

**Towards partial fulfilment for Undergraduate Degree Level Programme  
Bachelor of Technology in Computer Engineering**

***A Project Report on:***

**Legal Document Analysis**

Prepared by:

Admission No.

Student Name

U17CO020

Ishank Jain

U17CO084

Anshul Goyal

U17CO110

Subham Chandrakant Agrawal

U17CO111

Pritesh Kumar Tripathi

Class : B.TECH. IV (Computer Engineering) 8<sup>th</sup> Semester

Year : 2020-2021

Guided by : Dr. Rupa G. Mehta



**DEPARTMENT OF COMPUTER ENGINEERING  
SARDAR VALLABHBHAI NATIONAL INSTITUTE OF TECHNOLOGY, SURAT  
- 395 007 (GUJARAT, INDIA)**

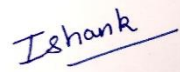



# ***Student Declaration***

This is to certify that the work described in this project report has been actually carried out and implemented by our project team consisting of

<b>Sr.</b>	<b>Admission No.</b>	<b>Student Name</b>
1	U17CO020	Ishank Jain
2	U17CO084	Anshul Goyal
3	U17CO110	Subham Chandrakant Agrawal
4	U17CO111	Pritesh Kumar Tripathi

Neither the source code there in, nor the content of the project report have been copied or downloaded from any other source. We understand that our result grades would be revoked if later it is found to be so.

## **Signature of the Students:**

<b>Sr.</b>	<b>Student Name</b>	<b>Signature of the Student</b>
1	Ishank Jain	
2	Anshul Goyal	
3	Subham Chandrakant Agrawal	
4	Pritesh Kumar Tripathi	

# Certificate

*This is to certify that the project report entitled Legal Document Analysis is prepared and presented by*

Sr.	Admission No.	Student Name
1.	U17CO020	Ishank Jain
2.	U17CO084	Anshul Goyal
3.	U17CO110	Subham Chandrakant Agrawal
4.	U17CO111	Pritesh Kumar Tripathi

*Final Year of Computer Engineering and their work is satisfactory.*

---

---

SIGNATURE:

*Rupa G. Mehta*

GUIDE

JURY

HEAD OF DEPT.

# Abstract

*In the Common Law System, a legal practitioner needs to identify and refer to previous judgements to form beneficial arguments against opponents in the court. However, this practice becomes very challenging due to the presence of a large number of legal judgements. In recent works, Doc2Vec embedding is used to learn the semantic judgement embedding due to its excellent capability to preserve the semantics. However, training Doc2Vec on a large volume of judgments demands enormous computational resources, which the commodity hardware cannot sustain efficiently. To overcome this, the project divides the entire judgement corpus into multiple blocks and then document embedding is trained on each of these blocks distributively using the MapReduce framework. The random distribution of the judgements into blocks is inefficient as each block would have any random set of judgements. Therefore, this project performs overlapping graph clustering on the citation-based network representation of judgements. The citation-based network is a graph constructed from the references information present in the judgements. The resultant clusters, thus formed, consist of referentially similar judgements rather than any set of arbitrary judgements. During clustering it is observed that the number of clusters formed are large, and most of these clusters consisted of fewer numbers of judgments. This project proposes the Centrality Based Graph Cluster Merging (CBGCM) algorithm, which uses eigenvector centrality to merge small clusters into large clusters, reducing the overall number of clusters. Consequently, document embedding is trained on this set of final clusters distributively. Conclusively, a Legal Judgement Search Engine User Interface is implemented to allow a user to statistically search any judgement and also recommend judgements similar to any judgement using the trained Doc2Vec embeddings.*

**Keywords:** *Common Law System - Clustering - Similarity Analysis - Doc2Vec - Legal Judgements - MapReduce - Legal Judgement Search Engine*

# Contents

<b>List of Figures</b>	<b>vi</b>
<b>List of Tables</b>	<b>vii</b>
<b>List of Acronyms</b>	<b>viii</b>
<b>List of Symbols</b>	<b>ix</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Applications . . . . .	1
1.2 Motivation . . . . .	2
1.3 Objectives . . . . .	2
1.4 Contribution . . . . .	3
1.5 Organization of project report . . . . .	3
<b>2 Theoretical Background</b>	<b>5</b>
2.1 Types of Legal Systems . . . . .	5
2.2 Legal Documents . . . . .	6
2.3 Measures of centrality in graphs . . . . .	8
2.4 Doc2Vec Embedding . . . . .	9
2.5 Performance measures for classification problem . . . . .	10
2.6 Overview of Hadoop . . . . .	12
2.6.1 Hadoop Distributed File System (Hadoop Distributed File System (HDFS))	12
2.6.2 MapReduce . . . . .	13
<b>3 Literature Survey</b>	<b>15</b>
3.1 Information Retrieval . . . . .	15
3.2 Similarity Analysis of Legal Judgements . . . . .	17
3.2.1 Text-Based Similarity Analysis . . . . .	17
3.2.2 Link-Based Similarity Analysis . . . . .	18
3.3 Clustering of Legal Judgements . . . . .	18
3.3.1 Text Based Clustering . . . . .	19
3.3.2 Link Based Clustering . . . . .	19
3.3.3 Topics Based Clustering using Latent Dirichlet Allocation . . . . .	19
3.4 Graph-based clustering algorithm . . . . .	20

3.4.1	Clique-based methods . . . . .	20
3.4.2	Label Propagation based methods . . . . .	21
3.4.3	Other Approaches . . . . .	22
3.5	Summary . . . . .	23
<b>4</b>	<b>Proposed Work</b>	<b>24</b>
4.1	Logical Development . . . . .	24
4.2	Proposed Methodology . . . . .	26
4.2.1	Data Extraction and Data Preparation . . . . .	26
4.2.2	Graph Clustering . . . . .	26
4.2.3	Cluster Merging . . . . .	26
4.2.4	Data Distribution . . . . .	28
4.2.5	Document Embedding using Doc2Vec . . . . .	28
4.3	Summary . . . . .	28
<b>5</b>	<b>Simulation and Results</b>	<b>29</b>
5.1	Data Extraction and Pre-Processing . . . . .	29
5.2	Evaluation Metrics . . . . .	30
5.3	Experiment Setup . . . . .	31
5.4	Cluster Formation Result Analysis . . . . .	32
5.4.1	Clustering Results Analysis . . . . .	32
5.4.2	Cluster Merging Result Analysis . . . . .	34
5.4.3	Expert Score based Results Analysis . . . . .	35
5.5	Doc2Vec Model Training Result Analysis . . . . .	36
5.5.1	Hyperparameters for training Doc2Vec . . . . .	36
5.5.2	Training the model on Hadoop multi-node . . . . .	38
5.6	Conclusion . . . . .	39
<b>6</b>	<b>Legal Judgement Search Engine</b>	<b>40</b>
6.1	Data Extraction and Preprocessing . . . . .	40
6.2	Experimental Setup . . . . .	41
6.3	Database Schema . . . . .	41
6.4	Elements of the Search Engine . . . . .	42
6.4.1	Home Page View . . . . .	42
6.4.2	Search View . . . . .	44
6.4.3	Judgement View . . . . .	46
<b>7</b>	<b>Conclusion and Future Work</b>	<b>48</b>
7.1	Conclusion . . . . .	48
7.2	Future Work . . . . .	49
	<b>References</b>	<b>54</b>
	<b>Acknowledgement</b>	<b>55</b>

# List of Figures

2.1	A Typical Case Document [1]	7
2.2	Citations in a Network of Legal Documents [1]	8
2.3	A framework for learning paragraph vector [2]	10
2.4	Confusion Matrix [3]	11
2.5	HDFS Architecture [4]	13
2.6	MapReduce Mechanism [5]	14
4.1	Block diagram describing the proposed methodology	25
5.1	Conductance vs. Threshold keeping iterations constant	33
5.2	Conductance vs Iteration for threshold = 0.2	34
5.3	No. of nodes considered vs Top K Eigen Nodes	35
5.4	After merging conductance vs Top K Eigen Nodes	35
5.5	Accuracy vs. Vector Size	37
5.6	F1 - Score vs. Vector Size	37
5.7	Accuracy vs. Window	37
5.8	F1 - Score vs. Window	37
5.9	Accuracy vs. Epochs	38
5.10	F1 - Score vs. Epochs	38
6.1	Schema	41
6.2	Home Page View	42
6.3	Error Message for No Input Provided	44
6.4	Search View	45
6.5	Judgement View	47

# List of Tables

5.1	Comparison of clustering results for algorithms . . . . .	32
5.2	Comparison of cluster merging techniques . . . . .	34
5.3	Average Recall at K for CaseMine data . . . . .	36
5.4	Training times of Hadoop multi-node clusters . . . . .	38



# List of Acronyms

**BigClam** Cluster Affiliation Model for Big Networks

**CBGCM** Centrality Based Graph Cluster Merging

**CPM** Clique Percolation Method

**EAGLE** Agglomerative Hierarchical Clustering based on maximal Clique

**HDFS** Hadoop Distributed File System

**HTML** Hyper Text Markup Language

**LDA** Latent Dirichlet Allocation

**LPA** Label Propagation Algorithm

**NMF** Non-negative Matrix Factorization

**SLPA** Speaker-Listener Label Propagation Algorithm

**TF-IDF** Term Frequency Inverse Document Frequency

# List of Symbols

$\cdot$  Dot Product of Vectors

$\in$  Belongs to

$\cap$  Intersection of Sets

$\sum$  Summation

$\cup$  Union of Sets

$\overrightarrow{AB}$  Vector

# Chapter 1

## Introduction

With the advancement in technology, there has been an increase in the amount of data available in digital format for different domains. This has led to difficulty in storing, processing and understanding the data. Legal domain is one such domain which has a large volume of documents available electronically. There are generally two types of legal systems - Civil Law System and Common Law System. Civil Law System is legal systems with a rigid set of rules and codes that are used in the decision-making process. In contrast, the Common Law System give the highest priority to previously delivered judgements. Therefore, a legal practitioner needs to form advantageous arguments against an opponent by identifying relevant previous judgements to the case. This form of legal system is prominent in India. A legal judgement is a text document of a closed case which describes the formal resolution given by a court following a lawsuit. Along with textual information, the judgements also contain citation information which are references to other judgements and codified statutes [6].

### 1.1 Applications

A legal practitioner generally begins to prepare his argument for a case at hand by exploring legal documents that are similar to it. These can be used as both: citations and reasoning, while preparing his defence. However, it is time-consuming for a legal practitioner to find relevant documents from the large corpus of documents. Thus, an automated system for finding similar judgements based on an input query is of importance.

## **1.2 Motivation**

In the Common Law System, a legal practitioner needs to identify and refer to previous similar judgements for the case at hand to form reasonable arguments against the opponent. However, this task is manually cumbersome, and an automated system is desired for finding similar documents. In order to compare the similarity between legal judgements, a document embedding technique is required to vectorise the judgement. Then, any suitable similarity measure can be used to find the similarity between the vectorised judgements. Recent works show that the Doc2Vec embedding is excellent for vectorising legal judgements, as it has an excellent ability to preserve the semantic meaning [7]. But, vectorising large volumes of judgements using Doc2Vec can cause scalability issues as it demands heavy computational resources, which regular commodity hardware cannot sustain.

Hence, to overcome the aforementioned problem, this project splits the entire judgement corpus into multiple blocks so that each of them can be trained parallelly in a distributed environment. Another major issue is that randomly distributing the judgments into blocks is inefficient. To overcome this issue, graph clustering can be performed on the network representation of judgements formed using the citation information present in them. Later, judgements present in each cluster can be vectorised using Doc2Vec distributively. The trained judgement vectors can then be used to find similarity between judgements and recommending similar judgements for any query judgement. These were the major factors that motivated the project.

## **1.3 Objectives**

The objective of this project is two folds. Initially, this project explores different overlapping graph-based clustering algorithms to group the referentially similar judgements. Later, Doc2Vec can be trained distributively on each of the clusters to learn the semantically rich judgment embedding. Finally, the trained judgement vectors can be utilized for finding similarity between judgements and recommending similar judgements for a given query judgement in the automated system.

## 1.4 Contribution

This project aims to recommend relevant judgements from a large corpus based upon the user query judgment. The project uses Doc2Vec embedding technique for vectorising the legal judgements and then cosine similarity measure is used to find similarity between two vectorised judgements. However, vectorising a large volume of judgments using Doc2Vec poses scalability issues. This project performs the following in order to achieve these objectives:

1. **Judgement Cluster Preparation:** To overcome the scalability issues, overlapping graph clustering is performed on the citation-based network representation of judgements. However, another issue is that clustering generates numerous clusters and most of these consist of a fewer number of judgments. Training document embedding on these small clusters is inefficient. To overcome this issue, this project proposes the Centrality Based Graph Cluster Merging (CBGCM) algorithm for merging such small clusters into large ones, thus reducing the overall number of clusters. These final clusters act as independent blocks of judgement.
2. **Training Doc2Vec embedding on each cluster:** Document Embedding is trained on each block of judgements distributively using the MapReduce framework.

Finally, this project also implements a Legal Judgement Search Engine which performs a statistical search that allows a user to search a judgement based on multiple parameters and keywords. Along with the statistical search, it also recommends similar judgements for any of the judgements selected by the user using the trained Doc2Vec embeddings.

## 1.5 Organization of project report

This chapter covers the introduction to the project along with its application, motivation, objective and overview. Chapter 2 presents a theoretical background of the terminologies in the legal domain as well as other important concepts needed to understand the project better. Chapter 3 is the Literature Survey which summarises the work done in the legal domain in finding similarity between legal judgements, clustering legal judgements and information retrieval systems. Later in Chapter 3, a review of various overlapping graph clustering algorithms is discussed. Our proposed methodology and logic development of the same is covered in Chapter 4. Chapter 5 discusses the

experimental simulation and the results obtained in the entire project. Chapter 6 gives a detailed information about the Legal Judgement Search Engine implemented in this project. Chapter 7 ends the report with conclusion and future work proposed.

# Chapter 2

## Theoretical Background

This chapter discusses the different terminologies related to legal systems and Indian Legal judgments. It discusses various graph-based centrality measures and gives a brief overview of Doc2Vec embedding. It also gives a brief overview regarding the performance measures used in classification problems. Finally, it provides an overview of Hadoop and its components: Hadoop Distributed File System (HDFS) and MapReduce.

