LEGAL DATA SUMMARIZER

Chirag Kumar, Prateek Sharma

JK Lakshmipat University Jaipur 302026 India

ARTICLE INFO

Keywords:
Big Data Analytics
Legal Data
Text Summarizer
Prediction
Indian Judiciary
Named Entity Recognition (NER)

ABSTRACT

This report proposes a text summarization mechanism to address the challenges of summarizing lengthy and unstructured legal documents in India. The objective is to develop a system that can extract important information from these documents and generate concise summaries resembling human-generated summaries. The approach involves treating summarization as a binary classification problem and using an Extractive Summarization Technique based on statistical features and word vectors. The system, denoted as V, selects summary statements from the input full text section R. Various classifiers, including logistic regression, gradient boosting, and neural networks, are employed to automate the summarization process.

1. Introduction

India has a common law system that prioritizes the doctrine of legal precedent over statutory law, and where legal documents are often written in an unstructured way. These legal judgements are often quite long, and typically taking long man hours to go through it. [1] In order to address the challenges posed by lengthy and unstructured legal judgments in India, a proposed solution is to develop summaries for these documents. These summaries would capture the essential experiences and viewpoints of the Supreme Court, providing valuable and educational information for judges, lawyers, practitioners, and students. However, the existing shortage of specialized legal professionals, coupled with their high cost, makes the task of summarizing these judgments difficult.[2] To overcome this challenge, the proposed solution involves the development of a Text Summarization mechanism. Text summarization aims to extract crucial information from the original document. Specifically within the legal domain, the objective of an automatic text summarization system is to generate summaries that closely resemble those generated by humans. Drawing from our observations of existing gist statements, we can approach the generation of summaries as a sentence classification problem. [3]

One approach to addressing the challenges posed by lengthy and unstructured legal judgments in India is to develop concise summaries for these documents. These summaries would encapsulate the important experiences and perspectives of the Supreme Court, offering instrumental and educational information for judges, lawyers, practitioners, and students. However, the task of summarizing such judgments is typically entrusted to a limited number of specialized legal professionals, who are not only expensive but also in short supply.[4] To tackle this problem, we propose the implementation of a Text Summarization mechanism. Text summarization involves extracting the crucial content from the original document. In the legal domain, the primary goal of an automatic text summarization system is to generate summaries that closely resemble those crafted by human experts. By leveraging our observations of existing gist statements, we can treat the process of generating summaries as a sentence classification problem.

ORCID(s):

1.1. Problem Statement

The objective of our project is to develop a mechanism for summarizing legal documents by framing it as a binary classification problem. To achieve this, we will employ an Extractive Summarization Technique that utilizes statistical features of each sentence. These features include sentence length, entity type, similarity degree, term frequency-inverse sentence frequency, and keywords. By analyzing these features, we can generate a summary of the document.[5]

Let's denote the statements of the full text section as R. Our goal is to build a service, V, that can select certain statements S from R, which will serve as the summary statements, G.

Mathematically, we can represent the service as follows:
$V(R) \rightarrow S$

To tackle this task of sentence selection, we approach it as a classification problem. Alongside relevant features derived from linguistic and legal perspectives, we utilize various types of word vectors. Our service, V, incorporates classifiers based on concepts such as logistic regression, gradient boosting, neural networks, and other methods.[6]

By leveraging these techniques and classifiers, we aim to create an effective and efficient mechanism for summarizing legal documents, enabling the extraction of key information and generating concise summaries.

2. Related Work

There has been a lot of work that has been done in this field of text summarization. And as we are also doing some similar job of text summarization i.e. legal data summarization so we have reviewed some of the papers in which the auuthors have contributed in various aspects. So before heading to our work we would like to discuss about some of the works that has been done by different authors as of now.

In paper [7], the authors have done a survey on text summarization from legal documents. The exponential growth of

online information in the legal domain has led to a significant increase in the importance of legal text processing. In this paper, their objective is to provide an extensive survey of various text summarization techniques that have emerged in recent years. Of particular interest is the field of legal text summarization, which holds immense significance within the legal domain. To begin, they offer a comprehensive introduction to text summarization, highlighting its fundamental concepts and principles. Subsequently, they explore the advancements made in both single and multi-document summarization, focusing on their relevance and application to legal text summarization. Their analysis then delves specifically into extraction-based approaches for summarizing legal texts, considering their effectiveness and suitability for this specialized field. They address the availability of different datasets and metrics that have been utilized to evaluate the performance of summarization techniques. By comparing the outcomes of various approaches, they assess their general applicability before shifting their focus towards the specific challenges and considerations of legal text summarization. Additionally, they highlight noteworthy summarization techniques that have shown promising results in this domain. In their study, they also provide a brief overview of software tools that have been developed specifically for legal text summarization, outlining their features and functionalities. These tools play a crucial role in assisting legal professionals and researchers in efficiently summarizing complex legal documents. So, they can conclude their survey by outlining potential future research directions in the field of legal text summarization. By identifying the gaps and opportunities for improvement, they aim to inspire further advancements and innovations in this area. This paper serves as a comprehensive overview of text summarization techniques, with a particular emphasis on the challenges and advancements within legal text summarization. By presenting a broad perspective and highlighting potential avenues for future exploration, they hope to contribute to the continued progress in this important research area. In paper [8], the authors provide a comprehensive survey of recent accomplishments in the field of text summarization, with particular emphasis on the summarization of legal texts, considering its significant potential in the legal domain. The paper begins by offering a concise introduction to automatic text summarization, highlighting its importance and relevance in the era of information overload. It then proceeds to discuss the latest advancements in extractive and abstractive text summarization techniques. Extractive summarization involves selecting and condensing important sentences or phrases from the source text, while abstractive summarization aims to generate concise summaries by understanding the meaning and context of the original text and expressing it in a novel way. The survey delves into the various approaches and methodologies employed in these techniques, providing an overview of their strengths and limitations. The paper explores the specific challenges and considerations involved in the summarization of legal texts. Legal documents are known for their complexity and specialized language, making them particularly challenging to summarize accurately. The authors recognize the potential benefits of employing automatic summarization as a tool in the legal domain, where practitioners and researchers often need to quickly extract key information from lengthy legal documents. Also, the paper concludes by outlining potential future directions for research and development in text summarization. This includes exploring innovative techniques, improving the accuracy and effectiveness of existing algorithms, and addressing domain-specific challenges, such as the intricacies of legal language. This paper provides a valuable overview of recent achievements in text summarization, paying special attention to the application of these techniques in the legal field. It serves as a useful resource for researchers, practitioners, and professionals seeking to navigate the ever-growing landscape of automatic text summarization, while also highlighting promising avenues for future exploration and advancement in the field.

The paper [9] at hand delves into the description of a novel method designed to facilitate the summarization of legal documents, thereby aiding legal experts in distilling the key ideas encapsulated within a judgment. The proposed approach centers around a comprehensive exploration of the document's inherent architecture and thematic structures, which in turn enables the construction of a table-style summary, ultimately enhancing the coherency and readability of the text. To actualize this method, the authors introduce a system known as LetSum. The components of LetSum are elaborated upon, providing a clear understanding of its inner workings and the rationale behind its design. The implementation of the system is also outlined, shedding light on the practical aspects of its deployment. The paper offers preliminary evaluation results, shedding light on the effectiveness and potential impact of LetSum. Through rigorous analysis and assessment, the authors seek to gauge the system's performance and its ability to accurately identify and present the key concepts within legal documents. These preliminary findings provide an initial glimpse into the system's capabilities and lay the foundation for further investigations and refinements. The introduction of LetSum holds significant promise for legal professionals by streamlining the process of comprehending and extracting essential information from complex legal texts. By employing a table-style summary, the system presents a concise overview that encapsulates the critical elements of a judgment, facilitating informed decision-making and expediting the research process for legal experts. As the field of legal document summarization continues to evolve, the LetSum system represents a valuable contribution that combines an exploration of document architecture, thematic structures, and the effective presentation of key ideas. It is important to note, however, that additional comprehensive evaluations and refinements are necessary to fully validate the system's efficacy and ensure its applicability across a diverse range of legal contexts and document types. Nonetheless, the initial results and insights presented in this paper lay a solid foundation for further

advancements in the summarization of legal documents and the development of practical tools for legal professionals.

The exponential growth of legal information has fueled the demand for systems that can assist legal professionals and ordinary citizens in effortlessly accessing relevant legal information. The survey paper [1] aims to delve into various text summarization techniques, with a specific focus on the realm of legal document summarization—an area of utmost importance within the legal field that facilitates quick comprehension of complex legal documents. To establish a solid foundation for understanding legal text summarization techniques, the paper begins with a comprehensive introduction to text summarization itself. It provides essential background knowledge to pave the way for the subsequent exploration and discussion. The paper meticulously examines and evaluates different approaches and methodologies employed in summarizing legal text. It delves into the nuances of each technique, shedding light on their strengths, limitations, and potential applications. Furthermore, the paper elucidates various tools that are currently available for summarizing legal text, presenting an in-depth analysis of their functionalities and features. This analysis equips readers with valuable insights into the existing technological landscape. The paper presents two compelling case studies that showcase the automatic summarization of heterogeneous legal documents from two distinct countries. These real-world examples offer practical illustrations of the application of legal document summarization techniques and highlight their efficacy in capturing the essence of diverse legal content. By offering a detailed review of state-of-the-art approaches, conducting a comparative analysis based on the provided case studies, and engaging in thought-provoking discussions regarding important research questions, this work emerges as an invaluable resource for researchers venturing into the realm of legal document summarization. It not only serves as an enlightening starting point for further exploration but also identifies key future research directions that warrant in-depth investigation. This survey paper underscores the profound significance of legal text processing in effectively dealing with the vast amount of legal information available today. By examining various text summarization techniques, shedding light on available tools, and presenting insightful case studies, the paper sets the stage for future research endeavors and provides valuable guidance for researchers in the field of legal document summarization. It serves as a beacon, directing researchers towards innovative solutions and promoting advancements in this critical domain.

In another paper [10], The study begins by examining the introduction and evolution of different deep learning models that have been utilized in text summarization. By delving into these models, the researchers uncover the significance of various preparatory mechanisms within the realm of deep learning, which can greatly enhance the process of legal text summarization. Special attention is given to reviewing the existing deep learning models that have been recently employed in the legal domain for different types of summarization tasks. One particular approach that has gained considerable attention is the use of sequence-to-sequence neural network architectures. These models have demonstrated their effectiveness in capturing the essence of legal documents and generating accurate summaries. Furthermore, the authors explore the advancements achieved through transfer learning techniques, wherein pre-trained language models are fine-tuned to adapt to the nuances of legal language. This refinement process has led to significant improvements in the accuracy and quality of legal text summarization. By conducting this systematic comparison of deep learning strategies in legal text summarization, the authors aim to provide valuable insights into the most effective approaches for extracting concise summaries from complex legal documents. The findings of this research can contribute to the development of automated systems that assist legal professionals, researchers, and practitioners in efficiently navigating and comprehending large volumes of legal text. Nonetheless, further experimentation and evaluation are necessary to validate the efficacy and applicability of these strategies across diverse legal contexts and document types.

Manual summarization of large volumes of text, particularly in the legal domain, requires significant human effort and time. Lawyers invest substantial amounts of time in preparing comprehensive legal briefs based on their clients' case files. To address this challenge, the field of automatic text summarization has emerged within the realm of Natural Language Processing (NLP), a subdiscipline of Artificial Intelligence. This paper [11] introduces a hybrid approach to automatically summarize legal cases by leveraging the k-means clustering technique and tf-idf (term frequency-inverse document frequency) word vectorizer. The proposed method aims to generate concise summaries that capture the key information from the original text. To evaluate the effectiveness of the generated summaries, they are compared against the case summaries prepared by lawyers for court appeals using ROGUE evaluation parameters, which assess the quality of the generated summaries. The comparison between the automatically generated summaries and the lawyer-prepared case summaries provides insights into the performance and accuracy of the proposed method. By examining the evaluation results, the strengths and weaknesses of the hybrid approach can be identified, leading to suggestions for further improvement. Automated text summarization holds significant potential for streamlining the legal documentation process. By reducing the manual effort required for summarization, lawyers and legal professionals can save valuable time and allocate their resources more efficiently. The proposed hybrid method aims to bridge the gap between manual summarization and automatic techniques, offering a promising solution for extracting essential information from legal cases. However, it is important to note that the proposed method is not without limitations. Legal cases vary in complexity and context, and the effectiveness of the hybrid approach may differ across different types of cases. Therefore, the authors acknowledge the need for ongoing research and development to enhance the proposed method

and address any shortcomings. In conclusion, this paper presents a hybrid method for automatic text summarization of legal cases, combining the k-means clustering technique and tf-idf word vectorizer. Through a comparative evaluation with lawyer-prepared case summaries, the proposed method's performance is assessed using ROGUE evaluation parameters. The findings contribute to the advancement of automatic text summarization in the legal domain, with suggestions provided for further enhancement of the proposed method.

