

Incorporating Big Data Aspects In Incremental Taxonomy Evolution



By

Kanwal Aalijah

Fall 2018-MS (CS-08)-00000274413

Supervisor:

Dr. Rabia Irfan

A dissertation submitted in partial fulfillment of the requirements for the degree of
Master of Computer Science

DEPARTMENT OF COMPUTER SCIENCE

SCHOOL OF ELECTRICAL ENGINEERING & COMPUTER SCIENCE


NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY (NUST),

ISLAMABAD, PAKISTAN

May, 2020

THESIS ACCEPTANCE CERTIFICATE

Certified that final copy of MS/MPhil thesis entitled "Incorporating big data aspects in incremental taxonomy evolution" written by KANWAL AALIJAH, (Registration No 00000274413), of SEecs has been vetted by the undersigned, found complete in all respects as per NUST Statutes/Regulations, is free of plagiarism, errors and mistakes and is accepted as partial fulfillment for award of MS/M Phil degree. It is further certified that necessary amendments as pointed out by GEC members of the scholar have also been incorporated in the said thesis.

Signature: _____ 

Name of Advisor: Dr. Rabia Irfan

Date: 10-Jun-2020

Signature (HOD): _____

Date: _____

Signature (Dean/Principal): _____

Date: _____

Approval

It is certified that the contents and form of the thesis entitled "Incorporating big data aspects in incremental taxonomy evolution" submitted by KANWAL AALIJAH have been found satisfactory for the requirement of the degree

Advisor : Dr. Rabia Irfan

Signature:  _____

Date: 10-Jun-2020

Committee Member 1:Dr. Sharifullah Khan TI

Signature:  _____

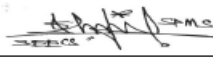
Date: 10-Jun-2020

Committee Member 2:Asad Ali Shah

Signature:  _____

Date: 11-Jun-2020

Committee Member 3:Dr. Safdar Abbas Khan

Signature:  _____

Date: 11-Jun-2020

ABSTRACT

Big data is a breakthrough technology that has been developed over the years. Big data means very large set of data that grows in ever increasing rate and the volume of data is of size Exabyte (10^{18}). It refers to extensively varied data that is processed at very high velocity. Big data is being generated from social media, websites, personal electronics, apps etc. This volume of data currently exceeds the computational capabilities of conventional systems. Unstructured text data produced from several sources needs to be processed and organized. Taxonomy effectively organizes the data in today's digital world. Data in today's digital world is growing at a rapid pace. A taxonomy generated for big data should represent the underlying data and changing theme of data. When this existing taxonomy is evolved, again it should reflect changes that has occurred in data. There is a need of incremental taxonomy generation technique that handles the fast arriving big data of documents and arranges it in a hierarchical structure and also on the next input stream of data it evolves the existing hierarchical structure by adjusting the new data stream. In order to cater the fast arriving big data, the technique needs to run on a parallelization framework so that the running time of incremental taxonomy generation process can be reduced and to improve the scalability challenges of current incremental taxonomy generation techniques. This work presents a technique for incremental taxonomy generation for unstructured text data on Apache Spark framework. The proposed technique not only generates the taxonomy on parallelization framework but also incrementally updates the existing taxonomy. The technique is tested in comparison to non-incremental taxonomy generation techniques. It was found that the proposed technique generates a taxonomy by taking less time as compared with existing taxonomy generation techniques that can make taxonomy utilization more effective. The proposed technique was also tested in comparison to incremental taxonomy generation techniques. Through experiments it was found that the proposed technique updated an existing taxonomy in considerably less time as compared with the existing incremental taxonomy generation techniques. The proposed technique updates the existing taxonomy in seconds, whereas previous algorithms were taking time in minutes and hours for the process. This research work also provides a comparison between two prominent big data environments i-e Apache Hadoop and Apache Spark so it could be investigated

that which big data environment is better suited for a clustering problem like incremental taxonomy generation. Through experiments it was found that Apache Spark is faster and well better suited for a clustering problem like taxonomy generation as compared with Apache Hadoop. The proposed technique was also ran on different configurations of Apache Spark, to find out the optimal number of cores for running any hierarchical clustering jobs on Apache Spark.

Dedication


This thesis is dedicated to my husband who has always encouraged me for everything that I want to do. Thank you for believing in me.

Certificate of Originality

I hereby declare that this submission titled "Incorporating big data aspects in incremental taxonomy evolution" is my own work. To the best of my knowledge it contains no materials previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any degree or diploma at NUST SEECS or at any other educational institute, except where due acknowledgement has been made in the thesis. Any contribution made to the research by others, with whom I have worked at NUST SEECS or elsewhere, is explicitly acknowledged in the thesis. I also declare that the intellectual content of this thesis is the product of my own work, except for the assistance from others in the project's design and conception or in style, presentation and linguistics, which has been acknowledged. I also verified the originality of contents through plagiarism software.

Student Name:KANWAL AALIJAH

Student Signature: _____



Acknowledgement

I am thankful to Allah Almighty Who guided me throughout this work at every step for every new thought and direction.

I would also like to express special thanks to my supervisor Dr. Rabia Irfan for her guidance during the course of my research. This work would not have been possible without her guidance and involvement. Her guidance helped me to overcome the difficulties faced during this research work. I am extremely thankful to my beloved parents who supported me throughout in every endeavor.

Finally, I owe thanks to a very special person in my life, my husband, who gave me constant support during my MS degree. He was always there to support me in times when I thought it won't be possible to complete this degree. I really appreciate my daughter Zunaira for understanding my absence and the patience she showed during my studies is commendable. Words can never truly justify to the fact that how grateful I am to my family for helping and understanding me through the degree and research work.

Table of Contents

Chapter 1.	INTRODUCTION	1
1.1	Motivation.....	1
1.2	Research Problem	3
1.3	Research Objective	4
1.4	Applicability of the Work.....	4
1	Browsing And Searching Of Information Over The World Wide Web	5
2	Customer Support	5
3	Website Organization:	5
4	Content-as-a-Service.....	6
5	RESTFul API.....	6
6	Use of Taxonomy E-Commerce	6
7	Use of Taxonomy in Sentiment Analysis	7
8	Use of Taxonomy in Spam Filtration.....	7
9	Use of Taxonomy in Genre Classification.....	7
10	Use of taxonomy in predicting future trends.....	7
1.5	Thesis Organization	7
Chapter 2.	LITERATURE REVIEW	9
2.1	Basic Process of Taxonomy Generation.....	9
2.1.1	Data Preprocessing.....	10
1	Natural Language Processing based approach.....	10
2	Non-NLP based approaches.....	11
2.1.2	Data Modeling	11
1	Vector Based Approach	11
2	Concept based approach	11
2.1.3	Hierarchy Formation.....	12
1	Clustering based approach	12
2	Graph based approach.....	13
3	Rules based approach.....	13

2.1.4	Node Labeling.....	13
2.2	Types of Taxonomy Generation	14
2.2.1	Non-Incremental Taxonomy Generation	14
1	TaxGen.....	15
2	TaxaMiner.....	16
3	TaxoLearn.....	16
4	OntoLearn	16
2.2.2	Incremental Taxonomy Generation	16
1	AdaptTaxa.....	17
2	EvoTaxa	18
3	IHTC Taxa	18
4	TIE	19
2.3	Hierarchical Clustering For Big Data Sets.....	19
2.3.1	Hierarchical clustering algorithms.....	20
1	BIRCH	20
2	CURE.....	20
3	ROCK	20
4	Chameleon	20
5	ECHIDNA.....	21
6	SNN.....	21
7	CACTUS.....	21
8	GRIDCLUST	22
2.3.2	Density based clustering algorithms	22
1	DBSCAN	22
2	OPTICS.....	22
3	DBCLASD.....	22
4	GDBSCAN	22
5	DENCLUE.....	23
2.3.3	Grid based clustering algorithms	23
1	STING.....	23

2	Wave Cluster.....	23
2.3.4	Model based clustering algorithms	23
1	PROCLUS.....	23
2	ORCLUS.....	24
2.3.5	Partition Based Clustering Algorithms For Big Data	24
1	FCM Fuzzy CMEANS algorithm	24
2.4	Critical Analysis.....	24
Chapter 3.	Background and Preliminaries	26
3.1	Preface.....	26
3.2	MapReduce	27
3.2.1	MapReduce Framework.....	27
3.3	Apache Hadoop.....	29
3.3.1	HDFS	30
1	Name Node	31
2	Data Node	31
3.3.2	Yarn.....	31
3.3.3	Example of a Hadoop Job	32
3.3.4	Limitation of the Hadoop Framework	32
3.4	Apache Spark	33
3.4.1	Resilient Distributed Datasets (RDDs)	34
3.4.2	Spark Streaming.....	35
3.4.3	MLlib	35
3.4.4	Example of Spark.....	36
3.5	Choice of Big Data Environment For the Implementation.....	37
Chapter 4.	Proposed Algorithm	38
4.1	Taxonomy Generation	38
4.1.1	Loading Data.....	38
4.1.2	Data preprocessing.....	38
1	Removal of stop words	39
2	Stemming	39

4.1.3	Data modeling	40
1	Term frequency-inverse document frequency	40
2	Hashing Trick/ Kernel Trick.....	42
3	Feature vectorization using hashing trick (HashingTF).....	42
4	MurmurHash3	43
4.1.4	Formation of hierarchy	43
1	HAC	43
2	Cosine Similarity	44
3	Ward Method	45
4.1.5	Cluster labeling/Node labeling	46
4.1.6	Conversion into a tree graph.....	47
1	Newick Trees	47
4.2	Taxonomy Evolution	48
4.2.1	TreeMerge.....	48
1	NJMerge:.....	48
4.3	Summary	49
Chapter 5.	Evaluation	50
5.1	Evaluation metrics for clustering quality	50
1	Silhouette Score	50
2	Davies Bouldin Score	51
5.2	Experiments And Results.....	52
5.2.1	Experiment for generation process	52
5.2.2	Experiment for evolution process:.....	55
5.2.3	Running time evaluation for different cores of apache spark.....	58
5.2.4	Running time evaluation on Apache Hadoop Vs Apache Spark	60
1	Running Time Evaluation.....	62
2	Performance Evaluation.....	62
5.3	Discussion	62
Chapter 6.	Conclusion	64
6.1	Discussion on research gap.....	64

6.2	Discussion on research objectives	64
1	Research Object 1 - To solve the scalability issues of current clustering-based incremental taxonomy generation algorithms.....	64
2	Research Object 2 - To propose/improve a scalable algorithm to update/evolve taxonomy in presence of a new document.....	65
3	Research Objective 3: Explore the comparison of Apache Hadoop and Apache Spark using the proposed technique.....	65
4	Research Objective 4: Validate the significance of the obtained result.....	66
6.3	Contributions.....	67
6.4	Limitation of the work and future direction.....	67
Appendix A: Algorithm For The Proposed Technique.....		69
	Generation Part of Algorithm	69
	Evolution Part of Algorithm	69
Appendix B: Software and System Specification.....		71
	B1: Software Specifications:.....	71
	B2: System Specifications:	71
References.....		72

Table of Tables

Table 5.1 Results For Quality-Based Evaluation Using Silhouette’s Score- Taxonomy Generation	52
Table 5.2 Results For Quality-Based Evaluation Using Davies-Bouldin’s Score- Taxonomy	53
Table 5.3 Results For Time Based Evaluation Of Runtime Of Proposed Technique.....	54
Table 5.4 Results For Quality-Based Evaluation Using Silhouette’s Score- Taxonomy Generation	56
Table 5.5 Results For Quality-Based Evaluation Using Davies-Bouldin’s Score- Taxonomy Evolution.....	57
Table 5.6 Results For Time-Based Evaluation-Taxonomy Evolution.....	58
Table 5.7 Time Taken To Evolve Taxonomy Using Different Cores of Apache Spark	59
Table 5.8 Results For Time-Based Evaluation – Running Time Of Algorithm On Apache Hadoop Vs Apache Spark.....	61

Table of Figures

Figure 1.1 Data Generation Over The Years	1
Figure 2.1 Basic Process of Taxonomy Generation.....	10
Figure 2.2 Non Incremental Taxonomy Generation	15
Figure 2.3 Incremental Taxonomy Generation	17
Figure 3.1 MapReduce Function.....	28
Figure 3.2 Apache Hadoop Ecosystem.....	29
Figure 3.3 HDFS Architecture	30
Figure 3.4 Introduction of YARN in Hadoop version 2.0	31
Figure 3.5 Hadoop WordCount Program.....	32
Figure 3.6 Apache Spark Ecosystem	33
Figure 3.7 Resilient Distributed Datasets	34
Figure 3.8 Spark Streaming	35
Figure 3.9 MLlib – Machine learning Library for Apache Spark.....	36
Figure 3.10 Spark Word Count Program	36
Figure 4.1 Data Preprocessing Pipeline	39
Figure 4.2 Term Frequency.....	40
Figure 4.3 Inverse Document Frequency	41
Figure 4.4 Term Frequency – Inverse Document Frequency	42
Figure 4.5 Implementation of TF-IDF using HashingTF on Apache Spark.....	43
Figure 4.6 Cosine Similarity Calculation.....	44
Figure 4.7 Hierarchical Clustering Dendogram Produced from Ward’s Clustering Method	45
Figure 4.8 Centroid Calculation.....	46
Figure 4.9 Newick Tree for the Taxonomy T_{gen}	48
Figure 4.10 Evolution of Trees using NJMerge.....	49
Figure 5.1 Silhouette Score for Proposed Technique and TaxGen.....	53
Figure 5.2 Davies Bouldin for Proposed Technique and TaxGen	54
Figure 5.3 Time To Generate The Taxonomy Proposed Technique Vs Taxgen	55

Figure 5.4 Silhouette Score for Proposed Technique and TIE 56
Figure 5.5 Davies Bouldin for Proposed Technique and TIE..... 57
Figure 5.6 Time Taken To Evolve Taxonomy Using Proposed Technique and TIE 58
Figure 5.7 Time Taken To Evolve Taxonomy Using Different Cores of Apache Spark 60
Figure 5.8 Results for Time-Based Evaluation – Running Time of Algorithm on Apache Hadoop
Vs Spark..... 61

Chapter 1. INTRODUCTION

This chapter is an introduction to the motivation behind this research work. It describes the research problem and discusses the research objectives that are formulated in order to fill the research gap. The applicability of the research work is also discussed in this chapter. The chapter is organized as follows: Section 1.1 presents the discussion on motivation, Section 1.2 discusses the research problem, Section 1.3 presents the research objective, Section 1.4 presents the applicability of this research work and Section 1.5 presents the organization of this research document.

1.1 Motivation

During the past two decades, communication using the electronic media has gained significant popularity in today's world and has acquired a dominant role in developed societies. Electronic media provides many services such as World Wide Web (WWW), electronic social networks etc. Data is being produced in a large volume on daily basis using these services [1]. These services are used as primary communication and entertainment tool. IBM¹ has published data stats report on annual data generation in year 2017. The report outlines that 90% of all the data has been generated in last two years. The world is going through massive data revolution.

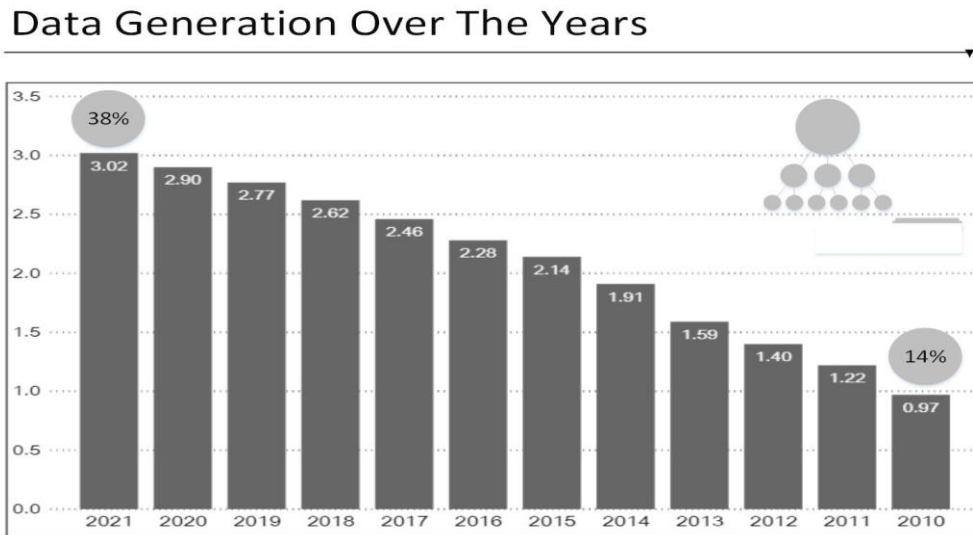


Figure 1.1 Data Generation Over The Years

¹ <https://www.ibm.com/investor/article/ibm-provides-historical-data-04042019.html> (Last viewed on 14-05-2020)

The Figure 1.1 Data Generation Over The Years, depicts the data generation by the year 2021, statistics are taken from Statista².

Electronic publishing, digital scientific libraries, ebooks, emails, blogs etc. are few of the applications where there is a need to manage, process and manipulate large volumes of unstructured textual data. In order to draw useful information from this data, it should be processed and transformed. This information needs to be organized into a structured form, like a taxonomy. A taxonomy is a hierarchical structure that organizes the given data in parent-child relationships [2]. Taxonomy effectively categorizes and organizes the data [3] by providing standardization in case of exchange of information. Taxonomy is used for knowledge management [4]. Taxonomy arranges information in a hierarchical structures that makes the search for information and navigation easier [5]. Data analytics, text mining and information retrieval all uses taxonomy as their knowledge base [6, 7, 8, 9, 10]. If a taxonomy is generated manually it becomes time consuming and complicated ; this brought the need of a process that generates taxonomy automatically. Taxonomy generation is of two types: incremental and non-incremental. Non-incremental taxonomy generation technique regenerates taxonomy from scratch on introduction of new documents into the system. Kashyap et al [11] proposed taxonomy generation technique that uses Principal Direction Divisive Partitioning (PDDP) approach [12] to generate taxonomy. Espinosa-Anke et al [13] proposed a conditional random field classifier for taxonomy generation. Velerdi et al [14] proposed a graph-based technique for taxonomy generation. All these techniques successfully generate taxonomy but at the time of introduction of new textual data into the system, they regenerate the taxonomy from scratch to get the updated taxonomy and therefore they are non-incremental in nature. This process is time exhaustive when dataset is very large. There came a need of technique that builds a taxonomy on top of existing taxonomy on introduction of new documents into the system. Such a technique is called incremental taxonomy generation or can be termed as taxonomy evolution. AdaptTaxa [15] uses a supervised incremental taxonomy generation technique. J. Yao et al [16] and Marcacini et al [17] incrementally generates taxonomy for dynamic tag space and terms respectively. Irfan et all [18] proposes an incremental taxonomy generation technique for updating an existing taxonomy.

Existing techniques for automatic taxonomy generation are able to produce taxonomy for small to large datasets. However, the data now a days is produced is in massive volume and keeps increasing its growth rate. With the emerging trend of big data and the cloud computing, the data is being produced from varying sources [19] that is stored and processed electronically and automatically [20]. The three Vs of big data are defined as variability, velocity and volume [21]. The massive data produced from multiple sources is defined as the Volume. The fast flowing nature of this voluminous data is referred to the Velocity. Data is generated from multiple sources this is referred as Variability. There has been an emerging need to organize this big data into a structure for drawing useful information from it otherwise it will lead to missing new directions,

² <https://www.statista.com/>

ideas and insights from this data [19]. Big data is fast evolving and emerging. A taxonomy generated for big data should represent the underlying data and changing theme of data. When this taxonomy is evolved, again it should reflect changes that has occurred in data. A massive portion of this voluminous, fast-evolving and varying-natured big data belongs to the category of unstructured textual data such as scholarly articles, technical reports, organizational policy documents, etc. [22]. The processing and input-output capacity of single machine cannot handle this growth of big data. Due to this growing data volumes many applications are moving towards parallel data processing. In order to handle this big data, computations need to be scaled out across parallel and distributed environments. Several challenges are faced when dealing with the cluster environment. The first challenge that there is a need to re-write the application that can run in a parallel environment. Google presented an algorithm for parallel and distributed computing called MapReduce [23] that divides a task at hand into number of smaller map and reduce jobs. MapReduce is a programming paradigm that works by decomposing the problem into multiple map and reduce tasks. This not only decreases the time to complete a job but also increases its efficacy. In order to organize and arrange the big data of unstructured documents the incremental taxonomy generation algorithm needs to be written for a distributed and parallel environment. In this work, we have developed a novel technique of incremental taxonomy generation based on MapReduce paradigm using Apache Spark. MapReduce decreases the running time of an algorithm and provides fault tolerance [24]. MapReduce environment can improve the scalability challenges of current incremental taxonomy generation techniques. The proposed technique generates and evolves the taxonomy in a much smaller time as compared with the existing techniques with better hierarchical quality.

1.2 Research Problem

The taxonomy generation can be performed either non-incrementally or incrementally. A non-incremental technique regenerates a taxonomy, every time new documents are introduced into the system where as incremental taxonomy generation technique on introduction of new documents into the system updates the existing taxonomy by adjusting the newly introduced documents. The running of the entire process of taxonomy generation as in the case of non-incremental taxonomy generation, is time consuming and computationally complex. Many existing techniques updates the taxonomy in such a fashion. Techniques such as TaxGen [25], TaxaMiner [11], Taxolearn [26], OntoLearn [14] updates taxonomy in non-incremental fashion in order to show the changes that has been made to the underlying data.

An incremental taxonomy generation technique evolves or updates the existing taxonomy by adjusting the newly introduced documents into the system without regenerating the taxonomy from scratch. The process runs very limited number of steps for taxonomy generation process. It runs the steps only for newly arrived text documents. This makes an incremental approach of taxonomy generation a better choice when it comes to time efficiency. Very few techniques have focused on incremental taxonomy generation. Techniques such as EvoTaxa [16], AdaptTaxa [29], JHTCTaxa [17], TIE [31], focus on incremental taxonomy generation. The technique EvoTaxa

[16] technique is specifically designed for tag data. AdaptTaxa [29] focuses on incremental taxonomy generation technique for unstructured textual data. It adopts a supervised approach that requires training data. The technique IHTCTaxa [17] uses an unsupervised hierarchical clustering-based approach by adjusting the newly introduced documents using IHTC algorithm. This algorithm supports a 2-ary taxonomy where a taxonomic node can have two children nodes at maximum. In real life especially in case of big data of unstructured textual documents a 2-ary taxonomy does not portray the true nature of the data. Also the technique needs to be unsupervised as data now a days is generated from different dimensions and those dimensions cannot be always predicted through training data. The taxonomy representing real-world dataset is usually n-ary in nature. TIE [31] is an algorithm incremental taxonomy generation algorithm that updates the taxonomy in the presence of new documents. TIE [31] uses hierarchical clustering approach to find the nearest cluster for a newly arrived document. Based on the cohesion and tightness scores the new document is grouped in to the cluster. All the techniques of taxonomy generation discussed here, be it non-incremental or incremental, although produce good quality taxonomy, however, lacks the focus on rapidly increasing, voluminous and varying natured big data. Since we are living in an age of big data, so there is a need of algorithms and techniques that are efficient and scalable to process this kind of data. Furthermore, none of these methods focus on the use of the concept of parallelization for making an efficient and scalable algorithm for incremental taxonomy generation. There is a need of incremental taxonomy generation technique that handles the fast arriving big data of documents and arranges it in a hierarchical structure and also on the next input stream of data it evolves the existing hierarchical structure by adjusting the new data stream. To deal with the fast arriving big data, the technique needs to run on MapReduce [23] paradigm so that the running time of incremental taxonomy generation process can be reduced and to improve the scalability challenges of current incremental taxonomy generation techniques.

1.3 Research Objective

In order to solve the above-mentioned problems, following research objectives were formulated:

- 1) To solve the scalability issues of current clustering-based incremental taxonomy generation algorithms
- 2) To propose/improve a scalable algorithm to update/evolve taxonomy in presence of a new document
- 3) To explore the comparison of Apache Hadoop and Apache Spark using the proposed technique and find out which big data tool is better suited for a clustering problem like incremental taxonomy generation.
- 4) To validate the significance of the obtained results utilizing the statistical and analytical methods

1.4 Applicability of the Work

Hierarchical structure of taxonomy can be utilized in many different fields. Some of the application areas where this work can be of utmost importance are discussed here:

1 Browsing And Searching Of Information Over The World Wide Web

The search engines deployed on the Web provides the results based on ranking. Internet search has become difficult due to the abundant amount of data present online. Taxonomy is hierarchical structure that arranges information in a hierarchy. If a search engine is based on a taxonomy it makes the search operation fast and efficient. If the hierarchical structure of the taxonomy is used for web search it will assist user to make sense of relationships between various terms and concepts. The work presented in [33] studies the user intellectual behavior in performing navigation. The work suggests that the use of taxonomy can be useful in such search operations. The work presented in [34] has presented the use of taxonomy to assist browse and search operation. As the data now days is fast evolving and of changing nature. The search operation on this big data is even more difficult and cumbersome. If this fast evolving Big Data is arranged in a taxonomy, it will make search operation faster and efficient.

2 Customer Support

Customer support portals provide customers with easy access to support and also the self-service options. A customer support portal has options for self-service ticketing, knowledge bases, communities and more. If back-end of such a customer support is based on taxonomic hierarchical structure; it will make the portal much more user friendly. The taxonomic structure will enables the customers to quickly navigate to the information that is needed for resolving their issues. The portal can also provide a list of knowledge base articles, arranged in a taxonomic structure, in order to support the tickets section. This will be beneficial in a case that the customer may resolve his query without even raising the ticket.

In customer service portals, the service agents bridge the gap between customers' vocabulary and business terms. Customer's vocabulary might not be suitable to use it for search. The work presented in [35] extracts a taxonomy from textual data, and uses it as a means to re-construct the customer's vocabulary. This taxonomy basically takes input of any search terms given by the user and converts it into terminology that would be actually used by a professional customer service agent. This kind of work can be used in organizations which are minimizing the human effort in customer services. The work presented in this thesis can be used in such a scenario.

