Phrase Based Pattern Matching Framework
for Topic Discovery and Clustering

by

Ramanpreet Singh

Bachelor of Technology, GGSIP University, 2010

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

Master of Computer Science

In the Graduate Academic Unit of Faculty of Computer Science

Supervisor(s): Ali Ghorbani, Ph.D., Faculty of Computer Science
Examining Board: Michael Fleming, Ph.D., Faculty of Computer Science, Chair
Huajie Zhang, Ph.D., Faculty of Computer Science,
Donglei Du, PhD, Faculty of Business Administration

This thesis is accepted by the

Dean of Graduate Studies

THE UNIVERSITY OF NEW BRUNSWICK

December, 2013

©Ramanpreet Singh, 2014
Abstract

In text mining, one of the major challenges is to discover understandable topics of discussion, and at the same time statistically valid underlying document grouping. The word order and word co-occurrence information are very crucial in understanding the meaning of a document. Vector space and bag of word models are poor candidates for topic discovery focused clustering algorithms. Phrase based models have proven to be promising in extracting meaningful topics in a given set of documents. In this thesis, a new framework has been proposed which simultaneously performs topic discovery and clustering in linear time. The core of this framework is the new document model and algorithm to perform efficient pattern matching for exact, prefix, postfix, and infix matching of phrases in linear time. The document model uses concepts from graph theory and the theory of automata to efficiently and intelligently match, index, track, and analyze interesting patterns.

The generic nature of the framework enables to perform various text mining applications such as query enhancement, keyword extraction, and indexing, to name a few. The primary focus has been on discovering meaningful topics in
a set of documents and building a story or context around them. The model is also capable of tracking already discovered topics. The proposed model is efficient enough to be able to capture the essence of the present data and make a link between past and future data. To capture the natural language in the text, instead of just matching words or terms; phrases, entities, and word sense enrichment techniques are also used. With this, we were able to get the essence of the topic discussed in a document even if it did not have an exact string match. The idea of story building is new in this work. The concept of “Knowledge Graph” and “more than just keyword” search are also introduced.

In various conducted experiments, the scalability, space, and time performance are compared with the benchmark phrase based document models and the industrial standards. The F-Measure, entropy, and human evaluation are used to validate the topics and stories obtained. The results are promising and highly encouraging.
Dedication

To the culture and soil of Punjab, the land of five rivers.
I would like to acknowledge my sincere thanks to my supervisor Dr Ali A. Ghorbani for his constant feedback and guidance. His input and direction helped me to pursue this research.

I would also like to extend my thanks to Damien Dubois; without him I would never have had the courage and motivation to carry out my work.

Finally, I would like to thank iTacit Inc. employees for giving me exposure to the business side of this research and applying it in their real time product, where a bridge between theory and practice has been realized.
# Table of Contents

Abstract ........................................... ii

Dedication ........................................... iv

Acknowledgments .................................... v

Table of Contents .................................. ix

List of Tables .................................. x

List of Figures .................................. xii

Symbols, Nomenclature or Abbreviation .......... xiv

1 Introduction ...................................... 1

   1.1 Summary of Contribution ....................... 2

   1.2 Thesis Organization ............................. 3

2 Literature Review ................................ 5

   2.1 Introduction .................................. 5

   2.2 Feature Extraction and Selection ............... 8
# 3 Proposed Framework

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Overview</td>
<td>38</td>
</tr>
<tr>
<td>3.2</td>
<td>Framework Overview</td>
<td>40</td>
</tr>
<tr>
<td>3.3</td>
<td>Feature Extraction Phase</td>
<td>43</td>
</tr>
<tr>
<td>3.3.1</td>
<td>RAKE Algorithm</td>
<td>45</td>
</tr>
</tbody>
</table>
3.3.2 Entity Extraction ........................................ 47
3.3.3 Document Mapping ................................. 48
3.4 Linear Pattern Matching and Parsing Unit ................. 48
  3.4.1 Proposed Data model .............................. 52
    3.4.1.1 Theory of Automata ......................... 52
    3.4.1.2 Structure Overview ....................... 53
  3.4.2 PMM Construction ................................. 56
  3.4.3 An Example .................................... 62
  3.4.4 Underlying Graph Indexing Model ................. 69
3.5 Topic and Story Mining ............................... 71
  3.5.1 Story/Context Mining ............................ 75
  3.5.2 Topic Profile ................................ 78
3.6 Beyond Keywords: Sense Search ........................ 79
3.7 Other Use Cases ....................................... 80
  3.7.1 Query Expansion ................................ 81
  3.7.2 KeyPhrase Extraction ............................ 82
  3.7.3 Vocabulary Tracker .............................. 83
  3.7.4 Summarization ................................ 83
  3.7.5 Entity-Entity Interaction ...................... 83
3.8 Concluding Remarks ................................ 84

4 Experimental Results .................................. 86
  4.1 Introduction ...................................... 86
List of Tables

2.1 Complexity of String Matching algorithms .................. 34
4.1 UofWData: Comparison With Benchmarks .................... 100
4.2 Topics Discovered in UofWData .............................. 104
4.3 Query Senses Discovered in UofAData ....................... 108
4.4 UofAData: Comparison With Benchmarks .................... 109

List of Figures

2.1 Suffix Tree Document Model ................................. 25
2.2 Document Index Graph ...................................... 26
2.3 Inside DIG Node ........................................... 27
2.4 String matching algorithms .................................. 31
3.1 Framework Overview: Black Box nature ...................... 41
3.2 Feature Extraction Process ................................. 49
4.10 RCV1-Subset: Clustering Time Trend . . . . . . . . . . . . . . 111
4.11 What Infomous does? . . . . . . . . . . . . . . . . . . . . . . . 112
4.12 Cloud Generated by Infomous . . . . . . . . . . . . . . . . . . 113
4.13 News Topics Generated by our framework . . . . . . . . . . . 114
4.14 Vocabulary cloud for Hotel Reviews . . . . . . . . . . . . . . 116
List of Algorithms

1. RAKE Algorithm ............................................. 46
2. Construction of the Graph .................................. 57
3. Entering phrase in the graph ................................. 58
4. PMM: Calculation of Failure Function ...................... 61
List of Symbols, Nomenclature

or Abbreviation

VSM  Vector Space Model
BoW  Bag of Words
TDT  Topic Detection and Tracking
DIG  Document Index Graph
STC  Suffix Tree Clustering
LPMP Linear Pattern Matching and Parsing
HWAT Human Way of Analyzing Text
DARPA U.S. Government’s Defence Advanced Research Projects Agency
TDT  Topic Detection and Tracking
ReAD Read, Activate and Decay
Σ*  A set of finite length strings
Σ  A finite alphabet
ε  An empty string
W|X String W is a suffix of String X
W[X String W is a prefix of String X
O  Big O Notation
Θ  Average case time complexity
M  A finite automata
Q  Set of finite states
q₀  Start State, where q₀ ∈ Q
A  Accepting state
δ  transition function of M
Chapter 1

Introduction

In an incoming stream of unstructured text data, it is always desired to manage documents in some natural grouping based on their topic of discussion. A document may be grouped under several topics. By topics we mean different perspectives a document covers. For instance, a document discussing soccer may discuss soccer game rules, or its history. Thus, it becomes very important to assign different topic tags to a document, which could be used for easy browsing and further analysis.

Topic discovery is a task of discovering and tracking events or interesting patterns in a text stream. It is an event based information management and organization. As such there are no standalone algorithms defined to discover topics. As per the requirements, a combination of various text mining modules, working together, makes a framework specifically suited to extract
the topics. In the literature, most of the topic discovery models, more or less, use clustering algorithms as the backbone. The reason is quite obvious: to generalize a topic on a set of documents, the documents must be similar in some aspects. There are also other important parts and pieces, which are central to any topic mining framework, such as feature extraction, indexing, and similarity modules, to name but a few.

In this research, we also propose a generic framework, which not only performs topic discovery, but is also able to perform other text mining tasks, such as keyword extraction, clustering, query enhancement, phrase level indexing, meaning search, vocabulary tracking, and summarization.

1.1 Summary of Contribution

The main purpose of this research is to propose a generic framework to perform various text mining tasks all in one place, which can deal with large data efficiently in linear time. The contribution of this research can be summarized as follows:

- The core of this framework is the new document model, which uses graph theory and the theory of automata to efficiently store and analyze the salient features of the data.
• A generic text mining framework, which can perform various applications with little adaptation.

• A linear algorithm to read, store and discover the topics and group documents with similar perspectives.

• A human centric approach to topic discovery in which the labels are understandable by humans.

• The idea of building stories and context around flat topics.

• The concept of incrementally building the Knowledge Graph and the concept of “more than just keyword” search are also introduced in this work.

1.2 Thesis Organization

The rest of the thesis is organized as follows:
In Chapter 2, a thorough survey of general text mining, feature extraction, clustering, co-clustering, topic discovery, graph-based document models, and pattern matching algorithms is conducted. The state of the art in topic discovery research is also discussed.
Chapter 3, in the beginning, briefly discusses the motivation for the proposed framework. An overview of the complete framework is described. Benchmark document models are also discussed. The core of the framework is the document model, which uses concepts from graph theory and the theory of
automata. The algorithm design and analysis of constructing a document model from text documents is described. A real life example is used to explain the capabilities of the proposed document model. Topic and story discovery are discussed in details. Other use-cases of the framework are also described. The concept of Knowledge Search is also described in this chapter.

Chapter 4 describes the experiments conducted in order to justify the proposed framework’s design. The document model is fed with real data and the statistics are compared with the benchmark document models. The results of topic discovery and clustering are also compared with the published benchmark algorithms using standard evaluation measures. Various experiments are conducted to show the potential of using the framework in different applications. The idea of “more than just keyword” search is also discussed with experiments.

Finally, the conclusion and possible future directions for research are given in Chapter 5.
Chapter 2

Literature Review

2.1 Introduction

Both data classification and clustering have existed in the literature for more than five decades [24]. Due to the changing nature of data, algorithms also need to be adapted accordingly. In the past, the data size was under manageable limits of human experts and the algorithms were not very efficient. Over time, it was realized that there is a lot of potential knowledge in the data that needs to be mined. The motivation led to an establishment of an independent research area for machine learning and data mining. Machine learning gave new insights into how to look at data. Machines now could understand and discover knowledge out of data.

Due to the advancement in Internet technology and its ease of use, the amount
of data in the form of text has started growing. The techniques designed to work on numerical data have now been modified to work on text data. Classification and clustering turned out to be very useful concepts for text data. Before the boom in the amount of useful text data, it was considered very ambitious for a machine to predict category and natural grouping of text documents. Jain et al. [24] describes various algorithms over the last 5 decades to find implicit groupings in data. Almost all of them can be used for text mining, as a text document can be easily represented in a vector space model (VSM) [33].

Over the past decade, the use of text over the Internet increased exponentially; not only has the amount of data increased but its nature has also changed. In the past, long static text documents were dominant; then came the time for e-mails. With e-mails, people started connecting to each other. Chat systems enhanced the power of being connected to many people at one place at the same time. The time information attached with text documents started providing dynamic temporal data. As a result, data moved away from being static; it started evolving and changing, news feeds being an example. Then with the boom of social media, the way text data is seen has completely changed. Data is not isolated anymore; it is all connected. A piece of text is connected to an entity or another piece of text through some related topics and context.
Therefore, in today’s world, a piece of text carries with it a lot of information, such as time, entities, topics and concepts, and context. With all this information there is a lot of hidden knowledge inside the text world. Some of the popular tasks of discovering knowledge from text documents are:

- Categorizing a document to a predefined set of categories.
- Grouping similar documents.
- Understanding the implicit emerging topic of discussion in a group of related documents.
- Indexing text documents in order to retrieve documents based on a query.
- Summarizing a piece of text.
- Finding alerting patterns in streaming text such as in newsfeeds.
- Finding sentiment, opinion and quality of a document.
- Organizing web documents for easy browsing and many other similar tasks.

More or less, all the tasks listed above consist of basic text mining functions such as text cleaning, feature extraction, parsing, indexing, and search.

In the following sections, a brief literature survey and state of the art in topic discovery and related areas is discussed, in order to set the stage for
the proposed framework, which is described in Chapter 3.

2.2 Feature Extraction and Selection

Feature selection is a fundamental task in text mining. The final results of classification or clustering largely depend on how well the features were selected and weighted [51]. As described in [51], when the data are labeled, the task of feature extraction is more statistical in nature. The subset of features selected are guided and evaluated with the help of labels and an optimal subset of features is selected in order to get the best output. But, in the case of unsupervised learning when the labels are not known, the feature selection task is more complicated due to the lack of a guiding factor. The task is then reduced to extracting linguistic features, keywords, and phrases from a document. The selected features are validated using their usage in other documents of the corpus.

One of the popular feature extraction measures is Term Frequency Inverse Document Frequency (TFIDF). Term frequency (TF) helps in identifying the importance of a term in a local context and document frequency (DF) normalizes the importance in global context. Over the past many years, TFIDF has proven to be useful [33]. However, due to changes in types of text data and the increasing amount of data, there is a need for other quick and
efficient feature selection alternatives.

2.2.1 Single vs Multi Document Feature Extraction

When the number of text documents is very large, parsing all the documents for feature selection becomes a slow and time consuming task. In order to speed up the process, single document feature selection is seen as a compromising technique at the expense of minimal reduction in final accuracy, as compared to what is achieved with multiple document feature extraction techniques.

A novel approach proposed in [35] finds potential features from a document without looking into the global corpus. They used the idea of co-occurrence distribution of words in a document. Another proposed model in [38] extracts features from a single document by exploiting the word distribution in the document. They derived a graph from a document, in which vertices are words and two words are linked to each other through dissimilarity measures [38].

Single document feature extraction is also very useful when the global data boundaries are not fixed and are dynamic in nature, especially in streaming text data, where it is hard to update the global information each time a new document comes into the stream. Yan et al. [50] proposes an effective feature selection process for large scale streaming data. Their proposed model is
based on using orthogonal centroid information when a new document stream is added to an existing set of documents [50]. Regardless of single or multi-document feature extraction, “what is a feature?” is an important question in the text mining community.

2.2.2 Possible features

In text, features could be single keywords, a combination of keywords making up a phrase, n-grams, entities, part-of-speech tags and other linguistic and lexical features [32, 35, 38, 23]. A phrase is an ordered sequence of keywords making up a new meaning different from the constituting keywords. Use of phrases and other lexical features in text mining have shown improvement in precision without deviating much from recall [53]. Phrases are also less sensitive to noise [20]. Other interesting features which could be extracted are Named Entities and Parts of Speech. Benchmark Named Entity Recognizer (NER) and Part Of Speech (POS) Tagger are provided by The Stanford Natural Language Processing Group [17, 44]. The current version of NER comes with the ability to tag seven types of entities in the text, i.e. Time, Location, Organization, Person, Money, Percentage, and Date, whereas a POS tagger can tag all parts of speech in the English language. This information proves to be very useful in enhancing the summary of document(s) by quickly identifying the entities involved.

