minimal analyst input and identify patterns that an analyst may not recognize due to large volume of data. These tools thereby help the analyst to save the time and reduce his work load. The following sections explain algorithms for the above mentioned tools.

2.1.1. Automated Keyword Tool

Automated Keyword Tool (AKT) belongs to the training phase. It takes training documents similar to the testing documents as input. It is used to generate keyword groupings using the Term Frequency-Inverse Document Frequency (TF-IDF). These definitions are explained in Section 2.1.1. The training documents are concatenated and preprocessed. Each and

every word (except stop-words) is assigned a weight. Using the computed weight, keywords are identified. The second and third keywords in the keyword grouping are identified using the proximity measure. These keyword groupings are saved in files called Keyword Descriptor Files (KDFs). The generated KDFs are used as input to the Automated Learner Tool (ALT). [5]

The following flow chart explains the working of Automated Keyword Tool.

Figure 2-1 Flow Chart for AKT

Various algorithms are followed to generate Keyword Sets. These include an algorithm for TF-IDF computation and an algorithm for keyword group generation. The following sections give the description of these algorithms along with an example.

2.1.1.1. Algorithm for TF-IDF Computation

Algorithm TF_IDF_Computation

Input: Training document, phrases list, and the corpus. If a set of documents are to be taken as training documents, then they are placed in a single document and given as input.

Output: List of words with the TF-IDF product, arranged in descending order of their product value.

1. Preprocess the training document.
2. Calculate the Term-Frequency (TF) for all the words in the preprocessed document.
3. For each file in the corpus, preprocess the file. If the stemming check box is checked, then stem the words in the document.
4. For each term in the training document, calculate the number of documents in which the term appears.
5. Calculate the Inverse Document Frequency (IDF).
6. All the words that have IDF value as infinity are ignored.
7. Generate the TF-IDF product for each of the remaining terms. Store the word along with the TF-IDF product in a list.
8. Sort the list of words according to TF-IDF product. Reverse the list.

Details of Step 1:

- a) Replace phrases. For example the word "Black-Market" has entirely different meaning when compared to the two words "black" and "market" when separated. Such words are considered as a single phrase rather than separate words. The words that are to be considered as a phrase are given as input by the user. [5]
- b) Remove Null Strings.

Details of Step 2:

- a) Scan the document. If the word appears for the first time, store it as a key in the hash-table with value 1. Otherwise, find out the value (in the hash-table) with key as word and then increment the value by 1.
- b) TF for a word is calculated using the formula,

$$TF = \frac{\text{Number of times it occurs in the document}}{\text{Total number of words in the document}}$$

Details of Step 5:

- a) IDF is calculated using the formula,

$$IDF = \log\left(\frac{\text{Number of files}}{\text{Number of documents in which the term appears atleast once}}\right)$$

1. Keyword Group Generation Algorithm

Algorithm Keyword_group_generation

Input: The list containing words with corresponding TF-IDF product, the number of terms (**n**) to be considered in list 1, the number of terms (**n1**) to be considered in list 2, and the constant ‘**a**’ used in the computation of weight (proximity measure). The first list contains all the first set of keywords and the second list contains all the second and the third keywords in each keyword group.

Output: Keyword groups.

1. Form a list (list1) from the top ‘**n**’ words of the input TF-IDF list.
2. Consider a keyword from list1.
3. Search for the keyword and if it is found, then start the window from that location. Two windows are formed such that one window starts with the keyword and the other ends with the keyword.
4. In the window compute the proximity measure from keyword to every other word in the window using the forward and backward distances.
5. Proceed to the next location and go to step 3 until the entire document is searched for the keyword or the window extends beyond the length of the document.

6. Generate a list of words that have higher proximity measure with respect to the keyword.
7. Consider the top **n1** highly correlated words (having greater proximity measure) with the keyword and append them to list2.
8. Go to step 2 until all the keywords in the list1 have their corresponding correlated words.
9. Apply various combinations by taking a single word from list1 and two words from list2.
10. Ignore the duplicate groups. The remaining are the keyword groups of interest.

Details of Step2:

k- Keyword, **i**- Location of keyword, **Win_size**- Window size

Details of Step3 and Step4:

- a) Search each and every word in the document starting from the location **i+1** up to **i+win_size**. If it is a stop word or the keyword itself, then ignore the word.
- b) If not, then search for the encountered word in the list of words with proximity measures. If the word is not present in the list then add the word to the list with proximity measure as,

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

If it is already present, then add the weight to the previously computed weight as shown below,

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

Here '**a**' is a constant given by the user as input and '**x**' is the distance between the keyword and the word under consideration.

- c) Now start the search in the backward direction starting from the location **i-1** to **i-win_size**. Apply same procedure for computing backward distance as well.

Details of Step5:

- a) When the window is entirely searched, start searching for the keyword from the location $i+1$.

Details of Step10:

- a) The groups having the same set of keywords in another order or the groups containing repeating keywords are considered as duplicates and ignored.

Example for Deriving keyword groups from the two lists:

1. From the sorted list of words given by TF-IDF, the top ' n ' highly scored words are returned. The number ' n ' is given as input to the Automated SSMInT. For example, let n be 5. Let the five words be '**a**', '**b**', '**c**', '**d**', and '**e**'. These words are shown in List1.

A
B
C
D
E

2. Inside a window, the weights are calculated from a keyword in the list to every other word in the window using the window-weight in both the forward and the backward directions. This process is repeated for each and every window. A list of highly correlated words is formed from the keyword. From the sorted list of weights, ' n_1 ' most strongly correlated words are taken. Let ' n_1 ' be 3. The algorithm ensures that there are no duplicates.

First, keyword '**a**' is taken and weights are calculated from '**a**' to other words in the window in both forward and backward directions. This is done for all the windows formed around **a**. '**a₁**', '**a₂**' and '**a₃**' are found to be top three highly correlated words with '**a**' and they are represented as List2. Also '**a**' is not equal to '**a₁**' or '**a₂**' or '**a₃**'. Similarly let '**b₁**', '**b₂**', and '**b₃**' be the top three highly correlated words with **b**.

a1
a2
a3
b1
b2
b3
:

3. Keyword Sets (each containing 3 keywords) are generated with all possible combinations of above words in the two lists. One word is taken from list1 and the remaining two words are taken from list2. If there are duplicates in a keyword group, that combination of keywords is discarded. The final list of keyword groups is as shown below,

a,a1,a2
a,a1,a3
a,a2,a3
b,a1,a2
b,a1,a3
b,a2,a3
b,b1,b2
b,b1,b3
b,b2,b3

2.1.2. Automated Learner Tool

Automated Learner Tool (ALT) discovers Semantic Signatures in the training documents without the aid of analyst. It can select the semantic signature clusters. It takes KDFs and Learning documents as input. All the generated clusters are saved as Semantic Signature Descriptor Files (SSDFs). [5]

Figure 2-2: Flow Chart for ALT

The following is an algorithm that briefly explains the functioning of ALT.

2.1.1.2. Semantic Signature Generation Algorithm

Algorithm SSD_Generation

Input: Training documents, and KDF files.

Output: Semantic Signature Descriptor files

1. Preprocess the document and convert it into string array containing all the words in the document as elements.
2. For each KDF file, invoke the Weight Calculation algorithm with input as the training document, window size, and the Keyword Set. This algorithm returns a set of document vectors.
3. Prune the document vectors that do not represent any interaction between all the keywords.
4. Invoke the clustering algorithm with the remaining document vectors as input and the number of clusters into which the document vectors are to be divided.
5. Generate the SSD files relating to the KDF.
6. Go to Step 3 until all KDF files have their related SSD files.

As shown in the above algorithm, Automated Learner Tool (ALT) uses two algorithms. They are the Weight Calculation algorithm, and the K-means clustering algorithm. The following sections explain the algorithms along with examples.

2.1.1.3. Weight Calculation Algorithm

Algorithm Weight_Calculation:

Input: String array containing the words of the training document, Keyword Set (let them be kw1, kw2, kw3), and window size.

Output: Jagged array storing document vectors.

1. Start searching for any of the keywords in the keyword set. Start the window from that keyword position. Extend the window up to '**window_size**' words starting from that keyword.

2. Compute the forward distances of each and every word from all the three. Populate the 3-by-3 array.
3. Proceed to the very next location immediately after the window.
4. Go to Step 1 and repeat the process until all the words are searched or the end position of the window extends beyond the position of the last word in the document.

Details of Step 2:

- a) Forward distance between two keywords in a particular window is the sum of all the weights of a particular keyword from another keyword in the same window. For example, w_{12} is the sum of all the weights from the kw_1 to kw_2 in the same window.
- b) In the 3-dimensional array, the first subscript is the window number. The second and third subscript represents the keyword numbers. For example if $weight[1,2,3]=x$, then x represents the forward distance between keyword 2 and keyword 3 in window 1. The forward distance represents the interaction between the two keywords in the given window number.
- c) Jagged array is an array of arrays. Each element of the 3D array is a 2D array that clearly indicates the interaction between the keywords in a window. The following is the pictorial representation of the 2D array for one particular window.

W11	W12	W13
W21	W22	W23
W31	W32	W33

- d) Each of the 2D arrays (representing the interaction between the keywords in a window) listed in row-major order comprises a *document vector*. The Weight Calculation algorithm therefore returns a set of document vectors.

2.1.1.4. Pruning of Document Vectors

Now it is required to prune the document vectors that do not capture interaction between all the three words. The following condition is used for the refinement,

Only the document vectors having non-zero value in $[(W_{12} \text{ or } W_{21}) \text{ and } (W_{13} \text{ or } W_{31}) \text{ and } (W_{23} \text{ or } W_{32})]$ are considered and all the remaining document vectors are eliminated.

These refined document vectors become input for the K-means Clustering algorithm.

2.1.1.5. Clustering Algorithm

Algorithm K-means_Clustering:

Input: Refined document vectors, and number of clusters (k) into which the document vectors are to be divided.

Output: Two-dimensional array containing centroid vectors of the clusters, and the array containing radius of the clusters.

1. Choose k random vectors as centroid vectors for the k clusters.
2. Populate the two-dimensional centroid array with the randomly chosen vectors.
3. Consider a document vector from the set of refined document vectors (if it is first run) or document vector from any of the cluster (if clusters are already formed) and find the distance of the vector from the centroid vectors of all the clusters.
4. Assign the vector to the cluster to which it is closer.
5. Go to step 3 and repeat the steps until all the document vectors are clustered or until all document vectors rearrange themselves among the clusters.
6. For each and every cluster, compute the mean of all the document vectors (that fall under the cluster).
7. Check if the computed mean is equal to the existing centroid. If not then populate the centroid array of the corresponding cluster with the new computed mean.
8. Repeat the steps from 3 until stability is reached (the centroid of at least one cluster does not change after computing the mean of all the vectors in that cluster), or after reaching the maximum number of iterations.
9. Compute the radius of each and every cluster.

Details of Step 4:

- a) If it is a Euclidean distance, then the assign the document vector to the cluster whose centroid vector is at a minimum distance from it. The distance between the centroid vector and the document vector is computed using the formula,

$$Distance = \sqrt{[(w_{11}^2 - w_{11}^c)^2 + (w_{12}^2 - w_{12}^c)^2 + (w_{13}^2 - w_{13}^c)^2 + \dots]}$$

In the above equation, $(w_{11}, w_{12}, w_{13}, w_{21}, w_{22}, w_{23}, w_{31}, w_{32}, w_{33})$ are the coordinates of document vector and $(w_{11}^c, w_{12}^c, w_{13}^c, w_{21}^c, w_{22}^c, w_{23}^c, w_{31}^c, w_{32}^c, w_{33}^c)$ are the coordinates of centroid vector.