3 Website Organization:

A website is organized using information architecture. Web pages can be ordered or tied to a specific taxonomic elements. Multiple taxonomic elements can be used to create a microsite. A microsites focuses on a particular product Visitors may browse website by topics or categories. This approach is similar to Amazon.com. Visitors are able to browse products on amazon website on the back-end there is a taxonomy built against each product, keyword or a topic. Taxonomies are used at back-end for searching the said

website. When the content is modeling correctly it makes it easier for the visitors to search and find the right information quickly. The work presented in [32] presents a taxonomy for organization of web-content. The work presented in this thesis can be used for a back-end engine for such a website.

4 Content-as-a-Service

Content building and its re-use plays an important part in marketing. In marketing content is created once but is re-used. This is an essential in order to provide a consistent customer experience. Content-as-a-service is basically an ability to create content and then deliver it to multiple websites and portals. Content building, its re-use and its use as content service cannot be done without a well-defined taxonomy. SiteCore³ provides a platform to use taxonomies in order to perform classification of marketing activities and campaigns. Taxonomic tags can be used in taxonomies used for marketing data that can assist user in tracking the website activities in depth thereby providing the user with deeper insights about the content.

5 RESTFul API

A taxonomy can be used as a base or a back end engine of a RESTFul API. In a RESTFul API, content is pulled and pushed into different applications. The REST API basically sends requisition for the content that needs to be populated on the interface. The content is requested based on content type and other content metadata. A correctly designed taxonomy can assist in getting the right content into publishing channel. RESTFul API can be used as a base of different kinds of systems such as Web content management system, a financial application or a mobile application. This work can be used in building such an application. WordPress⁴ is an example of such a RESTFul API that utilizes taxonomies either by listing terms in a taxonomy and listing all of the tags used on the current post.

6 Use of Taxonomy E-Commerce

E-commerce websites are great examples of taxonomy utilization. In an E-commerce provides quick access to the products/services to the visitors. In order to show related products for a specific product, taxonomies can be utilized. As taxonomy is a hierarchical structure, different products can exist as parent, co-parents and children. Similarly for any kind of search operations related products, websites and blogs can be shown using the similar approach using a taxonomy. BigCommerce⁵ presents a product taxonomy. This

³ <https://doc.sitecore.com/users/81/sitecore-experience-platform/en/marketing-taxonomies.html>

⁴ <https://wordpress.org/>

⁵ <https://www.bigcommerce.com/blog/product-taxonomy/#how-product-taxonomy-can-boost-sales>

product taxonomy not only assists in organization of products but also helps in find the buying potential of a customer. Similar to BigCommerce5 the presented research work can be adopted/used for E-commerce websites.

7 Use of Taxonomy in Sentiment Analysis

The area of Sentiment analysis is vast and is growing at a fast pace. Many nomenclatures have emerged without any organization pattern. The work presented in [36] proposes a hierarchical taxonomy for the problems and methods in the Sentiment Analysis field. This work allows indices the existing concepts in the Sentiment Analysis field and makes the navigation easier. It also facilitates the operations related to previously published research. This proposed work can also be used to create such a domain specific taxonomies.

8 Use of Taxonomy in Spam Filtration

With increase in Big Data and its volume the number of unwanted emails i-e spam has also increased. This has brought the need in creation of a system that effectively filters the unwanted emails. Spam filtering, is a process which separates the spam messages from legitimate messages. The work presented in [37] builds a taxonomy that assists in spam filtration.

9 Use of Taxonomy in Genre Classification

Genre classification, taxonomies can be used for determining the genre of a text. The work proposed in [38] uses taxonomy for classification of music genre.

10 Use of taxonomy in predicting future trends

Taxonomy can be used for discovering future trends and emerging ideas, this work can be utilized to discover novel insights from data. The work in [39] proposed a domain-specific taxonomy for scientific community. The work proposed in [13] used the taxonomy inorder to find out the future research directions.

1.5 Thesis Organization

This research document is organized as follows:

Chapter 2-Literature Review: Existing techniques for non-incremental taxonomy generation, incremental taxonomy generation are discussed in detail and also the hierarchical clustering algorithms for big data. The chapter throws light on basic taxonomy generation process that has been used by existing techniques and algorithms.

Chapter 3-Background and Preliminaries: The preface of big data techniques and tools used in this research work are discussed in this chapter. The chapter throws light on MapReduce algorithm and discusses it in detail. This chapter also discusses the big data environment; Apache Spark and Apache Hadoop that are used in this research work. This chapter also explores that why Apache Spark was chosen as the base tool for the implementation of this work.

Chapter 4-The Proposed Technique: This chapter discusses the proposed technique that has been developed for the generation and evolution of taxonomy for the big data of unstructured text documents. This technique runs on Apache Spark on its Python implementation that is Pyspark.

Chapter 5-Evaluation: Comparison of the proposed technique with the existing non-incremental and incremental taxonomy generation techniques were performed in this chapter. The comparison is done on the basis of time to generate and evolve a taxonomy and clustering quality parameters. This chapter also explores the comparison of Apache Hadoop and Apache Spark using the proposed technique.

Chapter 6-Conclusion: Findings of the research and contributions are discussed here. Moreover, its limitations and future work is also discussed in this chapter.

Chapter 2. LITERATURE REVIEW

A taxonomy generation process builds a meaningful hierarchical structure from the corpora of unstructured text documents [3, 4]. The process of taxonomy generation is of two types namely: incremental taxonomy generation and non-incremental taxonomy generation. A non-incremental taxonomy generation technique [11, 25, 26] re-builds the taxonomy on top of existing taxonomy whenever new documents are introduced into the system. Whereas, an incremental taxonomy generation [15, 16, 17] technique evolves the existing taxonomy to adjust the newly introduced text documents and the process can also be termed as taxonomy evolution. Be it an incremental taxonomy generation process or non-incremental taxonomy generation process, a basic taxonomy generation algorithm is used in order to build the initial taxonomy. So in this chapter approaches adopted by different techniques for basic taxonomy generation will be discussed in detail. Moreover, to identify the research gap, incremental and non-incremental taxonomy generation techniques will also be explored. This work focuses of on a clustering-based taxonomy generation techniques, so different clustering-based techniques especially for handling the big data will also be discussed here in this chapter.

The organization of this chapter is as follows: Section 2.1 visits the basic process of taxonomy generation that has been used by different research works so far. Section 2.2 discusses the various techniques and works that have performed non-incremental taxonomy generation and incremental taxonomy generation. Section 2.3 explores the clustering algorithms specifically built in order to incorporate the properties of big data.

2.1 Basic Process of Taxonomy Generation

The basic process of taxonomy generation generates a hierarchical structure from the corpora of textual documents. There are many techniques out there which have been successfully generating a basic taxonomy. Most of these techniques use similar steps and methods for the generation of taxonomy. The commonly used steps of taxonomy generation are: Data Pre-processing, Data Modeling, Hierarchy Formation and Node Labeling. Different works have used different techniques in order to perform these steps. This section discusses the common steps of taxonomy generation and also discusses the different perspectives that have been used by different techniques in order to perform these basic taxonomy generation steps. The Figure 2.1 Basic Process of Taxonomy Generation

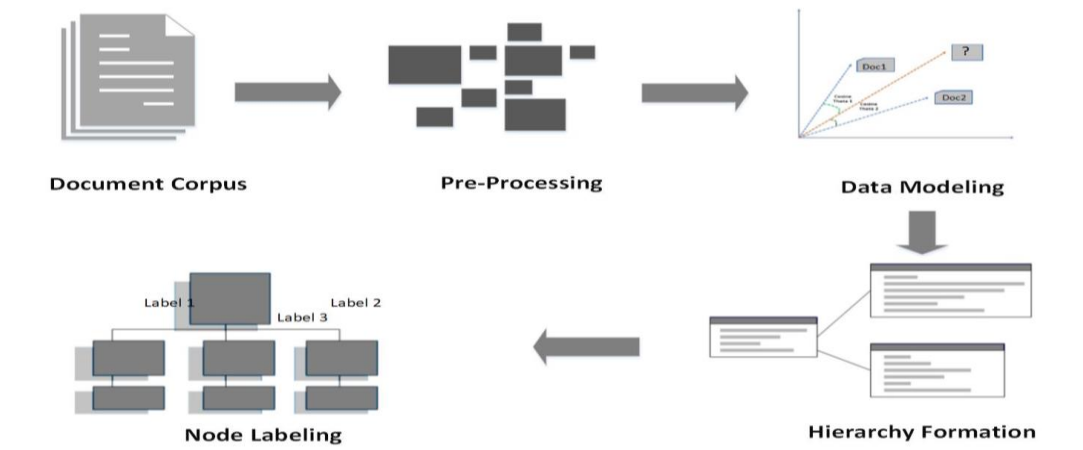


Figure 2.1 Basic Process of Taxonomy Generation

The first step in the process is ingestion of document corpora into the system, next preprocessing is performed on the textual data. Based on the creation of data model the hierarchical structure is created and finally the structure is then labeled.

2.1.1 Data Preprocessing

Data Pre-processing means bringing the given text corpora into a format that is predictable, analyzable and can be used by the algorithm. Data pre-processing includes the processes that makes the data ready for processing [51]. cleansing process of data preprocessing removes irrelevant details from the text. Different Taxonomy Generation techniques have used different methods to perform Data Pre-processing such NLP [52] Based Approach and Non-NLP Based Approach. The details of these techniques are discussed in this section.

1 Natural Language Processing based approach

Natural language processing (NLP) [52] enables machines to understand, interpret, manipulate, and to draw meaningful information from human language. Some of the techniques use basic NLP techniques for Pre-Processing, where as some techniques have used advanced NLP techniques for Data Pre-Processing. The techniques that uses the basic NLP techniques for preprocessing use Tokenization, Stop Word Removal, Stemming and parts of speech tagging as the part of data cleansing process. Initial term extraction is done by using basic natural language processing techniques in the work of [53, 54, 9, 2, 57, 58, 59, and 20]. The techniques presented in [61, 62, 63, 64] use some latest techniques for initial term extraction. For initial term extraction basic NLP processes and tools work well. The extracted initial terms maybe further refined by using advance NLP techniques. In order to extract terms that are not noisy and irrelevant Entity recognition and Noun-phrase extraction are used. Entity recognition refers to the fact that the names of important people, organizations and places are actually the most relevant

information in the data. This technique was used by Muller et al [25]. Noun phrase extraction refers to the fact that the phrases in which a noun is used as a principal phrase is the most important piece of information that is extracted from the data. Kashyap et al. [11] and Dietz et al. [26] used this technique. Using lexico-syntactic patterns technique relevant and important terms can also be extracted from given data. Cimiano et al. [65], Neshati et al. [66] and Ponzetto and Strube [67] have used this technique.

2 Non-NLP based approaches

Non-NLP based technique for data preprocessing involves two types of techniques namely data collection and data representation. The work presented in [68, 16] use data collection for preprocessing. In order to generate a taxonomy from raw data, data representation methods are used. The techniques presented in [69, 70] are the examples non NLP based techniques. For relational datasets Li and Anand [69] generated a taxonomy for relational data sets and emphasized that there is a contrast between taxonomies being generated from traditional text corpora and relational datasets. Taxonomy generation technique proposed in [70] generate taxonomy from the linked dataset. The linked data structure provides information about class types and object types.

2.1.2 Data Modeling

Once Data Pre-processing and cleansing has been done the next stage is of Data-Modeling. The text documents or the raw data is to be represented as a model and then built in to a hierarchy in later stages. There are different techniques used for data modeling for Taxonomy Generation such as Vector Based Approach, Concept Based Approach etc. The details of the techniques and the works that has used them are mentioned below:

1 Vector Based Approach

Many taxonomy generation techniques use vector space modeling to build a data model. Vector space modeling, determines the occurrences of important and relevant terms. The terms here are actually received from data pre-processing stage [71]. In this approach the entire data set is represented in the form a vector. This vector represents the document terms relationship. In order to match the similarity of the particular document with other the VSM model is then used. Taxonomy generation techniques proposed in [25, 53, 11, 54, 68, 9, 70, 61] uses vector space technique for data modeling.

2 Concept based approach

Concept based approach builds a model using external sources and domain specific measures. It is more precise in data representation and is also semantically better than the vector space model [72]. The techniques [5, 73, 58] use external sources for building the data model. Sanchez and Moreno [5] built a taxonomy for web documents for a peculiar domain using the keywords. Liu et al. [73] suggest that a more effective taxonomy can be constructed for data that belongs to continuously changing domains by using set of

keywords. Keywords are identified using the external knowledge source [27] i-e by using a search engine. Then a bag of words model was created using the results and concepts were identified for building the data model. Techniques discussed in [67, 10] also use concept based approach for data modeling. For the. Dietz et al. [26] suggested that for construction of domain specific taxonomy the domain pertinence and domain consensus [14] can be used to identify domain specific concepts that are present in data. Word sense disambiguation [39] was applied in order to determine the true context. The work presented in [60, 57, 14, 74] also use the domain specific measure for data modeling.

2.1.3 Hierarchy Formation

After the Data Model has been created the next step is of Hierarchy Formation. In this step, relationships between documents are identified using the modeled data. For making hierarchical relationships different approaches are used such as Clustering Based Approach, Graph Based Approach, and Rules Based Approach etc. The details of the techniques and the research works that has used these techniques are discussed in this section.

1 Clustering based approach

Agglomerative hierarchical clustering and divisive hierarchical clustering are the two basic methods of hierarchical clustering used by the current techniques. The agglomerative hierarchical clustering starts with all the text documents in an individual cluster. The technique merges the clusters using a similarity criteria. Similarity criteria can be cosine similarity, Jaccard similarity and Euclidean similarity [75]. The techniques presented in [25, 5, 26, 73, 17, 66] use agglomerative hierarchical clustering in hierarchy formation step of taxonomy generation. In divisive hierarchical clustering all data objects are considered in one cluster, then with each iteration they are divided. This division is based on some similarity or distance criteria, like Cosine measure [75]. The technique proposed in [11, 30] for the formation of hierarchy uses a divisive method of hierarchical clustering. Some techniques use Divisive Hierarchy Clustering and HAC both. The work in [77] compares divisive and agglomerative methods of hierarchical clustering and proves that the taxonomy obtained through agglomerative hierarchical clustering is better as compared to that of divisive hierarchical clustering. Taxonomy generation techniques in these papers [54, 69, 73, and 64] presents a merger of divisive and agglomerative clustering techniques. Chuang and Chien [54] presented the HAC+P that initially generated a hierarchical structure and agglomerative hierarchical clustering was used. Once hierarchical clusters are portioned a multi branched hierarchical structure is then obtained. Li and Sarabjot [69] clustering algorithm that groups together similar data objects using divisive and agglomerative clustering both. Liu et al. [73] and Ceesay and Hou [64] used Bayesian rose tree algorithm proposed in [78] to build a hierarchy. This algorithm is agglomerative clustering algorithm that has multiple branches. Using the conditional probability similar items are grouped together and produces a hierarchical structure.

2 Graph based approach

A taxonomic structure can be considered as a tree also. A graph is also a tree that having no cycle. So we can say a taxonomy is also a graph. The approach reflects on the fact that a graph based method can also be used to generate and evolve a taxonomy [79]. Each node in a taxonomy can be considered as a vertex in a graph. A graph based method for taxonomy generation, labeling is not required. Taxonomy generation techniques such as [80, 10] use the algorithms from fundamental graph theory. Qi et al. [80] Presented a graph-based algorithm for adjusting a taxonomy in order to assist the users in searching the web content. Camina [10] experimented and compared various graph algorithms for generating taxonomy. Techniques using graph algorithms with heuristics. Some techniques [14, 63] generate taxonomy by combining graph based heuristics. Velardi et al. [14] generates taxonomy by extracting definitions from the web for given domain-specific concepts then trains the algorithm to extract the hypernym information from the searched definitional sentences. Then terms were connected in hypernym relations with each other to form a taxonomy. Taxonomy generation techniques proposed in [20, 28] also combine heuristics with the graph-based approach in to order to generate hierarchy.

3 Rules based approach

A few taxonomy generation techniques formulated rules in order to identify the hierarchical relationships and to generate a taxonomy. Considering the nature of the data, a rule based technique provides more control on the process of taxonomy generation. Rules can be based on external knowledge sources, formal concept analysis and sub sub-summption. The techniques presented in [81, 58, 62] use external knowledge for extraction of relationships. The work presented in [67] analyzed Wikipedia's semantic network and extracted the hierarchical relationships. [58] Incorporate multiple external knowledge sources: Wikipedia,. In [62] a technique is proposed that identifies the accurate hierarchical relationship between the terms. The algorithm contains a heuristics based hypernym detection system that extracts the final taxonomy, In [65] a taxonomy generation technique is proposed that formulates the hierarchical structure based on the formal concept analysis. [57, 74] generated taxonomy using the sub-summption rules. Conditional probability is used by sub-summption rules in order to identify the hierarchical relationships between terms

2.1.4 Node Labeling

In the stage of nodes labeling, the unlabeled nodes are then assigned the labels [20]. For clustering based approach taxonomy generation nodes labeling is important. In graph based techniques the hierarchical structure formed in actually the final taxonomy. Where as in a clustering based approach the hierarchical structure generated gives no information without labels. Heuristic Based Approach and Centroid based approach is mostly used for Node Labeling. Heuristic based approach is either done by using frequently occurring terms or by using external knowledge.

Commonly occurring terms have also been used by a few techniques as a label of that particular cluster. Frequently occurring terms in a cluster are used as node labels in the techniques presented in [64, 60, 74, 6, 55, 56, 84]. The techniques [85, 2, 26] have used external knowledge in order to label the cluster. The work presented in [85] for each term in the cluster the hypernym information was collected using WordNet. Then the common hypernyms were then used as the labels for the cluster. Paukkeri et al. [2] used a similar term in order to label the cluster using a reference taxonomy as an external source. Dietz et al. [26] performed label identification by first collecting hypernym information from WordNet to be used as labels for that cluster. The work presented in [86] used a reference taxonomy related to the domain for collecting hypernym information. Some techniques for taxonomy generation use, centroid based approach for assigning labels to the clusters. The technique utilizes the top terms of a cluster for labeling purpose. The work presented in [11] used the similar approach for labeling the nodes. Generalized labels were assigned to the nodes in order to further prune them. Generalized labels were taken from the children clusters. These labels were then assigned to the parent clusters. Work presented in [26] also uses the Centroid-based nodes labeling technique.

2.2 Types of Taxonomy Generation

Taxonomy generation is performed either incrementally or non-incrementally. This section will explore different research works that have used incremental taxonomy generation and non-incremental taxonomy generation.

2.2.1 Non-Incremental Taxonomy Generation

Some techniques perform taxonomy generation non-incrementally. A non-incremental taxonomy generation technique utilizes the basic process of taxonomy generation. It then works in a fashion that the moment new documents are introduced in to the system, the entire process of taxonomy generation is ran again from scratch in order to accommodate newly introduced documents.

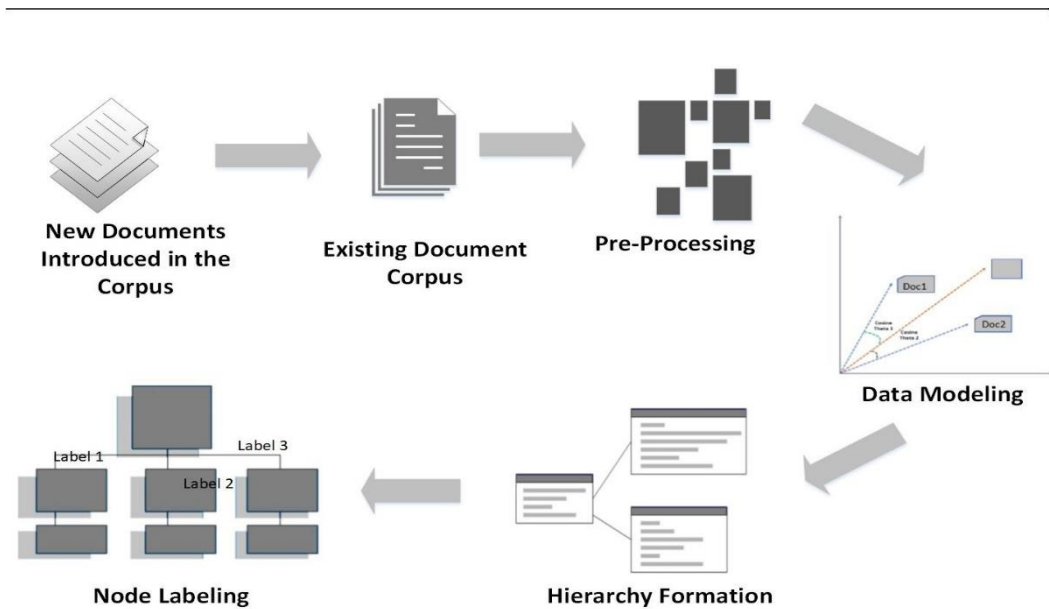


Figure 2.2 Non Incremental Taxonomy Generation

The Figure 2.2 shows the process of generating a taxonomy using non-incremental approach. It can be seen that on introduction of the new documents in to the system, the previous taxonomy is discarded and again all the steps of taxonomy generation are performed and a new taxonomy is built from scratch. This section explores the techniques and algorithms that have adopted non-incremental taxonomy generation strategy.

1 TaxGen

TaxGen [25] was presented as an automatic taxonomy generation algorithm for unstructured data. The algorithm was developed for building a taxonomy for unstructured news wires for the period of one year. The algorithm uses hierarchical clustering algorithm (HCA) for building the underlying structure for taxonomy generation. In the beginning of this process each document is a cluster itself. The clusters for the bottom are made first. Once the bottom clusters have been identified, the algorithm starts working upwards merging the similar clusters. Scalability is provided using this approach as it uses only a subset of document initially. A separate routing step is used to route the remaining documents to the respective categories. In order to evolve the Taxonomy this technique uses a topic categorization tool. Sample documents from each category is then used to extract the dictionary called category. These categories in case of already built taxonomy are the taxonomic nodes. These documents are then assigned to their respective categories based on the terms extracted from them. This algorithm is a supervised technique and requires training data.

2 TaxaMiner

TaxaMiner [11] was also an addition in the pool of existing techniques of taxonomy non-incremental generation. This technique uses the NLP techniques for pre-processing of the textual data set. Noun phrases are extracted from the documents using part-of-speech tagging and chunk parsing. Principal direction divisive approach and bisecting algorithm of K means is used to cluster the dataset. Taxonomy is extracted using cluster cohesiveness. For the re-generation of taxonomy the entire procedure of taxonomy generation is re-run and the previous taxonomy is replaced by a new taxonomy that gives an updated view of the data set.

3 TaxoLearn

TaxoLearn [26] is also a non-incremental taxonomy generation algorithm. It requires two corpus of documents. The first corpus selected is domain specific. The second corpora is of unrelated domain. To find out the related concepts the second corpora is used. For the construction of the domain taxonomy, noun phrases are extracted first. The relevant concepts of the domain are detected using the noun phrases. After this similarities is calculated between the selected concepts. On the basis of similarities the taxonomy is constructed. Hierarchy is built using hierarchical clustering algorithm [77]. The algorithm is unsupervised.

4 OntoLearn

OntoLearn [14] is actually a ontology construction technique that generates ontology for a category or a topic. Taxonomy is hierarchical structure based on the relationships in the data they useful for organization of information. Ontologies on the ground level are basically taxonomies. Ontologies are at a higher level as compared with taxonomies as they provide richer information, including information about the relationships among entities. The process of ontology construction process consists of document clustering, extraction of terms, extension of vocabulary of term and selection of relevant terms manually. In the first step the clusters of document are created. For document clustering Latent Semantic and K-Means clustering is used. A tool called TermExtractor is used for extraction of terminology from document clusters. Relevant terms are selected by TermExtractor using documents belonging to specific domain. For the extraction of terms different techniques are used. Names of the concepts are chosen on the basis of knowledge that is gained from the previous step. Correct relationships are identified using the Formal Concept Analysis [87] (FCA).

2.2.2 Incremental Taxonomy Generation

Some techniques perform taxonomy generation incrementally. An incremental taxonomy generation technique utilizes the process of incremental taxonomy generation as underlying technique. An incremental taxonomy generation technique works in a fashion that in-presence of new documents in to the system the process doesn't re-build the entire taxonomy from scratch;

instead new documents are adjusted in the existing taxonomy based on similarities to the existing document clusters. This makes an incremental approach for taxonomy generation time efficient and computationally less complex as compared with the non-incremental approach. The Figure 2.3 shows the process of incremental taxonomy generation.

