6.1 Introduction to Text Summarization

Text Summarization is condensing the source text into a shorter version preserving its information content and overall meaning. It is very difficult for human beings to manually summarize large documents of text. The Internet normally provides more information than is needed. Therefore, a twofold problem is encountered: searching for relevant documents through an overwhelming number of documents available, and absorbing a large quantity of relevant information.

The goal of automatic text summarization is condensing the source text into a shorter version. Summaries may be classified by any of the following criteria:

- **Detail**: Indicative/informative
- **Granularity**: specific events/overview
- **Technique**: Extraction/Abstraction
- **Content**: Generalized/Query-based

![Figure 6.1 A Summarization Machine](image)

An ideal summarization machine would look like the one shown in Figure 6.1. An indicative summary gives the main focus of the document and contains only a few lines whereas an informative summary is generally long and can be
read in place of the main document. The granularity decides the extent to which we want the summary to be broken into i.e. short, medium, detailed (specific event related or an overview) etc.

When the summary is the result of a query asked it becomes a query related otherwise it is a general summary. Topic-oriented summaries focus on a user's topic of interest, and extract the information in the text that is related to the specified topic. On the other hand, generic summaries try to cover as much of the information content as possible, preserving the general topical organization of the original text.

Text Summarization methods are more popularly classified into extractive and abstractive summarization. An extractive summarization method consists of selecting important sentences, paragraphs etc. from the original document and concatenating them into shorter form. The importance of sentences is decided based on statistical and linguistic features of sentences. An Abstractive summarization attempts to develop an understanding of the main concepts in a document and then express those concepts in clear natural language.

Text Summarization can be divided into the following areas:

- Selection based (tf-idf, ranking, etc.)
- Understanding based (syntactic analysis, semantic analysis)
- Information Extraction / Information Retrieval

The selection methods are more popular than the understanding based as the latter are connected to the Natural Language Processing (NLP). The extractive methods are based on tf-idf (term frequency-inverse document frequency), cluster based methods, the Latent Semantic Analysis (LSA) which is based on singular value decomposition or concept based summarization which is based on the vector space model. There are other methods also which are based on graphs, neural networks, fuzzy logic, regression, etc. Some of the above mentioned topics have been discussed earlier in the classification and clustering section.

The extraction based summarization methods studied are:

- Single document summarization
- Multi-document summarization
- Topic models
As part of further research in the field of abstraction, I have developed an algorithm based on the semantics i.e. a part-of-speech (POS) tagger. It is called the ‘Multi-Liaison Algorithm’ and it is explained in section 6.8.

For all types of summarization techniques the pre-processing steps generally remain same as discussed before. However there is a slight difference in text summarization. The details are given below.

6.2 Tokenization

Though tokenization is required to find the term frequencies, we store the sentences of the document separately and the weights assigned are to the sentences also. In some cases the position of the sentences is very important because the term weights depend on the position of the sentences in which the terms occur i.e. the title, the first paragraph, the last paragraph etc. These positions are given more weightage.

Removal of stop words and stemming remain the same.

6.2.1 Sentence Scoring

In extractive summarization it is important that from the document or set of documents we find out first which sentences are more important for the summary than the rest. This is possible only if some ranking / scoring is associated with them. There are four types of words which generally affect the sentence scores:

1. Cue words: These are the indicative words of the document which give some hint or analysis of the content like “summary”, “reflects”, “conclusion”, “purpose” etc. These types of words are to be given more weightage.

2. Content Words (keywords): These are generally the nouns in sentences. Generally sentences containing proper nouns are considered important. These can also be the words which are acronyms, capitalized or italicized.

3. Title words: If a document has a title, generally the words in the title represent the main concept on which the document is based, so these words are important and are given extra weightage.

4. Location: the location of the sentence is very important. The first line and the last paragraph are more or less very important for the summary.
Chapter 6: Text Summarization

The sentence scoring has been done as follows:
\[ S_i = w_1 \cdot C_i + w_2 \cdot K_i + w_3 \cdot T_i + w_4 \cdot L_i \]  \hspace{1cm} (6.1)

Where,
- \( S_i \) – score of sentence \( i \)
- \( C_i \) – score of sentence \( i \) based on cue words
- \( K_i \) – score of sentence \( i \) based on keywords
- \( T_i \) – score of sentence \( i \) based on title words
- \( L_i \) – score of sentence \( i \) based on its location
- \( w_1, w_2, w_3, w_4 \) – are the weights assigned

In short, for document summary, score of a sentence is dependent on the frequency of the words in that sentence, their related weightage as per the details given above and the sum of it.

6.3 Single document summarization

Whenever summarization is be done it is necessary to know to what length the main document should be summarized (size of summary as compared to size of the document). This is also known as the compression rate. For example a 10 sentence document when compressed by 10% results in a one line summary.

Once each sentence is scored those sentences are ranked based on the descending order of their scores. Then depending on the compression rate the top sentences are selected as part of the summary.

6.4 Multi-document summarization

When summary is required from multiple documents, it is necessary that the documents are related to each other as far as the main content topics are concerned. In case we need to summarize multiple documents which are of mixed types, the first step is to applying text clustering on them so as to form clusters of same types of documents. Once these clusters are formed, for each cluster a separate summary can be generated.

Since the individual summary is generated from multiple documents belonging to a cluster, there is always a possibility that similar sentences from different
documents selected and repeated in the final summary. To make sure that the inter-sentence similarity is low, the following formula can be applied:

$$Cosine (ti, tj) = \frac{\sum_{h=1}^{k} t_{ih} t_{jh}}{\sqrt{\sum_{h=1}^{k} t_{ih}^2 \sum_{h=1}^{k} t_{jh}^2}}$$

(6.2)

Where,

i, j – the ith and jth sentences

\(t_i, t_j\) – term frequencies of ith and jth sentences

Depending on the similarity measures and compression ratio, top ranking but non-overlapping sentences are selected from multiple documents. The limitation in this method is the sequence in which the sentences from different documents would be displayed. This can be handled by noting the location of the selected sentences in its respective document (starting, middle, and end) and try to output each sentence as per its location.

