Law Case Recommendation System

Clustering based recommendation based on document summary embedding

Meet Nitin Mandhane
MT2022061
IIIT Bangalore
Bengaluru, India
Meet.Mandhane@iiitb.ac.in

Narasimhan N
MT2022062
IIIT Bangalore
Bengaluru, India
Narasimhan.N@iiitb.ac.in

Jacob Mathew
MT2022150
IIIT Bangalore
Bengaluru, India
Jacob.Mathew@iiitb.ac.in

***Abstract*—By consolidating the law cases from indiankanoon.org website through manually scraping the required data, we have aimed to summarise, vectorise and provide a methodology to perform clustering, similarity and recommendation of similar cases to a given prompt or case.**

Further, the learning's from this approach have been refined and a future approach to obtain direct embeddings from entire text rather than from summary has been theorised to improve performance albeit with a penalty to training cost and time

***Index Terms*—summarising, embedding, recommendation, law cases, scrapping**

I. Introduction

One of the important aspects of any court system in the country is to have transparency and access to all court cases that have verdicts passed. This allows for the public to scrutinise and generalise the verdicts opening up for interpretation and debates that lead to harmonious building of a democratic society.

However, to be able to access all cases and search them for cases that you are interested in is an insurmountable task, especially into the future since the number of cases continue to increase in the coming years.

Moreover, in a legal battle, case law is also a crucial component in which a lawyer can quote a previous ruling to increase their odds in the current case.

Case law, also used interchangeably with common law, is law that is based on precedents, that is the judicial decisions from previous cases, rather than law based on constitutions, statutes, or regulations. Case law uses the detailed facts of a legal case that have been resolved by courts or similar tribunals [1].

This also saves time since the court can rely on previous rulings rather than spend more time deliberating on the issues. The time savings are also crucial since the number of cases that are backlogged in the court are astounding.

54,000 pending cases in Supreme Court, 43,00,000 pending cases in High courts and 2,76,000,000 pending cases in subordinate courts [2].

The problem is also exasperated with the way legal documents are stored and displayed. There are a lot of references, line numbers and legal requirements that even searching for a keyword requires sifting through a lot of page breaks and div tags.

The final nail in the coffin is that a simple search for keywords will be non intelligent. Most cases will be indictment under multiple sections as well as lots of legalese that is common in all documents. In such situations just using key words or regular expressions to search will give subpar results. Additionally, the ability to search based on a sentence or prompt or similarity to another case is much more beneficial to lawyers and laymen.

With these motivations, the project's ground rule is to explore and build a pipeline that can be repeatable to scrape and create a dataset of all legal data from 1947 to the present. Then, comb through the dataset to extract the summaries and vectors. The vectors will then be used to cluster and find similar cases on the basis of similarity measures.

II. Related Works

Jenish Dhanani, Rupa Mehta, Dipti Rana, Sabu M. Thampi, El-Sayed M. El-Alfy, and Ljiljana Trajkovic presented a paper that recommended legal documents on the basis of the similarity of judgements [3]. They obtained embeddings by running the judgement of a legal document through Doc2Vec algorithm and then performing pairwise comparison which increases quadratically i.e $O(n^2)$ as the number of cases increases. They have concluded their paper with encouraging performance as measured with accuracy scores like F1 Scores and MCC Scores.

We build upon this paper by running the entire legal case document through a summarising algorithm to generate a summary for the legal case. This means while the mentioned paper was restricted to the vectorization of the judgement alone, we are summarising the entire judgement so that we are also generating vectors of similar dimensions so as to keep time complexity comparable to the reference paper's pipeline.

Shounak Paul, Arpan Mandal, Pawan Goyal, Saptarshi Ghosh presented a paper on fine tuning a summarising model for Indian legal documents [4].

The paper takes a model that has been pretrained on US SEC filings and litigations. They have then fine tuned it on Indian legal data and have made the model available on hugging face which we have imported and set up as to summarise the data.

The motivation of pretraining transformers for Indian Legal text was to eliminate the bias and model hallucination that

occurs using a generalised summarising model that has been trained on US SEC rulings and legal text.

We have also tried out a generalised summarising algorithm that we ended up using as part of our primary pipeline [5]. The advantage of using such a model was the more layman summary that it generates.

III. Dataset Curation

A. Dataset Ideology

The first step is to create a dataset that we scrap from indiankanoon.org. We use the BeautifulSoup package to allow us to load and scrape the web page for which we are providing the link.