Hence, the feature selection step involves answering two important questions: what type of features?, and from single document or multiple documents?
An interesting algorithm to quickly extract important phrases is Rapid Automatic Keyword Extraction (RAKE) [42]. It is an unsupervised, domain and language independent algorithm used to extract meaningful and logical phrases from a single document. The way the algorithm works is very similar to how humans extract keywords from any given document. Humans do not depend on the whole corpus to extract the important keywords. It is based on a simple observation that most of the manually selected keywords rarely contain stop words, punctuation signs or other words carrying less lexical meaning. The algorithm uses these patterns as delimiters and generates the ranked list of weighted n-gram phrases, where $n$ could be of variable length unlike the fixed length n-gram feature extraction process [42]. The results of RAKE are very encouraging and are almost comparable to benchmark feature extraction algorithms [42], but with significant improvements in the speed.

One of the important characteristics of RAKE is that it can be trained to work on any language or domain by just giving the list of stop words and phrase delimiters for that domain or language. Hence, this makes RAKE a very generic algorithm which can be altered and plugged into any text mining framework.
2.3 Clustering

Clustering has been studied widely for almost five decades [24]. There is a large number of research papers/algorithms published on this subject [4, 33, 24]. However, still there is no perfect algorithm for clustering that will perform optimally in every domain/application. In this section, a quick overview of text clustering is given; for more details on the taxonomy of various clustering algorithms, refer to [4, 24, 33, 25].

In text mining, clustering is the process of discovering the natural grouping of documents that are similar in some aspect/facet. As such, there is no universally agreed definition of what a perfect grouping is. The grouping is performed in such a way that for a given aspect, the documents that are in the same group are more similar to each other than to those in the other groups. Good clustering exhibits high intra-cluster similarity and low inter-cluster similarity [33]. The following are the basic components, present in any clustering approach, that decide its overall performance.

- Document representation and feature definition: A document is mapped to the desired feature space using either single word features, n-gram features, phrases, or some other quantitative and categorical features. A document could be mapped to a vector space model (VSM), to a phrase based model or to a graph based data model [4]. There might be an additional step to reduce the feature space.
• Defining the pattern proximity measure in the selected feature space, for example, using cosine similarity in the vector space model.

• Clustering algorithm: Using the proximity measure for grouping documents. The algorithm also includes the stopping criteria. It should also converge to optimize a defined objective function [18, 41].

• Grouping evaluation and producing the final output.

• Labeling the clusters (if needed).

Various combinations of the above defined components give different classes of algorithms. Major clustering algorithm classes are: partitive algorithms, divisive and agglomerative hierarchical algorithms, density based algorithms, spectral clustering, phrase based clustering, high dimensional and scalable algorithms [4, 24, 33, 25, 26].

All these algorithms address the following challenges in different ways to optimally solve the clustering problem for a given type of data and domain.

• Best fit Clustering: Best split of a dataset in an optimal number of clusters.

• Cluster Validation: Defining “what is a good cluster?”.

• Curse of Dimensionality: The dimensionality of a feature space grows exponentially with the increase in the size of the corresponding dataset. In order to speed up the algorithm it is always desired to reduce the feature space and select the best feature subset.
• Nature of Data: With the varying nature of data, clustering algorithms need to be tuned. For large or streaming data, it will be expensive to re-cluster. An incremental approach is recommended for such cases.

• Complexity: Most of the clustering algorithms are computationally expensive. Due to the changing nature and size of data, it is always desired to design optimal algorithms with realistic run-time bounds.

• Stability: A good clustering algorithm should be stable to converge to an optimal objective state.

• Subjectivity Issue: In text clustering, defining “what is a cluster?” is entirely subjective [30]. A document may be grouped into a class of “sports” or into a class of “activities”. Even humans cannot agree on one grouping of documents; different aspects give rise to different ways to group the documents.

• Cluster Labeling: This is one of the major issues in text document clustering. Again, it is a subjective issue to label a cluster. Research has shown that coming up with one label for a group of documents is difficult for humans too; how we can expect computers to give us one [30]? Hence, instead of just one label, work has been done to present a connected list of labels to summarize the cluster [4, 8, 30, 34].

It is important to understand the difference between purely statistical clustering (or grouping) vs. meaningful topic discovery. A perfect grouping of
documents may have too poor of a label to understand them [4]. It has been
an area of research to find a compromising solution to group and label the
clusters together. Simultaneous grouping and labeling mutually guide each
other to come to a statistically correct grouping, yet still having meaningful
labels. This idea is referred to as Co-Clustering in the literature [16, 4].

2.4 Co-Clustering features and Documents

In a traditional vector space model, the term-document matrix could be
very sparse. The problem of clustering rows is that of clustering documents,
whereas that of clustering columns is that of clustering words [4]. Clustering
words and documents is a dual problem [16]. This dual property is very
important in topic discovery as the words/features describing the same topic
are often used together [30]. A linked co-occurrence graph between various
features could give an understandable topic description with a statistically
valid grouping to back the labeling.

One of the main focuses of this research is also towards this approach. The
idea is to develop a linear co-clustering algorithm to determine potential topic
and cluster documents around those potential topics. The cluster should
help in determining potential features making up the topic and the potential
feature graph should help in grouping documents from various aspects.

In the next section, particular focus has been given to the clustering algorithms
that are best suited for grouping text documents in order to discover the
2.5 Topic Discovery

In simple words, topic discovery is a process of identifying various topics or subject matters discussed in a corpus of documents. When data is dynamic in nature (for example, news stream), the idea of topic discovery consists of discovering a new topic, tracking the existing topic for any alerting pattern and connection between various topics to give a summarized overview in a given time window. The framework which determines the feature cluster and document cluster simultaneously is referred to as topic modeling [22]. In the literature, there is no universally agreed definition of topic discovery algorithm; it varies from application to application [7].

The survey of work done in the area of topic discovery is discussed in the next section, followed by the conclusion of what is the “state of art” in topic discovery research.

2.5.1 The Pilot Study

In 1997, a pilot study on topic detection and tracking (TDT) [8] laid down the foundation for research in the topic discovery domain. It was majorly supported by the U.S. Government’s Defence Advanced Research Projects Agency (DARPA). The focus was to gather researchers to set a dedicated track for research in news-specific and event based organization of information in
in-flowing stream of text. Five major tasks [7] were defined by the researchers in the initial phase of the program: 1. Segmentation, 2. First story detection, 3. Cluster Detection, 4. Tracking and 5. Story link detection. As the program proceeded, tasks 2, 3 and 4 gained momentum and came out as significant areas to work on for the research community.

In the pilot study, major participants were DARPA, Carnegie Mellon University (CMU), University of Massachusetts (UMass) Amherst and Dragon System [8]. The data was taken from the Reuters and CNN news stream and the target was to detect the appearance of new topics in the stream and to track their re-appearance and evolution. For new event detection, the CMU approach used incremental clustering, which used a conventional vector space model [7]. They clustered the stories in a bottom up fashion in a given time window. UMass tested two approaches: The first approach was based on query generation from relevance feedback algorithms. The idea was to identify the triggering nouns and phrases in the corpus and to generate queries out of it [7]. These queries were used to match other stories discussing the same event. The second approach was somewhat similar to the CMU approach in the sense that they used bottom up agglomerative clustering. The only difference was, they used the query as the proximity measure in the clustering. The UMass approach can be summarized as: instead of a vector space model, they used a query representation for a document, which further comprised of important triggering features and phrases. These features are then used to group different stories. This approach is somewhat similar to
the frequent term set clustering approach [10]. The results of this type of approach are more understandable than the traditional Bag of Word (BOW) based Vector Space Model (VSM).

2.5.2 Previous work in Topic Discovery

Following the TDT program, which ended in the year 2001, many researchers started to work on various aspects of topic discovery. A very obvious intuition that came out of the research was that clustering is the central backbone of the topic discovery model. This intuition was formalized in later years, as a lot of researchers published different variations of clustering algorithms specifically suited for topic discovery [16, 10, 21, 34].

From the year 2001 to 2006, clustering was performed primarily with a vector space model. The focus was to modify the existing clustering algorithms for the topic discovery domain. But the success in understanding clusters was not very significant with VSM; the general sense and the meaning of the labels were very difficult to understand.

The notion of finding alternatives for VSM was further strengthened with the feasibility demonstration published in 1998 by Zamir et al. [52]. They published an alternative to VSM by introducing a suffix tree document model. This new model not only mapped the suffixes of web snippets into a tree structure, but also grouped the snippets at every depth, if there happens to be a match with suffixes of the other snippets. Based on the shared suffixes they identified base clusters, which were then combined to get final clustering. The
idea of “what is a phrase?” was still not very well defined in this model. The algorithm split the snippet for all possible suffixes and parsed them through the tree. This way, there were many redundancies, making it unsuitable for larger documents.

Beil et al. [10] in 2002 gave new direction to the clustering community in the form of an official publication. They proposed an algorithm for clustering that made use of frequent term sets, a concept used in association rule mining. The idea was not to cluster the high dimensional documents, but rather to consider low dimensional frequent term sets as cluster candidates and group documents around them. The results were significant, and easily understandable. The clustering obtained from their algorithm was well arranged into the topic hierarchy. The basic idea of their approach was that frequent term sets are good candidates for potential cluster centers and topics for that cluster. The merging and splitting of related frequent term sets gave rise to representative cluster descriptions or topics.

Inspired by the use of frequent term sets as features instead of single independent words and the suffix tree document model [52], Kamel et al. in 2004 [21] proposed a phrase based directed graph indexing model. Their model efficiently utilized the sub-graph overlapping property to determine the similarity between documents. In 2005, using the graph model, they designed an algorithm to extract key phrases called the core phrase extraction algorithm [20]. The clear conclusion that came out was that the underlying document graph model is capable not only of performing pattern similarity
checking but also other important text mining tasks.

In 2007, Jiantao et al. [39] proposed a topic oriented semantic orientation algorithm. They made use of ontologies to generate topic semantic annotation for each document. The use of ontologies enriched the document representation in the space where finding concept related documents was easier than simple word matching.

Yanjun Li et al. in 2007 [31] proposed a clustering algorithm based on the frequency of the word meaning sequence instead of the simple bag of word approach. A word meaning sequence is the concept expressed by synonyms and associated terms. This representation is useful in dealing with an understandable representation of a document, dimensionality reduction, labeling and overlapping concepts.

Wang et al. [46] and Yuen-Hsien Tseng et al. [45] also proposed an approach to enrich word sense using wordnet [37] and using the structure of the document, if available, to mine the appropriate topic. Chen et al. proposed a framework to detect topics using Natural Language Processing (NLP) and Information Retrieval (IR) [13].

Another model was proposed by Waten et al. in 2008 [47]. They discovered topics by clustering keywords. The central idea was:

“A good clustering of the words is the one that results in a good clustering of the documents and a good clustering of documents is the one that results in a good clustering of the words”.

An algorithm proposed by Fang Li et al. in 2009 [29] proposed a new way
to perform topic discovery. They used Non-negative Matrix Factorization (NMF) to deal with the high dimensionality and testor theory to find a topic for each cluster.

Massey et al. in 2011 [34] published a very simple and linear time algorithm to discover topics. Unlike long computationally expensive approaches, they proposed an approach that determines a topic as a human would do. They proposed the ReAD principle, where ReAD stands for Read, Activation, and Decay. The human brain reads the words, and particular areas of the brain get activated. As we read, some activation areas decay and some get more activated. Over time, the concept associated with the most activated regions in the brain contain the topic of the read document.

Zaiane et al. in 2013 [30] published a topic oriented graph clustering algorithm which borrowed ideas from the concept of community mining in social network analysis. Rather than grouping the documents first, they extracted phrases from each document to build a co-occurrence graph and then found communities to form document clusters. Their approach could actually describe a cluster from various aspects rather than just a one word cluster label, unlike the traditional word based vector models.

2.5.3 State of the Art

From the pilot study (1998) till now, the task of topic discovery evolved as “new event detection and tracking” using clustering techniques as the backbone.
Co-Clustering features and documents turned out to be a promising approach for topic discovery. Instead of simply using a vector space model and a simple bag of word approach, the focus moved towards the use of other features, more than just words, such as phrases, frequent term sets, enriched words and parts of speech. These types of features really make sense in topic discovery as one of the main tasks is to come up with understandable topic labels. Various proximity measures in new feature spaces were published in order to suit the labeling algorithm.

The notion of having one label for a cluster also moved towards having a connected graph of labels to describe a given cluster. However, the evaluation of topics is still somewhat subjective in nature because most of the labeled datasets expect the results to be compared for the partition of documents space, whereas the focus of topic discovery is to discover meaningful descriptions by compromising on the best partition. The algorithm also needs to be incremental in nature to cope with the stream of text. More work is done towards making the labeling process linear and simple instead of computationally expensive algorithms which focus more on partitioning rather than understanding the clusters.

There are two benchmark models, Suffix Tree Clustering (STC) [52] and Phrase based Document Index Graph (DIG) [21], which satisfy most of the characteristics of the state of the art in topic discovery. STC and DIG are near-linear time phrase based models with an underlying graph structure as the backbone to quickly index, retrieve and use it, for similarity analysis to
discover and group the documents and label them at the same time.

The framework in this research also proposes an automaton based pattern matching document model designed to perform various text mining tasks, and specifically topic mining in simple and linear run time.

The next section briefly describes the underlying document model of the two benchmark algorithms in order to understand the difference between them and the model proposed in this framework.

2.6 Phrase based Document Models

Vector space and bag of word models are not good candidates for topic discovery focused clustering algorithms [20, 30]. The word order and co-occurrence is not maintained and, hence, the meaning is lost and the task is reduced simply to word match and frequency count. Consider two sentences: “Jack is sitting on a chair” and “A Chair is sitting on jack”. For the bag of word and VSM approach, these two sentences are perfectly similar by cosine similarity. However, they have completely different meanings. Capturing the word co-occurrence and word order is very important. A very promising document model, other than VSM, to capture this information is graph structure. STC and DIG are based on graph structure to use this information and perform clustering that is more meaningful and informative.
2.6.1 Suffix tree Clustering (STC)

STC was originally proposed to cluster short web snippets [52]. A snippet is split into all possible suffixes and every <suffix-snippetID> pair is parsed through the suffix tree. The underlying document model is a “trie” (a compact tree).

Each node of the tree represents a group of snippets and the suffix phrase common to the set of snippets. A Node is also called the base cluster with the same label as the suffix phrase, common to the underlying snippet set. The length of the set and suffix phrase is used to give a score to the base cluster. Base clusters are then combined to get the final clustering based on a connected component graph algorithm [52]. The algorithm is reported to have an $O(n\log(n))$ run-time.

The underlying tree structure (taken from [52]) for the sentences: 1: “cat ate cheese”, 2: “mouse ate cheese too” and 3: “cat ate mouse too” is shown in Figure 2.1.