### 2.1 Types of Legal Systems

There are mainly two types of Legal Systems: Civil Law System and Common Law System. Along with these, there are Customary Law and Religious Law as well as some mixed legal systems which consist of different combinations of the aforementioned systems.

- **Civil Law System:** It is a legal system with a rigid set of rules and codes that are exclusively used in all of their decision-making process in the legal domain. They do not include any amendments or upgrades based on new judgments made by the court. This form of legal system is common in European countries like France and Germany.
- **Common Law System:** It is a legal system which also includes some predefined rules, but the previous decisions of the court play a vital role in decision making. This form of legal system is common in USA and India.

## 2.2 Legal Documents

After a particular case is closed, all of its details including the rights and liabilities of all parties involved in the legal proceeding and the formal decision made by the court following the lawsuit are constituted in the legal judgment. They are used for referral in the future if any practitioner comes across a similar situation and requires guidance from the older precedent. An example of legal judgment is shown in Figure 2.1 below. Following are the terminologies related to a legal judgement:

- **Name of Judgement:** The heading of the judgment consists of the name of both the parties (the petitioner and the respondent) involved in the judgment, the court and the final date of oral argument.
- **Petitioner:** The person or organization that presents the petition in the court is known as the petitioner.
- **Respondent:** The person or organization against whom the petition has been filed in the court is known as the respondent.
- **Judge's Names:** This is the list of judges that were present on the bench and were involved in the proceeding.
- **UniqueID:** It is the unique identification number given to each judgment by a certain court reporter so that the judgment could be referenced in the future using its corresponding ID. A court reporter is a person who records court proceedings during their run and then creates a summary from it. Two of the most common court reporters in India are AIR (All India Reporter) and SCR (Supreme Court Reporter). The format of a UniqueID varies among different reporters. As can be seen in Figure 2.1, the UniqueID of this judgment is 1960 AIR 571, 1960 SCR(2) 841.
- **Citation:** Whenever a practitioner refers to an older judgement for their current proceeding, they have to mention those judgements in their current record. This is known as a citation.
- **In-Citation:** These are the judgements which are citing to a given judgement.



**KHANDESH SPG. & WVG. MILLS CO. LTD. V. THE RASHTRIYA GIRNI  
KAMGAR SANGH, JALGAON [1960] INSC 1 (2 January 1960)**

02/01/1960 SUBBARAO, K.

SUBBARAO, K.

GAJENDRAGADKAR, P.B.

GUPTA, K.C. DAS

**Judges' name**

**Name of Judgment**

**CITATION:** 1960 AIR 571 1960 SCR (2) 841

**Unique IDs of judgment**

**CITATOR INFO :**

**Unique IDs of judgments which are referring current judgment**

E 1960 SC1006 (5) RF 1967 SC 122 (22) RF 1968 SC 963 (34) R 1969 SC 612 (8) R 1972 SC 330 (10,11) RF 1972 SC1954 (15)

**ACT:**

**Acts under whom current dispute falls**

Industrial Dispute-Bonus-Full Bench Formula--Rechaibilita- tion -Reserves used as working capital-Mode of Proof.

**HEADNOTE:**

**A short summary of judgment**

In ascertaining the surplus available for the payment of bonus according to the Full Bench formula the Industrial Court allowed the statutory depreciation but did not give any credit for the rehabilitation amount claimed. The Industrial Court estimated the amount required for rehabilitation at Rs. 60 lakhs; out of this amount it deducted Rs. 51 lakhs representing the reserves and the balance of Rs. 9 lakhs spread over a period Of 15 years gave the figure of Rs. 60,000 as the amount that should be set apart for the year in question for rehabilitation. This amount being less than the statutory depreciation the Industrial Court held that the appellant was not entitled to any deduction on account of rehabilitation as a prior charge. The appellant contended that the balance-sheet disclosed that the entire reserves had been used as working capital and consequently the said reserves should not be excluded from the amount claimed towards rehabilitation.

Held, that the appellant had failed to prove that the reserves had in fact been used as working capital and as such the amount was rightly deducted by the Industrial Court from the amount fixed for rehabilitation.

**Case citation**

The Associated Cement Companies Ltd. v. Its Workmen. [1959] S.C.R. 925, referred to.

In view of the importance of the item of rehabilitation in the calculation of the available surplus it was necessary for tribunals to weigh with great care the evidence of both parties to ascertain every sub-item that went into or was subtracted from the item of rehabilitation. If parties agreed, agreed figures could be accepted. If they agreed to a decision on affidavits, that course could be adopted. But in the absence of agreement the procedure prescribed by 0.

XIX, Code of Civil Procedure had to be followed. The accounts, the balance-sheet and profit and loss accounts were prepared by the management and the labour -had no hand in it. When so much depended on this item it was necessary that the Industrial Court insisted upon a clear proof of the item of rehabilitation and also gave a real and adequate opportunity to labour to canvass the correctness of the particulars furnished by the employers.

Indian Hume Pipe Company, Ltd. v. Their Workmen. [1960] 2 S.C.R. 32, Tata Oil Mills Company Ltd. v. Its Workmen [1960] 1 S.C.R 1. and Anil Starch Products Ltd. v. Ahmedabad Chemical Workers' Union. C.A. No. 684 Of 1957 (not reported), referred to, 842

CIVIL APPELATE JURISDICTION: Civil Appeal No.257 of 1958.

**Case citation**

Figure 2.1: A Typical Case Document [1]

- **Out-Citation:** These are the judgements which are cited by a given judgement.
- **Acts:** It defines a law which has been enacted by the legislation, and explains the legal specifications of the matter at hand.
- **Headnote:** It contains a summary of the events and decisions that took place during the proceeding and the final judgment. These headnotes help a practitioner while referring to a particular judgment so that they can go through it rather quickly without going through all of its details.

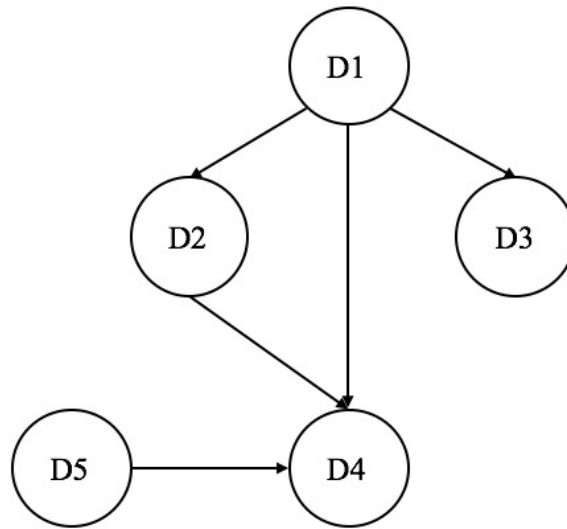


Figure 2.2: Citations in a Network of Legal Documents [1]

In Figure 2.2, D1, D2, D3, D4 and D5 are legal documents, and a directed edge between nodes denotes their citations. The out-citations of D1 are D2, D3, and D4, while the in-citations of D4 are D1, D2, and D5. As shown here, two documents can simultaneously linked directly (D1-D4) and indirectly (D1-D2-D4).

### 2.3 Measures of centrality in graphs

Centrality in a graph indicates the relative importance of nodes in the graph compared to the rest of the nodes. Every node can be significant in some way, depending on how the “importance”

is defined. There are various centrality measures, however, the most widely used techniques are: Degree centrality, Eigenvector centrality, and Betweenness centrality [8].

**Degree centrality** is one of the most straightforward centrality measures. The degree centrality of a node indicates the fraction of nodes it is connected to [8]. Consider a graph  $G$ , let  $v$  be a vertex in  $G$  and  $n$  be the total number of nodes in  $G$ , then

$$Centrality_D(v) = \frac{degree(v)}{n - 1} \quad (2.1)$$

**Eigenvector Centrality** is a form of extended degree centrality that measures a node's centrality based on the importance of its neighbours. It takes the centrality of neighbouring nodes also into consideration. The eigenvector centrality [8] of an  $i^{th}$  node in the graph is the  $i^{th}$  component in the eigenvector corresponding to the greatest eigenvalue of the equation

$$Ax = \lambda x \quad (2.2)$$

where  $A$  is the adjacency matrix of the graph,  $\lambda$  is the largest eigenvalue, and  $x$  is the eigenvector corresponding to this eigenvalue.

**Betweenness centrality** considers the global structure of the graph. It indicates the number of times a node lies on the shortest path between other nodes [9].

$$Centrality_B(v) = \sum_{s,t \in V} \frac{\sigma(s, t|v)}{\sigma(s, t)} \quad (2.3)$$

where  $V$  is set of nodes,  $\sigma(s, t)$  is number of shortest  $(s, t)$  paths, and  $\sigma(s, t|v)$  is number of those paths passing through  $v$  other than  $s, t$ .

## 2.4 Doc2Vec Embedding

Doc2Vec [10] is a document embedding technique that can be used for feature representation of a text corpus. It's an extension of Word2Vec, which is used for fixed-length vector representation of the word, preserving their semantic information. Doc2Vec is used to generate a vector for an entire document, while Word2Vec generates vectors for each word.

Le and Mikolov have built on the traditional Continuous Bag of Words (CBOW) representation of

the Word2Vec model in order to build their model. In CBOW a sliding window is created around the center word to predict the surrounding word. Figure 2.3 shows that, by adding an additional vector of Paragraph ID to the CBOW Word2Vec model, they have reformed the traditional model of learning word vectors into representing information unique to the document. Paragraph ID is a unique vector to which every paragraph of the document is mapped, which are then used as features for the paragraph as a whole. As these vectors are learned using unsupervised techniques, they can generalise well to tasks having scarcity of labelled data. They also inherent the semantics and order of words themselves, mapping related words closer in the vector space.

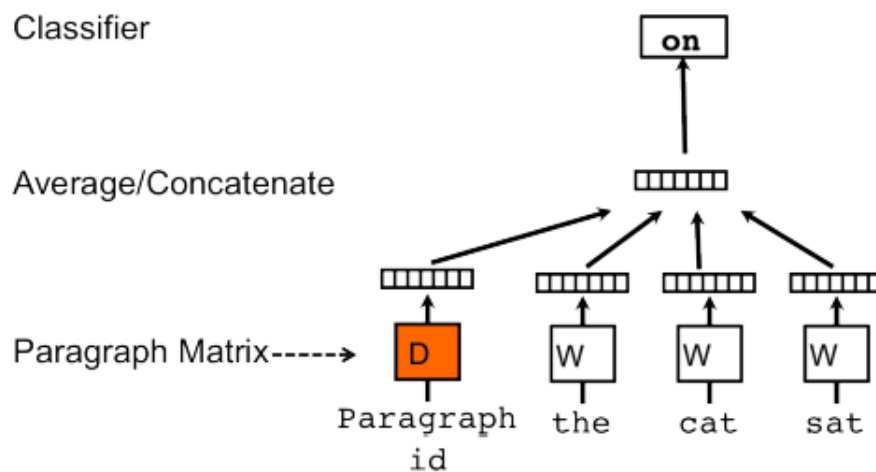


Figure 2.3: A framework for learning paragraph vector [2]

## 2.5 Performance measures for classification problem

Confusion Matrix [3] is an evaluation matrix used to evaluate a prediction model's performance by comparing the actual values to the predicted values. It has 4 parameters that are used to calculate different measures: True Positive (TP), True Negative (TN), False Positive (FP), False Negative (FN).

- **Accuracy:** Accuracy is the ratio of correct predictions to the total number of predictions.

The accuracy evaluation matrix is observed to be a good evaluation metric when a balanced

		Actual Values	
		Positive (1)	Negative (0)
Predicted Values	Positive (1)	TP	FP
	Negative (0)	FN	TN

Figure 2.4: Confusion Matrix [3]

data set was considered.

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \quad (2.4)$$

- **Precision:** Precision is the ratio of correct positive predictions to the total positive predictions. Higher precision indicates that false positives are fewer.

$$Precision = \frac{TP}{TP + FP} \quad (2.5)$$

- **Recall:** Recall is the ratio of correct positive predictions to the total actual true values. Higher recall indicates that false negatives are fewer.

$$Recall = \frac{TP}{TP + FN} \quad (2.6)$$

- **F1 score:** F1 score is the mean of precision and recall, which aims to generalise precision and recall metrics. This evaluation measure has been taken into consideration since the

testing dataset is imbalanced.

$$F1 - Score = \frac{2 * Precision * Recall}{Precision + Recall} \quad (2.7)$$

## 2.6 Overview of Hadoop

Apache Hadoop is an open source software toolbox to effectively solve problems with vast volumes of data and computations using a network of multiple systems [4]. There are majorly two parts of Apache Hadoop: The storage component of the distributed file system (HDFS) and the MapReduce programming model which is the processing part.

### 2.6.1 Hadoop Distributed File System (HDFS)

Hadoop Distributed File System (HDFS) [11] is a part of the Hadoop framework. HDFS is a storage system for storing many datasets designed to run clusters on commodity hardware. It uses the concept of rack awareness, where each rack contains a set of machines that are connected with a common switch. A data block is stored on different racks simultaneously so that multiple copies of the data is maintained in the system. Due to this replication feature, HDFS provides a fault-tolerant storage system with high data availability during hardware failure. Also, with vertical and horizontal scalability and cost-effectiveness features, HDFS is the most reliable storage system.

HDFS follows the concept of master-slave architecture. The main machine that distributes the tasks into smaller blocks is known as the master (NameNode), and the machine that receives an individual block from the master are known as slaves (DataNode).

**NameNode (Master):** The NameNode is the centrepiece of the HDFS architecture. It regulates the namespace present in the filesystem and manages the task distribution among slave nodes as it has file access to clients. NameNode is responsible for the mapping of blocks to DataNode. Further, it stores the metadata of every block present in the filesystem, and ensures that the Data Node is alive by receiving regular reports. In case of failure of a slave, NameNode chooses the slave replica to continue the process.

