
TEAM 2

Index

1. ABSTRACT

With crimes increasing monotonically, it becomes essential to impart justice to the victims. To look for relevant information, one needs to study several previous judgments. Reducing the time spent on research can speed up the judicial process drastically. The time consumption mostly happens in two areas - searching for the right document and understanding that document. To start with, being able to get hold of the appropriate judgments or other legal documents is the essential task for any domain experts or common people. Once a document is obtained, the next most fundamental task is to read and re-read it and come to conclusions. To resolve these issues, there is a need for an efficient search system which can provide searching options based upon multiple views and inferences from the documents. To reduce the time spent in reading documents, we intend to present the information in the judgments visually through semantic networks.

2. MOTIVATION

The concept of 'relevance' is crucial to legal information retrieval, but because of its intuitive understanding, it goes undefined too easily and unexplored too often.

Meet a farmer X, who works hard day-in-day-out on the crop field. He is the only financial source of the family. Due to tuberculosis, he suffered an early death, leaving his family helpless.

Although he did have life insurance at a reputed company.
But after his death the organization is refusing the widow, Y, to pay the covered amount. The problem is that she has no legal knowledge and experience in reading legal proceedings.
Our system will be of great help for people like these. Anyone can search the previous cases and other relevant information similar to their problem from our legal search system.

We discuss an end-to-end conceptual framework on relevance within legal information retrieval. It is based on a typology of relevance dimensions used within general information retrieval science and deep learning but tailored to the specific features of legal information. This framework can be used for the development and improvement of legal understanding.

3. RELATED WORKS

Many works have been done previously in this field. There exist many legal search systems that provide these functionalities

1. Manupatra
2. Westlaw India
3. Legal Crystal

Most of these systems are not very efficient and lacks relevant content.

ISSUES: They use the traditional methodology of information retrieval (e.g. keyword search) and fails to capture the

semantics of the case. Legal experts still rely on Google search to find relevant documents. Thus, a domain-specific search system is required for legal search and retrieval, especially in the Indian context. Some of them are paid services.

4. DATASET

Original Data provided contains:

1. All_FT folder which contains all the files of complete cases.
2. Acts Folder which is further divided into central and state folders. They contain files of respective acts.
3. actlist.txt lists all acts.
4. subject_keywords.txt which contains subjects and catchwords for each case.
5. doc_path_ttl_id.txt which contains the name and id for each cases.

We need to extract the data from the above files and form various .txt and json files for easy and fast processing. These files basically contains mapping from one feature to other.

Several mappings were created like:

1. case_to_acts.json which maps each case with the acts cited in the case.
2. case_to_subject.json which maps each case to all the subjects which are related to that case. There were around 53k cases which contains in total 63 different subjects. This is useful in categorizing the cases. Similarly reverse mapping of subject_to_case were formed.
3. caseCitations.txt which maps the cases cited in each case. This is helpful in forming the citation graph which in turns decides the ranking in which cases appear on the final webpage.