It can be seen that a suffix tree is a rooted directed tree with edges containing the string phrase information and nodes containing the set of document IDs. Furthermore, the tree only considered the suffix matches and has a lot of repetition for the same keyword. There is a lot of redundancy in storing the string suffixes, causing the number of nodes to become very high, making it unsuitable for large documents.
2.6.2 Document Index Graph (DIG)

Hammouda et al. [21] proposed a novel document representation model that captures the word order information in phrases and builds an inverted list of phrases to documents. The DIG model is based on graph theory and uses the concept of sub-graph overlapping property to find similar phrases and computes similarities between the documents.

A document is represented in the form of a set of phrases. Every phrase is parsed in a directed multi-graph [19]. Each word becomes a node and two nodes are connected if the two nodes occur in conjunction with each other, as shown in Figure 2.2. Each node contains the document’s information and also contains a list of edges going through it in order to capture the sentence information [20, 21], as shown in Figure 2.3.

DIG, as compared to STC, does not generate a new node for a given word,
thereby avoiding redundancy. This controls the number of nodes in the graph. The problem of document similarity now becomes the problem of finding the percentage overlap of shared sub-graphs.

Both the document models map the document into a graph space. The construction of the graph is different for both. While feeding and growing the graph, pattern matching is performed in a naive way. There could be more intelligent ways to grow the graph so that it does not make unnecessary moves and search for information while growing the graph. Concepts such as Failure Transitions and Next State [43] can enhance the building of the graph.

In the next section we will discuss the literature on pattern matching algorithms and the state of the art. The motivation is to propose a graph data model with built in capabilities of efficient pattern matching, unlike STC and DIG.
2.7 Pattern Matching

2.7.1 Introduction

String pattern matching has always been central to text mining tasks [5, 27, 9]. There are lots of examples such as bibliography search, find and replace option, matching the keywords and finding similar pieces of text [9], where it is required to search and match a keyword or a set of keywords. In all cases, it is preferable to have an efficient, fast algorithm to quickly locate the occurrences of a pattern in a given piece of text.

Figure 2.3: Inside a DIG Node [21]
2.7.2 Basic string matching terminology

Let $\Sigma^*$ be a set of finite length strings.

$$\Sigma^* = \{\text{String}_1, \text{String}_2, \ldots\}$$

Each character of a string $\in \Sigma$, where $\Sigma$ is a finite alphabet. Let $\epsilon$ be an empty string, $\epsilon \in \Sigma^*$. The length of a string $X$ is denoted by $||X||$. Let $X$ and $Y$ be two strings; then the concatenation of $X$ and $Y$, represented by $XY$, will have length

$$||XY|| = ||X|| + ||Y||.$$

**Prefix of a String:** String $W$ is a prefix of $X$, denoted by $W\lbrack X$, if $X=Wy$, for some $Y \in \Sigma^*$ and $||W|| \leq ||X||$.

**Suffix of a String:** String $W$ is a suffix of $X$, denoted by $W\rbrack X$, if $X=YW$, for some $Y \in \Sigma^*$ and $||W|| \leq ||X||$.

The empty string is both a suffix and prefix of any string. Let $X$, $Y$ be two strings and let ‘A’ be a character. We have $X\rbrack Y$, $X$ is suffix of $Y$, if and only if $XA\lbrack YA$. For example, let $X = cc$, and $Y = bcc$, we have $X\rbrack Y$ [14]. If we concatenate ‘a’ to both the strings, we have cca$\lbrack bcca$, which is also true.

For a pattern $P[1\ldots m]$, a $k$-character prefix is denoted by $P_k$, where $P_k = P[1\ldots k]$, the first $k$ characters of pattern $P$. $P_0$ is an empty string, $P_m$ is the full pattern $P[1\ldots m]$. For a text $T[1\ldots m]$, a $k$-character prefix is denoted by $T_k$, where $T_k = T[1\ldots k]$, the first $k$ characters of text $T$. 

28
**Problem Statement:** Finding all occurrences of a pattern in a text. A pattern may be a set of words or a single word.

**Instance:** Text $T[1....n]$ where $||T|| = n$; Pattern $P[1....m]$ where $||P|| = m$ and $m <= n$. Let $\Sigma$ be a finite alphabet. An example of $\Sigma$ can be $\Sigma = \{0, 1\}$ or $\Sigma = \{a, b, \ldots, z\}$. $P$ and $T$ are called strings of characters $\in \Sigma$.

**Question:** Can we find all valid shifts(s), with which pattern(P) occurs in text(T)?

A pattern $P$ occurs with valid shift $s$ in $T$, if

$$0 \leq s \leq n - m$$

and

$$T[s + 1, s + 2 \ldots s + m] = P[1 \ldots m]$$

Let us take an example:

Let $\Sigma = \{a, b, c\}$

$$T = [abbcabbaa]$$

$$P = [bca]$$ Shift $s = 2$

It is evident from the example that pattern $P$ occurs in $T$ with a valid shift of two.

The next section will explain *the state of the art* in string matching, from the least efficient to the most efficient method to search and match a string
pattern. This is followed by an explanation of the Aho-Corasick algorithm [6], which has been a motivation for the proposed framework.

2.7.3 Literature survey

Since the 1970’s, around 80 algorithms have been published for the string pattern matching problem [3]. The very obvious and oldest algorithm for searching a string pattern P in a text T is the Brute Force approach that locates all the occurrences of P in T in time O(mn), where \(\|P\| = m\) and \(\|T\| = n\). This is a quadratic time algorithm [14]. Other improved algorithms theoretically should be better than quadratic time. Various researchers have come up with different ideas [14]. The work done by Knuth, Morris, Prat(KMP) and Alfred Aho [5, 27] had a significant contribution in the string matching community. They came up with near linear time algorithms for string matching based on the concept of finite automata [6]. To date their work is considered as a milestone in the string matching community.

Even though the results were close to optimal, with rising applications and needs of users, researchers started to come up with different variants which work better for dedicated applications. This gave room to the community to design new algorithms or a variant of milestone algorithms to suit the needs. Figure 2.4 shows 35 major algorithms in the literature, placed in a tree structure, describing parent-child relationship between various algorithms.

Figure 2.4 helps to understand, what are the milestone algorithms and what
are their variants? Algorithms number 2, 6, 13 (shaded grey) are the parents of the majority of the child algorithms. By looking at the details, one will find that they more or less use a base model (base idea) from the corresponding parent algorithm.

The naive Brute Force approach has quadratic time complexity. Finite automata based algorithms take $O(m||\Sigma||)$ time for building the automaton and $O(n)$ time in the search phase [5, 14]. This linear time searching makes finite automata a stable and optimal model for string matching [43]. The only room for improvement is in the preprocessing phase, which has been addressed by the KMP algorithm [27]. It is also one of the milestone algorithms in the
literature. Their algorithm is also based on finite automata; the only difference in their approach is that the pre-processing time, $O(m)$, is independent of the length of the alphabet. The searching time is the same, $O(n)$, where $n$ is the length of the text to be parsed. The only problem with the finite automata based approach is that we need to have a pattern beforehand to build the machine. However, in real life search, (for example, on the web or any test document), we may not have the predefined list of patterns. Another algorithm in the milestone list is the Boyer-Moore algorithm [11], which is considered as one of the most efficient and effective string-matching algorithms in real life applications, where a pattern is not defined beforehand. It is mostly used in text editor programs. It takes $O(m+c)$ time in the pre-processing phase, where $c$ is a constant, and $O(mn)$ time in the searching phase. It does a right to left scanning of a pattern. It uses two pre-computed shifts, a good-suffix shift and a bad-character shift to shift the window to the right and match the pattern.

### 2.7.3.1 Other algorithm’s complexity

Table 2.1 gives an idea of various algorithms and their complexities.

<table>
<thead>
<tr>
<th></th>
<th>Algorithm</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Brute Force algorithm</td>
<td>$O(mn)$</td>
</tr>
<tr>
<td>2</td>
<td>Deterministic Finite Automaton algorithm</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>3</td>
<td>Karp-Rabin algorithm</td>
<td>$O(mn)$</td>
</tr>
<tr>
<td>4</td>
<td>Shift Or algorithm</td>
<td>$O(n)$</td>
</tr>
<tr>
<td></td>
<td>Algorithm</td>
<td>Time Complexity</td>
</tr>
<tr>
<td>---</td>
<td>-----------------------------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>5</td>
<td>Morris-Pratt algorithm</td>
<td>O(m+n)</td>
</tr>
<tr>
<td>6</td>
<td>Knuth-Morris-Pratt algorithm</td>
<td>O(m+n)</td>
</tr>
<tr>
<td>7</td>
<td>Simon algorithm</td>
<td>O(m+n)</td>
</tr>
<tr>
<td>8</td>
<td>Colussi algorithm</td>
<td>O(n)</td>
</tr>
<tr>
<td>9</td>
<td>Galil-Giancarlo algorithm</td>
<td>O(n)</td>
</tr>
<tr>
<td>10</td>
<td>Apostolico-Crochemore algorithm</td>
<td>O(n)</td>
</tr>
<tr>
<td>11</td>
<td>Not So Naive algorithm</td>
<td>O(mn)</td>
</tr>
<tr>
<td>12</td>
<td>Boyer-Moore algorithm</td>
<td>O(mn)</td>
</tr>
<tr>
<td>13</td>
<td>Turbo BM algorithm</td>
<td>O(n)</td>
</tr>
<tr>
<td>14</td>
<td>Apostolico-Giancarlo algorithm</td>
<td>O(n)</td>
</tr>
<tr>
<td>15</td>
<td>Reverse Colussi algorithm</td>
<td>O(n)</td>
</tr>
<tr>
<td>16</td>
<td>Horspool algorithm</td>
<td>O(mn)</td>
</tr>
<tr>
<td>17</td>
<td>Quick Search algorithm</td>
<td>O(mn)</td>
</tr>
<tr>
<td>18</td>
<td>Tuned Boyer-Moore algorithm</td>
<td>O(m+n)</td>
</tr>
<tr>
<td>19</td>
<td>Zhu-Takaoka algorithm</td>
<td>O(mn)</td>
</tr>
<tr>
<td>20</td>
<td>Berry-Ravindran algorithm</td>
<td>O(mn)</td>
</tr>
<tr>
<td>21</td>
<td>Smith algorithm</td>
<td>O(mn)</td>
</tr>
<tr>
<td>22</td>
<td>Raita algorithm</td>
<td>O(mn)</td>
</tr>
<tr>
<td>23</td>
<td>Reverse Factor algorithm</td>
<td>O(mn)</td>
</tr>
<tr>
<td>24</td>
<td>Turbo Reverse Factor algorithm</td>
<td>O(n)</td>
</tr>
<tr>
<td>25</td>
<td>Forward Dawg Matching algorithm</td>
<td>O(n)</td>
</tr>
</tbody>
</table>
Table 2.1: Complexity of String Matching algorithms

<table>
<thead>
<tr>
<th></th>
<th>Algorithm</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td>Backward Nondeterministic Dawg Matching algorithm</td>
<td>O(mn)</td>
</tr>
<tr>
<td>27</td>
<td>Backward Oracle Matching algorithm</td>
<td>O(mn)</td>
</tr>
<tr>
<td>28</td>
<td>Galil-Seiferas algorithm</td>
<td>O(n)</td>
</tr>
<tr>
<td>29</td>
<td>Two Way algorithm</td>
<td>O(n)</td>
</tr>
<tr>
<td>30</td>
<td>String Matching on Ordered Alphabets algorithm</td>
<td>O(n)</td>
</tr>
<tr>
<td>31</td>
<td>Optimal Mismatch algorithm</td>
<td>O(mσ)</td>
</tr>
<tr>
<td>32</td>
<td>Maximal Shift algorithm</td>
<td>O(mn)</td>
</tr>
<tr>
<td>33</td>
<td>Skip Search algorithm</td>
<td>O(mn)</td>
</tr>
<tr>
<td>34</td>
<td>KMP Skip Search algorithm</td>
<td>O(n)</td>
</tr>
<tr>
<td>35</td>
<td>Alpha Skip Search algorithm</td>
<td>O(mn)</td>
</tr>
</tbody>
</table>

The work done by Aho and Corasick [6] had a significant contribution in the string matching community. They came up with a linear time algorithm for string matching based on the concept of finite automata. The algorithm is simple, fast and efficient in finding all the occurrences of a finite set of patterns in a text. This is useful in applications where certain keywords need to be located in a text.

The algorithm consists of two parts. First a finite state pattern matching machine is constructed from a set of keywords. Then, in the second step, this state machine is used to parse text documents linearly to find all the occurrences of keywords in the document. They specifically designed the...
algorithm for a bibliographic search.
The core of the algorithm is the concept of a go-to graph and a failure function [6]. Failure transition helps to avoid unnecessary transitions if a pattern fails to match at some given state. Skut et al. at Google Switzerland [43] also suggests the use of failure transitions and the compression of transition tables, which reduces the memory footprint up to 20-fold.
The framework proposed in this work has been inspired by Aho-Corasick’s work on failure state transitions. Instead of being limited to finding just a fixed set of keywords, this work views the problem from a different perspective. The pattern set is no longer fixed, it is incremental in nature, hence the patterns are read and matched at the same time. The concept of introducing a document cluster around the patterns, implicit information flow from one state to another using failure transitions, and the ability to increment the model at any time, are new in this work. The complete details of the framework are described in Chapter 3.

2.8 Concluding Remarks

The purpose of topic discovery is coming up with a meaningful topic discussed in a set of documents. In this chapter, various core technologies that help to perform valid and meaningful discovery of topics are discussed.
A survey of the feature selection literature suggests that the use of features,
more than just words, has enhanced performance in the past. To maintain
the meaning, word co-occurrence and ordered pairs of words are taken as
phrases. Features such as phrases, named entities and parts of speech turned
out to be promising candidates for modern-world text mining problems.
A survey of the clustering literature suggests that bag of word approach
and the vector space model might perform well with grouping, but when it
comes to labeling the clusters, its performance does not fulfill the human
user requirements. A compromise between perfect grouping and labeling is
suggested in co-clustering, where features and documents are clustered simult-
aneously. Various clustering algorithms that are suited for topic discovery
are discussed. Phrase based models using graph data models seem to be
promising candidates in order to efficiently perform clustering in a phrase
feature space.
Two benchmark data models, which satisfy most of the characteristics of the
state of the art in topic discovery, are also briefly described. They both have
near-linear time phrase based models with underlying pattern matching capa-
bility. A thorough survey is conducted for various string pattern matching
algorithms in the literature. Finite automaton based algorithms perform in
linear time in most of the applications.
Keeping in mind the various lessons from past research and the state of the
art discussed in this chapter, the framework in this research proposes an
automaton based pattern matching document model. The model is designed
to perform various text mining tasks and specifically topic mining simply and
in linear run time. The complete details of the framework are described in Chapter 3.
Chapter 3

Proposed Framework

3.1 Overview

In this chapter, the proposed framework for topic discovery and its theoretical feasibility are presented. The framework has been inspired by the way a human determines topics for a document, while still using the computational power of machines, thus making it valuable, valid, understandable and at the same time fast and computationally effective. The framework is capable of dealing with large amounts of data, which is normally far away from human processing.