- b) If it is cosine distance, then the maximum value is taken into consideration while assigning the document vector to a cluster. The following is the formula used for calculation of the distance:

$$Distance = \frac{[(w_{11} * w_{11}^c) + (w_{12} * w_{12}^c) + (w_{13} * w_{13}^c) + (w_{21} * w_{21}^c) + \dots]}{\sqrt{(w_{11}^2 + w_{12}^2 + w_{13}^2 + w_{21}^2 + \dots)} * \sqrt{(w_{11}^c^2 + w_{12}^c^2 + w_{13}^c^2 + w_{21}^c^2 + \dots)}}$$

Details of Step 9:

- a) If it is Euclidean distance, then cluster radius would be the distance from centroid to vector within the proximity and at a maximum distance.
- b) If it is cosine distance, then cluster radius would be the distance from the centroid to the vector at a minimum angle.

2.1.1.6. Generation of SSD Files

Finally, a string array is formed. It has the following data structure:

dv1; dv5; dv10
dv4; dv3; dv12
dv8; dv15; dv9
:

The above data structure is a string array. Each element of the above represented array represents a cluster and the value is the set of document vectors that fall under that particular cluster. The number of elements in the array depends on the number of clusters formed (based on the user input). This information is written to SSD file. Thus a set of SSD files (less than or equal to number of clusters) are generated for each and every KDF file. All the generated SSD files are stored at the specified location.

2.1.3. HAG Tool

Hits Array Generator Tool (HAGT) generates the document-semantic signature matrix, called the Hits Array. Hits for each document against each semantic signature are calculated using the distance measure specified in the SSD files. The matrix is populated using the hit frequencies of the document vectors of testing documents on semantic signatures. This data along with other information (testing document location, SSD file name, etc.) is stored in a file called Hits Array Descriptor File (HADF). [5]

Figure 2-3: Flow Chart for HAG Tool

In the Hits Array Generator tool, two algorithms are used. They are Hits Array Generation algorithm and Weight Calculation algorithm. The following sections explain about the HAG algorithm used and also about the Hits Array with an example.

2.1.1.7. Hits Array Generation Algorithm

Algorithm Hits_Array_Generation:

Input: SSD file, testing documents

Output: Hits Array (a 2D array that has file number as one dimension and SSD file number as another dimension).

1. Consider a file from the list of testing files.
2. Consider SSD file from the list of SSD files.
3. Extract the keywords, centroid vector, and radius of the cluster from the SSD file.
4. Preprocess the considered testing document and convert it into set of words.
5. Initialize integer variable “hits” to the value 0.
6. Execute the Weight Calculation algorithm with input as the set of words and the keywords extracted from the SSD file. It returns a set of document vectors.
7. Consider a document vector from the list of document vectors.
8. Compute the distance of the document vector from the extracted centroid vector. If it satisfies the criteria for proximity, then increment the hits value by 1.
9. Go to step 7 until all the document vectors distance from the centroid vector is calculated.
10. Set the value corresponding to the testing file and SSD file in the 2D array to the hits value.
11. Go to step 2 until all SSD files are visited.
12. Go to step 1 until all the files are visited.

Details of Step 8:

The proximity criteria is as follows,

- a) If it is Euclidean distance, then the criteria is

$$distance \leq radius$$

Distance is computed using the formula,

$$Distance = \sqrt{[(w_{11}^2 - w_{c_{11}}^2)^2 + (w_{12}^2 - w_{c_{12}}^2)^2 + (w_{13}^2 - w_{c_{13}}^2)^2 + \dots]}$$

- b) If it is Cosine distance, then the criteria is

$$distance \geq radius - 0.01$$

Distance is computed using the formula,

$$Distance = \frac{[(w_{11} * w^c_{11}) + (w_{12} * w^c_{12}) + (w_{13} * w^c_{13}) + (w_{21} * w^c_{21}) + \dots]}{\sqrt{(w_{11}^2 + w_{12}^2 + w_{13}^2 + w_{21}^2 + \dots)} * \sqrt{(w^c_{11}^2 + w^c_{12}^2 + w^c_{13}^2 + \dots)}}$$

The following is the structure of Hits Array.

	SSD1	SSD2	SSD3	..	..		..	..
F1	H1	0	H2	0		0		0
F2	0	0	0	0		0		0
F3	0	0	0	H5		H8		H7
F4	0	0	0	0		0		0
F5	H8	0	0	0		H4		0
:	:	:	:	:		:		:

Figure 2-4: Structure of Hits Array

In the above table, H1, H2, etc. are the number of hits generated. This Hits Array is written on a text file and stored at the specified location.

2.1.4. SSR Tool

Semantic Signature Refinement (SSR) tool is used to intelligently refine noisy and redundant semantic signatures which are generated automatically by the ALT. Semantic Signatures are grouped depending on the relevancy and clustered using K-Means in order to overcome the problem of strong redundant relevant semantic signatures dominating some good but not so strong signatures. This tool selects the semantic signatures depending on the weight assigned to them using the Singular Value Decomposition (SVD). The Hits Array is reconstructed using the refined semantic signatures. This further assists the process of classifying the documents in the testing corpus. [5]

Figure 2-5: Flow Chart for SVD

Figure 2-6: Flow Chart for Cluster Generation

The following section explains about the algorithms for refining the semantic signatures.

Algorithm Semantic_Signature_Refinement

Input: Hits Array Descriptor File

Output: Refined Semantic Signature Descriptor Files, Refined Hits Array Descriptor File

1. Parse the Hits Array descriptor file to generate the list of documents, SSD file names, and the Hits Array.
2. Store the location of the testing documents, the SSD file names, and the Hits Array in single dimensional string arrays. Each element of the string array (storing hits) contains a string of numbers (hits).
3. Convert the Hits Array into a 2D array storing integer elements by splitting each element of the 1D Hits Array.
4. Perform singular value decomposition on the 2D integer Hits Array.
5. Cluster the columns of the 2D integer Hits Array into 15 clusters using K-means algorithm. Here columns represent SSD number.

6. Assign a weight to each and every Semantic Signature with the help of singular value decomposition.
7. Generate a 3D array that stores Semantic Signature weights arranged according to cluster-id.
8. Consider each cluster and store the Semantic Signatures in descending order of their weights. Consider the top 10 Semantic Signatures in each cluster. These are the refined Semantic Signatures for each of the clusters.
9. Store the refined SSD files at the location specified.
10. Reconstruct the Hits Array with refined Semantic Signatures.
11. Generate the Hits Array Descriptor File and store it.

Details of Step 1:

Hits Array Descriptor File (HADF) is an XML file. The HADF contains information about the location of the testing documents, the SSD file names and the corresponding Hits Array. It is required to parse this file using an XML parser in order to obtain the information.

Details of Step 2:

1D Hits Array is as follows,

10101010.....
00000000....
11010101....
11111111....
00000000.....

Details of Step 3:

It is converted into 2D Hits Array as shown below,

1	0	1	0	1	0	1	0	.....	
0	0	0	0	0	0	0	0	.....	
1	1	0	1	0	1	0	1	.....	
0	0	0	0	0	0	0	0	.....	

Details of Step 4:

Singular Value Decomposition (SVD) of M is defined as the factorization of M into the form $U\Sigma V^t$ where M is a $m \times n$ real or complex matrix, U is a $m \times m$ real or complex unitary matrix, Σ is a $n \times n$ diagonal matrix and V^t is a $n \times n$ real or complex unitary matrix. ^[4]

$$M_{[m \times n]} = U_{[m \times n]} \Sigma_{[n \times n]} V^t_{[n \times n]}$$

Details of Step 6:

The weight of each Semantic Signature is computed using the following formula,

$$Weight(SSD_i) = \sqrt{(V_{[i \times 1]} * V_{[i \times 1]}) + (V_{[i \times 2]} * V_{[i \times 2]}) + \dots (V_{[i \times n]} * V_{[i \times n]})}$$

Clustering the testing documents:

The refined Hits Array is given as input to the clustering algorithm. The rows of the array are clustered depending on the hits value. This in turn classifies the documents depending on their similarity.

3. Corrections

3.1. Pruning Keyword Sets in ALT:

While constructing the Keyword set, all possible permutations and combinations are applied between the keywords belonging to the two lists by taking one keyword from list1 and two keywords from list2. But many of the generated keyword sets are not required. As discussed earlier, let the five words in the list1 be ‘a’, ‘b’, ‘c’, ‘d’ and ‘e’. Let ‘a₁’, ‘a₂’ and ‘a₃’ be the top three highly correlated words with ‘a’ and ‘b₁’, ‘b₂’ and ‘b₃’ be the top three highly correlated words with ‘b’. Now the keyword sets generated are as follows,

a,a1,a2
a,a1,a3
a,a2,a3
b,a1,a2
b,a1,a3
b,a2,a3
b,b1,b2
b,b1,b3
b,b2,b3
:

We can observe that keyword ‘b’ is inappropriately combined with ‘a1’, ‘a2’ and ‘a3’. These keyword sets are not of interest and add noise. So such keyword sets can be eliminated. The new keyword sets after eliminating the unnecessary sets are as follows,

a,a1,a2
a,a1,a3
a,a2,a3
b,b1,b2
b,b1,b3
b,b2,b3
:

3.2. Increasing Weight of Keywords with Infinite IDF Value

If a word appears many times in a document and does not at all appear in an unrelated corpus then the word can be a significant word. When such a word is ignored then we could not retrieve important keywords that could uniquely represent the content of the document. So instead of ignoring the words, they should be assigned an IDF value greater than all the IDF values. So, a value (**MAX_VAL+1**) is assigned to them.

Example: For example, consider two text files "*Rose.txt*" and "*Jasmine.txt*". Let *Rose.txt* be the training document and *Jasmine.txt* be the corpus document. Here the keyword "*Rose*" would have IDF value as infinity because there is no word "*Rose*" in the corpus document ("*Jasmine.txt*"). So the keyword “*Rose*” would be ignored. But it is an important keyword for "*Rose.txt*". So, if all the words that have IDF value as infinity are ignored, then there is a chance of ignoring such important keywords. So, they should be given a weight greater than all the existing IDF values.

3.3. Removing documents that do not generate hits from the final clustering results

Few papers may not generate hits with the SSDs generated from the training phase. Representing these documents in the final output of cluster results might lead to confusion. It makes the process of analyzing the clusters difficult. So these papers are removed from the final cluster representation.

3.4. Testing the Hits Array before sending it as input to Semantic Signature Refinement Algorithm

Semantic Signature Refinement Algorithm works only for the condition where number of rows is greater than the number of columns. So number of Documents should always be greater than the number of SSDs for the routine to work correctly. This might not always be the case. So a check is included in the code to make sure that the number of rows is greater than the number of columns in the Hits Array.

Few enhancements are made to ensure that the code operates in the case where the documents are less than the SSDs in the Hits Array. These are described briefly in section 4.3.

4. Enhancements

4.1. Eliminating keyword sets based on the proximity measure used in the Keyword Tool

4.1.1. Data Structures

For finding the proximity among all the three keywords in a set, it is required to scan the document repeatedly. This is an overhead on the tool. So, a data structure that stores the locations of all the words (excluding stop words) is constructed. An array is used for this purpose.

For constructing the array that stores the locations, it is required to construct a few supporting data structures.