**DataNode (Slave):** DataNodes majorly have the work of storing data by reading/writing the requests by the clients. DataNodes are responsible for creating, replicating, or deleting blocks on the request from the NameNode. Each slave node operates on its respective block of data to achieve

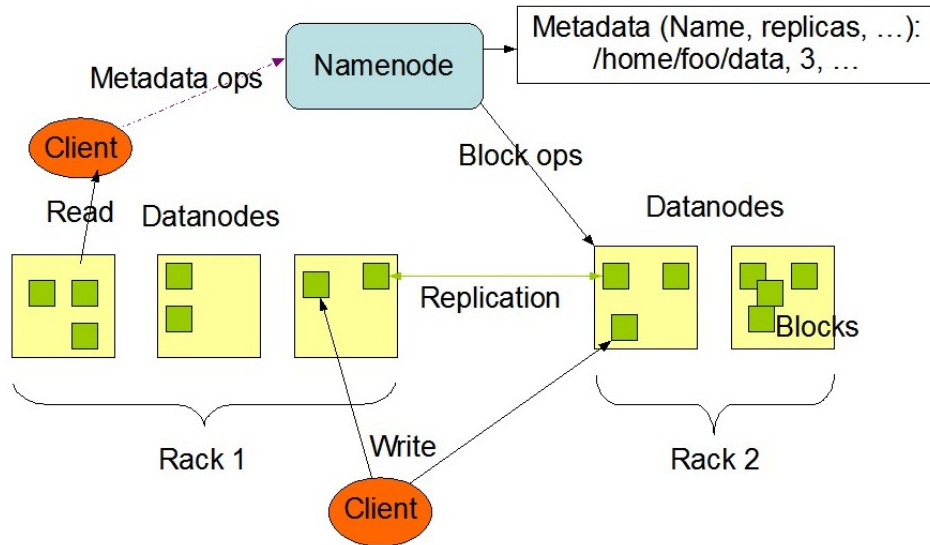


Figure 2.5: HDFS Architecture [4]

parallel processing. DataNodes are also required to report a master regarding the health of HDFS by sending a heartbeat and report to the NameNode.

### 2.6.2 MapReduce

MapReduce [12] is a technique that divides a large set of data into smaller blocks so that each of them can be processed parallelly. These blocks are then passed through a set of machines where each machine operates on a single block of data. This type of mechanism helps in increasing the efficiency of the system by reducing the overall computational time. Figure [2.6] shows the working of the MapReduce function.

The process flow of the MapReduce [12] is as follows:

1. Initially, the input data is present in key/value pairs, where it is split into different segments, and each data segment is distributed to different DataNodes.
2. The MapReduce code is then passed to each of these DataNodes whose operations are controlled by NameNode.
3. Each mapper DataNode passes the data segment through the *map()* function, which converts them into intermediate key/value pairs.

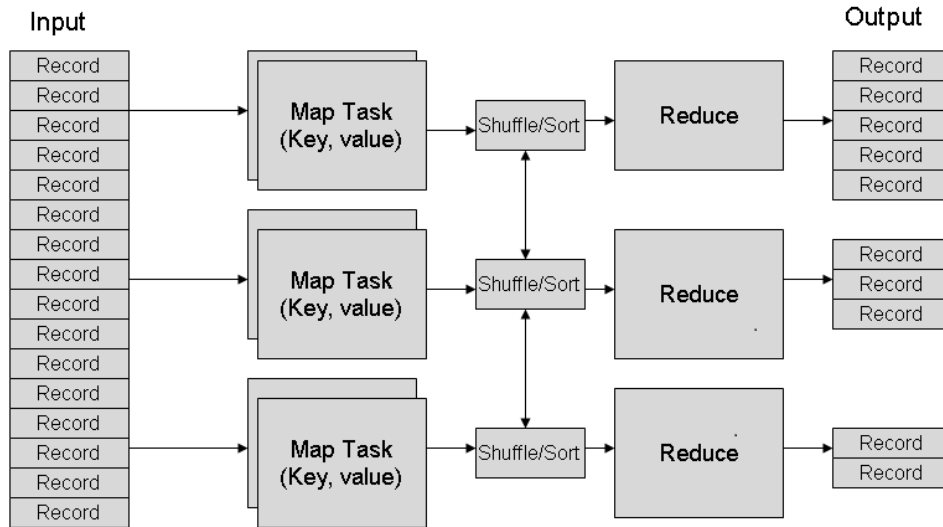


Figure 2.6: MapReduce Mechanism [5]

4. These intermediate key/value pairs are then transferred to a reducer DataNode that sorts all the key/value pairs so that all the data having the same key are present together.
5. The reducer DataNode passes these key/value pairs to the *reduce()* function, which combines all the data with the same key.
6. All these reduced key/value pairs are combined to make the final output file.



# Chapter 3

## Literature Survey

This chapter briefly discusses the existing literature in the field of legal domain in three significant areas: Information Retrieval, Similarity Analysis of Legal Judgements and Clustering of legal Judgements. Later, various overlapping graph clustering algorithms are reviewed.

### 3.1 Information Retrieval

A major concern of Information Retrieval is understanding the user query and then presenting the results based on the input query. Information can be retrieved in two ways, by using text content of the document or the citation information of present in the document.

Text-based models are of three types, set-theoretic models, algebraic models and probabilistic models. Set-theoretic models are Boolean models. In this, the input query and all the judgements are treated as a set of terms and the relevance between the query and the document is based on set-theoretic operations. Algebraic models are models in which document and user query are represented in vector forms. The Vector Space model is the most common algebraic model. In this, each element in the vector represents a term, and each term is associated with some different weights. There are many weighting schemes available, and TF-IDF [13] is one of them. Probabilistic models retrieve the documents by ranking them based on the relevance of the documents with the user query. Link-based models use only the link information in the documents to retrieve documents.

Analysing the extraction of relevant legal judgements using citation-based information, Raghav [6]

proposed ranking of the legal judgements in the corpus for a given input legal judgement. Initially, the author ranked the judgements based on the text-based approach and later based on citation information present in them. The author also proposed a paragraph-based ranking approach which used a modified OkapiBM25 model to extract paragraph-level scores between two judgements by considering the paragraphs present in the headnote of the judgements. It is concluded that the combination of paragraph-based and a citation-based ranking method performed better than text-based ranking method.

Maxwell et al. [14] discussed the concepts of Knowledge Engineering (KE), as well as Natural Language Processing (NLP), based Information Retrieval. Merits and limitations of each of the methods were evaluated. Although KE-based informational retrieval considers a lawyer's approach of problem-solving for information retrieval so that it becomes feasible for practical application, the manual intervention required in this model makes it impractical to be implemented on a large scale. Similarly, NLP-based information retrieval considers different paragraphs and summaries from the judgements to create a model where it automatically retrieves relevant information based on current judgement. As is mentioned by Opijnen et al. [2], only considering the NLP models for information retrieval would create a much less practical model since the practitioners will not be able to use their domain knowledge and experience in the information retrieval process. A hybrid NLP is suggested by Maxwell et al. [14], which would consider the concepts of NLP and also include the manual intervention in the algorithm like KE-based Information retrieval.

Legal query understanding is a complex problem which requires multiple natural language processing tasks. It requires decomposing the legal query into its individual components and then understanding the underlying difference in them. Efforts have been made to understand legal query intent classification and Named Entity Recognition using Deep Neural Networks. A Shankar et al. [15], constructed various DNN models using RNN, CNN, LSTM, a combination of the before and finally ensemble model. Experiments with other machine learning models were also performed. To train the neural network models, word vectors were created to be used as word embeddings. It is concluded that the ensemble model outperformed all other neural network models.

## 3.2 Similarity Analysis of Legal Judgements

In the Common Law system, a lawyer, after getting a case in his hand, prepares for his arguments by manually finding a similar judgement in the database. Suppose, he finds a judgement 'J' related to his case; he then focuses on finding judgements similar to judgement 'J'. Now this work is really cumbersome. This section explores the work done in finding similarity between legal judgements.

### 3.2.1 Text-Based Similarity Analysis

Mandal et al. [7] had implemented the concepts of neural network based similarity analysis like word embedding(Word2Vec) and document embedding(Doc2Vec) and compared them with the traditional methods of text-based (using TF-IDF and computing cosine similarity [13] and LDA) similarity analysis. It is compared on the basis of what portion of judgement is considered for finding similarity. Before Doc2Vec and Word2Vec embedding techniques, models used to consider the frequency and priority of certain keywords and use that as the basis while comparing documents. However, it is proven that Doc2Vec and Word2Vec embedding methods perform far better and also consider the entire document in the process whereas the text-based methods were only considering paragraphs/headnotes during their execution. TF-IDF stands for Term frequency – Inverse document frequency. It's a numerical statistics to indicate the relevance of any keyword in a corpus of documents [13].

In Cosine Similarity [1], initially, the documents are converted into a set of terms by pre-processing which involves removal of stop-words and stemming. Then, the union of all the terms present in all documents is performed to get a set of terms which represent the entire collection. Numeric weights are assigned to all the terms in the document, which represents the relevance of the term concerning the corpus of documents. The weights given to a term may be elucidated as its position coordinate in the document space. A document vector can represent each document with the terms in it.

To find cosine similarity of two documents represented by Document Vector  $\vec{d_1}$  and  $\vec{d_2}$  the measure is defined as

$$\text{Cosine} - \text{Similarity}(\vec{d_1}, \vec{d_2}) = \text{cosine}(\vec{d_1}, \vec{d_2}) = \frac{\vec{d_1} \cdot \vec{d_2}}{|\vec{d_1}| |\vec{d_2}|} \quad (3.1)$$

### 3.2.2 Link-Based Similarity Analysis

Kumar et al. [1] observed that the Link-based approach is more effective than text-based similarity measures for finding similar legal judgements. However, link-based similarity approaches can fetch only a small set of judgements because of an insufficient number of case-citations and due to the probable presence of a high time gap between two judgements. Thus, Kumar et al. proposed the use of paragraph-link. A paragraph link is said to exist between two paragraphs if there are a threshold number of paragraphs which are similar by text-based similarity approach (using TF-IDF and computing Cosine Similarity). Bibliographic coupling is then applied between two documents by comparing paragraph links instead of out-citations. That is, after calculating the number of common paragraph links between two documents, if it is above a certain threshold, the documents are considered to be similar.

Bibliographic Coupling [1] is a method in which, instead of the contents, two documents are compared based upon common references between them. It is a link based similarity measure and widely used in comparing research papers. According to this, two documents are considered similar if they cite more than a threshold number of common documents.

$$\text{Bibliographic} - \text{coupling}(D1, D2) = OC_{D1} \cap OC_{D2} \quad (3.2)$$

Here  $OC_{D1}$  represents the out-citations of document D1 and  $OC_{D2}$  represents the out-citations of document D2.

### 3.3 Clustering of Legal Judgements

Clustering is a method of aggregating the data points into clusters such that the data points within the cluster are more similar when compared to data points in other clusters. Broadly, clustering is of two types, hierarchical clustering and partitional clustering. Hierarchical clustering is of two types, agglomerative clustering and divisive clustering. Agglomerative Hierarchical clustering is a bottom-up approach where, initially, all the data points are considered as a single cluster. Then clusters are combined to form larger clusters. Whereas Divisive clustering is a top-down approach where initially, all data points are considered into the same cluster. Then the larger clusters are broken into smaller ones. The ideal number of clusters required depends on the nature

of the dataset or the requirements of the user. In partitional clustering algorithms, all clusters are recognised simultaneously.

### **3.3.1 Text Based Clustering**

Apart from legal judgements, efforts have been made in clustering general text documents. P. Bafna et al. [16], have clustered news articles, emails, research papers and Reuters using Hierarchical Agglomerative Clustering and Fuzzy C Means clustering. Initially, they converted the corpus of documents into a document matrix using TF-IDF and then performed clustering of the documents using Hierarchical Agglomerative Clustering and Fuzzy C Means clustering. It was found that the Hierarchical Agglomerative Clustering algorithm performs better than Fuzzy C Means for news articles and Reuters documents, whereas Fuzzy C Means performed better for research papers clustering. For email documents, both the methods have similar performance.

### **3.3.2 Link Based Clustering**

Raghav [6] performed the clustering of legal judgements delivered by the Supreme Court of India in between 1970 and 1973, using different approaches. Initially, the judgements were clustered using text-based clustering methods. TF-IDF is used to convert the judgements into word vectors. Further link-based clustering is performed on the judgements by using the citation information present in the judgements and also establishing paragraph-links between the judgements. Considering only citation information between judgements can fetch only a small set of judgements because of an insufficient number of case-citations and due to the probable presence of a high time gap between two judgements. Thus, Raghav proposed that using PL links between the judgements increases the number of judgements participating in the clustering process. It was observed that the link-based clustering using both paragraph-link and citation information outperformed text-based clustering approach.

### **3.3.3 Topics Based Clustering using Latent Dirichlet Allocation**

To improve the information retrieval system in the legal domain for finding similar judgements, Kumar V. and Raghuveer K. [17], proposed to cluster the documents into disjoint subsets, with each subset consisting of documents most relevant to a topic using Latent Dirichlet Allocation (LDA). The judgement part of the legal document is considered for document clustering. First,

the data is pre-processed by removing stop words and legal terms which give no information about the case. Then, the legal judgements are converted to topics by LDA, which is a probabilistic generative model to model a collection of documents by topics [18]. It uses Dirichlet distribution to represent the topics. The number of topics generated from LDA is assumed to be equal to the number of topics the corpus described. The documents are then clustered by finding the cosine similarity between each document with the topics, such that documents within a cluster are very close to the topic. It is found that legal documents can be clustered by finding the cosine similarity between legal documents and topics.

### **3.4 Graph-based clustering algorithm**

The citation information between the judgements can be used to construct a graphical network of legal judgements, and clustering algorithms can be applied on these to find clusters. Clustering of graphs is generally of two types: overlapping clustering algorithms and non-overlapping clustering algorithms [19]. This project reviews overlapping clustering algorithms which can be classified into these categories.

- Clique-based method
- Label Propagation based method
- Other Approaches

#### **3.4.1 Clique-based methods**

A clique [20] is a subgraph of a graph which is complete. So a clique in a graph can be found and expanded to obtain a cluster. CFinder [21] is an algorithm which is used to detect dense overlapping clusters in graphs. It uses the Clique Percolation Method(CPM) to find a k-size clique. A k-clique is a connected subgraph of size k. The algorithm works by finding all k-cliques in the graph. Once all the cliques are found, a clique graph is constructed where the nodes in the graph represent a clique. An edge is present between two clique nodes if the cliques share k-1 common nodes in the graph. Finally, each connected components in the clique graph are considered as a cluster.