It is important to understand the way humans analyze text mining problems before describing the proposed framework and its applications. In this work, the human way of analyzing text has been named HWAT (human way of analyzing text).
**What is HWAT?** It stands for Human Way of Analyzing Text. Even though there are many well developed and efficient machine learning and data mining algorithms to analyze text data, its subjective nature still needs human intelligence and evaluation. These methods are not as efficient as how a human would solve such problem.

When a human reads a text document, some noticeable and seemingly important elements of interest are kept in the memory. As more documents are read, some elements become more prominent and start making sense and some fade away, residing in the sub-conscious mind, that may or may not be triggered in future documents. All the documents are read in a linear fashion. The elements, along with statistical information, are aggregated in the memory. With past knowledge and intelligence, the human makes sense of these elements, and moreover of the documents.

Depending upon how well the stored information is being utilized, the human can summarize all the read documents, tag them with important concepts or major entities, determine the polarity of the text, and much more. The knowledge is constantly being upgraded and expanded upon in the brain, which makes the human capable of understanding documents and extracting knowledge out of them. However, human capabilities are limited when the size of a document set increases, whereas machines have proven to be efficient, fast and tireless. Therefore, combining the human’s effective and machine’s efficient processing powers is required in order to extract useful and meaningful knowledge out of text documents.
In this work, a Linear Pattern Matching and Parsing (LPMP) unit has been proposed, which is capable of reading text documents in a linear fashion, the same as a human would. Moreover, it also stores and self analyzes the information to perform various text knowledge mining tasks such as topic discovery, clustering, inverted indexing, results-centric query enhancement, identifying significant phrase patterns, and multi-document summarized overview.

Similar to the HWAT process, we propose this framework where knowledge elements are read, stored and analyzed over time. If there is no prior information present about an element, the system waits to aggregate more information, and if the element is already present in the memory, then more knowledge is gathered. The hypothesis is that in the end, the overlap between various significant knowledge elements, extracted from various documents, are rich enough to represent the whole document set.

3.2 Framework Overview

In this section, a brief overview of the proposed framework is given. All components are described in complete detail as they come along. The most important one is the Linear Pattern Matching and Parsing (LPMP) unit and its use in clustering and topic discovery.

Figure 3.1 shows all the processes of the framework, with the black box placeholders. It is important to note here that there could be many other
applications and uses of an LPMP unit that have not been covered in this work.

The very first black box is the Data Source. This box is responsible for gathering data from various sources such as news streams, twitters, blog-posts, emails and share-points. The purpose of this box is to provide a consistent stream of text data into the framework. This box also acts as a buffer collector for incoming data in order to have a smooth stream of text.

Next is the Pre-Processing box, where the text stream is cleaned, filtered and made suitable for the next processes. Feature selection is also performed in
LPMP is the central processing unit of this framework, where a stream of cleaned text is being parsed linearly. Important phrases and entities are stored and statistics are aggregated over time. Informative elements are not only stored but also matched at the same time as utilizing the pattern matching mechanism proposed in this box. The knowledge graph is built and stored in the graph database [48]. Based on a self triggering and alerting mechanism, various use cases are also called upon to perform various text mining tasks.

Next is the co-clustering module; this makes use of the information stored in the LPMP unit and performs the grouping of documents and at the same time labels them in order to discover automatic taxonomies [30]. The approach is motivated from co-clustering [16], where grouping and labeling are performed at the expense of each other. An evaluation scheme for the results is also described in this module. This module also presents some new concepts of building a story around a topic, which helps in understanding the context of a topic.

Indexing of the documents is performed in this module. For quick retrieval of information contained in documents, having an inverted file is very important. The indexing engine, in this work, is a bit different than traditional flat indexing. Along with flat indexing, sentence based graphical indexing
of information is also done, hence indexing the concepts not just the keywords.

The knowledge graph provides an ability to search more than just a keyword, thereby providing a search the meaning option in traditional search engines. Consider a query, jaguar, of which possible meanings could be jaguar cars, cat jaguar, or mac-os. Traditional query enhancements are centered around keyword matches. But in this framework, suggestions are made based on the most related concepts around a given query.

The main purpose of this framework is to utilize and incorporate the state of the art text mining approaches, mainly for topic discovery, to provide a capable plug and play functional unit to perform various text mining tasks. One of the main characteristics of this framework is that it is designed to work in the same way as the human brain thinks of a text mining problem, while also having the tireless computational power of machines to solve problems. The next sections describe the internal details of the modules.

### 3.3 Feature Extraction Phase

A text document is a set of strings and characters for a machine, but for a human it is more than just a string. For example:

*Joe has been working as a knowledge extraction engineer at IBM*
The above sentence contains a set of characters for a machine, but a set of knowledge elements for human. Joe is a person, who works at IBM since the year 2005, and his role is Knowledge Extraction Engineer.

In this work, a set of important phrases and entities are extracted from a text document. The hypothesis is, *this set is capable enough to represent and convey central concepts and ideas discussed in the whole document*. All other information such as stop words and broken phrases\(^1\) are less important to give unique representation to a document. The phrases are extracted from a single document in order not to depend on the whole corpus, which slows down the performance of the system in most of the traditional text mining processes.

We use the single-document *Rapid Automatic Keywords Extraction* (RAKE) algorithm for extracting weighted phrases\(^{[42]}\) and *Stanford’s Named Entity Recognizer* to extract entities from a text document. These extracted elements not only represent the documents but also become potential candidate topics for the whole corpus. The idea comes from both co-clustering\(^{[16]}\) and descriptive clustering \(^{[15]}\), where labeling is not done once the grouping has been achieved. Both grouping and labeling mutually guide each other. This work uses a very similar approach where candidate labels, or important knowledge elements, are extracted. Using the LPMP unit and the clustering process,

\(^1\)Broken phrases are single words which are supposed to be used in conjunction with other words to make sense. For example: Artificial Intelligence, both *artificial* and *intelligence* have different meaning, but when used together have a new meaning.
valuable and valid topics for the whole corpus are extracted.

3.3.1 RAKE Algorithm

RAKE is an efficient, single document oriented, domain and language independent phrase extraction algorithm [42]. It is based on the observation that phrases rarely contain stop words, punctuation and minimal lexical meaning words. The black box nature of the algorithm takes a text document as an input and returns a list of ranked and weighted phrases as an output. The algorithm is as follows:
Algorithm 1 RAKE Algorithm

1: INPUT: ( String inText, <stopwords>, <phrase delimiters >)

2: array[ ] candidate phrases ← inText.split(stopwords,phrase delimiters)

3: for all i ← 1 to c:length(candidate phrases) do

4: array[ ] unique words ← Candidate phrases[i]

5: end for

6: for all i ← 1 to n:length(Unique words) do

7: array [ ][ ] co-occurrence matrix ← co-occurrence frequency of unique
   word[i] with all other words in the list

8: end for

9: for all i ← 1 to n:length(Unique words) do

10: use information recorded in co-occurrence matrix to calculate:

11: \( F = \text{freq}(W) \)

12: \( D = \text{deg}(W) \)

13: \( \text{Score} = D/F \)

14: end for

15: for all i ← 1 to c:length(candidate phrases) do

16: For every Candidate Keyword overall score is calculated by summing
   \( \text{Score} \) of its constituent keywords.

17: end for

18: Sort the list of candidate phrases

The algorithm takes a text document and a list of delimiters as an input.
Delimiters could contain a list of stop words, a set of punctuation signs, words having minimal lexical meaning, and if required non content bearing words. In the next step the text is split into candidate keywords using stopwords and punctuation signs as delimiters. A candidate keyword could be a single word, or a combination of keywords. For every unique keyword in the candidate keyword list, a word co-occurrence matrix is created. For every word \( W \) in the list, frequency \( freq(W) \), degree \( deg(W) \) and the ratio of degree over frequency \( deg(W)/freq(W) \) are calculated, where \( deg(W) \) favors words that co-occur frequently in candidate keywords and \( freq(W) \) contributes to how often a word occurs in a text regardless of whether it is in conjunction with other words. Furthermore, the keywords occurring predominately are favored by the ratio of the two measures \[42\]. To calculate the score of a candidate keyword, the score of its constituent keywords are added. The final output of RAKE is the list of ranked weighted phrases, which is a compact representation of the whole document.

### 3.3.2 Entity Extraction

The phrases that are extracted by RAKE are a set of co-occurring strings. The keywords consisting of these phrases could carry more meaning for a human. Consider a string phrase extracted by the RAKE algorithm: “Product Manager Joe Smith”; it can be enriched with an additional layer of knowledge by annotating the strings with entity type. These annotations and extra layers of knowledge have proven to be rewarding in the later stages of topic discovery.
This research uses Stanford’s entity extraction module to extract three types of entities: Person, Organization and Location. The module could also extract other entities such as Time, Money and Percentage, but these extra features are not useful in establishing the uniqueness in story and topic discovery processes.

A document may or may not contain entities. If it does, the list of entities is extracted. For every entity in a list, its weight is determined by taking the ratio of frequency over the total number of entities discovered in the text.

### 3.3.3 Document Mapping

The list of entities, extracted from a document, is fed in as a delimiter list in the phrase extraction phase by RAKE. This is done so that the strings that are already extracted as entities are not extracted again as string phrases.

As shown in Figure 3.2, entities are extracted first and then phrases are extracted. The results are stored in a list which consists of both weighted entities and phrases. Hence, the document is now mapped in phrases and entities space, which acts as a compact representation of the document.

### 3.4 Linear Pattern Matching and Parsing Unit

As discussed in Chapter 2, when the requirement is to discover understandable topics, purely vector space clustering fails to provide meaningful labels.
Co-clustering [16] is compromising way in which features and documents are clustered simultaneously, with the assumption that a good clustering of the document is one that provides good clustering of the features (keywords, n-grams or phrases), therefore providing a meaningful cluster description.

In the literature, there are two benchmark data models [52, 21] that use graph theory to perform phrase based co-clustering. The idea is to extract candidate topic labels and, with the help of graph matching algorithms, to group documents around them. In the end, the topic is described either by a single specific topic label or by a set of keywords in order to get the general sense of the topic discussed, as it is hard for a computer to give just a one word description to a set of documents [30].

The first is the suffix tree model (STC) [52], which uses a trie as an underlying structure. The tree grows as the suffixes are added. For each suffix added, a
new branch is started from the root node. If the branch encounters a subpart of a suffix already existing, it just updates the shared path; otherwise it grows a new path. There are lots of redundancies in the creation of edges carrying the keywords. The growth mechanism of the tree is not optimal. Also the phrase matching mechanism is very naive as it does not utilize the information of already matched suffixes.

The second is the Document Indexing Graph (DIG) [21], which uses a graph as the underlying structure. The key phrases extracted from the documents are added in the graph structure with document and sentence information. Every node is a word in a sentence. Hence, the number of nodes in the graph is the same as the total number of unique words in the corpus. In terms of redundancies, DIG is promising as it utilizes the same node every time an existing word needs to be added to the graph. Thus, the growth mechanism of the graph is better than STC. The pattern matching is also reduced to finding shared paths in the graph; however, keeping information of all the shared paths and edge table in every node is work intensive, and it is also intensive to know for every word to be added if the node already exists or not. In the case of a larger corpus, the number of nodes rises and comparing all the nodes every time a new node needs to be added is computationally not promising.

Another model proposed by Li et al. in a research collaboration between the University of Alberta and Google, Canada [30], uses a graphical underlying structure to build the communities of keywords, map the documents on top of
these communities, and then use the concept of community mining in social network analysis to discover the coherent hierarchical clusters and automatic taxonomies. The model is an approximation to frequency based grouping of document sets around keywords. The approach is very simple to implement, making it a good candidate for real time applications. This has always been a strategy of search engines like Google, to use simple and understandable approaches, instead of computationally expensive ones.

In all the models, there are two major parts, 1) Graph growing mechanism and 2) Pattern matching approach, along with an information aggregation mechanism. An optimal graph growth and near-to-linear pattern matching mechanism is desired. There is a possibility to grow a graph intelligently by using transition tables and failure states [43]. For pattern matching, there exist ways to intelligently utilize the information already gathered for a given match and utilize it later to avoid repeat matching [6, 27]. An optimal data model should accurately and completely capture the salient features of the data [21].

In the next section, we propose a new underlying data model which allows both the growth of a graph and pattern matching in an intelligent way by utilizing concepts from graph theory and the theory of automata.
3.4.1 Proposed Data model

The proposed model is not just an extension of STC or DIG; rather it tackles the problem of building the data model from a different perspective, using the notion of state machine. The theory of automata will be explained briefly in the next section, in order to better understand the proposed data model, followed by details of the data model.

3.4.1.1 Theory of Automata

A finite automaton $M$ is a 5-tuple $(Q, q_0, A, \Sigma, \delta)$ where $Q$ is a finite set of states, $q_0 \in Q$ is the start state, $A \subseteq Q$ is a set of accepting states, $\Sigma$ is a finite alphabet, and $\delta$ is a transition function of $M$, that is, $\delta : Q \times \Sigma \rightarrow Q$.

The automaton starts from the start state $q_0$, then it reads the input symbol ‘a’. If the present state is $q$, then it makes a transition from state $q$ to the next state defined by a transition function $\delta(q, a)$. If the next state belongs to an accepting state $A$, we say the machine $M$ has accepted the pattern read so far, otherwise it is rejected [14]. In some cases, if the next state is a reject state, a transition function directs the next state to some previously accepted state [6], where the pattern could continue to match the other possibilities without starting from the beginning - saving the unnecessary building of states. The machine induces a final state function $\phi$ from $\Sigma^*$ to $Q$ such that $\phi(w)$ is the state where $M$ ends up after scanning the string $w$. Thus, $w$ is accepted if and only if $\phi(w)$ belongs to $A$. This function is also called the
output function [6] frequently in the literature.

The definition of automata is open to the application it is being designed for. An automaton has **finite input** if it accepts a finite sequence of symbols, is a **finite state machine** if it contains a finite number of states, is **deterministic** if for a given current state and input symbol there is one and only one possible next state.

In this work, we design a Deterministic Finite State Machine, called a Pattern Matching Machine (PMM), on top of an underlying graph data model to efficiently match the phrase patterns in the documents and group documents around those concepts, while at the same time utilizing the underlying node and edge structure.

### 3.4.1.2 Structure Overview

PMM is a directed state machine, with nodes and edges. Figure 3.3 represents the atomic unit of the data model. The details of the various parts are as follows:

- **Start State:** is a root node containing the paired list $< \Sigma \rightarrow q>$ of input symbols with corresponding next state transitions. For input alphabet $\Sigma \notin < \Sigma \rightarrow q>$, the next state is built and the symbol is added in the list for future matching.

- **State Node:** In this model a state node not only contains the state
number, but also contains statistics about phrases and documents. This information helps the state machine to grow further and at the same time as matching the patterns. The following information is stored in a state node:

- **State Number**: The total number of states is finite. During a transition, the machine jumps to one and only one state. Start state is given the number zero.

- **Depth** is the measure of the degree of separation from the start node to the current node.