Purely extractive summaries often give better results compared to automatic abstractive summaries. This is due to the fact that the problems in abstractive summarization, such as semantic representation, inference and natural language generation, are relatively harder compared to a data-driven approach such as sentence extraction. In fact, truly abstractive summarization has not reached to a mature stage today. Existing abstractive summarizers often depend on an extractive preprocessing component. The output of the extractor is cut and pasted, or compressed to produce the abstract of the text.

Limitations of Extractive Methods are:

- Extracted sentences usually tend to be longer than average. Due to this, part of the segments that are not essential for summary also get included, consuming space.
- Important or relevant information is usually spread across sentences, and extractive summaries cannot capture this (unless the summary is long enough to hold all those sentences).
- Conflicting information may not be presented accurately.
Chapter 6: Text Summarization

6.5 Comparison of Text Summarization methods

There are a number of different methods that have been developed for Text summarization and the base of these methods is either related to statistics, mathematics or NLP. A comparative is given in Table 6.1.

<table>
<thead>
<tr>
<th>Main concept of the method</th>
<th>Working</th>
<th>Method type</th>
</tr>
</thead>
</table>
| Tf-idf based summary      | Based on simple heuristic features of the sentences:  
  - Position in the text  
  - The overall frequency of the words they contain  
  - Key phrases indicating the importance of the sentences  
  - A commonly used measure to assess the importance of the words in a sentence is the inverse document frequency | Extractive Method |
| Centroid-based summarization, a well-known method for judging sentence centrality and then selecting the sentences | The measures used are:  
  - Degree  
  - LexRank with threshold  
  - Continuous LexRank inspired from the prestige concept in social networks. | Extractive method |
| Lexical chains            |  
  - Basically lexical chains exploit the cohesion among an arbitrary number of related words  
  - Lexical chains can be computed in a source document by grouping (chaining) sets of words that are semantically related  
  - Identities, synonyms, and hypernyms / hyponyms (which together define a tree of “is a” relations between words) are the relations among words that might cause them to be grouped into the same lexical chain. | Abstractive method |
| A graph based representation |  
  - A document cluster where vertices represent the sentences and edges are defined in terms of the similarity relation between pairs of sentences  
  - This representation enables us to make use of several centrality heuristics defined on graphs | A combination of Extractive and Abstractive methods |
Maximum Marginal Relevance Multi Document (MMR-MD) summarization

- (MMR-MD) summarization is a purely extractive summarization method that is based on Maximal Marginal Relevance concept proposed for information retrieval
- It aims at having high relevance of the summary to the query or the document topic, while keeping redundancy in the summary low
- It can accommodate a number of criteria for sentence selection such as content words, chronological order, query/topic similarity, anti-redundancy and pronoun penalty

Cluster based methods

- Documents are usually written such that they address different topics one after the other in an organized manner
- They are normally broken up explicitly or implicitly into sections i.e. themes
- If the document collection for which summary is being produced is of totally different topics, document clustering becomes almost essential to generate a meaningful summary.

Latent Semantic Indexing

- This method uses the concept of the Singular Value Decomposition (SVD)
- The process starts with the creation of a terms by sentences matrix
- After applying the SVD as discussed before, the sentences with the highest index i.e. best sentences describing the salient topics of the text are selected

The above is not an exhaustive list of methods but covers the most popular and commonly used ones. Variants of the above methods are also available. Some very good Text Summarization tools have been developed. They are:

**MEAD**

MEAD is a publicly available toolkit for multi-lingual summarization and evaluation. The toolkit implements multiple summarization algorithms (at arbitrary compression rates) such as position-based, Centroid, TF*IDF, and query-based methods. Methods for evaluating the quality of the summaries
include co-selection (precision/recall, kappa, and relative utility) and content-based measures (cosine, word overlap, bigram overlap).

MEAD v1.0 and v2.0 were developed at the University of Michigan in 2000 and early 2001. MEAD v3.01 – v3.06 were written in the summer of 2001, an eight-week summer workshop on Text Summarization was held at Johns Hopkins University. More details are available at:
http://www.clsp.jhu.edu/ws2001/groups/asmd.

**SUMMARIST**
This tool provides the abstracts and the extracts for English, Indonesian, Arabic, Spanish, Japanese etc. documents. It combines the symbolic world knowledge i.e. dictionaries like the WordNet and other lexicons as well as robust NLP processing techniques to generate the summaries. SUMMARIST is based on the following equation:
Summarization = topic identification + interpretation + generation
This tool is developed by the Natural Language Group at the University of Southern California.

**ROUGE (Recall-Oriented Understudy for Gisting Evaluation)**
ROUGE is a set of metrics and a software package used for evaluating automatic summarization and machine translation software in natural language processing. The metrics compare an automatically produced summary or translation against a reference or a set of references (human-produced) summary or translation. Lin and Hovy’s designed this package. For the inception of ROUGE, please refer Lin & Hovy's HLT-NAACL 2003 (Lin and Hovy 2003) paper.

**SUMMONS**
McKeown and Radev (1995) presented a system called SUMMONS which summarizes related news articles. The SUMMONS is a genre specific system which operates in the terrorist domain. The goal of the system is to generate fluent, variable–length summaries. SUMMONS is based on traditional language generation architecture and has two main modules for doing content planning and linguistic operations. The content planner consists of paragraph
planner and combiner. The linguistic component is made up of a lexical chooser, ontologizer and a sentence generator.

6.6 Sample Output of Text Summarizer

A sample output of the MEAD summarizer is given below. The input are three text files, the summary generated is compressed by first 10% and then 25%. File1 is of 1319, File2 of 1307 and File3 of 1067 characters. Though the system is for Dutch, its gives a good output for English language.

Input to the summarizer were three files:

TEXT 1
I have assessed Ami in the lab assignments where I found that she has the potential of a very good programmer. She was also effectively involved in organizing university level technical event “Dwianki” where I was mentor for the same. I observed that she had the quality to work independently as well as in a group with equal ease. Her dedication to work for the best is substantiated by her excellent grades in all the courses I have handled. Considering her overall academic distinctions and achievements, I place her among top 5% of the students associated with me in recent years. I am happy to see that she has decided to take her education to a higher level by pursuing a Masters degree at your Graduate School. She is a person with pleasing demeanor and has good communication skills. She always had the passion to learn new things and I am sure that she will continue to explore new horizons with the same zeal. I am confident that she will not only continue to be a promising and competitive student but would also be capable of efficiently discharging her roles as research/graduate assistant. I strongly recommend her for higher studies with deserving financial assistance. I feel that her academic proficiency and potential for research make her one of the truly outstanding candidates I have come across.