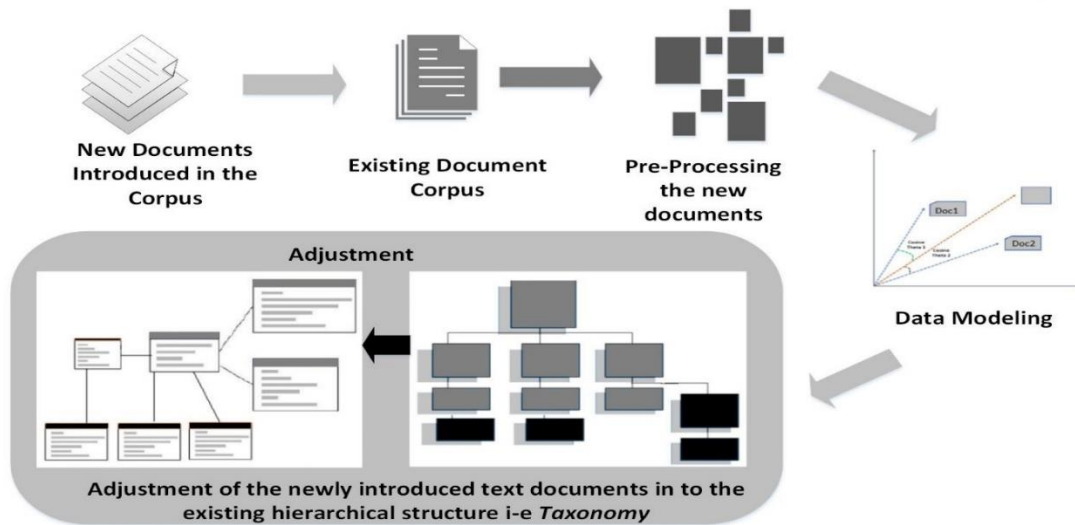


Figure 2.3 Incremental Taxonomy Generation

It can be seen in the Figure 2.3 that on introduction of new document in to the corpora the taxonomy generation steps such preprocessing and data modeling are performed only on the new documents and finally the documents are adjusted in to the existing clusters. This approach is promising for fast arriving voluminous big data as time to adjust new documents is far less as compared with rerunning the entire process from scratch. Some of the incremental taxonomy generation techniques are discussed below:

1 AdaptTaxa

AdaptTaxa [15] generates the taxonomy incrementally for group profiling problem. This technique generates the taxonomy whenever the changes in the underlying data occurs. The changes occurred in the data specifically reflect the changes in the group's interest. The problem of Taxonomy Evolution was transformed to graph based optimization problem. The technique uses the graph based approach to search and select the best possible taxonomy at a particular time from the hyperspace of many taxonomies. AdaptTaxa uses the supervised learning approach for generating and evolving the taxonomy. The supervised technique learns the hierarchical model from the hyperspace of the hierarchies. It chooses the hierarchy that best fits the taxonomy according to existing condition. Supervised techniques require the availability of training data. Adapt Taxa uses two types of approaches for searching the optimal hierarchy. The approaches are Greedy Approach and top down approach. The greedy approach probabilistically searches the best

possible hierarchy by finding the difference between hierarchical nodes and selecting the one giving the maximum likelihood increase. In top-down approach nodes at the top level are checked to find the best possible node to expand and adjust the pre-defined hierarchy

2 EvoTaxa

EvoTaxa [16] generated the taxonomy incrementally for particularly large collection of tags. In this technique association rules graph was generated. In the association rules graph the vertices are actually the tags based on support and confidence values these tags are connected. Support measures the frequency of co-occurrence. The confidence measures the conditional probability of a tag in presence of another and identifies the parent child relationship. The next step is graph based taxonomy extraction graph and association rule graph is given as input to this step. Manipulation on the association rule graph is done by the taxonomy extraction step. Only those associations are kept which don't add to noisy associations. For evolving the taxonomy four types of strategies are used namely local taxonomy evolution, historical taxonomy evolution, one step fusion taxonomy evolution and historical fusion taxonomy. Local taxonomy evolution regenerates taxonomy at every new time interval. Historical taxonomy evolution regenerates new taxonomy by combining values of the historical taxonomy's association rule graph with current taxonomy's association rule graph. One step fusion taxonomy evolution takes the previous step's taxonomy's association rule graph and fuses with the current's association rule graph. Historical fusion taxonomy Takes the historical taxonomies' association rules graph and fuses with the proceeding step taxonomy and current association rules graph. The technique successfully generated and evolves the taxonomy but it does so for tag data only. In real world we have data in unstructured form that needs to be arranged in a taxonomy.

3 IHTC Taxa

IHTCTaxa [17] uses unsupervised incremental hierarchical clustering approach to generate a taxonomy for unstructured textual data. IHTC (Incremental Hierarchical Term Clustering) algorithm that considers the problem of hierarchical clustering as online in contrary to the batch mode non-incremental hierarchical clustering, like HAC [76] and bisect K-means [30]. First frequently occurring k terms are extracted from the document collection. The co-occurring terms are then connected to each other and they form a co-occurrence graph. Shared Nearest Neighbor (SNN) measure is used to identify the similarity between terms in the co-occurrence graph [88]. Dendrogram is then formed by agglomerative clustering of similar terms. When new documents arrive, frequently occurring terms are extracted from it. The IHTC algorithm then calls up the UpdateGraph function to update the co-occurrence graph of newly arrived terms with the existing terms.

4 TIE

TIE [18] algorithm was as advancement in the domain of incremental taxonomy generation. The algorithm uses an incremental approach to generate a taxonomy on introduction of new data in to the system TIE incrementally updates the existing taxonomy. The algorithm saves time and complexity by adjusting the newly introduced documents into the system rather than re-building the entire taxonomy from scratch. The proposed algorithm can take input of an existing taxonomy generated from a clustering based automatic taxonomy generation technique. It can then incrementally evolve the input taxonomy by incorporating the newly introduced documents into the system .The algorithm was tested in comparison to regeneration technique. Results shows that taxonomy evolution generates/evolves a taxonomy in considerably shorter time as compared with a taxonomy generation technique. Results also showed that the lexical and hierarchical quality of the taxonomy generated through an incremental taxonomy generation technique or non-incremental taxonomy generation technique is comparable but the time complexity is sufficiently reduced when using incremental taxonomy generation technique.

2.3 Hierarchical Clustering For Big Data Sets

For effectively handling the big data, it needs to be tamed. Taming big data means converting the complex data in to usable form [90]. Huge volume of unstructured data is being generated from different resources and services like Social Networks, IOTs, cloud computing [91] etc. It is getting problematic to control this volume of information with every passing day. In order to make this large amount of data useable and searchable its needs to be arranged in some kind of structure. This big data can be clustered in to compact form that still reflects the underlying nature of the data i-e a taxonomy. Underlying technique for building a hierarchical structure in a taxonomy generation and evolution technique is mostly clustering based technique. Clustering techniques are very useful when unstructured data needs to be organized in hierarchy [89]. Clustering techniques basically groups the data based on its similar properties and content. A lot of research has been done in order to develop clustering to incorporate the three Vs of big data that is Velocity, Volume and Variety [21]. Various clustering algorithms [92, 93, 94, 95] have been developed in order to cluster the big data. These algorithm are built keeping in view the shape size, noise, dimensionality, computation of algorithm, shape of cluster [96, 97]. These techniques are divided into different classes such as Density Based, Partition Based and Model Based etc. This research work focuses on solving the scalability issues of clustering based incremental taxonomy generation algorithms. For this reason the previously developed algorithms for big data clustering were studied so they can be used as a base line for the hierarchical structure of the taxonomy. Discussion on these algorithms for big data clustering is presented below:

2.3.1 Hierarchical clustering algorithms

Clustering is performed using two approaches i-e Agglomerative (top-bottom) and Divisive (bottom- top). All the data points are considered individual cluster in agglomerative. The algorithm iteratively merges the clusters based on a similarity criteria. In divisive approach initially all data points are in one cluster and there are progressively divided as the algorithm goes on. There has been research done on hierarchical clustering algorithms for big data these algorithms are discussed in detail in this section.

1 BIRCH

BIRCH [100] is an incremental technique in which multi-dimensional metric data points are dynamically clustered .BIRCH is an hierarchical agglomerative technique in which the Clustering Feature (CF-Tree) keeps adjusting the quality of sub clusters in increments. The algorithm loads the data into the memory. Then constructs a CF Tree by performing one scan of the data making the next stages of technique faster and less sensitive. To perform global clustering the algorithm uses the pre-existing technique for clustering on CF leaves. The technique reassigns the data points to the closest centroid by performing multiple data scans.

2 CURE

Clustering algorithms mostly favors clusters with spherical shapes and similar sizes. These algorithms mostly are very sensitive to outliers. An algorithm named CURE [101] was proposed for clustering big data that is not sensitive to outliers. In this algorithm the divisive hierarchical clustering is used. This algorithm identifies the clusters that have wide variation in shape and size. The technique keeps merging the adjoining clusters till stopping criteria is met.

3 ROCK

Using clustering underlying patterns in the data can be identified. In order to deal with boolean and categorical data ROCK [102] algorithm was proposed. Conventional clustering algorithms use the distances between points. Concept of links that measures the proximity between the data pairs, is introduced in this work. This algorithm employs the links and not the distances when it merges the clusters. In the beginning all points are considered as a cluster and a heap is built for all of the clusters. On the basis of the criterion function a measure is calculated. The clusters having largest value of the criterion are merged. Detailed complexity results were presented for ROCK. The results from experiments show that ROCK not only generate better clustering quality and is also scalable.

4 Chameleon

For dealing with high and variable clusters techniques such as Chameleon [103] were developed. Chameleon gives good results for highly variable clusters using the

interconnectivity and closeness of clusters. Static model doesn't take in to account the information about the nature of individual cluster while performing the merging operation. Most of the other existing techniques uses the static approach. Chameleon algorithm keeps an eye on interconnectivity as well as closeness while merging similar pair of clusters. Two-phase algorithm is used for finding the clusters in the data set, in Chameleon technique. In the first phase, a cluster is formulated using partitioning approach. Several relatively small sub-clusters using the data points. This is done by using a graph partitioning algorithm to cluster the data items. During the second phase, clusters are formulated by continuous combining of sub-clusters. In this phase agglomerative hierarchical clustering approach is used for selection of pairs of clusters. Only such pairs of clusters are selected whose inter connectivity and closeness is within the threshold level. The clusters having highest inter connectivity and closeness are then merged.

5 ECHIDNA

In order to cluster network data ECHIDNA [104] technique was developed. The technique works by extracting the data from the network. This data consists of six tuple values consisting of categorical and numerical attributes.

Each record in the data then builds the hierarchical trees of clusters that is called CF-Tree. Combined distance function inserts the record in to the closest cluster. The combined distance is taken account for all the attributes of the CF tree. Once all the clusters are created the significant nodes then form the cluster trees. The formulated cluster trees are then compressed to generate a report from which information can be drawn.

6 SNN

Clustering basically depends on density and similarity distance. Clustering becomes increasingly difficult as dimensions increases. An algorithm SNN [105] was presented that works well for high dimensional data. Sum of similarities among the neighbor data points is used as similarity measure in this algorithm. Such an algorithm is successful in eliminating noise and builds clusters that are associated by non-noise points. This way outliers are detected and eliminated. Also solves the issue of finding clusters that are of various sizes, shapes and density. The algorithm works by building hierarchy from top to bottom. For set of points a proximity is maintained. Nearest neighbor approach is used to formulate the clusters.

7 CACTUS

CACTUS [45] algorithm was specially developed for clustering of the categorical big data. This algorithm is fast and is summarization based algorithm. CACTUS algorithm has two important characteristics one that it requires only two scans of the dataset and finds clusters in subsets of all attributes. This feature makes it suitable for a data set where number of attributes is very large. The algorithm builds a hierarchical structure and

generates maximum segments and clusters. Algorithm works by connecting the data points whose frequency is large. Clusters are formulated on the basis of attribute value pairs.

8 GRIDCLUST

This technique [46] discusses clustering a hierarchical clustering method that using a grid structure. The data set is partitioned into a grid structure. During this partition the topological distributions are maintained. Density values are calculated after assigning the data is assigned in to the grids. Then the grids are sorted based on their values. Most dense grid is considered as the cluster center. Clusters are formed using the remaining blocks using the neighbor search algorithm.

2.3.2 Density based clustering algorithms

In clustering algorithm that uses a density approach, the data objects are categorized into different types of points such as noise, border and core points. Based on the densities the core are connected to each other formulating the clusters.

1 DBSCAN

DBSCAN [108] presents a technique that uses three points namely core point, border point and noise point to form connectivity. The algorithm formulates a graph using set of points. An edge is created from a point in the neighborhood to another point in the neighborhood. If any of the core points are not left behind then it terminates the algorithm. The technique keeps running till clusters are formed incorporating all the core points.

2 OPTICS

OPTICS [109] is an extension of DBSCAN algorithm that also uses core, border and noise points. This technique is faster than that of DBSCAN. The algorithm takes in to considering the data points, and selects a core points if certain minimum number of points are found in the core distance.

3 DBCLASD

DBCLASD [110] is basically a technique for mining of very large datasets that utilizes a connectivity and application based clustering algorithm. Based on a query this technique constructs the set of candidates. If the distance between the set of candidates has expected distribution the data points stay within the cluster. A data point is considered as bad candidate if it doesn't follow the expected distribution. The algorithm continues till all the points for an expected distribution forms a cluster.

4 GDBSCAN

GDBSCAN [111] algorithm forms the clusters using the point objects and the spatial attributes. An attribute object P is chosen that gathers all the object densities that are accessible from P. Based on the neighborhood and minimum weight cardinality the

densities are calculated. Based on these densities of the points it forms the clusters. The process is ran over and over again until all the points are classified.

5 DENCLUE

The DENCLUE [112] algorithm uses kernel density estimation in order to build a model. Estimated density function defines a local maximum that in turn defines a cluster. The algorithm works in a fashion that the points belonging to the local maxima are put in to the same cluster. The algorithm takes so many unnecessary steps in the beginning of the algorithm to reach to a local maxima thereby it never converges to the maximum but reaches close to it. The algorithm also makes use of Gaussian kernels that adjusts the step size without bringing in an extra cost.

2.3.3 Grid based clustering algorithms

The Grid based works by partitioning the data set. Partitioning of the data set results in to a grid structure. On the basis of the grid structure the Clusters are formed. Subspace and hierarchical clustering techniques are used in grid based clustering algorithms. Grid algorithms are very fast processing algorithms as compared with the other techniques.

1 STING

STING [113] technique is akin to hierarchic BIRCH algorithm. A hierarchical grid is used to store spatial data in to rectangular cells. 4 child cells are created be portioning each cell. Probability is calculated for each cell whether it is irrelevant or relevant. Same calculation are applied on the cells that are relevant. In order to form clusters the regions of relevant cells are found.

2 Wave Cluster

The algorithm [114] works with big data sets for numerical attributes and those data sets which have multi-resolution. Detects outliers easily. The algorithm fits all the data points into a cell. The wavelet transform applied to filter the data points. After that the discrete Data points are filtered using the wavelet transform thereby applying the high amplitude signals to cluster interiors. High frequency is used to find the boundary of the cluster. In order to find sharp clusters and to eliminate noise, signals are then applied to the attribute space.

2.3.4 Model based clustering algorithms

Two approaches are used for model based clustering algorithms neural network and statistical approach. A few model based clustering algorithms are discussed in this chapter that focuses on the clustering of very large datasets.

1 PROCLUS

PROCLUS [115] algorithm uses medoids. The medoid technique is pretty similar to K – medoids clustering criteria. Algorithm initializes by considering selecting the data points

in random. Next, medoids of clusters are selected as data point and subspace of each medoid is defined. The next phase is the phase of refinement here in this stage the best medoids are selected from the set of medoids. Only those medoids are selected that has all the dimensions. Medoids are selected based on other medoids closest to the best medoids. Data points within a certain distance will form clusters.

2 ORCLUS

ORCLUS [116] algorithm uses non-axis parallel subspace for clustering. Three stages define the algorithm merge, assignment and subspace. Assignment phase assigns the data points to the nearest cluster centers. Sub space determination phase in the merge phase makes co-variance matrix for each cluster. Those clusters that have similar directions are then merged.

2.3.5 Partition Based Clustering Algorithms For Big Data

In partition based clustering algorithm [98], each data object is placed in one cluster. The clusters are then divided in to a number of partitions going through a number of iterations. Non-Convex shaped clusters are found using Partition based algorithms.

1 FCM Fuzzy CMEANS algorithm

FCM Fuzzy C-means algorithm [99] is basically uses K-means to partition the data in to cluster. The algorithm first calculates the cluster centroids, objective values and the fuzzy matrices. It computes the membership values that are basically stored in the matrix. The algorithm constantly checks the objective values in order to make sure that it stays between the limits with in the continous iterations. If the value of objective function is less than the stopping condition then the algorithm stops. Until the stopping conditions are met the process keeps running and keeps partitioning and clusters are formulated.

2.4 Critical Analysis

In this section, first the basic process of taxonomy generation was studied. Then the incremental and non-incremental taxonomy generation techniques were reviewed in detail. On arrival of new data in to the system, a non-incremental taxonomy generation technique re-runs the entire process of taxonomy generation from scratch and re-builds the entire taxonomy incorporating the new documents in to the system and replaces the old taxonomy with a new one. Re-running of the whole process of taxonomy is time consuming especially when we have fast arriving voluminous big data of unstructured text documents. An incremental taxonomy generation technique updates the taxonomy in presence of new documents, without re-building the entire taxonomy from scratch. Hence it runs a fewer steps for taxonomy generation in order to incorporate the newly arrived documents. This makes an incremental approach for taxonomy generation time efficient, computationally less complex and better suited for dealing with big data as compared with non-incremental taxonomy generation technique.

Non-incremental or incremental taxonomy generation techniques discussed here, produce good quality taxonomy. However, it was observed that these techniques essentially lack the focus on rapidly increasing, voluminous and varying natured big data. There is a need of algorithms and techniques that are efficient, scalable and capable of processing of big data in parallel fashion. To handle fast arriving big data in parallel, the technique needs to run on MapReduce [23] paradigm so that the running time of taxonomy generation can be reduced and the scalability challenges of current incremental taxonomy generation techniques could be solved. In this work, we in particular focus on clustering-based incremental taxonomy generation techniques.

New challenges of big data makes it difficult to apply conventional clustering techniques. Large data volume and time complexity of clustering algorithms lead to the problem of deployment of clustering algorithms for big data in order to obtain the results in reasonable time. Clustering algorithms dealing with big data are generally classified into two categories [117]: single machine clustering techniques and multiple machine clustering techniques. Clustering techniques for big data studied in this section are specifically designed for dealing with big data but to be ran on single machines. As discussed these techniques are divided into partitioning based clustering techniques, hierarchical clustering techniques, grid based clustering techniques, model based clustering techniques. All these techniques have their own advantages and dis-advantages. Partitioning based clustering technique has a disadvantage that it requires a pre-defined value of K parameter to be given by user for a clustering solution the value of K is often non-deterministic [119]. In hierarchical clustering technique once a stage is completed it cannot be un-done. All the hierarchical clustering algorithms suffer from the reason stated above [119]. As the density based clustering algorithms contains noise objects because they work in a fashion that the clusters are defined as dense regions separated by low density areas [118], thus are not suited for very large data sets. The clustering algorithms that are based on a model measures are slow and unsuitable for very large data set for a classification problem as they utilize the multivariate probability distribution. The size of the grid is usually much smaller than the size of the database. In case of highly irregular data distributions, using a single uniform grid might not be a good idea as a single uniform grid will fail to provide the required clustering quality and also is not able to fulfil the required time requirement [119]. New challenges of big data can be solved using multiple machine clustering techniques and obtain results in much smaller time. Distributed and parallel algorithms divides the data into various smaller data partitions and distributes them on different machines. This makes the overall running time of the algorithm smaller and increases its scalability. MapReduce algorithm is a job rationing algorithm designed for distributed execution of a task on a many servers that provides a good base for the implementation of such parallel algorithms for data clustering. The next section discusses the MapReduce environment and tools used for big data processing.

Chapter 3. Background and Preliminaries

This chapter discusses the technologies used in this research work. This chapter briefly discusses the MapReduce paradigm. The details and workings of MapReduce environments: Apache Spark and Apache Hadoop are discussed in this chapter. This chapter is organized as follows: section 3.1 discusses the preface, section 3.2 discusses MapReduce Framework, section 3.3 explores Apache Hadoop, section 3.5 discuss Apache Spark and section 3.5 explores the selection of big data environment for this research work.

3.1 Preface

The world has converted into an information society that highly depends on data [40]. Development of cloud computing technologies has led to the generation and processing of huge amounts of data [41]. This ever increasing data volumes has brought great values but at the same time has brought forward many challenges. The information systems produce large amounts of data every passing second. It looks like that the world has reached to the point where the current systems are overloaded with the data. To process such large amount of data, there is a need of systems that has an enormous capacity for storage and processing power. The processing capabilities of current systems is bounded because of hardware limitation and technologies, but the growth of the big data volume is unlimited [42].

Big data has high velocity, volume and comes in variety of information that requires new forms and ways of data processing in order to get effective insights from the data [21]. These insights and information can be useful for effective and efficient decision-making system that could benefit different companies. Most of this data is being generated in the form of unstructured data [43]. The unstructured data requires management and classification. The internet, World Wide Web (WWW), various social media applications such as Facebook, Twitter, Blogs , Wikipedia etc, are some of the biggest contributors towards the unstructured Big Data [44]. The unstructured data, is a collection of facts that holds a lot of information; this information can be made available if proper analysis is being done on the data. This data is growing with tremendous speed and volume. This high speed and massive volume of the big data has reached to a point where the world is experiencing a massive data revolution [42]. There is an increase in internet users by every passing day and that is one of the major source of increasing big data. This big data comes with challenges of data storage, analysis, visualization and processing. The processing capacity of existing systems was not enough to handle the large amount of data [45]. Hence, there was a critical need for tools that were capable of investigating and processing large scale data in order to acquire value from it. In this chapter, we will have a look at prominent tools and environment used for big data processing and analysis.

3.2 MapReduce

The existing techniques that deal with large amount of data such as data warehousing and database management systems, are not sufficient in order to deal with the ever increasing big data. As the data size is tremendous so it cannot be stored on a single machine [42]. Therefore, there was a need of technique that can process the data that is stored on different machines and still give one output. Hence, a cluster of commodity machines that are interconnected using a network were designed to store the data. Using the interconnected commodity machines the data comes in from various machine so the performance becomes a primary concern [46]. Also since the data will be coming in from different machines, the technique must be able to reconcile the failure of any machine, at any point of time without effecting the performance of overall system. Making a system fault-tolerant is a critical challenge. A parallel application has to encounter the challenges of load balancing, data parallelization, serialization, data distribution and scheduling program execution across multiple systems. Researchers at Google presented a new programming model called Map Reduce [23], which was able to solve the challenges of efficient processing of massive datasets using large clusters.

MapReduce solves the problems faced in parallelizing the data across the cluster of individual machines [23]. It provides a simple model for parallel computing by solving the problems of data partition, scheduling of tasks, handling of machine failure and inter-machine communications are reduced. MapReduce is a programming paradigm that works by decomposing the problem into multiple map and reduce tasks. Input is given in the form key/value pairs to mapper function. This input pair is then passed to reducer which are passed as an input to the reduce function. Associated with the intermediate key the reducer merges the intermediate values. The working of MapReduce is explained in detail below:

3.2.1 MapReduce Framework

The type of input to a MapReduce job is based on the application itself [23]. MapReduce job outputs a set of key and value pairs, that is composed by the Map function. The equation (1) below defines key and value pairs:

$$[(k_1, v_1), \dots, (k_n, v_n)]: \forall i = 1 \dots n: (k_i \in K, v_i \in V) \quad (1)$$

Here, k_i represents key for the i^{th} input and v_i implies the value for the i^{th} input. V is the domain of values. Key-value pairs are divided into of subsets and are rationed across the various nodes in a cluster to process. This is done by a Map function. The intermediate results are also generated in the form of key and value pairs. The map function is given below:

$$Map: K \times V \rightarrow L \times W \quad (2)$$

$$(k, v) \rightarrow [(l_1, x_1), \dots, (l_r, x_r)] \quad (3)$$

Where, L and W are key and value attributes. They also represent the intermediate key-value pairs. During the map phase, each single key-value input pairs: k, v is graphed into many key-value pairs: $[(l_1, x_1), \dots, (l_1, x_r)]$ using the same key, but distinct values. Key-value pairs obtained as a result in this phase are used as the inputs for reduce functions. The reduce function is defined as:

$$Reduce: L \times W^* \rightarrow W^* \tag{4}$$

In the reduce function, all the mean results are grouped using the same key. This aggregation is performed generate the intermediate results with the identical key values. The result is sorted and all the key values pairs are brought together so that it can be processed. This process if done when $L \times W^*$ is processed, $L \times W^*$ is generated. These are basically the inputs to the MapReduce functions. The intermediate results with the same keys: $L \times W^*$ are mapped into a new result list W^* . Equation (5) expresses the MapReduce function.

$$MapReduce: (K \times V)^* \rightarrow (L \times W)^* \tag{5}$$

Many map and reduce tasks are performed on various machine individually in order to visualize parallelization. Apache Hadoop and Apache Spark are prominent big data processing environments that uses MapReduce algorithm for processing and analyzing the data.

MapReduce Function

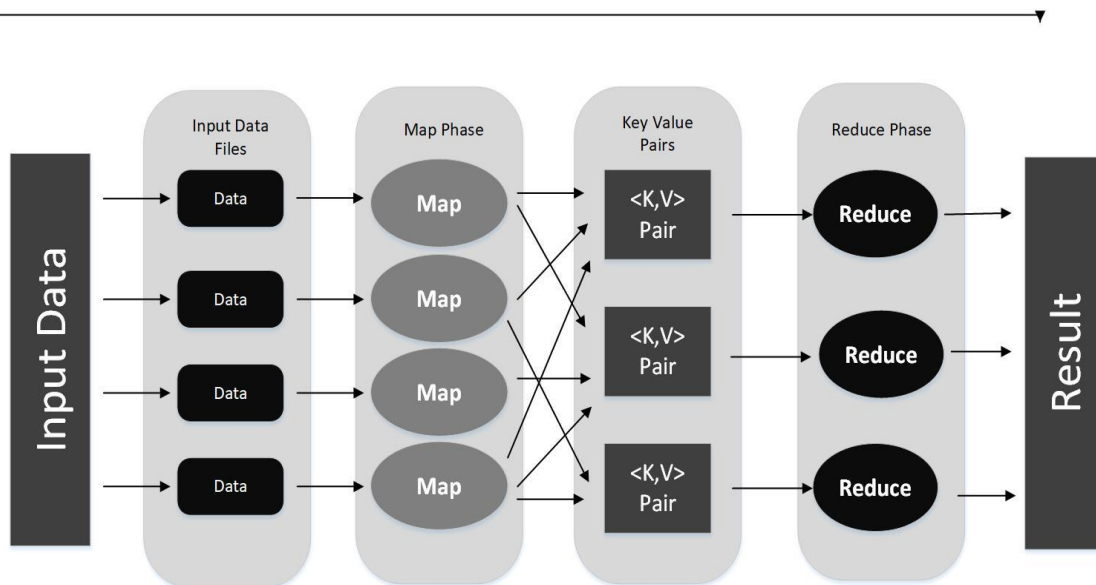


Figure 3.1 MapReduce Function

Figure 3.1 MapReduce Function shows execution pattern of a MapReduce job including its pattern of Map and Reduce and its division of the Job into key and value pairs. The next section of this chapter discusses Apache Hadoop and Apache Spark in detail.

