6-2014

Scalable Multi-document Summarization Using Natural Language Processing

Bhargav Prabhala

Scalable Multi-Document Summarization Using Natural Language Processing

by

Bhargav Prabhala

A Thesis Submitted
in
Partial Fulfillment of the
Requirements for the Degree of
Master of Science
in
Computer Science

Supervised by

Dr. Rajendra K.Raj

Department of Computer Science

B. Thomas Golisano College of Computing and Information Sciences
Rochester Institute of Technology
Rochester, New York

June 2014

The thesis “Scalable Multi-Document Summarization Using Natural Language Processing” by Bhargav Prabhala has been examined and approved by the following Examination Committee:

Dr. Rajendra K.Raj
Professor
Thesis Committee Chair

Dr. Xumin Liu
Assistant Professor

Dr. Leon Reznik
Professor

Dedication

To my family for allowing me to pursue my dream...

Acknowledgments

I would like to thank my committee chair Dr. Rajendra K. Raj, for guiding me all through when I was stumbling in the dark.

Abstract

Scalable Multi-Document Summarization Using Natural Language Processing

Bhargav Prabhala

Supervising Professor: Dr. Rajendra K.Raj

In this age of the Internet, Natural Language Processing (NLP) techniques are the key sources for providing information required by users. However, with the extensive usage of available data, a secondary level of wrappers that interact with NLP tools have become necessary. These tools must extract a concise summary from the primary data set retrieved. The main reson for using text summarization techniques is to obtain this secondary level of information. Text summarization using NLP techniques is an interesting area of research with various implications for information retrieval.

This report deals with the use of Latent Semantic Analysis (LSA) for generic text summarization and compares it with other models available. It proposes text summarization using LSA in conjunction with open-source NLP frameworks such as Mahout and Lucene. The LSA algorithm can be scaled to multiple large-sized documents using these frameworks. The performance of this algorithm is then compared with other models commonly used for summarization and Recall-Oriented Understudy of Gisting Evaluation (ROUGE) scores. This project implements a text summarization framework, which uses available open-source tools and cloud resources to summarize documents from many languages such as, in the case of this study, English and Hindi

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Automatic text summarization is a complex research area in which considerable amount of work has been done by researchers. With the amount of information available on the Internet, the ability to create a summarized presentation of the information has become essential. Many summarization methods have been suggested in the past and have worked efficiently on single documents [13] or small sets of articles on specific topics [11]. Many browser applications and search engines such as Google, Bing, and Yahoo provide brief descriptions of results using these methods.

Multi-document summarization (MDS) has gained the attention of researchers in recent years due to the challenges it presents in providing well-summarized reports. The issues faced by MDS are the size constraints of documents, limited memory in computing resources, and redundancy of similar sentences in multiple documents. This report resolves these issues in MDS by using Natural Language Processing (NLP) techniques and open-source machine learning frameworks such as Mahout [20] and Lucene [10].

1.1 Background

NLP is a developing area in computational linguistics that uses many machine learning algorithms to evaluate, analyze, and replicate textual information. NLP is a means of simplifying human–computer interactions through learning mechanisms such as pattern recognition, parts-of-speech (POS) tagging, and textual summarization. NLP plays a key

part in allowing machines to understand and interact with humans. IBM's Watson, a system designed for providing answers to specified questions, uses DeepQA [7] software for question analysis, decomposition, hypothesis generation filtering, and the synthesis of answers to questions raised in native languages. Text summarization, the goal of which is to produce an abridged version of a text that is important or relevant to a user, is an important step in question answering.

1.1.1 What is text summarization?

A good summary should be concise and relevant to the original context while incurring minimal information loss. There are various types of summaries, as follows:

- Extraction vs. Abstraction
- Generalized vs. Query-model

Based on how a summary is created, it can be categorized into an abstractive or extractive summary. With the extractive approach, key sentences or phrases are selected from the original document to reflect the theme. The output generated conveys the essence of the actual text using the extracted phrases. In the abstractive approach, the input document is analyzed and the generated summary reflects the sentiment of the original text but is represented in different words.

If a summary tries to obtain a high-level understanding of the input document and caters to all key points in the document, it is considered a generic summary. If a summary is generated based on a query by the user and is relevant only to a specific topic, it is considered a query-based summarization.

Sentence and sentiment analysis is still being developed, and hence, abstractive summary generation using machine learning is difficult. Many attempts have been made by researchers to improve abstractive summaries using POS tagging, WordNet, Named-Entity Recognition, and other NLP techniques. The first step toward an abstractive summary is

to obtain a well-formed extractive summary. In this report, we worked towards obtaining well-defined extractive generic summaries.

1.1.2 Multi-Document summarization

MDS is a summary of a collection of articles on a specific topic. The Text Analysis Conference (TAC) conducts an evaluation of summarization models presented by researchers each year. A model should be able to minimize the redundancy of content, and the compression ratio should be less than 10 percent to qualify for TAC evaluation. Complexity in the sentences extracted using MDS in each document is also necessary for TAC evaluation. When dealing with multiple articles on the same topic, there may be some overlap in sentences from the various articles. For example, headlines from two different papers on a Russian space mission reported the following:

"Russian Soyuz rocket starts mission to space station with 3-person international crew onboard." - Washington Post

"Soyuz taking Russian-US-Japanese team to international space station." - Reuters

When these two articles are summarized, both the sentences have high term frequency and sentence similarity, so both could end up in the summary if the learning algorithm does not include the proper constraints. In MDS, limiting the size of a summary is also a tedious task; in two articles on a topic, there could be different opinions, both of which have to be captured in the summary, and this would only increase the file size. In this report, we addressed the aforementioned issues.

Most of the summarization mechanisms generate output in four stages: pre-processing, evaluation, information selection, and output generation (see figure 1.1). The pre-processing stage involves removing meta-data, titles, and figures from the original corpora for further analysis and the evaluation of information. The evaluation stage involves the analysis of datasets using machine learning algorithms to obtain the information required for the next steps. Sentences are reviewed and selected using sentence clustering algorithms in the

information selection stage. Finally, the selected sentences are put together for output generation.

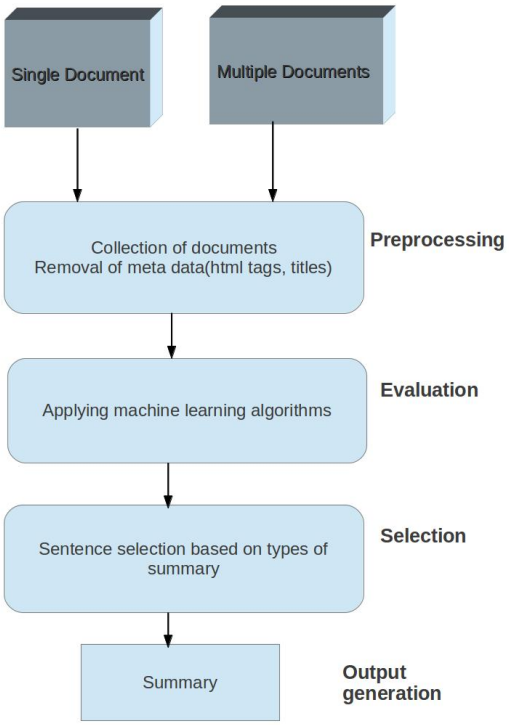

Figure 1.1: Stages of the text summarization system.

1.1.3 Multilingual summarization

Due to technological advancements, books and literature from various languages around the world have been digitized. Even though the literature is available in digital libraries to audiences around the globe, accessiblity to these documents is limited. End users may desire certain information about a foreign book before having to translate it into a language known to them. Multilingual MDS is very useful in these type of situations. Multilingual summarization will take a very similar approach to summaries, normal MDS when extracting summaries but adds a few additional steps in the preprocessing stage. The additional

steps in the preprocessing stage involve language/encoding detection, lemmatization, stemming, and indexing. We will discuss each of these steps further in the sections on the design and implementation of summarization methods for documents.

1.2 Related Work

Since the early 1960s, several summarization methods have been proposed by researchers. The initial works primarily focused on extracting sentences based on prominent terms, sentence length, and random selection. The most prominent study was conducted by H.P. Luhn using term frequencies and word collections from *The Automatic Creation of Literature Abstracts* [13]. The research aimed to obtain generic abstracts for several research papers. This approach was able to handle only single documents with less than 4000 words total. Another important approach in the early stages was proposed by Edmundson [6] using term frequencies and emphasizing the location of the sentences. Sentences at the beginning and end were given priority over other sentences. In the recent past, several methods have been proposed based on statistical, graph based, and machine learning approaches.

1.2.1 Statistical approach

In the statistical approach, features in sentences are selected based on concepts such as co-occurrences of words and classification. Summarist [12], for example, is a system proposed by Hovy and Lin and is based on a statistical approach for sentence extraction from single documents. It defines the optimal position policy by selecting words based on a certain distribution and extracts sentences containing the filtered words. Nomoto [19] proposed using bayesian classification in text summarization; the system was defined for single document summarization (SDS) in Japanese and, in some cases, evaluated more accurately than other systems. There are also papers [5] that used lexical chains and bayesian method for sentence extraction. These approaches used supervised algorithms, which had to be trained in a specific domain to obtain good summary results. This approach was not efficient for new corpora from a different domain.

1.2.2 Graph-based approach

In the graph-based approach, each node represents a sentence from the text, and nodes are connected to each other using edges. Nodes are connected to each other only when there are common terms between two nodes and the consine similarity between them is above a certain threshold. From the graph based approach, topic-driven summarization can be done by obtaining a sub-graph that is similar to the topic. For a generic summary, the most connected node from each sub-graph is selected for the summary. This approach was used in ranking web pages by Google's PageRank [21], which serves to index and search web pages. Summarization systems like *TextRank [16], LexRank [22], and Hypersum [24]* use this approach for text. These systems obtain good summaries but get complicated when introduced to MDS.