TEXT 2
In my course of interaction with him I have come to know Deepal as an exceptionally sincere and assiduous student. He has good understanding of theoretical aspects on one hand and its application to practical problems on
other hand. His lab work is consistent and he has performed exceedingly well in all his university lab examinations. This confirmed his capability of grasping the core concepts of the subjects and clear understanding of the basic principles. Deepal has mature personality and his attitude towards peer is co-operative and congenial. I have seen him produce very good results on complex projects that required great attention to details without compromising on the quality. 'Sparsh - Multi-touch Interaction System', final year project consisted of real time video processing and developing application to take the advantage of multi-touch sensing, which awarded Best project in two National level competitions. His keen analytic mind, systematic work habits, determination to pursue any chosen assignment to a successful conclusion provide an excellent blend of qualities required for successful pursuit of a graduate program. I am confident that given an opportunity, he will excel in his field of study. I therefore strongly recommend him for admission in your esteemed institution.

TEXT 3
As an Associate Professor of the Computer Science Department of The M S University, I can describe he has very good logical ability and intuitive thinking which makes him a very talented student. Moreover, he focuses more on conceptual learning and has the habit of thinking out of the box. He has excellent communication skills and always solves the problems assigned to him with a systematic and analytic approach. His diligence and dedication complement his good qualities. The quality I like the most about Nishant is that he is extremely humble and down to earth. He is an innovative thinker and is really good at writing. He has a good grasping power and his approach towards his work is always positive. He always has the attitude of learning from his mistakes. Apart from his studies, he actively participated in extracurricular activities and was the Training and Placement Coordinator of his batch. He was responsible for the campus recruitment of the Computer Science Department. Moreover, he always displayed good team spirit and was very supportive.
Output summary: (25%)

[1] I have assessed Ami in the lab assignments where I found that she has the potential of a very good programmer.

[2] I am happy to see that she has decided to take her education to a higher level by pursuing a Masters degree at your Graduate School.

[3] She always had the passion to learn new things and I am sure that she will continue to explore new horizons with the same zeal.

[4] In my course of interaction with him I have come to know Deepal as an exceptionally sincere and assiduous student.

[5] He has good understanding of theoretical aspects on one hand and its application to practical problems on other hand.

[6] As an Associate Professor of the Computer Science Department of The M S University, I can describe he has very good logical ability and intuitive thinking which makes him a very talented student.

[7] Moreover, he focuses more on conceptual learning and has the habit of thinking out of the box.

[8] He has excellent communication skills and always solves the problems assigned to him with a systematic and analytic approach.

Output summary: (10%)

[1] I have assessed Ami in the lab assignments where I found that she has the potential of a very good programmer.

[2] In my course of interaction with him I have come to know Deepal as an exceptionally sincere and assiduous student.

[3] As an Associate Professor of the Computer Science Department of The M S University, I can describe he has very good logical ability and intuitive thinking which makes him a very talented student.

[4] He has excellent communication skills and always solves the problems assigned to him with a systematic and analytic approach.

The screen shots are given below:
Automatic Multi-document Summarization Demo for Dutch

The automatic summarization demo can sum up to three documents simultaneously. The summarizer works as follows. It starts with recognizing the separate sentences in the texts. Next it computes for each sentence an importance score. The system sorts the sentences on their importance and extracts the most important 25% (by default) as summary.

Type or paste pieces of Dutch text in the boxes. (The maximum input size of this demo is 10000 characters.) You can get the latest news here or you can go to this page in which we already filled in the fields for you. If you have any questions you can post a message here.

Chapter 6: Text Summarization

Figure 6.2 Screen shots of MEAD Summarizer
6.7 Topic Model

6.7.1 Introduction to Topic Model

A topic model is a type of statistical model for discovering the abstract "topics" that occur in a collection of documents. An early topic model was probabilistic latent semantic indexing (PLSI), created by Thomas Hofmann in 1999. Latent Dirichlet allocation (LDA) is perhaps the most common topic model currently in use. The topic model is a statistical language model that relates words and documents through topics. It is based on idea that documents are made up of a mixture of topics, where topics are distributions over words. The Table 6.2 contains the conceptual comparison of various topic models.

With the increasing availability of other large, heterogeneous data collections, topic models have been adapted to model data from fields as diverse as computer vision, finance, bioinformatics, cognitive science, music, and the social sciences. While the underlying models are often extremely similar, these communities use topic models in different ways in order to achieve different goals.

Table 6.2 Conceptual comparison of various topic models

<table>
<thead>
<tr>
<th>Model Name</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latent Semantic Analysis (LSA)</td>
<td>+ significant compression over simple tf-idf representation + Original high-dimensional vectors are sparse but the corresponding low-dimensional latent vectors will not be sparse, makes it possible to compute meaningful association between pairs of doc even no common term + can capture some aspects of basic linguistic notion such as polysemy and synonymy.</td>
<td>- Not capable of handling dynamic document collection - No generative model - No statistical standard methods - Output is not interpretable</td>
</tr>
<tr>
<td>k-means (cluster-model/mixture of unigrams)</td>
<td>+posses fully generative semantics</td>
<td>- document is considered to fall in a single cluster i.e. topic</td>
</tr>
</tbody>
</table>
### Probabilistic Latent Semantic Analysis (pLSA)

| + consider the document to be made up of more than one topic |
| - No probabilistic model at the doc level (i.e. no generative model-how the document can be generated) which leads to very serious problem of: number of parameters grows linearly with the size of corpus. |
| - How to assign the probability to document outside the training set is not defined. |
| - No assumption about how the mixture weight θ is generated. |

### Latent Dirichlet Allocation (LDA)

| + provides a proper generative model |
| + robust and versatile |
| + domain knowledge is not required |
| + as an unsupervised learning technique, human-intensive task of finding labeled examples for training set is completely eliminated. |
| - although, time and space complexity grows linearly with the number of documents, computations are only practical for modest-sized collections of up to hundreds of thousands of documents. |

I have studied the topic model which is also a part of Text Mining in general and text summarization in particular. An approach called the Gibbs Sampling, a Markov Chain Monte Carlo method, is highly attractive because it is simple, fast and has very few adjustable parameters.