3.3 Apache Hadoop

Hadoop is a framework based on MapReduce [23] algorithm. The framework is able to write applications that computer large amounts of data in-parallel fashion on large clusters. The data is usually in the size of multi-terabyte. The size of the cluster is of thousands of nodes. Hadoop provides efficient, reliable and fault tolerance system for processing of big data. There are many different tools and products in Hadoop Ecosystem.

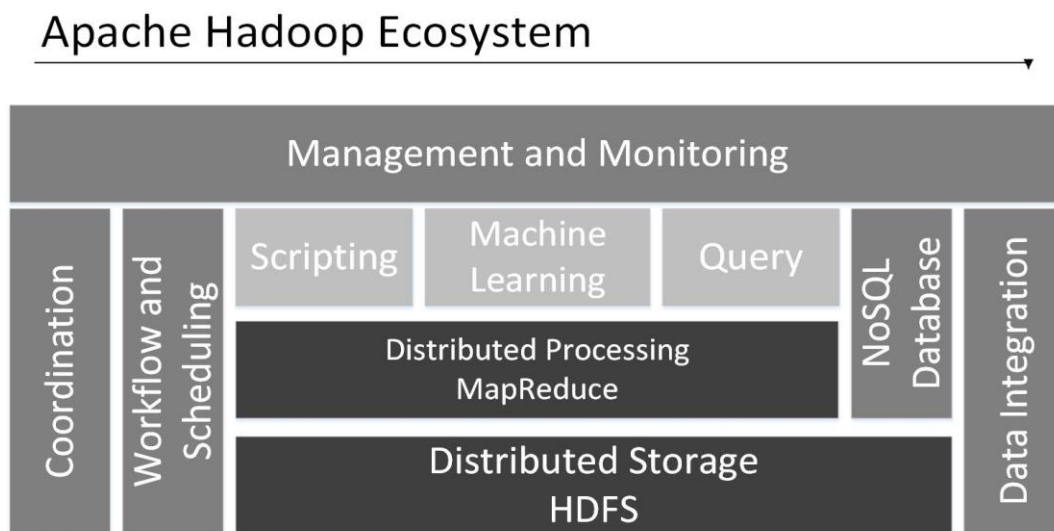


Figure 3.2 Apache Hadoop Ecosystem

Figure 3.2 shows the main components that collectively form a Hadoop ecosystem. It can be seen that Hadoop comprises of Distributed Storage HDFS, Distributed Processing MapReduce, a NoSQL database, an engine for data integration, data coordination, a scheduler for workflow and a unit for data management and monitoring.

A Hadoop job divides the input into data chunks that are independent. Parallel fashion is used to process the data chunks in to set of map and reduce tasks. The framework takes care of the cycle of inputs and outputs. Output of one map task is input of anther reduce task. The framework automatically deals with the output mappers and gives them as input to the reducer functions. The Hadoop framework schedules, monitors and re-executes the failed jobs.

Applications itself implements the appropriate interfaces and the abstract-classes in order to specify the input/output locations and to supply map and reduce functions with these inputs. The Hadoop job client then submits the job (jar/executable etc.) and configuration to the Resource Manager. A Resource Manager distributes the configuration to the workers schedules tasks and

monitor them. The Hadoop framework consists of a Resource Manager, a Node Manager and a Mapper Master per application. It abstracts the challenges of distributed computing hence became a popular tool for big data processing. Hadoop was one of the earliest system to present a parallel processing framework. The two important components of Hadoop are HDFS and YARN, which are explained below. Moreover, the working of the Hadoop framework is also discussed in detail with an example followed by a sub-section explaining its limitation.

3.3.1 HDFS

Hadoop Distributed File System [47], is referred as HDFS, is a single reliable file system. HDFS is reliable file system as it offers the monitoring of failures of data blocks. Each data block has its replica stored on another block and incase of failure data can be retrived from other block. This feature of HDFS makes it easier to use commodity hardware for processing of big data. It does not impose any restriction on schema (structure of data) when it is handling the job of checking for failures and balancing the division of data blocks. Map Reduce further eases the complexities of big data processing by introduction of the concept of parallelization, distribution and fault tolerance in program execution. The MapReduce framework and HDFS runs on the set of nodes which schedules the tasks on the nodes, effectively. On these nodes the data is already available resulting in high aggregate bandwidth across the cluster. HDFS comprises of two major components namely Name Node and Data Node.

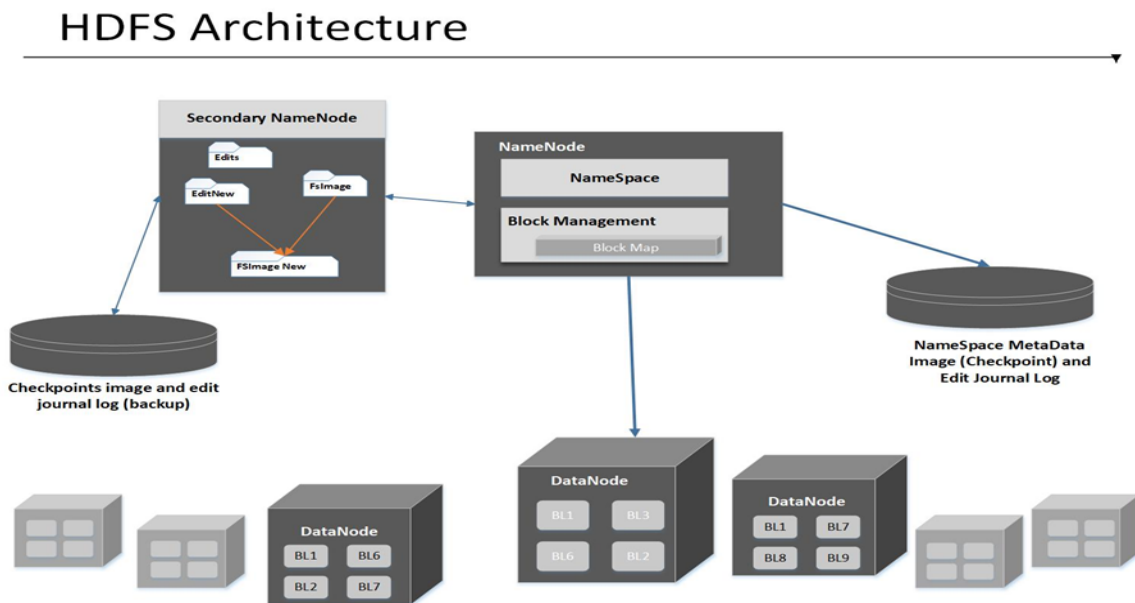


Figure 3.3 HDFS Architecture

As it can be seen in Figure 3.3 that HDFS consists of Name Node and Data Node. The details are given below:

1 Name Node

Name Node is the prime node. It contains metadata (data about data) that requires a few resources than the data nodes. The data nodes basically the commodity hardware that stores the actual data. Undoubtedly, making Hadoop cost effective. The HDFS architecture also contains Secondary Name Node. Secondary Name Node is dedicated node in HDFS cluster. Its main function is to take checkpoints of the file system’s metadata present on Name Node.

2 Data Node

The Data Node is basically a block server that stores the data in the local file. Data Node makes Hadoop cost effective.

3.3.2 Yarn

The first version of Hadoop used to handle all responsibilities such as scheduling, managing the job execution, and interfacing and manage the flow of data. Increasing number of specialized applications require different processing models. The processing models demand attention. Second version of Hadoop was developed on top of existing MapReduce model. MapReduce’s batch processing model is not suitable working model for all applications, including for those that require a lot of iteration during the execution such as machine learning algorithms or graph processing algorithms. In order to deal with this Yarn [47] was developed.

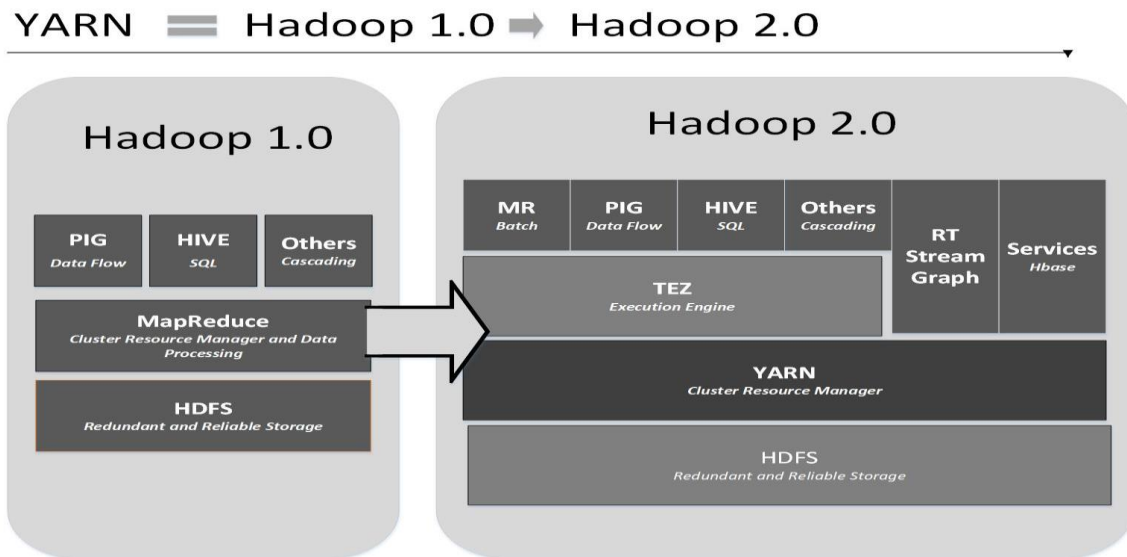


Figure 3.4 Introduction of YARN in Hadoop version 2.0

Yarn stands for Yet Another Resource Negotiator. It separates MapReduce from resource manager, workflow manager and fault-tolerance. It allows other frameworks to be reside on top of it. The original Hadoop Framework was modified to use Yarn. The Figure 3.4 shows the transformation of Hadoop 1.0 into Hadoop 2.0 with introduction of YARN. As it can be seen a few of the other services were also introduced along with YARN.

3.3.3 Example of a Hadoop Job

Figure 3.5 shows an example of a Hadoop Job. It shows working of a program that count occurrences of words in a given file using MapReduce algorithm on Apache Hadoop.

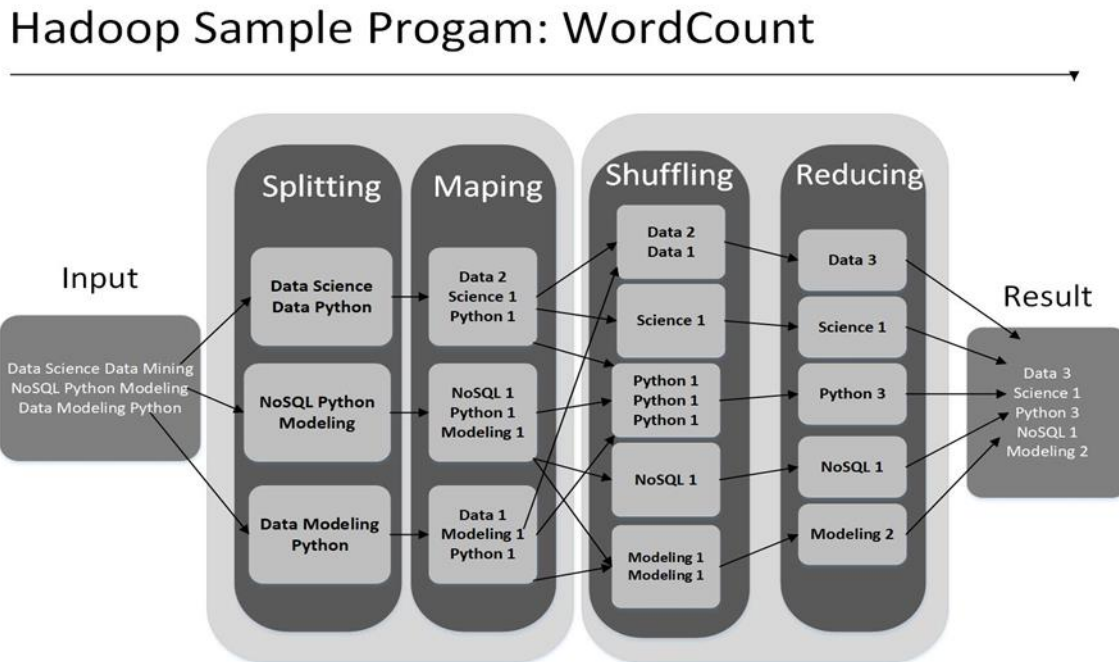


Figure 3.5 Hadoop WordCount Program

It can be seen in Figure 3.5 that the input data is split in to terms by using the split operation. Then the split terms go through the process of mapping in which the occurrence of words common in one given file are counted. Then the mapped data goes into the process of shuffling. Where similar terms are grouped together but their frequency is not added in this step. After the process of Shuffling the data goes into Reduce stage where the frequencies of the common words are added hence giving out word count of all data terms as output.

3.3.4 Limitation of the Hadoop Framework

The initial version of Hadoop had technical limitations [49] that the current system has solved by forcing a linear dataflow framework on distributed programs in the Hadoop cluster. Hadoop reads the data from disk while performing I/O operations, maps the data using a function. After that the results of the map function are reduced and stored on reduction results on disk. Everything was to

be read and written to disk. This process make the implementation of the algorithms difficult that contain a lot of iterations and repetitions [129]. Algorithms working in multiple iterations visit their data repeatedly and performs the data analysis on the side. Most of these are the training algorithms that are used in machine learning systems. The current version of Hadoop doesn't have the capability for processing of iterative machine learning algorithms and if processed it will take a lot of time to finish a job. This provided a need of a technology that would solve these issues. This lead to evolution of Apache Spark [50].

3.4 Apache Spark

Apache Spark was developed to facilitate the iterative and machine learning algorithms. Spark was born along with its important component the Resilient Distributed Datasets (RDDs) [50]. The RDDs performs in-memory computations on Big Data. It runs on large clusters and nodes. It runs in a way to provide fault-tolerance. Apache Spark also has many Discretized Streams. The discretized streams were developed in order to provide high-level programming API. The high level programming API provides flexibility and efficiency in fault recovery to distributed data processing. Spark is a high-level system that supports the batch processing, stream processing and also gives a support for running of machine learning algorithms that work in iterations. Spark's high-level application programming has become very popular as compared to its counterpart Hadoop. Apache Spark can be programmed using Scala, python, java and R.

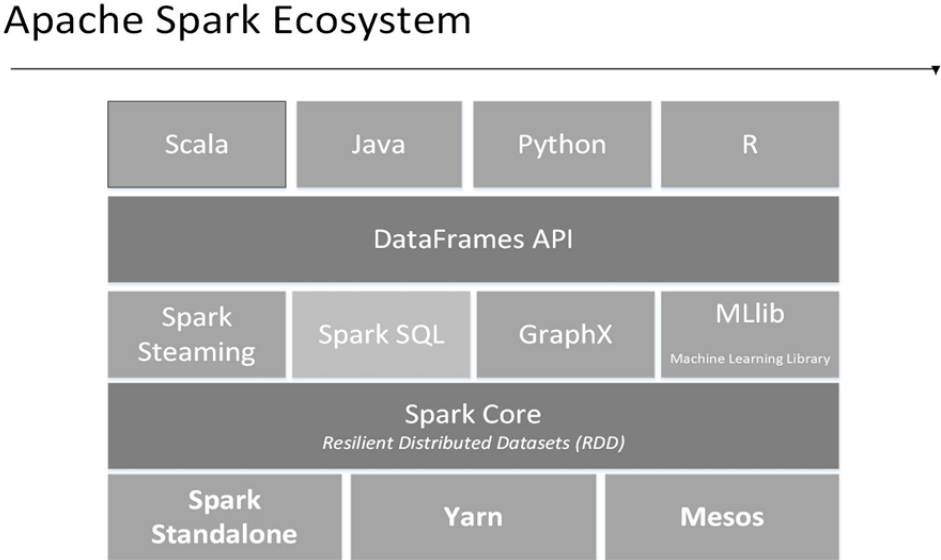


Figure 3.6 Apache Spark Ecosystem

The Figure 3.6 shows the Apache Spark Ecosystem. The spark architecture is layered, Resilient Distributed Datasets sits on top of Yarn and Spark Standalone. Then comes the Spark Streaming, Spark SQL and Machine Learning component MLib. On top of that Data Frames API sits and

then at the top application layer sits that includes all the languages such as R, Java, Scala and Python. Below, we discuss main components of Apache Spark, followed by a working example.

3.4.1 Resilient Distributed Datasets (RDDs)

RDD is the key feature of spark [50]. RDD is basically a collection of immutable objects. These objects are basically stored into different partitions. Whenever an RDD was modified, a new RDD is generated. The generation of new RDD leaves the previous RDD unconverted. It provides fault tolerance due to the intelligence that decides when to regenerate and when to re-compute the dataset. Spark Core provides the distributed task dispatching, task scheduling, and basic I/O processing. The application programming interface is the center of RDD and provides the abstraction. RDDs support two types of operations: Transformations and action transformations. Transformation return pointers to new RDDs. Actions transformation return values or results to the driver program. Multiple transformations and actions can work together in a Spark job.

Resilient Distributed Datasets

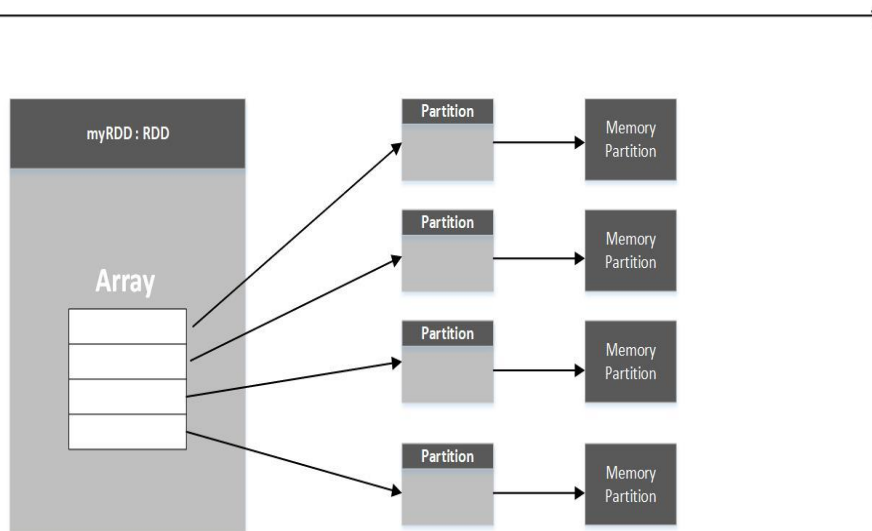


Figure 3.7 Resilient Distributed Datasets

The Figure 3.7 shows the abstraction of RDDs how they map the array of data into partitions in the memory. This gives programmer the ability to partition RDDs across machines. The portioning occurs on the basis of a key in each record. RDDs are computed lazily from the lineage graph. Programmers write a driver program, which connects to a cluster of workers. One or more RDDs are in the driver program. The actions or transformation actions are invoked on them. Spark code on the driver tracks the RDD lineage. Workers are the long-lived processes that can store RDD partitions in memory across operations.

3.4.2 Spark Streaming

It uses spark's capability to perform streaming analytics. In Spark Streaming⁶ the data is absorbed in mini-batches. The transformations are performed on those batches. This design basically helps in facilitating the code that is written for batch analytics so that it can also be used in streaming analytics.

Spark Streaming

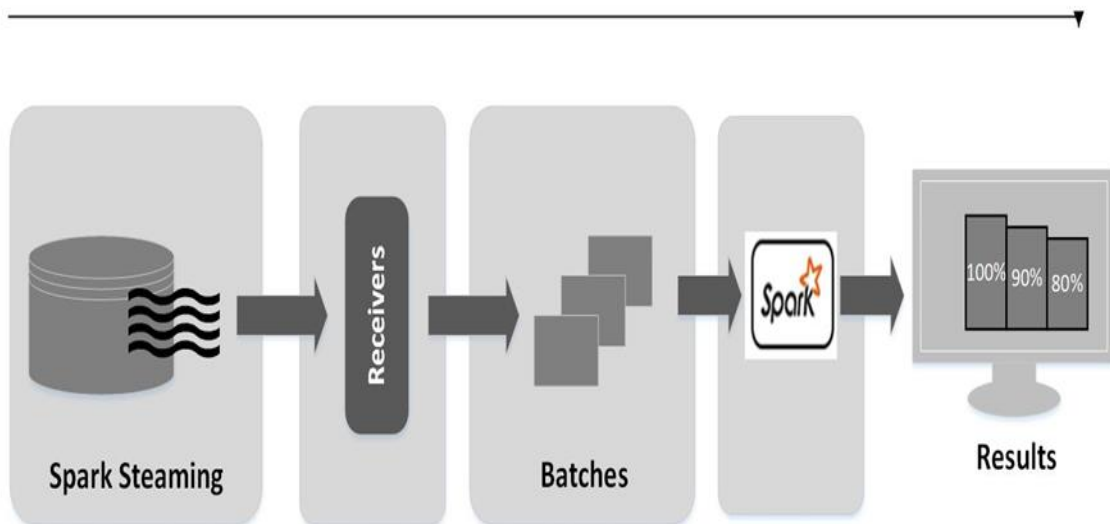


Figure 3.8 Spark Streaming

The Figure 3.8 shows the work flow of Spark Streaming. Spark Streaming basically receives input data streams. This data stream is live and is constantly updated. After receiving the input stream it is divided into batches. After that Spark engine processes it and generates the final stream of results. The result is also generated in stream of batches. High-level abstraction called discretized stream or DStream is achieved by Spark Streaming.

3.4.3 MLlib

MLlib⁷ is a distributed machine learning framework that runs on top of spark core. The distributed memory-based spark architecture causes the Spark jobs to runs nine times faster than the disk-based implementation. Many machine learning algorithms have been implemented and transported to MLlib, which helps to process large scale machine learning.

⁶ <https://spark.apache.org/docs/latest/streaming-programming-guide.html>

⁷ <https://spark.apache.org/docs/latest/ml-guide.html>

Mllib → Machine Learning Library

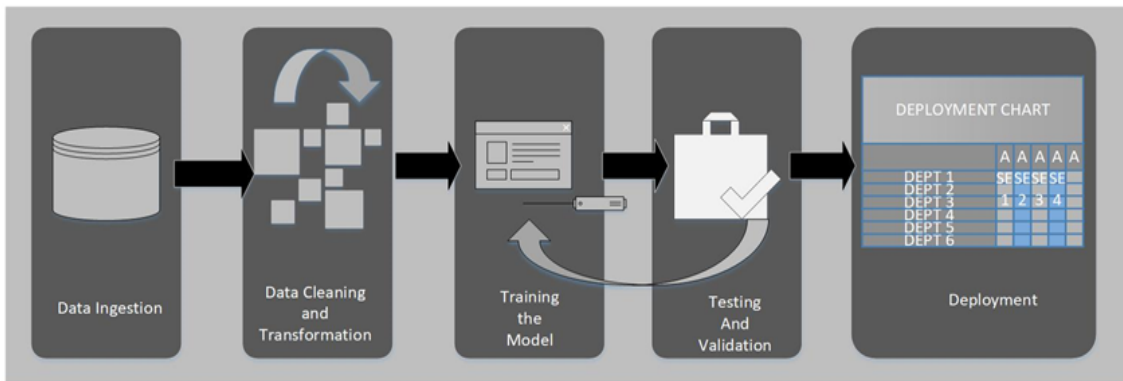


Figure 3.9 MLib – Machine learning Library for Apache Spark

The Figure 3.9 shows the working of Spark MLib. It shows the basic steps like any other machine learning algorithm. It starts with data input then data goes through the process of preprocessing and then the model is trained and validated. The underlying process of this machine learning job is MapReduce. MLib successfully performs supervised and unsupervised learning for machine learning tasks and trains the model to give output.

3.4.4 Example of Spark

Figure 3.10 Spark Word Count Program shows an example of a Spark Job. It shows working of a program that counts occurrences of words in a given file using Apache Spark.

Spark Sample Program : Word Count

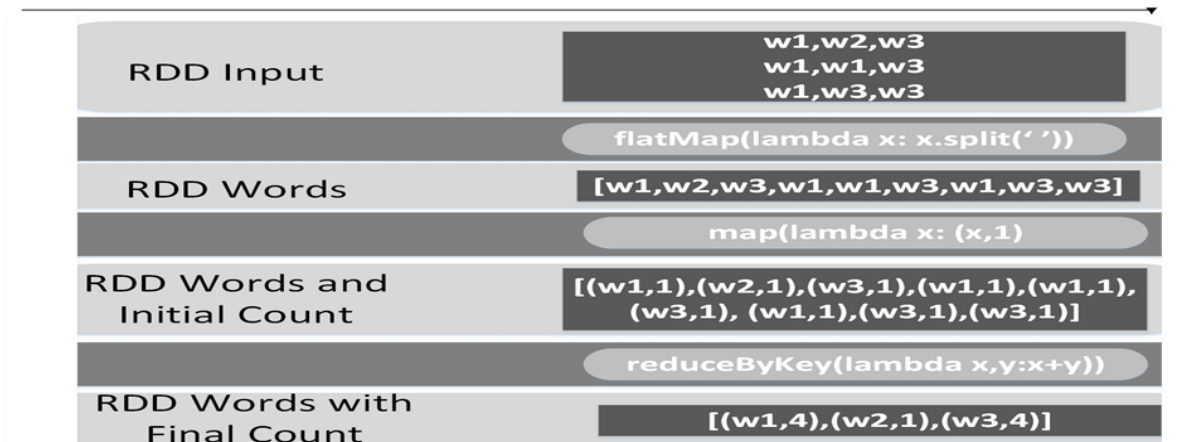


Figure 3.10 Spark Word Count Program

Execution of word count program is shown in the figure 3.10. First of all the data is loaded in to RDD. After that RDD splits the words in the input using Map task. RDD then counts their occurrences again using a Map task. After that by using Reduce task it counts the occurrences of all the words in the input and gives it as the final result. Apache Spark performs word count in much less time and less steps as compared with Apache Hadoop.