1.2.3 Machine learning (ML) approach

Summarization techniques using machine learning algorithms combined with NLP have been more widely used in recent years. Machine learning algorithms such as Naive-Bayes, special clustering, classification methods, decision trees, and Markov models are being used in the creation of summaries. Some of the noteworthy ML approaches includes using the Hidden Markov Models (HMM) [4] and Naive-Bayes [18]. These approaches work well with SDS but do not scale well to multi-documents. Learning algorithms such as Latent Semantic Analysis (LSA) and Latent Dirichlet Allocation (LDA) [2] work well with single documents and have been highly recommended by TAC. LSA will be explained in detail in the next section.

LSA in summarization

LSA is the most prominent algebric learning algorithm used for Information Retrieval (IR) from textual data. LSA is widely used in various applications for the dimension reduction of large multi-dimensional data. Singular Value Decomposition (SVD) is the most commonly used learning algorithm for information retrieval. It is known by different names in different

fields: Principal Component Analysis (PCA) in signal processing, KarhunenLove (KL) Transform in image processing, and LSA in textual processing. LSA works very well in single document summarization [23] and has been deemed effective by TAC. Because LSA is an unsupervised learning algorithm, it can be used across different domains without any corpora training. LSA obtains summaries in three stages: *Input matrix, singular value decomposition, and sentence selection*.

Input matrix(A): In this stage, the original documents are transformed into a matrix that can be processed in the SVD step. The input dataset is fit into a vector space model, where each unique word in a sentence is represented by the number of times it occurs in the document. The document in this model is a bag of words from which the sets of vectors are obtained and transformed into a matrix. For example, consider the following dataset:

"The woman brought the kid"
"The woman took the kid to park"
"The kid came to the park"

In the above example, the dataset is an extract of a passage that is represented in different ways. The set of vectors representing the words in the passage and their positions in a matrix form is shown below.

	Sentence1	*Sentence2*	*Sentence3*
the	2	3	2
woman	1	1	0
brought	1	0	0
kid	1	1	1
took	0	1	0
to	0	1	1
park	0	1	1
came	0	0	1

The input matrix for summarization constitutes an identity matrix representing the number of documents and word frequency (W) matrix shown above. The identity matrix for these three documents is shown here:

$$D = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

To multiply the

$$3x3$$

identity matrix, term frequency is transposed to match the number of columns in the matrix. The input matrix for decomposition is W^T:

$$A \leftarrow W^T = \begin{bmatrix} 2 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 3 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 2 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}$$

The matrix representation of the words in the dataset can be subjected to various information retrieval algorithms to extract valuable data. Transformation of a document involves many NLP methods including segmentation, stemming, word filtering, and sentence removal. Values in the matrix signify the importance of terms in a document and are calculated using different weighting metrics. The selection of a weighting metric can affect the end results. Some of the weighting metrics are term frequency (tf), term frequency-inverse document frequency (tf-idf), log entropy, and binary representation (0, 1). All these steps are explained in design implementation section.

Singular value decomposition, is a statistical tool for dimension reduction and feature selection. SVD factorizes a given input matrix into three matrices, as shown above (Figure 1.2).

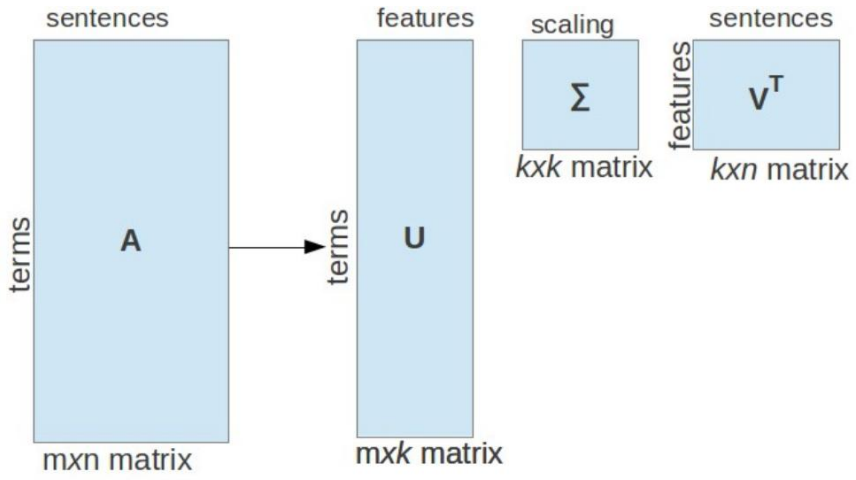

Figure 1.2: Singular value decomposition

$$A = U\Sigma V^T \tag{1.1}$$

A: Input matrix (m*n)

U: Orthogonal left singular matrix (m*k)

Σ: Scalar values matrix (k*k)

V^T: Orthogonal right singular matrix (k*n)

The input matrix comprises values that are defined by the weighting metric and the number of times a term appears in a sentence. Term frequency per sentence is usually zero or one if the input matrix is a large (m x n) matrix with rows defined by terms and columns by sentences. After the decomposition of the input matrix, two orthogonal matrices and one diagonal matrix were obtained. SVD is similar to eigen value decomposition, the only difference being that SVD can be applied on rectangular matrices, but eigen decomposition is strictly used for square matrices.

The values of each matrix can be obtained from the original input matrix.

- The left singular matrix, U, is comprised of eigen vectors of AA^T
- The diagonal matrix, Σ, is comprised of eigen values from $\sqrt{(AA^T)(A^TA)}$

- The right singular matrix, V^T, is comprised of eigen vectors of $A^T A$

If we were to consider previous example,

$$U = \begin{bmatrix} 0.46 & -0.73 & -0.51 \\ 0.77 & -0.04 & -0.64 \\ -0.45 & 0.68 & -0.58 \end{bmatrix} \tag{1.2}$$

$$\Sigma = \begin{bmatrix} 5.03 & 0 & 0 \\ 0 & 1.56 & 0 \\ 0 & 0 & 1.09 \end{bmatrix} \tag{1.3}$$

$$V^T = \begin{bmatrix} -0.82 & -0.24 & -0.09 & -0.35 & -0.14 & -0.25 & -0.24 & -0.10 \\ 0.10 & 0.47 & 0.49 & 0.06 & -0.02 & -0.42 & -0.43 & -0.40 \\ 0.01 & 0.22 & -0.41 & -0.31 & 0.63 & 0.10 & 0.10 & -0.53 \end{bmatrix} \tag{1.4}$$

The V^T matrix, which provides insight into sentences and features, is used to determine which sentences have the relevant information required.

The matrix U provides a relationship between terms and features. Any information related to terms can be extracted from the U matrix. For information related to sentences and features, right matrix V^T can be used. Reduction in dimension is based on the number of features (k) required. SVD works well in dimension reduction and text summarization, but it has a few disadvantages. SVD is a time consuming process, and it has to be re-worked every time there is any change to the original matrix. In addition, it cannot identify homonyms and words with different meanings. These issues can cause problems for large text analysis and summarization.

In the sentence selection stage, different algorithms can be used on the right singular matrix(V^T) obtained from the SVD on input matrix. V^T is comprised of *features x sentences*, where *features* are represented by values in rows and *sentences* by columns. In his paper [9], Gong discusses usage of sentences with highest value for each feature. Steinberger *et al.* [23] used top k sentences with a sum of feature values. Clustering algorithms can be used to select sentences similar to centroids.

1.3 Hypothesis

The hypothesis of this project is that a text summarization framework using available open source tools and NLP techniques will provide optimized summary results in terms of quality and scalability. The implementation of such a generic framework will allow support to multi-document and multi-lingual summaries. The LSA-based text summary framework will provide summaries with reasonable quality compared to existing systems but will have an edge in terms of scalability.

1.3.1 Summary quality

Summary quality is the primary criterion for determining the efficiency of summary frameworks. Determining the quality of summaries by comparing them with results obtained from other frameworks are necessary steps.

1.3.2 Scalability

As the current systems available do not provide much flexibility in the area of scaling summaries to larger sets of documents, this framework will attempt to bridge that gap. Use of Hadoop-based NLP tools and cloud resources a system's design better facilitates the handling of scalability issues in this framework.

1.4 Roadmap

The remainder of this report discusses the design, implementation, and analysis of the text summarization method. Section 2 and Section 3 describe the detailed design and implementation of the framework, respectively. Section 4 analyzes the summaries and evaluates them against metrics. Text summaries obtained from other systems will be compared with our result set in this section. Section 5 provides the scope for future work and the conclusion.

Chapter 2

Design

The general design for the text summarization system includes two major components.

- Dataset collection
- Summarization of data

2.1 Data collection

Datasets for text summarization have to match certain criteria, the datasets publicly available and specific news related datasets do not provide content redundancy. Handling redundancy in documents is an important feature in our system. For this reason, Data collection has been included into the design of summarization system to provide tightly coupled integration between data and summary model. Initial design did not include data collection as the proposal was to run the summary model against publicly available news datasets [8]. The review sand quality of those datasets weighed against their usage. Inclusion of data collection stage augments our design to extend it to include summarization of data stream in future. In the data collection step, major stages are crawling the web and data extraction. A generic design involving components in each stage can be seen in Figure 3.

2.1.1 Crawling the web

In this stage, we use Apache Nutch [17] web crawler to crawl selected list of URLs and parse the contents of the page. Nutch works on top of Hadoop framework which allows

faster crawling. Seed URLs and crawl depths can be changed to fetch different sets of web pages. Following are the steps involved in crawling,

- Injection - In this step, URLs from input seed are injected into CrawlDB after validating each URL against filter criteria in Nutch configuration.
- Generation - Using the URLs injected into CrawlDB, filter criteria and crawl depths list of URLs fit for fetching the web content are generated.
- Fetch - Each generated url is fetched and its web content which includes html contents, text, meta-data, and headers are stored in segment files.
- Parsing Segment - Segment files are Hadoop readable index files which are parsed for required content in this step.
- Storing in CrawlDB - content obtained from parsing the segment files is stored in CrawlDB for later consumption.