- **Type of feature**: either a string phrase or an entity.

- **Underlying pattern** is the concatenation of all the words along the edges starting from the start node to the current node.
– List of **document-weight pairs**, captures the information about the document in which the underlying phrase occurred, along with its local weight in the document.

– List of **document-time pairs**, to keep track of the time a phrase was added by each document. This information helps to understand the dynamics of a topic being discussed.

– **Size** of the node is determined by the number of documents in the document list.

– **Edge List** contains the list of edges (or words) outgoing from a given node.

– **Failure Function** defines the failure state transition to the next accepting state when a pattern match fails.

– **Output Function** is a boolean value to flag whether the underlying phrase is completed or not.

**• Edge**: carries information about input symbol \( \Sigma \). In this work, a symbol is a single word of a phrase. Every edge also contains information about the next transition state; that is, \( \delta : Q \times \Sigma \rightarrow Q \). Two nodes are connected if the underlying word of the destination node appears successive to the underlying word of the source node. As the phrase is read, several states are connected through the edges. The path from the start node to end node is the full phrase. Therefore, a phrase or
sentence structure is being maintained in this model.

Having defined the atomic unit of the data model, the next section now describes the construction of the data model in order to capture the information, in documents, in a well formed graphical structure.

### 3.4.2 PMM Construction

The data model is built incrementally as documents are processed. Building a state machine graph is a two-step process.

In **step 1**, the document is mapped to feature space; each phrase of the document is converted into an ordered list of keywords. Now commencing from a start state $q_0$, keywords are entered. If the keyword’s edge already exists, the statistics of the underlying node are updated with the new document information; otherwise a new edge and node are created. The edges are only added when necessary. The growth rate of the states becomes stable when the dictionary becomes stable. When the whole phrase is read, the output function in the corresponding node is flagged *TRUE*. The total number of keywords in the phrase is the same as the length of the path from the start node to the end node. Hence, a phrase is converted to a path in a graph. Algorithms 2 and 3 describe the pseudo algorithm in more details.
Algorithm 2 Construction of the Graph

1: INPUT(List of Documents $D = [d_1,d_2...d_n]$)
2: for all $i \leftarrow 1$ to $n$ do
3:   Extracted Phrases $\langle\text{phrase}, \text{weight}\rangle \leftarrow \text{rake}(d_i)$
4:   Extracted Entities $\langle\text{entity}, \text{weight}\rangle \leftarrow \text{EntityExtraction}(d_i)$
5:   for all $j \leftarrow 1$ to $p$: Length(Extracted Phrases) do
6:     enter($\text{Phrase}_j, \text{weight}, \text{ditime}, \text{Doc}_i\text{ID}$);
7:   end for
8:   for all $m \leftarrow 1$ to $e$: Length(Extracted Entities) do
9:     enter($\text{Entity}_m, \text{weight}, \text{ditime}, \text{Doc}_i\text{ID}$);
10:  end for
11:  end for
12:  Compute Failure Function()
13:  Print the Graph

Algorithm 2 Analysis: It can be seen that the for loop in line 2 runs for a number of times equal to the number of documents $n$. Now for each document $d_i$, Algorithm 1 is called to extract a weighted pair of phrases. Similarly, entities are extracted in the same way at line 4. Now for every phrase in document $d_i$, the enter() function is called to enter the phrase in the state machine graph. Once all the documents are entered in the graph, failureFunction() is called at line 12 to finalize the output function and compute the failure transition state. The run time of Algorithm 1 is linear [42]. Hence, the overall
Algorithm 3 Entering phrase in the graph

1: \textbf{enter}(Phrase_j, weight, d_{time}, Doc_iID)
2: currentState \leftarrow q_0: \text{start state}
3: \textbf{for all} i \leftarrow 1 \text{ to } K: \text{(Keywords in Phrase)} \textbf{do}
4: \hspace{1em} \textbf{if} Keyword \in \text{CurrentState.EdgeList} \textbf{then}
5: \hspace{2em} nextState \leftarrow \text{edge.transitionState}
6: \hspace{2em} nextState.updateDocument-Weight List
7: \hspace{2em} nextState.updateDocument-Time List
8: \hspace{2em} \textbf{if} i = k \textbf{then}
9: \hspace{3em} nextState.outputFunction = TRUE
10: \hspace{2em} currentState \leftarrow \text{startState}
11: \hspace{2em} \textbf{else}
12: \hspace{3em} currentState \leftarrow nextState
13: \hspace{2em} \textbf{end if}
14: \hspace{1em} \textbf{else}
15: \hspace{2em} nextState = \text{new StateNode(PhraseType, CurrentNode.Depth)};
16: \hspace{2em} currentNode.addEdgeList(Keyword, nextState);
17: \hspace{2em} nextState.setPhraseType = \text{phraseType}
18: \hspace{2em} nextState.setDepth = currentNode.depth+1
19: \hspace{2em} nextState.setPhrase = Concatenate("CurrentNode.Phrase", "Keyword")
20: \hspace{2em} nextState.addDocument-Weight List
21: \hspace{2em} nextState.addDocument-Time List
22: \hspace{2em} \textbf{if} i = k \textbf{then}
23: \hspace{3em} nextState.outputFunction = TRUE
24: \hspace{2em} currentState \leftarrow \text{startState}
25: \hspace{2em} \textbf{else}
26: \hspace{3em} currentState \leftarrow nextState
27: \hspace{2em} \textbf{end if}
28: \hspace{1em} \textbf{end if}
29: \textbf{end for}
run time complexity of Algorithm 2 is $O(np)$, which is a linear term of $n$ as $p \ll n$. The algorithm to enter a phrase to a graph is described as Algorithm 3.

**Algorithm 3 Analysis:** When a new phrase is to be entered in the graph, `CurrentNode` is the set to start state, $q_0$, in line 2. The phrase is split into a list of ordered keywords. The for loop at line 3 runs for all $K$ keywords in the phrase. For every keyword $P_k$, it checks if there is a path from `CurrentNode` to `NextNode` at line 4. If the path does not exist, the algorithm goes to the `else` part at line 14. A new state is being initialized at line 15. A new Edge is created with `CurrentNode` as the source and `NewNode` as the destination at line 16. The depth of the `NextNode` is set as the depth of `CurrentNode` + 1 in line 18. The underlying phrase of the `NextNode` is set as the underlying phrase of `CurrentNode + P_k` in line 19. The information about the document weight and time pair is also added to `NextNode` in line 20 and 21. If all the keywords are read for a given Phrase, `CurrentState` is set as `StartState` to start from the beginning to enter a new phrase and `OutputFunction` is set to TRUE, meaning some phrase completed at this node, else `NextState` becomes the `CurrentState` and the process to add other keywords continues. The algorithm runs for $K$ time and for each keyword it checks if the edge exists. The list of edges are stored in `HashMap`, which has an average lookup time of $\Theta(1)$ [49, 5, 14]. Hence, the overall complexity of Algorithm 3 is of the order of $K$, $O(K)$, where $K$ is the number of keywords in a phrase.
In Step 2, once all the phrases are mapped to the state machine, the partial output function and failure function need to be completed and computed. Computing the failure function adds intelligence to the given graph. It actually helps to avoid unnecessary transitions and, if for any state the transition fails to proceed further, the failure function guides the machine on which state to go to next. This property of the data model helps in efficiently performing the pattern matching. If a pattern has been matched half-way, and the next keyword does not match the next one in the edge list, the transition function is guided by the failure function to find which other branch has already matched the pattern matched so far. This avoids the process of going back to the start state and starting to match again. The idea of the failure function will become clear later in this section, where it will be explained with the aid of an example.

Computing the failure function is a process of incrementally computing the failure state at every depth using the depth of the previous state. For nodes at depth one, the failure function is directed to the start state. Algorithm 4 explains how to compute the failure function of an incomplete graph created in step one.

**Algorithm 4 Analysis:** The algorithm takes the StartState as an input parameter. All the states of depth one are added in the queue and the FailureState is set as the StartState. The while loop at line 9 runs as long as the queue is not empty. The queue contains all the states at depth one. Each
Algorithm 4 PMM: Calculation of Failure Function

\begin{enumerate}
\item **INPUT:** (startnode)
\item queue ← empty
\item Edges< String > ← startnode.getAllEdges
\item for all i ← 1 to e: Length of Edgeset<> do
\item currentState ← e\(_i\).getNextState
\item currentState.failureState ← startstate
\item queue ← queue ∪ currentState
\item end for
\item while queue ≠ empty do
\item State ns ← queue.nextState
\item queue ← queue − ns
\item if ns.edgeList ≠ empty then
\item for j ← 1 to ns.Size do
\item State tempState ← j\(_th\) edge in ns
\item queue ← queue ∪ tempState
\item state ← tempState.getFailureState
\item while j\(_th\) edge \∉ state.edgeList do
\item state ← state.getFailureState
\item end while
\item temp.setFailureState ← state.getTransitionfor(j\(_th\)edge)
\item state.AddDocumentList ← temp.getDocumentList
\item end if
\item end while
\end{enumerate}

state is loaded from the head of the queue and put into a new state variable, \(ns\), and at the same time deleted from the queue. The queue elements are pushed ahead in a first come first serve pattern. From line 12 to 16 the algorithm traverses through the states until it reaches the fail state. Along the traversal, the states are added in the queue. While traversing through one path, when the fail state is encountered, the state at one depth before this state is used to set the failure state for this state. Hence, in this iterative pro-
cess all the failure states of depth d are computed using the states at depth d-1.

One important step is to transfer the document information from one node to another. If A is the source state and its failure state is B, then all the document information from state A is transferred to state B. This step is very crucial in performing effective pattern matching between the various possibilities. This will be cleared with an example in the next section. The run time of Algorithm 4 is bounded by the sum of the length of the phrases. The while loop at line 9 runs until the queue is not empty. But it can also be seen that the elements from the queue are also deleted as they are processed. In the worst case, all the unique keywords could be attached to the start node. If we have P phrases in total and the sum of all the unique keywords of P is $K_p$. Hence, there could be in the worst case $K_p$ state nodes starting from the start node. Hence, the initial size of the queue is $K_p$. As the size of queue changes while we process to other depths, there would be a constant, $m$, in the factor to compute the run time complexity. The overall complexity of Algorithm 4 is $O(mK_p)$.

### 3.4.3 An Example

Let us now take some sample phrases and make a state machine graph. This section will also help to emphasize the importance of the failure function, how the proposed model intelligently, in linear time, matches the patterns, and how the overall model behaves with real data.
Consider the following 4 phrases, with their corresponding weight and time information:

- $D_1$: boston bombing ($w_1, t_1$)
- $D_2$: casualties boston bombing ($w_2, t_2$)
- $D_3$: boston attack causalities ($w_3, t_3$)
- $D_4$: boston bombing several casualties happened ($w_4, t_4$)

Let us now take one phrase at a time and incrementally build the state machine graph.

**Enter $D_1$: boston bombing ($w_1, t_1$)**

![State Machine graph: Enter($D_1$)](image)

In Figure 3.4, State 2 is an output state which means the phrase ended at this node. There is only one document in States 1 and 2. Here we can see the phrase structure is now converted to the directed graph path. Let us enter
another phrase on top of the current graph.

Enter $D_2$: casualties boston bombing ($w_2$, $t_2$)

Figure 3.5: State Machine graph: Enter($D_2$)

Figure 3.5 shows that three more nodes, 3, 4 and 5, are added to the existing graph model. State 5 is another output node.

Enter $D_3$: boston attack casualties ($w_3$, $t_3$)

With $D_3$ added, the edge for “boston” is shared. From node 1 there is a branch going to nodes 6 and 7. The document list and the corresponding information of node 1 is updated with two edges going out, “bombing” and “attack”. Node 7 is another output node added at this stage.

Enter $D_4$: boston bombing several casualties happened ($w_4$, $t_4$)

As $D_4$ is added node 1 and 2 are shared and from node 2 there is a branch to node 8. Nodes 1 and 2 now contain $D_4$’s document information. Now that
there are no more phrases to be added, the initial building of the graph is complete. Some nodes have document groupings and some only have a single document within them. The next step is to compute the failure function for each node and to make the existing model complete in order to match any sequence of pattern.

To understand the importance of the failure function, let us take an example. In Figure 3.7, even though all the phrases have been entered in the graph, it is not yet complete. In other words, it is not yet efficient in matching all possibilities of the phrase combination. From the above graph, the phrase “boston bombing” is shared between $D_1$ and $D_4$ at node 2. But “boston bombing” also occurs in $D_2$. Documents $D_2$, $D_3$ and $D_4$ should all be grouped under
Figure 3.7: State Machine graph: Enter($D_4$)

the phrase “casualties”, but they are all sitting alone in node 3, 7 and 9. The incomplete graph model cannot match the pre, post and infixes, leading to an incomplete grouping of the documents. Therefore, a mechanism is needed to match phrases in all possible ways in order to capture the implicit grouping in the data.

The Suffix tree model (STC) [52] can solve this problem by splitting all of the phrases into all of the possible suffixes and creating separate tree branches to match all the possible phrases—generating many redundancies in the process. DIG [21] would solve this problem by having one node for a word, and it would match all post, pre and infix matches. This way requires intensive indexing of nodes; every time a node is added it needs to be checked if the node exists or not. Another bottleneck to this approach is that there is an overwhelming amount of information stored in one node. However, storing
only the context specific information for a word in a node is advisable. Every word has some local perspective and if we put all the information of a word in one place, we might lose the context of a word. Moreover, there will also be a large list of edges and sentence information lookup every time we need to traverse the node. Therefore, a trade off between redundancy and intensive lookup is required.

We proposed this model where the construction process is finite and deterministic, making it fast and reliable. It also does not need to maintain a long lookup list for edge and sentence information. At the same time, all possible phrase matching could be performed linearly without having any restriction of one node per keyword, making it flexible in nature.

Let us compute the failure function for each state according to Algorithm 4. Figure 3.8 shows the computed failure states for each state. Using this information, the flow of information happens in the graph, making this model capable enough to capture the features of data in a very simple, linear, memory efficient way, unlike STC and DIG.

In Figure 3.9, after adding the failure function transaction states, the implicit information flows inside the graph can be seen. The node information of node 7 and 9 are copied in node 3, making it a node which completely captures the information stored in the data. The phrase “casualties” now groups the
documents $D_2$, $D_3$ and $D_4$ together, which was not the case before. The phrase “casualties” occurs in the beginning (prefix) of $D_2$, in the end (postfix) of $D_3$ and in middle (infix) of $D_4$.

Hence, with the concept of failure transitions, all the possible combinations of
matching phrases could be performed. The model is now capable enough to intelligently match the phrase. Mixing concepts from graph theory, automata and failure transition theory, the proposed model proves to be a promising, yet powerful data model that captures the salient features of data in a linear and simple way.

### 3.4.4 Underlying Graph Indexing Model

The proposed data model is capable of capturing salient features of data. However, the information first needs to be indexed efficiently in order to use the model for various text mining tasks.