3.5 Choice of Big Data Environment For the Implementation

Apache spark has many benefits over Apache Hadoop. In Apache Spark it is easier to develop applications because they are being constructed in a unified API. Apache Spark makes it more productive to combine processing tasks. Moreover Apache Hadoop does processing by performing I/O operations from disk. This makes the processing and analyzing of the data slower. Also reading, writing and storing the data is done in one engine then it is passed it to another engine for processing this again puts an impact on processing speed and overall running time of the job. On the other hand, Apache Spark achieves diverse functions over the same data by performing in memory computations using RDD. This makes Apache Spark a technology that enabled the applications to perform tasks that were not possible with previous systems. Initially the research work was implemented on Apache Hadoop but it lacked the support of machine learning libraries that our algorithm required for the implementation of incremental taxonomy generation. Apache Hadoop would be a good tool if non-incremental taxonomy generation was the research objective. The algorithm presented in this research work is for incremental taxonomy generation and requires a support of machine learning algorithms that are already available in MLlib of Apache Spark hence Apache Spark was chosen as the environment for the complete implementation of this research work. In order to support our choice, we have also demonstrated the suitability of the Apache Spark for machine learning application by comparing the performance of these two parallelization frameworks, the details of which are given in Chapter 6. The next chapter discusses the methodology for this research work.

Chapter 4. Proposed Algorithm

This chapter proposes a novel technique which has adopted the concept of parallelization for making an efficient and scalable algorithm for taxonomy generation and evolution. The technique performs incremental taxonomy generation. The technique first generates the taxonomy and after that updates the taxonomy upon introduction of new documents in to the system. This section is grouped into two further sections: first subsection discusses the Taxonomy generation process based on parallelization framework i-e Apache Spark. The second subsection will discuss how the taxonomy is updated on introduction of new documents in to the system. We are using the terminology Taxonomy Evolution in order to explain the taxonomy update process. The algorithm is based on Apache Spark.

4.1 Taxonomy Generation

This work proposes a novel technique that generates a taxonomy from the corpora of textual documents. The technique uses the hierarchical agglomerative clustering technique as an underlying technique to generate the taxonomy. The proposed technique performs taxonomy generation on Apache Spark framework and is divided in to six main steps loading the data, Data preprocessing, Data modeling, Formation of hierarchy, Cluster labeling and Conversion in to tree graph.

4.1.1 Loading Data

Input text documents are loaded in to resilient distributed dataset (RDD). RDD speeds up the process of loading and processing the data using the parallelization. As machine learning algorithms tend to re-use or share the data among multiple jobs this gives birth to the problem that this data needs to be stored in some intermediate stable distributed store. This makes over all computations slower as it will involve multiple I/O operations. RDD solves this problem by providing in memory computations. These in memory computations are fault tolerant as well. RDD has the ability to load the multiple text files at the same time hence speeding up the process of loading the data. A driver program in RDD divides the data in to many slices that are sent to multiple nodes. Instead of saving the actual data RDD saves the transformations on the data. If an RDD partition is lost, the transformation can be replayed.

4.1.2 Data preprocessing

The quantity of the data grows exponentially with the dimension in the input data. Data generated from real time processing is generally incomplete, inconsistent and lacks certain behavior that may lead to errors in computation. In short we can say that the data is noisy and contain outliers. Hence there is a need to process that data to resolve above mentioned issues [51]. It has been proved that time taken to preprocess can take 50% to 80% of the entire running of an algorithm .In machine learning data preprocessing is the step, in which we transform the data and encode it so our algorithm can easily process it. It can be safely said that preprocessing will significantly decrease

the running time of a clustering algorithm for textual data. The data preprocessing technique is further divided in to two parts i-e Stop Word Removal, and Stemming of words.

Data Preprocessing

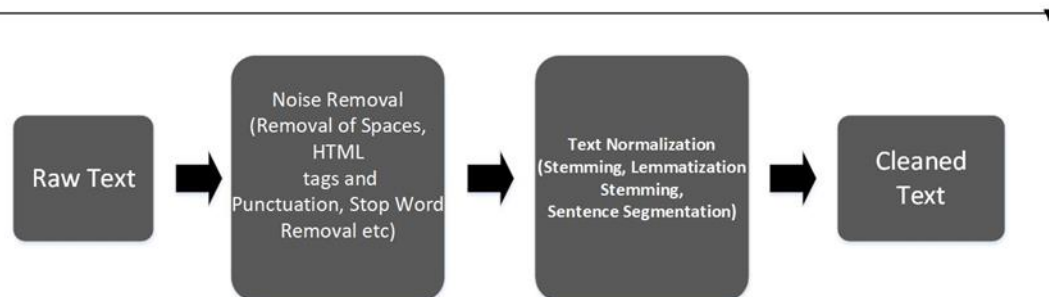


Figure 4.1 Data Preprocessing Pipeline

Figure 4.1 Data Preprocessing Pipeline shows the Data Preprocessing Step. First noise is removed then stemming is applied and that process gives cleaned text as output.

1 Removal of stop words

Frequently used words in English language are useless in the field of information retrieval. They provide little to no information hence referred as stop words. Removal of such words is not a hard or fast rule. For the tasks of text clustering and classification we want to focus more on the words which provide meaning and are helpful in text clustering. It has been observed that the 10 most common words in the English language that are mostly used in the documents, are about 20-30% of tokens in a text [120]. Stop word removal has many benefits. On removing stop words, size of data decreases and also the time to train the model also decreases. By performing stop words removal there are very fewer meaningful tokens left which ultimately increases the classification accuracy and potentially improves the performance. In this algorithm we have used NLTK⁸ framework for stop words removals. NLTK is a popular framework for python programs that works with the human language data.

2 Stemming

In information retrieval the process of stemming means the word to its base word or root. The process of stemming reduces the words to their stems, and affixes to their suffixes and prefixes to their roots that are known as lemma. This process is an important part of Natural language understanding and natural language processing. Stemming is basically

⁸ <https://www.nltk.org/>

a part of linguistic studies and artificial intelligence. Stemming and AI knowledge extract meaningful knowledge from vast sources of big data. Additional forms of the word may need to be searched to get better understanding of the word. R. Snowball Stemmer [121] has been used for this purpose.

4.1.3 Data modeling

In this step, the pre-processed data is modeled using Vector Space Modeling (VSM) technique. VSM technique is an algebraic model through which the text documents are represented [122], where documents are represented as vectors. Vector space modeling determines the model based on the properties of data, it builds the model based on the term and document frequencies rather than relying on the external knowledge sources.

Vector Space modeling basically finds out the important and relevant terms in the data set. This stage uses the words obtained from stages data preprocessing. Every document is represented using a vector. The vector identifies the document terms relationship by recording the frequency of occurrence of terms in the document and uses them as the values of vector.

1 Term frequency-inverse document frequency

Term frequency-inverse document frequency is used as a feature vectorization method that is an indicator of how important a term is in the corpus. In Apache Spark TF-IDF is performed using MLlib. TF-IDF is mathematically defined as⁹:

Given a term t , a corpora D and a document d ; Term frequency $tf(t,d)$ is a number of times that a term t appears in a document d . Figure 4.2 shows that the number of time the six different terms term1, term2, term3, term4, term5 and term6 appear in in three different documents; doc1, doc2, doc3.

Term Frequency

	term1	term2	term3	term4	term5	term6
doc1	0	0	1	0	1	1
doc2	0	0	1	1	0	1
doc3	1	1	0	0	1	0

Figure 4.2 Term Frequency

⁹ <https://spark.apache.org/docs/latest/mllib-feature-extraction.html>

The document frequency $df(t, D)$ is basically that how many documents contain that term t . The terms which are recurring in most documents do not usually contain a lot of useful information about that particular document e.g., “a”, “the”, and “of”. Inverse document frequency $idf(t, D)$ is a numerical measure of the weight of information provided by a term provides given as:

$$idf(t, D) = \log\left(\frac{|D| + 1}{df(t, d) + 1}\right) \quad (6)$$

Here $|D|$ represents the total number of documents in the corpus. As we are using logarithm IDF becomes 0 if a term appears across all the documents. For such cases we use Laplace smoothing in order to avoid dividing by zero. The Figure 4.3 shows the calculation of Inverse Document Frequency for the term1, term2, term3, term4, term5 and term6. In the calculation of Inverse Document Frequency, Laplace smoothing is also applied.

Inverse Document Frequency

Formula $idf(t, D) = \log\left(\frac{|D| + 1}{df(t, d) + 1}\right)$

$$\begin{aligned} \text{term1} &= \log\left(\frac{3+1}{1+1}\right) = 1 \\ \text{term2} &= \log\left(\frac{3+1}{1+1}\right) = 1 \\ \text{term3} &= \log\left(\frac{3+1}{2+1}\right) = 0.4150 \\ \text{term4} &= \log\left(\frac{3+1}{1+1}\right) = 1 \\ \text{term5} &= \log\left(\frac{3+1}{2+1}\right) = 0.4150 \\ \text{term6} &= \log\left(\frac{3+1}{2+1}\right) = 0.4150 \end{aligned}$$

Figure 4.3 Inverse Document Frequency

Once the Inverse Document Frequency $idf(t, D)$ has been calculated and also Term Frequency $tf(t, d)$ has been calculated; based on this, Term Frequency-Inverse Document Frequency $tfidf(t, d, D)$ is calculated. This measure is basically the product of $tf(t, d)$ and $idf(t, D)$, mathematically denoted as:

$$tfidf(t, d, D) = tf(t, d) * idf(t, D) \quad (7)$$

The Figure 4.4 shows the calculation of Term Frequency-Inverse Document Frequency $tfidf(t, d, D)$.

Term Frequency-Inverse Document Frequency

$$\text{Formula } tfidf(t, d, D) = tf(t, d) * idf(t, D)$$

	term1	term2	term3	term4	term5	term6
doc1	0.000	0.000	0.4150	0.000	0.4150	0.415
doc2	0.000	0.000	0.4150	1.000	0.000	0.415
doc3	1.000	1.000	0.000	0.000	0.415	0.000

Figure 4.4 Term Frequency – Inverse Document Frequency

In MapReduce Paradigm i-e on Apache Spark, Term Frequency-Inverse Document Frequency $tfidf(t, d, D)$ is not calculated in simple fashion, rather a number of Map and Reduce tasks are carried out for the implementation of $tfidf(t, d, D)$. Apache Spark implements $tfidf(t, d, D)$ using hashing trick or kernel trick.

2 Hashing Trick/ Kernel Trick

Hashing trick¹⁰ is an efficient way of vectorising features by converting the arbitrary features into indices using the form of a vector or a matrix. It convert documents into a numerical representation so that they can be used as input to a similarity function. It applies the hash function to these features and uses their hash values as indices directly.

3 Feature vectorization using hashing trick (HashingTF)

Hashing trick is used by a feature vectorizer in order to build a vector of a pre-defined length. It works by applying a hash function h to the features (e.g., terms). The default feature dimension is 262,144. It then uses the hash values directly as feature indices and updates the resulting vector at those indices. Here, we assume that features actually means feature vector. A hash function is used to map a raw feature. Mapped indices are used for calculations of term frequencies. The computation of global term for index mapping can be avoided by using mapped indices. Otherwise can prove time expensive for a large corpus. The hash function used is MurmurHash3¹¹.

¹⁰ https://en.wikipedia.org/wiki/Feature_hashing

¹¹ https://en.wikipedia.org/wiki/MurmurHash#cite_note-2

HashingTF

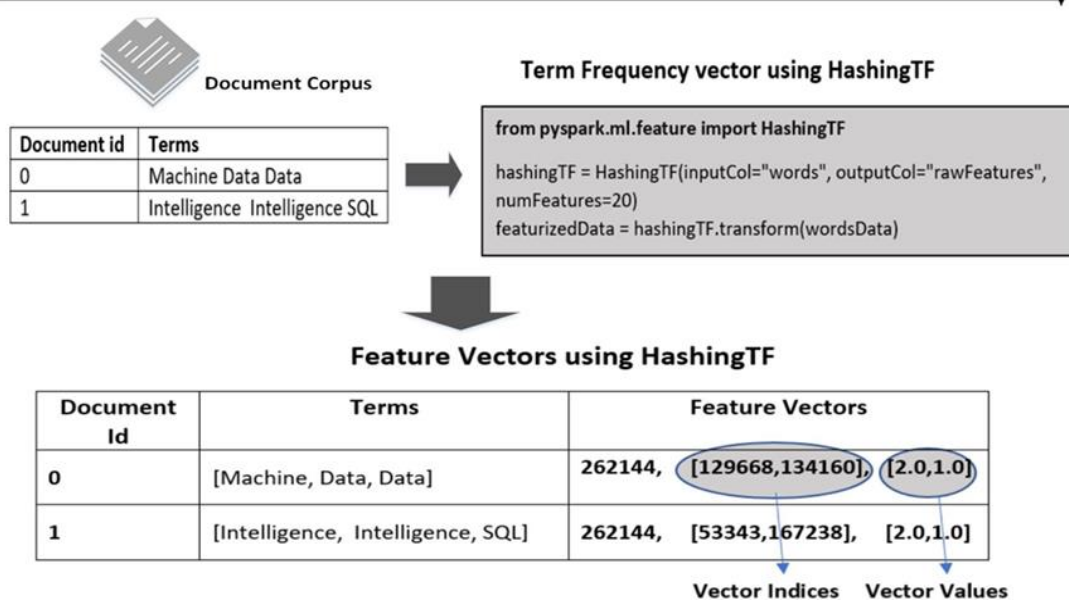


Figure 4.5 Implementation of TF-IDF using HashingTF on Apache Spark

It can be seen in the Figure 4.5 the vector's dimension is set to default. The default value is 262,144. Also, term 'Machine' is mapped to index 134160 by the hashing function and has frequency equal to 1. Similarly in the same way insights can be gained about other terms.

4 MurmurHash3

Murmurhash3 is a modern non-cryptographic hash function. It has a low collision rate and high performance. Low collision rate makes it suitable for general hash-based lookups. The name comes from two basic operations, multiply (MU) and rotate (R), used in its inner loop. Murmurhash comes in different versions. The current version is MurmurHash3 which yields a 32-bit or 128-bit hash value.

4.1.4 Formation of hierarchy

In this step, relationships between documents are identified based on the modeled data obtained from the previous steps. In order to make hierarchical relationships, Hierarchical Agglomerative Clustering (HAC) is used.

1 HAC

Hierarchical clustering algorithms considers each document as a single cluster at the initial stage. As the algorithm progresses the then cluster pairs began to merge. The process continues until all clusters have been merged into a single cluster. The single cluster then contains all the documents. Bottom-up hierarchical clustering is therefore called hierarchical agglomerative clustering or HAC. Hierarchical clustering does not require a

pre-specified number of clusters. Hierarchical clustering algorithm merges or separates a cluster based on similarity measure. The similarity measure used here is called cosine similarity.

2 Cosine Similarity

Cosine Similarity is used to compare two documents and find out about the similarity. Cosine similarity calculates the cosine of angle between two vectors. The vectors used here are the document vectors generated from the previous steps. This metric is measurement of orientation that tells how close two vectors are in the vector space. Here emphasis is laid on the angle between the word count but not its magnitude. Cosine Similarity can be mathematically defined as:

Given the document vectors \vec{a} and \vec{b} for documents A and B respectively having feature size n , the $similarity(A, B)$ will be calculated as follows:

$$similarity(A, B) = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \|\vec{b}\|} = \frac{\sum_{i=1}^n a_i b_i}{\sqrt{\sum_{i=1}^n a_i^2} \sqrt{\sum_{i=1}^n b_i^2}} \quad (8)$$

Cosine similarity is measured against the tf-idf matrix and generates the similarity between each document and the other documents in the corpus. Figure 4.6 shows the calculation of similarity among the documents in the corpus using cosine similarity

Cosine Similarity Calculation

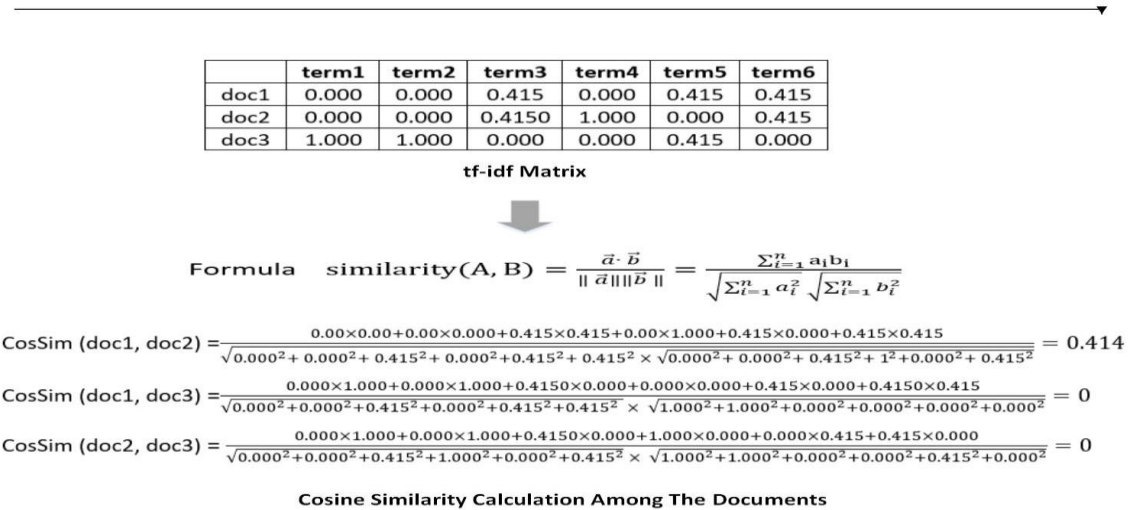


Figure 4.6 Cosine Similarity Calculation

Using the above Cosine Similarity among the documents in the corpus the similarity matrix S_{gen} is created. This Research work for the generation of a hierarchical structure uses the

Ward's method [123]. The similarity matrix S_{gen} is given as input to Ward's method in order to perform hierarchical clustering.

3 Ward Method

Ward's method [123], merges two clusters based on the optimal value of the objective function. In this technique the distance between two clusters should be within the optimal value of the objective function. Objective function used here is error sum of squares.

$$\Delta(A, B) = \sum_{i \in A \cup B} \|\vec{x}_i - \vec{m}_{A \cup B}\|^2 - \sum_{i \in A} \|\vec{x}_i - \vec{m}_A\|^2 - \sum_{i \in B} \|\vec{x}_i - \vec{m}_B\|^2 \quad (9)$$

$$= \frac{n_A n_B}{n_A + n_B} \|\vec{m}_A - \vec{m}_B\|^2 \quad (10)$$

When \vec{m}_j is the center of cluster j and where \vec{m}_j is the center of cluster j , and n_j is the number of points in it. Merging cost is defined by Δ . The sum of squares starts out at zero. In the beginning is every point has its own cluster and then the clusters grow by performing merge operation. That is why sum of square is zero in the beginning. By using the Ward's method, the growth is kept small. For two pairs of clusters that are far apart at an equal distance, the Ward Method [124] would merge the smaller clusters first.

Hierarchical Clustering Using Ward's Method

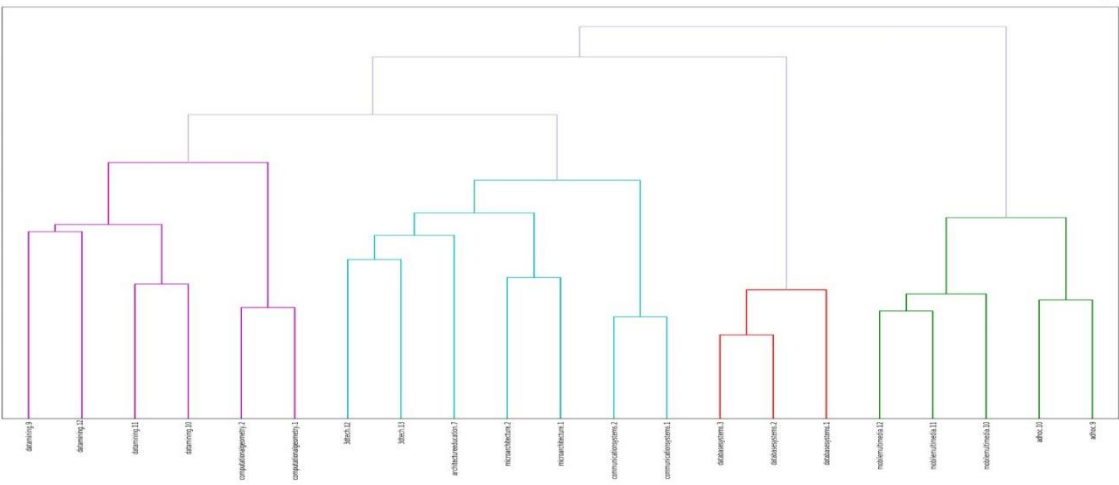


Figure 4.7 Hierarchical Clustering Dendrogram Produced from Ward's Clustering Method

Figure 4.7 shows the hierarchical clustering dendrogram created using Ward's Method. In this figure a dendrogram has been used in order to show the hierarchical clustering. A dendrogram is a diagram that shows the hierarchical relationship between objects.

4.1.5 Cluster labeling/Node labeling

Cluster labeling is an important step for any kind of clustering. As in particular tasks of textual data arrangement and analysis, humans interact with cluster. Therefore, labeling is important. Cluster-internal labeling computes a label that completely depends on the cluster itself, and doesn't use the information from other clusters or uses external knowledge. The hierarchical structure created in the previous step is unlabeled. At this step labels are identified for the unlabeled clusters. The technique¹² of cluster internal label is used. In this technique labeling a cluster is being done using the title of the document closest to the centroid \vec{c}_{cen} . \vec{c}_{cen} is the middle point of a cluster. It is basically the average representation of all documents in a cluster [130]. The formula for calculation of Centroid \vec{c}_{cen} is given by:

$$\vec{c}_{cen} = \frac{\sum_{i=1}^m \vec{d}_i}{m} \quad (11)$$

Where c is a cluster having m documents mapped in t dimensional vector space. Let \vec{d}_i be the vector of an i^{th} document in the cluster c . Figure 4.8 shows the calculation of centroid for a given cluster.

Centroid Calculation For A Cluster

Formula	$\vec{c}_{cen} = \frac{\sum_{i=1}^m \vec{d}_i}{m}$					
	angles	los	new	post	times	york
D1	0.000	0.000	0.415	0.000	0.415	0.415
D2	0.000	0.000	0.415	1.000	0.000	0.415
D3	1.000	1.000	0.415	0.000	0.000	0.000
Σ	1.000	1.000	1.245	1.000	0.415	0.830
$\frac{\Sigma}{m=3}$	0.333	0.333	0.415	0.333	0.138	0.276
$\vec{c}_{cen} = (0.333, 0.333, 0.415, 0.333, 0.138, 0.276)$						

Figure 4.8 Centroid Calculation

For labeling technique the titles of the documents in a cluster were chosen as labels. The titles were selected as labels because titles are easier to read as compared with the list of top terms in a cluster. Once the all the clusters are computed after that their centroids are calculated. The

¹² <https://nlp.stanford.edu/IR-book/html/htmledition/cluster-labeling-1.html>

technique then labels the cluster using the name of the document that is nearest to the centroid of that particular cluster.

4.1.6 Conversion into a tree graph

In this step, a taxonomy T_{gen} has been created. The process of generation of initial taxonomy has been completed here. The Taxonomy T_{gen} created here in this step will be further used for the process of taxonomy evolution. In order to use this taxonomy for evolution afterwards, T_{gen} is further converted into a Newick Tree Graph¹³.

1 Newick Trees

The Newick¹⁵ is a standard for representing the trees in computer readable form making use of nested parentheses. The tree basically ends with a semicolon. The bottom-most node in this tree is an interior node not a tip. Matched parentheses represent interior nodes. Between them are the representations of the nodes that are immediately descended from that node that is separated by commas. The tree file can be shown by the following sequence:

$$(B, (A,C,E) , D); \quad (12)$$

Real numbers are used in order to incorporate the branch lengths. It is done by using a real number with or without a decimal point after a node proceeded by a colon. This represents the length of the branch immediately below that node. Thus the above tree might have lengths represented as:

$$(B:2.0,(A:6.0,C:7.0,E:9.0):5.0,D:12.0); \quad (13)$$

The tree starts on the first line of the file, and can continue to subsequent lines. The Figure 4.9 shows the Newick Tree generated from the Taxonomy T_{gen} . This Newick Tree will be used in the process of Taxonomy Evolution that is explained in next section

¹³ <https://pypi.org/project/newick/>

Newick Trees

```
(((((datamining.9:1.28,3dtech.11:1.28):0.05,(datamining.11:0.92,datamining.10:0.92):0.41):0.42,(computational
geometry.2:0.76,computationalgeometry.1:0.76):0.99):0.33,(((1.09,3dtech.13:1.09):0.17,architectureeducation.
7:1.25):0.15,(microarchitecture.2:0.97,microarchitecture.1:0.97):0.44):0.22,(communicationsystems.2:0.70,commu
nicationsystems.1:0.70):0.93):0.45):0.40,((databasesystems.3:0.57,databasesystems.2:0.57):0.31,databasesystem
s.1:0.88):1.59):0.21,((mobilemultimedia.12:0.74,mobilemultimedia.11:0.74):0.12,mobilemultimedia.10:0.85):0.5
2,(adhoc.10:0.81,adhoc.9:0.81):0.56):1.31);
```

Figure 4.9 Newick Tree for the Taxonomy T_{gen}

4.2 Taxonomy Evolution

When new documents are introduced into the system, the new documents go through the same process of Taxonomy Generation (as discussed in the previous subsection 4.1). Once the process of taxonomy generation is finished a new tree structure T_{evo} is constructed that represents the newly introduced textual document. A similarity matrix S_{evo} is also produced along with T_{evo} , is produced. Now in order to evolve the taxonomy, TreeMerge [125] technique is used.