3. Workflow

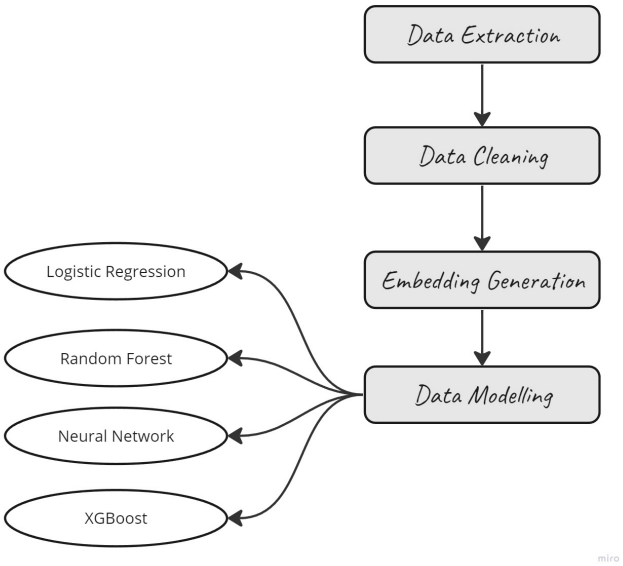


Figure 1: Workflow of our Legal Data Summarization project

Step 1: Data Extraction In this initial step, the project focuses on extracting legal data from various sources, such as legal documents, court cases, or online repositories.[12] This can involve web scraping techniques, accessing databases, or obtaining data through APIs. The goal is to gather a comprehensive dataset that contains the legal text required for the text summarization task.[13]

Step 2: Data Cleaning Once the legal data has been extracted, it often contains noise, irrelevant information, or formatting issues that could hinder the summarization process.[14] Data cleaning is crucial to ensure the quality and consistency of the dataset. [15] This step involves removing irrelevant sections or metadata, eliminating duplicate or redundant data, and standardizing the format of the text. Special characters, punctuation, and stopwords may also be removed to streamline the subsequent analysis.[16]

Step 3: Embedding Generation (Feature Engineering) Textual data cannot be directly used as input for most machine learning algorithms. In this step, the textual data is transformed into numerical representations that algorithms can understand.[17] This process, known as embedding generation or feature engineering, involves converting words or documents into dense vectors. Techniques such as Word2Vec, GloVe, or Doc2Vec are commonly employed to generate meaningful representations that capture the semantic and contextual relationships within the text. These embeddings capture important features of the legal text, allowing the models to learn patterns and make accurate predictions.[18]

Step 4: Data Modeling Data modeling is the core step of the text summarization project. It involves training and evaluating machine learning models to perform the summarization task. In this project, several models are considered: Logistic Regression, Random Forest, Neural Network, and XGBoost. Each model has its own strengths and characteristics. The models are trained using the embedded text data as input, and their objective is to predict or classify key sentences or generate summaries.[19]

The models are evaluated based on various metrics such as precision, recall, F1 score, or ROUGE scores to assess their performance in summarizing legal text accurately. The best-performing model(s) can be selected for deployment and further refinement.[20]

By following these steps, the project aims to extract legal data, clean and preprocess it, generate meaningful embeddings, and train models to summarize the legal text effectively. Each step plays a crucial role in the overall process, contributing to the development of a robust and accurate legal data text summarization system.

4. Data Description

4.1. Data Source

The legal judgments and their summaries used in our project were obtained from the Legal Information Institute of India (LIIofIndia). LIIofIndia operates as a not-for-profit publisher, offering free access to the content it provides for individual end-users. Users are permitted to read, print, and copy materials from LIIofIndia for personal use and other uses allowed by copyright law.

LIIofIndia has compiled a comprehensive collection of primary and secondary legal materials, which includes cases, legislation, treaties, journal articles, and law reform documents. These materials are obtained through agreements with various sources and rights-holders, as well as by other means such as scanning documents that are no longer under copyright or with the permission of copyright holders.

For our specific problem statement, the data we are interested in is available at the provided link and covers the period from 1970 to 1990. This dataset from LIIofIndia serves as the basis for our text summarization project in the legal domain.

4.2. Data Retrieval

The initial step in our project pipeline is to access and retrieve the necessary data for analysis and building machine learning models since no ready-made dataset is available. Here is an outline of the data extraction procedure:

Web Scraping: We used the Beautiful Soup library to extract data from the source website. The complete judgment

web pages spanning the years 1970 to 1990 were iterated and saved locally.[21] [22]

Filtering HTML Documents: From the collection of downloaded HTML documents, we focused on those with a standard judgment format. In this format, the summary section is typically labeled as "Head note," and the full text or judgment section is labeled as "Civil Appellate Jurisdiction." However, some HTML pages had spelling mistakes or followed a different format inconsistent with our standard format. We excluded such documents from our analysis.[23]

Extracting the Summary (Head Note): The head note or summary section was extracted by applying a filter using regular expressions. Specifically, we extracted the text between the phrases 'Head note:' (case insensitive) and 'Civil Appellate Jurisdiction' (case insensitive).[24]

Extracting the Full Text (Civil Appellate Jurisdiction): Similarly, the civil appellate jurisdiction or full text section was extracted using the same strategy. We captured the text between 'Civil Appellate Jurisdiction' (case insensitive) and 'LIIofIndia' (case insensitive).[25]

For a visual reference of the procedure, please consult the figure below.

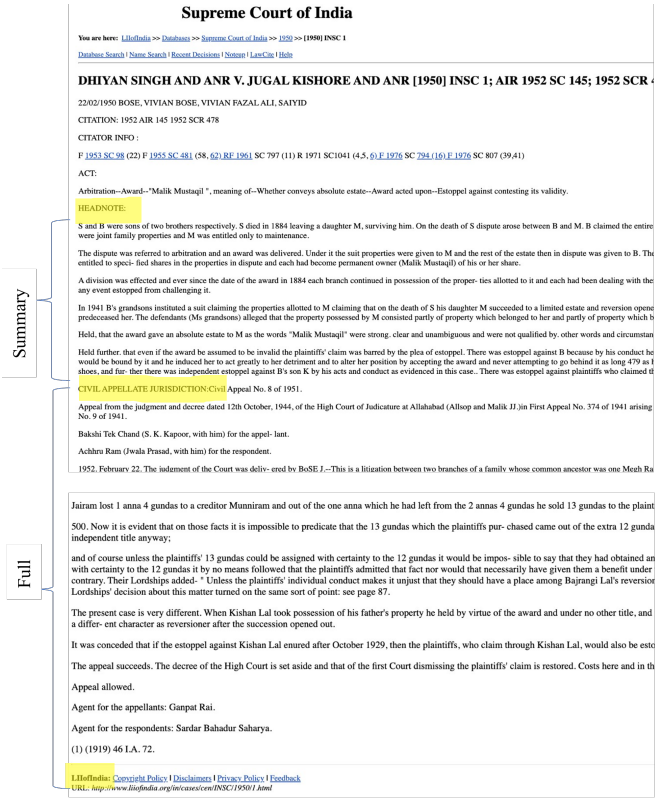


Figure 2: Representation of an html page to highlight extraction mechanism

4.3. Challenges

1. During the web scraping of the source website, there were instances where the server responded with a status code of 410, indicating that the requested resource is permanently deleted. However, the requested page could still be viewed on the website. To address this issue, a workaround was implemented by specifying the user-agent as a field in the request header. By including a user-agent identifier, such as 'PGDBAStudents', in the header while making the request to the server, it helped to mitigate the server's response of 410 and allowed successful retrieval of the desired web page.

2. The HTML pages from the source website did not have a proper Document Object Model (DOM) structure, which made it challenging to extract the required data using Beautiful Soup. Due to this irregular structure, the extraction process required significant effort. To overcome this problem, regular expressions were utilized to extract the necessary data. Multiple cycles of testing and adjustments were performed to ensure that the regular expressions accurately matched the desired content in various web pages.[26]

By employing regular expressions, specific patterns in the HTML pages were identified and targeted for data extraction. This approach allowed for the successful retrieval of the required information, despite the lack of a standardized DOM structure.

The combination of specifying the user-agent in the request header and leveraging regular expressions helped overcome the challenges faced during web scraping, enabling the extraction of the necessary data for further analysis and processing.

4.3.1. Data Insights

A few insights on the extracted corpus are as follows: 1. A total of 4917 legal judgements and their corresponding summaries were obtained. 2. Total Size of extracted data amounted to 187 MB in size 3. The developed data's structure is of format [Summary, Full, Doc], where Summary : Head Note of Judgement Data Full : Judgement Data or Civil Appellate Jurisdiction Doc : (year)$_{(document_n umber)}$.*html*

A sampled view of the yielded data is as follows:

	Summary	Full	col_len
0	the appellant terminated the service of a wor...	civil appeal no. 1914 of 1968. appeal by spec...	31013
1	the first respondent, in 1938, obtained a dec...	civil appeal no. 350 of 1970. appeal by speci...	20014
2	the assessee was a private limited company. i...	civil appeals nos. 2380 and 2381 of 1966. app...	13207
3	notices under s. 22(2) of the income-tax act,...	civil appeals nos. 23 5 and 236 of 1967. appe...	8822
4	d. b and j were partners in a firm which carr...	civil appeals nos. 1277 to 1279 and 1280 to 1...	13558

Figure 3: Snapshot of our structured dataframe

5. Data Preprocessing and Normalization

Once the data was extracted as a neat dataset, data preprocessing needed to be done. This stage involves text cleaning, and normalizing text to bring text components like words to some standard format. This enabled standardization across document corpus, which helps in building meaningful features and helps in reducing noise due to irrelevant symbols etc.[27]

Following are different preprocessing techniques followed.

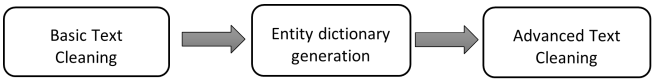


Figure 4: General Preprocessing Scheme.

5.1. Basic Text cleaning

Before implementing Named Entity Recognition (NER), some basic cleaning techniques were applied to the text. These techniques include:

1. Reducing Extra Spaces: In between the words to single space. This was done as spaces are being recognized as tokens during tokenization.[28]

For instance, “Indian Extradition Act” → “Indian Extradition Act”

2. Cleaning Short Hands And Expanding Contractions: By nature, short hands and contractions pose a problem for NLP and Text analytics and there should be a definite process for dealing these. So, a mapping for contractions and short hands and corresponding expansion was used for this type of cleaning.[29]

For example: ‘Respondent’s family members ‘re coming out’→ ‘Respondent family member are coming out

3. Replacing Special Characters: Some of the special characters are extensively used in formal or informal writings.[30]

By applying these cleaning techniques, the text is standardized and prepared for further processing, including Named Entity Recognition. These steps help ensure that the text is in a suitable format for accurate analysis and interpretation.

5.2. Implementation of Named Entity Recognition (NER)

Motivation Behind Implementing NER: In the document corpus, there were numerous unique proper nouns used in the same context, such as names of persons, months, nationalities, etc. If these proper nouns were not grouped under a single entity, each would contribute as a unique term in the vocabulary. This would result in an increased vocabulary size and would not effectively capture the essence and context of these words.[31]

To address this issue, Named Entity Recognition (NER) was employed. NER is a technique used in Natural Language Processing to identify and classify named entities in text. It allows for the identification of specific categories such as person names, locations, organizations, dates, and more.[32]

By applying NER, different proper nouns are tagged under the specific entity they belong to. This helps in grouping related terms and reducing the vocabulary size by treating them as single entities. It captures the semantic meaning and context of the proper nouns, allowing for better analysis and interpretation of the text.[33]

So, we can say that NER plays a crucial role in identifying and categorizing named entities, enabling the effective handling of proper nouns in the text corpus. It helps in reducing vocabulary size, capturing the essence of the words, and improving the overall understanding of the text.[34]

Entity Dictionary Generation: To determine the entity to which a proper noun belongs, the spaCy library, a free and open-source natural language processing library in Python, was utilized. The entire document corpus was traversed to create an "Entity Dictionary" that maps different proper nouns to their corresponding entities, such as person, number, date, act, etc.

The Entity Dictionary generated is a one-time activity performed on the corpus, and it saves computational time by predefining the mappings between proper nouns and their entities. The dictionary is created based on the analysis of the corpus and serves as a reference for entity recognition. For instance, Entity dictionary = "october":"date", "canadian":"NORP", "Jawahar":"Person"

Using this dictionary, the different proper nouns in the text corpus can be replaced with their respective entity labels. This allows for easier analysis and understanding of the text, as proper nouns are mapped to their corresponding entities.

By leveraging the Entity Dictionary, the recognition of proper nouns and their associated entities becomes more efficient and accurate, enhancing the overall NLP capabilities of the system.

Challenges faced during NER Implementation

1. Data Challenge: In some cases, proper nouns were represented in a shortened form in the summary compared to their full form in the full text. For example, "Brijlal" in the full text was abbreviated as "B" in the summary. The NER algorithm in SpaCy was unable to recognize "B" as a person entity. This discrepancy in representation posed a challenge for accurate entity recognition.