4.1.1.1. Supporting Data Structures

Supporting data structures include a hash table to store the count of number of times each keyword appears in the document, a hash table to store the start location of each keyword in the consolidated array, and an intermediary hash table for storing the location of next position of insertion in the array for a particular keyword (used solely for the purpose of constructing the array containing the consolidated list of locations).

Let us consider the text file *Rose.txt*.

Rose in the sky, Rose in the sea, Bowers of scarlet sky rose, Take my heart and me. God was good to make, This December weather, all this sky a rose garden, Rose and fire together.

To the East are burning, Rose in a garden, Rose in a rosy field, Hesper for their warden. Yonder to the West, all afire, Mirror now some rare splendid, Rose of their desire.

4.1.1.1.1. Hash table storing frequency of words

This data structure stores the count of number of times each word appears in the document.

Key	Value
Rose	8
Sky	3
Sea	1
:	:
:	:

Figure 4-1: Hash Table for Frequency

4.1.1.1.2. Hash table storing starting locations (in the location array) of each and every word

This data structure is mainly used for dynamically accessing the start location of a particular word in the location array.

Key	Value
Rose	1
Sky	9
Sea	12
:	:
:	:

Figure 4-2: Hash Table for Start Location

This data structure is constructed using the hash table that stores frequency of words.

For the very first word, it is 1. For the remaining words, it is

Start location = sum of frequencies of all the words that appear before this word in the list+1.

For example, start location of *Sky*= 8+1. Start location of *Sea*= (8+3) +1=11+1=12.

4.1.1.1.3. Hash table storing next location

This data structure is used while constructing the array storing the locations. The hash values of this data structure indicate the locations in the location array. The next location for a particular word indicates the next position (in the location array) at which the position (in the document) of this word is to be inserted. The value of the key keeps on changing for each subsequent insertion of location in to the consolidated array. If the word is scanned for the first time in the document, then the word is stored in the hash table with value as the base location of the corresponding word (using start location hash table). For each subsequent occurrence of the keyword, the existing value is incremented by 1. For instance, after scanning the first occurrence of the word "*sky*" in the document, the next-location hash table is as shown below.

Key	Value
Rose	1
Sky	10

After scanning the second occurrence of the word "*Sky*" in the document, the hash table is as follows.

Key	Value
Rose	2
Sky	11
Sea	12
Bowers	13
Scarlet	14

Figure 4-3: Hash Table for storing next location

Notice that the value for keywords *Rose*, *Sky* is incremented by 1 indicating that each of them occurred twice in the document. Next inserting location (if they occur for the third time) for *Rose* is 2 and that of *Sky* is 11.

4.1.1.2. Construction of the Data Structure (array storing the locations)

While scanning a document if a word is encountered, its next location is found out using the next location hash table. Depending on the value corresponding to the word, its position (in the document) is inserted into location array. Now the array that stores the locations is as follows (ignoring all the stop words).

Figure 4-4: Array Storing Locations

The words "in", "the", "of", "my", "me", "and", "was", etc. are considered as stop words. Ignoring these words, all the remaining words are taken into consideration and their respective locations are stored in the array data structure. Thus, at the end we have an array containing a dedicated chunk of indexes for each keyword in the document.

4.1.2. Proximity Measure

While finding the keyword2 (a_1) and Keyword3 (a_2) using the proximity measure, it is automatically ensured that 'a' is closer to ' a_1 ', and 'a' is closer to ' a_2 '. It is also required to make sure that ' a_1 ' is closer to ' a_2 '. For ensuring that there is interaction between all the 3 keywords in most of the document vectors, the following criterion is used:

$$Max[loc(a_1), loc(a_2)] - Min[loc(a_1), loc(a_2)] \leq Window\ Size$$

and

$$Number\ of\ windows\ holding\ this\ condition \geq 1$$

Maximum or minimum of the locations of a, a_1 , and a_2 are determined using the location array. This condition ensures that each word is within $2 * window\_size$ distance from each of the other two words for a majority of document vectors.

Previously, a huge numbers of Keyword Sets were generated from AKT. In order to avoid the overhead on the ALT, only the top 5 frequent words on the first list were considered in the AKT. The implementation of the above discussed condition in the Keyword Tool prunes most of the Keyword Sets that do not have all the keywords in a closer proximity to each other. The remaining Keyword Sets capture the meaningful content from the learning document. As a

result, the number of keywords considered on the first list can be increased tremendously. This process ensures that most of the content from the learning document is captured.

4.2. Proximity Measure used in the Hits Array Generator Tool

If a particular keyword in the Keyword Set occurs many times in a testing document, then the document generates hits with the associated Semantic Signature. In this case, that word might be used in a context different from the context in the original learning document. But there are more chances for the training document to be clustered with the learning document that generated the Semantic Signature. For example, consider the Keyword Set “UML, based, result”. It is generated from the learning document that is related to Unified Modeling Language. The words “based” and “result” are also used in the learning document to explain information related to the same concept. But these words can be common in other documents that might not be related to the UML concept. So the training document can get inappropriately clustered with the learning document.

In order to handle this scenario, two conditions have been proposed. One of which is too restrictive and the other is comparatively less restrictive. These conditions are applied on the document vectors that are returned from the Weight Calculation algorithm. They assume that the first keyword in the keyword set is important as it is generated based on the frequency. Only the document vectors that meet the condition are taken into consideration while calculating the hits.

4.2.1. First Condition

The following condition is used in the HAG Tool. This condition is was found to be too restrictive while selecting the document vectors.

$$dv[1][2]! = 0 \text{ or } dv[2][1]! = 0$$

and

$$dv[1][3]! = 0 \text{ or } dv[3][1]! = 0$$

In the above condition, dv is a two-dimensional array that holds the interaction between the keywords in the Keyword Set. The above condition ensures that there is interaction between the first word and the other two words in at least one direction.

4.2.2. Second Condition

This condition is less restrictive and ensures that there is interaction between the first word and at least one of the other two words in at least one of the two directions. The condition is as follows:

$$dv[1][2]! = 0 \text{ or } dv[2][1]! = 0$$

or

$$dv[1][3]! = 0 \text{ or } dv[3][1]! = 0$$

Experiments were conducted using both the conditions. The results are compared in the later sections.

4.3. Handling the case where the number of documents is less than the number of Semantic Signatures in Semantic Signature Refinement Tool

The SSR routine returns $U_{[m \times n]}, \Sigma_{[n \times n]}, V_{[n \times n]}$ matrices when the Hits Array is sent as input to it. When the set of learning documents is small, the number of rows is significantly less than the number of columns. In this case, the routine returns $U_{[m \times n]}, \Sigma_{[(m+1) \times (n+1)]}, V_{[n \times n]}$. Matrix multiplication is not possible with the three matrices.

In order to handle this case, the transpose of the Hits Array is sent as input to the SSR algorithm instead of the Hits Array. It returns $U'_{[m \times n]}, \Sigma'_{[n \times n]}, V'_{[n \times n]}$ matrices. In this case, U' matrix is used for assigning weight to the Semantic Signatures. The enhanced algorithm for Semantic Signature Refinement tool is as follows.

Algorithm Semantic_Signature_Refinement

Input: Hits Array Descriptor File

Output: Refined Semantic Signature Descriptor Files, Refined Hits Array Descriptor File

1. Parse the Hits Array descriptor file to generate the list of documents, SSD file names, and the Hits Array.
2. Store the location of the testing documents, the SSD file names, and the Hits Array in single dimensional string arrays. Each element of the string array (storing hits) contains a string of numbers (hits).
3. Convert the Hits Array into a 2D array storing integer elements by splitting each element of the 1D Hits Array.

4. Cluster the columns of the 2D integer Hits Array into 15 clusters using K-means algorithm.
5. Go to step 7 if the number of rows is greater than the number of columns.
6. Transpose the Hits Array and store it in the same array.
7. Send the Hits Array as input to the SSR routine.
8. Assign a weight to each and every Semantic Signature with the help of singular value decomposition.
9. Generate a 3D array that stores cluster-id as 1st dimension, the SSD names as 2nd dimension, and the weight assigned to the Semantic Signature as the 3rd dimension. This array stores Semantic Signatures and the corresponding weights arranged according to cluster-id.
10. Consider each cluster and store the Semantic Signatures in descending order of their weights. Consider the top 10 Semantic Signatures in each of the clusters. These are the refined Semantic Signatures for the cluster.
11. Store the refined Semantic Signatures' SSD files at the location specified.
12. Reconstruct the Hits Array with the refined Semantic Signatures.
13. Generate the Hits Array Descriptor File and store it.

Details of Step 8:

If $rows \geq columns$, then weight for each Semantic Signature (SS) is computed as shown below,

$$Weight(SS_i) = \sqrt{(V_{[i \times 1]} * V_{[i \times 1]}) + (V_{[i \times 2]} * V_{[i \times 2]}) + \dots (V_{[i \times n]} * V_{[i \times n]})}$$

If $rows < columns$, then the weight for each SSD is computed as follows,

$$Weight(SS_i) = \sqrt{(U'_{[i \times 1]} * U'_{[i \times 1]}) + (U'_{[i \times 2]} * U'_{[i \times 2]}) + \dots (U'_{[i \times n]} * U'_{[i \times n]})}$$

5. Supporting Tools

Chapter 5 describes the tools that are used to retrieve specific documents of interest and to automate the process of measuring document retrieval rate. For measuring the precision of document retrieval, it is required to extract documents belonging to a specific category, divide the SSD files into groups, assign group ID to each of these SSDs, and find the count of the

documents retrieved by each of the filter. Tools designed for these purposes include Automated Text File Extractor (ATFE), Automated Group Inserter (AGI), and Automated Value Extractor (AVE). The following sections give a detailed explanation about working of each tool.

5.1. Automated Text File Extractor

As a part of the experiment, a huge-corpus of very short XML documents is used. These documents are categorized into sub-topics by using category tags, which are provided with the data. In the training phase of the experiment, the documents belonging to a specific category from the corpus are used. It is highly impossible to manually search the entire corpus for documents of a particular category. So this process is automated using ATFE tool.

5.1.1. Input for the Automated Text File Extractor Tool

The input for the Automated Text File Extractor Tool is an XML file (that contains a category code) and the category of documents to be retrieved. In the XML file, the value of the category tag indicates the category to which the document belongs and the value of the paragraph tag indicates the content of the document. The following is the structure of the XML document.

```

<?xml version="1.0" encoding="iso-8859-1" ?>
<newsitem itemid="158959" id="root" date="1996-11-01" xml:lang="en">
<title>CANADA: Court approves Horsham, Trizec merger.</title>
<headline>Court approves Horsham, Trizec merger.</headline>
<dateline>TORONTO 1996-11-01</dateline>
<text>
<p>Horsham Corp and Trizec Corp Ltd said on Friday that an Ontario Court has
approved the merger of the two companies into a new real estate firm, Trizec Hahn
Corp.</p>
</text>
<copyright>(c) Reuters Limited 1996</copyright>
<metadata>
<codes class="bip:countries:1.0">
<code code="CANADA">
<editdetail attribution="Reuters BIP Coding Group" action="confirmed" date="1996-11-
01" />
</code>
</codes>
<codes class="bip:industries:1.0">
<code code="I21000">
<editdetail attribution="Reuters BIP Coding Group" action="confirmed" date="1996-11-
01" />
</code>
<code code="I85000">
<editdetail attribution="Reuters BIP Coding Group" action="confirmed" date="1996-11-
01" />
</code>
</codes>
<codes class="bip:topics:1.0">
<code code="C18">
<editdetail attribution="Reuters BIP Coding Group" action="confirmed" date="1996-11-
01" />
</code>
<code code="C181">
<editdetail attribution="Reuters BIP Coding Group" action="confirmed" date="1996-11-
01" />
</code>
<code code="CCAT">
<editdetail attribution="Reuters BIP Coding Group" action="confirmed" date="1996-11-
01" />
</code>
</codes>
<dc:element="dc:date.created" value="1996-11-01" />
<dc:element="dc:publisher" value="Reuters Holdings Plc" />
<dc:element="dc:date.published" value="1996-11-01" />
<dc:element="dc:source" value="Reuters" />
<dc:element="dc:creator.location" value="TORONTO" />
<dc:element="dc:creator.location.country.name" value="CANADA" />
<dc:element="dc:source" value="Reuters" />
</metadata>
</newsitem>

```

Figure 5-1 Structure of Input XML file

5.1.2. Working of Automated Text File Extractor Tool

Automated Text File Extractor Tool (ATFET) takes XML documents and category tags as input. For each XML document in the folder, it parses the file and extracts the value of the code tag. If it matches with the input category code then it extracts the data corresponding to the paragraph tag. Thus, the content of all the documents belonging to the specified code are placed in a text file and saved at the specified location.