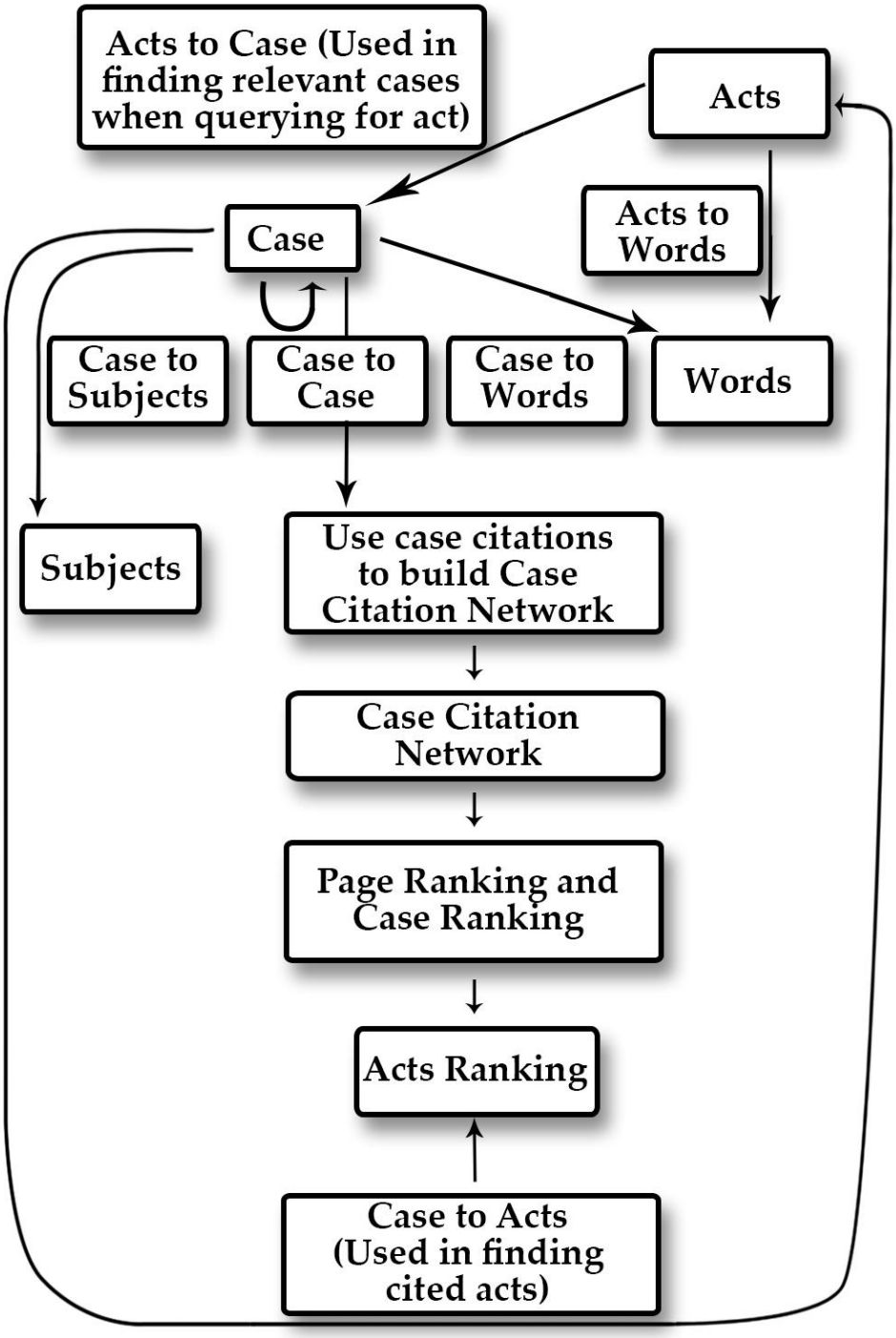
Several other mapping were also formed like case_to_judges, case_to_catchwords, catchword_to_cases, act_to_cases, case_to_date etc. These all were very important part of preprocessing of the data to get faster results.

5. METHODOLOGY

Our system retrieves information in a pipelined fashion. The query is pre-processed in the initial step covering spell check, case filtering, tokenization etc.

Output query is classified into one of the following four types:

Type 1 query: A set of legal/non-legal keywords
Example: C*ulpable homicide, murder*


Acts to Case (Used in finding relevant cases when querying for act)
Acts
Acts to Words
Case
Case to Subjects
Case to Case
Case to Words
Words
Subjects
Use case citations to build Case Citation Network
Case Citation Network
Page Ranking and Case Ranking
Acts Ranking
Case to Acts (Used in finding cited acts)

Type 2 query: Name of a particular Act/section of an Act
Example: *IPC 302, Indian Institutes of Management Act, 2017*

Type 3 query: Title of a particular case
Example: *State Of Maharashtra v/s Sitaram Popat Vetal And Anr.*

Type 4 query: A query in natural language
Example*: I ordered some materials online on Shoppers stop. The amount was deducted from my account. But shoppers stop have not delivered the materials and also they are not ready to refund my money which I have already paid. I want to take strict action against them*.

Depending on the type of query the outputs are retrieved from the pre-trained model and rendered to the web-page.

5.1 PREPROCESSING

Retrieving information from raw-data can be extremely time-consuming and the presence of irrelevant information may lead to wrong results. The problem of natural language understanding has to be taken into consideration.