EAGLE (agglomerative Hierarchical clusterinG based on maximal cliqueE) [22] is a hierarchical agglomerative clustering algorithm to find overlapping communities. The first step in the algorithm

is to determine all maximal cliques. A maximal clique is a clique which cannot be extended by adding even a single node. These maximal cliques are then considered as initial communities. A dendrogram of these communities is considered, and the communities with the highest similarity are merged, forming new communities. This step is carried till there is no merging possible.

The major drawback of CFinder and EAGLE algorithm is a large number of unclustered nodes. PercoMCV [23] is a clique based overlapping clustering algorithm which solves this problem. The algorithm runs in two steps. In the first step, it uses the Clique Percolation algorithm(CPM) to find a clique of dimension  $k$ . The two cliques found are considered adjacent if they have  $k-1$  nodes common between them, and they are considered in the same cluster. The algorithm uses 2 stacks to combine all such adjacent cliques, and a set of final clusters is obtained. In the second step, the nodes which are not clustered is considered, and eigenvector centrality is used to cluster them. The eigenvector centrality works on the principle that any node which is connected to the central node of a cluster, belongs to that cluster.

### **3.4.2 Label Propagation based methods**

Label Propagation algorithm (LPA) is an iterative process where labels are assigned to unlabelled points, and a set of nodes are grouped into a community by the propagation of the same property, action or information in the network in the form of labels.

Boleslaw K et al.[24] proposed a Speaker-listener label propagation algorithm (SLPA) that mimics human communication behaviour. In SLPA, each node has a memory that stores knowledge of frequently recognised labels instead of erasing them. Also, the more a label is observed by a node, the more probable it is to spread the label to other nodes, thus imitating people's preference of spreading most frequently discussed ideas. The algorithm consists of the following steps.

1. Initially, the respective Node ID (ie., Unique label) is used to initialize its memory.
2. Then, the following steps are performed until the stop criterion is fulfilled:
  - (a) One node is selected as a listener.
  - (b) Each neighbour of the chosen node sends out a single label consistent with a particular speaking rule, like picking a random label from its memory with likelihood proportional to the occurrence frequency of its label in the memory.

(c) The listener accepts one label from the collection of labels received from its neighbours consistent with certain listening rules, like selecting the foremost popular label from what is observed in the current step.

3. Finally, the post-processing depending upon the labels in the memories of nodes is utilised to produce the communities. In post-processing, a threshold value of probability is defined. If the probability of appearance of a label is less than this, the label is discarded from the node's memory. After this, connected nodes having a specific label are grouped and formed into a cluster. If a node has multiple labels, then it belongs to multiple clusters and is grouped accordingly.

Giulio Rossetti et al. [25] proposed Demon which is an approach where the adjacent nodes were determining the cluster in which their neighbours would belong. Ego network which is defined as the subgraph containing all the neighbouring nodes and edges. This approach gave robust results initially but had flaws in EgoNet structure which were increasing noise in every iteration. To overcome that, Giulio Rossetti [26] proposed a bottom-up solution for the community discovery problem. This algorithm is known as Angel. The primary objective of Angel is to reduce the computational complexity of existing node-centric approaches by using the merging method and hence to create the final community partition.

Firstly, the algorithm loops over each node to create all possible view points of the network. Further EgoMinusEgo [27] is used to obtain the network structure. Also, Angel removes the root node, because it is directly connected to all nodes in its ego-network, links that would contribute to noise in local communities identification. Once the ego-minus-ego network is obtained, Angel uses the label propagation method to detect the local communities. This step is important because Angel is iterated over each node of the graph. Finally, local communities would still not be having the real community structure. To get more accurate communities, Angel uses PrecisionMerge function to reduce community size to avoid the existence of full-fledged communities. Hence, the merging strategy is applied until communities cannot be further merged.

### **3.4.3 Other Approaches**

J Leskovec et al. [28] proposed a cluster affiliation model named BigClam, which can detect communities in large scale networks with dense overlaps. Since most of the earlier models assumed



communities to be sparsely connected, they could not predict the communities in a dense network accurately. Hence, to overcome this issue, this particular model is proposed.

The process begins with generating a Cluster Affiliation model to describe the organization of networks depending upon community affiliations. Next,  $K$  communities are detected by fitting the BigClam to the original graph  $G$  by using non-negative matrix factorization NMF, which increases the scalability of the model. This result is then optimized by using a gradient ascent algorithm where the weights of one node is modified by updating the weights of all the neighbouring nodes. Also, this model can detect communities in very large-sized networks because of the implementation of non-negative matrix factorization.

### **3.5 Summary**

A legal document consists of textual information as well as referential information. This two information has been exploited in this literature to find similarity between legal judgements. In recent works, Doc2Vec embedding has been utilized to learn the semantic judgement embedding due to its excellent capability to preserve the semantics. Thus, this project uses Doc2Vec as the document embedding technique. This project uses overlapping graph clustering to solve the problem of enormous computational requirements for a large corpus of judgements by dividing the entire corpus into multiple clusters. This chapter presents a review of various overlapping graph clustering algorithms and based upon literature [21, 29, 30], PercoMCV, Angel, SLPA and BigClam algorithms are selected.

# Chapter 4

## Proposed Work

The project aims to initially split the entire judgement corpus into multiple blocks in order to overcome the scalability issues posed by large volume of judgements. For splitting the judgements into blocks, overlapping graph clustering is performed so that each block consists of referentially similar judgements. Then, Doc2Vec embedding can be trained on each of the blocks distributively using the MapReduce Framework.

### 4.1 Logical Development

Manually finding similar legal judgements from a large corpus of documents is cumbersome. Thus, an effective technique is required to efficiently find similar judgments, saving time for lawyers and other legal practitioners. In recent works, Doc2Vec embedding is used for vectorising the legal judgements. However, processing a large volume of judgments is computationally expensive, which the regular commodity machine can not handle. To handle such requirements, the document embedding technique is trained in a distributed environment using MapReduce. Instead of randomly distributing the data into smaller sets, the citation information present in the judgements is exploited to construct a graphical network. Further, the graph is divided into clusters by performing overlapping graph clustering, and Doc2Vec is trained on each cluster using MapReduce. Finally, using the trained document embedding of the legal corpus, judgements similar to the query judgement are recommended from clusters.

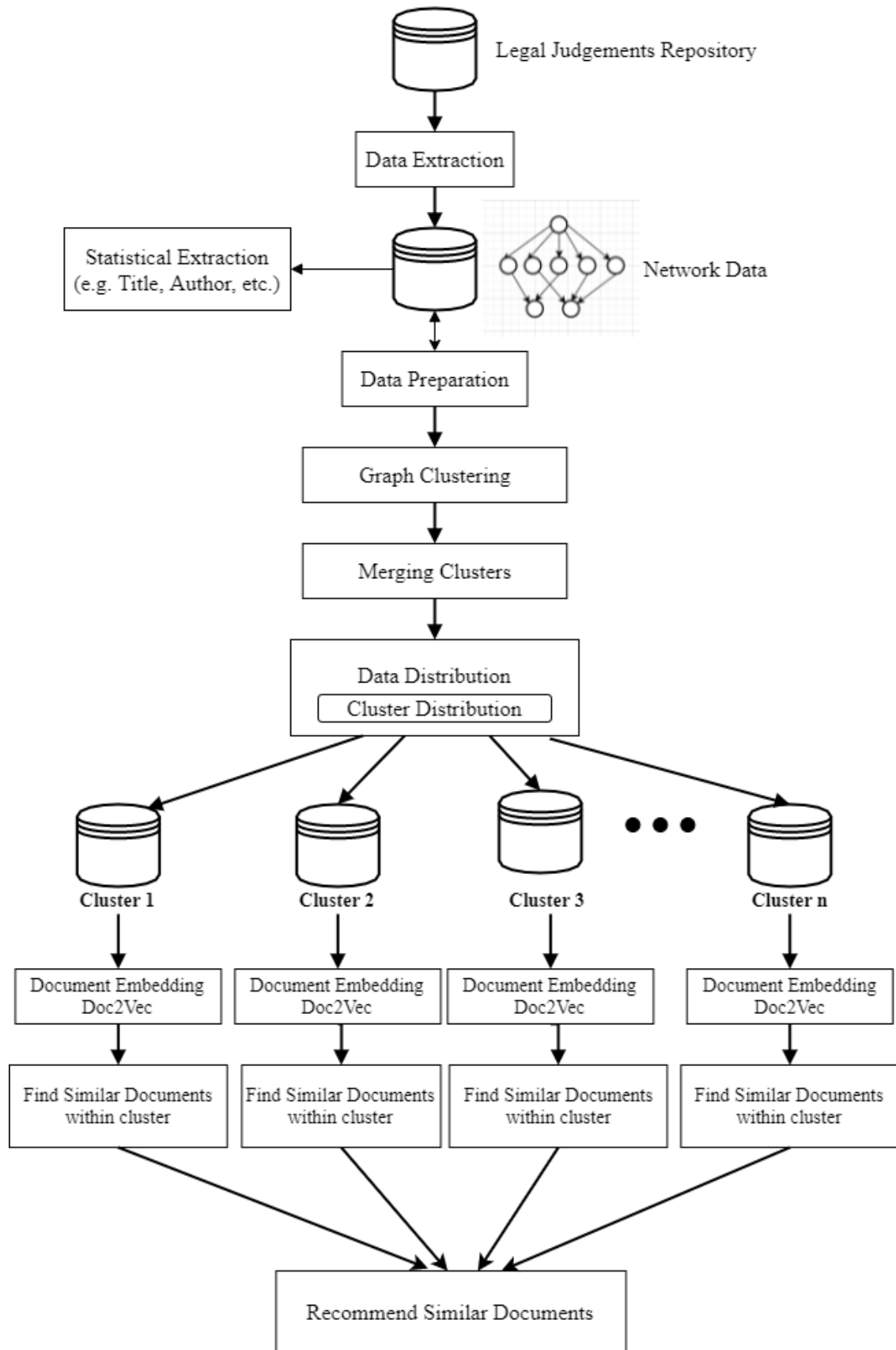


Figure 4.1: Block diagram describing the proposed methodology

## **4.2 Proposed Methodology**

In order to achieve the objectives mentioned above, a systematic approach is proposed, as depicted in the block diagram Figure 4.1. The proposed work consists of five major phases. The first phase is Data Extraction and Data Preparation. The next phase is Graph Clustering where various overlapping graph clustering algorithms are experimented. Graph clustering results into formation of large number of clusters. So, the third phase discusses the proposed algorithm CBGCM. Once the final clusters are obtained, the data needs to be prepared for distributed processing which is discussed in Data Distribution phase. The final phase is training the document embeddings on the distributed environment using MapReduce Framework.

### **4.2.1 Data Extraction and Data Preparation**

The legal documents are highly unstructured. Initially, the Supreme Court Judgements are fetched from the IndianKanoon [31] website in the form of HTML pages. The citation information in these HTML pages is present in between the anchor tags. These are extracted and used to construct a graphical representation of the judgements, and then graph-based clustering algorithms is applied to it. Apart from this, the text content of each of these judgements is also required for training the document embeddings. This is also extracted from the HTML pages, and appropriate text pre-processing is performed.

### **4.2.2 Graph Clustering**

Clustering is performed on the network representation of the judgements. The main reason behind clustering is that referentially similar judgements are present in the same cluster, and Doc2Vec can be applied to this set of judgements. This project uses overlapping graph clustering because a judgement can belong to multiple case-matters. From the literature survey, four overlapping graph clustering algorithms, PercoMCV, Angel, BigClam and SLPA are chosen, and experimentation is performed to find the most suitable algorithm for the legal judgement dataset.

### **4.2.3 Cluster Merging**

A significant issue after performing clustering is that it results in a large number of clusters. The results show that there are very few large clusters and numerous small clusters. Training Doc2Vec on

such small clusters is not as efficient since these clusters contain too few judgments. Another disadvantage of having many clusters is that more workers will be required in the MapReduce framework. This project proposes the Centrality Based Graph Cluster Merging(CBGCM) algorithm, which merges the small clusters into larger ones, thus, reducing the overall number of clusters.

---

**Algorithm 1** A proposed algorithm to merge the small clusters into large clusters

---

```

1: procedure
  CENTRALITYBASEDGRAPHCLUSTERMERGING(clusters, edge_list, min_size, k)
2:   big_clusters  $\leftarrow$  emptyset()
3:   small_clusters  $\leftarrow$  emptyset()
4:   for each cluster c  $\in$  clusters do
5:     if c.size()  $\geq$  min_size then
6:       big_clusters.add(c)
7:     else
8:       small_clusters.add(c)
9:     end if
10:  end for
11:  for each cluster i  $\in$  small_clusters do
12:    merging_index  $\leftarrow$  -1
13:    max_conn  $\leftarrow$  0
14:    for each index j  $\in$  big_clusters.size() do
15:      central_nodes  $\leftarrow$  GetKCentralNodes(big_clusters[j], k)
16:      total_conn  $\leftarrow$  GetConnection(i, centralNodes, edgeList)
17:      if total_conn  $\geq$  max_conn & Conductance(i  $\cup$  big_clusters[j])  $\leq$ 
        Conductance(big_clusters[j]) then
18:        merging_index  $\leftarrow$  j
19:        max_conn  $\leftarrow$  total_conn
20:      end if
21:    end for
22:    if merging_index  $\neq$  -1 then
23:      Merge(big_clusters[merging_index], i)
24:    end if
25:  end for
26:  return big_clusters
27: end procedure

```

---

N. Kasoro et al. [23], in the clustering algorithm PercoMCV, proposed using eigenvector centrality for merging unclustered nodes into the clusters. An unclustered node linked to the central nodes of any cluster is a member of the central node community. This method can be exploited for merging *small\_cluster* into the *big\_cluster*. The clusters with a size greater than a threshold are considered

as *big\_clusters*. Thus, a *small\_cluster* with connections to central nodes of large cluster belongs to its central node community and can be merged with it.