Figure 5-2 Working of Automated Text File Extractor

5.2. Automated Group Inserter

The refined Semantic Signatures are divided randomly into 4 groups. These Semantic Signatures' SSD files are given as input to the Automated Value Extractor Tool (AVET). AVET needs to recognize the group (to which a particular SSD file belongs) for calculating the count of documents extracted by each of the group. So the AGI tool is used to insert an additional tag called *group* into the SSD file with the value as the ID of the group to which the SSD file belongs. The following sections explain the input, the working, and the output of the AGI tool.

5.2.1. Input for the Automated Group Inserter Tool

A folder containing SSD files and the Group ID to be inserted is given as input to the AGI tool. The figure below depicts the structure of the SSD file.

```
<AutomatedLearnerTool version="1.1">
<kdfSource>C:\Users\udaya\Desktop\video_partitioning_adaptive.KDF</kdfSource>
<source folder="no" file="yes">C:\Users\Kiran\Desktop\adjero_h_main.txt</source>
<clusterer name="kmeans">6</clusterer>
<centroid r="0.708296971721544" distanceMeasure="CD">0.0851, 0.7176, 0.0815, 0.0913, 0.0326,
0.0513, 0.3067, 0.3167, 0.0308</centroid>
<vectors>0.3162, 0.7661, 0.3846, 0.3363, 0.3162, 0.4138, 0.7082, 0.6565, 0.3162, 0.5173, 0.6073, 0.4829, 0.
8547, 0, 0, 0.807, 0.9806, 0</vectors>
<stemming used="no" stemmer="porter"></stemming>
<windowLength length="20"></windowLength>
<functionConstant a="5"></functionConstant>
<keywords>adaptive, video, partitioning</keywords>
<synonyms> see, watch; hear, listen </synonyms>
<phrases> throat singing; throat cancer </phrases>
</AutomatedLearnerTool>
```

Figure 5-3: Structure of Semantic Signature File

5.2.2. Working of the Automated Group Inserter Tool

The SSD folder and the group ID is given as input to the AGI tool. For each SSD file in the folder, this tool inserts the *group* tag with value as the given ID at the specified location in the SSD file. After insertion, it saves all the modified SSD files in a folder at a given location.

Figure 5-4: Working of Automated Group Inserter Tool

5.2.3. Output of the Automated Group Inserter Tool

The following is the modified SSD file with the *group* tag (highlighted in bold) inserted. Now this SSD file can be given as input to the AVE tool. This tool can easily parse the document to extract the group ID of the SSD file.

```

<AutomatedLearnerTool version="1.1">
<kdfSource>C:\Users\udaya\Desktop\video_partitioning_adaptive.KDF</kdfSource>
<source folder="no" file="yes">C:\Users\Kiran\Desktop\adjeroh_main.txt</source>
<clusterer name="kmeans">6</clusterer>
<centroid r="0.708296971721544" distanceMeasure="CD">0.0851, 0.7176, 0.0815, 0.0913, 0.0326,
0.0513, 0.3067, 0.3167, 0.0308</centroid>
<vectors>0.3162,0.7661,0.3846,0.3363,0.3162,0.4138,0.7082,0.6565,0.3162;0.5473,0.6073,0.4829,0.
8547,0,0,0.807,0.9806,0</vectors>
<stemming used="no" stemmer="porter"></stemming>
<group id="1"></group>
<windowLength length="20"></windowLength>
<functionConstant a="5"></functionConstant>
<keywords>adaptive, video, partitioning</keywords>
<synonyms> see, watch; hear, listen </synonyms>
<phrases> throat singing; throat cancer </phrases>
</ AutomatedLearnerTool >

```

Figure 5-5: Structure of Enhanced Semantic Signature File

5.3. Automated Value Extractor Tool

After inserting the group ID, the refined SSD files and the HAD file generated from the testing document are given as input to the Automated Value Extractor Tool (AVET). It outputs the count of the number of documents extracted by the single filter or a combination of filters.

5.3.1. Working of the Automated Value Extractor Tool

The refined SSD files (having group ID) and the HAD file are given as input to the AVEFT. The AVEFT parses the HAD XML document and the SSD files. Using the Hits Array and the group ID from the SSD file, it computes the number of documents retrieved by each of these filters and displays the results as output on the console. The following section describes the algorithm for the AVET.

Figure 5-6: Working of Automated Value Extractor

5.3.2. Algorithm for Automated Value Extractor Tool

Algorithm Value_Extraction

Input: Hits Array Descriptor File, Refined SSD Files with Group ID

Output: Count of files extracted by the filters

1. Parse the Hits Array descriptor file to generate the list of documents, SSD file names, and the Hits Array.
2. Consider a document (row) from the Hits Array.

3. Set the flags corresponding to all the counters to false.
4. Consider a SSD file (column) from the Hits Array.
5. Find the value in the Hits Array corresponding to the document and the SSD.
6. If the value is greater than zero, then parse the SSD file to extract the Group ID. If it is 0, go to step 4.
7. Set the flag corresponding to the group to true.
8. Go to step 4 until all SSD files are considered.
9. Check the flags and increment the corresponding counters by 1.
10. Go to step 2 until all the documents are considered.

Details of Step 9:

Let f_1, f_2, f_3 , and f_4 be the flags. After each row is processed all these flags are checked. If f_1 is true and all the remaining flags are false, then the counter corresponding to filter 1 is incremented by 1. If both f_1 and f_2 are set to true, then the values of counters n_1, n_2, n_{12} are incremented by 1. If the flags f_1, f_2, f_3 are set to true, then the values of counters $n_1, n_2, n_3, n_{12}, n_{13}, n_{23}, n_{123}$ are incremented by 1. Likewise, all of the counters are incremented.

Example for explaining the functioning of the tool

Let the Hits Array be,

	SSD ₁	SSD ₂	SSD ₃	SSD ₄	SSD ₅	SSD ₆	SSD ₇	SSD ₈
D₁	2	0	0	0	0	0	0	0
D₂	0	5	0	0	0	0	0	0
D₃	0	0	6	0	0	0	0	0
D₄	0	0	0	0	6	9	0	0
D₅	0	0	0	0	0	0	5	0
D₆	1	0	2	5	0	0	0	0
D₇	3	0	2	0	5	0	0	0
D₈	10	0	1	0	3	0	0	2

Figure 5-7: Hits Array

The above shown Hits Array represents the hits generated by 8 documents with 8 SSD files. Let g_1, g_2, g_3, g_4 and their corresponding Group IDs be as follows,

$g_1 = \{SSD_1, SSD_2\}$ and group ID=1

$g_2 = \{SSD_3, SSD_4\}$ and group ID=2

$g_3 = \{SSD_5, SSD_6\}$ and group ID=3

$g_4 = \{SSD_7, SSD_8\}$ and group ID=4

In the above example,

D₁ is retrieved by filter **1** because SSD_1 belongs to g_1

D₂ is retrieved by filter **1** because SSD_2 belongs to g_1

D₃ is retrieved by filter **2** because SSD_3 belongs to g_2

D₄ is retrieved by filter **3** because SSD_5 belongs to g_3

D₅ is retrieved by filter **4** because SSD_7 belongs to g_4

D₆ is retrieved by filter **1** and filter **2** because SSD_1 belongs to g_1 , and SSD_3 and SSD_4 belong to g_2

D₇ is retrieved by filters **1, 2, and 3**; SSD_1, SSD_3 , and SSD_5 belong to g_1, g_2 , and g_3 respectively

D₈ is retrieved by filters **1, 2, 3, and 4**; SSD_1, SSD_3, SSD_5 , and SSD_8 belong to g_1, g_2, g_3 , and g_4

Let the counters be **n₁, n₂, n₃, n₄, n₁₂, n₁₃, n₁₄, n₂₃, n₂₄, n₃₄, n₁₂₃, n₁₂₄, n₂₃₄, n₁₂₃₄**.

n₁: counter for number of documents retrieved by the filter 1=5

n₂: counter for number of documents retrieved by the filter 2=4

n₃: counter for number of documents retrieved by the filter 3=3

n₄: counter for number of documents retrieved by the filter 4=2

n₁₂: counter for number of documents retrieved by the filters 1 and 2=3

n₁₃: counter for number of documents retrieved by the filters 1 and 3=1

n₁₄: counter for number of documents retrieved by the filters 1 and 4=1

n₂₃: counter for number of documents retrieved by the filters 2 and 3=3

n₂₄: counter for number of documents retrieved by the filters 2 and 4=1

n_{34} : counter for number of documents retrieved by the filters 3 and 4=1

n_{123} : counter for number of documents retrieved by the filters 1, 2, and 3=2

n_{124} : counter for number of documents retrieved by the filters 1, 2, and 4=1

n_{134} : counter for number of documents retrieved by the filters 1, 3, and 4=1

n_{234} : counter for number of documents retrieved by the filters 2, 3, and 4=1

n_{1234} : counter for number of documents retrieved by the filters 1, 2, 3, and 4=1

6. Experimental Setup

Chapter 6 describes the experimental approach used to benchmark the performance of enhanced tools in the automated package.

6.1. Experiments for Evaluating the Performance of Enhanced Tools in the Package

6.1.1. Objective

The objective of the experiment is to assess the performance of the enhanced AKT, ALT, and the HAG Tool. It also aims at comparing the two proposed conditions in the HAG Tool.

6.1.2. Corpus

The corpus used in this experiment was collected by Barnes, Eschen, Para and Peddada [6] [7], which contains 54 research based papers out of which 9 papers are written by 9 Professors of Lane Department of Computer Science and Electrical Engineering, West Virginia University and remaining 45 papers are the collection of 5 references taken from each of the nine main papers. The 9 papers written by 9 Professors are termed as main papers in this document. Even though it is a small corpus, it contains papers with diversified content. As it is a small corpus, a manual analysis of the result is feasible. [5]

6.1.3. Notation

As it is difficult to refer and represent the full names of professors every time, two letters of their name are used as abbreviation. Table 6.1 gives information about the professors' names and the abbreviations.

Professor Name	Abbreviation
Dr. Donald Adjeroh	AD
Dr. Bojan Cukic	CU
Dr. Hany Ammar	HA
Dr. Katerina Goseva - Popstojanova	KA
Dr. Natalia Schmid	NA
Dr. Daryl Reynolds	RE
Dr. Arun Ross	RO
Dr. Tim Menzies	TI
Dr. Mathew Valenti	VA

Table 6-1: Abbreviations for Professor's Names

6.1.4. Procedure

The automated tools are trained on the nine main papers. The Keyword Sets and the Semantic Signatures are extracted from each of the 9 main papers separately using AKT and ALT, respectively. All the 45 reference papers are taken as the corpus used to calculate the IDF. The 54 papers are taken as the testing set which includes the main and reference papers.