2.1.2 Data extraction

Using Nutch, we crawl the web and fetch content from blogs, articles and websites related to the topic. The segments obtained from the crawler are parsed using TikaParser, HTMLParser classes in Nutch and the required text is extracted. Content obtained from web can be in different formats like .pdf, .txt, .html, raw text is extracted from these file formats using Tika [14].

2.2 Summarization of data

The key components in the data summarization (*input matrix, SVD, sentence selection*) are explained below. A generic design involving components in each stage can be seen in Figure 4. NLP frameworks used in each of the stages are explained below.

2.2.1 Input matrix generation

In the input matrix generation phase, the input is text dataset which has to be summarized and the intermediate output is a sparse tf-idf matrix.

TF-IDF matrix

Lucene API supports indexing of text in different formats like .html, .txt, several other document formats. Preprocessing will involve removal of html tags, extraction of text from different formats, and conversion of text from different languages to unicode. Segmentation of content is done based on groups of phrases, collection of sentences or single sentences. It is known that usage of single sentences provides better results. In this project, we will segment text based on sentences. Using different segmentation methods will affect the nature of matrix, i.e., it will be sparse or dense. Word grouping is done through stemming where words like "running," "runner," "run" are considered as "run". Using Lucene API, words and sentences are filtered to match pre-defined criteria. Lucene is used to index terms in each sentence. Lucene stores meta-data from documents in the form of key-value pairs, where key is the term and value is number of times it occurs in the sentence. Information related to sentences is provided in the index. The output of this phase is tf-idf sparse matrix which will be used for SVD.

2.3 SVD on input matrix

In this phase, SVD is used on the input matrix to obtain feature-sentence relation.

2.3.1 SVD using Mahout

Mahout API provides access to machine learning algorithms, which can be run on hadoop framework to parallelize the jobs. Indexed terms from each sentence are converted to vectors based on a selected weighting metric (*tf-idf, tf, log-entropy*). Lanczos algorithm which is used for eigen value decomposition is modified to obtain SVD term vectors and extracted

features-sentences matrix (V^T).In the sentence selection, the extracted V^T matrix is used against different metrics to obtain different sentence selection processes. Some of the procedures followed in this report are clustering using K-Means, Extracted sentence threshold. Based on different sentence selection approaches we end up with different group of sentences being included into the summary.

2.3.2 Parallelization strategy

Right and left singular eigen vectors can be obtained from matrix A by matrix multiplication AA^T, A^TA respectively. If A is not a square matrix, a few optimizations have to be done to make it a square matrix. Matrix multiplication can be parallelized very effectively as a specific row does not have dependency over other rows or columns. Map-reduce framework will be used to parallelize the indexing of documents and the lanczos algorithm. As most of the underlying algorithms, be it Lanczos, matrix multiplication, KMeans clustering all of these are modified to take advantage of Hadoop framework.

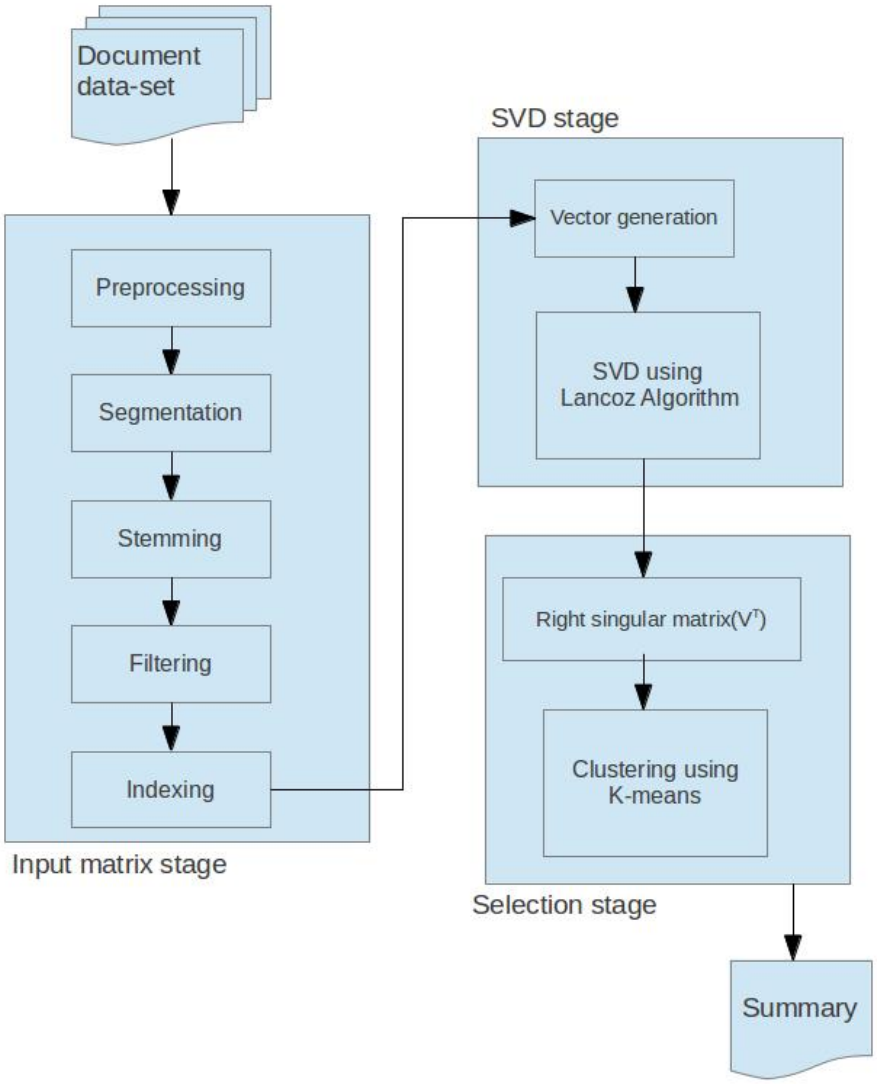

Figure 2.1: System Design.

Chapter 3

Implementation

3.1 Data Extraction from the Web

The data extraction system was written in Java using Eclipse. As the underlying crawler is implemented in Java, we implemented the system on top of it. Data was extracted using Nutch crawler, which is an open-source framework written in Java and which ties in well with Hadoop. This was our primary reason for using Nutch. We also added new classes to the existing framework to fetch web pages that match the system's filter criteria.

3.1.1 Class overview

As a system, data extraction contains several classes. Each stage in crawling has its own class and helper classes. Supporting classes or additional features are part of the Nutch ecosystem in the form of plugins.

The Crawl class is the main class within data extraction, and it works with the Injector, Generator, and Fetcher classes to execute each task assigned to it. The Injector class is responsible for setting the root URL provided by the user in the CrawlDB. The Generator creates segment files that store information extracted from web pages and generates subsequent URLs that are to be crawled. Finally, the Fetcher class, with help of the FetcherThread and FetcherItem classes, scrapes through the queue of URLs from the CrawlDB and dumps the content of the URLs into segment directories.

The content of the web pages can be redirected to the SOLR for indexing and text-mining related tasks, or Nutch has plugins to export content from URLs to No-Sql databases such as Cassandra. In the current version of our data extraction system, we use the raw segment files and extract textual content from them. The data from classes such as Fetcher and Generator are stored in the Segments directory. Each class has its output written in files, which list the key-value pairs that are of key-value pairs readable by Hadoop based applications.

Figure 3.1 shows different classes that interact in the data extraction stage.

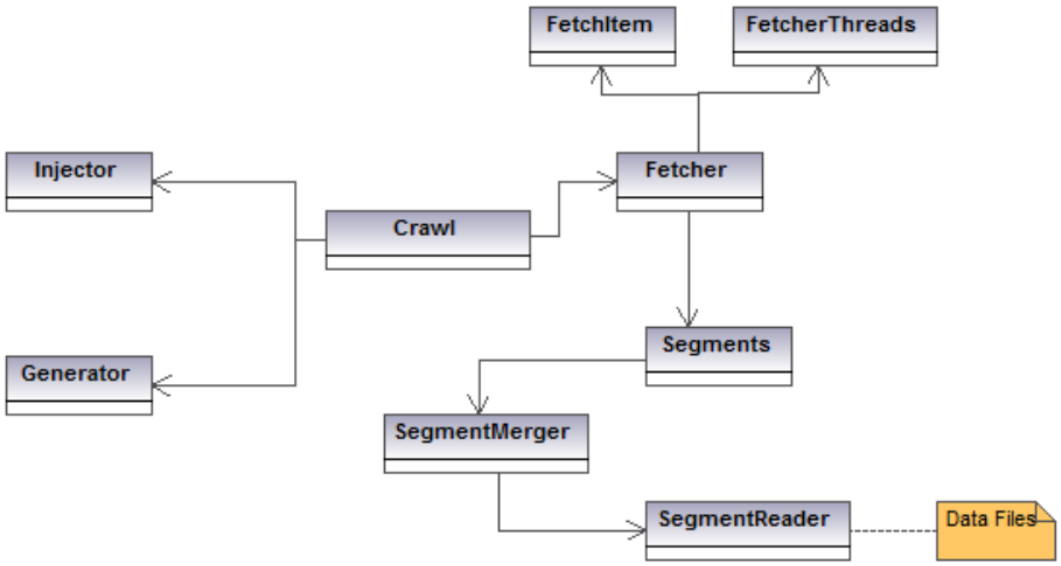

Figure 3.1: Data extraction classes structure

3.1.2 Classes interaction

Figure 3.2 illustrates how the various classes interact with each other and with the data that is written to segment files.