A *small\_cluster* is merged into the *big\_cluster* which has the maximum number of connections to its central nodes. While merging, a criterion is set, that the two clusters are merged if and only if the resultant cluster has a conductance score less than the conductance of the previous *big\_cluster*, thus preserving the *big\_cluster* quality.

The most widely used methods for finding the central nodes of a cluster are [8] : Degree centrality, EigenVector centrality and Betweenness centrality. All these techniques have been tested on the graphical representation of legal judgements. CBGCM algorithm [1] represents the pseudo code of the proposed method.

#### **4.2.4 Data Distribution**

Processing a large volume of judgments demands enormous computational resources, which the commodity machine can not justify. To justify such demand, this project learns a judgment embedding using a distributed framework like MapReduce. Each cluster obtained after cluster merging is considered as a single judgement set, and document embedding is trained on each of these by a separate worker in the MapReduce framework. Apache Hadoop [4] is used for the MapReduce framework for experimental purposes.

#### **4.2.5 Document Embedding using Doc2Vec**

This project utilises Doc2Vec as embedding technique and finding similarity between the judgements. Once the cluster are formed and the data is distributed, Doc2Vec is trained on a multi-node Hadoop cluster using the MapReduce Framework.

### **4.3 Summary**

This project, thus, systematically presented an overview of the methodology proposed. By dividing our problem statement into two broad parts, namely clustering the graph and applying Doc2Vec using MapReduce framework on each cluster formed, we intend to modularise our problem-solving approach. By intensively studying previous works regarding the same, the methods proposed achieves the stated objectives.

# Chapter 5

## Simulation and Results

This chapter describes our experimental simulations and results. It consists of a description of our initial data preprocessing techniques, along with the description and testing results of the various algorithms utilized for performing overlapping graph clustering. It also discusses the selection of optimal hyperparameters for SLPA, an algorithm on which the best results were obtained.

Later, the cluster merging is performed using the proposed CBGCM algorithm [1] to reduce the total number of clusters. The algorithm has experimented with three widely used centrality measures, and the best hyperparameter for CBGCM Algorithm is determined corresponding to the most optimal centrality measure. The performance of the final clusters formed is compared with the expert scores. Finally, this chapter presents the performance obtained by training Doc2Vec embedding varying different hyper-parameters in order to obtain their optimal values. Also, the performance improvement obtained by training the embeddings on multi-node Hadoop cluster is presented.

### 5.1 Data Extraction and Pre-Processing

The experiments have been conducted using the dataset consisting of judgements delivered by the Supreme Court of India from 1947-2016 over the years. These judgments have been fetched from the Indian Kanoon website [31]. The collected data consists of a total of 48950 judgements.

For clustering, in order to construct a graphical representation of the judgements, the citation information in every judgement which is enclosed between the anchor tags with IDs is extracted. The

extracted citation information is stored in a separate file in the form of edge list representation of the graph. The formed graph consists of 80118 nodes and 424184 edges. The nodes in the graph represent the judgements as well as articles present in the Indian Legal System. The constructed graph consists of many isolated connected components. Out of all the connected components, more than 99% of the nodes of the graph belonged to the largest connected component. This component is extracted and considered for the clustering purpose.

For training the Doc2Vec embedding, the judgement text is extracted from the HTML pages. Within the HTML pages, all the text content is present in a div tag with class “judgements”. Along with this, It is necessary to preprocess the legal judgements data to better train the model. Traditional text preprocessing techniques are used to remove the inconsistency and non-uniformity in the judgement text. The following traditional preprocessing steps are applied to the highly unstructured judgements data:

- Transforming the text content into the lower case.
- Individual words are extracted and tokenized with the help of white spaces, line breaks and paragraphs.
- Removal of stop-words such as “a”, “an”, “the”, etc. and words less than 3.
- Elimination of white spaces, newline and punctuation marks.

## **5.2 Evaluation Metrics**

This section defines the various evaluation metrics which have been utilised for the experiments that are conducted in this project. This includes the various performance metrics for evaluating the clustering on graphical data [32] and other performance metrics for comparison with the expert scores.

1. Number of nodes clustered: This represents the total number of nodes clustered using a clustering algorithm. An algorithm which can cluster large numbers of judgements is considered.
2. Conductance: This represents the fraction of total edge volume that points outside the community. The value of Conductance is between 0 and 1. For good clusters, this value should



be as small as possible.

$$Conductance = \frac{c_s}{2m_s + c_s} \quad (5.1)$$

where  $c_s$  is the number of community nodes and  $m_s$  is the number of community edges.

3. **Overlapping (%)**: This represents percentage of overlapping over all clusters. Very low overlapping percentage indicates that the clusters formed are crisp, whereas, very high overlapping percentage indicates large number of redundant nodes in multiple clusters. Thus, a value neither too high nor too low is desirable.

$$Overlapping(\%) = \frac{\text{No. of nodes belonging to more than one clusters}}{\text{Total no. of unique nodes in all clusters}} \quad (5.2)$$

4. **Recall at K**: Given the top K recommended judgements for a query judgement, this represents the proportion of the judgements which belong to the same cluster as of the query judgement [33].

$$Recall\ At\ K = \frac{\text{No. of judgements belonging to the same cluster of query judgement}}{\text{Top K judgements for a query judgement}} \quad (5.3)$$

### 5.3 Experiment Setup

This project has two major motives. First is to perform overlapping graph clustering on the citation representation of legal judgements to form clusters, and second, is to train doc2vec embedding on each of the formed clusters in a distributed environment. Four overlapping graph clustering algorithms were considered based on literature. They are PercoMCV [23], Angel [26], SLPA [24] and Big Clam [28]. Experiments for overlapping graph clustering are performed on a machine with Intel i5-7300HQ processor running at 2.5 MHz using 8 GB of RAM, running on Ubuntu 20.04 LTS. NetworkX [9] and CDLib [34] libraries are used for implementation purposes.

For training the formed clusters distributively, the Hadoop multi-node cluster is configured, consisting of 3 computing nodes, where each machine consists of 16GB of RAM, with an Intel i7-7700 processor with 8 CPUs running on Ubuntu 14.04. Out of these three nodes, one node acts as master as well as slave while two nodes act as slaves only. Each node is configured with MapReduce

(Hadoop 2.7.2 with YARN Resource Manager), and Gensim for training the Doc2Vec embeddings. The Hadoop Distributed Files System is used for the distributed storage of the judgement cluster sets.

## 5.4 Cluster Formation Result Analysis

This section discusses the various experimentation carried out and presents their results obtained in the process of cluster formation. The final clusters formed after this act as separate blocks of judgements on which Doc2Vec embedding is trained using the MapReduce framework.

### 5.4.1 Clustering Results Analysis

Overlapping graph clustering is performed using the four algorithms [23, 24, 26, 28] and evaluated based on three parameters: Number of nodes clustered, Conductance, and Percentage of Overlapping. It can be observed from Table 5.1 that SLPA gives best results in comparison to the others. It considers all the nodes and also gives better Conductance scores as compared to other algorithms. Also, the overlapping percentage obtained from SLPA is neither too high nor too low.

PercoMCV is a clique-based clustering algorithm. Due to the structure of the legal judgments, there are fewer cliques possible. So, a large number of nodes remain unclassified as they do not belong to any cliques. Angel being a node-centric approach, does not give desirable results as the noise in every iteration increases at each step. BigClam is usually preferred for large-sized networks (millions of nodes), so it does not perform well with the legal network. Due to the inherent nature of SLPA, collection of nodes with more incoming edges are likely to propagate the same labels between them and hence will be more likely to be placed into the same community. Thus, SLPA gives satisfactory results and is further utilised.

Table 5.1: Comparison of clustering results for algorithms

Algorithm Name	No. of nodes clustered	Conductance	Overlapping (%)
PercoMCV	31319	0.8457	63.867
Angel	33023	0.8653	2.5728
BigClam	34410	0.8872	96.070
SLPA	79607	0.3721	11.782

SLPA algorithm has two parameters: threshold and number of iterations. Experiments with varying parameters were conducted with SLPA to find out the optimal values of these parameters. First the value of threshold is varied keeping the number of iterations constant at 30, to find out the best threshold for the algorithm.

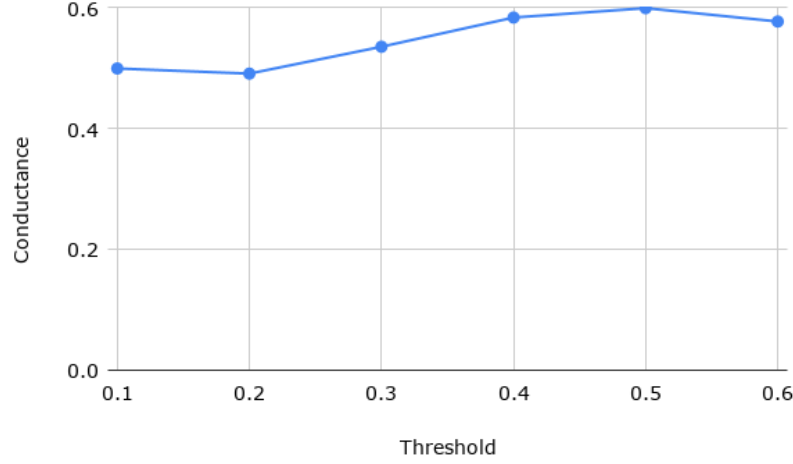


Figure 5.1: Conductance vs. Threshold keeping iterations constant

SLPA considers only label information that reflects the underlying network structure during evolution. Later, the communities are formed when the stored data is post-processed. If the probability of a specific node is smaller than a given threshold  $r \in [0, 1]$ , then the label is discarded. Nodes with similar labels are aggregated into a cluster. Nodes having more than one label belong to multiple overlapping clusters. So, as the value of  $r$  decreases, the number of overlapping clusters identified by the algorithm increases. As it can be seen from Figure 5.1,  $r=0.2$  gives the best Conductance scores and thus, it is selected. Now, the number of iterations is varied, keeping the threshold value fixed at 0.2.

It can be seen from Figure 5.2, as the number of iterations increases, the number of clusters decrease and a better score for Conductance and is obtained. This is because as the number of iterations increases, the nodes are better able to update the labels during the evolution process of SLPA when listeners update their labels according to those propagated by the speakers. From Figure 5.2 it can be seen that for 160 iterations the minimum Conductance is obtained. Hence, the parameter values are selected as number of iterations = 160 and threshold = 0.2.

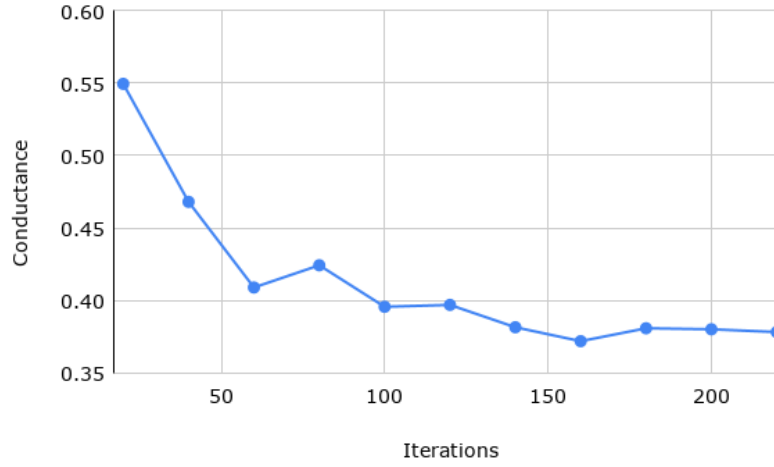


Figure 5.2: Conductance vs Iteration for threshold = 0.2

#### 5.4.2 Cluster Merging Result Analysis

This section discusses the results after merging clusters using the proposed CBGCM algorithm [1]. The most widely used techniques to find centrality are Degree centrality, EigenVector centrality and Betweenness centrality. The performance of these centrality measures is evaluated based on the final Conductance of merge clusters and the Number of overall nodes considered. The number of nodes considered by all techniques are approximately equal and there is a slight difference in conductance scores. However, the final conductance of Eigenvector centrality is the least. So, all other experiments have been carried out using EigenVector Centrality.

Table 5.2: Comparison of cluster merging techniques

Centrality Measure	No. of nodes considered	Conductance
Degree centrality	78976	0.2755
Eigenvector centrality	78985	0.2732
Betweenness Centrality	78983	0.2774

The two hyperparameters for the CBGCM algorithm are *min\_size* and *k*. *min\_size* is the minimum size at which a cluster is considered as *big\_cluster* and *k* is the number of top *k* central nodes to be taken into consideration. For this project, clusters of size more than 1000 are considered as *big\_cluster*. The value of *k* has been varied to find the value at which the maximum number of nodes are considered. From the Figure 5.3 and Figure 5.4 it can be seen that after the value of *k* =

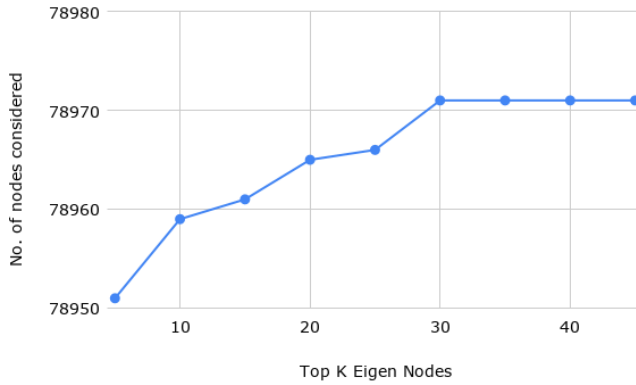


Figure 5.3: No. of nodes considered vs Top K Eigen Nodes

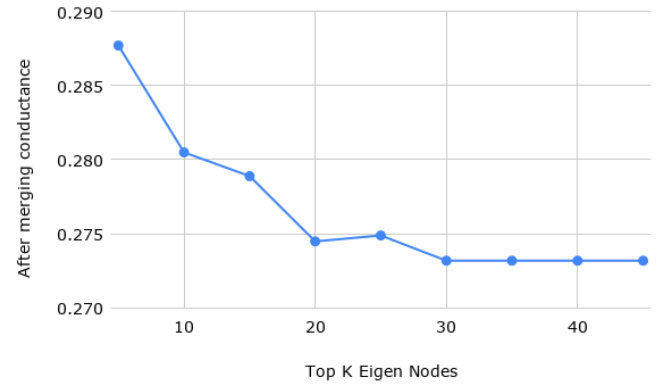


Figure 5.4: After merging conductance vs Top K Eigen Nodes

30 the number of nodes considered becomes constant and the lowest value of conductance is also obtained.