2. SpaCy NER Inefficiency: The SpaCy NER algorithm faced difficulties in recognizing and tagging Indian names of people, states, and organizations. Indian names such as "Ram" and "Jawahar," as well as Indian states like "Rajasthan," and organizations like "Tata Iron and Steel Company" were not properly tagged with their respective entity labels. This inefficiency of the NER algorithm affected the accuracy and completeness of entity recognition.

These challenges highlight the limitations and shortcomings of the NER implementation using SpaCy. Addressing these issues required additional efforts, such as adapting the algorithm to handle variations in representation and enhancing its ability to recognize specific types of entities, particularly in the context of Indian names and entities.

5.3. Advanced Text Cleaning

After implementing Named Entity Recognition (NER) and replacing proper nouns with their corresponding entities, further text cleaning techniques were applied to remove noise and retain the underlying meaning. The following techniques were employed:

1. Removing Stop Words and Punctuations: Stop words, which are commonly occurring words with little significance, were removed from the text to focus on words that carry more contextual meaning. Punctuation marks were also removed as they add little significance to the text.[35] For instance, 'Indian Extradition Act was passed which provided' → 'Indian Extradition Act passed provided'

2. Skip Continuous Entity String: After NER implementation, continuous entity strings like '24th October, 1994' may be replaced with multiple occurrences of the entity label ('date date date'). However, in such cases, the complete date should be treated as a single entity ('date') instead of multiple occurrences. Custom code was written to address this issue. [36]

For example, 24th October ,1994 → date date date → date (single)

3. Replacing Proper Nouns With Entity Tags:The entity dictionary generated during the NER implementation was used to replace proper nouns with their respective entity tags. This helped in preserving the entity information in the text. [37] [38]

For Instance, 'Indian Extradition Act passed provided' → 'Indian law passed provided'

4. Lemmatization: Lemmatization, using tools like spaCy, was applied to normalize text components such as words and pronouns. Lemmatization reduces different variants of words to their basic root form, called a lemma. This process helps to bring variations of words to a common form.[39]

For Instance, 'Indian law passed provided' → 'Indian law pass provide'

By applying these text cleaning techniques, the noise in the text was reduced, and the meaningful content was retained, making the text more suitable for further analysis and processing.

5.4. Final Result

Below is an example of text sentence in document corpus before text preprocessing and after preprocessing passing through Basic Cleaning, Implementing NER, Advanced Cleaning.

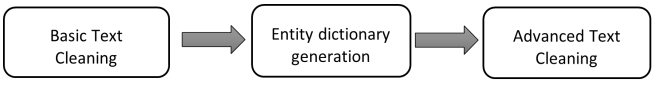


Figure 5:

5.5. Challenges Faced in Preprocessing

. Lack Of Computational Power Preprocessing on complete 5000 docs lead to run time crashing due to limited availability of RAM in local machines as well as Google Collab. Work Around: Dividing the complete data frame into multiple batches of size 30 and saving the preprocessed data frame iteratively for each batch into drive. This avoided loss of work even if Collab crashes and was able to resume on the remaining batches that need to be pre processed.

6. Feature Engineering

6.1. Normalization and PCA (for TF.iDF Sparse Matrix

During the project, normalization and dimensionality reduction techniques were applied to handle the challenges related to the dense matrix conversion of TF-IDF data and memory limitations. The following steps were followed:

1. Normalization:

a. To address the issue of classification models not being able to interpret sparse matrix data, the TF-IDF sparse matrix was normalized. b. The normalize function from the preprocessing package of scikit-learn was used to normalize the data along the columns. 2. Dimensionality Reduction using Latent Semantic Analysis (LSA):

a. Since the dense matrix conversion of some achieved TF-IDF data required a large amount of memory (174.67 GB), dimensionality reduction was performed using Latent Semantic Analysis (LSA). b. LSA is a technique in natural language processing that analyzes relationships between documents and the terms they contain by generating a set of concepts related to the documents and terms. c. LSA was achieved through dimensionality reduction, which helps in reducing the feature space while preserving the underlying structure and relationships.[40] 3. Centering the Data:

a. Prior to performing the dimensionality reduction operation, the data was centered. b. Centering the data ensures that the mean of each feature is zero, which can be important for some dimensionality reduction techniques. c. The normalize function from scikit-learn's preprocessing package was used to center the data along the column.[41] 4. SVD Transformation:

a. The features were generated by applying Singular Value Decomposition (SVD) transformation to the normalized and centered data. b. SVD is a matrix factorization technique that decomposes a matrix into three separate matrices: U, , and V to the power T. c. By selecting an optimal number of features (in this case, 500) based on memory capabilities and the goal of retaining the maximum number of features possible, the data was transformed using SVD. By performing normalization, dimensionality reduction using LSA, centering the data, and applying SVD transformation, we were able to generate a reduced set of features that could be used for further analysis and modeling while overcoming the memory limitations associated with dense matrix conversion. [42]

6.2. Cosine Similarity Generation (for all embedding approaches)

We have three different sentence embedding arrays post the developed embedding process, namely full text sentences embeddings, summary sentences embeddings, and gold standard sentences embeddings.[43] Assumption: Here, we assume that a given full text sentence is similar to only a single summary sentence. This assumption is important, because the summary which we're using is not an extractive summary, but an abstractive summary. So, for a given

document, we compute the cosine similarity between all possible pair of full text sentence to a summary sentence, using the developed schematic embeddings of the pair of sentences.[44] Mechanism: The data of each embedding array was first normalized to make each sentence vector a unit vector in magnitude. Normalization was achieved using SciKit Learn's normalize function under 'l2' constraint. For each document, the following operation was done to develop all possible pair of summary-full text sentence pair's cosine similarity.

Ci = SiTFi

Where, Si: Matrix containing normalized summary sentence embeddings for document i Fi: Matrix containing normalized full text sentence embeddings for document i Ci : Matrix with each element (<j, k>) representing cosine similarity if jth summary sentence and kth full text sentences for document i

Under the above assumption, we find the maximum cosine similarity of a given full text sentence to any summary sentence of the same document, and develop this for all the full text sentences.[45] A visualization of the mechanism is as follows:

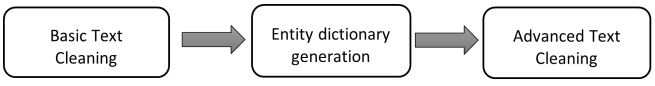


Figure 6: Step-by-step visual representation of generating importance of sentences using their embeddings

6.3. Embedding Generation

6.4. TF-IDF with Name and Entity Recognition

We used the TF-IDF vectorizer function from the feature extraction package of scikit-learn's utility tools to generate the TF-IDF embeddings of the sentences. This process resulted in a numerical matrix with 6265 features. The output of the TF-IDF vectorizer was in sparse matrix format, which is an efficient way to represent high-dimensional data with many zero entries.

To reduce the dimensionality of the TF-IDF embeddings, we applied PCA (Principal Component Analysis) to the matrix. PCA is a dimensionality reduction technique that transforms high-dimensional data into a lower-dimensional representation while preserving the most important information.

To measure the effectiveness of the dimensionality reduction process, we conducted an experiment on a sample of 50 documents. They generated the importance vector feature column using TF-IDF alone and another time using TF-IDF followed by PCA for dimensionality reduction. They then calculated the cosine similarity between the two resulting vectors and obtained a value of 0.77.

The high cosine similarity of 0.77 between the vectors indicated that the dimensionality reduction process using PCA was effective. It provided the us with confidence that the reduction in dimensionality did not result in significant information loss, and the resulting feature vectors still captured the essential characteristics of the original TF-IDF embeddings.

A brief summary of the developed corpus is as follows:

Number of Judgement Documents	**4,917**
Number of Sentence Segments	43448
Total number of sentences in the gist	13587
Proportion of Gist Segments	31.27%
Number of Gold Standard Sentences	6286
Dimension of Sentence Embeddings	100

Figure 7: Summary of the developed corpus

The distribution of the cosine similarity, thus achieved, is as follows:

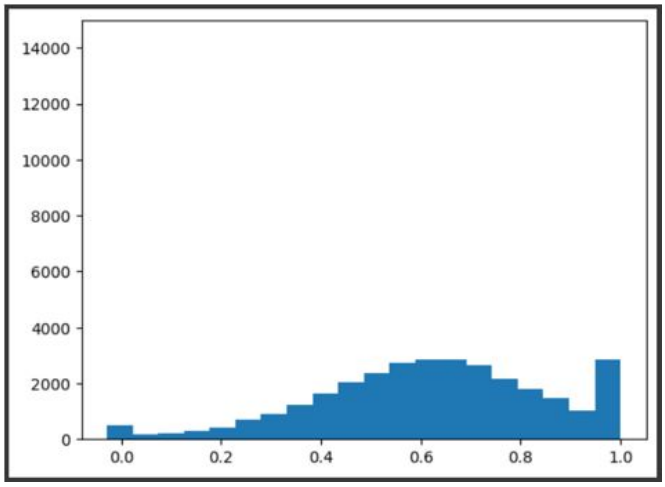


Figure 8: Cosine Similarity Distribution for the given model

Based on the above distribution, we intended to classify roughly 25 percent of the documents as important. So, a threshold of 0.95 was kept to classify a full text sentence as important or not. Importance of sentences with cosine similarity more than or equal to 0.95 was marked as 1, and the remaining sentences were marked 0. With this, we had prepared a dataset which had 29.68 percent important sentences and 70.31 percent not so important sentences. Note: the importance of sentence here is in context to that sentence being part of the summary sentence.

7. Model Fitting

The embedded features of the sentences, along with their corresponding importance classifications, were used as input features for sentence importance prediction. The data was divided into two sets: a training dataset and a validation dataset. The training dataset consisted of 70 percent of the prepared data, while the validation dataset comprised the remaining 30 percent. Machine learning models, including Logistic Regression, Random Forest Classifier, Extreme Gradient Boosting (XGB) Classifier, and simple Neural Networks, were utilized to incorporate statistical structures in the data for sentence classification. The models were developed using

the training dataset, which means they were trained on this data to learn patterns and relationships between the embedded features and their importance classifications. The goal was to create models that could accurately predict the importance of sentences based on their embedded features. The developed models' generalization ability and applicability were then tested using the validation dataset, which was unseen by the models during the training phase. This allowed the us to assess how well the models could generalize to new, unseen data and evaluate their performance on the task of sentence importance prediction.

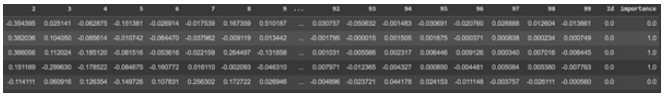


Figure 9: Embedding of a sentence and their corresponding importance

The whole prepared datasets were divided into two sets, training and validation dataset. Training dataset comprised of 70 percent of the data, and validation dataset the other 30 percent. The models were developed over training datasets, and the generalization ability and applicability of the developed models were tested using validation dataset, unseen to the model.

7.1. Evaluation Metric

Accuracy is not a good measure in our case. We have chosen F1 score as our metric. F1 score is harmonic mean of the recall and the precision.

F1 score = 2*(precision + recall)/precision + recall

It is also called the F Score or the F Measure. In another way, the F1 score conveys the balance between the precision and the recall. It heavily penalises a lower value of recall and/or precision. In order to have a significant F1 score, we need to have both the recall and the precision to be higher.

7.2. Classification Probability Threshold

The decision for converting a predicted probability or scoring into a class label is governed by a parameter referred to as the "decision threshold," "discrimination threshold," or simply the "threshold." The default value for the threshold is 0.5 for normalized predicted probabilities. The threshold was customized to yield maximum F1 score on validation results. A typical distribution of validation F1 score and the validation accuracy with varying threshold is as follows:

Here, x-axis represents the threshold value of probabilities, and the y-axis represents the values. The above curve was developed for Deep Neural Network Model for Word2Vec developed embeddings.

7.3. Models Developed

1. Logistic Regression: Logistic Regression was used as a statistical model to predict the binary class variable of sentence importance using the embedded features. The parameter space of the model was explored with different regularization techniques, namely L1 and L2 regularization, along with varying values of C.

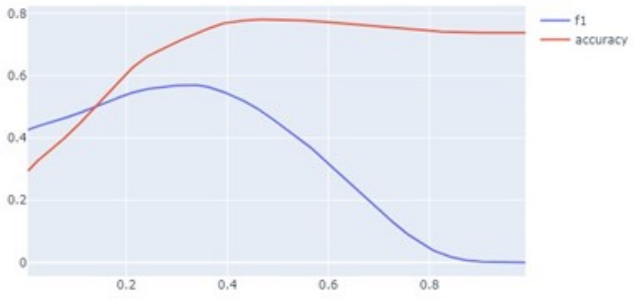


Figure 10: Plot of validation F1 score and accuracies with varying thresholds

Regularization helps prevent overfitting by adding a penalty term to the loss function, which controls the complexity of the model. L1 regularization promotes sparsity by shrinking some of the feature coefficients to zero, while L2 regularization shrinks the coefficients towards zero but does not necessarily make them exactly zero.

The goal was to find the combination of regularization technique and C value that maximized the F1-score on the validation dataset. F1-score is a metric that balances precision and recall, providing a measure of the model's overall performance.