As part of the research, I have tried to derive a scalable algorithm which leads to reduction in the space complexity of the original Gibbs sampling for topic model. The concept used to reduce the space complexity is partitioning the dataset into smaller sets and then executing the algorithm for each partition. This reduces the space requirement without any impact on the time complexity. The enhanced Gibbs sampling algorithm has been implemented and experimented on four different datasets.

This work has been published in ‘International Journal of Computer Information Systems’ by Silicon Valley Publishers (UK), ISSN: 2229-5208, October 2011 issue and is available at:  
http://www.svpublishers.co.uk/#/ijcis-oct-2011/4557969965. Before actually implementing the algorithm it was necessary to understand the LDA, the Gibbs Sampling and then propose a new approach. The step-wise and precise study and implementation is as given in the next section.

#### 6.7.2 Latent Dirichlet Allocation

Latent Dirichlet allocation (LDA) is a generative probabilistic model of a corpus. The basic idea is that documents are represented as random mixtures
over latent topics, where each topic is characterized by a distribution over words. The graphical model representation of LDA is shown in Figure 6.3.

LDA assumes the following generative process for each document \( w \) in a corpus \( D \):

1. Choose \( N \sim \text{Poisson}() \).
2. Choose \( \alpha \sim \text{Dir}(\alpha) \).
3. For each of the \( N \) words \( w_n \):
   (a) Choose a topic \( z_n \sim \text{Multinomial}(\theta) \).
   (b) Choose a word \( w_n \) from \( p(w_n \mid z_n, \beta) \), a multinomial probability conditioned on the topic \( z_n \).

There are three levels to the LDA representation. The parameters and their significance are:

1. \( \alpha \) and \( \beta \) are corpus level and are sampled once in the process of generating a corpus.
2. \( \theta \) is document-level variable, sampled once per document.
3. \( z \) and \( w \) are word-level variables and are sampled once for each word in each document.

To compute the posterior distribution of the hidden nodes for a given document i.e. the inference to implement the LDA is given by:

\[
p(\theta, z \mid w, \alpha, \beta) = \frac{p(\theta, z, w \mid \alpha, \beta)}{p(w \mid \alpha, \beta)} \quad (6.3)
\]

The distribution is difficult to be estimated because of the denominator which is a normalizing constant.
The key idea behind the LDA model for text data is to assume that the words in each document were generated by a mixture of topics, where a topic is represented as a multinomial probability distribution over words. The mixing coefficients for each document and the word topic distributions are unobserved (hidden) and are learned from data using unsupervised learning methods. Blei et al. introduced the LDA model within a general Bayesian framework and developed a variational algorithm for learning the model from data. Griffiths and Steyvers subsequently proposed a learning algorithm based on collapsed Gibbs sampling. Both the variational and Gibbs sampling approaches have their advantages: the variational approach is arguably faster computationally, but the Gibbs sampling approach is in principal more accurate since it asymptotically approaches the correct distribution.

6.7.3 Gibbs Sampling

Introduction

Gibbs sampling is an example of a Markov chain Monte Carlo algorithm. The algorithm is named after the physicist J. W. Gibbs, in reference to an analogy between the sampling algorithm and statistical physics. The algorithm was described by brothers Stuart and Donald Geman in 1984, some eight decades after the passing of Gibbs. As mentioned before Griffiths and Steyvers proposed the collapsed Gibbs sampling.

The Smoothed LDA

Before discussing Gibbs sampling it is necessary to understand how the LDA is smoothed\(^1\) because of the problem with the original one. One problem that might arise with the original LDA model as shown in Figure 6.3 is that, the new document outside of training set is likely to contain words that did not appear in any of the documents in a training corpus, and zero probability would be assigned such words. To cope with the situation, the 'smoothed' model is shown in Figure 6.4. The strategy used is, not to estimate the model parameters explicitly, but instead considering the posterior distribution over the assignments of words to topics, \(P(z|w)\). The estimates of \(\theta\) and \(\Phi\) are then

\(^1\) The detailed explanation of the smoothed LDA and the equations in given in the bibliography – [43] to [54].
obtained by examining this posterior distribution. Evaluating $P(z|w)$ requires solving a problem that has been studied in detail in Bayesian statistics and statistical physics, computing a probability distribution over a large discrete space.

Here, $\alpha$ and $\beta$ are hyper parameters, specifying the nature of the priors on $\theta$ and $\Phi$. Although these hyper parameters could be vector-valued, for the purposes of this model we assume symmetric Dirichlet priors, with $\alpha$ and $\beta$ each having a single value. These priors are conjugate to the multinomial distributions $\theta$ and $\Phi$, allowing us to compute the joint distribution $P(w, z)$ by integrating out $\theta$ and $\Phi$.

![Graphical model representation of smoothed LDA](image)

**Figure 6.4** Graphical model representation of smoothed LDA

### 6.7.4 The Gibbs Algorithm for LDA

After applying a number of steps to the equation 6.3, the conditional distribution as mentioned by Griffiths et al. is:

$$p(z_n = t | z_n, w) \propto \frac{(\beta + c_{wt}^{-n}) (\alpha + c_{td}^{-n})}{(c_t^{-n} + W\beta)(c_d^{-n} + T\alpha)} \tag{6.4}$$

The different terminology used in the equation 6.4 is given in Tab. 6.3.

---

Table 6.3 Terms and their meanings for equation 6.4

<table>
<thead>
<tr>
<th>Term</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_{wt}^{-n}$</td>
<td>Number of instances of word w assigned to topic t, not including current one</td>
</tr>
<tr>
<td>$C_{t}^{-n}$</td>
<td>Total number of words assigned to topic t, not including current one</td>
</tr>
<tr>
<td>$C_{td}^{-n}$</td>
<td>Number of words assigned to topic t in document d, not including current one</td>
</tr>
<tr>
<td>$C_{d}^{-n}$</td>
<td>Total number of words in document d not including current one</td>
</tr>
</tbody>
</table>

Having obtained the full conditional distribution, the Gibbs Sampling algorithm is then straightforward. The $z_n$ variables are initialized to values in $\{1, 2 \ldots T\}$, determining the initial state of the Markov chain. The chain is then run for a number of iterations, each time finding a new state by sampling each $z_n$ from the distribution specified by the equation 6.4. After enough iterations for the chain to approach the target distribution, the samples are taken after an appropriate lag to ensure that their autocorrelation is low. The algorithm is presented in Figure 6.5.