So the first step is to collect the URLs of the web pages that we are going to scrape. We restrict ourselves to Supreme Court cases alone because these cases will have no appeals and hence no duplicate case files.

The next thing that we note is the quantity of the dataset. A total of 47,000 cases are there. We decide to restrict to 8,000 cases and work on this subset to infer relevant findings and observations that can later be extended to the entire dataset

B. Dataset Scrapping

We start by scrapping all the URLs year wise from 1947 to 1990 which comprises roughly 8000 cases. We create a document of case links for every year.

We then iterate over each year's document, scraping all the cases within that year. This forms the initial part of our dataset.

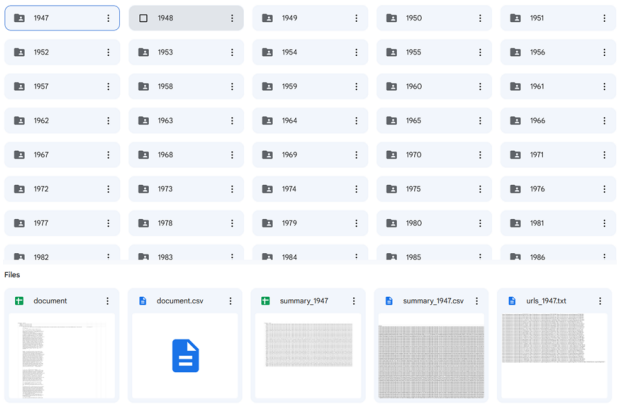

Fig. 1. Scrapped output sorted year wise

C. Dataset Summarisation

For every case that we scrap we pass it as input to the summarising model. The model sets the required number of tokens in the summary based on the length of the input data i.e the length of the words in the case.

We have majorly used the Distilbart-cnn-12-6 model to summarise our cases. It is trained on a specially curated dataset created by journalists from CNN and BBC. This lends to the simple and layman readable nature of the summaries.

We experimented with Legal LED which was a highly rated summarising algorithm specifically tuned for legal data. The

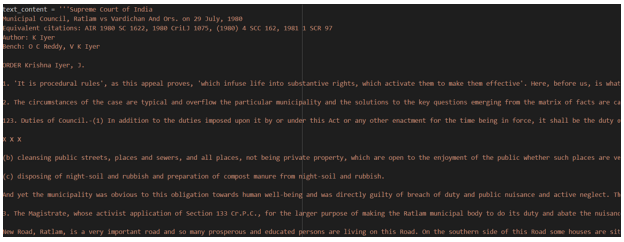

Fig. 2. Sample input text to distilbart cnn-12-6 model

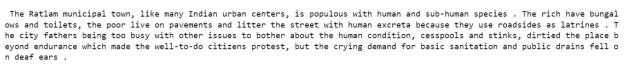

Fig. 3. Sample output from distilbart cnn-12-6 model

summaries generated by the model were far more resembling the legalese that lawyers use. However, the model was trained on the US SEC litigations and rulings thus resulting in the model hallucinating US Court names and locations. This led to us eventually abandoning the use of this model.

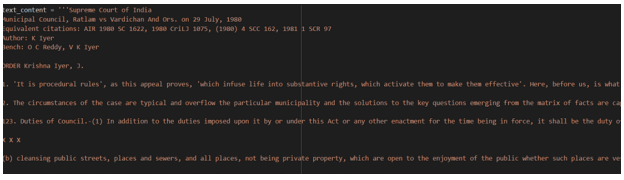

Fig. 4. Sample input text to legal LED model

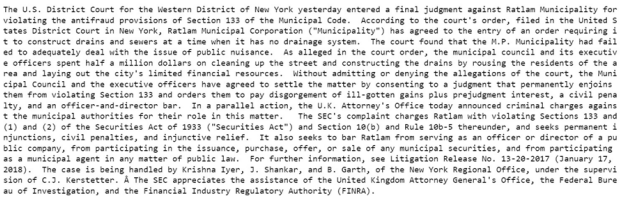

Fig. 5. Sample output from Legal LED model

We finally stumbled upon inLegal BART which was specifically curated for Indian legal documents. The researchers from IIT Kharagpur have taken the Legal LED Model and fine trained it on a lot of Indian legal documents and cases which resulted in a model that gave legal descriptions matching the Indian cases without any hallucinations.

However, running inLegal BART took much longer due to the requirement of CUDA. While Distillbart was able to summarise 20 documents in 5 minutes, inLegal bart was taking in excess of 10 minutes to summarise each case. We will address these shortcomings in the future research section. Hence in the interest of a more time efficient yet dirty result, we decided to use Distbart for summarising the cases that we scrape.