Additionally, the classification probability threshold was tuned to find the optimal value that maximized the evaluation metric. The classification probability threshold determines the point at which the predicted probabilities are converted into class labels. By adjusting this threshold, the model's sensitivity and specificity can be modified, leading to different trade-offs between precision and recall.

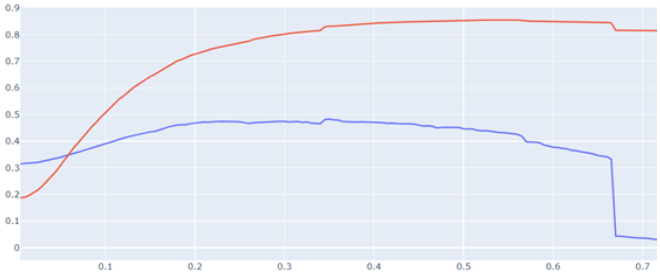


Figure 11: Plot of validation F1 score and accuracies with varying thresholds for logistic regression model

Accuracy	**0.8540453074433657**
F1-Score	0.4461727384363488
Precision	0.7654494382022472
Recall	0.3148469093009821

Figure 12: Model Results

2. Random Forest Classifier: The Random Forest Classifier was used to classify whether a sentence is part of

the summary or not. This classifier is an ensemble learning method that combines multiple decision trees to make predictions.

During the model training process, the parameter space of the Random Forest Classifier was explored by tuning different parameters. Some of the parameters that were tuned include the number of estimators, the minimum number of samples required to be at a leaf node ('minsamplesleaf'), and the minimum number of samples required to split an internal node ('minsamplessplit').

By varying these parameters over a range of values, the goal was to find the combination that maximized the F1-score on the validation dataset. The F1-score is a metric that balances precision and recall, providing an overall measure of the model's performance.

Additionally, similar to Logistic Regression, the classification probability threshold was tuned to find the optimal value that maximized the evaluation metric. By adjusting this threshold, the model's sensitivity and specificity can be modified, leading to different trade-offs between precision and recall.

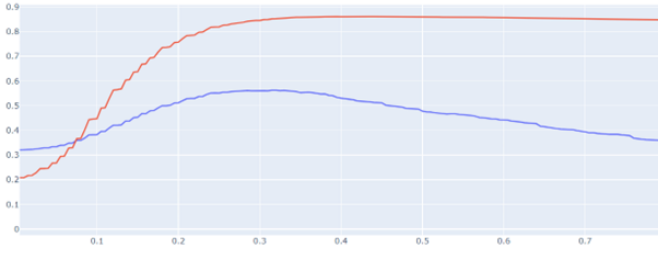


Figure 13: Plot of validation F1 score and accuracies with varying thresholds for logistic regression model

Accuracy	**0.8596548004314994**
F1-Score	0.47603705195328233
Precision	0.785904255319149
Recall	0.3414211438474878233

Figure 14: Model Results

3. XGB Classifier: The XGBoost (Extreme Gradient Boosting) Classifier was utilized to train the classification model for determining sentence importance. XGBoost is a gradient boosting algorithm that combines multiple weak learners (decision trees) to make predictions.

The objective function used for training the XGBoost Classifier was 'binary:logistic', which is appropriate for binary classification problems. The goal was to maximize the evaluation metric 'ROC AUC' (Receiver Operating Characteristic Area Under the Curve), which is a measure of the model's ability to distinguish between the two classes.

To optimize the hyperparameters of the XGBoost Classifier, Bayesian optimization was employed. Bayesian optimization involves building a probabilistic model of the objective function and using it to select the most promising hyperparameters to evaluate in the true objective function. This approach helps in efficiently exploring the hyperparameter space and finding the optimal combination of hyperparameters.

During the hyperparameter tuning process, 5-fold cross-validation was utilized to evaluate the performance of different hyperparameter configurations. This technique helps in estimating the model's performance on unseen data and reduces the risk of overfitting.

Finally, the trained XGBoost Classifier was evaluated on the test dataset using F1 scores, which provide a measure of the model's accuracy in predicting sentence importance.

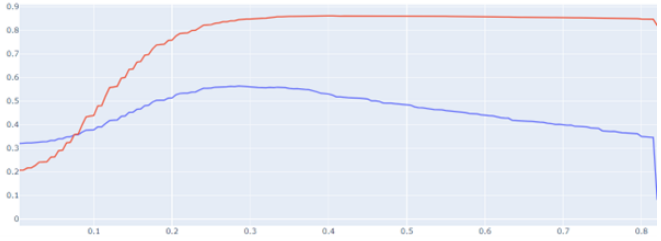


Figure 15: Plot of validation F1 score and accuracies with varying thresholds for XGB classifier

Accuracy	**0.8604099244875943**
F1-Score	0.48405103668261557
Precision	0.7812097812097812
Recall	0.3506643558636626

Figure 16: Plot of validation F1 score and accuracies with varying thresholds for XGB classifier

4. Neural Network: Deep neural networks were employed using TensorFlow 2.0 for the task of sentence importance classification. The architecture of the neural network, including the number of layers and hidden units, was determined through experimentation to maximize the F1 score on the validation dataset. The loss function used for training the neural network was binarycrossentropy, which is suitable for binary classification problems. The Adam optimizer was chosen as the optimization algorithm, and different combinations of learning rates and decay rates were explored to find the optimal configuration that maximized the evaluation metric. During the training process, the model was iteratively updated by backpropagating the gradients computed from the loss function through the network's layers. This process helps in adjusting the model's parameters to minimize the loss and improve its predictive performance. Similar to the previous models, the classification probability threshold was tuned to achieve the maximum value of the

evaluation metric. By adjusting the threshold, the trade-off between precision and recall can be optimized, depending on the specific requirements of the task. The neural network model was trained on the training dataset and evaluated on the validation dataset to assess its generalization ability. The F1 score was used as the evaluation metric to measure the model's accuracy in predicting sentence importance.

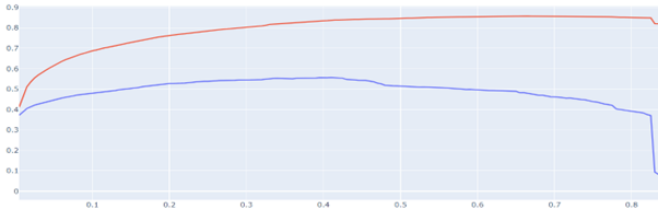


Figure 17: Model Results

Accuracy	**0.8487594390507012**
F1-Score	0.5094471658502449
Precision	0.6459627329192547
Recall	0.42056614673599074

Figure 18: Model Results

7.4. Challenges in Model fitting

- Resource Constraint: We had limited memory access, with very low RAM capacities. This hindered us to perform more accurate embeddings by getting more feature representations. - Computational Power: hindered us from exploring other advanced Neural Network architectures like LSTM Models, Bi-Directional RNN Models, and Attention Models due to lack of faster processing units like GPUs.

8. Results Discussion

The F1 scores corresponding to different models trained on the sentence embeddings is given below. The scores are calculated by fitting the trained model on validation dataset.

	Accuracy	F1 Score	Precision	Recall
Logistic Regression	0.8540	0.4461	0.7654	0.3148
Random Forest Classifier	0.8596	0.4760	0.7859	0.3414
Deep Neural Networks	0.8487	0.5094	0.6459	0.4205
XGB Classifier	0.8604	0.4840	0.7812	0.3506

Figure 19: TF-IDF with NER

Based on the F1 scores of the validation dataset, it is observed that the Deep Neural Network (DNN) model outperformed the other models, including Logistic Regression, Random Forest, and XGBOOST. The DNN model achieved higher F1 scores across all four types of models trained.

The higher F1 scores indicate that the DNN model had better precision and recall, resulting in a more balanced performance in classifying sentence importance. It suggests that the DNN model was able to capture more complex patterns and relationships in the data, leading to improved predictive accuracy.

It is important to note that the choice of the best model depends on the specific requirements and characteristics of the task at hand. While the DNN model showed superior performance in this case, other models may still be suitable for different scenarios, depending on factors such as interpretability, computational efficiency, and the size and nature of the dataset.

Overall, based on the F1 scores obtained in this evaluation, the Deep Neural Network model demonstrated the highest performance among the classifiers trained.

9. Future Scope

1. We have achieved good accuracy and F1-score using machine learning method for the purpose of extractive summarization. We can extend our study to abstractive summarization of Indian legal documents. 2. We can use advanced models like BERT, LSTM to classify sentences whether it is an important one or not. 3. We can further improve our results by implementing ensemble and stacking to the classifiers that we have already implemented.

10. ANNEXURE

10.1. Data Extraction

```
import requests
import urllib
import pandas as pd
import os
from bs4 import BeautifulSoup
import re
---
def saveashtml(s,file_name):
    with open(file_name,'w') as f:
        f.write(s)
        f.close()
---
headers = {'User-Agent': 'whatever'}

start_year = 1988
end_year = 2000
url = "http://www.liiofindia.org/in/cases/cen/INSC/"
path = "newfolder/"

for curr_year in range(start_year,end_year+1,1):
    flag = True
    i = 1
    while flag :
        url_new = url + str(curr_year) + "/" + str(i) + ".html"
        file_name = path + str(curr_year) + "_" + str(i) +
             ".html"
        result = requests.get(url_new,headers = headers)
        if result.status_code == int(404):
            flag = False
```

```
            i = 1
        else:
            saveashtml(result.text,file_name)
            i = i+1
---
# Extract to data frame from HTML pages
def extract2df(soup,filename):
    head_str = ""
    civil_str = ""

    full_flag = False
    summary_flag = False

    # Extract for Summary
    sum_start = "HEADNOTE:".lower()
    sum_end = "CIVIL APPELLATE JURISDICTION:".lower()

    # Extract for Full text
    full_start = sum_end
    full_end = "LIIofIndia:".lower()

    p_list = soup.find_all('p')

    for elem in p_list:
        elem = " ".join(elem.text.lower().split())
        if not summary_flag:
            head = re.search(sum_start + "(.*)" + sum_end ,
                 elem )
            if head != None:
                head_str = head.group(1)
                summary_flag = True

        if summary_flag & (not full_flag):
            civil = re.search(full_start + "(.*)" + full_end ,
                 elem )
            if civil != None:
                civil_str = civil.group(1)
                full_flag = True
                break;
    if full_flag:
        df_obj =
             pd.DataFrame({"Summary":[head_str],"Full":[civil_str],"Doc":[filename]})
    else:
        return []
    return df_obj
---
df = pd.DataFrame()

directory =
     "/Users/prati/Downloads/Compressed/Extractive_Summartization-master/Necessary
     Codes/newfolder/"

i = 0
for filename in os.listdir(directory):
    i = i + 1
    if i modulous 500 == 0:
% - modulous
        print(i)
    if filename.endswith(".html"):
        fname = os.path.join(directory,filename)
        f = open(fname)
        soup = BeautifulSoup(f,'html.parser')
        df_obj = extract2df(soup,filename)
        if len(df_obj) == 0:
            f.close()
            continue
        df = df.append(df_obj)
        f.close()
df_train = df.reset_index().drop(["index"],axis =1)
print(len(df))
print(len(df_train))
---
df_train.to_pickle("/Users/prati/Downloads/Compressed/Extractive_Summarizati
     Codes/pklfile/data.pkl")
```