With a set of samples from the posterior distribution $P(z \mid w)$, statistics that are independent of the content of individual topics can be computed by integrating across the full set of samples. For any single sample we can estimate $\Phi$ and $\Theta$ from the value $z$ by:

\[
\Phi'_{w} = \frac{\beta + C_{wt}^{-n}}{C_{t}^{-n} + \alpha \beta} \tag{6.5}
\]

\[
\Theta'_{d} = \frac{\alpha + C_{td}^{-n}}{C_{d}^{-n} + \alpha} \tag{6.6}
\]

These values correspond to the predictive distributions over new words $w$ and new topics $z$ conditioned on $w$ and $z$. The algorithm for Gibbs sampling LDA is
shown in Figure 6.5. The dimensions required in this algorithm are shown in Tab. 6.4 and the details of the arrays required are shown in Tab. 6.5.

Table 6.4 Dimensions required in Gibbs Algorithm

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D$</td>
<td>Number of documents in corpus</td>
</tr>
<tr>
<td>$W$</td>
<td>Number of words in vocabulary</td>
</tr>
<tr>
<td>$N$</td>
<td>Total number of words in corpus</td>
</tr>
<tr>
<td>$T$</td>
<td>Number of topics</td>
</tr>
<tr>
<td>$ITER$</td>
<td>Number of iterations of Gibbs sampler</td>
</tr>
</tbody>
</table>

Table 6.5 Arrays used in Gibbs Algorithm

<table>
<thead>
<tr>
<th>Array</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wid(N)</td>
<td>Word ID of $n^{th}$ word</td>
</tr>
<tr>
<td>did(N)</td>
<td>Document ID of $n^{th}$ word</td>
</tr>
<tr>
<td>z(N)</td>
<td>Topic assignment to $n^{th}$ word</td>
</tr>
<tr>
<td>Cwt(W,T)</td>
<td>Count of word $w$ in topic $t$</td>
</tr>
<tr>
<td>Ctd(T,D)</td>
<td>Count of topic $t$ in document $d$</td>
</tr>
<tr>
<td>Ct(T)</td>
<td>Count of topic $t$</td>
</tr>
</tbody>
</table>
6.7.5 Analysis of Gibbs Algorithm

The time and space complexity of the Gibbs sampling algorithm as shown in Figure 6.5 is:

**Time Complexity** ~ $O(\text{ITER} \times N \times T)$

**Space Complexity** ~ $O(3N + (D + W)T)$

To understand the limitations of the existing algorithm, consider a million-document corpus with the following size parameters:

$D = 10^6$

$W = 10^4$

$N = 10^9$
Chapter 6: Text Summarization

For this corpus, it would be reasonable to run with $T = 10^3$ topics and $\text{ITER} = 10^3$ iterations. Using the space complexity equation as given above, the required memory would be,

\[(3 \times 10^9 + (10^6 + 10^4)) \times 10^3 = 4 \text{ Giga Bytes}\]

This memory requirement is beyond most desktop computers and this makes Gibbs sampled topic model computation impractical for many purposes. As observed from the space complexity equation, the memory requirement increases because of $N$ – the total number of words in a corpus which is getting multiplied three times.

To reduce this space complexity problem, I have proposed the Enhanced Gibbs sampling algorithm.

### 6.8 The Enhanced Gibbs sampling algorithm

To reduce the space requirement of the original Gibbs algorithm, I have applied the concept of partitioning the word set $N$ and then executing the algorithm instead of loading the whole word set in a single run. With this we can achieve the reduction in space requirement as the size of $N$ now reduces without any impact on the time complexity.

After each run on a partition the result is stored in separate variables and there is absolutely no need to merge the results of each partition. The variables are treated as global variables for all partitions.

Suppose we consider three partitions of the original word set $N$. The space complexity becomes:

Space Complexity $\sim O \left(3 \times \frac{N}{P} + (D + W) T\right)$

Where $P$ is the total number of partitions,

$\sim O \left(3 \times \frac{N}{3} + (D + W) T\right)$

$\sim O \left(N + (D + W) T\right)$

The space requirement reduces considerably. Meanwhile the time complexity becomes:

Time Complexity $\sim O \left(\text{ITER} \times \frac{N}{P} \times T \times P\right)$

$\sim O \left(\text{ITER} \times N \times T\right)$

The time complexity does not change since the algorithm is executed as many times as the number of partitions but for a smaller word set each time.
The proposed algorithm

The enhanced algorithm would require the following steps for execution:

- Read each document, perform tokenization, remove stop words, and apply case folding.
- Generate document-word matrix.
- Generate the vocabulary of the unique words in the collection.
- From the document-word matrix, generate the sparse arrays containing the vocabulary index and document index of each word.
- Apply the Enhanced Gibbs Sampling algorithm to extract the topic from the collection.
- Output the result.

The algorithm is as shown in Figure 6.6.
### Chapter 6: Text Summarization

#### 6.8.1 Implementation of the Enhanced Gibbs Sampling Algorithm

This algorithm was implemented and tested on four datasets by varying the parameter values. It was implemented using MATLAB 7.0.1.

---

**Input:** document-word index, vocabulary-word index, vocabulary, parameters value.

**Output:** topic wise word distribution

**Procedure:** as described below

//initialization of Markov chain initial state

```plaintext
for all partition p ∈ [1, P] do
  for all words of the current partition p, n ∈ [1, N/P] do
    sample topic index z(n) ~ Mult(1/T)
    // increment the count variables
    Cwt(wid(n),t) ++ , Ctd(t,did(n))++, Ct(t) ++ ;
  end for
end for
```

// run the chain over the burn-in period, and check for the convergence.
// Generally for the fixed number of iteration and then take the samples at // appropriate lag.

```plaintext
for all partition p ∈ [1, P] do
  for iteration i ∈ [1, ITER] do
    for all words of the current partition p, n ∈ [1, N/P] do
      topic = z(n)
      // decrement all the count variables, as not to include the // current assignment
      Cwt(wid(n),t) -- , Ctd(t,did(n))-- , Ct(t) -- ;
      for each topic t ∈ [1, T] do
        P(t) = (Cwt(wid(n),t) + β)(Ctd(t,did(n)) + α ) / (Ct(t) + W β))
      end for
      sample topic t from P(t)
      z(i) = t
      // increment all the count variables to consider this new // topic assignment
      Cwt(wid(n),t) ++ , Ctd(t,did(n))++, Ct(t) ++ ;
    end for
  end for
end for
```