Off-Line: These are the precomputations required for the system

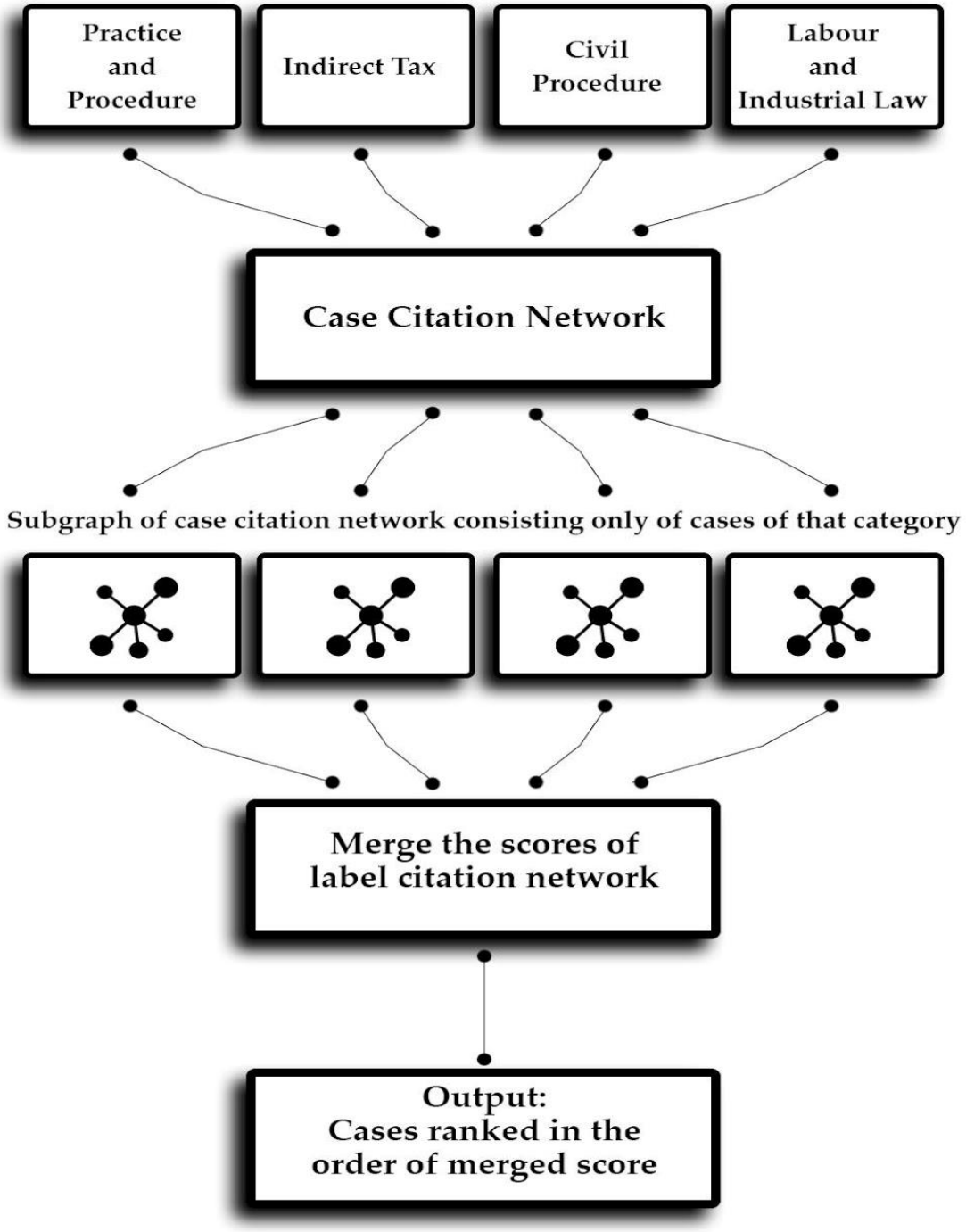
s

- **Indexing -** Set of standard legal words are first extracted from the dataset on the basis of term frequency averaged over all the documents.

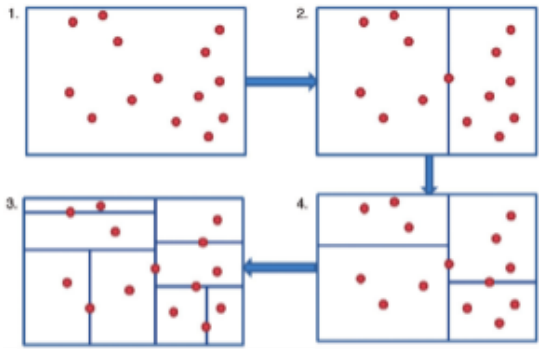
All the legal documents are then tagged in the list of these standard words for quick search and retrieval purpose.
Example: *1975_P_32 Hindustan Steel Limited v/s Presiding Officer, Industrial Tribunal and Another*