10.2. Data Cleaning

```
import csv
import requests
import xml.etree.ElementTree as ET
import numpy as np
import pandas as pd
import nltk
from nltk.tokenize import sent_tokenize,word_tokenize
from nltk.text import Text
---
from google.colab import drive
drive.mount('/content/drive')
---
temp = pd.read_pickle("/content/drive/MyDrive/Colab
     Notebooks/Necessary Codes/pklfile/data.pkl")
temp.shape
---
temp["col_len"] = temp.apply(lambda row:len(row.Full),axis =
     1,result_type = "expand")
temp.drop(["Doc"],axis = 1,inplace = True)
temp
---
import spacy
from spacy.lang.en import English
nlp = English()
nlp = spacy.load('en_core_web_sm')
nlp.max_length = 21787990
# nlp.add_pipe(nlp.create_pipe('sentencizer'))

nltk.download('stopwords')
nltk.download('punkt')
is_remove_stopwords = True
if is_remove_stopwords:
    from nltk.corpus import stopwords
    stopwords = set(stopwords.words('english'))
---
ENTITY_ENUM = {
    '': '',
    'PERSON': 'person',
    'NORP': 'nationality',
    'FAC': 'facility',
    'ORG': 'organization',
    'GPE': 'country',
    'LOC': 'location',
    'PRODUCT': 'product',
    'EVENT': 'event',
    'WORK_OF_ART': 'artwork',
```

```
    'LANGUAGE': 'language',
    'DATE': 'date',
    'TIME': 'time',
    'PERCENT': 'number',
    'MONEY': 'number',
    'QUANTITY': 'number',
    'ORDINAL': 'number',
    'CARDINAL': 'number',
    'LAW': 'law'
}

NUMERIC_TYPES = set([
    'DATE',
    'TIME',
    'PERCENT',
    'MONEY',
    'QUANTITY',
    'ORDINAL',
    'CARDINAL',
])
---
import re

def clean_string(text):
    text = str(text)
  # Replace weird chars in text
    text = re.sub(" ", "'", text) # special single quote
    text = re.sub("`", "'", text) # special single quote
    text = re.sub(" ", '"', text) # special double quote
    text = re.sub(" ", "?", text)
    text = re.sub(" ", " ", text)
    text = re.sub("", "e", text)

    # Clean shorthands
    text = re.sub("\'s", " ", text) # we have cases like "Sam
         is" or "Sam's" (i.e. his) these two cases aren't
         separable, I choose to compromise are kill "'s"
         directly
    text = re.sub(" whats ", " what is ", text,
         flags=re.IGNORECASE)
    text = re.sub("\'ve", " have ", text)
    text = re.sub("can't", "can not", text)
    text = re.sub("n't", " not ", text)
    text = re.sub("i'm", "i am", text, flags=re.IGNORECASE)
    text = re.sub("\'re", " are ", text)
    text = re.sub("\'d", " would ", text)
    text = re.sub("\'ll", " will ", text)
    text = re.sub("e\.g\.", " eg ", text, flags=re.IGNORECASE)
    text = re.sub("b\.g\.", " bg ", text, flags=re.IGNORECASE)
    text = re.sub(r"(W|^)([0-9]+)[kK](W|$)", r"\1\g<2>000\3",
         text) # better regex provided by @armamut
    text = re.sub("e-mail", " email ", text,
         flags=re.IGNORECASE)
    text = re.sub("(the[\s]+|The[\s]+)?U\.S\.A\.", " America
         ", text, flags=re.IGNORECASE)
    text = re.sub("(the[\s]+|The[\s]+)?United State(s)?", "
         America ", text, flags=re.IGNORECASE)
    text = re.sub("\(s\)", " ", text, flags=re.IGNORECASE)
    text = re.sub("[c-fC-F]\:\/", " disk ", text)

    # replace the float numbers with a random number, it will
         be parsed as number afterward, and also been
         replaced with word "number"

    text = re.sub('[0-9]+\.[0-9]+', " 87 ", text)

    # remove comma between numbers, i.e. 15,000 -> 15000

    text = re.sub('(?<=[0-9])\,(?=[0-9])', "", text)
    text = re.sub('\%', " percent ", text)
    text = re.sub('\&', " and ", text)

    # indian rupees
    text = re.sub("(?<=[0-9])rs ", " rs ", text,
         flags=re.IGNORECASE)
    text = re.sub(" rs(?=[0-9])", " rs ", text,
         flags=re.IGNORECASE)

    # the single 's' in this stage is 99% of not clean text,
         just kill it
    text = re.sub(' s ', " ", text)

    # reduce extra spaces into single spaces
    text = re.sub('[\s]+', " ", text)
    text = text.strip()

    return text

---
def vote_dictionary_generation(doc,vote_dict):

  # construct vote dictionary
  for token in doc:
    if token.lower_ not in vote_dict:
      vote_dict[token.lower_] = {}
    if token.ent_type_ not in vote_dict[token.lower_]:
      vote_dict[token.lower_][token.ent_type_] = 0
      # if the token has_vector is True, maybe we shouldn't
           record its
      vote_dict[token.lower_][token.ent_type_] += 1 # TODO:
           not sure if storing in lowercase form is safe ?

def
     entity_lookup_generation(vote_dict,word_ent_type_dict,word_ent_type_sec
  threshold_of_second_type = 3
  # vote for what should the type be
  for key in vote_dict:
    # non-type has lower priority
    if '' in vote_dict[key]:
      vote_dict[key][''] = vote_dict[key][''] - 0.1

    ents = list(vote_dict[key].keys())
    bi_list = [
             ents,
             [vote_dict[key][ent] for ent in ents]
             ]
    # if several ent_type_ have same count, just let it go,
         making them share same ent_type_ is enough
    # TODO: if have time, can design a better metric to deal
         with second graded type
    sorted_idx = np.argsort(bi_list[1])
    if sorted_idx.shape[0]>1:
```

```
      best_idx = sorted_idx[-1]
      second_idx = sorted_idx[-2]
      word_ent_type_dict[key] = bi_list[0][best_idx]
      if bi_list[1][second_idx]>threshold_of_second_type:
        word_ent_type_second_dict[key] = bi_list[0][second_idx]
    else:
      best_idx = sorted_idx[-1]
      word_ent_type_dict[key] = bi_list[0][best_idx]
---
## This function is used in our case - Checks entity type of
     a token
def token_type_lookup(token, report_detail=False):

    if type(token)==str:
        token = nlp(token)[0]

    key = token.lower_

    try:
        if report_detail:
            print(ENTITY_ENUM[word_ent_type_dict[key]], ' <=
                 ', {ENTITY_ENUM[ent_t] :
                 vote_dict[key][ent_t] for ent_t in
                 vote_dict[key]} )

        return word_ent_type_dict[key]

    except KeyError:
        return ''

def is_token_has_second_type(token):

    if type(token)==str:
        token = nlp(token)[0]

    key = token.lower_

    try:
        return key in word_ent_type_second_dict
    except KeyError:
        return False

def token_second_type_lookup(token, report_detail=False):

    if type(token)==str:
        token = nlp(token)[0]

    key = token.lower_

    try:
        if report_detail:
            print(ENTITY_ENUM[word_ent_type_second_dict[key]],
                 ' <= ', {ENTITY_ENUM[ent_t] :
                 vote_dict[key][ent_t] for ent_t in
                 vote_dict[key]} )

        return word_ent_type_second_dict[key]
    except KeyError:
        return ''
---
exception_list = set(['need']) # spaCy identifies need's
     lamma as 'ne', which is not we want

numeric_types = set(['DATE', 'TIME', 'PERCENT', 'MONEY',
     'QUANTITY', 'ORDINAL', 'CARDINAL'])

def process_text_with_spacy(spacy_obj, debug=False,
     show_fail=False, idx=None):

    def not_alpha_or_digit(token):
        ch = token.text[0]
        return not (ch.isalpha() or ch.isdigit())

    result_word_list = []
    res = ''


    previous_ent_type = None

    is_a_word_parsed_fail = False
    fail_words = []

    for token in spacy_obj:

        global_ent_type = token_type_lookup(token)


        if token.text in exception_list:

            previous_ent_type = None
            result_word_list.append(token.text)

        # skip none words tokens
        elif not_alpha_or_digit(token) or token.text==' ' or
             token.text=='s':
            previous_ent_type = None
            if debug: print(token.text, ' : remove punc or
                 special chars')


        # if the "remove stop word" flag is set to True
        elif is_remove_stopwords and token.lemma_ in stopwords:
            previous_ent_type = None
            if debug: print(token.text, ' : remove stop word')


        # contiguous same type, skip it
        elif global_ent_type==previous_ent_type or
             token.ent_type_==previous_ent_type:
            if debug: print('contiguous same type')
        elif global_ent_type in NUMERIC_TYPES and
             previous_ent_type in NUMERIC_TYPES:
            if debug: print('contiguous numeric')
        elif token.ent_type_ in NUMERIC_TYPES and
             previous_ent_type in NUMERIC_TYPES:
            if debug: print('contiguous numeric')


        # number without an ent_type_
        elif token.text.isdigit():

            if debug: print(token.text, 'force to be number')

            if previous_ent_type in NUMERIC_TYPES:
                pass
```

```
        else:
            previous_ent_type = 'CARDINAL' # any number
                 type would be okay
            result_word_list.append('number')

elif global_ent_type!='':

        result_word_list.append(ENTITY_ENUM[global_ent_type])
        previous_ent_type = global_ent_type
        if debug: print(token.text, ' : sub ent_type:',
             ENTITY_ENUM[global_ent_type])

       elif token.lower_==token.lemma_ and
            token.text[1:]!=token.lemma_[1:] and
            is_token_has_second_type(token):
        second_type = token_second_type_lookup(token)
        result_word_list.append(ENTITY_ENUM[second_type])
        if debug: print(token.text, ' : use second
             ent_type:', ENTITY_ENUM[second_type])
        previous_ent_type = second_type


    # words arrive here are either "extremely common" or
         "extremely rare and has no method to deal with"
    else:
        # A weird behavior of SpaCy, it substitutes [I,
             my, they] into '-PRON-'
        if token.lemma_=='-PRON-':
            result_word_list.append(token.lower_)
            res = token.lower
            previous_ent_type = None

        # the lemma can be identified by GloVe
        elif nlp(token.lemma_)[0].has_vector:
            result_word_list.append(token.lemma_)
            res = token.lemma_
            previous_ent_type = None

        # the lemma cannot be identified, very probably a
             proper noun
        elif is_token_has_second_type(token):
            second_type = token_second_type_lookup(token)
            result_word_list.append(ENTITY_ENUM[second_type])
            res = ENTITY_ENUM[second_type]
            previous_ent_type = second_type
            if debug: print(token.text, ' : use second
                 ent_type in else :',
                 ENTITY_ENUM[second_type])

        # the lemma is not in glove and Spacy can't
             identify if it is a proper noun, last try,
        #     if the word itself can be identified by
             GloVe or not
        elif nlp(token.lower_)[0].has_vector:
            result_word_list.append(token.lower_)
            res = token.lower_
            previous_ent_type = None
            if debug: print(token.text, ' : the token
                 itself can be identified :', token.lower_)
        elif token.has_vector:
            result_word_list.append(token.text)
            res = token.text
            previous_ent_type = None
            if debug: print(token.text, ' : the token
                 itself can be identified :', token.text)

        # Damn, I have totally no idea what's going on
        # You got to deal with it by yourself
        # In my case, I use fasttext to deal with it
        else:
            is_a_word_parsed_fail = True
            fail_words.append(token.text)
            previous_ent_type = None

            if debug: print(token.text, ' : can\'t
                 identify, replace with "something"')


    if show_fail and is_a_word_parsed_fail:
        if idx!=None:
            print('At qid=', idx)
        print('Fail words: ', fail_words)
        print('Before:', spacy_obj.text)
        print('After: ', ' '.join(result_word_list))
        print('============================================================

    return np.array(result_word_list)
---
len(temp)
---
## Generates vote dictionary which is used for generating
     Entity dictionary
import json
import time

# STEP 2: fill batch number according to your convinience

batch_size = 30

# STEP 3: batch wise processing
# batch no to be filled inside the range value
# tabulate the batch number according to your allocation of
     document to be processed
for i in range(0, 10):
  vote_dict = {}
  word_ent_type_dict = {}
  word_ent_type_second_dict = {}
  max_length = 0

  start = i * batch_size
  end = start + batch_size - 1
  start_time = time.time()
  print("Start Target: ", start)
  print("End Target : ", end)

  for i,row in temp.iterrows():
    if(i>=start and i <= end):
      print("vote dict, doc no: ", i, "of length: ",
           row['col_len'])
      for col in temp.columns:
        doc = nlp(clean_string(row[col]).lower())
        vote_dictionary_generation(doc, vote_dict)
```

```
entity_lookup_generation(vote_dict, word_ent_type_dict,
     word_ent_type_second_dict)

# STEP 4: change the path accordingly
# would suggest not to change the naming convention
with open('/content/drive/MyDrive/Colab Notebooks/Necessary
     Codes/newdictionary/word_ent_type_dict_' + str(start)
     + '_to_' + str(end) +'.json', 'w') as fp1:
    json.dump(word_ent_type_dict, fp1)

end_time = time.time()
print("nword_ent_type_dict Write Successful in time : ",
     (end_time - start_time)/60 , " min")

# try:
df = pd.DataFrame()

start_time = time.time()
for i,row in temp.iterrows():
  summary = []
  full = []
  ori_full = []

  # For Summary
  if(i>=start and i <= end):
     print("summary, doc no: ", i, "of length: ",
          len((row["Summary"])[:-110]))
     for sent in
          sent_tokenize(clean_string((row["Summary"])[:-110]).lower()):
        doc = nlp(sent)
        s = process_text_with_spacy(doc) # Advanced Text
             Cleaning
        res = " ".join(s)
        summary.append(res)

  # For Full text
  if(i>=start and i <= end):
     print("full, doc no : ", i, "of length: ",
          len((row["Full"])[56:-468]))
     for sent in
          sent_tokenize(clean_string((row["Full"])[56:-468]).lower()):
        doc = nlp(sent)
        s = process_text_with_spacy(doc)
        res = " ".join(s)
        full.append(res)
        ori_full.append(sent)

  sum_join = "*".join(summary)
  full_join = "*".join(full)
  ori_full_join = "*".join(ori_full)

  df_obj = pd.DataFrame({"summary":[sum_join],
       "full":[full_join], "full_orignal":[ori_full_join],
       "Id":[i]})
  df = pd.concat([df,df_obj])

# STEP 5: change path to the destination folder accordingly
# would suggest not to change the naming convention
(df.iloc[start:end+1]).to_pickle('/content/drive/MyDrive/Colab
     Notebooks/Necessary
     Codes/newdataframe/dataframe_processed_' + str(start)
     + '_to_' + str(end) +'.pickle')

  end_time = time.time()
  print("nDataframe Write to Disk Successful in time: ",
       (end_time - start_time)/60 , " min")
print(vote_dict)
---
vote_dict
---
try:
  df = pd.DataFrame()
  for i,row in temp.iterrows():
    if i % 500 == 0:
      print(i)
    summary = []
    full = []
    # For Summary
    for sent in
         sent_tokenize(clean_string(row["Summary"]).lower()):
         # Clean_string - Basic Text Cleaning
        summary.append(sent)
    # For Full text
    for sent in
         sent_tokenize(clean_string(row["Full"]).lower()):
        full.append(sent)

    sum_join = "*".join(summary)
    full_join = "*".join(full)

    df_obj =
         pd.DataFrame({"Summary":[sum_join],"Full":[full_join],"Id":[i]})
    df = pd.concat([df,df_obj])
except Exception as e:
  print("saving df of length",len(df))
  print(e)
---
print(df)
df.to_csv("/content/drive/MyDrive/Colab Notebooks/Necessary
     Codes/data.csv")
---
```