The output of the summarising algorithm is stored in a CSV Format that stores every row as a pair of case's URL and its summary.

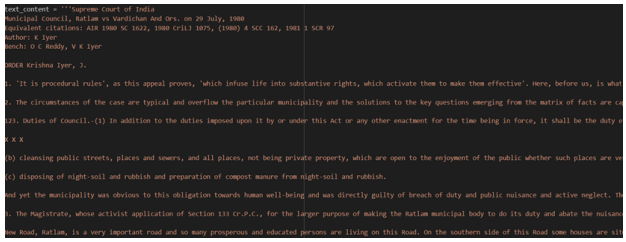

Fig. 6. Sample input text to legal LED model

The Ratlam Municipal Council, Ratlam vs Vardichan And Ors., ;, challenged the sense and soundness of the High Court's affirmation of the trial court's order directing the construction of drainage facilities and the like which had spiralled up to this Court. It was contended on behalf of the Municipality that : (1) The circumstances of the case were typical and overflow the particular municipality and the solutions to the key questions emerging from the matrix of facts are capable of universal application, especially in the Third World humanscape of silent subjection of groups of people to squalor and of callous public bodies habituated to deleterious inaction; (2) Section 133 Cr.P.C. casts a mandate: "Duties of Council" In addition to the duties imposed upon it by or under this Act or any other enactment for the time being in force, it shall be the duty of the Council to make reasonable and adequate matters within the limits of XXXXXXX(X)(b), namely, (i) cleansing public streets, places and sewers, and (ii) removing all obstructions and nuisances which are not open to public enjoyment; (iii) abating the pollution caused by the use of roads and

Fig. 7. Sample output from Legal LED model

IV. Vectorisation

Once the dataset has been created we are going to convert the textual summary into a vector. Conversion to vector space will allow us to mathematically perform calculations that measure similarity and further give recommendations. The vectorization approach consisted of trying out Word2Vec and TF-IDF methods with TF-IDF being used as the final vectorisation algorithm.

```
In [75]:  1 km1.vectorized_summaries

Out[75]: <8819x17032 sparse matrix of type '<class 'numpy.float64'>'
                with 201954 stored elements in Compressed Sparse Row format>
```

Fig. 8. Total vector space of all embeddings

V. Similarity and Clustering Algorithms

Once we get the vectors for each case summary we are able to perform various algorithms to try and determine the closest vectors to recommend the similar cases.

Between the two options of euclidean distance and cosine similarity, we have chosen cosine similarity due to its robust performance and the intuition that while vectors might have larger distance between them, if they are pointing in the same direction then they generally have higher similarity.

This is further reinforced by the fact that each case's vector has anywhere between 35 and 50 dimensions but the matrix comprising all case's vectors has dimension of 8000 x 42,000 suggesting that there is large variance of the embeddings causing the euclidean distance to be abnormally high. This implies that we can rely on the cosine similarity to identify the closest similar cases.

A. K Nearest Neighbours

This algorithm is a direct derivation of using cosine similarity to try and find the K closest vectors that represent the summary of cases. This means that we are finding the K most similar cases to the vector. And the current vector was created from either a prompt or from a summary of some other case.

The algorithm for K Nearest Neighbours runs without any need for clustering and hence is the most straightforward method to generate similar cases recommendation.

B. K Means Clustering

Instead of just depending on the closest cases based on a similarity metric alone, we explore the option of clustering in the dataset vector space to extract clusters of common cases and then recommend within the cluster.