The root URL directory contains seed URLs from which the URL traversal graph is built. It is provided by users through the inject() method. CrawlDB is updated with injected

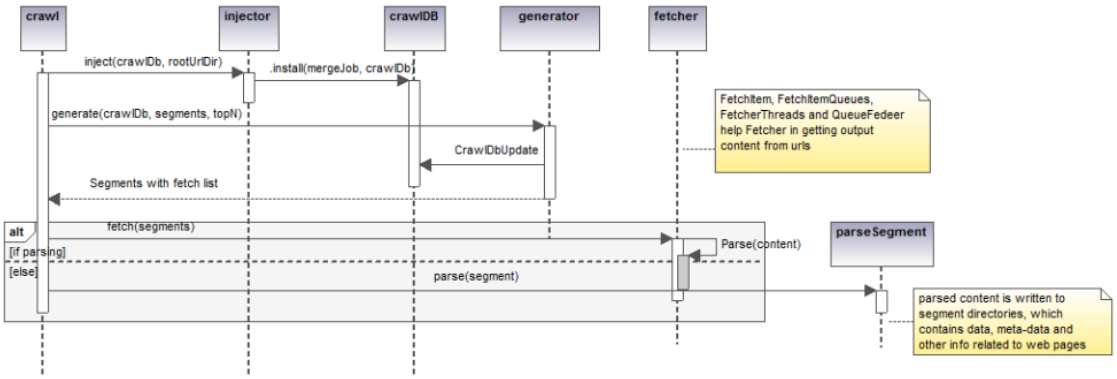

Figure 3.2: Data extraction classes interaction

URLs and keeps track of visited URLs and the ones to be visited. The Generator updates the CrawlDB with the reverse mapping of a URL and its connected URLs such that no URL is visited twice, thus avoiding loops. The URL content is obtained through a queue-based fetcher and uses QueueFeeder to populate the FetchItem queues. Each Fetcher thread picks a Fetch Item from the queue and uses URL from it to extract content.

In our context based data extraction model, we created a plugin for filtering fetches from the queue. This plugin ensures that only the content which has keywords specified by the user is parsed and stored as segment files for further data summarization.

Segment files dumped by fetcher are Hadoop based files which contain parsed data, text, generated data and other script content from the URL. Crawling information from URLs can span across several segment files and SegmentMerger class can be used to combine all the segment files into a single file. SegmentReader class is used to extract required information from the different segment files.

3.2 Summarization of input data

The summarization system is written in Java using Eclipse IDE. We used the Hadoop-based machine learning library, Mahout, and utilized its scalable Map-Reduce (MR) based

algorithms in the application. We leveraged existing Mahout algorithms and customized them to fit our needs. The summarization system has components integrated from Java libraries such as Lucene, Tika, and Mahout that make the application scalable.

3.2.1 Class overview

There are multiple components involved in summarization systems; because of their complex nature, we will cover only the core classes in each component.

The Launcher is the main class, and it interacts with different component classes. The DistributedLancozSolver extends from the LancozSolver and SSVDSolver, two classes that implement SVD based on lanczos algorithm and SVD through the stochastic approach. Each of these algorithms have been hadoopified to leverage the MR model. Following are the individual classes that provide the required functionality to the above-mentioned core classes.

The LancozState class sets up initial vectors and states, diagonal matrix populated with the matrix computation and vectors from matrix decomposition. The HdfsBackedLancozState extends the LancozState, which, along with the aforementioned operations, also sets a path for the vector files on the distributed file system. The original matrix, A, is transformed into a Tridiagonal matrix followed by eigen decomposition, thus obtaining eigen vectors and values. The TimesSquared class is useful when original matrix, A, is not symmetric. When the 'cleanSVD' option is enabled, the EigenVerificationJob class performs validation on the obtained eigen vectors. This is done based on the maximum error allowed for an eigen vector to be excluded from the set.

The SSVDSolver class approaches SVD through the stochastic method. This algorithm is faster when compared to the Lanczos algorithm. The SSVDSolver, with the help of the SSVDHelper class, creates a seed matrix. The matrix multiplication of the original sparse matrix and seed matrix is passed to the QJob class, which performs a QR decomposition

on the matrix. The BtJob and AbtJob classes then perform an intermediate operation on the outputs from the above-mentioned classes. Based on user input, the required output matrices U and V can be obtained. Figure 3.3 shows how the various classes' interact in the data summarization stage.

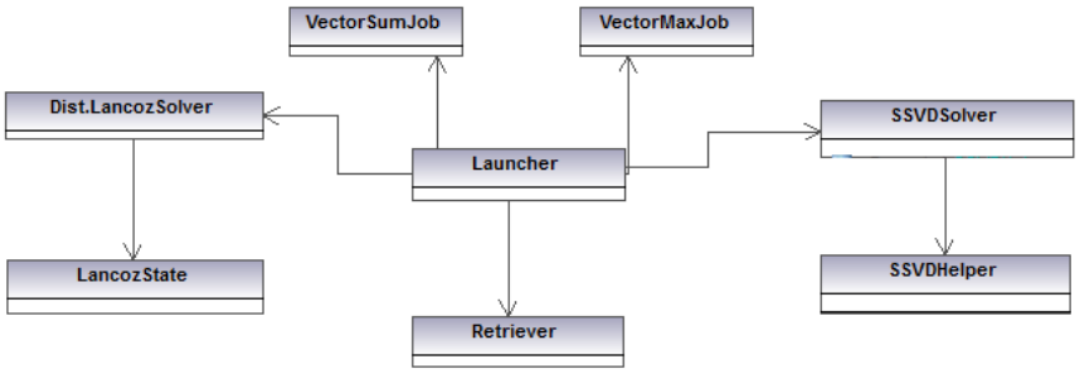

Figure 3.3: Summarization of data classes structure

3.2.2 Classes' interaction

Figure 3.4 shows the program flow and the interaction of different components in the summarization system. The figure provides an illustration of the interaction between core classes and their methods. Elaborate sequence diagrams can be viewed in the Appendix.

Data preprocessing

Data obtained from the web crawler is in a raw text format and is provided as an input to the Launcher. The Launcher can then take other file formats, such as .pdf and .html pages, and use the Tika parser to extract required text. The text is passed to the SentenceDetector, which compares the text with its available English language sentence bin. Based on its evaluation, the SentenceDetector divides the text into a list of sentences.

The summary system will work on the list of sentences provided by this class. Words on the sentences are stemmed, and indexing of words is performed by the Lucene components.

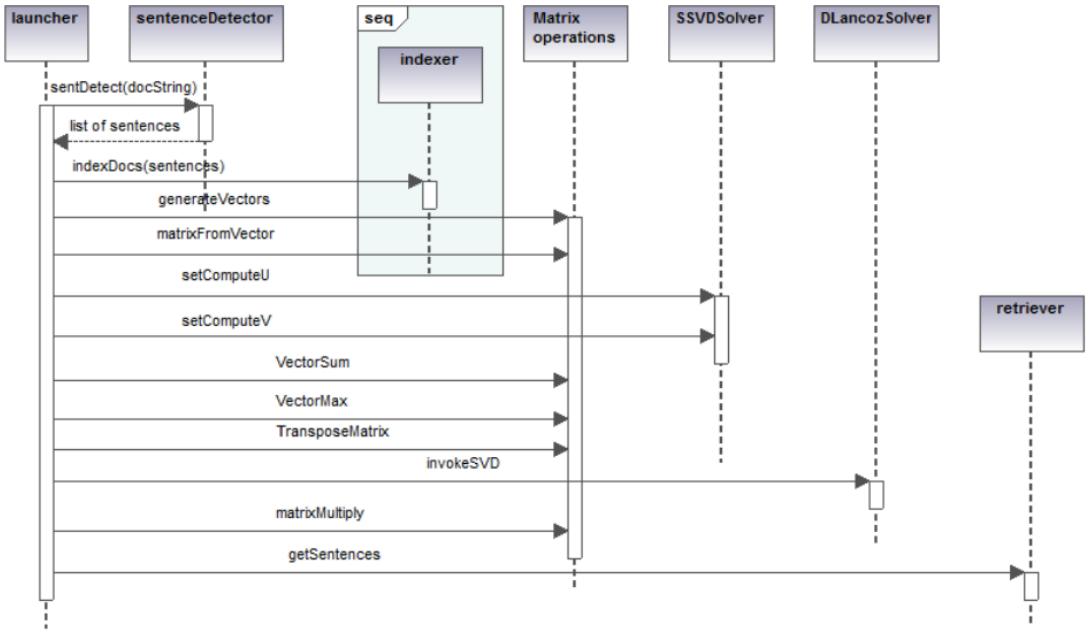

Figure 3.4: Summarization of data classes' interaction

A TF-IDF vector is obtained from the indexDocs method of the Lucene classes. In the figure all the matrix computation operations are grouped together for ease of understanding. Different classes comprise these operations. Finally, the VectorMatrix class converts a vector into a matrix, the MatrixMultiplicationJob performs multiplication on two given matrices, and the TransposeMatrix class transposes a given matrix.