### 5.4.3 Expert Score based Results Analysis

This section compares the performance of the final clusters formed with the expert scores as well as CaseMine [35] dataset.

A set of 50 documents were analysed by legal experts, who gave a score of 0-10 for each pair of documents. Document pairs scoring 7 or above are labelled as similar. For comparing the performance of the clusters with the expert scores, it is checked whether the similar documents belong to the same cluster or not. Out of 50 judgement pairs, 10 judgement pairs were declared as similar by legal experts. Out of these 10 judgement pairs 9 belonged to the same cluster. So the overall recall of the clusters formed is 0.9.

For further comparison of the performance, 98 random judgements delivered by the Supreme Court of India were considered. For each of these judgements, 10 relevant judgements have been fetched from CaseMine. The Recall at K for each 98 judgements has been calculated for different values of K. The average value of Recall at K at different values of K is given in the Table 5.3.

Table 5.3: Average Recall at K for CaseMine data

Value of K	Average Recall (%)
1	84.94
2	85.204
3	84.354
4	84.439
5	85.102
6	84.864
7	84.257
8	84.694
9	85.034
10	84.082

## 5.5 Doc2Vec Model Training Result Analysis

After applying the proposed CBGCM algorithm [1] with optimal hyperparameters, 13 merged clusters are obtained. Each of these clusters represents referentially similar judgements. Since the project focuses on recommending similar judgements to the query judgement, the judgements' text must be vectorised for efficient processing. Mandal et al. [7] found that Doc2Vec outperformed other document embedding techniques. Thus, this project conducts various experiments for finding optimal hyperparameters to train the Doc2Vec model. Once each cluster of judgements is vectorised, similar judgements can be then found based on the cosine similarity of vectorised documents.

### 5.5.1 Hyperparameters for training Doc2Vec

The hyperparameters of training Doc2Vec model for which optimal values need to find out are:

- **Vector size (vSize):** dimensionality of the feature vectors.
- **Window (w):** the maximum distance between the current and predicted word within a sentence.
- **Epochs (iter):** Number of iterations (epochs) over the corpus.

For evaluating the performance of Doc2Vec models, a dataset of 50 documents analysed by legal experts [7], who gave a score of 0-10 for each pair of documents, is considered. From this dataset, document pairs scoring above 7 are considered similar, and those with a score of 5 or below are considered not similar. Now, cosine similarity is found between each pair of vectorised documents. The threshold is set at 0.5, i.e., document pairs having a cosine similarity of more than or equal to 0.5 are considered to be similar, and less than 0.5 are considered to be not similar. Accuracy and F1 Score have been utilised as the evaluation metrics.

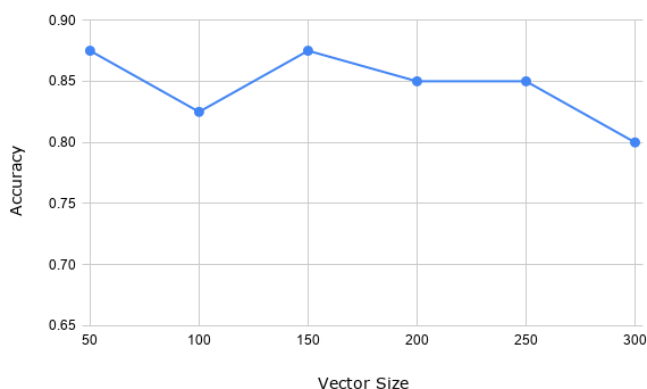


Figure 5.5: Accuracy vs. Vector Size

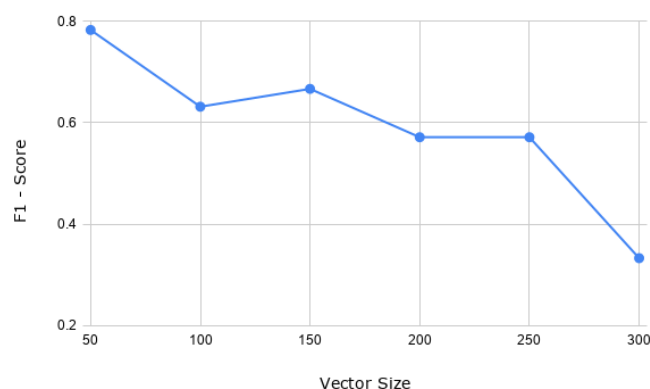


Figure 5.6: F1 - Score vs. Vector Size

First, Vector Size is varied, keeping the window fixed at 10 and epochs set to 10. As shown in Figure 5.5 and Figure 5.6, the highest accuracy of 0.875 and an F1 - Score of 0.7826 is obtained for Vector Size = 50; thus, it is selected.

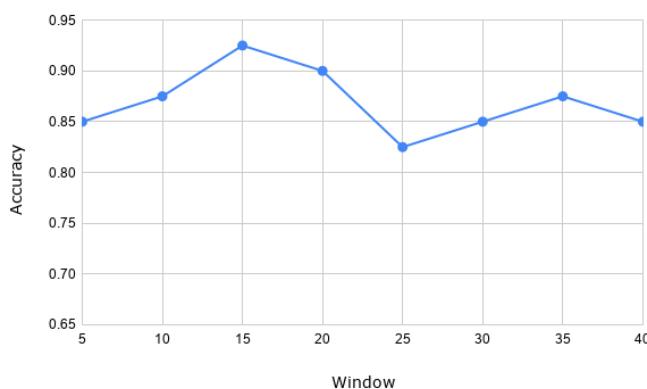


Figure 5.7: Accuracy vs. Window

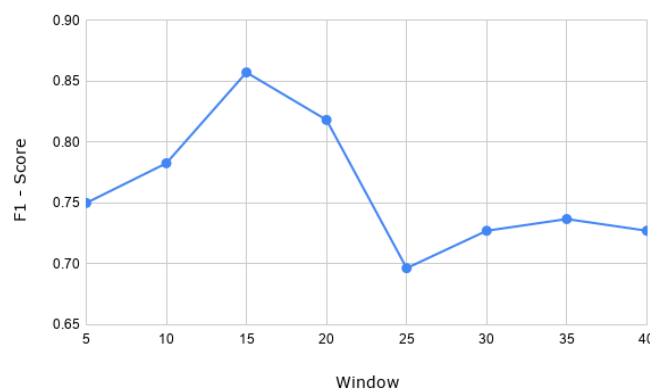


Figure 5.8: F1 - Score vs. Window

Now, the window values are varied, keeping the Vector Size fixed at 50 and epochs set to 10. As

it can be seen from Figure 5.7 and Figure 5.8, a Window = 15 gives the highest accuracy of 0.925 and an F1 - score of 0.8571; thus, it is selected.

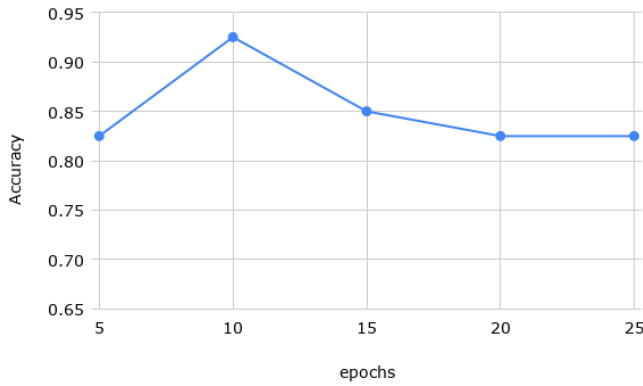


Figure 5.9: Accuracy vs. Epochs

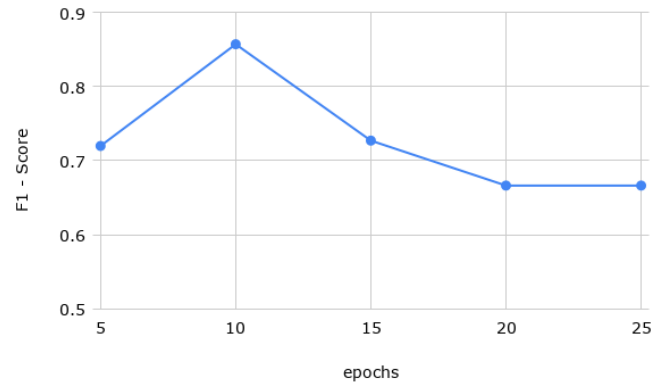


Figure 5.10: F1 - Score vs. Epochs

Finally, epochs were varied, keeping the Vector Size fixed at 50 and Window fixed at 15. It is evident from Figure 5.9 and Figure 5.10 that epochs = 10 gives the highest accuracy of 0.925 and an F1 score of 0.8571; thus, it is selected.

Hence, from the various experiments performed, Vector Size = 50, Window = 15 and epochs = 10 are obtained as the best optimal hyperparameters for training the Doc2Vec model.

### 5.5.2 Training the model on Hadoop multi-node

Table 5.4: Training times of Hadoop multi-node clusters

Number of Slaves	Training Time (in seconds)
1	204.667
2	153
3	82.334

Once the optimal hyperparameters for training the Doc2Vec model is obtained, experiments were conducted on a multi-node Hadoop cluster to measure the time taken to train the Doc2Vec model on each cluster of judgements. Experiments are conducted by varying the number of slaves in the Hadoop clusters from one to three and the improvements in the training time are recorded. The model is trained 3 times for each configuration of nodes and the average training time has been taken into consideration. As shown in Table 5.4, the training time for three slaves Hadoop



clusters is significantly lesser than the training time periods for a single slave Hadoop cluster and two slaves Hadoop cluster. Thus, using a Hadoop multi-node cluster, the training time period of Doc2Vec models can be significantly reduced.

## **5.6 Conclusion**

This chapter presents the implementation of the objectives of the project. The project's first task is to divide the judgment corpus into multiple blocks for learning document embedding on each block distributively. To achieve this objective, overlapping graph clustering is performed on the citation-based network representation of legal judgements. Four overlapping graph clustering algorithms were chosen based on literature for experimental purposes: PercoMCV, Angel, BigClam and SLPA. The SLPA algorithm gives promising results and is selected as the overlapping graph clustering algorithm for this project and further experiments are performed to obtain the optimal hyperparameters of the SLPA algorithm.

Clustering results in the formation of a large number of clusters. Training Doc2Vec on such small clusters is not as efficient since these clusters contains very less number of judgments. Thus, this project proposes the CBGCM algorithm to merge small clusters into larger ones, reducing the overall number of clusters. Experiments are performed to determine the optimal centrality measure, and Eigenvector centrality gives promising results and is chosen as the centrality measure of the CBGCM algorithm. The quality of final clusters formed after merging is evaluated with the expert scores judgment pair dataset and the CaseMine tool judgment pair dataset.

The next major task is to train the Doc2Vec embeddings in a distributed environemt. Experiments are performed to obtain the optimal values of the hyperparameters and the performance of the formed embeddings is evaluated using 50 judgement pairs rated by experts. Finally, the Doc2Vec embeddings are trained on a multi-node Hadoop cluster by varying the number of slaves. It is observed that the training time reduces significantly on increasing the number of slaves.

# Chapter 6

## Legal Judgement Search Engine

This chapter discusses the Legal Judgement Search Engine user interface. This search engine is a statistical search engine that allows a user to search for a judgment delivered by the Supreme Court of India based on various combinations of a judgment's metadata. The metadata includes the judgement ID, Judge Name, Petitioner, Respondent, and date fields related to judgements. Along with the statistical search, the Doc2Vec trained models are also embedded in the interface to recommend similar judgements for a selected judgement.

First, the data extraction and pre-processing are discussed, followed by the user interface's database schema, and in the end, a detailed discussion on the various views of the user interface is presented.

### 6.1 Data Extraction and Preprocessing

The dataset consists of judgements delivered by the Supreme Court of India from 1947-2016. These judgments have been fetched from the Indian Kanoon [31] website in the form of HTML pages. The collected data consists of a total of 48950 judgements. From these HTML pages, the metadata about every judgement is extracted using BeautifulSoup. This metadata includes the judgement ID, title, author of judgement, bench, petitioner, respondent and date of judgement. This metadata forms the parameters of the statistical search interface. The HTML pages stored are also used to retrieve the text content of the arguments made by the respondent and the petitioner in the proceedings.

## 6.2 Experimental Setup

The Legal Judgement Search Engine user interface is constructed using the Django [36] framework in Python, and MySQL [37] is used as the database. Apart from these, Beautiful Soup [38] is used for extracting and preprocessing text from the HTML pages of judgements. Gensim [39] is used for recommending similar judgements for a selected judgement by loading the trained Doc2Vec models. The entire interface is built on a machine with an Intel i5-7300HQ processor running at 2.5 MHz using 8 GB of RAM, running on Ubuntu 20.04 LTS.