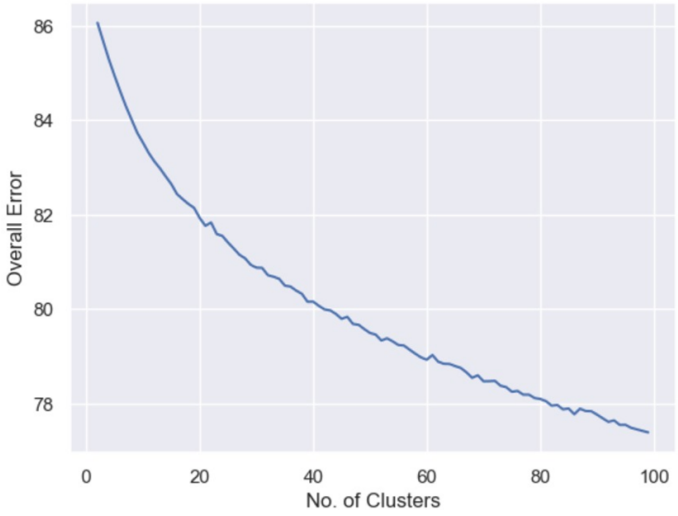

Fig. 9. Total error v/s cluster number graph

The logic is that boundary cases, particularly in sparse vector space, will respond better to clustering than nearest neighbours as sometimes clusters can be identified in sparse spaces. To find the optimal K value i.e the number of clusters, we can brute force the algorithm to run from K = 2 to a high enough number, plotting the total loss of error against the number of clusters. The ideal point will be where there is a sharp change in the slope of the graph which we define as an elbow point. That point's K value corresponds to a good number of clusters that is balanced between over fitting and under fitting.

C. DBSCAN

We also explored DBSCAN or *Density-Based Spatial Clustering of Applications with Noise* as an additional clustering algorithm to try and let the dataset speak of itself.

The DBSCAN algorithm decides on the number of clusters and the boundary of clusters by itself. The algorithm relies on us primarily tuning the hyper parameter of maximum density which is the maximum distance beyond which the algorithm will determine that this is another cluster.

However, even with relentless tuning of hyper parameters as well as trying both TF-IDF and Word2Vec we face severe clustering issues with DBSCAN. These will be discussed in

the analysis section but case in point, we decided to abandon using DBSCAN as part of the pipeline.

VI. Analysis

A. K Nearest Neighbours

Using cosine similarity along with the choice of pre-processing the data or not, K Nearest neighbours was able to give good recommendations for a given prompt or case summary.

However, there were some edge cases where it also started suggesting cases that had some common sections while the premise of the case was different. This boiled down to the fact that if we were to cluster the cases then cases on the edge would sample close to more than one cluster and hence the odd recommendation. However the summarising and vectors of the summary ensured that the cases were somewhat related and hence the system never presented cases that were totally unrelated.

B. K Means Clustering

We tried out cluster sizes from 2 to 100 and have a smooth reduction in accuracy till cluster size 20. Beyond cluster size 20 the reduction becomes much less smooth and the graph shows jagged drops which intuitively inspires much less confidence in trying to tune for that many clusters.

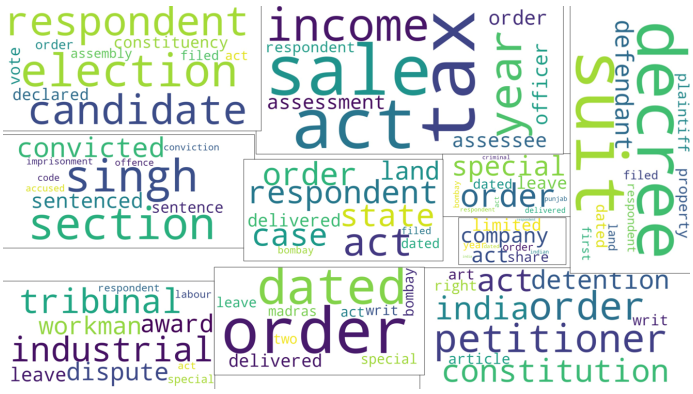

Fig. 10. Top words in each cluster

We also have the prior knowledge that we have to choose clusters of minimum size to get good related cases, else we will start getting over fitting based on the section numbers of each case. This was further reinforced by the fact that there are broadly only a few categories of cases that we need to worry about with each case having its own specific law sections.

Eventually, we decided to choose cluster number of 10 which upon manual verification for a wide range of prompts gave good recommendations.

From there the recommendations were restricted to cases that were within the cluster itself. So this prevented recommending cases from the next cluster. This slightly improved performance for cases that were suspected to be on the edge of clusters.

C. DBSCAN

With the improvement seen going from K Nearest Neighbours to K Means Clustering, the next goal is to explore the potential results of letting the algorithm itself choose clusters based on the density of vectors. This could bring out hidden meaning within the vector space that the previous rigid algorithms could not explore.

However, repeated fine tuning of hyper parameters nor different vectorisation algorithms proved to give sufficiently acceptable results.

The only positive inference was that DBSCAN was much better at identifying corner cases and classifying them as an outlier which led to some minority type cases being classed as a separate cluster. This was lost in the generality of K Means Clustering and K Nearest Neighbours.