Matrix decomposition

The SSVDSolver and DistributedLancozSolver are two algorithm classes that perform SVM in two different ways; the obtained matrices are used in sentence selection. The SVDSolver uses the stochastic SVD approach in factorizing complex matrices obtained from previous steps. This approach contains multiple MR sequential steps, and these will be discussed in sections below. Using the SSVDSolver, each of the matrices U, V, and Σ, can be obtained indivdually. We require *featuresXsentences* matrix, V, which can be obtained using the setComputeV method. The SSVDHelper class is used to invoke the matrix

operation based method in different supporting classes. The DistributedLancozSolver uses lanczos algorithm for the decomposition of the sparse tf-idf matrix. There are multiple MR sequential steps spanning across various stages in synthesizing the sentence matrix and extracting features from it, but this approach is considerably slower than the SSVD approach and the output from this is only a *featuresXsentences* matrix. The output matrix from both of these methods is sent to the selection methods which filter sentences and retrieve top sentences with minimal collision based on a threshold value.

Sentence selection

The *featuresXsentences* matrix obtained from the SVD classes is then refined based on features per row. For this filtration, we passed on the matrix to the VectorSumJob and VectorMaxJob classes, as these provide the sum and maximum values per row, respectively. Based on the Sum/Max criteria, we select 'k' indices, which were retrieved from the Lucene index by the Retriever class. We further refined the results by passing the matrix through k-means clustering, but it was noticed that the above steps were sufficient and clustering adds an additional time overhead.

3.2.3 Parallelization strategy

For optimal results in all three stages of implementation, we leveraged on indexing and mining APIs dependent on Hadoop. In this section, we will discuss the generic and parallelized versions of algorithms used in data extraction, summary, and selection stages.

Data collection In data collection, the Generator class generates fetchlists that are injected into the CrawlDB. This step of updating segments can run in a multi-threaded format. In the Generator step,each fetchlist is filtered so content can be written to multiple segments.

The Fetcher uses the pub-sub model with one producer, the QueueFeeder, and many subscribers, fetcher threads. The QueueFeeder populates a group of *fetchItemQueues* based on

the fetchlist from the crawlDB. Each *fetchItemQueue* has a few fetch items that have metadata related to the pages to be fetched. The QueueFeeder will manage as many queues as the deployed crawl hosts, providing fetch items to queues and fetch threads that are allocated based on the availability of items. Internally, all the indexing in the crawlDB is optimized using Lucene API. The following is the data collection process flow.

Algorithm 1 Data collection process flow

▷ %comment: Crawl consists of four stages: Generate, Fetch, Parse, UpdateDB%
slaves[]= index(of all fetch threads)
masterNode=QueueFeeder (host)
generatedPages[]=pages(from crawlDB)
$groups \leftarrow groupPages(generatedPages)$
for $subGroup$ in $groups$ **do**
▷ %comment: QueueFeeder groups a few fetch items and assigns them to fetcher-thread%
$fetcherThread(subGroup)$
▷ %comment: Fetcher-thread fetches URL content for its group%
end forend for
repeat ▷ masterNode parses through fetched pages from web-table
Send webpage to slave
if not (webpage parsed by other slaves) **then**
parse the page and save content to webtable
end if
until All pages are completed or User interrupts

Algorithms in the summary stage In this section, we elaborate on the Lanczos method, Stochastic SVD algorithms, and their parallelized versions used in the summary model. *Single value dexomposition* (SVD) or *Eigen value decomposition* (EVD) in thoery are defined as the following:

Theorem 1 (SVD/EVD)

If A $\in R^{\text{mxn}}$ is a real matrix, then there exist orthogonal matrices

$$U = [u_1, \ldots, u_m] \in R^{mxm} \text{ and } V = [v_1, \ldots, v_m] \in R^{nxn} \text{ such that}$$

$$\Sigma = U^T AV = diag(\sigma_1, \ldots, \sigma_k), \text{ where } k = min(m, n)$$

$$\text{where } \sigma_1 \geq \sigma_2 \geq \sigma_3 \ldots \sigma_k$$

The Lanczos procedure is used in SVD to determine the optimal eigen vectors. Mentioned below is the modified approach of the Lanczos method to selectively orthogonalize a matrix and obtain eigen values [15]. The basic premise of the optimized Lanczos method

Algorithm 2 Modified Lanczos method

Input: Matrix A^{nxn}, random seed n-vector *b*, max error ϵ and num of iterations *m*
Output: *k* Eigen values $(\sigma_1, \ldots, \sigma_k)$, *Eigen vectors* Σ^{nxk}

$\beta_0 \leftarrow 0, \nu_0 \leftarrow 0, \nu_1 \leftarrow b/||b||$
for i in range(1 . . . m) **do**
  $\nu \leftarrow A\nu_i$
  $\alpha_i \leftarrow \nu_i^T \nu$
  $\nu \leftarrow \nu - \beta_{i-1}\nu_{i-1} - \alpha_i\nu_i$ // orthogonalize using previous two basis vectors
  $\beta_i \leftarrow ||\nu||$
  $T_i \leftarrow$ Tridiagonal matrix comprisingα, βvalues
  $QDQ^T \leftarrow$ EVD(T) // Eigen value decomposition of Tridiagonal matrix
  for j in range(1 . . . i) **do**
    if $\beta_i|Q[i,j]| \leq \sqrt{\epsilon}|T_i|$ **then**
      $r \leftarrow V_iQ[:j]$
      ▷ %Selectively orthogonalize%
      $\nu \leftarrow \nu - (r^T\nu)r$
    end if
  end for
  ▷ %Recompute constant β%
  $\beta_i \leftarrow ||\nu||$
  if $\beta_i = 0$ **then**
    break //for loop
  end if
  $\nu_{i+1} = \nu/\beta_i$
end for
$T \leftarrow$ Tridiagonal matrix comprising α, β values
$QDQ^T \leftarrow$ EVD(T) // Eigen value decomposition of Tridiagonal matrix
$\sigma[1 \ldots k]$ = top K diagonal values of D
$\Sigma \leftarrow V_mQ_k$ // get Eigen vectors

can be understood from the above algorithm. If A was the large sparse matrix to be decomposed, we began with a random seed vector *b*. In each iteration, we calculated new seed vectors by multiplying previous vectors with the initial sparse matrix. For each iteration, the seed vector calculated became part of nxm matrix Q_m, where β_i is the i^{th} column of

Q. For a given iteration m, it has been defined that $T_m = Q_m^* A Q_m$ and the tri-diagnol matrix T is comprised of α and β values. It can be stated that the eigen values of T_m are approximately close to the eigen values of A. The Eigen vectors of the input matrix can be obtained by a combination of $V_m Q_m$, meaning, vectors from T and Q_m. It can be deduced from the algorithm that most expenive computations in this method are vector multiplications and matrix multiplications. These operation can be easily fit into the MR framework, making the selective orthogonalization lanczos method the best fit for parallelization. This approach provides minimal errors and closer approximations when compared to other parallel methods.

Stochastic SVD (SSVD) approach, similar to the Lanczos method, seeks to decompose a large sparse matrix into smaller, denser matrices. Mathematically, $A \approx U \Sigma V^T$

Theorem 2 (SSVD)

If A is a large sparse matrix and a random matrix Ω is used to make a dense matrix Y such that, $Y = A\Omega$. Basis Q can be derived from Y, which is used to reduce input matrix to form $Q_T A = B$. Matrix B has reduced rank and features only on which SVD can be performed to obtain U, Σ, V^T individually. The obtained matrices are approximately similar to the matrices derived from the original matrix A.

As shown in the algorithm, if the intermediate input matrix B is too large to be stored/processed in memory, it can further be decomposed. The decomposition of matrix B can either be done by right orthonormal matrix decomposition LQ or XQ decomposition.

$$B = X Q_B$$
$$X = U_X \Sigma V_X^T$$
$$B = U_X \Sigma V^T \leftarrow U_X \Sigma V_X^T Q_B$$

Algorithm 3 Sequential flow of SSVD

Input: Large sparse matrix A, random seed Ω that has (k+l) columns,
where k $\leftarrow$ desired rank of SVD, l $\leftarrow$ buffer (≥ 10)
Output: U, Σ, V or all three matrices

1: $Y = A\Omega$ // Compute Y from A, Ω ▷ % Perform QR decomposition on Y %
2: $QR = Y$ ▷ % Use Q, to obtain reduced form of Y %
3: $B = Q^T A$
4: $SVD(B) = U_B \Sigma_B V_B^T$
5: repeat Steps 2, 3 if B is large enough for SVD

$$V^T \leftarrow V_X^T Q_B$$

As we needed only V^T for sentence selection, we could stop at this step. For the parallel implementation, if the input matrix A is too large for multiplication in-memory, it can be divided into chunks, and multiplications in the algorithm are to be done on individual hosts. The MR implementation of the above generic SSVD algorithm is shown in the following algorithm,

Algorithm 4 MR implementation of SSVD

Input: $A_i \leftarrow$ chunk of matrix A
Output: Intermediate matrices B, Q; final output V^T

1: Mapper steps:
2: $Y_i = Mult(A_i, \Omega)$
3: $Q_i R_i \leftarrow QRDecomp(Y_i)$ // Perform QR decomposition
4: Compute $Q_i^T A_i$
5: **for** e **do**ach horizontal block j in $Q_i^T A_i$
6: $collect(j, [i, Q_i^T A_i])$
7: **end for**
8: Reduce steps
9: for given j horizontal block
10: collect vertical blocks of B_j
11: decompose $B_j \rightarrow X_j Q_{xj}^T$
12: $write(X_j, Q_{xj}^T)$
13: Assemble $X_j \rightarrow L$
14: $X \rightarrow U_x \Sigma V_x^T$
15: collect all V_x for j blocks
16: $V^T \leftarrow Map(V_x^T, Q_B)$

V^T from both the approaches is used in sentence selection. In our system, we use two metrics to extract feature based sentences from right singular vectors. The VectorSum approach uses the sum of the features and selects k-maximum from them, while the VectorMax approach selects sentences that have maximum values; we fit both the approaches into MR model.