In traditional flat indexing, a keyword-document inverted list is stored in long tables. Upon query, the inverted list lookup is done and the resulting intersecting set of documents is returned.

This work approaches the indexing problem from a completely different perspective. It not only stores the flat inverted indexes, but on top of the flat layer, it adds a layer of connected knowledge elements, to easily mine and utilize the information. A graph database (Neo4j) [48] provides the backbone to the framework by storing the data model. A graph database can store the data in connected form. The queries are performing traversals in the graph. Neo4j has flat indexing by the popular Apache Lucene engine [1]. The information on nodes and edges can be indexed. It is a lookup list for data to a node or edge. A simple example is shown in Figure 3.10.
In this work, the node’s underlying phrase and keyword on the edge are indexed and the state machine graph is stored in the graph database. For a given phrase pattern, it returns the node or edge that contains the phrase. Information about the documents that contain the phrase is present in a node. Furthermore, the retrieved node is also connected to other nodes through edges. Hence, through traversals, the phrase and sentence structure is maintained along with phrase co-occurrence information. This graphical indexing capability turns out to be more useful than just flat indexing. In the next sections, it will become evident how well this data model and indexing scheme can benefit in performing various text mining tasks in an efficient, understandable and easy way.
In the following sections, various tasks have been described which could be performed utilizing the proposed data and indexing model. The main focus is on topic and story mining and on the meaning search module.

### 3.5 Topic and Story Mining

The documents are mapped to phrase space, and phrases are then mapped to graphical space. Pattern matching and the absorption of information is happening in the data model. Behind the scenes, simultaneous occurrences of various events give rise to potential topics in the data. The major one is the matching of patterns and grouping of documents around those topic seeds. There is an implicit co-clustering phenomenon occurring. The potential topic candidates are matched and at the same time documents are gathered around them. The document set provides an idea whether a candidate is worth becoming a topic or not. It may also be possible that a noisy pattern is occurring in all documents; hence, using only frequency may not be a good measure to discover all topics. STC [52] only uses the frequency measure to extract the base clusters from their data model. In their data model, there are several heuristics that are being used to extract the differentiating topics and stories, discussed in the data.

- **Minimum Support:** Minimum Support (\text{min sup}) is defined as the
minimum number of documents a node (cluster) should have in order to be considered for a potential topic.

- **Importance of Nodes:** Every node satisfying the min sup criteria is a candidate for a potential topic. The information content stored in each node helps to give a weighted rank to the node. Some nodes have more content and some have less. It is, in some sense, the same as the HWAT process where after reading all documents the overlap of the portions of the brain that are more activated contains the candidate for the topics of discussion.

![Figure 3.11: Node Importance](image)

Every node has a list of Document-Weight pairs. The importance of a node is defined as the sum of the weights of the underlying phrase,
multiplied by the ratio of the total number of documents $N$ in the corpus and the document frequency of the node.

$$Importance = \sum_{i=1}^{m} W_i \times \log\left(\frac{N}{\left|DocList\right|}\right)$$  \hspace{1cm} (3.1)

The local score of the phrase in a document provides the local importance of a phrase in a document; the document frequency neutralizes the effect of a phrase in the global context. Thus, a phrase with a very high weight in one document may be ranked low in the global context. Unlike TFIDF [33], which uses the whole corpus to calculate the values, we just calculate the importance with a subset of documents which are only grouped under a given node. Therefore, the importance of the phrase is not being diminished in the bigger context.

- **Completeness of topic:** Consider a scenario described in Figure 3.12. The importance and minimum support of node 1 and 2 are the same; taking “artificial” as one potential topic and “intelligence” as another independent topic is not suggested. The meaning of the phrase is captured when the phrase boundaries are maintained. The edge information helps to measure the completeness of the phrases. In the given scenario, the phrase “artificial” has one edge and the phrase “artificial intelligence” has 4 edges, meaning that the latter pattern is a good candidate for being the parent of sub-topics; therefore, it should be given more weight.
Figure 3.12: Completeness of topic

- **Topic Overlap**: The implicit nature of data is having topics overlapping each other. The underlying set of documents provides an easy way to know the document overlap percentage between the various nodes. This measure helps when the hierarchy of topics needs to be mined.

- **Time Factor**: In a stream where data comes with time information, it is advised to suggest topics that have been trending in a given time window.

With all these heuristics a ranked list of topics is generated for a given set of documents. A parameter $n$, to select the first $n$ topics, is used to adjust the granularity of the topic details. Implicit to topics which are discovered, the documents are also clustered. Therefore, the performance of the topic
extraction can also be evaluated on the side by comparing with the actual clusters of documents. Since grouping documents is very subjective in nature, perfect clustering cannot be achieved. The main focus of this work is to mine topics with various confidence measures—grouping being one of them. The actual grouping that is performed by the annotator might be different than the one done by this framework. It is all about the perspectives from which we look at the group of documents.

An interesting task, that has been researched in this work, is shifting the direction of topic mining to story/context mining.

### 3.5.1 Story/Context Mining

The topics that are discovered are good enough to describe the data. However, flat topics, without any context, might not make any sense. One way to extend flat clustering is to have hierarchies of topics. In most of the work in hierarchical clustering, hierarchies are created at the document level, although true conceptual and subjective hierarchies might be different. Consider an example of hierarchical clustering performed by a carrot² search engine; it is believed to be the benchmark in hierarchical clustering. For the query “Jaguar”, it returns the hierarchy as shown in Figure 3.13.

It can be seen that for the parent topic “jaguar cars(25)”, there is a child topic “wikipedia(3)”. The topic “wikipedia(3)” has been identified just because in

²http://search.carrotsearch.com/carrot2-webapp/search (Last accessed on July 04, 2013)
the search results there might be some pages for “wikipedia jaguar”, in which
the combination “jaguar cars” may have been used. However, conceptually
“wikipedia” is not a sub topic of “jaguar cars”.

In this work, the problem of taking flat clustering and merging topics to
create hierarchies has been dealt with using a different perspective. Instead
of merging, topic linking has been performed. Now a topic is not isolated, it
is connected with other topics-making up a story by giving context to a topic.
To link the topic to another topic, we make use of the document overlap
measure. The best K connected topics for a given topic are selected. The
degree of the node suggests the central topics with other surrounding topics. With a time window on top of it, we can see how the story actually evolved over time. Parameter K can be seen as the turning knob to know either deep or upper level details. From a human perspective, this way of looking at the document set is more useful than just giving the forced hierarchies. Forced hierarchy is the traditional hierarchy which forces all the documents into one big “blob” at the root level.

Figure 3.14: Query “Jaguar”: Results by proposed framework

Figure 3.14 shows the actual results by the proposed framework applied on the documents retrieved for the query “jaguar”. The results are only for one perspective of the query, which is “jaguar cats”. The visualization is poor as it is out of the scope of this work, but in the background all the data for interaction is there.
More results, experimentation, and evaluation on various scenarios are detailed
in Chapter 4.

### 3.5.2 Topic Profile

An interesting addition to topic mining research has been done in this work by introducing the concept of topic profiling. The idea is to drill down one more level to know the details of a topic. Some basic information can easily be mined with the proposed data and indexing model, which turns out be very useful for human understanding.

- **Age of Topic** is defined by the time difference between the last document added and the first time a document was added to a topic.

- **Topic Time Trend** is a graph showing the daily frequency of a given topic over all documents.

- **Periodicity of a Topic** is defined as how often a topic is being discussed. It can be measured by taking the median of the number of documents vs. the time graph.

- **Trending topic** is defined by taking the average frequency of a topic for a given time window. If the frequency of the topic goes above average, it is said to be trending.

- **Entity-Entity Graph** is an interaction of various entities inside a given set of documents grouped under a topic under consideration.
• **Type of topic** gives an annotation to a topic for either an entity or a string phrase and can add a lot to the context when forming the story.

## 3.6 Beyond Keywords: Sense Search

A query, which is a single or set of ordered keywords, may have a varied sense and context. Receiving a list of ranked web pages with mixed senses is not desirable. For example, the query “jaguar” might have the meaning “jaguar cars”, “mac os” or “jaguar cat”. Each sense can have its own deeper sense.

Let \( N \) be the number of documents being returned by a search engine for any given query \( q \). The problem statement is now to extract various senses of the query and present the overview of document grouping with understandable labels [40].

In the proposed framework, a knowledge graph from the documents is built. The knowledge graph contains an entity related to another entity through some topics or a topic linked to another topic. Instead of directly returning the flat list of documents, presenting the graphical view of topics linked to each other helps the user to visually explore and search for what he/she might want to search.

The query entered is consulted with the inverted index in the graph database, which returns the nodes and edges that contain the query word. Now for those nodes, the topic mining process is already performed in the background. With the measures defined above for topic mining, a knowledge graph of
the topic entity is built. This graph theoretically contains all of the various perspectives from which the document set can be seen, providing various facets from which to search. In an experiment performed for this work, web pages for the query “jaguar” were collected and all those documents were run through the framework. In the end the graph with different senses and their sub senses was mined. “Jaguar cars”, “jaguar cats”, “mac-os”, “guitars” were mined as four major contexts and inside each context there were other sub topics, providing a contextual search and explore capability to the user. A Snapshot\(^3\) of the knowledge graph created for another dataset, of webpages, used by the DIG model in [21], is shown in Figure 3.15. The detailed results are shown and discussed in Chapter 4.

### 3.7 Other Use Cases

In this section we briefly describe other possible use cases that can be implemented utilizing the generic nature of the proposed data and indexing model.

\(^3\)The visualization was not the main focus of this work. The framework needs a proper UI layer to visualize all the salient features of the knowledge graph which are not visible in the current version.
3.7.1 Query Expansion

Query expansion is now very common in search engines. Most query expansion algorithms take user query logs to match the closest query to the entered query. Depending only on the user log could be misleading. One would have noticed on search engines that some weird query is being suggested because it would have been entered by many users but actually in the results no relevant
result shows up. Therefore, a mix of query log based and document level query expansion is desirable.

Given the query log, this framework can easily parse all the patterns in the state machine and, next time a query is entered, it will be matched by the pattern matching mechanism and a possible expansion could be suggested. The other kind of expansion is document level expansion. In this, documents are read and the knowledge about the sentence and phrase structure is stored in the state machine graph. Now, when a query is being asked, the node will be consulted from the graph database’s indexing mechanism. For each node the edge list will be a candidate for possible query expansion. Since each edge is connected to another node and each node has its own importance value, the query expansion candidates can be ranked based on the importance and not just by frequency.

3.7.2 KeyPhrase Extraction

With the concept of node importance and phrase completeness, the phrase in the data model can easily be ranked and can be used to perform feature extraction tasks. Hence, a simplistic form of the framework can also just act as a feature selection process.
3.7.3 Vocabulary Tracker

In the applications where counting the words in a defined vocabulary is required, the proposed model can easily adapt to construct a state machine with the given set of keywords, and then parse the documents for all the occurrences of a word defined in the state machine.

This can also be used in frequency based weighted classifiers to determine the tone or sentiment of a given document. In each case the lookup vocabulary will change.

3.7.4 Summarization

With the identification of key phrases, the data model also maintains the sentence structure of the documents. For a given document, picking the top K sentences corresponding to important phrases, one can quickly generate the summary in readable form for a given document.

This way of summarizing is very simple but with further research it is expected that the framework will act as a promising candidate to adapt to perform advanced summarization.

3.7.5 Entity-Entity Interaction

For a given document corpus, having an interaction map for various entities with each other could be very useful in some applications such as twitter, blogs and other social networks. With little modification to existing data
models, we can generate a knowledge graph just for entities. In the feature extraction stage, we can just extract named entities and turn off the phrase extractor. Now the state graph will only be built with named entity phrases and various entities will be matched as documents are read. In the end, the topic discovery process will produce a knowledge graph containing only the entities with their corresponding importance and interaction level.

There could be many more text mining tasks which could have been possible to perform using the proposed framework with little modification by keeping the core concept. The tasks described in this section are not well researched and are not the state of the art in their domains. But what we wanted to show is that the framework is powerful and generic enough to perform various tasks without many modifications.

3.8 Concluding Remarks

Inspired by how a human extracts topics for a document, this framework has been designed to perform various text mining tasks, in a simple, understandable and linear way, with a focus on topic and story mining. To the best of our knowledge, this framework has been designed keeping in mind the state of the art in topic discovery and the industry expectations of the technology. The data model that is the core of this framework utilizes concepts from graph theory and finite state automata theory to build a knowledge graph for
a set of documents. This, in some sense, acts like a human brain which reads and remembers potential topics and decomposes the non-important ones in the sub conscious mind. The graph database provides a powerful backbone to the framework. The concept of topic discovery has been extended to story discovery, giving context to flat isolated topics. The idea of sense search, not just keyword search, has also been introduced.

The framework has been kept very simple with a focus on its practical usability. The framework can be easily extended to add more pieces for database storage, a pre-processing phase, a feature extraction phase and other use cases. A layer of professional GUI can enhance the usability and understanding of the discovered results. With that, it can be customized to perform a dedicated task with a proper visualization. Some possible use cases have also been suggested in the end of this chapter.

The framework has been applied on various kinds of data, such as news articles, RSS feeds, web pages, query returned documents, and user reviews. Results have also been compared with the industry standards. All the results and experimentation will be given in Chapter 4.
Chapter 4

Experimental Results

4.1 Introduction

To evaluate the performance of the proposed framework, this chapter is dedicated to discussing the results obtained by performing various experiments. The data are collected from various domains. In text mining, the majority of the results are subjective in nature. It needs to be understood that if humans cannot agree on one solution, a machine cannot be expected to provide one. In the next section, various experimental setups are explained and results are discussed.
4.2 Description of Datasets

The availability of grouped and labeled text data sets suitable for topic discovery and clustering is limited. The proposed framework is capable of performing various text mining tasks. For demonstration, text data sets are taken from sources such as news feeds, file systems, SharePoint and web pages. The “Data Source” module in the proposed framework acts as a buffer collector. The purpose is to provide a consistent data stream to the framework. The following datasets have been used in this work.

- **UofWData**: This data set is used in the benchmark DIG data model [21]. It is a collection of 314 web pages\(^1\) manually divided into 10 major clusters with a moderate degree of overlap.

- **UofAData**: This data set is used by Li et al. in [30] to automatically generate taxonomies. The collection contains 666 web documents\(^2\) collected for various queries, which have ambiguous meaning. For example, the query “jaguar” contains subtopics that have various senses such as “jaguar car”, “jaguar cat”, or “mac-os”.

- **RCV1-SubSet**: This data set is a subset of manually categorized news wire stories by Reuters Ltd. The subset contains 11839 documents and 118 categories. The complete details of the dataset are documented by

\(^1\)The data set can be downloaded at: http://pami.uwaterloo.ca/~hammouda/webdata/

\(^2\)The document set was requested directly from the authors.
Lewis et al. in [28]. The data set is best suited for text categorization, but it can still be used to evaluate the entropy of a grouping.