- **Summarization -** Summarization is a challenging sub-task of the broader text-to-text generation field of NLP. Summaries are usually generated by extracting 'important' portions of the text. Extraction-based methods are often used because abstraction-based summarization is an open problem in NLP. We have to explore the following methods:
  a) SUMY
  b) Case Summarizer
  c) RBM (Restricted Boltzmann Machine)
  d) Summarization Bot (Slack)

Comparison: The summarization bot performs the best although we only have limited access to that. CaseSummarizer and RBM also give a very good result.


Practice and Procedure
Indirect Tax
Civil Procedure
Labour and Industrial Law
Case Citation Network
Subgraph of case citation network consisting only of cases of that category
Merge the scores of label citation network
Output: Cases ranked in the order of merged score

- **k-D Tree Construction -** Finding conceptually similar words of the query type 1 keywords. For that, we are using word embeddings (GLoVe). Searching for the entire space is very computationally expensive. By using k-D Tree the searching operation becomes highly optimized.



KD Tree splitting in 2 dimensions

- **Citation graph -** Build the graph with the case id as a node. If case A cites case B then there will be an edge from node A to node B in the graph.

$$PR(u) = \sum_{v \in B_u} \frac{PR(v)}{L(v)}$$

PR(u):PageRank of case u, L(u): number of cases which cites u i.e. number of cases link to u in Citation graph. Bu: set containing all cases linking to case u.
The algorithm involves a damping factor for the calculation of the PageRank.

- **Keywords Extraction**
From each case document, words were extracted out and stopwords like 'and', 'of', 'in' etc. were filtered out. These words were then lemmatized to their base forms(for ex running -> run). For each document the top keywords were identified according to the term frequency–inverse document frequency (tf-idf) statistic.

On-line: These preprocessing steps are computed on query during searching.

- **Spell Correction -** The user inputs a query into our web-based search portal. But one can enter a query which may contain spelling mistakes, slang and word-breaking issues.

  Before giving the query to our query classifier, we corrected the query in case it contains errors. We used Bing Spell Check API for this purpose.

  But the problem with the API is that it gives JSON output which needs to be adjusted to form the actual corrected string. This is handled by our API which takes string query and gives corrected string output.

- **Linguistic Pre-Processing -** Depending on the type of the query the natural language text is pre-processed.

- **Fuzzy String Matching** (fuzzywuzzy) - An API for fuzzy string matching. Given two strings, this uses various algorithms to give a measure of the match between the two strings.

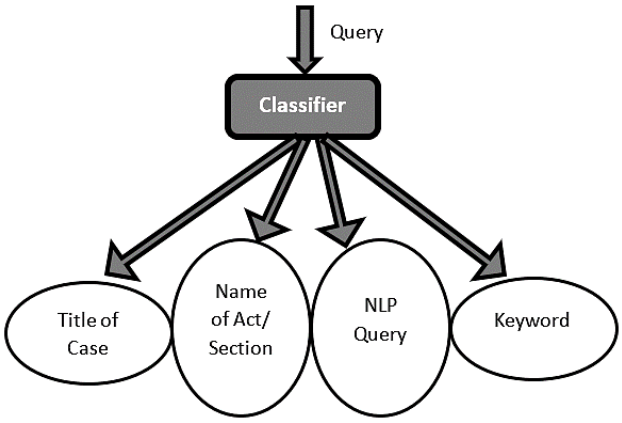
5.2 QUERY CLASSIFIER

The user inputs a query into our web-based search portal. To reduce the query-search time, we classified the user's query into the most probable query type out of the four specified types based on the score value.

For rudimentary classification, we used obvious unique attributes of various queries. For instance - Query 3(as described in the problem statement) always has the word "versus" or some of its variation in the query string. Query 2 mostly had the word "Act" or "Bill", or the name of the act in some form in the query string. If the query string has these attributes, then a positive high score is assigned to the Query type.