4.2.1 TreeMerge

The Tree Merge technique takes taxonomies $\{T_{gen}, T_{evo}\}$ and similarity matrices $\{S_{gen}, S_{evo}\}$ as input. After the input has been taken the next step is building of a compatibility super tree T_s . The compatibility super tree is built by running NJMerge [126] algorithm on it. The Super tree methods construct trees from smaller trees for overlapping subsets of the taxonomy.

1 NJMerge:

NJ Merge is ran on the input pair $\{T_{gen}, T_{evo}\}$ and similarity matrix $\{S_{gen}, S_{evo}\}$ as auxiliary information. In Newick trees numbers are used to represent branch length. These numbers are incorporated in the structure as shown in Figure 4.9. NJMerge calculates the true neighbors of Tree T_{evo} comparing the branch length of the tree structure T_{evo} with the T_{gen} . Sum of branch lengths is calculated for all the branches of both the tree structure. The pair that shows the smallest length is considered as a true neighbor. Once the true neighbours have been identified the next step is merging of the two trees. The pairs of trees are then merged using Strict Consensus Merger. A merged tree T_{merged} is built simply by inserting the leaf nodes based on the branch distances mathematically denoted as:

$$T_{merged} = (T_{gen}) U (T_{evo}) \quad (14)$$

$$\mathcal{L}(T_{merged}) = \mathcal{L}(T_{gen}) U \mathcal{L}(T_{evo}) \quad (15)$$

NJMerge

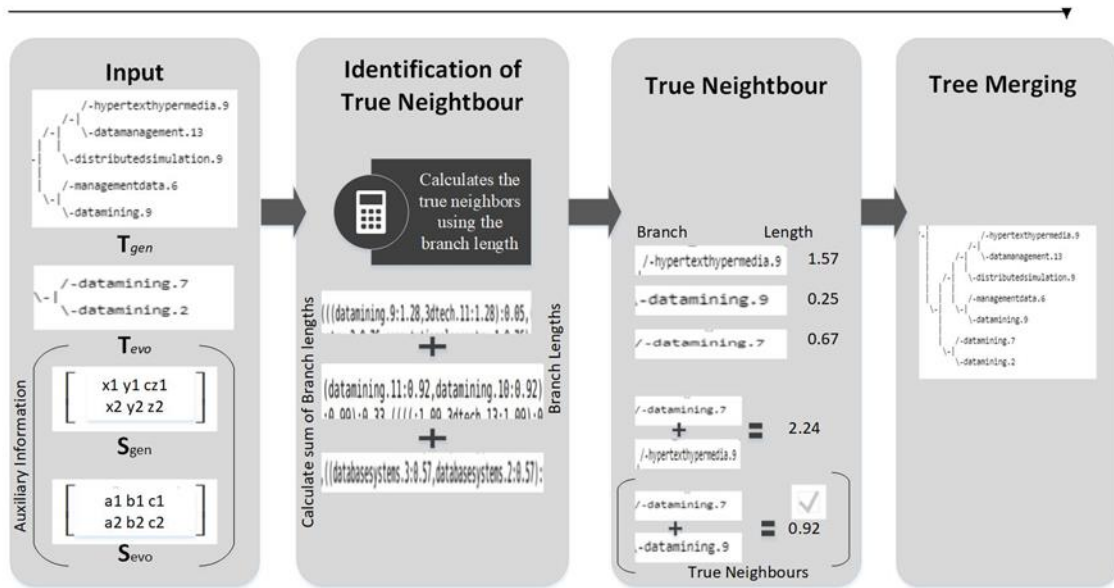


Figure 4.10 Evolution of Trees using NJMerge

Where, $\mathcal{L}(T_{gen})$ and $\mathcal{L}(T_{evo})$ are the leaf nodes set of T_{gen} and T_{evo} respectively and $\mathcal{L}(T_{merged})$ is the leaf node set obtained from the merger of the two. The merged tree T_{merged} represents the final evolved taxonomy. The figure 4.10 shows the process of evolution of input pair of trees $\{T_{gen}, T_{evo}\}$ using the NJMerge algorithm. The algorithm takes the similarity matrices and the taxonomies as the input. After that the branch lengths for both the taxonomies are calculated. Then the branches of the T_{evo} are merged with the branches of T_{gen} which show the minimum sum of branch length. Hence given a final evolved tree T_{merged} as output.

4.3 Summary

The proposed technique successfully generates a taxonomy for text documents from a given corpora. The technique also successfully evolves the previously created taxonomy in a very limited time frame. The base of the technique is MapReduce that has the capability to reduce the time by processing the data in parallel and provides fault tolerance by using distributed file system. Map reduce environment helps in improvement in the scalability challenges of current taxonomy generation and evolution techniques. The algorithm runs on Apache Spark environment. The running time and clustering quality of the presented technique has been compared with an already existing technique of taxonomy generation and also a technique of taxonomy evolution. The next section discusses the evaluation of the proposed technique.

Chapter 5. Evaluation

This technique presented in this research work was evaluated on a dataset based on time and quality parameters. Four different experiments were performed. In the first experiment (discussed in section 5.2.1), generation part of the proposed algorithm was evaluated by comparison with an existing non-incremental taxonomy generation technique. In the second experiment (discussed in section 5.2.2) the entire algorithm (generation as well as the update taxonomy process) was evaluated by comparison with an existing incremental taxonomy generation technique. In the third experiment (discussed in section 5.2.3), time for evolution was measured by running the algorithm on different number of Apache Spark cores. In the fourth experiment (discussed in section 5.2.4), taxonomy generation part of the algorithm was evaluated by running the algorithm on both Apache Hadoop environment as well as on Apache Spark and their running times were compared. For the dataset, 2000 scientific documents were downloaded from ACM Digital Library¹⁴. These documents were then converted into text files using Apache Tika¹⁵. Rest of this section will discuss evaluation metrics, experiments and test results.

5.1 Evaluation metrics for clustering quality

As mentioned earlier that the taxonomy obtained went through two different types of evaluation: Time-Based and Quality-Based. Time efficiency was computed using the running time of the algorithm for generation and evolution of taxonomy. The quality metrics used to evaluate the hierarchical clustering quality are Silhouette's score [127] and Davies-Bouldin's score [128].

1 Silhouette Score

Silhouette's score [127] is calculated using intra-cluster distance a and the mean nearest cluster distance b for each sample, mathematically represented as:

$$\text{Silhouette's Score} = \frac{(b - a)}{\max(a, b)} \quad (16)$$

b is the distance measured between a sample and a cluster that is nearest to that but not the part of it. The range of Silhouette's score is between $[-1, +1]$. Zero values show that the clusters are overlapping. Negative values indicate that a document has been assigned to a wrong cluster and a different cluster is more similar. A higher score indicates that

¹⁴ <https://dl.acm.org/>

¹⁵ <https://tika.apache.org/>

the document has been matched to its own cluster and is poorly matched to neighboring clusters.

2 Davies Bouldin Score

Davies-Bouldin's [128] score is basically the ratio of sum of within-cluster scatter to between-cluster separation. To define Davies-Bouldin's score we need to define dispersion measure S_i , the cluster similarity measure C_i and the separation D_{ij} between i^{th} and j^{th} clusters.

$$S_i = \left(\frac{1}{|C_i|} \sum_{x \in C_i} D^p(x, c_i) \right)^{\frac{1}{p}}, p > 0 \quad (17)$$

Where $|C_i|$ the number of data is points in cluster C_i and c_i is the center of the cluster C_i .

$$D_{ij} = \left(\sum_{l=1}^d |v_{il} - v_{jl}|^t \right)^{\frac{1}{t}}, t > 1 \quad (18)$$

Where v_{il} and v_j are the centers of cluster C_i and C_j , respectively. Then the Davies-Bouldin's score would be defined as:

$$V_{DB} = \frac{1}{k} \sum_{i=1}^k R_i \quad (19)$$

Where k is the number of clusters and R_i is defined as:

$$R_i = \max R_{ij} \quad (20)$$

Where R_{ij} is the similarity measure between clusters C_i and C_j and is defined as:

$$R_{ij} = \frac{S_i + S_j}{D_{ij}} \quad (21)$$

Davies-Bouldin's score suggests, lower the values of the score, and better the clustering quality. Minimum score is zero. If two algorithms are being compared the algorithm with lower score will have well-defined and well-separated clusters.

5.2 Experiments And Results

Based on time and quality metrics discussed above, experiments were performed to check the running time capacity and quality of the taxonomy obtained using the proposed technique in comparison to other similar techniques. Two sets of experiments were performed, the details of which are given below:

5.2.1 Experiment for generation process

We compared the generation part of our technique with an existing technique TaxGen [22]. Taxonomy generation process using proposed technique and TaxGen was initially applied to 220 documents. The dataset was gradually increased by adding 30, 40, 50 and 60 documents into the system. Whenever the dataset was increased, the process of taxonomy generation was applied on the dataset using both the techniques to generate a taxonomy. The hierarchical clustering quality of the taxonomies generated using both the techniques is shown in Tables and 5.2. Running time of taxonomy generation process using the proposed technique and TaxGen is shown in Table 5.3.

Dataset	Proposed Technique	TaxGen
220	0.571	0.447
250	0.577	0.465
290	0.549	0.485
340	0.535	0.482
400	0.548	0.482

Table 5.1 Results For Quality-Based Evaluation Using Silhouette’s Score- Taxonomy Generation

Table 5.1 Results For Quality-Based Evaluation Using Silhouette’s Score- Taxonomy Generation shows the comparitibility of both the techniques based on the hierarchical clustering quality using the Silhouette’s score. Ideally the values should be between [-1, +1]. As mentioned earlier that the values inclined more towards 1 show better clustering quality and support the fact that the document has matched to a correct cluster. It can be observed that the Silhouette’s scores of the proposed techniques are higher than the values of TaxGen, showing that the clustering quality obtained using the proposed technique is better than that of TaxGen.

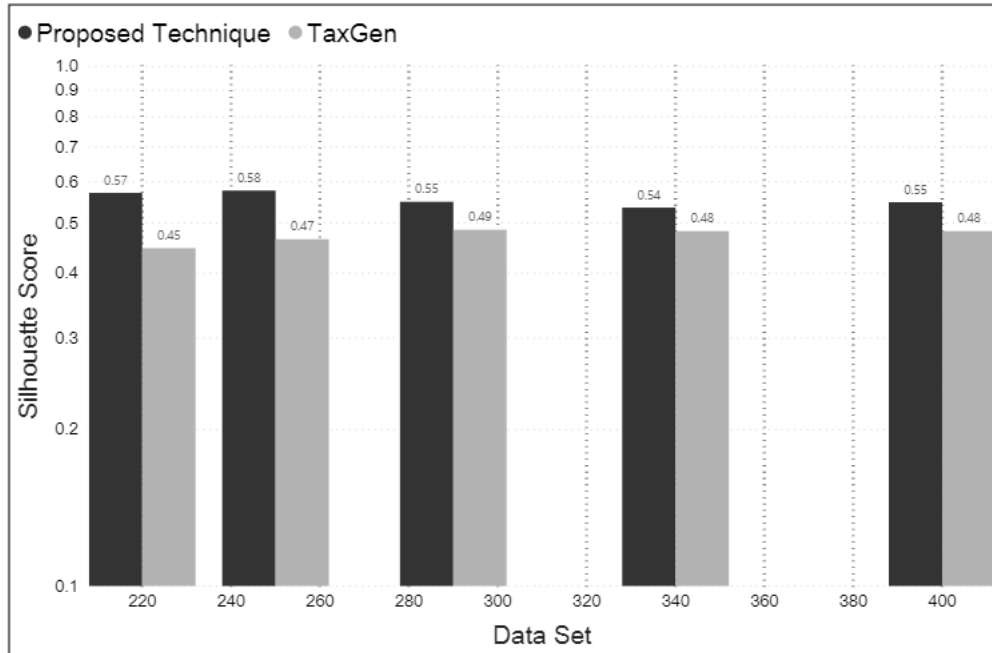


Figure 5.1 Silhouette Score for Proposed Technique and TaxGen

Figure 5.1 shows Silhouette score for TaxGen and the Proposed Technique against the dataset. It can be seen that the values of Silhouette Score for Proposed Technique are greater than that of TaxGen hence they show better clustering quality. A higher score suggests that the document has been correctly matched to its own cluster hence poorly matched to the neighboring clusters.

Dataset	Proposed Technique	TaxGen
220	0.303	0.734
250	0.363	0.755
290	0.592	0.728
340	0.363	0.732
400	0.595	0.687

Table 5.2 Results For Quality-Based Evaluation Using Davies-Bouldin's Score- Taxonomy

Table 5.2 Results For Quality-Based Evaluation Using Davies-Bouldin's Score- Taxonomy shows the comparison of both the techniques based on hierarchical clustering using Davies-Bouldin's score. As mentioned earlier that Davies-Bouldin's score states that lower the score better the clustering quality. It can be seen from the table that scores for the presented technique are smaller

in comparison with the TaxGen’s scores. Hence showing that the proposed technique is generating taxonomy with better clustering quality.

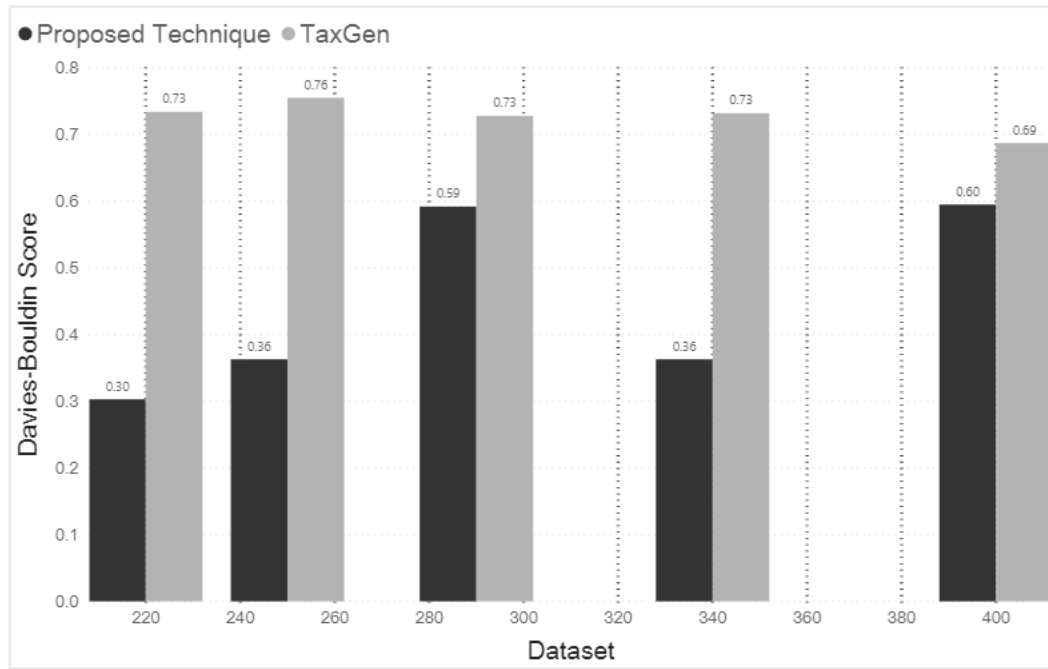


Figure 5.2 Davies Bouldin for Proposed Technique and TaxGen

Figure 5.2 Davies Bouldin for Proposed Technique and TaxGen shows Davies Bouldin score for TaxGen and Proposed Technique against the Dataset. The values of Davies Bouldin Score for Proposed Technique are lower than that of TaxGen hence they show better clustering quality.

Dataset	Proposed Technique (seconds)	TaxGen (seconds)
220	68.4	1914
250	74.4	2436
290	84	2826
340	98.4	3132
400	115.2	3618

Table 5.3 Results For Time Based Evaluation Of Runtime Of Proposed Technique

Table 5.3 shows the comparison based on running time of both the techniques. Time-based evaluation shows the running time for taxonomy generation using the proposed technique is much

smaller as compared with TaxGen. Hence the proposed technique would generate taxonomy faster as compared with TaxGen.

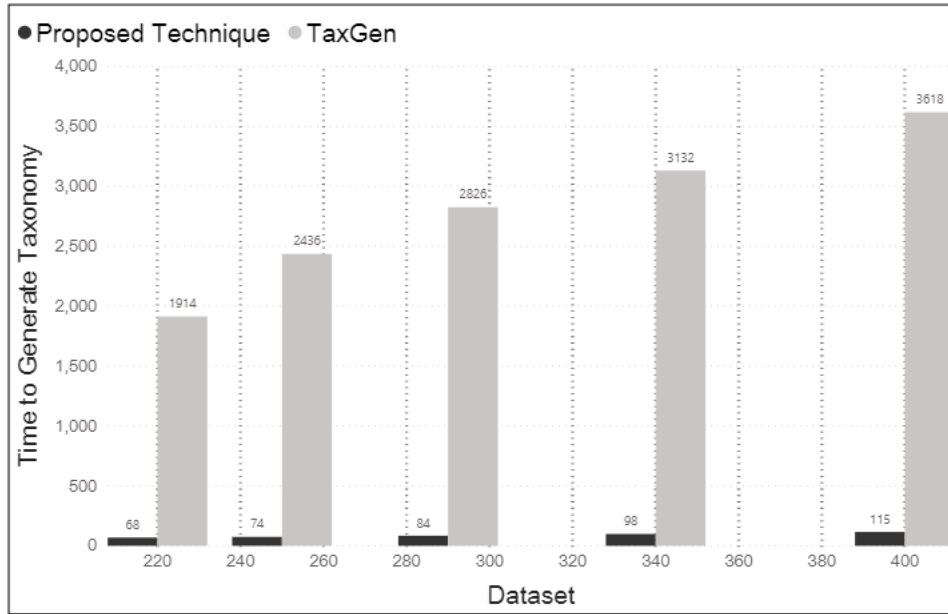


Figure 5.3 Time To Generate The Taxonomy Proposed Technique Vs Taxgen

Figure 5.3 shows the time of taxonomy generation process of Proposed Technique and running time of TaxGen. It can be seen that the time taken by Proposed Technique to generate taxonomy is much smaller then that of TaxGen.

5.2.2 Experiment for evolution process:

The evolution part of our technique was compared with an existing taxonomy evolution technique TIE [31]. Taxonomy evolution process using proposed technique and TIE was initially applied to 200 documents to generate a taxonomy. The dataset was gradually increased by adding 20, 30, 40, 50 and 60 documents into the system and taxonomy evolution was applied using both the techniques. The hierarchical clustering quality of the taxonomies evolved using both the techniques were compared, as shown in Tables 5.4 and 5.5. Running time of taxonomy evolution process using the proposed technique and TIE was also compared, as shown in Table 5.6.

Dataset	Proposed Technique	TIE
200+20	0.598	0.459
220+30	0.577	0.494
250+40	0.549	0.563
290+50	0.535	0.572

Dataset	Proposed Technique	TIE
340+60	0.595	0.626

Table 5.4 Results For Quality-Based Evaluation Using Silhouette’s Score- Taxonomy Generation

Table 5.4 shows the comparison of both the techniques on the basis of hierarchical clustering quality using the Silhouette’s score. The accepted values of Silhouette’s score should be between [-1, +1]. Values inclined more towards 1 show better clustering quality and support the fact that the document has matched to a correct cluster. It can be seen that the Silhouette’s scores of the proposed techniques are higher than the values of TIE in majority of the cases, hence showing that the clustering quality achieved with the proposed technique is better than TIE.

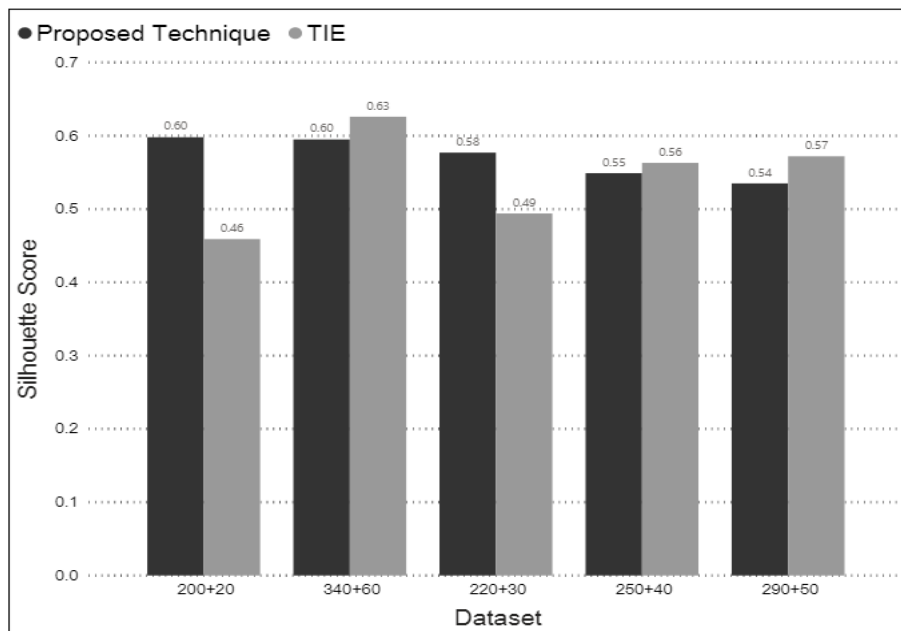


Figure 5.4 Silhouette Score for Proposed Technique and TIE

Figure 5.4 shows Silhouette score for TaxGen and Proposed Technique against the Dataset. It can be seen that the values of Silhouette Score for Proposed Technique are higher than that of TIE hence they show better clustering quality.

Dataset	Proposed Technique	TIE
200+20	0.447	0.688
220+30	0.465	0.734
250+40	0.485	0.759
290+50	0.482	0.637

Dataset	Proposed Technique	TIE
340+60	0.482	0.765

Table 5.5 Results For Quality-Based Evaluation Using Davies-Bouldin’s Score- Taxonomy Evolution

Table 5.5 shows the comparison on the basis of hierarchical clustering quality using Davies-Bouldin’s score. Lower scores show better clustering quality. It can be observed that values of Davies-Bouldin’s scores for the proposed technique are smaller than TIE’s scores. This shows that the clustering quality of the presented technique is superior as compared with TIE’s.

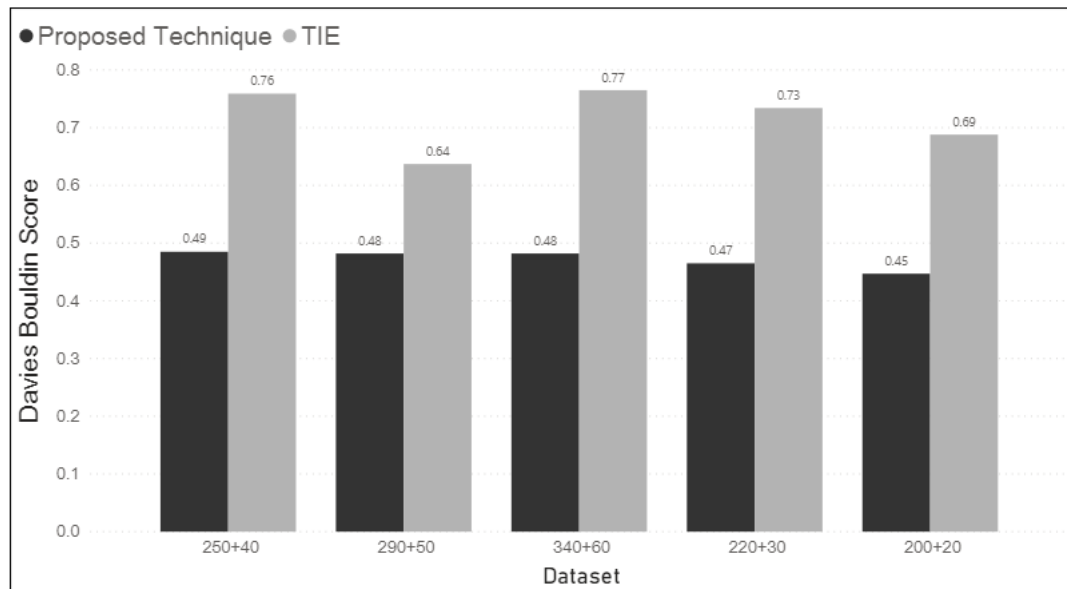


Figure 5.5 Davies Bouldin for Proposed Technique and TIE

Table 5.5 shows Davies Bouldin Score for TaxGen and Proposed Technique against the Dataset. It can be seen that the values of Silhouette Score for Proposed Technique are lower than that of TIE hence they show better clustering quality.

Dataset	Proposed Technique (seconds)	TIE(seconds)
200+20	4.44	426
220+30	4.08	624
250+40	7.86	708
290+50	11.34	1014
340+60	11.82	1386

Table 5.6 Results For Time-Based Evaluation-Taxonomy Evolution

Table 5.6 shows the comparison based on running time for evolution of taxonomy for both the techniques. Results show that the running time for taxonomy evolution using the proposed technique is much smaller as compared with TIE.