- **LiveRssFeeds**: This dataset is dynamic in nature. A reader module is made to read any RSS feed and provide a consistent stream of text to the framework. For this work, we use only news feeds from various news websites and generate a summarized topic overview of everyday news.

- **HotelReviews**: This dataset is used by Albornoz et al. in [12]. It is a collection of 1000 reviews extracted from www.booking.com. The reviews are tagged with a sentiment value. Although the data set is not suitable for clustering, it can be used to extract positive or negative topics discussed for a given hotel.

### 4.3 System Specifications

All the experiments have been performed on the following platform:

- **Machine Specifications**:
  
  **Model Name**: MacBook Pro, **Software**: OS X 10.8.2, **Processor Name**: Intel Core i7, **Processor Speed**: 2.8 GHz, **Number of Processors**: 1, **Total Number of Cores**: 4, **L2 Cache (per Core)**: 256 KB, **L3 Cache**: 8 MB, **Memory**: 16 GB.

---

3The corpus can be downloaded at: http://nlp.uned.es/~jcalbornoz/resources.html
• Development platform:

Eclipse Java EE IDE for Web Developers, Version: Juno Service Release 1, Java: v.1.7.0_15

4.4 Feature Extraction

The performance of Algorithm 1 to extract potential phrases, described in Chapter 3, is discussed in this section. Consider a piece of text as shown below [36].

<table>
<thead>
<tr>
<th>Input: Text Document</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Temporal Text Mining (TTM) is concerned with discovering temporal patterns in text information collected over time. Since most text information bears some time stamps, TTM has many applications in multiple domains, such as summarizing events in news articles and revealing research trends in scientific literature. In this paper, we study a particular TTM task – discovering and summarizing the evolutionary patterns of themes in a text stream. We define this new text mining problem and present general probabilistic methods for solving this problem through (1) discovering latent themes from text; (2) constructing an evolution graph of themes; and (3) analyzing life cycles of themes. Evaluation of the proposed methods on two different domains (i.e., news articles and literature) shows that the proposed methods can discover interesting evolutionary theme patterns effectively.”</td>
</tr>
</tbody>
</table>
Algorithm 1 generates a ranked list of phrases, with corresponding weights in the brackets.

<table>
<thead>
<tr>
<th>Output: Ranked List of Phrases</th>
</tr>
</thead>
<tbody>
<tr>
<td>discover interesting evolutionary theme patterns effectively(5.24), general probabilistic methods(3.6), analyzing life cycles(3.0), evolutionary patterns(3.0), revealing research trends(3.0), temporal text mining(2.83), text information bears(2.83), text information collected(2.83), discovering temporal patterns(2.7), text mining problem(2.5), discovering latent themes(2.27), evolution graph(2.25), text stream(2.25), different domains(2.0), many applications(2.0), scientific literature(1.75), summarizing events(1.75), time stamps(1.75), ttm task(1.66), literature(1.5), summarizing(1.5), ttm(1.33)</td>
</tr>
</tbody>
</table>

The results are easily understandable and the extracted phrases make sense to a human. The phrases have been extracted from a single document, unlike traditional feature extraction methods [4], which rely on the information from the whole corpus.

The behavior of the algorithm on varying numbers of documents is shown in Figure 4.1. The number of documents are plotted on the x-axis, and the time spent, in seconds, to extract phrases is plotted on the y-axis. As number of documents increases the curve tends to follow the linear trend. The approximate equation which fits the curve is $Y = 21.513X - 55$, which is linear. The average length of the documents given to the algorithm was 832
characters and the median document length was 540 characters.
In real life applications, where speed is a concern, this algorithm performs very fast. In our experiment, it was able to extract phrases from approximately 6000 documents in just 60 seconds. Since it is linear in time and there are no inter-dependencies between the documents, it can easily be parallelized to increase the throughput.
Once the phrases are extracted from the documents, a state machine graph is built. In the next section, statistical details of the graph construction are explained.

Figure 4.1: Feature Extraction: No. of documents vs. time(s)
4.5 Document Model Construction

The algorithms to construct the state machine data graph are explained in Chapter 3. Theoretically, it should be built in linear time as it reads the document and stores the information just once.

An experiment was conducted in which the time to build the graph was recorded vs. the number of documents read. Figure 4.2 shows the graphical relation.

![Graph Building Time: No. of documents vs. time(s)](image)

Figure 4.2: Graph Building Time: No. of documents vs. time(s)

It can be seen that, for a lesser number of documents, the time taken to build the graph is very short, almost on the order of seconds. For 1000 documents, it took only 13 seconds to build the graph. But as the size of the document set grows, the number of nodes and edges increases. The algorithm tends to follow its worst case run time, which is close to linear in this case. One can see that after 2000 documents, the curve is near to linear. The proposed data model is capable of reading and storing 10000 documents in just 120 seconds.
The performance time not only depends on the number of documents, but also on the number of phrase units in each document. The statistics shown in Figure 4.2 are generated using the RCV1-SubSet dataset. The average length of the document is 832 characters with maximum value of 13395 characters and median of 540 characters.

![Graph Growth Rate: No. of documents vs. Number of nodes](image)

**Figure 4.3: Graph Growth Rate: No. of documents vs. Number of nodes**

Apart from the construction time of the proposed data model, it is also important to analyze the growth rate of the graph. In order to be scalable, it is desired to have a controlled growth rate, not exponential. Figure 4.3 represents the growth rate of the graph with the number of documents on the x-axis vs. the number of state nodes on the y-axis. As can be seen, the growth of the graph tends to stabilize as the size of the document set increases. This behavior was expected from theory because as the document set increases, the vocabulary used in the documents tends to stabilize; therefore, fewer new nodes are created.
4.5.1 Comparison with DIG

The benchmark DIG document model is capable of processing 2000 average-sized news articles in as little as 44 seconds and moderate-sized web documents in a little over 5 minutes [21]. For fair evaluation, we used the same data set (RCV1-2340), which contains 2340 news articles extracted from Reuters, to evaluate the statistics of the proposed data model. The proposed model could read 2000 average-sized news articles in as little as 25 seconds. Moreover, to read 10000 articles, from a bigger subset of Reuters data set, it only took 120 seconds.

For the number of nodes generated, DIG performs better than the proposed model, but the difference is not very substantial. If DIG generated 29075 nodes for 1500 articles, our model generated 31141 states for the same 1500 articles. For the trade off between time taken and the number of nodes generated, the proposed model performs promising as compared to the DIG model. This behavior was theoretically expected because of the deterministic behavior of the proposed model, whereas in DIG, as the document set grows, the edge and sentence table information gets overwhelmed in a given node. Hence, lookup time is more than for the deterministic automata, where for a given input, there is only one output state in one move.
4.6 Topic Mining Experiments

As the document model is built, the implicit pattern matching mechanism matches and groups documents around the path of the graph. A careful selection of the most significant paths (nodes), in conjunction with other nodes, gives meaningful topic descriptions and stories. The underlying grouping of documents could help in the evaluation process. The focus of this work is on topic discovery, not on perfect grouping. Hence, a trade off between best grouping and satisfactory topics needs to be realized.

4.6.1 Evaluation Scheme

The problem of clustering and topic discovery is all about perspectives. Consider Figure 4.4; we have 3 objects, which can either be clustered by color as one perspective or shape as another perspective. Both of them are valid with respect to a given perspective.

In order to evaluate the quality of the topic mining results, some widely used evaluation metrics from the text mining literature have been adopted.

- **Purity**: The purity of the grouping, underlying every topic or story, is measured by using the distribution of the documents in the original groupings. If documents in the group belong to one major category, the purity is higher.

  The entropy measure [4, 21, 33] can be used to determine the purity of a cluster. The higher the purity of a cluster, the lower the entropy and
vice versa.

Let $Prob_{ij}$ be the probability that the documents in cluster$_j$ belong to class$_i$. The entropy, $E_j$, of cluster$_j$ with respect to class$_i$ is defined as:

$$E_j = - \sum_i (Prob_{ij} \times \log(Prob_{ij})) \quad (4.1)$$

If $N_j$ is the total number of documents in cluster$_j$ and $N$ is the total number of documents, then the overall entropy of all $m$ clusters is calculated as:

$$OverallEntropy = \sum_{j=1}^{m} \left( \frac{N_j}{N} \times E_j \right) \quad (4.2)$$
• **Precision and Recall** are two widely used measures in the Information Retrieval literature for validating the discovered groupings (Clusters) with the ground truth groupings (Classes).

If $N_i$ is the number of documents in ground truth class $i$, $N_j$ is the number of documents in cluster $j$, and $N_{ij}$ is the number of documents of class $i$ in cluster $j$, then

\[
P = Precision_{ij} = \left( \frac{N_{ij}}{N_j} \right) \tag{4.3}
\]

\[
R = Recall_{ij} = \left( \frac{N_{ij}}{N_i} \right) \tag{4.4}
\]

• **F-Measure** is the harmonic mean of precision and recall. The F-measure for class $i$ can be calculated as:

\[
F_{\text{measure}} = F_i = \left( \frac{2PR}{P+R} \right) \tag{4.5}
\]

The F-Measure guides how well the clustering is mapped to the known classes. The F-measure applies better in classification, but it can still be used to evaluate the probable mapping of a cluster to a class. For a cluster, its most probable class is the one which has the highest F-measure. The overall F-measure of all classes with respect to best mapped cluster is given by:
$OverallF = \left( \frac{\sum_i(N_i \times F_i)}{\sum_i|N_i|} \right)$ (4.6)

A higher F-Measure means the clusters are mapped to the original classes with higher accuracy.

- **Human Evaluation:** It is time consuming for humans to analyze text mining results; however, it is still considered as a benchmark in many applications, specifically in topic discovery. The discovered topics need to be validated by human experts. For a machine, it is hard to match the exact meaning of the topic with the ground truth. Therefore, human evaluation along with above described statistical measures are required in order to validate the topics. In this work it is being performed visually.

In this research, a mix of the measures described above is used to evaluate the results, because not every dataset comes with the ground truth grouping and labeling. Some datasets are used only to test the scalability, while some are used to perform specific tasks. The experiments performed on various datasets are discussed in the next sub-sections.

### 4.6.2 Experiments on UofWData

The *UofWData* is used in [21] to evaluate the performance of the benchmark DIG model. It is a collection of 314 web pages collected manually from the
University of Waterloo and Canadian websites, for 10 major topics described below, along with the number of documents contained in each.

<table>
<thead>
<tr>
<th>Ground Truth: Topic Label(No. of Documents inside)</th>
</tr>
</thead>
<tbody>
<tr>
<td>canadatra-transportation-roads(22), co-op(55), river-fishing(23), river-rafting(29), snowboarding-skiing(24), black-bear-attack(30), career-services(52), health-services(23), winter-canada(23), campus-network(32).</td>
</tr>
</tbody>
</table>

The average length of a document in the collection is 4696 characters, with median length of 2870, minimum length of 52, and maximum length of 55007 characters.

The data is used to discover topics and generate clusters by the approach proposed in this framework. The results are encouraging and motivating. The results have been compared with four benchmark algorithms in the literature:

- **Similarity Histogram based Incremental Clustering (SHC-DIG):** Uses DIG as the data model and calculates phrase based similarity between sets of documents [21] to perform clustering.

- **Hierarchical Agglomerative Clustering (HAC):** uses a traditional single-term, vector space model to find hierarchical clusters.

- **Single Pass Clustering (Single Pass)**

- **K-Nearest Neighbor Clustering (K-NN)**

The algorithms, mentioned above, are designed to perform only the clustering, not the labeling. However, in this framework, the topic discovery is performed
at the same time as clustering. Hence, the grouping of documents can still be evaluated by comparisons with the those generated using the above mentioned algorithms.

Table 4.1 compares the value of overall F-measure and Entropy of the clustering produced by the proposed model with other algorithms.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>F-Measure</th>
<th>Entropy</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHC-DIG</td>
<td>0.931</td>
<td>0.119</td>
</tr>
<tr>
<td>Proposed Framework</td>
<td>0.785</td>
<td>0.084</td>
</tr>
<tr>
<td>HAC</td>
<td>0.709</td>
<td>0.351</td>
</tr>
<tr>
<td>Single-Pass</td>
<td>0.427</td>
<td>0.613</td>
</tr>
<tr>
<td>k-NN</td>
<td>0.228</td>
<td>0.173</td>
</tr>
</tbody>
</table>

Table 4.1: UoWData:Comparisons with benchmark

In terms of F-Measure, the proposed model showed improvement of 10.7 % over HAC, 83.8 % over Single-Pass and 244 % over K-NN, which are based on single term vector space models, however, SHC-DIG performed better than the proposed model. But, if we look at the entropy level, the proposed model surpasses all other techniques with lowest entropy value of 0.084 (the lower, the better). Lower entropy value shows that the quality of the clusters produced is high.

The focus of this research was on producing high quality topics, not just grouping the documents. Hence, a compromise between the best grouping
and understandable labels is made; this is a reason for the low F-measure. Another reason for lower the F-Measure is poor recall values. In the data set, only 10 parent clusters are defined. There is no sub-topic information provided inside them. However, the proposed algorithm is capable of finding the sub-clusters within the larger cluster. The recall formulation maps the smaller sub-clusters to the big parent cluster, resulting in poor recall. On the other hand, precision does not have this mapping problem and proves to be a promising measure to evaluate the relevant grouping.

In order to understand the quality of the clusters produced by the proposed model, let us have a look at the precision graph. Figure 4.5 shows the precision trend of the clusters. The average precision is 0.80, with a maximum as 1.0. This indicates that the groups being made are relevant with respect to ground truth grouping.

Figure 4.5: UofWData: Clustering Precision
The homogeneity of all the clusters is shown in Figure 4.6. It can be seen that the maximum value of entropy is only 0.15, with the majority of values being zero. The lower the entropy, the better the grouping is.

![Figure 4.6: UofW: Clustering Entropy](image)

The total time taken to cluster 314 documents by the proposed model is around 24 seconds. This time also includes reading time, feature extraction time, graph building time and clustering time. Hence, for real time applications, a faster independent-clustering-only module can be designed to speed up the process.

The F-measure, precision, entropy and time taken suggest that the topics are discovered very quickly, are noise free, have promising quality, and are well mapped to the original clustering at the parent level. An important requirement of the whole process was to come up with meaningful topic
labels, yet with statistically valid grouping. Table 4.2 shows 10 parent topics originally present in the data and the topics/sub topics discovered by the proposed approach.

<table>
<thead>
<tr>
<th>Original Topics</th>
<th>Topics Discovered by the proposed approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>canada-transportation-roads</td>
<td>canada transportation act, transportation system</td>
</tr>
<tr>
<td>co-op</td>
<td>co-op education, work term, co-operative education, learning objectives, human resources</td>
</tr>
<tr>
<td>river-fishing</td>
<td>fishing trips, guided fishing, fly fishing</td>
</tr>
<tr>
<td>river-rafting</td>
<td>whitewater rafting, river rafting, day trips</td>
</tr>
<tr>
<td>snowboarding-skiing</td>
<td>skiing snowboarding, heli skiing, cold winters, great lakes, western canada</td>
</tr>
<tr>
<td>black-bear-attack</td>
<td>bear attack, bear country, black bears, grizzly bear, canadian rockies, great smoky</td>
</tr>
<tr>
<td>career-services</td>
<td>career services, job search, job interview, career resource center, work experience, human resources, full time</td>
</tr>
<tr>
<td>health-services</td>
<td>health services, long term</td>
</tr>
<tr>
<td>winter-canada</td>
<td>severe weather, heavy snowfall, environment canada, winter weather, weather conditions</td>
</tr>
</tbody>
</table>
It can be seen from Table 4.2 that the topics, when read in conjunction with each other, give an overall sense of the parent topic, while still keeping the detailed view of what is inside each topic.