Both AKT and ALT are run 9 times. Each of the 9 main papers is given as an input learning document for each single run of AKT. Each of the 9 main papers and the corresponding set of KDFs are given as input for each run of ALT. ALT gives SSDs as output. All the 9 SSD folders are combined into a single folder. The combined SSD folder and the testing set are given as input to the HAG Tool. The generated HADF is given as input to SSR tool for further refinement. It generates refined Semantic Signatures and the corresponding HADF. This HADF is given as input to SSR and the Hits Array is clustered using the K-means algorithm.

Figure 6-1: Flow chart for the Experimental Procedure

6.2. Multi-category Document Retrieval

6.2.1. Objective

The objective of this experiment is to analyze the ability of the automated tools in handling and retrieving very short documents specific to a category.

6.2.2. Corpus

For these experiments we use the Reuters Corpus Volume 1 (RCV1). This set of documents contain approximately one year of Reuters wire service articles to which a set of tags has been manually added to indicate the type of content or category of each article. All the documents present in this corpus are very small. The training and testing corpora for this experiment were taken from this corpus. Documents belonging to a specific category were selected for training phase of the experiment. In this study we used articles with the category tags General Science (GSCI). We chose to retrieve documents relating to the sub-topic of space science from among document with the GSCI tag.

6.2.3. Modifications

This experiment uses large volume of documents in both the training as well as testing phases. But, the length of each document is significantly smaller when compared to documents in the previous experiments. A set of experiments were conducted and a few modifications were made in the tools in order to make them suitable for this experiment. The following sections explain the experiments conducted, the observations made, and the modifications done.

6.2.3.1.Limitation of Maximum Number of Files in a Directory

At first, all the training and testing documents are stored on a flash drive and the experiment was conducted by giving the input from that device. This seriously limited the number of files that can be stored on a directory. A single directory can hold a maximum of 21,844 documents. The number of training documents was 67,654 and the corpus contained 1,250,766 documents. Later it was found that this is due to the file system (FAT) that is being used by the flash drive. All the files were moved to the hard-disk. This removed the limitation.

6.2.3.2.Change in the Weight Measure

All the XML documents in the Reuters database are news articles. So the words like “said”, “told”, etc. are very common. These words are comparatively less important. When the Semantic Signatures are chosen manually, the Keyword Sets that contain the words such as “said” can be avoided from consideration. But in automation, it is required to train the Keyword tool to intelligently identify such keywords and ignore them. So the weight measure is redefined as follows,

$$weight = TF \times \frac{1}{IDF^2}$$

This effectively reduced the weight of the above mentioned keywords. But, names of scientists, or writers occupied the top positions on the frequency list.

6.2.3.3. Increasing the Number of Frequent Words Considered

The number of words considered from the first list (built using TF-IDF) was increased to 300. But, words uniquely representing a category are far down in the list and so, 300 was not enough. The size of the first list would need to be increased significantly to 450. This was not efficient and increased load on AKT and ALT. Moreover, there was no significant improvement in the output produced. So the size of the first list was put back to 25 and the formula for IDF was revisited.

6.2.3.4. Redefining the IDF Equation

Previously the formula used in the IDF tool was

$$IDF = \log\left(\frac{\text{Number of files}}{\text{Number of documents in which the term appears atleast once}}\right)$$

All the documents in the corpus are significantly small and each and every document contains words like “said”, “told”, etc. repeated many times. In the learning phase all the documents are combined into a single text document. So IDF measure described above is insufficient to decrease the weight of such words. It was redefined as follows.

$$IDF = \log\left(\frac{\text{Number of files}}{\text{Total number of times the word occurs in entire corpus}}\right)$$

This effectively removed the common yet insignificant words from the first list. However they showed up in the second list. So TF-IDF is used along with the proximity heuristic in the second list.

6.2.3.5. Using both TF-IDF and Proximity Heuristic in Second List

As the corpus is very large, the bottom 1% of words arranged in descending order of TF-IDF are usually of less significance. So the bottom 1% words are prevented from being considered in the proximity list. Using both the TF-IDF and this proximity heuristic on the second list leads to the formation of more meaningful Keyword Sets that effectively capture semantic content specific to a topic.

6.2.4. Procedure

In this experiment, 67,654 files belonging to Reuters corpus are given as input to AFET tool along with the category tag GSCI. The AFET tool extracted 170 files belonging to GSCI

category. These 170 files were taken as training corpus and are not included as a subset of testing corpus. The testing corpus is also taken from Reuters corpus and consists of 67,952 newswire articles for the month of November 1996. 1,250,766 documents from Reuters Corpus are taken as our corpus (to calculate IDF). KDFs and SSDs are generated from all the training documents using the Automated Keyword Tool and Automated Learner Tool, respectively. These SSDs and the training documents are given as input to HAG Tool for the Semantic Signature refinement phase. The generated HAD file is given as input to SSR tool to produce a set of refined SSD files. The refined SSDs and the testing documents are then given as input to HAG Tool. The output of the HAG Tool (HADF) is analyzed to find the number of space related documents retrieved.

Figure 6-2: Procedure for Document Retrieval Experiment

6.3. Statistical Method for Estimating Precision of Document Retrieval: True Positive and False Positive Rates

6.3.1. Objective

The objective of this experiment is to analyze the ability of the automated tools in retrieving documents specific to a category by employing statistical hypothesis to measure the retrieval rates.

6.3.2. Assumptions and Theory Behind the Experiment

The reliability of the Semantic Signatures generated using the automated tools is measured using a statistical measure. The intersection between training and testing corpora is the empty set. Let us assume there are D documents in the testing corpus. Out of these documents, let N be the number of documents of interest. $(D-N)$ is the number of documents that are not of interest. To conduct this experiment, refined Semantic Signatures are generated from documents in the training corpus using SSR Tool. These refined Semantic Signatures are divided into various retrieval filters. All the filters are assumed to be independent of each other. We further determine the probabilities for filters retrieving documents of interest, probabilities for filters retrieving documents that are not of interest, and the percentage of documents that are of interest in the testing set of documents. We find the best fit values for these parameters by minimizing the Chi-Squared value. The below table is used in making a decision on the number of groups into which the Semantic Signatures are divided.

Number of filters	Parameters to be determined	Known Values
1	3 (p_1, q_1, d)	1 (n_1)
2	5 (p_1, q_1, p_2, q_2, d)	3 (n_1, n_2, n_{12})
3	7 ($p_1, q_1, p_2, q_2, p_3, q_3, d$)	6 ($n_1, n_2, n_3, n_{12}, n_{13}, n_{123}$)
4	9 ($p_1, q_1, p_2, q_2, p_3, q_3, p_4, q_4, d$)	15 ($n_1, n_2, n_3, n_4, n_{12}, n_{13}, n_{14}, n_{23}, n_{24}, n_{34}, n_{123}, n_{124}, n_{234}, n_{1234}$)

Table 6-2 Filter Count Determination

In the above table, p_x is the probability of retrieving documents of interest by filter x , q_x is the probability of retrieving documents that are not of interest by the filter x , d is the percentage of documents that are of interest in the testing document set, n_x is the number of documents

retrieved by x^{th} filter, n_{xy} is the number of documents retrieved both by filter x and filter y , and so on.

Upon observing the above table, we can conclude that the smallest number of filters that will allow us to fit the measured data with a positive number of degrees of freedom is 4. Specifically 9 parameters can be easily determined by 15 measured values. It has a positive degree of freedom ($15-9=6$). So we divided the SSDs into 4 independent filters in our experiment. The parameters in the experiment are as follows:

- p_1 - Probability of retrieving documents of interest by filter 1
- q_1 - Probability of retrieving documents that are not of interest by filter 1
- p_2 - Probability of retrieving documents of interest by filter 2
- q_2 - Probability of retrieving documents that are not of interest by filter 2
- p_3 - Probability of retrieving documents of interest by filter 3
- q_3 - Probability of retrieving documents that are not of interest by filter 3
- p_4 - Probability of retrieving documents of interest by filter 4
- q_4 - Probability of retrieving documents that are not of interest by filter 4
- d - Percentage of documents that are of interest in the testing set

The known values in this experiment are as follows,

- n_1 : number of documents retrieved by the filter 1
- n_2 : number of documents retrieved by the filter 2
- n_3 : number of documents retrieved by the filter 3
- n_4 : number of documents retrieved by the filter 4
- n_{12} : number of documents retrieved by the filters 1 and 2
- n_{13} : number of documents retrieved by the filters 1 and 3
- n_{14} : number of documents retrieved by the filters 1 and 4
- n_{23} : number of documents retrieved by the filters 2 and 3
- n_{24} : number of documents retrieved by the filters 2 and 4
- n_{34} : number of documents retrieved by the filters 3 and 4

n_{123} : number of documents retrieved by the filters 1, 2, and 3

n_{124} : number of documents retrieved by the filters 1, 2, and 4

n_{134} : number of documents retrieved by the filters 1, 3, and 4

n_{234} : number of documents retrieved by the filters 2, 3, and 4

n_{1234} : number of documents retrieved by the filters 1, 2, 3, and 4

In this experiment 9 parameter values are determined by minimizing chi-square value using 15 known values. The Chi-Square function we use is based on the statistical assumption that each of the filters is statistically independent of the other filters. Under this assumption we compute the estimated value of each of the measured quantities. For example the expected value for n_{23} can be computed as

$$\langle n_{23} \rangle = D \times d \times (p_2 \times p_3) + D \times (1 - d) \times (q_2 \times q_3)$$

A goodness of fit function using the sum of the terms of the form

$$\frac{(\langle n_{23} \rangle - n_{23})^2}{\sigma_{23}^2}$$

6.3.3. Procedure

The corpus used for this experiment is same as the one used for the previous experiment. The procedure is similar to the experiment described in section 6.2 up to generation of refined SSDs. These refined SSD files are now divided into four groups randomly. Group ID is assigned to each of the SSD files depending on the division using AGI tool. These groups are used to implement four filters. These filters retrieve documents that are similar to the ones used in generation of the SSD files. Refined SSDs and the testing documents are given as input to HAG Tool. The generated HADF and refined SSDs with group ID are given as input to AVE tool. AVE outputs the count of the documents retrieved by each of these filters and also by the combination of these filters. These values are given as input to the MATLAB routine. This routine determines the best-fit value for each of the 9 parameters by minimizing the Chi-Squared value.

Figure 6-3: Procedure for Chi-Square Test

7. Experimental Results

This chapter contains the results for the experiments described in Chapter 6. Section 7.1 describes the results of the experiment described in Section 6.1 for assessing the effects of enhancements on the output of the tools.

7.1. Experimental Results for Evaluating the Performance of Enhanced Tools

The following sub-sections explain inputs, outputs and the results for various experiments conducted as a part of analyzing the performance. Section 7.1.1 describes the results for the previous version of automated tools. Section 7.1.2 describes the results for the enhancements made on the Automated Keyword Tool and the Automated Learner Tool. Sections 7.1.3 and 7.1.4 compare the proposed conditions in the Hits Array Generator Tool. Section 7.1.5 shows the results obtained by increasing the number of keywords considered on the first list in AKT.

7.1.1. Results from Previous Version of Automated SSMinT

7.1.1.1. Inputs and Outputs for Training tools

In this experiment only the top 5 frequent keywords on list 1 in AKT were considered. The following table describes the input given and the output obtained from the previous version of the automated tools.