10.3. Embedding Maker

```
from google.colab import drive
drive.mount('/content/drive')
---
import os
import numpy as np
import pandas as pd

path = "/content/drive/MyDrive/Colab Notebooks/Necessary
     Codes/newdataframe/"

data = pd.DataFrame()

for i in os.listdir(path):
    df = pd.read_pickle(path+i)
    df['temp_col_len'] = df.full.str.len()
    print(i)
    data = pd.concat([data,
         df[df['temp_col_len']>1].drop(columns =
         ['temp_col_len'])], axis = 0, ignore_index=True)
```

```
data
---
GS_index = (np.array(data.index))[-50:]
GS_index
---
# Sentence tokenization
# and tagging of document number t sentences

temp = data.copy()

set = temp.index
# set = np.arange(0, 51, 1)
# set = [0, 1, 2]

# temp = data

full_sent = pd.DataFrame()
summ_sent = pd.DataFrame()

full_docs = []
summ_docs = []
GS_docs = []

ori_sent = pd.DataFrame()
GS_sent = pd.DataFrame()
GS_ori_sent = pd.DataFrame()

for i in set:
  if i not in GS_index:
    full = temp['full'].iloc[i]
    summary = temp['summary'].iloc[i]
    ori = temp['full_orignal'].iloc[i]

#     print(full)
#     break
    v = full.split("*")
    t1 = pd.DataFrame(v)

    v = summary.split('*')
    t2 = pd.DataFrame(v)

    v = ori.split('*')
    t5 = pd.DataFrame(v)

    t3 = np.linspace(i, i, num = len(t1))
    t4 = np.linspace(i, i, num = len(t2))

    # print(t1)

    if(len(t1)==len(t5)):
        full_docs = np.append(full_docs, t3)
        summ_docs = np.append(summ_docs, t4)
        full_sent = pd.concat([full_sent, t1],
             ignore_index=True)
        summ_sent = pd.concat([summ_sent, t2],
             ignore_index=True)
        ori_sent = pd.concat([ori_sent, t5], ignore_index=True)
    else:
        print("Doc No. : ", i,"has a problem")

  else:
    full = temp['full'].iloc[i]
    v = full.split("*")
    t1 = pd.DataFrame(v)

    ori = temp['full_orignal'].iloc[i]
    v = ori.split('*')
    t5 = pd.DataFrame(v)

    t3 = np.linspace(i, i, num = len(t1))

    if(len(t1)==len(t5)):
        GS_docs = np.append(GS_docs, t3)
        GS_sent = pd.concat([GS_sent, t1], ignore_index=True)
        GS_ori_sent = pd.concat([GS_ori_sent, t5],
             ignore_index=True)
    else:
      print("GS Doc No. : ", i-1060,"has a problem")

print('summary sentence tokenization')
print(len(summ_docs))
print(len(summ_sent))
print(len(full_docs))
print(len(full_sent))
print(len(ori_sent))

print(len(GS_docs))
print(len(GS_sent))
print(len(GS_ori_sent))

# summ_sent_mod = pd.DataFrame(summ_sent,
     columns=['Summary_Sentences'])
summ_sent_mod = summ_sent.set_axis(['Sentences'], axis=1,
     inplace=False)
# summ_sent_mod

full_sent_mod = full_sent.set_axis(['Sentences'], axis=1,
     inplace=False)
# full_sent_mod
---
# combining summary and full-text sentences for tfidf
     vectorization of sentences

#
     ------------------------------------------------------------------------

GS_sent_mod = GS_sent.set_axis(['Sentences'], axis=1,
     inplace=False)

tmp1 = pd.DataFrame(summ_sent_mod)
tmp2 = pd.DataFrame(full_sent_mod)
tmp3 = pd.DataFrame(GS_sent_mod)

combined = pd.concat([tmp1, tmp2], axis = 0,
     ignore_index=True, sort = False)
combined = pd.concat([combined, tmp3], axis = 0,
     ignore_index=True, sort = False)
combined.columns = ['sents']

#
     ------------------------------------------------------------------------

print(combined.shape)
---
```

```
GS_sent_mod = GS_sent.set_axis(['Sentences'], axis=1,
    inplace=False)
tmp3 = pd.DataFrame(GS_sent_mod)
tmp3.columns = ['Sents']
tmp3['doc_id'] = GS_docs
tmp3.shape
tmp3.to_pickle("/content/drive/MyDrive/Colab
    Notebooks/Necessary Codes/nerdata/gs_sent.pickle")
---
from sklearn.feature_extraction.text import TfidfVectorizer
    as tuttu
from nltk.tokenize import RegexpTokenizer
import scipy

#
    ------------------------------------------------------------

token = RegexpTokenizer(r'[a-zA-Z0-9]+')
tfidf_func = tuttu(ngram_range = (1, 1), tokenizer =
    token.tokenize, lowercase = True)
tfidf = tfidf_func.fit_transform(combined['sents'])

#
    ------------------------------------------------------------

tfidf.shape
---
# dimension reduction
from sklearn.decomposition import PCA, TruncatedSVD

svd = TruncatedSVD(100)
a = svd.fit_transform(tfidf[:,:])

del tfidf

print(a.shape)
print(a)
---
# ## Function to reduce the DF size
def reduce_mem_usage(df, verbose=True):
    numerics = ['int16', 'int32', 'int64', 'float16',
        'float32', 'float64']
    start_mem = df.memory_usage().sum() / 1024**2
    for col in df.columns:
        col_type = df[col].dtypes
        if col_type in numerics:
            c_min = df[col].min()
            c_max = df[col].max()
            if c_min > np.finfo(np.float16).min and c_max <
                np.finfo(np.float16).max:
                df[col] = df[col].astype(np.float16)
            elif c_min > np.finfo(np.float32).min and c_max <
                np.finfo(np.float32).max:
                df[col] = df[col].astype(np.float32)
            else:
                df[col] = df[col].astype(np.float64)
        if(col%20==0):
            print("column no : ", col, " done!")
    end_mem = df.memory_usage().sum() / 1024**2
    if verbose: print('Mem. usage decreased to {:5.2f} Mb
        ({:.1f}% reduction)'.format(end_mem, 100 *
        (start_mem - end_mem) / start_mem))
    return df

x2 = pd.DataFrame(a)
x2 = reduce_mem_usage(x2)
---
x2 = pd.DataFrame(a)
print(x2.shape)

summ_tfidf = x2.iloc[:len(summ_sent_mod)]
full_tfidf =
    x2.iloc[len(summ_sent_mod):len(summ_sent_mod)+len(full_sent_mod)]
gs_tfidf = x2.iloc[len(summ_sent_mod)+len(full_sent_mod):]

print(len(summ_docs))
print(len(full_docs))
print(len(GS_docs))
------------------------------------------------------------

summ_tfidf['Id'] = summ_docs
full_tfidf['Id'] = full_docs
gs_tfidf['Id'] = GS_docs

full_tfidf
---
from sklearn import preprocessing
------------------------------

#
    ------------------------------------------------------------

summ_tfidf_normalized =
    preprocessing.normalize(summ_tfidf.drop(['Id'], axis=1),
    norm='l2')
summ_tfidf_normalized = pd.DataFrame(summ_tfidf_normalized)
summ_tfidf_normalized.columns = summ_tfidf.drop(['Id'],
    axis=1).columns
summ_tfidf_normalized['Id'] = np.array(summ_tfidf.Id)

#
    ------------------------------------------------------------

full_tfidf_normalized =
    preprocessing.normalize(full_tfidf.drop(['Id'], axis=1),
    norm='l2')
full_tfidf_normalized = pd.DataFrame(full_tfidf_normalized)
full_tfidf_normalized.columns = full_tfidf.drop(['Id'],
    axis=1).columns
full_tfidf_normalized['Id'] = np.array(full_tfidf.Id)

#
    ------------------------------------------------------------
gs_tfidf_normalized =
    preprocessing.normalize(gs_tfidf.drop(['Id'], axis=1),
    norm='l2')
gs_tfidf_normalized = pd.DataFrame(gs_tfidf_normalized)
gs_tfidf_normalized.columns = gs_tfidf.drop(['Id'],
    axis=1).columns
gs_tfidf_normalized['Id'] = np.array(gs_tfidf.Id)


summ_tfidf_normalized
---
A = [[2,3],[5,4]]
preprocessing.normalize(A, norm='l2')
```

```
---
cosine_vec = []

#
     ------------------------------------------------------------------

for i in set:
  tmp_full_sent =
       summ_tfidf_normalized[summ_tfidf_normalized.Id==i].drop(['Id'],
       axis=1)
  tmp_summ_sent =
       full_tfidf_normalized[full_tfidf_normalized.Id==i].drop(['Id'],
       axis=1)

  M = tmp_summ_sent.dot(np.transpose(tmp_full_sent))
  cos_doc = (np.amax(M, axis = 1))
  cosine_vec = np.append(cosine_vec, cos_doc)

#
     ------------------------------------------------------------------

len(cosine_vec)
---
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

#
     ------------------------------------------------------------------

plt.hist(cosine_vec, bins=20)
plt.ylim(0, 15000)
plt.show()
---
imp = np.copy(cosine_vec)

#
     ------------------------------------------------------------------

for i in range(len(imp)):
  if(imp[i]>=0.83):
    imp[i] = 1
  else:
    imp[i] = 0

#
     ------------------------------------------------------------------

full_tfidf_normalized['importance'] = imp
full_tfidf_normalized['importance'].value_counts()
---
full_tfidf_normalized.to_pickle("/content/drive/MyDrive/Colab
     Notebooks/Necessary
     Codes/nerdata/tfidf_pca_result_final_500.pickle")
gs_tfidf_normalized.to_pickle("/content/drive/MyDrive/Colab
     Notebooks/Necessary
     Codes/nerdata/GS_tfidf_pca_result_final_500.pickle")
---
```