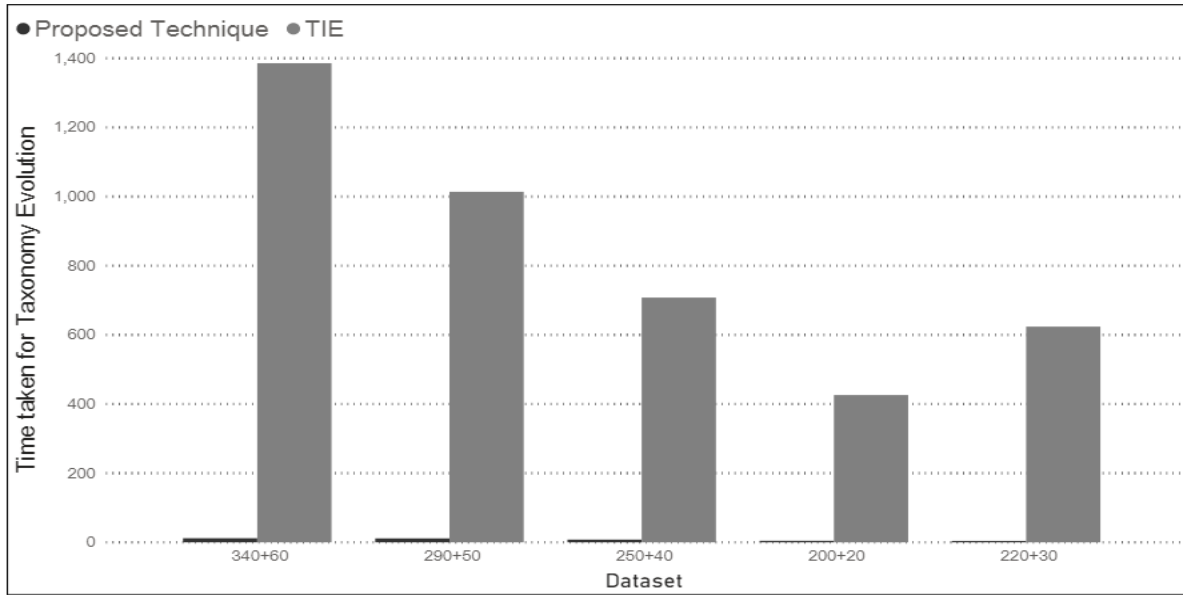


Figure 5.6 Time Taken To Evolve Taxonomy Using Proposed Technique and TIE

Figure 5.6 Time Taken To Evolve Taxonomy Using Proposed Technique and TIE shows the running time of taxonomy Evolution process of Proposed Technique and running time of TIE. It can be seen that the time taken by Proposed Technique in order to evolve taxonomy is much smaller then that of TIE.

5.2.3 Running time evaluation for different cores of apache spark

Apache Spark comes with various units and sub-units that aid in the process of running of Spark Jobs. By tuning the resources, parallelism, and using different data representation affect Spark job performance. The schema of the data (the way data is arranged) and number of cores for running a job are important factors.

Number of cores to be used can be implied with the `--executor-cores` flag when submitting a spark job. A spark job is submitted by invoking `spark-submit`. In `pyspark` `--executor-cores` flag are set from the command line. This can also be achieved by setting the `spark.executor.cores` property in the `spark-defaults.conf` file or on a `SparkConf` object.

In this part of experiment we ran our proposed Taxonomy evolution process using different number of cores in spark. Ideally in practice either 3, 5 or 8 cores are used for submitting and

running the Spark jobs. Initially 100 textual documents were used to generate taxonomy then 100 textual documents were added at each step for the process of evolution of taxonomy. Each step is evolving the taxonomy evolved in the previous step. Running time of taxonomy evolution process using the proposed technique against different number of Apache Spark cores, as shown in Table 5.7 Time Taken To Evolve Taxonomy Using Different Cores of Apache Spark.

Size of data	Time Taken		
	3 cores	5 cores	8 cores
100+100	7.97	6.79	6.29
200+100	6.29	7.12	6.25
300+100	8.15	8.29	8.08
400+100	10.11	9.72	9.83
500+100	12.28	12.09	11.99
600+100	14.23	14.06	13.68
700+100	16.68	15.98	15.44
800+100	18.57	18.07	17.69
900+100	19.48	19.87	18.46
1100+100	29.73	24.56	23.34
1200+100	34.97	26.78	27.12
1300+100	39.14	30.89	29.87
1400+100	42.96	32.02	30.38
1500+100	44.67	35.44	33.43
1600+100	45.23	37.89	36.32
1700+100	45.76	39.34	38.56
1800+100	46.26	41.56	40.53
1900+100	47.55	43.05	42.67

Table 5.7 Time Taken To Evolve Taxonomy Using Different Cores of Apache Spark

In Table 5.7 it can be seen that when taxonomy evolution process was ran on different number of Spark cores, the running time of algorithm for 8 cores gives the minimum time. When we specify the number of cores to be 8 that means each executor runs 8 tasks at a given time.

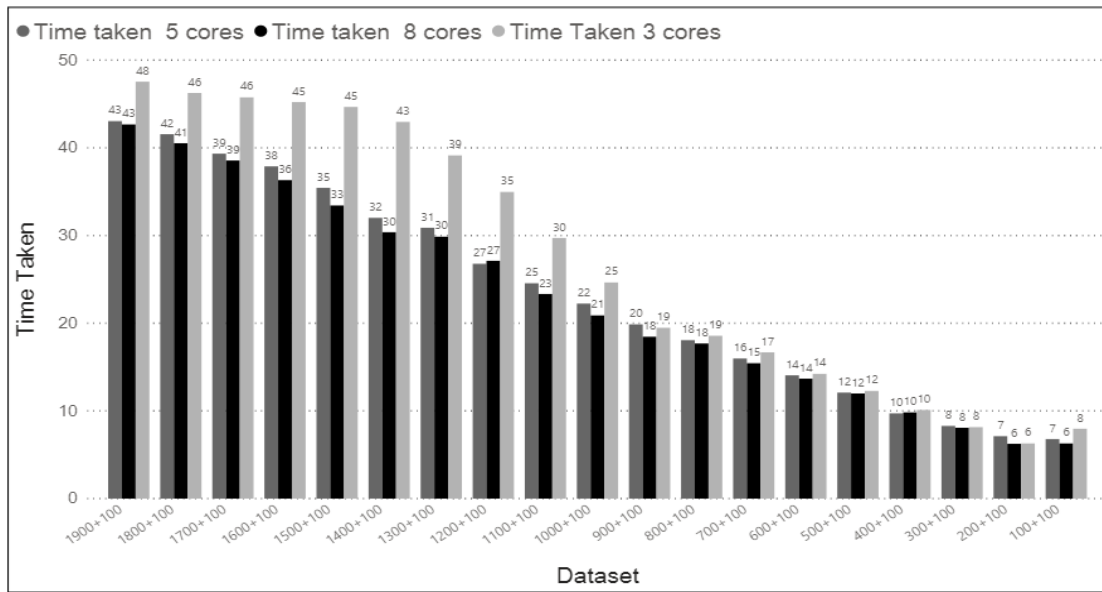


Figure 5.7 Time Taken To Evolve Taxonomy Using Different Cores of Apache Spark

The plot in the Figure 5.7 shows, that the running time for 8 cores is least however the running time using 5 cores is competing with 8 cores. It should be noted here initially when dataset is small the taken by all three cores to evolve taxonomy is more or less compareable but as the data set increases the significant difference can be seen in the running time. The impact of using different cores can be better visualized when dataset is even large.

5.2.4 Running time evaluation on Apache Hadoop Vs Apache Spark

In this part of experiment focus has been made on the performance and scalability of the two systems namely, Apache Hadoop and Apache Spark. Running time of the parallization part of algorithm was compared on both; Apache Hadoop as well as on Apache Spark. For the sake of this experiment we only generated the taxonomy Newick Trees were not generated for this experiment. The algorithm was ran on 3 cores of Apache Spark and compared it to the processing capability of Apache Hadoop. 3 cores were chosen because that is the minimum number of cores that can be chosen for running of any Spark job. The performance of the algorithm was evaluated using the time measures.

Dataset	Time Taken on Apache Spark	Time taken on Apache Hadoop
200	7.97	241.8
300	6.29	369.6
400	8.16	571.8

Dataset	Time Taken on Apache Spark	Time taken on Apache Hadoop
500	10.11	616.2
600	12.28	673.8
700	14.23	842.4
800	16.68	963
900	18.57	1083.6
1000	19.48	1228.8
1500	42.96	1830.6
2000	45.55	2310

Table 5.8 Results For Time-Based Evaluation – Running Time Of Algorithm On Apache Hadoop Vs Apache Spark

Table 5.8 Results For Time-Based Evaluation – Running Time Of Algorithm On Apache Hadoop Vs Apache Spark and Figure 5.8 shows the running time of the algorithm on both the environments. It can be seen that the running time of Apache Spark is much smaller as compared with Apache Hadoop. This can also be visualized in the vast difference between the running time of Apache Spark

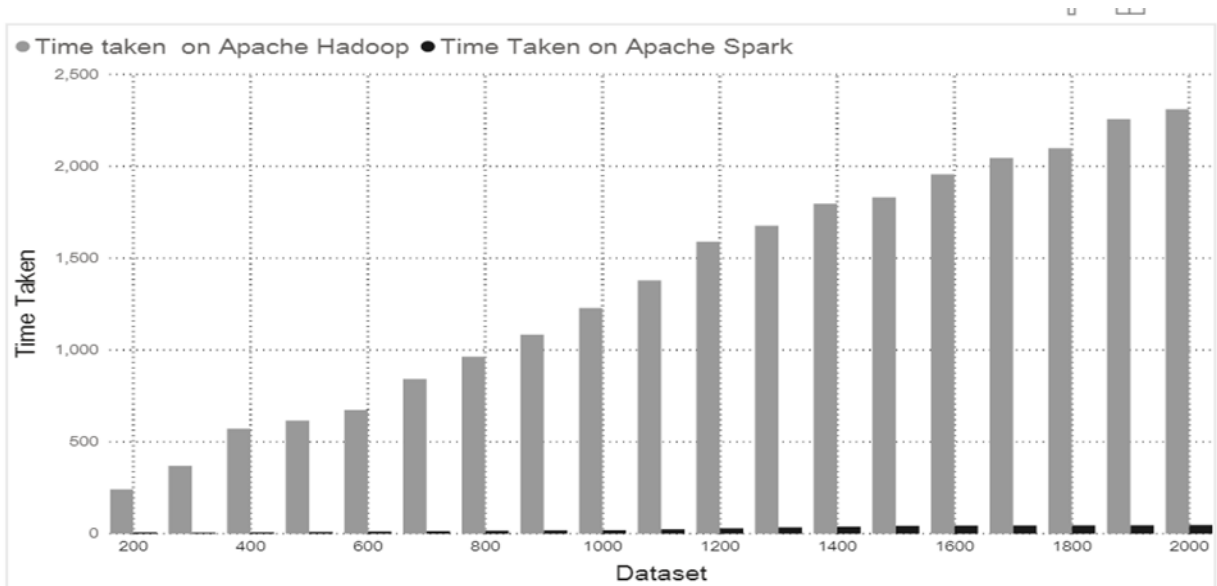


Figure 5.8 Results for Time-Based Evaluation – Running Time of Algorithm on Apache Hadoop Vs Spark

and Apache Hadoop. Insights drawn from this experiment are given below:

1 Running Time Evaluation

On the basis of running time of the technique on both the environments, their *Running Time-ratio* was calculated. The sum of running time of the algorithm were considered for Apache Spark And Apache Hadoop both and their ratio was taken. According to this time ratio, Spark is 53.05 times faster than that of Hadoop.

$$\text{Running Time – ratio} = \frac{\sum \text{Running Time of Algorithm on Hadoop}}{\sum \text{Running Time of Algorithm on Spark}} = 53.05$$

This makes Apache Spark suitable to be used for data clustering problems i-e Taxonomy. Although Apache Hadoop is also relatively faster as compared with the existing technique's results but Apache Spark has the least running time.

2 Performance Evaluation

According to the experiment Apache Spark has been found to run faster than Apache Hadoop. The reason for the speed of Spark are the in-memory calculations using RDD. Whereas Hadoop performs the calculation using the I/O reads and write from the disk. In memory computations make Spark stand out.

Performance, as measured by processing speed through experiments it was found that Spark performance is better in comparison with Hadoop, for following reasons:

- a) Spark performs computation using in-memory calculation hence no need to perform the I/O operations every time a job is running. Hence proving to be much faster for many applications especially for a clustering based problem.
- b) Spark's directed acyclic graphs enable optimizations between steps whereas Hadoop doesn't have any cyclic connection between MapReduce steps and levels. This means performance tuning cannot be done at that level.

5.3 Discussion

In this chapter the research work has been evaluated by comparison with an existing algorithm of Taxonomy Generation i-e TaxGen. Evolution part of the technique has also been compared with another algorithm of incremental taxonomy generation i-e TIE. The technique has been evaluated based on the running time and clustering quality. Clustering quality was evaluated using the Silhouette Score and Davies Bouldin Score. It was observed that the proposed technique shows better clustering quality as compared with the TaxGen and TIE. It can also be seen that the running time of the presented technique is much smaller as compared with the counter parts. The technique was also evaluated by running it on Apache Spark and Apache Hadoop both and their running time was compared. It was found that Apache Spark generated taxonomy in much smaller time as compared with Apache Hadoop. So it can be said that for a clustering problem like taxonomy

generation Apache Spark is much better choice. It was also evaluated by experiment that by using how many cores of Apache Spark the proposed technique can generate and evolve taxonomy faster. It was found that when data size is small number of cores do not matter. As the size of data grows using 8 cores can bring significant time improvement. The execution time taken for the analyses performed is critical in big data applications. The execution time is measured in order to evaluate the performance. Smaller execution times indicates that the program runs fast and gives good performance. It should also be noted that the proper resource utilization is also crucial in case of large datasets. A good application should give high performance with minimal resource utilization. Since the technique utilizes MapReduce algorithm as its core technique while running on Apache Spark, this makes the technique scalable. The next chapter concludes this research work.

Chapter 6. Conclusion

This chapter provides a detailed discussion about how the research objectives are being addressed by the work carried out in this thesis and highlights its contributions. Moreover, it also mentions the limitations of this work along with the future directions. The chapter is organized as follows: Section 6.1 presents the discussion on Research Gap, Section 6.2 summarizes the research objectives, Section 6.3 Presents the contributions and Section 6.4 presents the future work.

6.1 Discussion on research gap

This work inspects existing taxonomy generation and evolution techniques with the perspective of big data of unstructured text documents. By careful inspection of already existing techniques for taxonomy generation and evolution a research gap was identified. It was found that current techniques do not incorporate the ever increasing volume of big data. There is no technique so far that utilizes the parallelization framework for incremental taxonomy generation for big data of unstructured textual documents. Based on this identified research gap, research objectives were formulated. This work presented a Taxonomy Generation and Evolution technique that runs on Apache Spark and incorporates the ever increasing Big Data of unstructured text documents.

6.2 Discussion on research objectives

Based on the identified research gap i-e is current techniques do not incorporate the ever increasing volume of data, research objectives were formulated. These research objectives are given below:

1 Research Object 1 - To solve the scalability issues of current clustering-based incremental taxonomy generation algorithms

The pre-existing taxonomy generation techniques were generating and updating the taxonomy using considerable amount of textual data. These pre-existing techniques were not considering the fact of ever increasing big data of unstructured documents that needs to be organized into a hierarchical structure. The proposed algorithm solves the problem of scalability of current clustering-based incremental taxonomy generation algorithms by proposing a technique that is built on MapReduce paradigm and runs on Apache Spark, i-e is a tool for big data processing. The proposed technique runs the algorithm in the form of smaller Map and Reduce tasks which makes the processing on big data sets to be possible in relatively short time. Running of the algorithm in small tasks of Map and Reduce, makes it scalable for running on even bigger data sets. The technique successfully generates and evolves the taxonomy of big data of unstructured text documents, in considerably shorter time as compared with the pre-exisging techniques. Hierarchical structure is a basically the base of a taxonomic structure. Different kind of approaches have been used previously in order to build the hierarchal structure. Previous algorithm have been using different approaches for building the hierarchical structure such as Clustering Approaches, Graph Based Approach, Heuristic Based Approach, Rule Based

Approach. The proposed algorithm is based on Hierarchical Agglomerative Clustering Technique i-e Ward Method. Hierarchical clustering is used here because the proposed algorithm is unsupervised. Since the arriving Big Data is fast and in large amount, we can't specify pre-hand that how many clusters our hierarchical structure will have that is why the technique needs to be unsupervised.

2 Research Object 2 - To propose/improve a scalable algorithm to update/evolve taxonomy in presence of a new document

This work has proposed an incremental taxonomy generation technique using MapReduce on Apache Spark. This technique works by generating the initial taxonomy using the textual data that is initially available and converts the obtained Taxonomy into a tree graph. On introduction of new documents in to the system, the proposed technique updates the previously generated taxonomy by incorporating the new documents in the pre-existing tree-graph. The proposed technique generates an initial taxonomy by running the taxonomy generation process. After the generation of an initial taxonomy or a hierarchical structure, it converts it into a tree graph. This tree graph is called a Newick tree. When new documents are introduced in to the system the process of taxonomy generation is ran on the newly introduced documents a new hierarchical structure is constructed. This hierarchical structure is then converted into a Newick Tree. Now the this new Newick tree is merged with the already existing Newick tree using NJMerge algorithm NJMerge Merges the two Newick trees and gives the final merged taxonomy.

3 Research Objective 3: Explore the comparison of Apache Hadoop and Apache Spark using the proposed technique

Existing methods and techniques for taxonomy generation and evolution were not focusing on the use of the concept of parallelization for making an efficient and scalable algorithm for taxonomy generation and evolution. Hence there was a need of taxonomy generation and evolution technique that handles the fast arriving big data of documents and arrange it in a hierarchical structure so insightful information can be drawn from it. So in order to deal with the fast arriving big data, the technique was developed to run on MapReduce big data platform i-e Apache Spark. Since there is another popular MapReduce environment for big data i-e Apache Hadoop. The taxonomy generation part of the algorithm was also ran on Apache Hadoop in order to evaluate it performance. This was also done in order to find out the best big data environment to be used for a clustering problem such as taxonomy generation. Only generation part of the algorithm was tested o on Apache Hadoop as it does not provide machine learning that was required for the taxonomy evolution part of the algorithm hence only generation part of the proposed algorithm was tested. Through experiments it was found that Apache Spark is faster and well suited for a clustering problem like taxonomy generation as compared with Apache Hadoop.

4 Research Objective 4: Validate the significance of the obtained result

The detailed evaluations of the have been performed on the proposed technique in comparison to different Non-Incremental (TaxGen) and an Incremental Taxonomy Generation/Evolution Technique (TIE) .Time for running of the technique and the quality of produced taxonomy are important parameters to show the efficiency of any proposed technique. Hence time and clustering quality evaluations were adopted in order to check the validity and efficiency of the proposed technique. Reason for considering the time and quality measures are as follows:

- 1) The technique should give updated view of the taxonomy in considerable less amount of time. It should be updated with the speed of big data of new documents in to the system.
- 2) The evolved taxonomy should give true representation of the underlying data this can be checked with the clustering quality. Clustering quality parameter checks, whether a document is matched/added to its true cluster. Inter-cluster similarity for the documents need to be high.

This work is focused on unstructured textual data for this purpose scholarly articles from computing domain was selected for evaluation. The documents were divided into different folders. Base Taxonomy was built using the initial number of documents. After that new documents were introduced into the system, from the pre-divided folders.

The time-based evaluation shows that the proposed technique is taking comparatively less time to generate a Taxonomy as compared with TaxGen. The time based evaluations also show that the adjust new documents in an existing taxonomy as compared to incremental counterpart the proposed technique is taking considerably less time. In order to check the clustering quality Davies Bouldin Score and Silhouette Score was used. The obtained scores of the clustering indices for the Generation Process and Evolution Process show that the presented technique performs much better as compared with Incremental and Non-Incremental algorithms.

The environment, used for running of the proposed technique is Apache Spark, in order to compare its, running time with another Big Data Processing tool, the algorithm was also tested on Apache Hadoop to check the algorithm's performance on both the systems. The comparison was also drawn to identify the best big data tool for a clustering problem. Running time of the algorithm on Apache Spark is much smaller and efficient as compared with Apache Hadoop. The algorithm was also evaluated using the data-load performance test against different number of cores used in Apache Spark. The algorithm was ran on different sizes of data sets on 3, 5 and 8 cores of Apache Spark. This test was done in order to find out the optimal number of cores for running of the algorithm in order to achieve most efficient performance.

Based on the detailed and comprehensive evaluations it can be said that the proposed algorithm generates the taxonomy in considerably less time and also evolves and updates it in small amount of time. The Generation and Evolution is done by maintaining good quality of clustering in the underlying structure. Thus, it can result in an effective utilization of Taxonomy.

6.3 Contributions

The taxonomy generation and incremental taxonomy evolution technique proposed in this work is a novel solution that not only generates the taxonomy for big data of unstructured text data but also evolves the existing taxonomy whenever new documents are added into the system. The contributions of the work is as follows:

- ✓ The proposed technique generates a taxonomy in a shorter period of time as compared with existing taxonomy generation techniques that can make Taxonomy utilization more effective.
- ✓ The proposed technique evolves taxonomy in presence of new documents in to the system. This update is carried out in considerably less time as compared with the existing techniques. The proposed technique's Evolution time in in seconds, whereas previous algorithms were taking time in minutes and hours for the process.
- ✓ As clustering is the base of taxonomy generation algorithm, the clustering quality of the taxonomy generated from the proposed technique is comparable to clustering quality of the taxonomy generated using the existing taxonomy generation techniques. According to the Davies Bouldin Score and Silhouette Score, the clustering quality of the presented technique is higher than the existing techniques.
- ✓ The technique was ran on Apache Hadoop and Apache Spark, the technique gives a good comparison between the two environments and proves that for hierarchical clustering of big data, Apache Spark is much better option.
- ✓ The proposed technique was ran on different configurations of Apache Spark, it gives the optimal number of cores for running any hierarchical clustering jobs on Apache Spark.
- ✓ The proposed technique can update a graph taxonomy if given as input.

6.4 Limitation of the work and future direction

There is always a room for improvement in case of all kind of scientific works. Just like that there is a room for improvement in this work as well. The labels generated using this technique are not crisp as labels that are selected by human effort. Labeling technique can be further improved. Machine learning techniques may be used for further improvement of the labeling technique for the generated taxonomy. The proposed algorithm can only evolves a taxonomy that has been converted in to a Tree graph, in future further work can be done in order to incorporate taxonomy evolution process for any kind of taxonomy that is given as an input. There is a need for building

an application layer or a user interface on top of the existing technique. Currently this technique has no application layer. It runs in the form of a code and can be used only by a programmer. In future the application side for this proposed technique will be further explored in order to make it usable by any kind of user.

Appendix A: Algorithm For The Proposed Technique

Generation Part of Algorithm

Input: Unstructured Textual Documents \vec{d} in Corpora D'

Output: Generated taxonomy T_{gen}

Process:

1. **Load D' in RDD**
2. **for each $\vec{d} \in D'$ do**
3. Remove Stop words \rightarrow NLTK(\vec{d})
4. Apply Stemming to terms \rightarrow Snowball Stemmer (\vec{d})
5. **end for**
6. **for D' do**
7. calculate vector space model
8. output: D_{vsm}
9. **end for**
10. **for D_{vsm} do**
11. calculate similarity (\vec{d}_1, \vec{d}_n)
12. similarity matrix (\vec{d}_1, \vec{d}_n) $\rightarrow D_{sim}$
13. (\vec{d}_1, \vec{d}_n) \rightarrow maximum similarity \rightarrow cluster formation (c_n)
14. **end for**
15. **for c_n**
16. **if distance (c_n, c_m) \leq Ward Method Objective Function**
17. **merge cluster (c_n, c_m)**
18. **end if**
19. **end for**
20. **hierarchy formation (c_n, \dots, c_m) $\rightarrow H$**
21. **for each c_n do**
22. **calculate $C_{centroid}$**
23. **label $c_n(\text{Title}_{c_{centroid}})$**
24. **end for**
25. **convert hierarchy to Newick Tree \rightarrow NewickTree(H) $\rightarrow T_{gen}$**

Evolution Part of Algorithm

Input: Generated taxonomy T_{gen} , Document similarity matrix D_{sim} , Newly introduced unstructured text documents \vec{d}_{new} and document corpora D'_{new}

Output: Evolved Taxonomy T_{evo}

1. **Load** D'_{new} **in RDD**
2. **for each** $\vec{d}_{new} \in D'$ **do**
3. Remove Stop words \rightarrow **NLTK**(\vec{d}_{new})
4. Apply Stemming to terms \rightarrow **Snowball Stemmer** (\vec{d}_{new})
5. **end for**
6. **for** D'_{new} **do**
7. calculate vector space model
8. output: D_{vsm}
9. **end for**
10. **for** D_{vsm}
11. calculate similarity (\vec{d}_1, \vec{d}_n)
12. form similarity matrix (\vec{d}_1, \vec{d}_n) $\rightarrow D_{sim-new}$
13. (\vec{d}_1, \vec{d}_n) \rightarrow maximum similarity \rightarrow cluster formation (c_n)
14. **end for**
15. **for** c_n
16. **if** distance (c_n, c_m) \leq **Ward Method Objective Function**
17. **merge cluster** (c_n, c_m)
18. **end if**
19. **end for**
20. hierarchy formation (c_n, \dots, c_m) $\rightarrow H$
21. **convert hierarchy** to Newick Tree \rightarrow **NewickTree**(H) $\rightarrow T_{evo}$
22. **run NJMERGE** ($T_{evo}, T_{gen}, D_{sim}, D_{sim-new}$)
23. **Calculate branch lengths for** (T_{evo}, T_{gen}) $\rightarrow b_n$
24. **For each** b_n
25. **sum**($b_n + b_m$)
26. **end for**
27. **if**(**sum**(b_n, b_m) $<$ **sum**(b_n, b_p))
28. **merge**(b_n, b_m)
29. **end if**
30. **Output:** T_{merged}

Appendix B: Software and System Specification

B1: Software Specifications:

1. Python language (<https://www.python.org/>) is used as core language for this research work.
2. Apache Spark (<https://spark.apache.org/>) is the core environment used for the proposed technique.
3. For testing purpose Apache Hadoop (<https://hadoop.apache.org/>) was also used. For programming on Apache Hadoop, Java (<https://www.java.com/en/>) programming language was used.
4. For Data preprocessing, for stop word removal NLTK framework (<https://www.nltk.org/>) for python was used. For the stemming of terms Snowball stemmer (<https://www.nltk.org/howto/stem.html>) was used. Python library used was Sklearn (<https://scikit-learn.org/stable/>).
5. For dealing with data frames python library pandas was used (<https://pandas.pydata.org/>).
6. For dealing with mathematical and logical operations on python Library Numpy was used.
7. For evolution process and conversion into trees ete3 (<http://etetoolkit.org/>) python library was used.