Paper	AKT		ALT	
	<i>Input</i>	<i>Output (KDF)</i>	<i>Input (KDF)</i>	<i>Output (SSDs)</i>
AD	AD main paper and corpus	45	45 KDFs and AD main paper	40
CU	CU main paper and corpus	149	149 KDFs and CU main paper	99
HA	HA main paper and corpus	125	125 KDFs and HA main paper	128
KA	KA main paper and corpus	213	213 KDFs and KA main paper	122
NA	NA main paper and corpus	192	192 KDFs and NA main paper	110
RE	RE main paper and corpus	172	172 KDFs and RE main paper	81
RO	RO main paper and corpus	175	175 KDFs and RO main paper	64
TI	TI main paper and corpus	232	232 KDFs and TI main paper	34
VA	VA main paper and corpus	299	299 KDFs and VA main paper	50

Table 7-1: Inputs and Outputs for the training tools of SSMInT

In ALT, the K-Means clustering algorithm is used with cosine distance as distance measure. Generated document vectors are allowed to be grouped into 2 clusters. All generated clusters are saved. 9 SSD folders generated from the Learner Tool are combined into a single SSD folder. It has a total of 728 Semantic Signature Descriptor files.

7.1.1.2. Inputs and Outputs for Testing Tools

The following table shows the inputs and outputs for the testing tools of SSMinT.

Name of the tool	Input / Output	Description
HAGT	Input	54 papers and 728 semantic signatures
	Output	Hits Array Descriptor File
SSRT	Input	Hits Array Descriptor File
	Output (Refinement)	Hits Array Descriptor File and 110 potential semantic signatures.
	Input	Hits Array Descriptor File, K-Means clustering with cosine distance as distance measure. Document feature vectors are allowed to be grouped into 9 clusters.
	Output	Clusters of documents.

Table 7-2 Inputs and Outputs for Testing tools of SSMinT

7.1.1.3. Clustering Results

The papers in the testing corpus are placed in nine clusters. Table 7-3 gives the information about papers in each cluster.

	AD	CU	HA	KA	NA	RE	RO	TI	VA
C0	M,2				3,4				
C1		M,1,2,3,4,5	M,1,3,5	M,1,2,3,4,5				M,1	
C2						1			1
C3	1		2,4		M,1,2,5				
C4									
C5						5			
C6						M,2			2
C7							M,1,2,3,4,5		
C8	3,4,5					3,4		2,3,4,5	M,3,4,5

Table 7-3: Semantic feature vector clustering results

7.1.1.4. Interpretation of the Clustering Results

Two papers of AD, two papers of TI, and one paper of VA did not generate hits. They are not filtered from the final output. They occupy one of the 9 clusters leading to confusion in the output. Most of the references of TI and VA were placed into same cluster. TI's papers deal with "Software Development" and "Artificial Intelligence" whereas VA's papers deal with "Wireless Communication", "Wireless Networks", and "Co-op Diversity".

7.1.2. Results from Enhanced Version of Automated SSMinT

7.1.2.1. Inputs and Outputs for Training tools

In the previous version, the number of keywords considered from the first list is limited by the number of Keyword Sets produced by the AKT. There is huge number of Keyword Sets produced in AKT. So only the top 5 frequent keywords are considered from the first list. These Keyword Sets are further pruned in ALT.

With the enhancements made in AKT, most of the irrelevant Keyword Sets get eliminated in AKT itself. As a result, the number of Keyword Descriptor Files produced from AKT and Semantic Signature Descriptor files produced from ALT is reduced drastically. Therefore, the number of frequent words considered from the first list can be increased. In this experiment the number of words from the first list is increased to 10. The following table describes the input given and the output obtained from the training tools of SSMinT.

Paper	AKT		ALT	
	<i>Input</i>	<i>Output (KDFs)</i>	<i>Input (KDFs)</i>	<i>Output (SSDs)</i>
AD	AD main paper and corpus	51	51 KDFs and AD main paper	57
CU	CU main paper and corpus	49	49 KDFs and CU main paper	62
HA	HA main paper and corpus	56	56 KDFs and HA main paper	83
KA	KA main paper and corpus	55	55 KDFs and KA main paper	60
NA	NA main paper and corpus	53	53 KDFs and NA main paper	81
RE	RE main paper and corpus	55	55 KDFs and RE main paper	75
RO	RO main paper and corpus	42	42 KDFs and RO main paper	52
TI	TI main paper and corpus	48	48 KDFs and TI main paper	26
VA	VA main paper and corpus	56	56 KDFs and VA main paper	45

Table 7-4: Inputs and Outputs for learning tools

In ALT, the K-Means clustering algorithm is used with cosine distance as distance measure. Generated document vectors are allowed to be grouped into 2 clusters. All generated

clusters are saved. 9 SSD folders generated from the learner tool are combined into a single SSD folder. It has a total of 541 Semantic Signature Descriptor files.

7.1.2.2. Inputs and Outputs for Testing Tools

The following table shows the inputs and outputs for the testing tools of SSMinT.

Name of the tool	Input / Output	Description
HAGT	Input	54 papers and 541 semantic signatures
	Output	Hits Array Descriptor File
SSRT	Input	Hits Array Descriptor File
	Output (Refinement)	Hits Array Descriptor File and 126 potential semantic signatures.
	Input	Hits Array Descriptor File, K-Means clustering with cosine distance as distance measure. Document feature vectors are allowed to be grouped into 9 clusters.
	Output	Clusters of documents.

Table 7-5: Inputs and Outputs for Testing Tool

7.1.2.3. Clustering Results

The papers in the testing corpus are placed in nine clusters. Table 7.1.2.3 gives information about papers in each cluster.

	AD	CU	HA	KA	NA	RE	RO	TI	VA
C0		4	2,3					M,1,3	
C1						1,2			M,1,2,5
C2							M,1,2,3,5		
C3						M,3,4			4
C4					2				
C5					M,1,5,4				
C6			M,1,4,5	M,2,3,4,5			4		
C7						5			
C8	M,2,4	M,1,2,3,5		1	3				

Table 7-6: Semantic feature vector clustering results

7.1.2.4. Interpretation of Clustering Results:

Three papers of AD, three papers of TI, and one paper of VA did not generate any hits. These papers were filtered off from the output in order to avoid confusion. The following is the analysis of the individual clusters.

Cluster0 has some of the CU and HA papers (or their references) along with the main and reference papers of TI. This is because the reference paper 4 of CU, reference papers 2, 3 of HA has hits for a Semantic Signature with Keyword Set ‘clumps’, ‘number’, ‘small’. However, ‘clumps’ does not appear in any of the CU or HA papers. But, the words ‘number’ and ‘small’ appear many times in these papers. So these documents are hitting that Semantic Signature. Cluster1 has some of RE and VA papers (or their references). All these papers talk about the concept of “*Wireless Networks*”. Cluster2 contains the entire main and reference papers of RO. All these papers talk about the concepts “*Biometrics*” and “*Image processing*”. Cluster3 contains the main paper, the reference papers 3, 4 of RE and reference paper 4 of VA. These papers mostly talk about the concepts “*Wireless Networks*” and “*Co-op diversity*”. Cluster4 is occupied by the reference paper 2 of NA. Cluster5 contains main paper and 3 reference papers of NA. They talk about “*Image processing*” and “*Object/target recognition*”. Cluster6 contains HA’s main paper and 1, 4, 5 reference papers grouped along with most of the KA’s papers. This is because the HA’s papers have many hits for a Semantic Signature with keywords ‘probability’, ‘component’ and ‘reliability’. This Semantic Signature is generated from the KA main paper. Cluster7 has reference paper 5 of RE. It talks about the concept “*Wireless Networks*”. Cluster8 has AD’s main, and reference papers 2, 4, CU’s main, and CU’s reference papers 1, 2, 3, 5, KA’s reference paper 1, and NA’s reference paper 3. This is because the KA and NA reference papers have hits for a Semantic Signature with keywords ‘testing’, ‘random’, ‘failure’. This Semantic Signature is generated from the main paper of CU.

Cluster0 has reference paper 4 of CU and HA’s 2, 3 reference papers. This is because they have hits for a Semantic Signature with keywords “clumps”, “number”, “small” and another Semantic Signature with keywords “software”, “ieee”, “pp”. These Semantic Signatures are generated from the main paper of TI. Similarly, RO’s reference paper 4 generated hits for a Semantic Signature with keywords “search”, “random”, “complete”. Manual analysis of the paper revealed the fact that this is due to the higher frequency of the word “search” in the paper. The other two words are not at all found in the paper. KA’s reference paper 1 and NA’s reference paper 3 got clustered with CU’s papers. This is because they are generating hits for a Semantic Signature with keywords “testing”, “random”, “failure”. KA’s reference paper has a lot of hits for that Semantic Signature and few hits for the Semantic Signature generated from the main paper of KA. NA’s reference paper has hits only for the Semantic Signature generated from

CU's main paper. Therefore, we imposed additional conditions on the HAG Tool to ensure that there is interaction between the keywords in the keyword group of the Semantic Signature.

7.1.3. Results Obtained by Imposing the First Condition on HAG Tool

The first condition is explained in Section 4.2.1. It ensures that there is interaction between the first keyword with both the second and third keywords in at least one direction. The modifications are only made to the HAG Tool. So the inputs and the outputs for AKT, and ALT are same as the ones shown in Table 7-4.

7.1.3.1. Inputs and Outputs for Testing Tools

The following table briefs the inputs and outputs for the testing tools.

Name of the tool	Input / Output	Description
HAGT	Input	54 papers and 541 semantic signatures
	Output	Hits Array Descriptor File
SSRT	Input	Hits Array Descriptor File
	Output (Refinement)	Hits Array Descriptor File and 102 potential semantic signatures.
	Input (Clustering)	Hits Array Descriptor File, K-Means clustering with cosine distance as distance measure. Document feature vectors are allowed to be grouped into 9 clusters.
	Output	Clusters of documents.

Table 7-7: Inputs and Outputs for Testing Tool

7.1.3.2. Clustering Results

The papers in the testing corpus are placed in nine clusters. Table 7-8 gives the information about papers in each cluster.

	AD	CU	HA	KA	NA	RE	RO	TI	VA
C0								M	
C1					M,1,5				
C2				M,1,2,3,4,5					
C3									
C4						1,2,3,4			M,2,4,5
C5						M,5			
C6							M,2, 3		
C7									
C8	M,4	M,2,3,4,5	M,1,3,4,5		4				

Table 7-8: Semantic feature vector clustering results

7.1.3.3. Interpretation of Clustering Results

Most of the reference papers were clustered with the corresponding main paper. But, there are only a total of 36 papers that generated hits with the Semantic Signatures. All the remaining 18 papers did not generate hits with any of the existing Semantic Signatures generated in the learning phase. This is because the condition used is very restrictive in counting a hit by the testing document with the Semantic Signature. So the second condition is used on the HAG Tool. The following sections explain the clustering results obtained by applying the second condition on the HAG Tool.

7.1.4. Results Obtained by Imposing Second Condition on HAG Tool

The second condition is explained in Section 4.2.2. It ensures that there is interaction between the first keyword with at least one of the other two keywords in at least one direction. These modifications are only made to the HAG Tool. So the inputs and the outputs for AKT, and ALT are same as the ones shown in Table 7-4.

7.1.4.1. Inputs and Outputs for Testing Tools

The following table briefs the inputs and outputs for the testing tools.