10.4. Modelling - Logistic Regression

```
import pandas as pd
import numpy as np
import pickle
import gc
import warnings
     ------------------------------------------------------------------
warnings.filterwarnings('ignore')
from scipy.stats import uniform
---
from google.colab import drive
drive.mount('/content/drive')
---
data = pickle.load(open("/content/drive/MyDrive/Colab
     Notebooks/Necessary
     Codes/nerdata/tfidf_pca_result_final_500.pickle", "rb"))
X_pred = (pd.read_pickle("/content/drive/MyDrive/Colab
     Notebooks/Necessary
     Codes/nerdata/GS_tfidf_pca_result_final_500.pickle")).drop(columns
     = ['Id'])

gc.collect()
     ------------------------------------------------------------------
---
from sklearn.linear_model import Lasso, LogisticRegression
from sklearn.model_selection import train_test_split
                                    #to split the dataset for
     training and testing
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score

target = data.importance
     ------------------------------------------------------------------
data.drop(['importance', 'Id'], axis=1, inplace=True) #
     taking the training data features

# del data

X_train, X_test, y_train, y_test = train_test_split(data,
     target, test_size = 0.3, random_state = 32) # in this
     our main data is split into train and test
     ------------------------------------------------------------------
     ------------------------------------------------------------------
del data, target
gc.collect()
---
from sklearn.model_selection import RandomizedSearchCV
import time

model = LogisticRegression(max_iter=10000000)

# Create regularization penalty space
     ------------------------------------------------------------------
penalty = ['l1', 'l2']

# Create regularization hyperparameter distribution using
     uniform distribution
C = uniform(loc=0, scale=4)

# Create hyperparameter options
hyperparameters = dict(C=C, penalty=penalty)

# Create randomized search 5-fold cross validation and 100
     iterations
clf = RandomizedSearchCV(model, hyperparameters,
     random_state=42, n_iter=30, cv=3, verbose=1, n_jobs=-1,
     scoring = 'f1')
```

```
s = time.time()
best_model = clf.fit(X_train, y_train)
e = time.time()

# View best hyperparameters
print('Best Penalty:',
     best_model.best_estimator_.get_params()['penalty'])
print('Best C:', best_model.best_estimator_.get_params()['C'])
print("Model Operation Completed in :",(e-s)/60,"min")
---
from sklearn.metrics import precision_score, recall_score
model = LogisticRegression(max_iter=10000000, penalty = 'l2',
     C = 2.8322903)

import time

s = time.time()
model.fit(X_train, y_train)
e = time.time()

print("Model trained in", (e-s)/60,"min")
print(model.score(X_test, y_test))

pred = model.predict(X_test)
print(f1_score(pred, y_test))
---
cutoff = np.arange(0.005, 0.600, 0.005)
f1 = []

pred = (model.predict_proba(X_test))[:,1]

for i in cutoff:
    f1.append(f1_score(pred >= i, y_test))

# print(f1)

import plotly.graph_objects as go

fig = go.Figure(data = go.Scatter(x = cutoff, y = f1))
fig.show()
---
max_cutoff = cutoff[f1 == max(f1)]
max_cutoff
---
cutoff = np.arange(0.005, 0.995, 0.005)
f1 = []
acc = []

pred = (model.predict_proba(X_test))[:,1]

for i in cutoff:
    f1.append(f1_score(pred >= i, y_test))
    acc.append(accuracy_score(pred >= i, y_test))

# print(f1)

import plotly.graph_objects as go

fig = go.Figure()
fig.add_trace(go.Scatter(x = cutoff, y = f1, mode='lines',
     name='f1'))
fig.add_trace(go.Scatter(x = cutoff, y = acc, mode='lines',
     name='accuracy'))
fig.show()
---
from sklearn.metrics import precision_score, recall_score
start_time = time.time()

# Calculate accuracy
accuracy = model.score(X_test, y_test)
print("Accuracy:", accuracy)

# Calculate precision and recall
pred = model.predict(X_test)
precision = precision_score(y_test,pred)
recall = recall_score(y_test,pred)

print("Precision:", precision)
print("Recall:", recall)

stop_time = time.time()
print("Elapsed Time:", time.strftime("%H:%M:%S",
     time.gmtime(stop_time - start_time)))
---
GS_sent = pd.read_pickle("/content/drive/MyDrive/Colab
     Notebooks/Necessary Codes/nerdata/gs_sent.pickle")
GS_sent.columns = ["Sents", "doc_id"]
predictions = model.predict(X_pred)
probability_of_predictions = model.predict_proba(X_pred)

# this command gives the sentences that are detected
     important on Gold Standard Data
result = pd.DataFrame()

result = (GS_sent.iloc[predictions == 1])
result['probabilities'] =
     probability_of_predictions[predictions == 1][:,1]
result['doc_id'] = result['doc_id'] - 2153

# length filter of output results
length = []
for i in np.array(result['Sents']):
  length.append(len(i)>=50)

result['Sents'] = (result.iloc[length])['Sents']
result.dropna(inplace = True)

result
---
```

10.5. Random Forest

```
import pandas as pd
import numpy as np
import pickle
import gc
import warnings
import time
warnings.filterwarnings('ignore')
import json
from sklearn.model_selection import RandomizedSearchCV
---
```

```
data = pickle.load(open("/content/drive/MyDrive/Colab
     Notebooks/Necessary
     Codes/nerdata/tfidf_pca_result_final_500.pickle", "rb"))
X_pred = (pd.read_pickle("/content/drive/MyDrive/Colab
     Notebooks/Necessary
     Codes/nerdata/GS_tfidf_pca_result_final_500.pickle"))

gc.collect()
---
from sklearn.linear_model import Lasso, LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
                                #to split the dataset for
     training and testing
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score

target = data.importance
data.drop(['importance', 'Id'], axis=1, inplace=True) #
     taking the training data features

# del data

X_train, X_test, y_train, y_test = train_test_split(data,
     target, test_size = 0.3, random_state = 32) # in this
     our main data is split into train and test

del data, target
gc.collect()
---
model = RandomForestClassifier()

params_dist = {
    "n_estimators":[100],
    "min_samples_leaf": np.arange(1,60,10),
    "max_leaf_nodes": np.arange(2,60,10)
}

start_time = time.time()

optimal_model = RandomizedSearchCV(model,params_dist,scoring
     = "f1",n_jobs = -1)
y_pred = optimal_model.fit(X_train,y_train).predict(X_test)

end_time = time.time()

print("Elapsed Time:", time.strftime("%H:%M:%S",
     time.gmtime(end_time - start_time)))
---
score = f1_score(y_test,y_pred)

print("====================")
print("f1 score: {}".format(score))
print("Best Score: {}".format(optimal_model.best_score_))
print("Best Parameters:
     {}".format(optimal_model.best_params_))
---
start_time = time.time()

model = RandomForestClassifier()

model.fit(X_train, y_train)

print(model.score(X_test, y_test))

stop_time = time.time()
print("Elapsed Time:", time.strftime("%H:%M:%S",
     time.gmtime(stop_time - start_time)))

pred = model.predict(X_test)
print(f1_score(pred, y_test))
---
cutoff = np.arange(0.005, 0.995, 0.005)
f1 = []
acc = []

pred = (model.predict_proba(X_test))[:,1]

for i in cutoff:
    f1.append(f1_score(pred >= i, y_test))
    acc.append(accuracy_score(pred >= i, y_test))

# print(f1)

import plotly.graph_objects as go

fig = go.Figure()
fig.add_trace(go.Scatter(x = cutoff, y = f1, mode='lines',
     name='f1'))
fig.add_trace(go.Scatter(x = cutoff, y = acc, mode='lines',
     name='accuracy'))
fig.show()
---
from sklearn.metrics import precision_score, recall_score
start_time = time.time()

# Calculate accuracy
accuracy = model.score(X_test, y_test)
print("Accuracy:", accuracy)

# Calculate precision and recall
pred = model.predict(X_test)
precision = precision_score(y_test,pred)
recall = recall_score(y_test,pred)

print("Precision:", precision)
print("Recall:", recall)

stop_time = time.time()
print("Elapsed Time:", time.strftime("%H:%M:%S",
     time.gmtime(stop_time - start_time)))
---
GS_sent = pd.read_pickle("/content/drive/MyDrive/Colab
     Notebooks/Necessary Codes/nerdata/gs_sent.pickle")
GS_sent.columns = ["Sents", "doc_id"]
predictions = model.predict(X_pred)
probability_of_predictions = model.predict_proba(X_pred)

# this command gives the sentences that are detected
     important on Gold Standard Data
result = pd.DataFrame()

result = (GS_sent.iloc[predictions == 1])
result['probabilities'] =
     probability_of_predictions[predictions == 1][:,1]
```

```
result['doc_id'] = result['doc_id'] - 2153

# length filter of output results
length = []
for i in np.array(result['Sents']):
  length.append(len(i)>=50)

result['Sents'] = (result.iloc[length])['Sents']
result.dropna(inplace = True)

result
---
```

10.6. Neural Network

```
import pandas as pd
import numpy as np
import pickle
import gc
import warnings
warnings.filterwarnings('ignore')
---
data = pickle.load(open("/content/drive/MyDrive/Colab
     Notebooks/Necessary
     Codes/nerdata/tfidf_pca_result_final_500.pickle", "rb"))
X_pred = (pd.read_pickle("/content/drive/MyDrive/Colab
     Notebooks/Necessary
     Codes/nerdata/GS_tfidf_pca_result_final_500.pickle"))

gc.collect()
---
from sklearn.linear_model import Lasso, LogisticRegression
from sklearn.model_selection import train_test_split
                                #to split the dataset for
     training and testing
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score

target = data.importance
data.drop(['importance','Id'], axis=1, inplace=True) # taking
     the training data features

# del data

X_train, X_test, y_train, y_test = train_test_split(data,
     target, test_size = 0.3, random_state = 32) # in this
     our main data is split into train and test

del data, target
gc.collect()
---
!pip install tensorflow
---
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout,
     Activation, Flatten



x_train = tf.keras.utils.normalize(X_train, axis=1)
x_test = tf.keras.utils.normalize(X_test, axis=1)
print("Normalization Complete!")

model = tf.keras.models.Sequential()
# model.add(tf.keras.layers.Flatten()) # input layer
model.add(tf.keras.layers.Dense(512, activation=tf.nn.relu))
     # hidden layer 1
model.add(Dropout(0.4))
model.add(tf.keras.layers.Dense(512, activation=tf.nn.relu))
     # hidden layer 1
model.add(Dropout(0.4))
model.add(tf.keras.layers.Dense(256, activation=tf.nn.relu))
     # hidden layer 1
model.add(Dropout(0.4))
model.add(tf.keras.layers.Dense(1, activation=tf.nn.sigmoid))
     # output layer

model.compile(optimizer='adam', loss= 'binary_crossentropy',
     metrics=['accuracy'])
print("Model Ready")

model.fit(x_train, np.array(y_train), epochs=14,
     validation_data = (x_test, np.array(y_test)))
---
cutoff = np.arange(0.005, 0.995, 0.005)
f1 = []
acc = []

pred = (model.predict(X_test))

for i in cutoff:
    f1.append(f1_score(pred >= i, y_test))
    acc.append(accuracy_score(pred >= i, y_test))

# print(f1)

import plotly.graph_objects as go

fig = go.Figure()
fig.add_trace(go.Scatter(x = cutoff, y = f1, mode='lines',
     name='f1'))
fig.add_trace(go.Scatter(x = cutoff, y = acc, mode='lines',
     name='accuracy'))
fig.show()
---
cutoff = np.arange(0.01, 0.99, 0.01)
f1 = []

pred = (model.predict(X_test))

for i in cutoff:
    f1.append(f1_score(pred >= i, y_test))

# print(f1)

import plotly.graph_objects as go

fig = go.Figure(data = go.Scatter(x = cutoff, y = f1))
fig.show()

max_cutoff = cutoff[f1 == max(f1)]
print(max_cutoff)
```

```
---
import numpy as np
from sklearn.metrics import accuracy_score, precision_score,
     recall_score, f1_score

# Assuming you have the predicted labels and true labels
y_pred = model.predict(x_test)
y_pred = (y_pred > 0.5).astype(int) # Convert probabilities
     to binary predictions

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
---
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, f1_score

# Assuming you have the predicted labels and true labels
y_pred = model.predict(x_test)
y_pred = (y_pred > 0.5).astype(int) # Convert probabilities
     to binary predictions

accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Print the metrics
print("Accuracy:", accuracy)
print("F1 Score:", f1)

# Plotting the graph
x = ['Accuracy', 'F1 Score']
y = [accuracy, f1]

plt.bar(x, y)
plt.ylim(0, 1) # Set y-axis limits between 0 and 1
plt.ylabel('Score')
plt.title('Model Performance')
plt.show()
---
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, f1_score

# Assuming you have the predicted labels and true labels
y_pred = model.predict(x_test)
y_pred = (y_pred > 0.5).astype(int) # Convert probabilities
     to binary predictions

accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Print the metrics
print("Accuracy:", accuracy)
print("F1 Score:", f1)

# Plotting the graph
x = np.arange(2) # x-coordinates for the data points
labels = ['Accuracy', 'F1 Score']
y = [accuracy, f1]

plt.plot(x, y, marker='o', linestyle='-', color='b')
plt.xticks(x, labels)
plt.ylim(0, 1) # Set y-axis limits between 0 and 1
plt.ylabel('Score')
plt.title('Model Performance')
plt.show()
---
GS_sent = pd.read_pickle("/content/drive/MyDrive/Colab
     Notebooks/Necessary Codes/nerdata/gs_sent.pickle")
GS_sent.columns = ["Sents", "doc_id"]
predictions = model.predict(X_pred)
probability_of_predictions = model.predict(X_pred)

# this command gives the sentences that are detected
     important on Gold Standard Data
result = pd.DataFrame()

result = (GS_sent.iloc[predictions >= max_cutoff])
result['probabilities'] =
     probability_of_predictions[predictions >= max_cutoff]
result['doc_id'] = result['doc_id'] - 4946

# length filter of output results
length = []
for i in np.array(result['Sents']):
  length.append(len(i)>=50)

result['Sents'] = (result.iloc[length])['Sents']
result.dropna(inplace = True)


result
---
```