Using the topic overlap measure, described in Chapter 3, a visualization of the various topics interacting with each other is shown in Figure 4.7. It should be noted that the visualization was not the focus of this research. In the back-end, there was more information calculated such as how connected a topic is to another topic, ranking topics by importance, the most connected topic in the graph, and the type of topic. With a better visualization, the back-end results could enhance the understandability of the topic connectedness. The idea of topic overlap helps to build the context around a flat topic scheme. A topic may have different meanings in different contexts, but when topics are read in conjunction with connected topics, one can make better sense of a topic and the point of discussion.
4.6.3 Experiments on UofAData

The *UofAData* is used in [30] to automatically mine the taxonomies in a collection of web articles. It is a collection of 666 web pages collected by querying the Google search engine. The retrieved web pages are grouped together with the query as the label. There are 5 major clusters, with the following labels and underlying number of documents in each of them.
As can be seen, queries are selected that have ambiguous meaning. The task is to apply the concept of topic discovery to see if it can identify various senses of each query, by analyzing the retrieved list of web pages. It is not purely a clustering or grouping task, but rather a document-centric query sense disambiguation task.

Consider a query “Jaguar” typed in Google, returning a mix of web-pages conveying different senses of the query. With Google Knowledge Graph,
Google now tries to suggest some sense of a search by mapping the query to possible knowledge entities in the Knowledge Graph. Figure 4.8 shows the query senses returned by Google.

This extra help from Google does not change the web pages returned; rather it helps the user to better navigate and narrow down the search to a specific need.

We ran all 666 web pages through our framework to discover various perspectives by which these documents are grouped. In the original paper, the authors used an approach similar to our concept of extracting the keywords from the documents, building the keyword communities and mapping documents on top of them. For evaluation, they used human experts to verify various topics and senses discovered by their algorithm.

We also ran the same data through our framework. Table 4.3 shows the topics and senses discovered for each query.

<table>
<thead>
<tr>
<th>Query</th>
<th>Topics/Senses discovered by our framework</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jaguar</td>
<td><strong>Animal:</strong> (cat family, jaguar animal, big cats, natural habitat, jaguar rescue center, endangered species, largest feline)</td>
</tr>
<tr>
<td></td>
<td><strong>Car:</strong> (jaguar xk, xj, xf, sports car, jaguar dealer, swallow sidecar company, ss jaguar, person: william lyons )</td>
</tr>
<tr>
<td></td>
<td><strong>Mac-OS:</strong> (mac os, operating system )</td>
</tr>
<tr>
<td>Topic</td>
<td>Concepts</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Guitar</strong></td>
<td>(bridge pickup, neck pickup, vintage style floating tremolo)</td>
</tr>
<tr>
<td><strong>Tiger</strong></td>
<td><strong>Golf:</strong> (Tiger Woods, pga tour, San Francisco)</td>
</tr>
<tr>
<td></td>
<td><strong>Algorithm:</strong> (tiger hash algorithm, hash function)</td>
</tr>
<tr>
<td></td>
<td><strong>Animal:</strong> (tiger, habitat, cat family, largest cat)</td>
</tr>
<tr>
<td><strong>AVP</strong></td>
<td><strong>Volleyball:</strong> (avp tour, beach volleyball)</td>
</tr>
<tr>
<td></td>
<td><strong>Antivirus:</strong> (anti virus, software)</td>
</tr>
<tr>
<td></td>
<td><strong>Product:</strong> (avon products, market cap, long term)</td>
</tr>
<tr>
<td></td>
<td><strong>Movie:</strong> (avp movie)</td>
</tr>
<tr>
<td></td>
<td><strong>Airport:</strong> (international airport, Scranton)</td>
</tr>
<tr>
<td><strong>Penguin</strong></td>
<td><strong>Algorithm:</strong> (google penguin, algorithm, update )</td>
</tr>
<tr>
<td></td>
<td><strong>Club:</strong> (club penguin, kids)</td>
</tr>
<tr>
<td></td>
<td><strong>Hockey:</strong> (hockey league, Pittsburgh, Stanley cup, wilkes-barre)</td>
</tr>
<tr>
<td><strong>Michael Jordan</strong></td>
<td><strong>Player:</strong> (greatest basketball player, Chicago bulls, NBA, game winning shot)</td>
</tr>
<tr>
<td></td>
<td><strong>Researcher:</strong> (California Berkeley, machine learning, computer science)</td>
</tr>
</tbody>
</table>

Table 4.3: Query Senses Discovered in UofAData

All the topics shown in Table 4.3 are discovered purely by our framework, without any human intervention. The time taken to read all 666 web pages of average length 6185 characters, identify key phrases, and discover meaningful
topics was approximately 50 seconds.

For a real time application, this framework has the potential to be incorporated in any search engine to show a cloud of various senses discovered for a given query by the user.

The entropy of groupings obtained, underlying each topic, are also compared with the results from the paper [30], which originally used this data set.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Entropy or Purity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proposed Framework</td>
<td>0.109</td>
</tr>
<tr>
<td>Li and Zaiane et al. approach</td>
<td>0.127</td>
</tr>
<tr>
<td>K-Means</td>
<td>0.279</td>
</tr>
</tbody>
</table>

Table 4.4: UofAData: Comparisons with benchmark

The topics generated by our approach are more pure than the approach adopted by Li et al. [30]. This suggests that the senses discovered by our approach are well separated from each other, hereby giving a clear understanding of the various perspectives discussed in a set of web pages returned for a specific query.

Towards **Meaning Search**: An interesting observation can be made from the obtained results. As we can, see for each, query we have several discovered topics, which co-occur with each other, giving a topic community. These communities give sense to the query search. Examples of some topic communities generated by our approach are shown in Figure 4.9.
For a given query “Jaguar”, with only the query auto fill option, the queries that contain some occurrence of word the “jaguar” will be suggested. However, with the concept of Knowledge Graph, senses such as “mac-os”, “sports car” or “cat” could be suggested even though they do not have the direct occurrence of the word “jaguar”.

The purpose of this framework was to show the potential it has to perform various text mining tasks. Clearly, it has the potential to go beyond keyword search to meaning search in a linear and efficient manner. With proper investigation, the knowledge graph can be further enhanced to contain information for more entities and use them as the demands of the query and search world grow.
4.6.4 Scalablity test with RCV1-SubSet

RCV1-Subset is a collection of 11839 news wire stories from Reuters Ltd. In this work, this data set has been used to test the scalability of the topic discovery process. The average size of the document in this collection is 1000 characters with a maximum of 15000 characters.

Figure 4.10 shows the time trend of the clustering and topic discovery process vs. the number of documents. The time taken also involves reading the documents, extracting phrases from them, and performing topic discovery using the data model proposed in this work.

![Clustering Time](image)

Figure 4.10: RCV1-Subset: Clustering Time Trend

As can be seen, it is difficult to extrapolate the overall trend of the whole process from the limited information available in the range of 100 to 1000 documents. As the number of documents increases, the actual time trend starts to appear, which is near to linear. From a practical perspective, to
cluster 2000 documents, our approach took just 14 seconds, whereas to find topics in 10000 documents, it took around 68 seconds.

### 4.7 Comparison with Industry Standards

Infomous Inc. [2] provides a product that takes information from the internet and converts the online content into a cloud of topics for easy exploring and browsing. Their tag line is, “We show what’s trending, you choose what’s relevant”. Figure 4.11 shows what Infomous does.

![Infomous takes information from the Internet to provide you with a cool new way to explore content online](image)

**Figure 4.11: What Infomous does? [2]**

With close investigation, we found out that Infomous is not doing any magic in discovering the topics. It is simply using the frequency and co-occurrence
of words to draw an impressive visualization. Sometimes a stop word and noise also shows up in the cloud. But, when a human reads it, he/she may overlooks it, because of the impressive visualization.

However, the background data for the Infomous cloud, and even more than that, can be very easily generated with our framework. Infomous can read any RSS feed and present a topic cloud of the content inside; we applied it to generate the topic cloud for the RSS feed of New Delhi news: http://ibnlive.in.com/ibnrss/rss/india/delhi.xml. We then made our framework process the same RSS feed.

Figure 4.12 shows the cloud generated by Infomous. Some major stories, as seen, are “Rape in Delhi”, “Protests”, “Activities”, “Anti-Sikh Riot case”, and “Coal Scam”.

Figure 4.12: Cloud Generated by Infomous [2]
Figure 4.13 shows the results generated by our framework.

![News Topics Generated by our framework](image)

As can be seen, the visualization is not as impressive as Infomous, but the value inside the data is much more. For example, in Infomous, words “Sheila” and “Dikshit” are just keywords and are connected by an edge. However, in our results, it has been identified that it is a person “Sheila Dikshit” and it is linked to another element “Delhi cm”. In reality, Mrs. Sheila Dikshit is the chief minister of Delhi, India. In the graph, she is also linked to other entities and phrases. There are many other such meaningful elements in our graph,
which makes better sense to humans.

Even though our results show, more or less, the same stories, our stories have more details and meaning than the ones generated by Infomous. In our work, a story is being viewed from various perspectives, rather than just keywords. With the amount of details and knowledge that could be extracted by our approach, in future work, with better visualization our work can definitely surpass the Infomous cloud.

The proposed model is not just limited to processing RSS Feeds; it can be applied to any other kind of text data coming from Twitter, blogs, news, emails or SharePoint documents to discover perspective rich stories to give a good insight into the content and topics discussed.

4.8 Hotel Reviews Vocabulary Cloud

For the sake of demonstration that the framework can be applied on various domains, 15 manually annotated positive reviews were taken from “HotelReviews” data to generate an Infomous style vocabulary graph to give the gist of all the reviews in one view. Figure 4.14 shows the vocabulary cloud.

When a user is looking online for hotels, he/she might not have time to read all the reviews, but if this quick graph is provided on the side, the user can quickly get an overview. In the graph, we can say that the “Hotel has great staff, good food, has a nice location, with clean and nice rooms, which have
an Eiffel Tower view”. A similar graph can be generated for negative reviews too.

4.9 Concluding Remarks

In this chapter, the proposed framework has been tested and evaluated using different kinds of text data. The theoretical expectations line up with the experimental results.

One of the major challenges in topic discovery research is to come up with
meaningful topic descriptions. The results obtained for various data sets suggest that the topics discovered in this work are meaningful to humans with promising F-measure and purity of the underlying grouping (or clustering). The proposed model performed better than the traditional bag of word (BoW) approaches such as $K$-NN, HAC or Single-Pass.

The framework is also compared to the benchmark data models in terms of data model growth rate and pattern matching capabilities. Our model proves to be a promising candidate for performing various kinds of text mining tasks, all encapsulated in one framework.

The framework is flexible enough to be applied and adapted to various needs of applications. It has been applied to discover clusters and topics, to disambiguate query senses, and to generate summarized overview of the content.

From the results, it is evident that the tasks are performed in a linear, simple, and memory efficient manner. With little modification, this framework can easily be plugged into real time applications to perform various text mining tasks all in one place quickly.

With the encouraging results and potential to apply to various applications, there is definitely room for possible of improvements in this framework. The next chapter describes some of them.
Chapter 5

Conclusion and Future Work

5.1 Conclusion

Inspired by how a human thinks of a text mining problem, the purpose of this work was to design a simple yet efficient framework for text mining which could perform various tasks efficiently and in linear run time. To the best of our knowledge, this framework has been designed keeping in mind the state of the art technologies and industry expectations.

A thorough literature survey was conducted in the areas of feature extraction, clustering, co-clustering, topic discovery, graph data models and pattern matching algorithms. Various parts and pieces are taken to make a one place framework for performing various text mining tasks. It has been shown in the experimentation section how this framework can be used to perform various tasks. The generic nature of the framework allows it to be adapted to other
applications.

The core of this framework is the underlying data model, which uses concepts from graph theory and the theory of automata. Various tests and experiments are performed to validate the proposed data model. The results obtained are promising and encouraging. The concept of building up context around a topic is also new in this work.

The theoretical expectations line up with the real results. The discovered topics are validated by human experts and the underlying groupings are validated with the benchmark validation measures and papers.

In comparisons with the benchmark approaches, our method produced high quality topics with promising F-Measure and entropy. The topics obtained for UofW [21] and UofA [30] datasets were easily understandable by humans. The concept of topic communities or stories helped in getting the essence of the document collection. Our approach is also scalable in nature; in the experiments to cluster 2000 documents, our approach took just 14 seconds, whereas to find topics in 10000 documents, it took around 68 seconds. We have also compared our work with the industry standard Infomous Inc. [2] on live News feeds to determine topics in real time. The results obtained are promising and encouraging.

The experimentation results also suggest that this work has the potential to be integrated in online search engines and other text repositories to discover various perspectives and stories hidden inside the documents.

As with every work, there is scope for improvement in this framework. The
next section describes some possible future directions for this work.

5.2 Future Work

- The current version extracts only phrases and entities; however, extracting part of speech tags could also help in getting meaningful topics in the end.

- Enhancing the topic labels with Wordnet would also be an interesting direction to pursue. Adding Wordnet can help to roll up the topics to a common subject matter. For example, the topics Football and Hockey should roll up to the common subject matter of Sports.

- Coming up with a different scheme to rank topics by their importance should also be investigated. Topics could also be sorted by a combined measure of importance and time to show current hot topics.

- It will be interesting to know the common way between VSM and phrase based models. An approach combining their strengths would help the community to adapt the traditional algorithms for new problems. It would help them to improve with real world demands, where topic descriptions are of top priority, not just the grouping of documents.

- As can be seen, the framework was used to perform various applications; an investigation of other possible applications would be an interesting
future task.

- Phrase based Classifier is another interesting direction to consider. Taking a set of labeled documents and building a phrase-based classifier would be very useful.

- Optimizing the growth of the state machine graph should definitely be investigated in order to deal with big data.

- One of the very important future directions is to integrate the data security model inside the proposed data model to protect the data of one user from being shown to another.

- A proper visualization layer is also required to easily navigate through the discovered stories and to appreciate the valuable knowledge discovered in the back end.
References


International Conference on Data Mining (Washington, DC, USA), ICDM ‘02, IEEE Computer Society, 2002, pp. 203–.


Vita

Ramanpreet Singh

Education:
1. GGSIP University, 2006-2010, Bachelor of Technology in Electronics and Communication Engineering.

Publications:

Journal Papers

Refereed Conference Papers