VII. Results

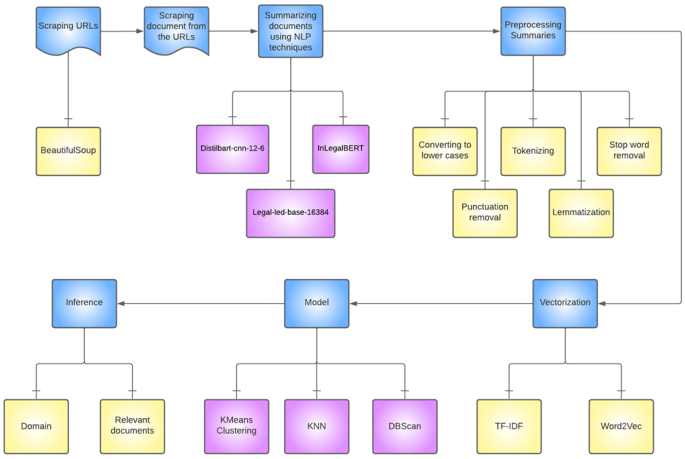

Fig. 11. End result pipeline

Our results can be broadly split into categories based on the various major stages of our pipeline:

A. Analysis of Dataset Scrapping

- Our first time constraint was the amount of time it took to scrape websites. We did not get access to indiankanoon.org's API and hence made full website requests to scrape the data from it.
- Multiple requests especially in a multi threaded design can lead to getting blacklisted by the DDoS service that they are using.

B. Analysis of similarity and clustering algorithms

- We found that a simple K Nearest Neighbours gives really good results in terms of the recommendations that it gives.
- We also note that it is able to handle auxiliary verbs better without preprocessing the data than with preprocessing. We attribute this to the fact that preprocessing removes these auxiliary verbs which sometimes the the

embeddings generation picks up on and uses to give better recommendations.

- However even without auxiliary verbs the recommendations are not extremely far off either.
- Consider the following figures where we compare the output between with preprocessing and without preprocessing:

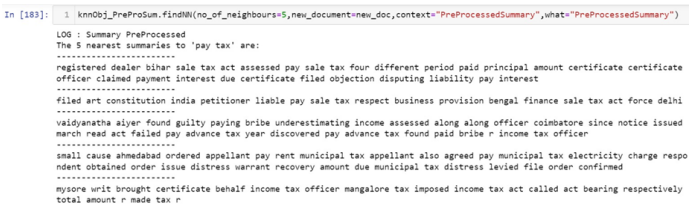

Fig. 12. The keyword with preprocessing is ”pay tax”

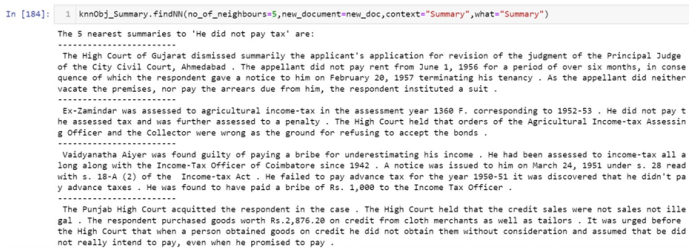

Fig. 13. The keyword without preprocessing is ”He did not pay tax”

- This shows that the top recommendation with preprocessing was trying to find all tax related cases.
- However the case without preprocessing gave results that are specific to cases that are not paying taxes.
- To make this effect felt through out the model requires more focus on the embedding algorithm that we are using to extract more attentive details.
- Trying out the algorithm for some common cases with prompts like ”The tenant fought with him” would give results as follows.
- This showcases the good performance of both algorithms provided the embeddings are good.

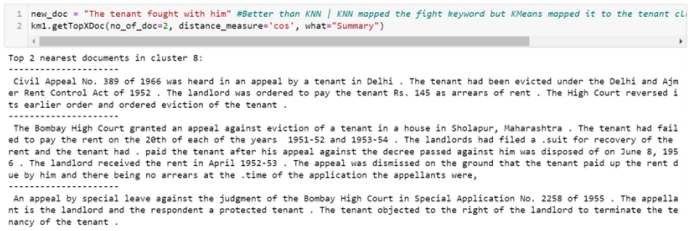

Fig. 14. Recommendations from K Means Clustering for ”The tenant fought with him” prompt

- On the contrary, the poor performance of DBSCAN has no notable recommendations. This is due to the poor clusters that is generates for the given dataset.