Deployment overview

The evaluation of the quality of summaries can be done on hosts with average compute resources. To evaluate the optimal levels of quantity that can be processed by our model, we needed high compute resources. We used Amazon web services (AWS) to perform the tests. The source code jars and preprocessed input data were put into the AWS S3 project bucket. For ease when running the tests, we used AWS elastic map-reduce (EMR) to write launcher scripts. Running the launcher scripts from the local host copied the jars into EC2 hosts from which the project is run. The use of launcher scripts, minimized the effort of running the tests on different configurations. We will provide detailed information about the different configurations on which the model was tested in the next section.

Chapter 4

Analysis

4.1 Evaluation approach

The proposed summarization models and the hyposthesis are evaluated in this section. The experimental analysis has been done keeping in mind two aspects, *quality and quantity*. The quality analysis focuses on linguistic quality, readability and compares the extracted summaries with human extracted summaries and existing models like *lexRank and centrality*. The quantitative tests evaluate the scalability and speed-up of obtained summaries. As we do not have access to other scalable summary models, we compare our proposed approaches *(Lanczos vs Stochastic)*.

4.2 Quality analysis

Evaluating summary models for quality and readability is not easy, as the ideal summary differ from person to person. We use automated metrics ROUGE and TESLA-S standardized by Document understanding conference (DUC), for evaluating unsupervised summary models. Both the metrics rely on calculating similarity between candidate summary and a list of reference summaries, usually extracted by a sub-set of humans.

4.2.1 ROUGE metrics

In our experiments, we calculated ROUGE-1, ROUGE-2 and ROUGE-L values for each summary. ROUGE-N (1, 2) and ROUGE-L are mentioned as following,

ROUGE-N

ROUGE-N, is n-gram co-occurences between candidate summaries and reference summaries, where *n* is the number of words to match. In the 1-gram metrics, we collect the ratio of number of single words matching by number of words in reference summary. 2-gram metrics, is the ratio of two continuous words matching in both the summaries. In the case of multi-documents summarization, the average of all the rouge n-gram values is considered.

$$\text{ROUGE-N} = \frac{\sum_{S \in ReferenceSummaries} \sum_{gram_n \in S} Count_{match}(gram_n)}{\sum_{S \in ReferenceSummaries} \sum_{gram_n \in S} Count(gram_n)} \tag{4.1}$$

$$\text{ROUGE-N}(Avg) = \frac{\sum R_n}{num.of documents} \tag{4.2}$$

Evaluation metrics are usually based on precision or recall. Precision is the ratio of number of matching words in candidate, reference summaries by number of words in the candidate summary.

$$Precision = \frac{Num.of matching n-units}{Num.of words in candidate summary} \tag{4.3}$$

Recall based models consider ratio of number of matching n-units by number of n-grams in the reference summary.

$$Recall = \frac{Num.of matching n-units}{Num.of words in reference summary} \tag{4.4}$$

Another metric based on both precision(p) and recall(r) known as F-measure(f) is evaluated as follows.

$$F-measure = \frac{(1+\beta^2) r * p}{r + \beta^2 * p} \text{where } \beta \leq 1 \tag{4.5}$$

ROUGE-L

Rouge-L considers longest common subsequence of words matching between candidate and reference summaries. One advantage of using Rouge-L over n-gram ROUGE metrics is that sentences without continuous match words can still contribute towards the score of

summary. The n-gram need not be defined prior to running the evaluation as it would pick continuous n-units by default. Precision, recall and F-measure for Rouge-L is as follows:

$$P_{LCS} = \frac{LCS(C,R)}{words_{candidate}} \tag{4.6}$$

$$R_{LCS} = \frac{LCS(C,R)}{words_{reference}} \tag{4.7}$$

Another metric based on both precision(p) and recall(r) known as F-measure(f) is evaluated as follows.

$$F_{LCS} = \frac{(1+\beta^2)R_{LCS} * P_{LCS}}{R_{LCS} + \beta^2 * P_{LCS}} \text{where } \beta \leq 1 \tag{4.8}$$

In all our evaluations, we set β value to 0.8.

For comparative analysis, we used a dataset from the DUC 2004 to utilize in the human-assessed reference summaries. The dataset was comprised of 50 topics, and each topic had 10 documents related to it. The average document contained 20 sentences and 400 words. The model summaries assessed by humans were limited to 100 words; in our tests, we did not consider words exceeding this limit in the candidate summaries that were part of the metric.

We extracted four summaries from each topictwo each for *Stochastic SVD, Lanczos SVD*. The names of the generated summaries are *ssvdMax, ssvdSum, svdMax, svdSum*, and we compared these models against standard models including lexRank [22], centrality [1], and baseline.

Centrality summarization is a greedy approach that considers the cosine similarity of vectors from candidate and reference summaries.

$$centrality = \frac{\sum sim(x,y)}{N} \tag{4.9}$$

The similarity function can be consine, dice or jard, but in this evaluation, we chose to use the cosine function.

Lex rank is a graph based approach for finding the sentence similarity between the models. The TF-IDF is calculated for the vectors and the consine similarity is calculated, and only those values that reach an optimal threshold are included in the subset of sentences forming the summary.

Baseline is an approach generally used in summarization evaluation to test the models against randomly selected sentences from the input dataset. We collected baseline summaries using two approaches. In the first approach, we randomly selected one sentence from each document per topic, the second approach was to pick the first sentence from each document. We placed an average of the maximum values from all the topics in the model.

All the models were evaluated based on ROUGE-1, ROUGE-2, and ROUGE-L metrics, and table 4.1 shows the values of recall, precision, and f-measure for each metric.

Model	Rouge-1			Rouge-2			Rouge-L		
	Avg_R	Avg_P	Avg_F	Avg_R	Avg_P	Avg_F	Avg_R	Avg_P	Avg_F
baseline	0.34263	0.33855	0.34053	0.07722	0.07630	0.07675	0.30093	0.29732	0.29907
ssvdSum	0.36855	0.36058	0.36452	0.08189	0.08010	0.08099	0.32513	0.30862	0.31340
ssvdMax	0.36364	0.35238	0.35792	0.06700	0.06490	0.06593	0.30713	0.29762	0.30230
svdSum	0.39312	0.38462	0.38882	0.09677	0.09466	0.09570	0.34398	0.33654	0.34022
svdMax	0.38084	0.36905	0.37485	0.10174	0.09856	0.10012	0.33661	0.32619	0.33132
lexRank	0.34889	0.33810	0.34341	0.05459	0.05288	0.05372	0.29238	0.28333	0.28778
centrality	0.36114	0.36661	0.36369	0.07496	0.07595	0.07542	0.28442	0.28874	0.28643

Table 4.1: metrics of stemmed summaries on all models

From the above table, it can be observed that the *svdSum* model performed well across all metrics compared to other models. *svdSum* is followed by *svdMax, ssvdSum, ssvdMax* respectively, It can be observed from this test that the summing of features is a better selection process than picking sentences with a maximum feature value.

Among the various metrics we evaluated, ROUGE-2 recall metric is considered to be a stable system for ranking a model by the DUC. Table 4.2 shows only Rouge-2 metrics for all the models, revealing that svdSum is again better based on Rouge-2 recall value. The lexrank and centrality models perform better compared to ssvdMax, ssvdSum models.

Model	Rouge-2		
	Avg_R	Avg_P	Avg_F
ssvdSum	0.04395	0.04499	0.04446
ssvdMax	0.03394	0.04193	0.03751
svdSum	0.07735	0.08127	0.07923
svdMax	0.06962	0.07057	0.07009
lexRank	0.05200	0.05603	0.05394
centrality	0.05800	0.06102	0.05932
baseline	0.06722	0.06445	0.06566

Table 4.2: Metrics of summaries that were stemmed but with no stop words

Rouge metrics can be evaluated for the models based on features such as stemming, excluding stop words. We have calculated metrics that use only stemming, metrics for which only stop words have been removed, and metrics for which both stemming and stop words have been removed. Figure 4.1 provides a graphical representation of models against the three aspects. The DUC recommends usage of metrics with stemming only, but the other approaches were used for comparison.

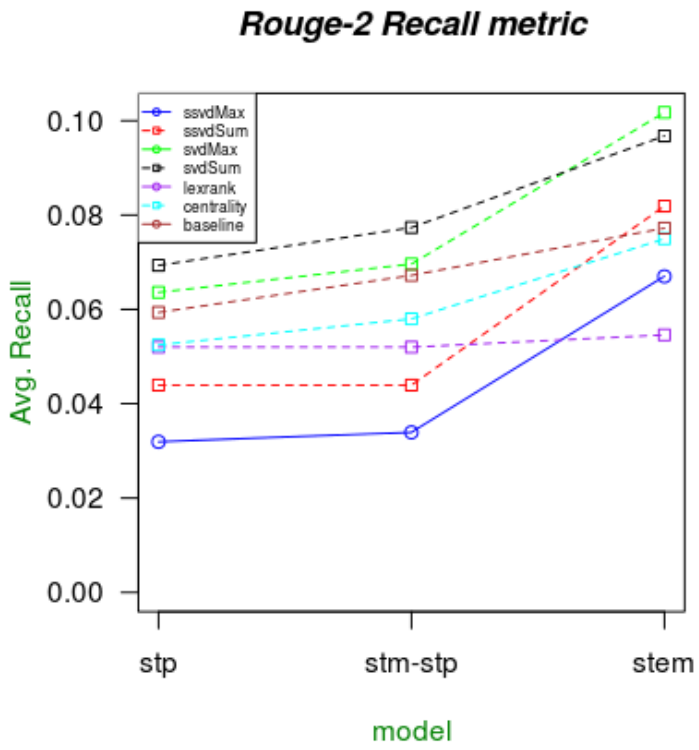

Figure 4.1: ROUGE-2 comparing all models

4.2.2 TESLA-S

This model has been developed recently, in comparison to the ROUGE model, and derives many features from it. An additional feature of TELSA, however, is that it includes weighted scores for the recall and precision values. The metric evalutes f-measure only to compare the models. Research papers on TESLA mention that it considers the readability of a summary along with its matching similarity when evaluating the models. Table 4.3 shows the f-measure of our summary models; the metric seems to favor the maximum feature-based sentence selection over the sum of features selection.