---

Figure 6.6 The Enhanced Gibbs Sampling Algorithm
The Datasets
To extract the topics we require a text dataset that is rich in different topics. There are large number of textual datasets available which can be most suitable for this type of implementation such as news articles, emails, literature, research papers and abstracts, technical reports. The datasets that we used were:

1. The Cite Seer collection of scientific literature abstracts
2. The NIPS dataset of research papers
3. The Times Magazine articles
4. The Tehelka Magazine articles

The result after the preprocessing is completed on the four datasets is shown in Tab. 6.6. This output is now used for the next step i.e. applying the Enhanced Gibbs sampling algorithm with different partitions.

Table 6.6 Output after pre-processing

<table>
<thead>
<tr>
<th>Parameter Values</th>
<th>Cite Seer</th>
<th>NIPS</th>
<th>Times Magazine</th>
<th>Tehelka Magazine</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of Total Words(N)</td>
<td>8320</td>
<td>51515</td>
<td>29601</td>
<td>17184</td>
</tr>
<tr>
<td>No. Unique Words(W)</td>
<td>683</td>
<td>1485</td>
<td>3820</td>
<td>1772</td>
</tr>
<tr>
<td>No. of Documents (D)</td>
<td>474</td>
<td>90</td>
<td>420</td>
<td>125</td>
</tr>
<tr>
<td>Time Taken in Seconds</td>
<td>120.656</td>
<td>661.532</td>
<td>410.359</td>
<td>244.86</td>
</tr>
</tbody>
</table>

6.8.2 Output and Comparison of the Enhanced Algorithm
This is the second phase i.e. applying both the Gibbs sampling and the Enhanced Gibbs sampling algorithms once the preprocessing is completed. A number of successive iterations are made through the topic assignment done by random sampling over the dataset. The proposed method does the same but instead of in a single step over the whole dataset, the dataset is divided into successive partitions and the algorithm is applied for each partition.

The output of both the algorithms with their comparisons is shown in Tab. 6.7 and Tab. 6.8. The algorithms were implemented on all the datasets with varying parameter values. I have displayed only two outputs related to the Cite Seer dataset in this section. Each dataset displayed similar results and
there was a considerable reduction in the space complexity when the Enhanced Gibbs sampling was used.

**Table 6.7 Output and comparison of both algorithms**

<table>
<thead>
<tr>
<th>Name of Arrays required by the algorithm</th>
<th>Original Algorithm</th>
<th>Proposed Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No Partition</td>
<td>Partition = 2</td>
</tr>
<tr>
<td><strong>Size</strong></td>
<td><strong>Bytes</strong></td>
<td><strong>Size</strong></td>
</tr>
<tr>
<td>ct (1,T)</td>
<td>1 x 30</td>
<td>240</td>
</tr>
<tr>
<td>ctd (T,D)</td>
<td>30x474</td>
<td>113760</td>
</tr>
<tr>
<td>cwt (W,T)</td>
<td>683x30</td>
<td>163920</td>
</tr>
<tr>
<td>did (1, N)</td>
<td>1x 8320</td>
<td>66560</td>
</tr>
<tr>
<td>wid (1, N)</td>
<td>1x 8320</td>
<td>66560</td>
</tr>
<tr>
<td>z (1, N)</td>
<td>1x 8320</td>
<td>66560</td>
</tr>
<tr>
<td><strong>Total Bytes</strong></td>
<td>477600</td>
<td></td>
</tr>
<tr>
<td><strong>Time Taken (secs)</strong></td>
<td>36.438</td>
<td></td>
</tr>
</tbody>
</table>

In Tab. 6.7 the values for the parameters are: \( T = 30, \) \( \text{ITER} = 1000, \) \( \alpha = 1.0 \) and \( \beta = .01 \) whereas in Tab. 6.8 the result with varying parameter values and partition values is displayed.

**Table 6.8 Summary of comparison of both algorithms**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>( T = 30 )</th>
<th>( \text{ITER} = 1000 )</th>
<th>( \alpha = 1 )</th>
<th>( \beta = 0.01 )</th>
<th>( T = 10 )</th>
<th>( \text{ITER} = 1000 )</th>
<th>( \alpha = 0.05 )</th>
<th>( \beta = 0.01 )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Time (sec)</strong></td>
<td><strong>Space (Bytes)</strong></td>
<td><strong>Time (sec)</strong></td>
<td><strong>Space (Bytes)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No Partition</td>
<td>36.438</td>
<td>477600</td>
<td>20.516</td>
<td>292320</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Partition = 2</td>
<td>36.063</td>
<td>377760</td>
<td>20.344</td>
<td>192480</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Partition = 3</td>
<td>35.734</td>
<td>344488</td>
<td>20.078</td>
<td>159208</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Partition = 4</td>
<td>35.64</td>
<td>327840</td>
<td>20.094</td>
<td>142560</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

As can be seen from the observations shown, space complexity reduces significantly whereas the time complexity reduces marginally.
6.8.3 Conclusion and future enhancements of Topic Model

The topic model is a statistical language model that relates words and documents through topics. It is based on the idea that documents are made up of a mixture of topics, where topics are distributions over words. Gibbs sampling for implementing LDA has been a very popular model for topic models as compared to alternative methods such as variational Bayes and expectation propagation. Gibbs Sampling, a Markov Chain Monte Carlo method, is highly attractive because it is simple, fast and has very few adjustable parameters.

While the time and space complexity of the topic model scales linearly with the number of documents in a collection, computations are only practical for modest-sized collections of up to hundreds of thousands of documents. In this paper we have proposed an enhanced Gibbs sampled topic model algorithm which scales better than the original as the space complexity gets considerably reduced.

There are number of extensions possible with the topic models, such as author-topic models, author-role-topic models, topic models for images, hidden Markov topic models. Parallel topic models are also an emerging area of interest. The future work will be concentrating on any such extension of the topic model.