## 6.3 Database Schema

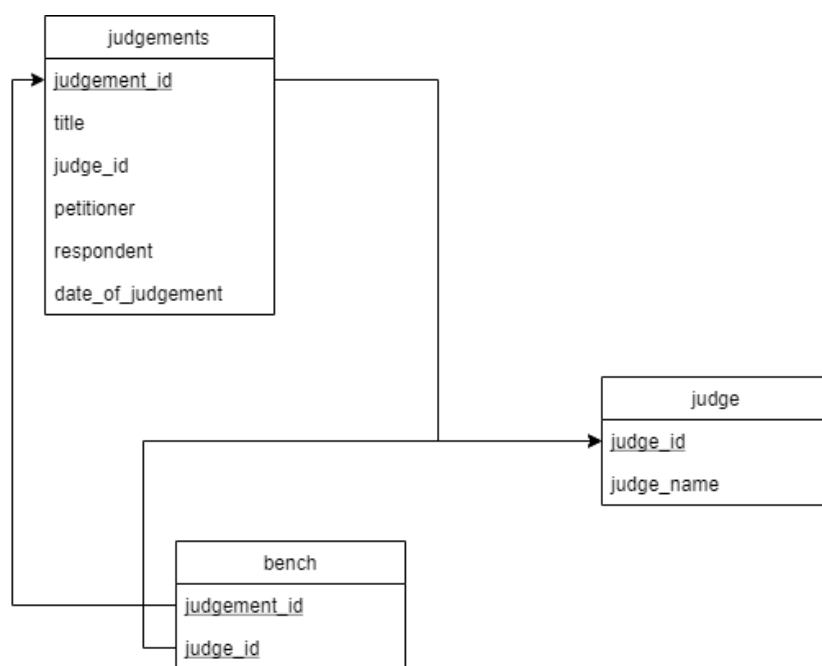


Figure 6.1: Schema

The database consists of three tables. They are:

1. **Judge Table:** This table consists of two attributes: **judge\_id** and **judge\_name**. The **judge\_id** is the primary key and is the unique ID of every Supreme Court judge of India. The attribute **judge\_name** refers to the name of the judge.
2. **Judgements Table:** This table contains the metadata about the judgements delivered by the Supreme Court of India and consists of multiple attributes. The **judgement\_id** is a unique ID

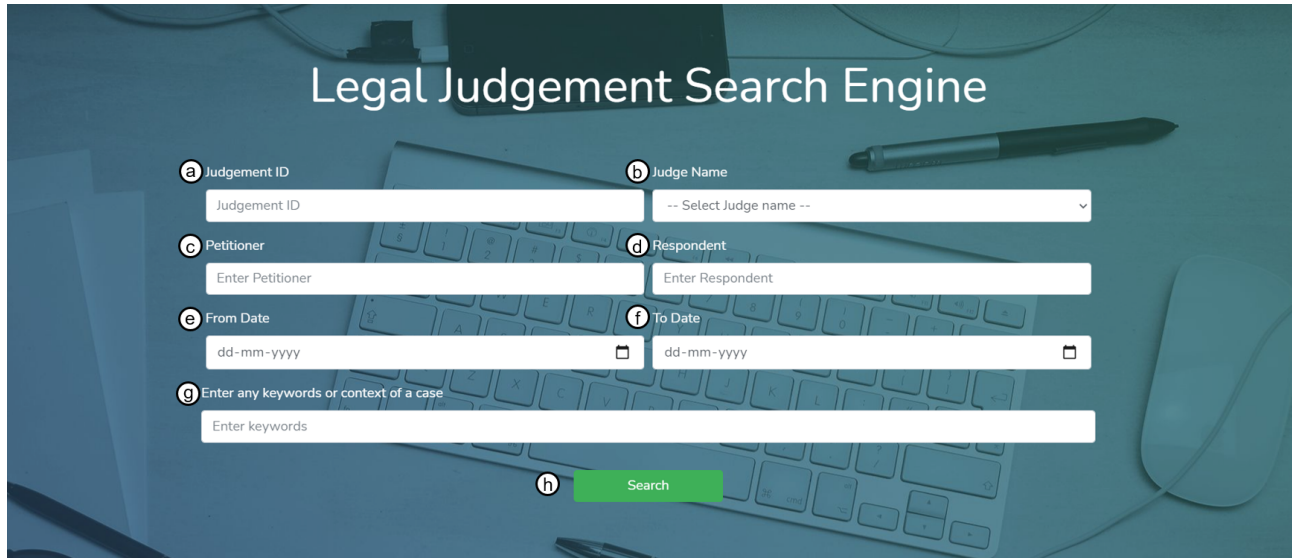


Figure 6.2: Home Page View

of a judgement and is the primary key. The title attribute represents the title of the judgement. judge\_id refers to the author of the judgment and has a foreign key constraint to the judge table. The attribute petitioner refers to the petitioner of the judgement, and the respondent refers to the respondent of the judgement. The date\_of\_judgement attribute refers to the date on which the judgement was given.

3. **Bench Table:** This table contains the bench information of every judgement. The attributes judgement\_id and judge\_id are foreign key constraints to the judgement table and judge table, respectively. These two together form the primary key of this table.

## 6.4 Elements of the Search Engine

This section discusses about the various views present in the Search Engine. The Search Engine consists of 3 major views: the Home Page View, the Search View and the Judgement View.

### 6.4.1 Home Page View

This is the homepage of the Search Engine user interface where the user can search for a particular judgement or a set of judgements by giving the different input parameters of a judgment like judge name, date, etc. as well as any text keyword as query input.

- (a) Here, the user can input the Judgement ID for the particular desired judgement. Each Judgement's ID is a unique identification number given to each judgment as per the IndianKanoon website.
- (b) Here, the user can input the Judge Name, which refers to the author of the judgement.
- (c) Here, the user can input the petitioner's name who has submitted the petition corresponding to the desired query judgement.
- (d) Here, the user can input the respondent's name who has responded to the petition corresponding to the desired query judgement.
- (e) Here, the user can input the starting date in order to view all the judgements delivered after that particular date.
- (f) Here, the user can input the ending date in order to view all the judgements delivered before that particular date.

If the user gives both starting date and ending date as the input, all the judgements delivered in between the two dates will be fetched and viewed. In this case, the ending date cannot be before the starting date.

- (g) Here, the user can input any number of keywords related to different case-matters and the judgments that contain those keywords will be retrieved. This has been implemented using inverted-index.
- (h) After providing any combination to the input field parameters, the user can click on the Search button to fetch all the judgement corresponding to the input query. This will redirect the user to the Search View.

For fetching judgements, at least one of the input fields must be provided. If no input is provided, then an error message is displayed informing the user to provide at least one of the input fields. Figure 6.3 shows this condition.

In the form shown in Figure 6.2 , form validation has been applied on all the input fields so that only the correct format corresponding to each input field is accepted. For, eg. The Judgement ID

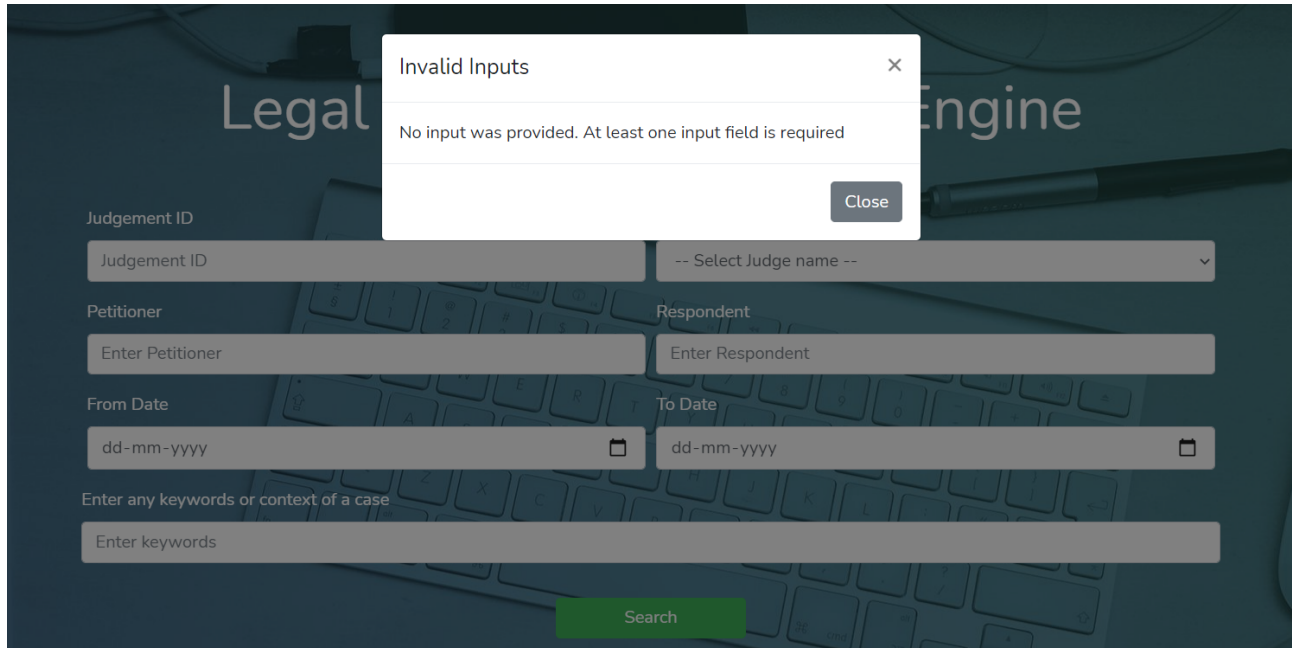


Figure 6.3: Error Message for No Input Provided

is an integer type input, so the Judgment ID input field (a) will only accept integer values, and any other input type will be prompted with an error message.

#### 6.4.2 Search View

This is the Search View which displays all the judgements that were retrieved as output corresponding to the input provided by the user in the Home Page View.

- (i) If the user wants to update the query input fields or can simply search for another judgment, then the user can press the “Search Another Judgment” button that will slide down the Search Form shown in Figure 6.4 from the top.
- (j) These are all the output judgements that were fetched corresponding to the input query. The image shows the first five judgements ordered by the date of judgements. By clicking on any of these judgements, the user is redirected to the Judgment View, where all the details corresponding to that judgement are shown.
- (k) For each judgement shown, all the metadata information is provided like the title, date, judge names etc., so that the desired judgement can be distinguished out of all fetched judgements.

i
Search Another Judgement

H.P. State Electricity Board Ltd vs Mahesh Dahiya on 18 November, 2016

Author : Hon'ble Mr. Justice Sharad Arvind Bobde  
Bench : Sharad Arvind Bobde, Ashok Bhushan  
Petitioner : H.P. State Electricity Board Ltd  
Respondent : Mahesh Dahiya  
Date : Nov. 18, 2016

Vivek Singh vs State Of U.P & Anr on 26 August, 2016

Author : Hon'ble Mr. Justice Sharad Arvind Bobde  
Bench : Sharad Arvind Bobde, Ashok Bhushan  
Petitioner : Vivek Singh  
Respondent : State Of U.P & Anr  
Date : Aug. 26, 2016

U.Subhadramma & Ors vs State Of A.P Rep.By ... on 4 July, 2016

Author : Hon'ble Mr. Justice Sharad Arvind Bobde  
Bench : Sharad Arvind Bobde, Amitava Roy  
Petitioner : U.Subhadramma & Ors  
Respondent : State Of A.P Rep.By  
Date : July 4, 2016

Eitzen Bulk A/S vs Ashapura Minechem Limited & Anr on 13 May, 2016

Author : Hon'ble Mr. Justice Sharad Arvind Bobde  
Bench : Fakkir Mohamed Ibrahim Kalifulla, Sharad Arvind Bobde  
Petitioner : Eitzen Bulk A/S  
Respondent : Ashapura Minechem Limited & Anr  
Date : May 13, 2016

M.K.Indrajeet Sinhji Cotton ... vs Narmada Cotto Coop.Spg.Mills ... on 26 April, 2016

Author : Hon'ble Mr. Justice Sharad Arvind Bobde  
Bench : Sharad Arvind Bobde, Amitava Roy  
Petitioner : M.K.Indrajeet Sinhji Cotton  
Respondent : Narmada Cotto Coop.Spg.Mills  
Date : April 26, 2016

i
k

i
1 2 3 4 5 6 »

Figure 6.4: Search View

45

- (l) Pagination is also provided at the bottom in the Search View so that only a limited number of judgements are displayed at a time on the screen for the ease of the user. The next judgments can be navigated by pressing to the relevant page number.

### **6.4.3 Judgement View**

The Judgement View contains all the details corresponding to the judgement that is selected from the Search View. Along with that, a recommendation of similar judgements corresponding to the current judgement is also presented on the right hand on the panel. For recommending similar judgements, the trained Doc2Vec models are used.

- (m) If the user wants to update the query input fields or simply search for another judgment, the user can press the “Search Another Judgement” button that will slide down the Search Form shown in Figure 6.5 from the top.
- (n) This is the title of the current judgement that is selected from the Search View. It contains all the metadata information of the judgement like the title, date, author and bench.
- (o) This is the entire judgment body of the current judgement containing all the arguments made in the case, as mentioned in the original Supreme Court website.
- (p) This right panel contains top-recommended judgements that are similar to the current judgement. These are ranked based on their similarity to the current judgement and are generated by the pre-trained models.
- (q) Pagination is also provided at the bottom of the right panel in the Judgement View so that only a limited number of judgements are displayed at a time on the screen for the ease of the user. The next judgments can be navigated by pressing the relevant page number.



Author: Sharad Arvind Bobde

Date of Judgement: Oct. 18, 2013

©

IN THE SUPREME COURT OF INDIA  
CRIMINAL APPELLATE JURISDICTION  
CRIMINAL APPEAL Nos. 1521-1522 OF 2011

PUTCHALAPALLI NARESH REDDY  
APPELLANT

VERSUKS

STATE OF A.P. & ETC.  
RESPONDENTS

WITH

CRIMINAL APPEAL NOS. 1102-1103 OF 2011

WITH

CRIMINAL APPEAL NOS. 1100-1101 OF 2011

WITH

CRIMINAL APPEAL NOS. 1093-1094 OF 2011

## JUDGMENT

S. A. BORDE, J.

1. This batch of appeal is preferred by the accused against the common judgment and order dated 23rd March 2004 in Criminal Appeal Nos. 954 and 956 of 2008. Criminal Appeal Nos. 1521-1522 of 2011 are filed by the Accused No. 1; Criminal Appeal Nos. 1102-1103 of 2011 are filed by the accused Nos. 2-7; Criminal Appeal Nos. 1100-1101 of 2011 by the accused Nos. 8-12; Criminal Appeal Nos. 1093-1094 of 2011 are filed by the Accused No. 2, Konduru Nagure Reddy. Since they arise out of the same incident/judgment they have been taken up together for disposal.

2. The crime in question is the murder of Mudi Parandhami Reddy in an agricultural field on 25.11.96 in Mettu village in Andhra Pradesh. According to the prosecution, the deceased was a leader of the Congress party in the area. He was suspected by the Accused No. 1 Puchalapalli Parandhami Reddy, a leader of the Telugu Desam Party, as being responsible for the murder of his father on 25.3.92. Land disputes also existed between Rajagopal Reddy (L.W. 2), his brother Pelluru Murali Reddy (P.W. 3) on one side and A-19, Pelluru Venu Reddy about the division of survey No.