10.7. XGBoost

```
import pandas as pd
import numpy as np
import pickle
import gc
import warnings
import time
warnings.filterwarnings('ignore')
import json
from sklearn.model_selection import RandomizedSearchCV
---
data = pickle.load(open("/content/drive/MyDrive/Colab
     Notebooks/Necessary
     Codes/nerdata/tfidf_pca_result_final_500.pickle", "rb"))
X_pred = (pd.read_pickle("/content/drive/MyDrive/Colab
     Notebooks/Necessary
     Codes/nerdata/GS_tfidf_pca_result_final_500.pickle"))

gc.collect()
---
from sklearn.linear_model import Lasso, LogisticRegression
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
                                #to split the dataset for
    training and testing
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score

target = data.importance
data.drop(['importance', 'Id'], axis=1, inplace=True) #
    taking the training data features

# del data

X_train, X_test, y_train, y_test = train_test_split(data,
    target, test_size = 0.3, random_state = 32) # in this
    our main data is split into train and test

del data, target
gc.collect()
---



model = RandomForestClassifier()

params_dist = {
    "n_estimators":[100],
    "min_samples_leaf": np.arange(1,60,10),
    "max_leaf_nodes": np.arange(2,60,10)
}

start_time = time.time()

optimal_model = RandomizedSearchCV(model,params_dist,scoring
    = "f1",n_jobs = -1)
y_pred = optimal_model.fit(X_train,y_train).predict(X_test)

end_time = time.time()

print("Elapsed Time:", time.strftime("%H:%M:%S",
    time.gmtime(end_time - start_time)))
---
score = f1_score(y_test,y_pred)

print("====================")
print("f1 score: {}".format(score))
print("Best Score: {}".format(optimal_model.best_score_))
print("Best Parameters:
    {}".format(optimal_model.best_params_))
---
# model = LogisticRegression(max_iter=10000000, C = 1)

start_time = time.time()

model = RandomForestClassifier()

model.fit(X_train, y_train)
print(model.score(X_test, y_test))

stop_time = time.time()
print("Elapsed Time:", time.strftime("%H:%M:%S",
    time.gmtime(stop_time - start_time)))
```

```
pred = model.predict(X_test)
print(f1_score(pred, y_test))
---
cutoff = np.arange(0.005, 0.995, 0.005)
f1 = []
acc = []

pred = (model.predict_proba(X_test))[:,1]

for i in cutoff:
    f1.append(f1_score(pred >= i, y_test))
    acc.append(accuracy_score(pred >= i, y_test))

# print(f1)

import plotly.graph_objects as go

fig = go.Figure()
fig.add_trace(go.Scatter(x = cutoff, y = f1, mode='lines',
    name='f1'))
fig.add_trace(go.Scatter(x = cutoff, y = acc, mode='lines',
    name='accuracy'))
fig.show()
---
from sklearn.metrics import precision_score, recall_score
start_time = time.time()

# Calculate accuracy
accuracy = model.score(X_test, y_test)
print("Accuracy:", accuracy)

# Calculate precision and recall
pred = model.predict(X_test)
precision = precision_score(y_test,pred)
recall = recall_score(y_test,pred)

print("Precision:", precision)
print("Recall:", recall)

stop_time = time.time()
print("Elapsed Time:", time.strftime("%H:%M:%S",
    time.gmtime(stop_time - start_time)))
---
GS_sent = pd.read_pickle("/content/drive/MyDrive/Colab
    Notebooks/Necessary Codes/nerdata/gs_sent.pickle")
GS_sent.columns = ["Sents", "doc_id"]
predictions = model.predict(X_pred)
probability_of_predictions = model.predict_proba(X_pred)

# this command gives the sentences that are detected
    important on Gold Standard Data
result = pd.DataFrame()

result = (GS_sent.iloc[predictions == 1])
result['probabilities'] =
    probability_of_predictions[predictions == 1][:,1]
result['doc_id'] = result['doc_id'] - 2153

# length filter of output results
length = []
for i in np.array(result['Sents']):
  length.append(len(i)>=50)
```

```
result['Sents'] = (result.iloc[length])['Sents']
result.dropna(inplace = True)

result
---
```

References

[1] D. Jain, M. D. Borah, and A. Biswas, "Summarization of legal documents: Where are we now and the way forward," *Computer Science Review*, vol. 40, p. 100388, 2021.

[2] V. Parikh, V. Mathur, P. Mehta, N. Mittal, and P. Majumder, "Lawsum: A weakly supervised approach for indian legal document summarization," *arXiv preprint arXiv:2110.01188*, 2021.

[3] P. Bhattacharya, K. Hiware, S. Rajgaria, N. Pochhi, K. Ghosh, and S. Ghosh, "A comparative study of summarization algorithms applied to legal case judgments," in *Advances in Information Retrieval: 41st European Conference on IR Research, ECIR 2019, Cologne, Germany, April 14–18, 2019, Proceedings, Part I 41*, pp. 413–428, Springer, 2019.

[4] D. Anand and R. Wagh, "Effective deep learning approaches for summarization of legal texts," *Journal of King Saud University-Computer and Information Sciences*, vol. 34, no. 5, pp. 2141–2150, 2022.

[5] P. Jeatrakul and K. W. Wong, "Comparing the performance of different neural networks for binary classification problems," in *2009 Eighth International Symposium on Natural Language Processing*, pp. 111–115, IEEE, 2009.

[6] C. Bentéjac, A. Csörgő, and G. Martínez-Muñoz, "A comparative analysis of gradient boosting algorithms," *Artificial Intelligence Review*, vol. 54, pp. 1937–1967, 2021.

[7] A. Kanapala, S. Pal, and R. Pamula, "Text summarization from legal documents: a survey," *Artificial Intelligence Review*, vol. 51, pp. 371–402, 2019.

[8] V. Gupta, N. Bansal, and A. Sharma, "Text summarization for big data: A comprehensive survey," in *International Conference on Innovative Computing and Communications: Proceedings of ICICC 2018, Volume 2*, pp. 503–516, Springer, 2019.

[9] A. Farzindar and G. Lapalme, "Legal text summarization by exploration of the thematic structure and argumentative roles," in *Text Summarization Branches Out*, pp. 27–34, 2004.

[10] R. Sheik and S. J. Nirmala, "Deep learning techniques for legal text summarization," in *2021 IEEE 8th Uttar Pradesh Section International Conference on Electrical, Electronics and Computer Engineering (UPCON)*, pp. 1–5, IEEE, 2021.

[11] V. Pandya, "Automatic text summarization of legal cases: A hybrid approach," *arXiv preprint arXiv:1908.09119*, 2019.

[12] V. Crescenzi, G. Mecca, P. Merialdo, *et al.*, "Roadrunner: Towards automatic data extraction from large web sites," in *VLDB*, vol. 1, pp. 109–118, 2001.

[13] A. H. Laender, B. A. Ribeiro-Neto, A. S. Da Silva, and J. S. Teixeira, "A brief survey of web data extraction tools," *ACM Sigmod Record*, vol. 31, no. 2, pp. 84–93, 2002.

[14] E. De Jonge and M. Van Der Loo, *An introduction to data cleaning with R*. Statistics Netherlands Heerlen, 2013.

[15] M. Volkovs, F. Chiang, J. Szlichta, and R. J. Miller, "Continuous data cleaning," in *2014 IEEE 30th international conference on data engineering*, pp. 244–255, IEEE, 2014.

[16] J. Van den Broeck, S. Argeseanu Cunningham, R. Eeckels, and K. Herbst, "Data cleaning: detecting, diagnosing, and editing data abnormalities," *PLoS medicine*, vol. 2, no. 10, p. e267, 2005.

[17] C. R. Turner, A. Fuggetta, L. Lavazza, and A. L. Wolf, "A conceptual basis for feature engineering," *Journal of Systems and Software*, vol. 49, no. 1, pp. 3–15, 1999.

[18] F. Nargesian, H. Samulowitz, U. Khurana, E. B. Khalil, and D. S. Turaga, "Learning feature engineering for classification.," in *Ijcai*, vol. 17, pp. 2529–2535, 2017.

[19] G. Simsion and G. Witt, *Data modeling essentials*. Elsevier, 2004.

[20] D. J. Livingstone, D. T. Manallack, and I. V. Tetko, "Data modelling with neural networks: Advantages and limitations," *Journal of computer-aided molecular design*, vol. 11, pp. 135–142, 1997.

[21] J. M. Patel and J. M. Patel, "Web scraping in python using beautiful soup library," *Getting Structured Data from the Internet: Running Web Crawlers/Scrapers on a Big Data Production Scale*, pp. 31–84, 2020.

[22] V. Krotov and L. Silva, "Legality and ethics of web scraping," 2018.

[23] S. V. Broucke and B. Baesens, *Practical Web Scraping for Data Science: best practices and examples with Python*. CreateSpace, 2017.

[24] R. Mitchell, *Web scraping with Python: Collecting more data from the modern web*. " O'Reilly Media, Inc.", 2018.

[25] J. Hammer, H. Garcia-Molina, J. Cho, R. Aranha, and A. Crespo, "Extracting semistructured information from the web," in *Proceedings of the workshop on management of semistructured data*, vol. 10, Tucson, Arizona: ACM, 1997.

[26] D. M. Thomas and S. Mathur, "Data analysis by web scraping using python," in *2019 3rd International conference on Electronics, Communication and Aerospace Technology (ICECA)*, pp. 450–454, 2019.

[27] S. Xu, Y. Qian, and R. Q. Hu, "A data-driven preprocessing scheme on anomaly detection in big data applications," in *2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 814–819, IEEE, 2017.

[28] J. Brownlee, *Data preparation for machine learning: data cleaning, feature selection, and data transforms in Python*. Machine Learning Mastery, 2020.

[29] H. Müller, S. Castelo, M. Qazi, and J. Freire, "From papers to practice: the openclean open-source data cleaning library," *Proceedings of the VLDB Endowment*, vol. 14, no. 12, pp. 2763–2766, 2021.

[30] K. Nongthombam and D. Sharma, "" data analysis using python," *International Journal of Engineering Research & Technology (IJERT)*, 2021.

[31] M. A. Tanenblatt, A. Coden, and I. L. Sominsky, "The conceptmapper approach to named entity recognition.," in *LREC*, pp. 546–51, 2010.

[32] R. Alfred, L. C. Leong, C. K. On, and P. Anthony, "Malay named entity recognition based on rule-based approach," 2014.

[33] W. Gunawan, D. Suhartono, F. Purnomo, and A. Ongko, "Named-entity recognition for indonesian language using bidirectional lstm-cnns," *Procedia Computer Science*, vol. 135, pp. 425–432, 2018.

[34] Z. Jie, P. Xie, W. Lu, R. Ding, and L. Li, "Better modeling of incomplete annotations for named entity recognition," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 729–734, 2019.

[35] Z. Pabarskaite, "Implementing advanced cleaning and end-user interpretability technologies in web log mining," in *ITI 2002. Proceedings of the 24th International Conference on Information Technology Interfaces (IEEE Cat. No. 02EX534)*, pp. 109–113, IEEE, 2002.

[36] R. Bose, "Advanced analytics: opportunities and challenges," *Industrial Management & Data Systems*, 2009.

[37] E. Lee and P. Kim, "Identifying text reuse using word net-based extended named entity recognition," in *Proceedings of the 2018 Conference on Research in Adaptive and Convergent Systems*, pp. 199–202, 2018.

[38] H. Isozaki and H. Kazawa, "Efficient support vector classifiers for named entity recognition," in *COLING 2002: The 19th International Conference on Computational Linguistics*, 2002.

[39] J. Plisson, N. Lavrac, D. Mladenic, *et al.*, "A rule based approach to word lemmatization," in *Proceedings of IS*, vol. 3, pp. 83–86, 2004.

[40] M. Moussa and I. I. Măndoiu, "Single cell rna-seq data clustering using tf-idf based methods," *BMC genomics*, vol. 19, pp. 31–45, 2018.

[41] I. Veritawati, I. Wasito, *et al.*, "Sparse data for document clustering," in *2013 International Conference of Information and Communication*

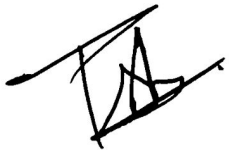
Technology (ICoICT), pp. 38–43, IEEE, 2013.
[42] W. L. Buntine and A. Jakulin, "Applying discrete pca in data analysis," *arXiv preprint arXiv:1207.4125*, 2012.
[43] X. Zhu, S. Su, M. Fu, J. Liu, L. Zhu, W. Yang, G. Jing, and Y. Guo, "A cosine similarity algorithm method for fast and accurate monitoring of dynamic droplet generation processes," *Scientific reports*, vol. 8, no. 1, pp. 1–14, 2018.
[44] V. Thada and V. Jaglan, "Comparison of jaccard, dice, cosine similarity coefficient to find best fitness value for web retrieved documents using genetic algorithm," *International Journal of Innovations in Engineering and Technology*, vol. 2, no. 4, pp. 202–205, 2013.
[45] S. Pal, M. Chang, and M. F. Iriarte, "Summary generation using natural language processing techniques and cosine similarity," in *Intelligent Systems Design and Applications: 21st International Conference on Intelligent Systems Design and Applications (ISDA 2021) Held During December 13–15, 2021*, pp. 508–517, Springer, 2022.

11. IPR Certificate

We, Chirag Kumar and Prateek Sharma, with this certify that the project work submitted by us entitled Legal Data Summarizer to our supervisors, Dr. Alok Kumar, in partial fulfillment of the requirements for the course is a bonafide work carried out by us and has not been previously submitted to any other course. We further certify that no part of this work shall be published, reproduced, or distributed in any form without the prior permission of our supervisors. We understand that any such unauthorized use of the project work may be considered a violation of academic ethics and result in severe penalties. We also affirm that the project work has been carried out under the ethical standards and guidelines set forth by our supervisors. We acknowledge that our supervisor has the right to make any modifications or revisions to the project work that may be deemed necessary. We also agree to abide by any additional terms and conditions as stipulated by our supervisor.

Date:19-05-2023

Signature of Student:
Chirag Kumar
2020BTechCSE022
JK Lakshmipat University, Jaipur



Signature of Student:
Prateek Sharma
2020BTechCSE093
JK Lakshmipat University, Jaipur