Model	Tesla-S (F-measure)
ssvdSum	0.18625
ssvdMax	0.2119
svdSum	0.1886
svdMax	0.2448

Table 4.3: F-measures of our summary models

4.3 Quantity analysis

In this section, we evalute the scalability and speed-up aspects of the model. As per the needs of evaluation, datasets were gathered using Nutch API. We used two datasets to evaluate the models in this section. The first dataset is a 200MB archive of legal documents related to several cases undertaken by the Australian judiciary in 2003. The second dataset is a 500MB crawl log from the Nutch of sub-reddit (r/technology). Both the datasets are available in an s3 project instance, which is accessible to EMR jobs deployed with the purpose of being used to generate summaries. As we did not have access to any other parallel data summarization models, comparison was completed using the Lanczos approach and SSVD approach. It is already known that the stochastic approach is better suited for parallelization than is the Lanczos approach.

The figure 4.5 shows the number of cores vs. the time plot, providing insight into the speed-up achieved by the models with increase of slaves in place. From the plot, it can be observed the stochastic model performs much better than the lanczos model. The dotted lines in the graph are the ideal speed-up times expected from the respective models. The reason for the difference between the expected and actual times could be due to various aspects like network delays, I/O delays, and bottlenecks at the reducer stages. Figures 4.2, 4.3, 4.4 show the usage of mappers and reduces in running the evaluations and the HDFS reads from the disk. The evaluation of dataset-1 allowed for an understanding of the models and their respective performances. To evaluate the scalability of the models, we tested them with dataset-2. Figure 4.9 shows the graph obtained by plotting time vs. cores for the larger dataset, and the trends from this graph are not different from the previously observed

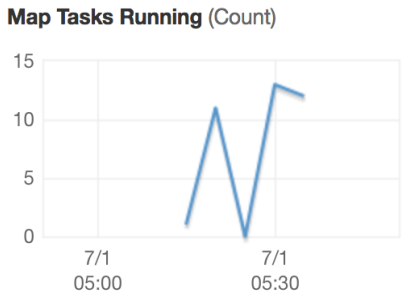

Figure 4.2: Num. of Mappers used

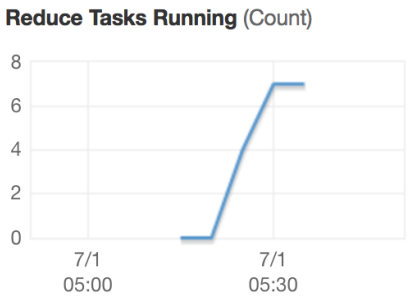

Figure 4.3: Num. of Reducers used

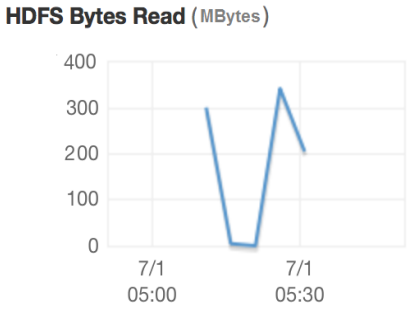

Figure 4.4: HDFS Bytes read

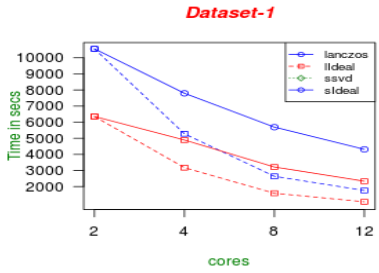

Figure 4.5: Time vs. cores graph for dataset-1

evaluation. To test the system limitations, we extended the evaluations to a much larger dataset of 1 GB, but due to inherent design limitations, the Lanczos model had memory issues. In the current Lanczos model, the complete input matrix has to be loaded into the memory to perform computations on it. This inherent design flaw should be rectified in future iterations of the model. The current workaround for this issue is to divide the input matrix into horizontal blocks and work with them in batches. The stochastic model does not have this issue as it performs matrix operations on chunks of input matrix but not on the whole.

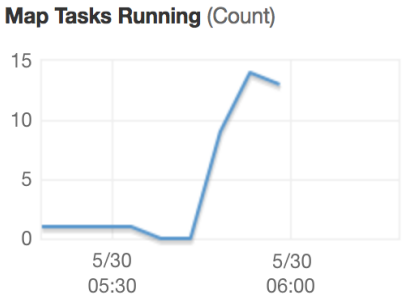

Figure 4.6: Num. of Mappers used

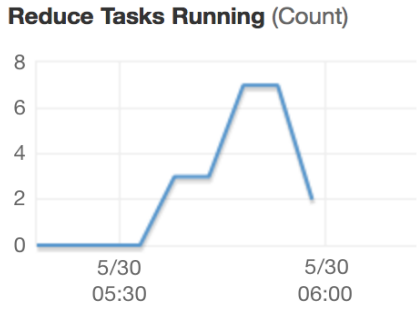

Figure 4.7: Num. of Reducers used

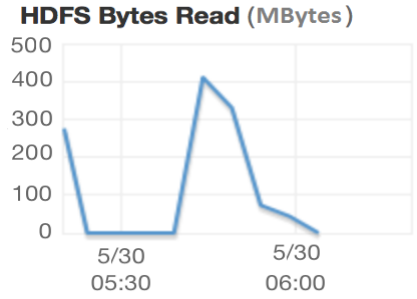

Figure 4.8: HDFS Bytes read

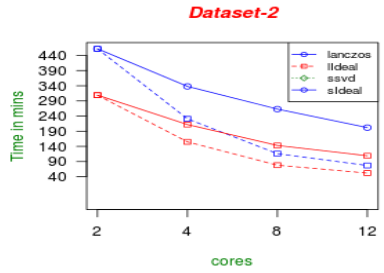

Figure 4.9: Time vs. cores graph for dataset-2

4.4 Multi-lingual summarization

The *Lanczos and Stochastic* models used in our system utilize the TF-IDF matrix as the input for their computations. We use the Lucene API to generate the matrix, as the framework supports unicode text. In the current system, the Lucene standard analyzer does not support stemming and stop word removal of text in the Hindi language. Although the system works on text documents in Hindi, we could not evaluate the model extensively on hindi datasets due to limitations in obtaining comparative models.

To evaluate our hypothesis, I converted sections of the legal documents dataset (dataset-1) used in quantity analysis. This dataset was fitted into our models without excluding stop words and stemming. We could not evaluate the quality of multi-lingual documents due to unavailability of reference summaries for the dataset.

इन आ क्रिटिकल रूलिंग फॉर थे नॉर्थ अमेरिकन नॅशनल बॅस्केटबॉल असोसियेशन आंड थे प्लेयर्स यूनियन, आर्बिट्रेटर जॉन फीरिक्क डिसाइड्स मंडे वेदर मोरे तन 200 प्लेयर्स वित गॅरेंटीड कॉंट्रॅक्ट्स शुड बे पैड ड्यूरिंग थे लॉकाउट.
इफ़ थे प्लेयर्स विन, थे ओनर्स विल बे लाइयबल फॉर अबौट द्लर्स 800 मिलियन इन गॅरेंटीड सॅलरीस, ऑल्दो थे हॅव वोड तो अपील इफ़ थे लूज़.
थे लीग ऑलरेडी हास सूड थे प्लेयर्स ओवर फीरिक्क'स जुरिस्डिक्षन.
इफ़ वी विन, ई थिंक इट जस्ट एम्बॉल्देन्स थे स्पिरिट आंड रिसॉल्व ऑफ थे प्लेयर्स, यूनियन डाइरेक्टर बिली हंटर साइड.
बुत ई डॉन'त थिंक तेरे विल बे एनिबडी सेलेब्रेटिंग बिकॉज़ तेरे'स नो गॅरेंटी तट इट विल एंड थे लॉकाउट.
इट ओन्ली मीन्स थे हॅव तो पे सम 200 प्लेयर्स, आंड थे'वे इंडिकेटेड तो उस देयर इनटेंट तो फाइल आन इम्मीडियेट अपील आंड टके इट आस फर आस थे हॅव तो इन ऑर्डर तो अवाय्ड पेमेंट.

Figure 4.10: Section of Hindi dataset

4.5 Hypothesis evaluation

One of the primary goals of this thesis was to study existing summarization models and design a system that can retain or improve the summary quality of these existing models. Based on the evaluations performed using ROUGE and Tesla-S metrics, our models faired better when compared to generic models such as LexRank and cosine similarity. Sentence selection based on sum of features performed much better than the widely used existing models. Our model also scored better in terms of excluding similar sentences from the final summary. Tesla-S metrics showed that our models scored better in terms of readability when compared to centrality and lexRank. Lanczos model with sum of features provided the best overall ROUGE scores but readability could be improved.

In our hypothesis, we emphasized learning the current machine learning APIs and utilizing them in our system. We made a conscious decision to include Lucene,and Mahout APIs in our summarization model and modified them to suit our needs. As we experienced difficulty in obtaining an apt dataset for our evaluations, the inclusion of Nutch was advantageous. It helped us to gather multiple documents on topics of our interest. Leveraging on top of these APIs allowed us to scale our system with minimal issues.

In terms of quantity analysis, the *stochastic* approach scaled very well compared to other models. The Lanczos model could be bettered by following the model in which the Stochastic design is done. The timings could be further reduced by using techniques such as NFS

caching and the inclusion of multiple reducers. We tested the systems performance on multi-lingual documents and the model readily works with them. But pre-processing of datasets which includes indexing of terms by excluding stop words, stemming, and collection of frequent terms should be done in detail to obtain better results.