6.9 The Multi-Liaison Algorithm

6.9.1 Introduction of the proposed algorithm

The Multi-Liaison algorithm is useful for extracting multiple connections or links between subject and object from natural language input (English), which can have one or more than one subject, predicate and object. The parse tree visualization and the dependencies generated from the Stanford Parser are used to extract this information from the given sentence. Using the dependencies I have generated an output which displays which subject is related to which object and the connecting predicate. Finding the subjects and objects helps in determining the entities involved and the predicates determine the relationship that exists between the subject and the object. The
subjects can either be nouns or even pronouns. Moreover, one subject can be related to multiple objects and vice-versa.

I have named this algorithm ‘The Multi-Liaison Algorithm’ since the liaison between the subjects and objects would be displayed. The word ‘liaison’ has been used since the relationship and association between the subjects and predicates are displayed. This output would be useful for natural language processing (NLP), information retrieval, information extraction and also abstractive text summarization.

This algorithm has been published in the ‘International Journal of Advanced Computer Science and Applications (IJACSA)’ by The Science and Information (SAI) Organization, ISSN: 2156-5570 (Online) & ISSN: 2158-107X (Print), Volume 2 Issue 5, 2011. It is available online at: http://thesai.org/Publication/Archives/Volume2No5.aspx.

6.9.2 The Stanford Parser

The Stanford Parser is a probabilistic parser which uses the knowledge of language gained from hand-parsed sentences to try to produce the most likely analysis of new sentences. This package is a Java implementation of probabilistic natural language parsers.

The Stanford dependencies provide a representation of grammatical relations between words in a sentence for any user who wants to extract textual relationships. The dependency obtained from Stanford parser can be mapped directly to graphical representation in which words in a sentence are nodes in graph and grammatical relationships are edge labels. This has been used to extract the relation between multiple subjects and objects when the sentence to be parsed is a little complicated. Stanford dependencies (SD) are triplets: name of the relation, governor and dependent.

6.9.3 The Parse Tree and Dependencies

The parse tree generated by the Stanford Parser is represented by three divisions: A sentence (S) having a noun phrase (NP), a verbal phrase (VP) and the full stop (.). The root of the tree is S.

The Stanford typed dependencies representation was designed to provide a simple description of the grammatical relationships in a sentence that can
easily be understood. The current representation contains approximately 52 grammatical relations. The dependencies are all binary relations. The definitions make use of the Penn Treebank part-of-speech (POS) tags and phrasal labels.

To find the multiple subjects in a sentence our algorithm searches the NP sub tree. The predicate is found in the VP sub tree and the objects are found in three different sub trees, all siblings of the VP sub tree containing the predicate. The sub trees are: PP (prepositional phrase), NP (noun phrase) and ADJP (adjective phrase).

6.9.4 The Multi-Liaison Algorithm details

To execute this algorithm, first we start with parsing a sentence by the Stanford parser and storing the result in some intermediate file so that it can be taken as input for this algorithm. The triplet extraction algorithm\(^3\) has also been considered before finding the liaisons.

The application was written in JAVA using Net Beans IDE 6.5 RC2. It parsed a single sentence of 12 words in 8.35 seconds and displayed the output as shown in the examples below. This algorithm works equally well with simple as well as complex sentences and the output is very clear and precise.

As shown in Figure 6.7, the Multi-Liaison Algorithm takes as input the POS of each word, the parse tree and the typed dependencies [9]. Two functions are then called, the first is the GET_TRIPLETS and the second is the GET_RELATIONSHIP.

---

\(^3\) Delia Rusu, Lorand Dali, Blaz Fortuna, Marko Grobelnik, Dunja Mladenic, “Triplet extraction from sentences” in Artificial Intelligence Laboratory, Jožef Stefan Institute, Slovenia, Nov. 7, 2008.
Figure 6.7 The Multi-Liaison Algorithm

As shown in Figure 6.8, the GET_TRIPLETS function takes as input the Stanford Parse Tree and by considering the nodes under the NP sub tree and the VP sub tree, finds all the subjects, objects and predicates. The GET_RELATIONSHIP finds and displays the relationships between the subjects and objects. The algorithm is displayed in Figure 6.9.
Function: GET_TRIPLET (Output_Str)
Returns: Multiple subjects, objects and predicates
[Read level 1 of Parse Tree – refer Figure 2]
If tree contains ‘NP’ or ‘NNP’ then
  Function GET_SUBJECT (NP sub tree)
Else
  Return error message
If tree contains ‘VP’ then
  Function GET_PREDICATE (VP sub tree)
  Function GET_OBJECT (VP sub tree)
Else
  Return error message

Function: GET_SUBJECT (NP sub tree)
Returns: Subject(s) and adjective(s)
For (all nodes of NP sub tree) do
  If NP sub tree contains ‘NN’ or ‘NNP’ or ‘NNS’ then
    Store POS as a subject
  If NP sub tree contains ‘JJ’ then
    Store POS as an adjective
Return the subject(s) and adjective(s)

Function: GET_PREDICATE (VP sub tree)
Returns: Predicate(s)
For (all nodes of VP sub tree) do
  If VP sub tree contains ‘VB?’ then
    Store POS as a predicate
  Else
    Return error message
Return the predicate(s)

Function: GET_OBJECT (VP sub tree)
Returns: Object(s)
For (all nodes of VP sub tree) do
  If VP sub tree contains ‘NP’ then
    For (all nodes of VP_NP sub tree) do
      If VP_NP sub tree contains ‘NP’ or ‘NN’ then
        Store POS as an object
      Else
        Return error message
    Else
      Return error message
  Else
    Return error message
Return the object(s)
Chapter 6: Text Summarization

6.9.5 Output of the Multi-Liaison Algorithm

As per the algorithm discussed above, the output is shown below. In the first example, the outputs of the Stanford parse as well as the output of the Multi-Liaison both are displayed including the parse tree. In subsequent examples the parse tree is not displayed but the tagging, dependencies and the Multi-Liaison output is displayed. Figure 6.10 displays the parse tree.
Figure 6.10 The Stanford Parse Tree

The old beggar ran after the man who was wearing a black coat.
Example 1: The old beggar ran after the rich man who was wearing a black coat

The Stanford Parser output:

Tagging:
The/DT old/JJ beggar/NN ran/VBD after/IN the/DT rich/JJ man/NN who/WP was/VBD wearing/VBG a/DT black/JJ coat/NN

Parse Tree:
(ROOT
  (S
    (NP (DT The) (JJ old) (NN beggar))
    (VP (VBD ran)
      (PP (IN after)
        (NP
          (NP (DT the) (JJ rich) (NN man))
          (SBAR
            (WHNP (WP who))
            (S
              (VP (VBD was)
                (VP (VBG wearing)
                  (NP (DT a) (JJ black) (NN coat))))))))))

Typed Dependencies:
det(beggar-3, The-1)
amod(beggar-3, old-2)
nsbj(ran-4, beggar-3)
det(man-8, the-6)
amod(man-8, rich-7)
prep_after(ran-4, man-8)
nsbj(wearing-11, man-8)
aux(wearing-11, was-10)
rcmod(man-8, was-10)
det(coat-14, a-12)
amod(coat-14, black-13)
dobj(wearing-11, coat-14)

The Multi-Liaison Output:
Subject: 1
    NN beggar
Predicate: 3
    VBD ran
    VBD was
    VBG wearing
Object: 2
    NN man    JJ rich
    NN coat    JJ black

Relationship:
    beggar - ran - man
    man - wearing - coat

Figure 6.11 Example 1
As shown above, the Multi-Liaison Algorithm displays the relationship between the subject and object (beggar and man) as well as the relationship between the two objects (man and coat).

Example 2: The dog and the cat ran after the mouse and the mongoose

**Tagging:**
The/DT dog/NN and/CC the/DT cat/NN ran/VBD after/IN the/DT mouse/NN and/CC the/DT mongoose/NN

**Typed Dependencies:**
det(dog-2, The-1)
nsubj(ran-6, dog-2)
det(cat-5, the-4)
conj_and(dog-2, cat-5)
nsubj(ran-6, cat-5)
det(mouse-9, the-8)
prep_after(ran-6, mouse-9)
det(mongoose-12, the-11)
prep_after(ran-6, mongoose-12)
conj_and(mouse-9, mongoose-12)

**The Multi-Liaison Output:**
Subject: 2
NN dog
NN cat
Predicate: 1
VBD ran
Object: 2
NN mouse
NN mongoose

**Relationship:**
dog - ran - mouse - mongoose
cat - ran - mouse - mongoose

Figure 6.12 Example 2
**Example 3:** Jack and I visited the zoo with our children

I have also considered pronoun as a subject and therefore have got the relationship with 2 subjects in terms of noun and pronoun.

**Tagging:**
Jack/NNP and/CC I/PRP visited/VBD the/DT zoo/NN with/IN our/PRP$ children/NNS

**Typed Dependencies:**
nsubj(visited-4, Jack-1)
conj_and(Jack-1, I-3)
nsubj(visited-4, I-3)
det(zoo-6, the-5)
dobj(visited-4, zoo-6)
poss(children-9, our-8)
prep_with(visited-4, children-9)

**The Multi-Liaison Output:**
Subject: 2
NNP Jack
PRP I
Predicate: 1
VBD visited
Object: 2
NN zoo
NNS children
PRP$ our

**Relationship:**
Jack - visited - zoo - children
I - visited - zoo - children

All the three examples shown in the figures above have different number of subjects and objects and the relationship between them is also not similar. The Multi-Liaison Algorithm output in this way can be very useful for Text Mining applications where a variety of sentences are to be mined.
6.9.6 Conclusion and future enhancements

The proposed algorithm which displays the relationships between subjects and objects in sentences where there are multiple subjects and objects. The Stanford parser output was used to generate this result. This algorithm would be usable not only by Text Mining experts and computational linguists but also by the computer science community more generally and by all sorts of professionals like biologists, medical researchers, political scientists, business and market analysts, etc. In fact it would be easy for users not necessarily versed in linguistics to see how to use and to get value from the simple relationship that is displayed so effectively.
Chapter 7: Future Enhancements

The enhancements in Text Mining have already been discussed in the related chapters. However one article\(^1\) interested me as it is very much related to what exactly Text Mining is supposed to do.

During a series of hearings, the U.S. Senate Select Committee on Intelligence showed that prior to September 11, 2001, the American intelligence community had collected a significant amount of data about the men who attacked the World Trade Center and the Pentagon. The various intelligence agencies were simply unable to connect the dots. In his report, Richard C. Shelby, then vice chairman of the committee, stressed that agencies need powerful new tools to analyze the huge volumes of information they bring in.

Text-mining software is one of the front-line tools that the government is now using to tease out valuable connections. These specialized search engines can quickly sift through mountains of unstructured text—anything that's not carefully arranged in a database or spreadsheet—and pull out the meaningful stuff. They can infer relationships within data that are not stated explicitly. It is something we do all the time automatically but is enormously complicated for computers. "We bridge the gap between information and action," says Barak Pridor, CEO of ClearForest, a text-mining company.

The result of years of research at facilities such as Bell Labs and the Palo Alto Research Center, Text Mining applications have long been used in business. But more government agencies, including the Defense Intelligence Agency, the Department of Homeland Security, and the FBI, are using them to evaluate the multitude of e-mail messages, phone call transcripts, memos, foreign news stories, and other pieces of intelligence data these agencies collect each day.

Software from companies such as Autonomy, ClearForest, and Inxight Software can locate words and phrases the same way an ordinary search engine does. But that's just the beginning. Such applications are clever enough to run conceptual searches, locating, say, all the phone numbers and

---

\(^1\) http://www.pcmag.com/article2/, Cade Metz, ‘Uncovering telltale patterns’
place names buried in a collection of intelligence communiqués. More impressive, the software can identify relationships, patterns, and trends involving words, phrases, numbers, and other data. Using statistical and mathematical analysis, the programs can sift through thousands of documents and determine how certain words relate to each other. If a news story says that "Zacarias Moussaoui was a follower of the Islamic cleric Abu Qatada while living in London," a Text Mining application can identify Moussaoui and Qatada as people, identify London as a place, and determine the relationship among the three.

In theory, a human analyst could pick up those connections easily, but manually sifting through the enormous volumes of information is often impractical. Fortunately, Text Mining applications can deal with these and other similar functions.