116. Litigation was pending between them in a Court. Therefore, Pelluru Murali Reddy (P.W. 3) and Rajagopal Reddy (L.W. 2) sought the mediation of the deceased Mudi Parandhami Reddy and it was decided that the deceased would mediate on 25.11.96.

3. On 24.11.96, Pelluru Murali Reddy (P.W. 3) and his brother Rajagopal engaged farm labour i.e. P.W. 1 (Vakati Ramaiah), P.W. 2 (Rayapati Venkata Ramaiah), P.W. 4 (Bandila Muthyalayiah) and P.W. 5 (Rayapati Kotiah) for ploughing the land. On the next day, i.e. 25.11.1996 the deceased had come to the land and sat on a ridge in the field while the ploughing was going on by the labourers. Around 8.00 a.m., A-19, Pelluru Venu Reddy and A-20, Pelluru Sreedhar Reddy came there and asked that the ploughing be stopped because there was a dispute over the land. Mudi Parandhami intervened and said that the dispute will be resolved later and asked the ploughing to

Similar judgements to this judgement

B. K. Channappa vs State Of Karnataka on  
10 November, 2006

Author : Hon'ble Mr. Justice Prafulla Chandra Pant  
Bench : Ashok Kumar Mathur, Lokeshwar Singh Pant  
Petitioner : B. K. Channappa  
Respondent : State Of Karnataka  
Date : Nov. 10, 2006

Majju & Anr vs State Of Madhya Pradesh  
on 19 October, 2001

Author : Hon'ble Mr. Justice Konakuppakatti  
Gopinathan Balakrishnan  
Bench : Umesh Chandra Banerjee  
Petitioner : Majju & Anr  
Respondent : State Of Madhya Pradesh  
Date : Oct. 19, 2001

Abdul Rashid Abdul Rahiman Patel ... vs  
State Of Maharashtra on 10 July, 2007

Author : Hon'ble Mr. Justice B.N. Agrawal  
Bench : Prakash Prabhakar Naolekar, Dalveer  
Bhandari  
Petitioner : Abdul Rashid Abdul Rahiman Patel  
Respondent : State Of Maharashtra  
Date : July 10, 2007

Sabbita Satyavathi vs Bandala  
Srinivasarao & Ors on 15 March, 2004

Author : Hon'ble Mr. Justice Bisheshwar Prasad Singh  
Bench : Nitte Santosh Hegde, Bisheshwar Prasad Singh  
Petitioner : Sabbita Satyavathi  
Respondent : Bandala Srinivasarao & Ors  
Date : March 15, 2004

Changdeo Nivrutti Kamathe vs State Of  
Maharashtra on 7 October, 2009

Author : Hon'ble Mr. Justice Harjit Singh Bedi  
Bench : Harjit Singh Bedi, R. M. Lodha  
Petitioner : Chanddeo Nivrutti Kamathe  
Respondent : State Of Maharashtra  
Date : Oct. 7, 2009

⑨

« 1 2 3 4 »

Figure 6.5: Judgement View

# Chapter 7

## Conclusion and Future Work

### 7.1 Conclusion

The Common Law System is the most prevalent legal system in India. It requires legal practitioners to identify and refer to previous judgements for a case at hand to prepare advantageous arguments against the opponents. This process is manually cumbersome and requires an automated system. However, large volumes of judgements result in scalability issues. To overcome the issue, this project uses the citation information present in the legal judgements to form a network representation of the corpus. Clustering is performed on this network to split the legal corpus into multiple blocks. Various overlapping graph clustering algorithms were explored, and it is observed that SLPA gives the promising results for the corpus of legal documents. Clustering results in the formation of a large number of clusters. Thus, this project proposes the CBGCM algorithm for merging small clusters into larger ones, reducing the total number of clusters.

Once the clusters are obtained, Doc2Vec model needs to be trained on each of the clusters. Doc2Vec has been used as the document embedding technique for the judgements as it is able to preserve the semantic similarity between the judgements in the vector space. Cosine similarity is utilized for finding similarity between the vector representations of the legal judgements. Finally, the Doc2Vec embeddings are trained on a Hadoop multi-node cluster and it is observed that increasing the number of slaves, greatly reduces the training time.

Finally, this project has implemented a Legal Judgement Search Engine which allows the user to search any judgement delivered by the Supreme Court of India based upon the metadata of the

judgement, i.e. judge name, petitioner, respondent, date of judgement or any keyword. Once, a user selects a judgement, apart from the proceedings of the judgement, the Search Engine also recommends top similar judgements using the above trained Doc2Vec embeddings.

## **7.2 Future Work**

The work proposed in this project can be extended to a larger corpus of judgements by considering all the judgements delivered by the judiciary systems present in India. The training of Doc2Vec embeddings can be performed on other distributed environments to present a comparative analysis with the Hadoop MapReduce environment. Also, the overall efficiency of the Doc2Vec models trained can be enhanced by using concepts such as pre-trained word embedding or other techniques. In the Legal Judgement Search Engine, the implementation of the keyword based search can be further improved by using inverted index map in binary representation. This would improve the space complexity and time complexity of recommending judgements by using binary operations.

# References

- [1] S. Kumar, P. K. Reddy, V. B. Reddy, and A. Singh, “Similarity analysis of legal judgments,” in *Proceedings of the Fourth Annual ACM Bangalore Conference*, COMPUTE ’11, (New York, NY, USA), pp. 23–27, Association for Computing Machinery, 2011.
- [2] M. Van Opijnen and C. Santos, “On the concept of relevance in legal information retrieval,” *Artificial Intelligence and Law*, vol. 25, p. 65–87, Mar. 2017.
- [3] S. Visa, B. Ramsay, A. Ralescu, and E. Knaap, “Confusion matrix-based feature selection,” *CEUR Workshop Proceedings*, vol. 710, pp. 120–127, 01 2011.
- [4] “Apache hadoop.”, Available: "<https://hadoop.apache.org/>"(Accessed on 30-01-2021).
- [5] V. Chauhan and M. Sharma, “A review: Mapreduce and spark for big data analytics,” *International Journal of Advanced Technology in Engineering and Science*, vol. 4, pp. 42–50, 06 2016.
- [6] R. Kalyanasundaram, “Exploring the knowledge of citations in legal information retrieval,” *AI4J–Artificial Intelligence for Justice*, no. 12, pp. 30–37, 2017.
- [7] A. Mandal, R. Chaki, S. Saha, K. Ghosh, A. Pal, and S. Ghosh, “Measuring similarity among legal court case documents,” in *Proceedings of the 10th Annual ACM India Compute Conference*, Compute ’17, (New York, NY, USA), p. 1–9, Association for Computing Machinery, 2017.
- [8] Q. Wu, X. Qi, E. Fuller, and C.-q. Zhang, “Follow the leader”: A centrality guided cluster-

- ing and its application to social network analysis,” *The Scientific World Journal*, vol. 2013, pp. 368568–1–9, 10 2013.
- [9] “Networkx : Network analysis in python.” , Available: "<https://networkx.org/documentation/stable//index.html>"(Accessed on 25-10-2020).
- [10] Q. Le and T. Mikolov, “Distributed representations of sentences and documents,” in *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML’14, p. II–1188–II–1196, JMLR.org, 2014.
- [11] R. Sethy and M. Panda, “Big data analysis using hadoop: A survey,” *International Journal of Advance Research in Computer Science and Software Engineering*, vol. 5, pp. 1153–1157, 08 2015.
- [12] J. Dean and S. Ghemawat, “Mapreduce: Simplified data processing on large clusters,” *Association for Computing Machinery*, vol. 51, p. 107–113, Jan. 2008.
- [13] S. Qaiser and R. Ali, “Text mining: Use of tf-idf to examine the relevance of words to documents,” *International Journal of Computer Applications*, vol. 181, pp. 25–29, Jul 2018.
- [14] T. Maxwell and B. Schafer, “Concept and context in legal information retrieval,” *Frontiers in Artificial Intelligence and Applications*, vol. 189, pp. 63–72, 01 2008.
- [15] A. Shankar and V. N. Buddarapu, “Deep ensemble learning for legal query understanding,” in *Proceedings of CIKM 2019 Workshop on Legal Data Analytics and Mining (LeDAM 2018)*, CEUR-WS. org, pp. 13–17, 01 2019.
- [16] P. Bafna, D. Pramod, and A. Vaidya, “Document clustering: Tf-idf approach,” in *in Proceedings of the 2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*, pp. 61–66, 03 2016.
- [17] R. K. V and K. Raghuveer, “Article: Legal documents clustering using latent dirichlet allocation,” *International Journal of Applied Information Systems*, vol. 2, pp. 27–33, May 2012. Published by Foundation of Computer Science, New York, USA.

- [18] D. M. Blei, A. Y. Ng, and M. I. Jordan, “Latent dirichlet allocation,” *Journal of Machine Learning Research*, vol. 3, p. 993–1022, March 2003.
- [19] E. S. Negara, “A review on overlapping and non-overlapping community detection algorithms for social network analytics,” *Far East Journal of Electronics and Communications*, vol. 18, pp. 1–27, 01 2018.
- [20] S. Gupta and D. P. Singh, “Recent trends on community detection algorithms: A survey,” *Modern Physics Letters B*, vol. 01, no. 10, pp. 2050408–1–24, 2020.
- [21] A. Amelio and C. Pizzuti, “Overlapping community discovery methods: A survey,” *Social networks Analysis and case studies*, vol. 8, pp. 105–125, 06 2014.
- [22] H. Shen, X. Cheng, K. Cai, and M.-B. Hu, “Detect overlapping and hierarchical community structure in networks,” *Physica A: Statistical Mechanics and its Applications*, vol. 388, no. 8, pp. 1706–1712, 2009.
- [23] N. Kasoro, S. Kasereka, E. Mayogha, H. Vinh, and J. Kinganga, “Percomecv: A hybrid approach of community detection in social networks,” in *ANT/EDI40*, vol. 151, pp. 14–19, 05 2019.
- [24] J. Xie, B. K. Szymanski, and X. Liu, “Slpa: Uncovering overlapping communities in social networks via a speaker-listener interaction dynamic process,” vol. 6, pp. 344–349, 09 2011.
- [25] M. Coscia, G. Rossetti, F. Giannotti, and D. Pedreschi, “Demon: A local-first discovery method for overlapping communities,” in *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’12, (New York, NY, USA), p. 615–623, Association for Computing Machinery, 2012.
- [26] G. Rossetti, “Exorcising the demon: Angel, efficient node-centric community discovery,” in *Complex Networks and Their Applications VIII* (H. Cherifi, S. Gaito, J. F. Mendes, E. Moro, and L. M. Rocha, eds.), (Cham), pp. 152–163, Springer International Publishing, 2020.
- [27] A. Epasto, S. Lattanzi, and R. Paes Leme, “Ego-splitting framework: From non-overlapping to overlapping clusters,” KDD ’17, (New York, NY, USA), p. 145–154, Association for Computing Machinery, 2017.

- [28] J. Leskovec, "Overlapping community detection at scale: A nonnegative matrix factorization approach," *WSDM 2013 - Proceedings of the 6th ACM International Conference on Web Search and Data Mining*, vol. 7, pp. 587–596, 02 2013.
- [29] S. Gupta and D. P. Singh, "Recent trends on community detection algorithms: A survey," *Modern Physics Letters B*, vol. 34, p. 2050408, 09 2020.
- [30] E. S. Negara, "A review on overlapping and non-overlapping community detection algorithms for social network analytics," *Far East Journal of Electronics and Communications*, vol. 18, pp. 1–27, 01 2018.
- [31] "Indian kanoon - search engine for indian law." , Available: "<https://indiankanoon.org/>" (Accessed on 20-08-2020).
- [32] J. Yang and J. Leskovec, "Defining and evaluating network communities based on ground-truth," *Knowl. Inf. Syst.*, vol. 42, p. 181–213, 02 2015.
- [33] "Recall and precision at k." , Available: "[https://medium.com/@m\\_n\\_malaeb/recall-and-precision-at-k-for-recommender-systems-618483226c54](https://medium.com/@m_n_malaeb/recall-and-precision-at-k-for-recommender-systems-618483226c54)" (Accessed on 15-02-2021).
- [34] "Cdlb - community discovery library." , Available: "<https://cdlib.readthedocs.io/en/latest/index.html>" (Accessed on 25-10-2020).
- [35] "Casemine: The most granular mapping of indian case law." , Available: "<https://www.casemine.com/>" (Accessed on 09-02-2021).
- [36] "Django." , Available: "<https://www.djangoproject.com/>" (Accessed on 01-04-2021).
- [37] "Mysql." , Available: "<https://www.mysql.com/>" (Accessed on 01-04-2021).
- [38] "Beautiful soup documentation." , Available: "<https://www.crummy.com/software/BeautifulSoup/bs4/doc/>" (Accessed on 01-04-2021).
- [39] "Gensim: Topic modelling for humans." , Available: "<https://radimrehurek.com/gensim/index.html>" (Accessed on 15-02-2021).

- [40] R. Venkatesh, “Legal documents clustering and summarization using hierarchical latent dirichlet allocation,” *IAES International Journal of Artificial Intelligence (IJ-AI)*, vol. 2, pp. 27–35, 03 2013.
- [41] D. Carvalho, V. D. Tran, V.-K. Tran, and L. Nguyen, “Improving legal information retrieval by distributional composition with term order probabilities,” *EPiC Series in Computing*, vol. 47, pp. 43–56, 06 2017.



# Acknowledgement

We would like to express our deep gratitude and indebtedness to our project guides, Dr Rupa G. Mehta, Associate Professor, Computer Engineering Department, SVNIT Surat and Mr Jenish Dhanani, PhD. Scholar, Computer Engineering Department, SVNIT Surat for their valuable guidance, useful feedback and co-operation with kind and encouraging attitude at all stages of the experimental work for the successful completion of this work. We would also like to thank Dr Mukesh A. Zaveri, Head of Department, Computer Engineering Department. We are also thankful to SVNIT Surat and its staff for providing this opportunity which helped us gain sufficient knowledge to make our work successful.