B2: System Specifications:

The proposed technique was developed and evaluated using the following system specifications:

- 64 bit, core i5, 1.6 GHz/core processor
- 32 GB RAM
- 1 TB Hard Drive

References

- [1] V. Turner, "The digital universe of opportunities: rich aata and the increasing value of the internet of things," 2014. D. Laney, "3D data management: Controlling data volume, velocity, and variety," META Group, 2001.
- [2] M.-S. Paukkeri, A. P. Garcia-Plaza, V. Fresno, R. M. Unanue and T. Honkela, "Learning a taxonomy from a set of text documents," *Applied Soft Computing*, vol. 12, no. 3, pp. 1138-1148, 2012.
- [3] R. Sujatha and B. R. Krishna Rao, "Taxonomy construction techniques--issues and challenges," *Indian Journal of Computer Science and Engineering (IJCSE)*, vol. 2, no. 5, pp. 661-671, October-November 2011
- [4] H. Hedden, *The accidental taxonomist*, Information Today Inc., 2010, p. 464.
- [5] D. Sanchez and A. Moreno, "Automatic generation of taxonomies from the WWW," *Practical Aspects of Knowledge Management*, vol. 3336, pp. 208-219, 2004.
- [6] T. Li and S. S. Anand, "Exploiting domain knowledge by automated taxonomy generation in recommender systems," *E-Commerce and Web Technologies*, vol. 5692, pp. 120-131, 2009.
- [7] G. Dawelbait, T. Mezher, W. L. Woon and A. Henschel, "Taxonomy based trend discovery of renewable energy technologies in desalination and power generation," in *2010 Proceedings of Technology Management for Global Economic Growth (PICMET)*, Phuket, Thailand, 2010.
- [8] S.-S. Weng and C.-K. Liu, "Using text classification and multiple concepts to answer e-mails," *Expert Systems with Applications*, vol. 26, no. 4, pp. 529-543, May 2004.
- [9] W. S. Spangler, J. T. Kreulen and J. F. Newswanger, "Machines in the conversation: detecting themes and trends in informal communication streams," *IBM Systems Journal*, vol. 45, no. 4, pp. 785-799, Oct 2006.
- [10] S. L. Camina, *A comparison of taxonomy generation techniques using bibliometric methods: applied to research strategy formulation*, Massachusetts Institute of Technology, 2010.
- [11] V. Kashyap, C. Ramakrishnan, C. Thomas and A. Sheth, "TaxaMiner: an experimentation framework for automated taxonomy bootstrapping," *International Journal of Web and Grid Services*, vol. 1, no. 2, pp. 240-266, Dec 2005.
- [12] D. L. Boley, "Principal Direction Divisive Partitioning," *Data Mining and Knowledge Discovery*, Volume 2(4), 1998.
- [13] L. Espinosa-Anke, H. Saggion and F. Ronzano, "TALN-UPF: Taxonomy learning exploiting CRF-based hypernym extraction on encyclopedic definitions," in *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, Denver, Colorado, 2015.

- [14] P. Velardi, S. Faralli and R. Navigli, "OntoLearn reloaded: A graph-based algorithm for taxonomy induction," *Computational Linguistics*, vol. 39, no. 3, pp. 665-707, 2013. Available: [10.1162/coli_a_00146](https://doi.org/10.1162/coli_a_00146) [Accessed 14 February 2020].
- [15] L. Tang, H. Liu, J. Zhang, N. Agarwal and J. J. Salerno, "Topic taxonomy adaptation for group profiling," *ACM Transactions on Knowledge Discovery from Data*, vol. 1, no. 4, pp. 1--28, February 2008.
- [16] J. Yao, B. Cui, G. Cong, and Y. Huang, "Evolutionary taxonomy construction from dynamic tag space," *World Wide Web*, vol. 15, no. 5-6, pp. 581-602, 2012.
- [17] R. M. Marcacini and S. O. Rezende, "Incremental construction of topic hierarchies using hierarchical term clustering," in *Proceedings of the 22nd International Conference on Software Engineering & Knowledge Engineering (SEKE)*, Redwood City, California, USA, 2010.
- [18] R. Irfan and S. Khan, "TIE: An algorithm for incrementally evolving taxonomy for text data," *2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA)*, Anaheim, CA, 2016, pp. 687-692, 2016.
- [19] X. Wu, X. Zhu, G.-Q. Wu and W. Ding, "Data mining with big data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 1, pp. 97-107, 2014.
- [20] A. McAfee, E. Brynjolfsson and T. H. Davenport, "Big data: the management revolution," *Harvard business review*, vol. 90, no. 10, pp. 60-68, 2012.
- [21] A. Katal, M. Wazid and R. Goudar, "Big data: issues, challenges, tools and good practices," in *2013 Sixth International Conference on Contemporary Computing (IC3)*, NOIDA, (outskirts of New Delhi), India, 2013.
- [22] R. Blumberg and S. Atre, "The problem with unstructured data," *DM REVIEW*, vol. 13, pp. 42-49, 2003.
- [23] Jeffrey Dean and Sanjay Ghemawat. *MapReduce: Simplified data processing on large clusters*. In *OSDI*, 2004.
- [24] K. Rohloff and R. Schantz, "High-performance, massively scalable distributed systems using the MapReduce software framework," *Programming Support Innovations for Emerging Distributed Applications on - PSI EtA '10*, 2010.
- [25] A. Muller, J. Dorre, P. Gerstl and R. Seiffert, "The TaxGen framework: automating the generation of a taxonomy for a large document collection," in *Proceedings of the 32nd Annual Hawaii International Conference on Systems Sciences (HICSS-32)*, Maui, Hawaii, USA, 1999.
- [26] E.-A. Dietz, D. Vandic and F. Frasinca, "TaxoLearn: A semantic approach to domain taxonomy learning," in *2012 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, Macau, China, 2012.

- [27] W. Wu, H. Li, H. Wang and K. Q. Zhu, "Probase: a probabilistic taxonomy for text understanding," in Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data (ICMD), Scottsdale, Arizona, USA, 2012.
- [28] L. A. Tuan, J.-j. Kim and K. S. Ng, "Taxonomy construction using syntactic contextual evidence," in Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Doha, Qatar, 2014.
- [29] L. Tang, H. Liu, J. Zhang, N. Agarwal and J. J. Salerno, "Topic taxonomy adaptation for group profiling," *ACM Transactions on Knowledge Discovery from Data*, vol. 1, no. 4, pp. 1--28, February 2008.
- [30] A. K. Jain, "Data clustering: 50 years beyond K-means," *Pattern Recognition Letters*, vol. 31, no. 8, pp. 651-666, June 2010.
- [31] R. Irfan and S. Khan, "TIE: An algorithm for incrementally evolving taxonomy for text data," 2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA), Anaheim, CA, 2016, pp. 687-692, 2016.
- [32] Delgado, Hugo. (2019). Taxonomy Organization of information for Web Content. Retrieved Apr 15, 2020, from <https://disenowebakus.net/en/taxonomy-information-web-content>
- [33] C. Khoo, W. Z. and C. A.S., "Task-based navigation of a taxonomy interface to a digital repository," *Information Research*, vol. 17, no. 4, 2013.
- [34] G. M. Sacco, "Dynamic taxonomies and guided searches," *Journal of the American Society for Information Science and Technology*, vol. 57, no. 6, pp. 792-796, April 2006.
- [35] B. Pereira, C. Robin, T. Daudert, J. McCrae, P. Mohanty and P. Buitelaar, "Taxonomy Extraction for Customer Service Knowledge Base Construction", *Lecture Notes in Computer Science*, pp. 175-190, 2019. Available: 10.1007/978-3-030-33220-4_13 [Accessed 15 April 2020].
- [36] Rodrigues, Ramon & Camilo-Junior, Celso & Couto, Thierson. (2018). A Taxonomy for Sentiment Analysis Field. *International Journal of Web Information Systems*. 14. 00-00. 10.1108/IJWIS-07-2017-0048.
- [37] Machine Learning for E-mail Spam Filtering: Review, Techniques and Trends - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/A-taxonomy-of-e-mail-spam-filtering-techniques_tbl1_303812063
- [38] Tao Li and M. Ogihara, "Music genre classification with taxonomy," Proceedings. (ICASSP '05). IEEE International Conference on Acoustics, Speech, and Signal Processing, 2005., Philadelphia, PA, 2005, pp. v/197-v/200 Vol. 5.
- [39] P. M. Nadkarni, L. Ohno-Machado and W. W. Chapman, "Natural language processing: an introduction," *Journal of the American Medical Informatics Association*, vol. 18, no. 5, pp. 544-551, 2011.
- [40] Webster, F. (2014). *Theories of the information society*. Routledge.

- [41] Yang, C., Yu, M., Hu, F., Jiang, Y., & Li, Y. (2017). Utilizing cloud computing to address big geospatial data challenges. *Computers, Environment and Urban Systems*, 61, 120-128.
- [42] Hu, H., Wen, Y., Chua, T. S., & Li, X. (2014). Toward scalable systems for big data analytics: A technology tutorial. *IEEE access*, 2, 652-687.
- [43] Golchha, N. (2015). Big data-the information revolution. *Int. J. Adv. Res*, 1(12), 791-794.
- [44] Sharma, S., Tim, U. S., Wong, J., Gadia, S., & Sharma, S. (2014). A brief review on leading big data models. *Data Science Journal*, 14-041.
- [45] Katal, A., Wazid, M., & Goudar, R. H. (2013, August). Big data: issues, challenges, tools and good practices. In *2013 Sixth international conference on contemporary computing (IC3)* (pp. 404-409). IEEE.
- [46] Egi, N., Greenhalgh, A., Handley, M., Hoerd, M., Huici, F., & Mathy, L. (2008, December). Towards high performance virtual routers on commodity hardware. In *Proceedings of the 2008 ACM CoNEXT Conference* (pp. 1-12).
- [47] Shvachko, K., Kuang, H., Radia, S., & Chansler, R. (2010, May). The hadoop distributed file system. In *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)* (pp. 1-10). Ieee.
- [48] Vavilapalli, V. K., Murthy, A. C., Douglas, C., Agarwal, S., Konar, M., Evans, R., ... & Saha, B. (2013, October). Apache hadoop yarn: Yet another resource negotiator. In *Proceedings of the 4th annual Symposium on Cloud Computing* (pp. 1-16).
- [49] J. Weets, M. K. Kakhani and A. Kumar, "Limitations and challenges of HDFS and MapReduce," *2015 International Conference on Green Computing and Internet of Things (ICGCIoT)*, Noida, 2015, pp. 545-549.
- [50] Zaharia, M., Xin, R. S., Wendell, P., Das, T., Armbrust, M., Dave, A., ... & Ghodsi, A. (2016). Apache spark: a unified engine for big data processing. *Communications of the ACM*, 59(11), 56-65.
- [51] García, S., Luengo, J., & Herrera, F. (2015). *Data preprocessing in data mining* (pp. 195-243). Cham, Switzerland: Springer International Publishing.
- [52] Brants, T. (2003, September). *Natural Language Processing in Information Retrieval*. In CLIN.
- [53] H.-C. Yang and C.-H. Lee, "A text mining approach on automatic generation of web directories and hierarchies," *Expert Systems with Applications*, vol. 27, no. 4, pp. 645-663, 2004.
- [54] S.-L. Chuang and L.-F. Chien, "Taxonomy generation for text segments: a practical web-based approach," *ACM Transactions on Information Systems (TOIS)*, vol. 23, no. 4, pp. 363-396, 2005.
- [55] Q. Zhao and S. S. Bhowmick, "Association rule mining: a survey," Singapore, 2003

- [56] A. Bluman, Elementary statistics a step by step approach, M. Rogers and W. J. Zeman, Eds., WCB McGraw-Hill, 1998.
- [57] J. de Knijff, F. Frasinca and F. Hogenboom, "Domain taxonomy learning from text: The subsumption method versus hierarchical clustering," Data and Knowledge Engineering, vol. 83, no. 0, pp. 54-69, 2013.
- [58] O. Medelyan, S. Manion, J. Broekstra, A. Divoli, A.-L. Huang and I. Witten, "Constructing a focused taxonomy from a document collection," in The Semantic Web: Semantics and Big Data, vol. 7882, P. Cimiano, O. Corcho, V. Presutti, L. Hollink and S. Rudolph, Eds., Springer Berlin Heidelberg, 2013, pp. 367-381.
- [59] P. Treeratpituk, M. Khabsa and C. L. Giles, "Graph-based approach to automatic taxonomy generation (GraBTax)," CoRR, vol. abs/1307.1718, 2013.
- [60] J. Woehler and F. Faerber, "Taxonomy generation for electronic documents". USA Patent US 7,243,092 B2, 2007.
- [61] H.-C. Yang, C.-H. Lee and H.-W. Hsiao, "Incorporating self-organizing map with text mining techniques for text hierarchy generation," Applied Soft Computing, vol. 34, pp. 251-259, September 2015.
- [62] E. Lefever, "LT3: a multi-modular approach to automatic taxonomy construction," in 9th International Workshop on Semantic Evaluation (SemEval 2015), Denver, Colorado, 2015.
- [63] L. Espinosa-Anke, H. Saggion and F. Ronzano, "TALN-UPF: taxonomy learning exploiting CRF-based hypernym extraction on encyclopedic definitions," in Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015), Denver, Colorado, 2015.
- [64] B. Ceesay and W. J. Hou, "NTNU: an unsupervised knowledge approach for taxonomy extraction," in Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015), Denver, Colorado , 2015.
- [65] P. Cimiano, A. Hotho and S. Staab, "Learning concept hierarchies from text corpora using formal concept analysis.," Journal of Artificial Intelligence Research (JAIR), vol. 24, no. 1, pp. 305-339, August 2005.
- [66] M. Neshati, A. Alijamaat, H. Abolhassani, A. Rahimi and M. Hoseini, "Taxonomy learning using compound similarity measure," in IEEE/WIC/ACM International Conference on Web Intelligence, Silicon Valley, California, USA, 2007.
- [67] S. P. Ponzetto and M. Strube, "Deriving a large scale taxonomy from wikipedia," in Proceedings of the 22nd National Conference on Artificial Intelligence - Volume 2, Vancouver, British Columbia, Canada, 2007.
- [68] P. Heymann and H. Garcia-Molina, "Collaborative creation of communal hierarchical taxonomies in social tagging systems," Stanford, California, USA, 2006.

- [69] T. Li and S. S. Anand, "Automated taxonomy generation for summarizing multi-type relational datasets," in 2008 International Conference on Data Mining (DMIN), Las Vegas, Nevada, USA, 2008.
- [70] N. Zong, D.-H. Im, S. Yang, H. Namgoon and H.-G. Kim, "Dynamic generation of concepts hierarchies for knowledge discovering in bio-medical linked data sets," in Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication (ICUIMC), Kuala Lumpur, Malaysia, 2012.
- [71] R. Baeza-Yates and B. Ribeiro-Neto, Modern information retrieval-the concepts and technology behind search, Second ed., Pearson Education Limited, 2011.
- [72] Li, H., Ma, B., & Lee, C. H. (2006). A vector space modeling approach to spoken language identification. *IEEE Transactions on Audio, Speech, and Language Processing*, 15(1), 271-284.
- [73] X. Liu, Y. Song, S. Liu and H. Wang, "Automatic taxonomy construction from keywords," in Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), Beijing, China, 2012.
- [74] K. Meijer, F. Frasincar and F. Hogenboom, "A semantic approach for extracting domain taxonomies from text," *Decision Support Systems*, vol. 62, pp. 78-93, 2014.
- [75] S.-S. Choi, S.-H. Cha and C. C. Tappert, "A survey of binary similarity and distance measures," *Journal of Systemics, Cybernetics and Informatics*, vol. 8, no. 1, pp. 43-48, 2010.
- [76] C. D. Manning, P. Raghavan and H. Schütze, Introduction to information retrieval, New York, NY, USA: Cambridge University Press, 2008.
- [77] M. S. G. Karypis, V. Kumar and M. Steinbach, "A comparison of document clustering techniques," in TextMining Workshop at KDD2000 (May 2000), Boston, Massachusetts, USA, 2000.
- [78] C. Blundell, Y. W. Teh and K. A. Heller, "Bayesian rose trees," arXiv preprint arXiv:1203.3468, 2012.
- [79] Navigli, R., Velardi, P., & Faralli, S. (2011, June). A graph-based algorithm for inducing lexical taxonomies from scratch. In *Twenty-Second International Joint Conference on Artificial Intelligence*.
- [80] X. Qi, D. Yin, Z. Xue and B. D. Davison, "Choosing your own adventure: automatic taxonomy generation to permit many paths," in Proceedings of the 19th ACM International Conference on Information and Knowledge Management (CIKM), Toronto, Canada, 2010.
- [81] T. Zhang, R. Ramakrishnan and M. Livny, "BIRCH: a new data clustering algorithm and its applications," *Data Mining and Knowledge Discovery*, vol. 1, no. 2, pp. 141-182, 1997.
- [82] G. Chartrand and O. R. Oellermann, Applied and algorithmic graph theory, McGraw-Hill, Inc., 1993.

- [83] J. Lafferty, A. McCallum and F. Pereira, "Conditional random fields: probabilistic models for segmenting and labeling sequence data," in Proceedings of the 18th International Conference on Machine Learning (ICML), Williamstown, Massachusetts, USA, 2001.
- [84] A. K. Jain, M. N. Murty and P. J. Flynn, "Data clustering: a review," *ACM Computing Surveys (CSUR)*, vol. 31, no. 3, pp. 264-323, 1999.
- [85] J. Vernau, "The business benefits of taxonomy," Kirkland, WA, USA, 2005.
- [86] W. Koff and P. Gustafson, "Data revolution," 2012.
- [87] Priss, U. (2006). Formal concept analysis in information science. *Annual review of information science and technology*, 40(1), 521-543.
- [88] L. Ertoz, M. Steinbach and V. Kumar, "Finding topics in collections of documents: a shared nearest neighbor approach," in *Clustering and Information Retrieval*, Springer, 2004, pp. 83-103.
- [89] Côté, J., Salmela, J. H., Baria, A., & Russell, S. J. (1993). Organizing and interpreting unstructured qualitative data. *The sport psychologist*, 7(2), 127-137.
- [90] Yasodha P, Ananathanarayanan NR. Analyzing Big Data to build knowledge based system for early detection of ovarian cancer. *Indian Journal of Science and Technology*. 2015 Jul; 8(14):1–7.
- [91] Subramaniaswamy, V., Vijayakumar, V., Logesh, R., & Indragandhi, V. (2015). Unstructured data analysis on big data using map reduce. *Procedia Computer Science*, 50, 456-465.
- [92] Abbasi A, Younis M. A survey on clustering algorithms for wireless sensor networks. *Computer Communications*. 2007 Dec; 30(14-15):2826–41.
- [93] Aggarwal C, Zhai C. A survey of text clustering algorithms. *Mining Text Data*. New York, NY, USA. Springer-Verlag: 2012. p. 77–128.
- [94] Brank J, Grobelnik M, Mladenic D. A survey of ontology evaluation techniques. *Proceedings Conf Data Mining and Data Warehouses (SiKDD)*; 2005. p. 166–9.
- [95] Xu R, Wunsch D. Survey of clustering algorithms. *IEEE Transactions on Neural Networks*. 2005 May; 16(3):645–78.
- [96] Fahad A, Alshatri N, Tari Z, Alamri A. A survey of clustering algorithms for Big Data: Taxonomy and empirical analysis. *IEEE Transactions on Emerging Topics in Computing*. 2014 Sep; 2(3):267–79.
- [97] Berkhin P. Survey of clustering data mining techniques in grouping multidimensional data. Springer. 2006; 25–71.
- [98] Boomija, M. D., & Phil, M. (2008). Comparison of partition based clustering algorithms. *Journal of Computer Applications*, 1(4), 18-21.

- [99] Bezdek JC, Ehrlich R, Full W. FCM: The Fuzzy C-Means Clustering algorithm. *Computers and Geosciences*. 1984; 10(2- 3):191–203.
- [100] Zhang T, Ramakrishna R, Livny M. BIRCH: An efficient data clustering method for very large databases. *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 1996 Jun; 25(2):103–14.
- [101] Guha S, Rastogi R, Shim K. Cure: An efficient clustering algorithm for large data bases. *Proceedings of the ACM SIGMOD international Conference on Management of Data*. 1998 Jun; 27(2):73–84.
- [102] Guha S, Rastogi R, Shim K. Rock: A robust clustering algorithm for categorical attributes. *15th International Conference on Data Engineering*; 1999. p. 512–21.
- [103] Karypis G, Han EH, Kumar V. Chameleon: Hierarchical clustering using dynamic modeling. *IEEE Computer*. 1999 Aug; 32(8): 68–75.
- [104] Mahmood AN, Leckie C, Udaya P. An efficient clustering scheme to exploit hierarchical data in network traffic analysis. *IEEE Transactions on Knowledge. Data Engineering*. 2008 Jun; 20(6):752–67.
- [105] Hubert L, Arabie P. Comparing partitions. *Journal of Classification*. 1985 Dec; 2(1):193–218.
- [106] Ganti V, Gehrke J, Ramakrishna R. CACTUS- Clustering Categorical Data Using Summaries. *Proceeding of the fifth ACM SIGMOD International Conference on Knowledge Discovery and Datamining*; 1999. p. 73–83.
- [107] Cao Q, Bouqata B, Mackenzie PD, Messiar D, Salvo J. A gridbased clustering method for mining frequent trips from largescale, event-based telemetries datasets. *The 2009 IEEE International Conference on Systems, Man and Cybernetics*; San Anonio, TX, USA. 2009 Oct. p. 2996–3001.
- [108] Ester M, Kriegel HP, Sander J, Xu X. A density-based algorithm for discovering clusters in large spatial databases with noise. *Proceedings ACM SIGKDD Conf Knowl Discovery Ad Data Mining (KDD)*; 1996. pp. 226–31.
- [109] Ankerst M, Breunig M, Kriegel HP, Sander J. Optics: Ordering points to identify the clustering structure. *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 1999 Jun; 28(2):49–60.
- [110] Xu X, Ester M, Krieger HP, Sander J. A distribution-based clustering algorithm for mining in large spatial databases. *Proceedings 14th IEEE International Conference on Data Engineering (ICDE)*; Orlando, FL. 1998 Feb 23-27. p. 324–31.
- [111] Varghese BM, Unnikrishanan A, Paulose Jacob K. Spatial clustering algorithms – An overview. *Asian Journal of Computer Science and Information Technology*. 2014; 3(1):1–8.

- [112] Hinneburg A, Keim DA. An efficient approach to clustering in large multimedia databases with noise. Proceedings ACM SIGKDD Conf Knowl Discovery Ad Data Mining (KDD) 1998. p. 58–65.
- [113] Wang W, Yang J, Muntz R. Sting: A statistical information grid approach to spatial data mining. Proceedings 23rd Int Conf Very Large Data Bases (VLDB); 1997. p. 186–95.
- [114] Sheikholeslami, G., Chatterjee, S., & Zhang, A. (1998, August). Wavecluster: A multi-resolution clustering approach for very large spatial databases. In VLDB (Vol. 98, pp. 428-439).
- [115] Milenova BL, Campos M. Clustering large databases with numeric and nominal values using orthogonal projections. O Cluster; 2006. p. 1–11.
- [116] Aggarwal CC, Yu PS. Finding generalized projected clusters in high dimensional spaces. Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data. 2000 Jun; 29(2):70–81.
- [117] A. S. Shirkorshidi, S. Aghabozorgi, T. Y. Wah, and T. Herawan, “Big Data Clustering: A Review,” In Computational Science and Its Applications–ICCSA 2014. Springer International Publishing, p. 707-720. 2014.
- [118] Moulavi, D., Jaskowiak, P. A., Campello, R. J., Zimek, A., & Sander, J. (2014, April). Density-based clustering validation. In Proceedings of the 2014 SIAM international conference on data mining (pp. 839-847). Society for Industrial and Applied Mathematics.
- [119] Zerhari, B., Lahcen, A. A., & Mouline, S. (2015, May). Big data clustering: Algorithms and challenges. In Proc. of Int. Conf. on Big Data, Cloud and Applications (BDCA'15).
- [120] Nelson, F. Kučera, H. (1982). Frequency Analysis of English Usage: Lexicon and Grammar . In: Boston, Houghton Mifflin. x + 561
- [121] M. F. Porter, “Snowball: A language for stemming algorithms,” 2001.
- [122] C. Jensen et al, “Vector-Space model,” Encyclopedia of Database Systems, pp. 3259-3263, 2009.
- [123] J. H. Ward Jr., "Hierarchical grouping to optimize an objective function," Journal of the American Statistical Association, 58, 236–244, 1963.
- [124] Ward, J. H., Jr. (1963), "Hierarchical Grouping to Optimize an Objective Function", Journal of the American Statistical Association, 58, 236–244.
- [125] E. Molloy and T. Warnow, “TreeMerge: A new method for improving the scalability of species tree estimation methods,” Bioinformatics, vol. 35, no. 14, pp. i417-i426, 2019.
- [126] E. Molloy and T. Warnow, “NJMerge: A generic technique for scaling phylogeny estimation methods and its application to species trees,” Comparative Genomics, pp. 260-276, 2018.

- [127] Peter J. Rousseeuw, “Silhouettes: A graphical aid to the interpretation and validation of cluster analysis,” *Computational and Applied Mathematics* 20: 53-65, 1987.
- [128] L. Davies. B. David, W. Donald, “A cluster separation measure,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*. PAMI-1 (2): 224-227, 1979.
- [129] Lin JJ. Mapreduce is good enough? if all you have is a hammer, throw away everything that’s not a nail!*Big Data*. 2012; 1(1):28–37.