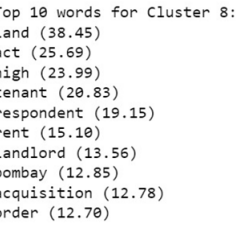

Fig. 15. The top words for the corresponding cluster of ”The tenant fought with him” prompt

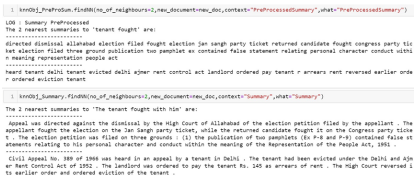

Fig. 16. Recommendations from K Nearest Neighbours for ”The tenant fought with him” prompt

- However while clustering algorithms will try and make every point fit into the given number of clusters, DBSCAN will generate small clusters as outliers were detected well.

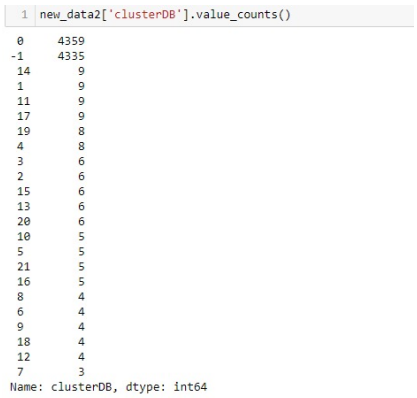

Fig. 17. Recommendations from K Nearest Neighbours for ”The tenant fought with him” prompt

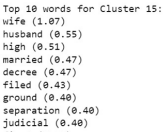

Fig. 18. DBSCAN picking up outlier cases

- DBSCAN picked up the outliers like divorce cases which are very little in the dataset which consisted from cases from 1947 and are from the supreme court.

VIII. Future Scope and Research

From these results there is future scope to further improve and create a much more intricate and accurate recommendation system. A simple addition will be to gain access to the indiankanoon.org API access to be able to much easily and quickly scrape the website for access to the entire database of cases. This also doesn't affect their servers with unnecessary request.

Following the footsteps of the research paper that ran similarity metrics only on the judgement, we summarised the entire paper to then later obtain the embeddings that were comparable to the content of the entire paper while still being computationally light by using methods like Word2Vec and TF-IDF. This gave fast yet dirty results that proved that such systems do have potentially to be further fine tuned and extended into full fledged public facing systems.

With growing research into LLM and GPT we have much more powerful embedding algorithms that utilise LLM models like BERT (Bidirectional Encoder Representation from Transformers), GPT (Generative Pre-trained Transformers), BART (Combination of BERT and GPT) and GLoVe (Global Vectors for Word Representation). Hence the first option will be to use such LLMs to generate the embeddings for the summaries itself.

Then with access to commercial computational technology like CUDA acceleration or commercial solutions like Open AI's API access we can then generate embeddings by passing the entire case itself. This will be the maximum information that we can extract from the dataset and thus allowing no loss of specificity. However the time required and access to such computational resources also questions the brute force nature of such methods which leads to some form of dataset selection.

A different perspective in terms of similarity measure is also a key area to explore. Rather than choosing between euclidean or cosine similarity, we can incorporate both using metrics like Mahalanobis distance.

We can start by manually indexing cases by quickly referencing the case sections that each case involves. This means that we can sort cases based on their IPC or Indian Penal Code Sections. Thus we can curate datasets containing a large enough coverage of all types of IPC sections and case types. While this requires manual intervention of a domain expert, we can cut down on the number of data points that need to be converted into embeddings and perform suitable and satisfactory clustering.

With these data preprocessing we also can look into triplet networks since the labels provided by the domain expert can be treated as the possible clusters and hence using triplet networks to optimise the vector space to reduce intra class variability and increase intra class distance (using metrics like Mahalanobis distance) will suit our problem statement as well.

However since this is a continuous stream dataset due to new cases coming each and everyday, we will have to consider how long before we retrain the recommendation system to include the new data points. This influences how up to date our recommendation system can be as a trade off between recent events versus time and computation needed to rerun algorithms

References

[1] Case Law, Cornel Law School. Definition.

[2] Insights into Editorial: A ticking bomb: the pendency problem of Indian courts. Statistics

[3] Jenish Dhanani, Rupa Mehta, Dipti Rana, Sabu M. Thampi, El-Sayed M. El-Alfy, and Ljiljana Trajkovic. 2021. Legal document recommendation system: A cluster based pairwise similarity computation. J. Intell. Fuzzy Syst. 41, 5 (2021), 5497–5509. link

[4] Pre-training Transformers on Indian Legal Text, Shounak Paul, Arpan Mandal, Pawan Goyal, Saptarshi Ghosh. link

[5] Distillbart-cnn-12-6. link