Name of the tool	Input / Output	Description
HAGT	Input	54 papers and 541 semantic signatures
	Output	Hits Array Descriptor File
SSRT	Input	Hits Array Descriptor File
	Output (Refinement)	Hits Array Descriptor File and 109 potential semantic signatures.
	Input	Hits Array Descriptor File, K-Means clustering with cosine distance as distance measure. Document feature vectors are allowed to be grouped into 9 clusters.
	Output	Clusters of documents.

Table 7-9: Inputs and Outputs for Testing Tool

7.1.4.2. Clustering Results

The papers in the testing corpus are placed in nine clusters. Table 7-10 gives the information about papers in each cluster.

	AD	CU	HA	KA	NA	RE	RO	TI	VA
C0							M,1,2,3,5		
C1					M,1,4				
C2	4	M,2,3,5, 4		1					
C3						M,1,3,4			M,4,5
C4					2				
C5					5				
C6								M,1,3	
C7						2,5			2
C8	M,2	1	M,1,3,4,5	M,2,3,4,5					

Table 7-10: Semantic feature vector clustering results

7.1.4.3. Interpretation of Clustering Results

The second condition is comparatively less restrictive. This resulted in increase of the number of documents hitting the Semantic Signatures. Upon examining the clustering results, it is found out that HA's reference paper 2 is completely eliminated as it has hits from only an Semantic Signature with keywords "clumps", "small", "number" and the paper doesn't contain the word "clumps". Similarly, NA's reference 3 is also eliminated as it only has hits for a Semantic Signature generated from the CU paper. KA's reference paper 1 was clustered with the

CU papers. This is because there is interaction between the first keyword and at least one of the other two keywords in at least one direction. Here among the keywords “testing”, “failure”, “subdomain”, there is interaction between the first two keywords. Reference paper 1 of KA and main paper of CU are to some degree related with each other. Both have the concept of using “*Random Testing*” techniques. Reference paper 4 of RO has hits previously from a Semantic Signature with keywords “search”, “random”, “complete”. It is now eliminated because there is no proper interaction between the first and at least one of the other two keywords.

A total of 44 documents appeared in all the clusters. But out of the 109 refined Semantic Signatures, there is only 1 Semantic Signature that represents the learning document (main paper) of AD. So it is less likely to retrieve the documents related to AD properly. Therefore, the number of frequent keywords considered from the first list was increased to 25. This approach increases the number of Semantic Signatures generated in the training phase. As a result there can be an increase in the number of documents generating hits and occupying clusters. The following sections explain the results obtained by increasing the number of keywords considered in first list.

7.1.5. Results obtained by increasing the number of keywords considered for the first list

The following table describes the input given and the output obtained from AKT and ALT by implementing the change in the AKT.

Paper	AKT		ALT	
	<i>Input</i>	<i>Output (KDFs)</i>	<i>Input (KDFs)</i>	<i>Output (SSDs)</i>
AD	AD main paper and corpus	121	121 KDFs and AD main paper	120
CU	CU main paper and corpus	127	127 KDFs and CU main paper	148
HA	HA main paper and corpus	128	128 KDFs and HA main paper	158
KA	KA main paper and corpus	130	130 KDFs and KA main paper	124
NA	NA main paper and corpus	121	121 KDFs and NA main paper	144
RE	RE main paper and corpus	133	133 KDFs and RE main paper	166
RO	RO main paper and corpus	124	124 KDFs and RO main paper	105

TI	TI main paper and corpus	129	129 KDFs and TI main paper	51
VA	VA main paper and corpus	130	130 KDFs and VA main paper	97

Table 7-11: Inputs and Outputs for learning tools

In ALT, the K-Means clustering algorithm is used with cosine distance as distance measure. Generated document vectors are allowed to be grouped into 2 clusters. All generated clusters are saved. 9 SSD folders generated from the learner tool are combined into a single SSD folder. It has a total of 1113 Semantic Signature Descriptor files.

The experiments described in Sections 7.1.3 and 7.1.4 were re-conducted to analyze the effect of increasing the number of words from the first list in AKT. The following sections describe the experimental results.

7.1.5.1. Results Obtained by Imposing First Condition on HAG Tool

The first condition is implemented in HAG Tool and 1113 SSD files and 54 testing documents are given as input to it. The following table shows the inputs and outputs given to the testing tools.

7.1.5.1.1. Inputs and Outputs for Testing Tools

Name of the tool	Input / Output	Description
HAGT	Input	54 papers and 1113 semantic signatures
	Output	Hits Array Descriptor File
SSRT	Input	Hits Array Descriptor File
	Output (Refinement)	Hits Array Descriptor File and 110 potential semantic signatures.
	Input	Hits Array Descriptor File, K-Means clustering with cosine distance as distance measure. Document feature vectors are allowed to be grouped into 9 clusters.
	Output	Clusters of documents.

Table 7-12: Inputs and Outputs for the Testing Tools

7.1.5.1.2. Clustering Results

The papers in the testing corpus are placed in nine clusters. Table 7-13 gives the information about papers in each cluster.

	AD	CU	HA	KA	NA	RE	RO	TI	VA
C0		1	M,1,2,4,5					1	
C1					M,1,4,5				
C2		M,2,3,4,5	3	M,1,2,4,5					
C3						M,3,4			4
C4						1,2			M,1,2,5
C5						5			
C6								M,3,5	
C7							M,3,5		
C8	M,1,2,3,4			3			2		

Table 7-13: Semantic feature vector clustering results

7.1.5.1.3. Interpretation of Clustering Results

A careful examination of the obtained clusters reveals some important factors. Now out of 110 Semantic Signatures, 9 are from AD's learning document. There is also almost equal representation of Semantic Signatures from each of the main documents in the final set of refined Semantic Signatures. Most of the reference papers were clustered with their corresponding main paper. 46 papers generated hits with the refined Semantic Signatures. 8 papers did not generate any hits. The second condition was implemented in the HAG Tool to check if it improves the results.

7.1.5.2. Results Obtained by Imposing Second Condition on HAG Tool

The second condition was implemented in the HAG Tool and 1113 SSD files are given as input to it along with the 54 testing documents. The following table shows the inputs and outputs given to the testing tools.

7.1.5.2.1. Inputs and Outputs for Testing Tools

Name of the tool	Input / Output	Description
HAGT	Input	54 papers and 1113 semantic signatures
	Output	Hits Array Descriptor File
SSRT	Input	Hits Array Descriptor File
	Output (Refinement)	Hits Array Descriptor File and 128 potential semantic signatures.
	Input	Hits Array Descriptor File, K-Means clustering with cosine distance as distance measure. Document feature vectors are allowed to be grouped into 9 clusters.
	Output	Clusters of documents.

Table 7-14: Inputs and Outputs for Testing Tools

7.1.5.2.2. Clustering Results

Table 7-15 gives the information about the papers in each cluster.

	AD	CU	HA	KA	NA	RE	RO	TI	VA
C0			2,4					M,1,3,5	
C1	M,1,2,3,4								
C2		M,1,2,3,4,5	M,1,3,5	M,1,2,4,5					
C3						3,4			M,1,3,4,5
C4					M,1,2,4,5				
C5						5			
C6						M,1,2			2
C7							M,1,2,3,5		
C8				3					

Table 7-15: Semantic feature vector clustering results

7.1.5.2.3. Interpretation of Clustering Results

Cluster0 has HA and TI papers. All these papers deal with “*Software Development*”. Cluster1 contains AD’s main paper and the reference papers. All these papers talk about “*Video Processing*”. Cluster2 has CU, HA, and KA’s papers. All these papers can be classified as papers about “*Software Testing*”, “*Software Reliability*”, and “*Software Development*”. Therefore, these papers can be regarded as papers relating to software. Cluster3 contains reference paper 3 and 4 of RE and the most of the papers of VA. These papers deal with the concept “*Wireless Communications*”, “*Co-Op Diversity*”. Cluster4 contains NA’s papers. Cluster5 contains reference paper 5 of RE. It is about “*Wireless Networks*”. Cluster6 contains main paper of RE along with reference papers 1 and 2. It also contains reference paper 2 of VA. Cluster7 contains RO’s main and reference papers. These papers talk about “*Biometrics*”. They deal with “*Wireless Communications*”. Cluster8 contains KA’s reference paper 3. It is placed in Cluster8 because it is generating many hits with Semantic Signatures containing keyword “software”.

Using this condition in HAG Tool increased the number of documents clustered to 49. NA’s reference paper 2, RO’s reference paper 1, and VA’s reference paper 3 showed up in the clusters. They were grouped with their main papers. There is also almost equal representation of Semantic Signatures from each of the main documents (learning documents) in the final set of refined Semantic Signatures.

7.2. Multi-Category Document Retrieval Experiment

67,654 files belonging to Reuters corpus are given as input to AFET tool along with the category tag GSCI. The AFET tool extracts 170 files belonging to GSCI category. Most of the documents in these 170 files are related to space topic. These 170 files are taken as training corpus. So the training tools are trained to extract documents relating to space. 14,262 documents are taken as testing corpus. 22 of them belong to space category. The HAD file obtained from the testing document is manually analyzed to find the number of space related documents retrieved by these tools. Out of 22 documents, 21 got retrieved. This clearly indicates the capability of automated tools in retrieving documents of a specific category.

7.3. Statistical Method for Estimating Precision of Document Retrieval Experiment

The experiment was conducted for different values of ' a ' (given as input to AKT by the user), and also for different window sizes. The following is the output for various experiments.

7.3.1. Results obtained by setting ' a ' to 5

In this experiment ' a ' value is 5, and window size is 20. The below table describes the results.

n_1	21
n_2	15
n_3	24
n_4	11
n_{12}	9
n_{13}	10
n_{14}	6
n_{23}	12
n_{24}	5
n_{34}	9
n_{123}	9
n_{124}	4
n_{134}	5
n_{234}	5
n_{1234}	4

Table 7-16 Known values obtained by setting ' a ' to 5

Parameter values are as follows.

p ₁	0.7007
p ₂	0.7338
p ₃	0.9633
p ₄	0.4714
q ₁	0.0007
q ₂	0.0002
q ₃	0.0006
q ₄	0.0002
d	0.0012

Table 7-17: Parameters obtained by setting 'a' to 5

7.3.2. Results obtained by setting 'a' to 10

In this experiment 'a' value is 10, and window size is 20. The below table describes the results.

n ₁	19
n ₂	6
n ₃	13
n ₄	20
n ₁₂	5
n ₁₃	7
n ₁₄	9
n ₂₃	5
n ₂₄	3
n ₃₄	6
n ₁₂₃	5
n ₁₂₄	3
n ₁₃₄	5
n ₂₃₄	3
n ₁₂₃₄	3

Table 7-18 Known values obtained by setting 'a' to 10

Parameter values are as follows.

p ₁	0.9518
p ₂	0.5078
p ₃	0.7970
p ₄	0.6850
q ₁	0.0006
q ₂	0.0000
q ₃	0.0003
q ₄	0.0009
d	0.0007

Table 7-19 Parameters obtained by setting 'a' to 10

Results obtained by setting 'a' to 15

In this experiment 'a' value is 10, and window size is 20. The below table describes the results.

n ₁	20
n ₂	15
n ₃	24
n ₄	17
n ₁₂	8
n ₁₃	17
n ₁₄	12
n ₂₃	9
n ₂₄	8
n ₃₄	15
n ₁₂₃	8
n ₁₂₄	6
n ₁₃₄	12
n ₂₃₄	7
n ₁₂₃₄	6

Table 7-20 Known Values obtained by setting 'a' to 15

Parameter values are as follows.