Chapter 5

Conclusions

5.1 Current Status

The system in its current status provides extractive summaries of a quality better than the compared models. The system architecture has Hadoop framework integrated into it that allows it to scale to multiple cores and summarize large datasets. It has been observed from the evaluations that the model works well for medium and large datasets, but it could be improved to work well with smaller datasets, as well. As the evaluation of summaries is an important aspect of the system, Nutch API has been included in the system to provide quality datasets. In the current model, we used a TF-IDF matrix generated from the input dataset for all the matrix computations. The current iteration of the system can extract sentences from multi-lingual datasets. It can be extended to include sentence detection, removal of stop words and grouping of similar words for better results.

5.2 Lessons Learned

From the onset, the summarization model was planned to be generic, and focus of the system was primarily on attaining summaries from large datasets without compromising on the quality. While doing so, some aspects like tone of the content and readability were not the prime focal areas. We came across some of the limitations of the system while evaluating it; a few limitations are mentioned below

5.2.1 General issues

A generic approach in providing summaries is difficult and having domain knowledge of the dataset can provide better results. As an example, the summarization of legal documents works better if the sentences containing legal jargon have more weighted scores in comparison to others.

We currently use TF-IDF scores of dataset to determine the input matrix, the reason for choosing this weighting measure is to include outlier sentences which have terms that occur rarely. This approach is useful when the dataset is large and the expected summaries are generic. If the dataset includes a sub-set of documents which unrelated to the topic, this method will not be able to eliminate noisy content from the summary.

The multi-lingual summarization is supported by our system but there are a few limitations in the existing model. The indexing of terms in Hindi document is done using HindiAnalyzer Class in Lucene. The existing analyzer does not exclude stop words and grouping of similar terms, the sentence detection of Hindi documents also needs to be extended to fetch sentences with weighted terms.

5.2.2 Tools and APIs

As we have multiple open-source tools integrated into the system, there was a bit of a learning curve in applying Mahout, Lucene, and Nutch to our needs. The documentation was not always helpful, but online forums and blogs were resourceful in getting most of the issues resolved.

The APIs used in the system have common dependencies from third-party libraries with different versions, the classes had to be re-built with compatible version for all the three APIs. Due to this limitation, we had to use Hadoop 0.21 which was the only version that worked without issues on all the third-party libraries used in the system.

In the current version, we are using Lucene API for indexing the dataset and storing TF-IDF values from it. IndexReader in Lucene does not support searching in HDFS directory due to which indexing and TF-IDF generation steps are limited to single master node.

The scalability of the system can be extended if these steps could leverage the distributed Hadoop architecture.

5.2.3 Data collection

The collection of datasets has been the most challenging issue, as the textual datasets should have redundancy in topics but there should also be a range of topics available in it. In the initial implementation stages, I intended to use the Reuters dataset and 20-Newsgroup datasets but I was suggested by Prof.Reznik not to use them as they do not cover the required criteria with regard to redundancy. For this reason, we included the Nutch API into the model to extract datasets from the web. Initial attempts to crawl the Web and extract datasets did not produce good results. In one of the attempts, I attempted crawling FBI public data records and was successful in saving the records. However, the records were not useful as most of them were scanned copies of handwritten documents. Our system was not able to extract content from scanned pdf files. Later, I scoped the crawler to extract content from reddit pages which I used for testing the scalability of the model.

5.3 Future Work

5.3.1 Quality

The following components could be included into future iterations of the system to improve its quality.

First, the use of weighted TF-IDF and log-entropy should be considered for providing the input vectors and matrices. Although it is known that TF-IDF matrices provide a better input than log-entropy models, to log-entropy models, it would be useful to extend the system to utilize all three methods.

Second, inclusion of sentiment analysis could be a useful feature in the system as it can help select the sentences specific to the tone of the sentiment.

Finally, an important use-case for summarizing texts is to classify them, so it would be

useful to improve the system to work better with small datasets.

5.3.2 Quantity

The Lanczos model works well for deriving quality summaries, but does not scale well compared to the stochastic approach. The current design limitation of having the whole matrix in the memory has to be fixed in this model. The model should prioritize the selection of more sentences from initial and final sections, as it has been observed that most writers include key topics in these sections. As more users are using smartphones and handheld devices, it would also be useful to make this system work in a mobile environment.
The indexing and matrix generation step can be extended to utilize Hadoop framework by integrating Apache Blur [3] API into the system design. Blur extends Lucene API to work with HDFS, this would provide additional speed-up.

5.3.3 Multi-lingual texts

We have observed that the system extracts summaries for multi-lingual texts but must be evaluated with a standard test-bed. In the case of Hindi documents, reference summaries should be created for standard datasets. Sentence detection is complicated for Hindi documents but a pre-processing step can be added to include delimiters after each sentence for training datasets.

5.4 Conclusion

The system proposed unsupervised extraction of generic summaries from large datasets. Based on the evaluation it can be determined that the quality of summaries is better than the existing models used in MDS. Usage of distributed ML frameworks in the system allowed us to extract information from large datasets quickly. Though the analysis provides satisfactory results, the system can be extended further by working on above mentioned limitations.

Bibliography

[1] Ghaleb Algaphari, Fadl M. Ba-alwi, and Aimen Moharram. Article: Text summarization using centrality concept. *International Journal of Computer Applications*, 79(1):5–12, October 2013. Published by Foundation of Computer Science, New York, USA.

[2] Rachit Arora and Balaraman Ravindran. Latent dirichlet allocation and singular value decomposition based multi-document summarization. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, ICDM '08, pages 713–718, Washington, DC, USA, 2008. IEEE Computer Society.

[3] Tim Williams and Cloudera, Inc. Blur incubation project, 2014.

[4] John Conroy and Dianne P. O'leary. Text summarization via hidden markov models and pivoted qr matrix decomposition. Technical report, In SIGIR, 2001.

[5] Hal Daumé, III and Daniel Marcu. Bayesian query-focused summarization. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, ACL-44, pages 305–312, Stroudsburg, PA, USA, 2006. Association for Computational Linguistics.

[6] H. P. Edmundson. New methods in automatic extracting. *J. ACM*, 16(2):264–285, April 1969.

[7] David A. Ferrucci. Ibm's watson/deepqa. *SIGARCH Comput. Archit. News*, 39(3):–, June 2011.

[8] Filippo Galgani, Paul Compton, and Achim Hoffmann. Combining different summarization techniques for legal text. In *Proceedings of the Workshop on Innovative Hybrid Approaches to the Processing of Textual Data*, HYBRID '12, pages 115–123, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics.

[9] Yihong Gong. Generic text summarization using relevance measure and latent semantic analysis. In *in Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2001.

[10] Otis Gospodnetic and Erik Hatcher. *Lucene in Action (In Action series)*. Manning Publications, December 2004.

[11] Sanda Harabagiu and Finley Lacatusu. Topic themes for multi-document summarization. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '05, pages 202–209, New York, NY, USA, 2005. ACM.

[12] Eduard Hovy and Chin-Yew Lin. Automated text summarization in summarist, 1999.

[13] H.P. Luhn. The automatic creation of literature abstracts. *IBM Journal*, 2:159–165, 1958.

[14] C. A. Mattmann and J. Zitting. *Tika in Action*. Manning Publications, 2011.

[15] Bernard Philippe Michael W. Berry, Dani Mezher and Ahmed Sameh. Parallel algorithms for the singular value decomposition.

[16] R. Mihalcea and P. Tarau. TextRank: Bringing order into texts. In *Proceedings of EMNLP-04and the 2004 Conference on Empirical Methods in Natural Language Processing*, July 2004.

[17] José E. Moreira, Maged M. Michael, Dilma Da Silva, Doron Shiloach, Parijat Dube, and Li Zhang. Scalability of the nutch search engine. In *Proceedings of the 21st Annual International Conference on Supercomputing*, ICS '07, pages 3–12, New York, NY, USA, 2007. ACM.

[18] Joel Larocca Neto, Alex Alves Freitas, and Celso A. A. Kaestner. Automatic text summarization using a machine learning approach. In *Proceedings of the 16th Brazilian Symposium on Artificial Intelligence: Advances in Artificial Intelligence*, SBIA '02, pages 205–215, London, UK, UK, 2002. Springer-Verlag.

[19] Tadashi Nomoto. Bayesian learning in text summarization. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, HLT '05, pages 249–256, Stroudsburg, PA, USA, 2005. Association for Computational Linguistics.

[20] Sean Owen, Robin Anil, Ted Dunning, and Ellen Friedman. *Mahout in Action*. Manning Publications, 1 edition, January 2011.

[21] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999. Previous number = SIDL-WP-1999-0120.

[22] Dragomir R. Radev. Lexrank: Graph-based lexical centrality as salience in text summarization. *Journal of Artificial Intelligence Research*, 22:2004, 2004.

[23] Josef Steinberger and Karel Ježek. Update summarization based on latent semantic analysis. In *Proceedings of the 12th International Conference on Text, Speech and Dialogue*, TSD '09, pages 77–84, Berlin, Heidelberg, 2009. Springer-Verlag.

[24] Wei Wang, Furu Wei, Wenjie Li, and Sujian Li. Hypersum: hypergraph based semi-supervised sentence ranking for query-oriented summarization. In *Proceedings of the 18th ACM conference on Information and knowledge management*, CIKM '09, pages 1855–1858, New York, NY, USA, 2009. ACM.

Appendix A

Code Listing

The complete application jars, java docs and source code can be found in the attached disc.

Appendix B

User Manual

The user manual for running the application locally and deploying it on a cluster is provided in README file in the attached disc.