p ₁	0.8220
p ₂	0.4810
p ₃	0.9759
p ₄	0.7448
q ₁	0.0002
q ₂	0.0004
q ₃	0.0003
q ₄	0.0001
d	0.0014

Table 7-21 Parameters obtained by setting 'a' to 5

7.3.3. Results obtained by changing window-size to 40

In this experiment window-size is changed to 40 and 'a' value is set to 5.

n ₁	33
n ₂	40
n ₃	37
n ₄	45
n ₁₂	27
n ₁₃	26
n ₁₄	28
n ₂₃	32
n ₂₄	33
n ₃₄	27
n ₁₂₃	26
n ₁₂₄	27
n ₁₃₄	26
n ₂₃₄	27
n ₁₂₃₄	26

Table 7-22: Known values obtained by changing window-size to 40

Parameter values are as follows.

p ₁	0.8693
p ₂	0.9753
p ₃	0.9107
p ₄	0.9525
q ₁	0.0003
q ₂	0.0005
q ₃	0.0005
q ₄	0.0010
D	0.0023

Table 7-23: Parameters obtained by changing window-size to 40

7.3.4. Results obtained by changing window-size to 50

In this experiment window-size is changed to 40 and 'a' value is set to 5.

n ₁	48
n ₂	39
n ₃	41
n ₄	38
n ₁₂	29
n ₁₃	31
n ₁₄	32
n ₂₃	26
n ₂₄	27
n ₃₄	34
n ₁₂₃	26
n ₁₂₄	26
n ₁₃₄	31
n ₂₃₄	26
n ₁₂₃₄	26

Table 7-24: Known Values obtained by changing window-size to 50

Parameter values are as follows.

p ₁	0.9677
p ₂	0.8159
p ₃	0.9568
p ₄	0.9797
q ₁	0.0010
q ₂	0.0008
q ₃	0.0006
q ₄	0.0003
d	0.0024

Table 7-25 Parameters obtained by changing window-size to 50

From the above results we can conclude that the parameters are insensitive to the 'a' value and sensitive to the window size. As the window-size increases, we can find a significant increase in the number documents retrieved by each of these filters.

8. Conclusions and Future Work

8.1. Conclusions

Enhancements made in the automated tools increased the efficiency of both the training as well as testing tools. It made the learning tools capture more semantic content. It reduced unnecessary combinations of keywords with other keywords in the proximity list. It employed a

proximity measure in AKT to generate genuine keyword groupings. It is ensured that each keyword is within a window distance from every other keyword in the grouping. Thereby gave a scope to go deep into the frequency list for possible keywords. It also reduced the load on ALT by reducing huge generation of Keyword Sets in AKT. The data structures are also modified to suit the requirements.

The experimental results produced from the above changes in the tools clearly indicate that these enhancements are successful in refining the results. The results described for the first experiment clearly indicate the effectiveness of improvements made in the learning tools. The two sub-experiments of Experiment 1 compare the two proximity conditions proposed for the HAG Tool. These experiments clearly indicate that these conditions are successful in eliminating weak dependencies between the documents and the Semantic Signatures. However, they prevent many documents from generating hits with the Semantic Signatures generated from Learner Tool. This implies that the semantic content captured in the Semantic Signatures is insufficient to cluster all the available documents even though they have content similar to the training documents. The second experiment is conducted by increasing the number of keywords considered from first list to 25. Sections 7.1.5.2 and 7.1.5.3 explain the results obtained by imposing proximity conditions on HAG Tool. These are compared and it is concluded that the *less restrictive* condition (7.1.5.3) works well and categorizes documents with similar content into meaningful clusters. The experiment described in Section 7.2 clearly indicates the capability of the enhanced automated tool in retrieving most of the documents relating to a specific category.

8.2. Future Work

The Semantic Signature Refinement Tool (SSRT) makes use of the Singular Value Decomposition (SVD). The Hits Array (HA) generated by HAG Tool is sent as input to the SVD routine. This routine decomposes HA into a diagonal matrix and two complex unitary matrices. The product of the three matrices is equal to HA. The diagonal matrix is as shown in the figure.

172.323985509019	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	115.854518283774	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	75.6197526391207	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	57.6511825437084	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	30.0663092064516	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	28.3196045170126	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	19.3545993920834	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	14.8429326147595	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	13.0508361177963	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	3.28886039161767	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	2.56573390874561	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1.58954478965545	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0.14862342416917	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	6.48394452120724E-15	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.78780656993696E-15	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9.6859768870957E-16	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2.04827998656305E-31	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-1: Diagonal Matrix

The above diagonal matrix is modified by manually setting one of the diagonal elements to 0. The new diagonal matrix after modification is as shown below,

172.323985509019	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	115.854518283774	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	75.6197526391207	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	57.6511825437084	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	28.3196045170126	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	19.3545993920834	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	14.8429326147595	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	13.0508361177963	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	3.28886039161767	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	2.56573390874561	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1.58954478965545	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0.14862342416917	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	6.48394452120724E-15	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.78780656993696E-15	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9.6859768870957E-16	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2.04827998656305E-31	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-2: Modified Diagonal Matrix

Let the modified diagonal matrix be Σ' . Product of the three matrices is computed as shown below.

$$HA' = U_{[m \times n]} \Sigma'_{[n \times n]} V^t_{[n \times n]}$$

Let the generated matrix be HA' . Now, HA is compared with HA' . A dot product is performed between the corresponding columns of the two matrices to find the effect of the modification on the Hits Array. The equation below describes the dot-product for the first column.

$$Dot\ Product[i] = \frac{(HA'[1][1] * HA[1][1] + HA'[2][1] * HA[2][1] + \dots + HA'[m][1] * HA[m][1])}{\sqrt{[(HA'[1][1])^2 + \dots + (HA'[m][1])^2] * \sqrt{[(HA[1][1])^2 + \dots + (HA[m][1])^2]}}}$$

The result obtained is then subtracted from 1. A value of 0 indicates that there is no difference between the columns in the two arrays.

The result of the product is stored in a one dimensional array as shown below.

SSD ₁	0
SSD ₂	0
SSD ₃	0
SSD ₄	0
SSD ₅	0
SSD ₆	0
SSD ₇	0
SSD ₈	0
SSD ₉	0
SSD ₁₀	8.85E-05
SSD ₁₁	8.85E-05
SSD ₁₂	8.85E-05
SSD ₁₃	8.85E-05
SSD ₁₄	8.85E-05
SSD ₁₅	0
SSD ₁₆	0
SSD ₁₇	0
SSD ₁₈	0
SSD ₁₉	0
SSD ₂₀	0
SSD ₂₁	0.006284273
SSD ₂₂	0.006284273
SSD ₂₃	0.006284273
SSD ₂₄	0.006284273
SSD ₂₅	0.006284273
SSD ₂₆	0.006284273
SSD ₂₇	0.006284273
SSD ₂₈	0.006284273
SSD ₂₉	0.006284273
SSD ₃₀	0
SSD ₃₁	0
SSD ₃₂	0

SSD ₃₃	0.020063874
SSD ₃₄	0.000658863
SSD ₃₅	0.000547591
SSD ₃₆	0.002756045
SSD ₃₇	0.002365074
SSD ₃₈	0.007481177
SSD ₃₉	0.000349471
SSD ₄₀	0
SSD ₄₁	0
SSD ₄₂	0
SSD ₄₃	0
SSD ₄₄	0
SSD ₄₅	0
SSD ₄₆	0
SSD ₄₇	8.85E-05
SSD ₄₈	0.001616697
SSD ₄₉	0.001168967
SSD ₅₀	5.16E-07
SSD ₅₁	0.000220894
SSD ₅₂	8.85E-05
SSD ₅₃	0.001014384
SSD ₅₄	0.000537457
SSD ₅₅	0.006284273
SSD ₅₆	0.006284273
SSD ₅₇	0.006284273
SSD ₅₈	0.006284273
SSD ₅₉	0.006284273
SSD ₆₀	8.85E-05
SSD ₆₁	8.85E-05
SSD ₆₂	0.000167968
SSD ₆₃	8.85E-05
SSD ₆₄	0.000135043
SSD ₆₅	8.85E-05
SSD ₆₆	8.85E-05
SSD ₆₇	9.91E-06
SSD ₆₈	8.85E-05
SSD ₆₉	0.002365074
SSD ₇₀	0.03337206

SSD ₇₁	0.004462393
SSD ₇₂	0.033450968
SSD ₇₃	0.006284273
SSD ₇₄	0.006284273
SSD ₇₅	0.000936562
SSD ₇₆	0.006284273
SSD ₇₇	1.74E-05
SSD ₇₈	0.033450968
SSD ₇₉	0.006284273
SSD ₈₀	0.020063874
SSD ₈₁	0.060610704
SSD ₈₂	0.006284273
SSD ₈₃	0.059440983
SSD ₈₄	0.008012786
SSD ₈₅	0.001895091
SSD ₈₆	0.046513978
SSD ₈₇	0.003478896
SSD ₈₈	0.000146714
SSD ₈₉	0.000206162
SSD ₉₀	0.000122728
SSD ₉₁	0.000353611
SSD ₉₂	2590099
SSD ₉₃	1.40E-05
SSD ₉₄	2.09E-06
SSD ₉₅	8.85E-05
SSD ₉₆	2.09E-06
SSD ₉₇	0.001417167
SSD ₉₈	8.85E-05
SSD ₉₉	8.85E-05
SSD ₁₀₀	8.85E-05
SSD ₁₀₁	8.85E-05
SSD ₁₀₂	8.85E-05
SSD ₁₀₃	0.000246983

Table 8-1 Difference in Hits Array

Careful examination of the above table reveals a few interesting facts about the way in which the modification in the diagonal matrix affects the Semantic Signatures. Some differences are significant. These differences can further be analyzed to find a pattern in which the Semantic Signatures change. Using these patterns, a decision can be made on calculating number of

clusters and for selecting a fixed number of semantic signatures from each cluster in SSRT (while refining semantic signatures). This helps to perform better refinement of Semantic Signatures s and can produce a better result.

Future work also includes but is not limited to:

- Tools in Automated SSMinT are capable of processing unstructured data in text, XML, or HTML formats. This functionality can be extended by enabling them to process PDFs, Dynamic web pages, etc.
- K-Means clustering is being used in the tools. It is an unstable algorithm. So implementation of other clustering algorithms can also be embedded in the tools.
- As of now all the generated clusters are saved as semantic signatures. Cluster selection criteria can be developed in the ALT instead.
- A demonstration of the Automated SSMinT language independence.

Bibliography

- [1] [Online]. Available: http://en.wikipedia.org/wiki/Text_mining.
- [2] M. Hearst, "What is Text Mining," 2003. Available: <http://people.ischool.berkeley.edu/~hearst/text-mining.html>
- [3] Q. Wu, E. Fuller, and C.Q. Zhang. Text document classification and pattern recognition. In Proceedings of the International Conference on Advances in Social Network Analysis and Mining (ASONAM'09), pp. 405 - 410. IEEE 2009
- [4] Q. Wu, E. Fuller, and C.Q. Zhang. Graph model for pattern recognition in text. In Mining and Analyzing Social Networks, pp. 1 - 20. Springer, 2010
- [5] R. R. Kota, Automated Discovery of Relevant Features for Text Mining. MS Thesis, West Virginia University, 2011.
- [6] U. K. Para, Computer-aided semantic signature identification and document classification via semantic signatures. MS Thesis, West Virginia University, 2010.
- [7] S. R. Peddada, Sensitivity of semantic signatures in text mining. MS Thesis, West Virginia University, 2010.