A small challenge, as we found out, was to clearly distinguish query 1 from query 4. It can be observed easily that query 1 comprises only of keywords like murder, death, robbery, burgled, while query 4, being in user's natural language, comprised of the keywords along with many stopwords. So, if the query contains the ratio of stopwords to total words above a certain minimum threshold, then it a type 4 query, otherwise, it is of type 1

Following this crude classification, we find the query type with the highest score. Then, a search is initiated for the query in the pre-processed datasets which were prepared for that query type(all datasets discussed below). This will naturally provide a set of legible matches. We then show the top results, based on a threshold, to the user.



5.3 MODEL

Type - I Query

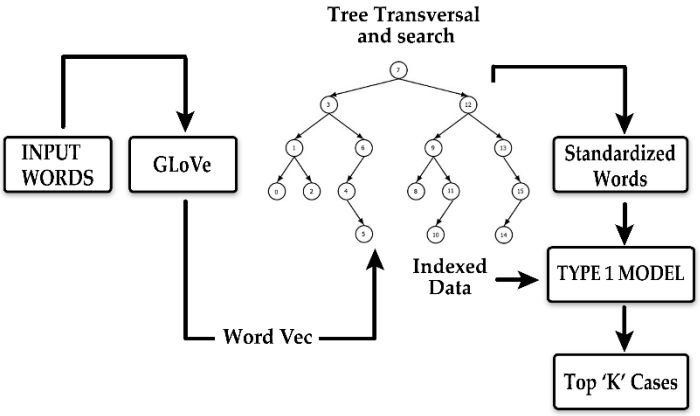
1. Given a list of keywords. It finds the similar words embeddings by querying k-D Tree.
2. Query top-K documents from the citation graph.
3. Score the documents using tf-idf.

Identifying Important Words:

Set of Words which represent a case or an act was derived using tf-idf methodology.

tf(t) = TF/N
idf(t) = $log_e(\frac{N}{df})$
tf-idf = TF(t)*IDF(t)
N : Total documents
TF : Term Frequency
df : Document Frequency

After finding tf-idf of every word in a document, those words are removed which occur rarely in the set of documents (misspelled words, slang terms, Names of individuals, companies).



Type - II Query

Challenges

For users following pair of queries means the same:

1. (*Frequent Use of Abbreviations*)
   Indian Penal Code
   And
   IPC
2. (*Spelling Mistakes*)
   Banking Service Commission Act
   And
   Banking Service Commission
3. (*Out of Order Query*)
   Central Board of Direct Taxes Act
   And
   Direct Taxes Act Central Board
4. (*Handling abbreviations*)
   IPC, 1961
   And
   Indian Penal Code, 1961

Approach

- **Preprocessing :**

Firstly we removed all the punctuations from the input query. Then used the Bing spell check API to correct spelling mistakes, identify slangs and correct some word-breaking issues to give cleaned query.
Abbreviations are mapped to the list of acts. Acts to Cases mapping is used which for an act lists the cases in which it was cited. Case Ranking is used which was formed using a case citation network.

- **Finding relevant acts :**

Fuzzywuzzy python library is used to get the acts which are matching to a given search query from the Acts list prepared. Acts are scored on the basis of the matching with the query.

- **Finding relevant cases :**

Using the acts obtained in the above step, the cases are extracted which cites these acts. The act to cases mapping is used here. The cases obtained are further ranked by the score given by the citation network.

- **Indexing of acts :**

To reduce the number of acts to search from, acts have been indexed according to the first character of all the important words.
And hence when a user gives a query in the form of abbreviation, we search in the list of acts under the characters of the given abbreviation.

Type - III Query

Challenges

For users following pair of queries means the same:

1. (Reversing the party names)
   Sohan Lal v Ram Prakash
   And Ram Prakash v Sohan Lal

2. (Using abbreviations)
   Central Bank of India v State of Gujarat
   And
   CBI vs the State of Gujarat
3. (Users omitting some words)
   The state of Maharashtra vs Mehboob
   And
   Maharashtra vs Mehboob
4. (Spelling Checking)
   Rampyari Bai vs
   Municipal Corporation
   And
   Municipal Corporation vs
   Rampyari Bal
5. A very large number of cases to search from, for the abbreviations.

Approach

- **Identifying and separating the number of parties entered by the user :** Preprocess the entered query by removing punctuations and reducing it to lowercase. Then search for separator keywords such as v, vs, versus. If keywords are found then there are two parties else only one. The sub-string before the keyword is party 1 and the one after is party 2.
  Assumption - The user enters one of the keywords from v, vs, versus, or their modified forms (such as 'Vs.', 'V.S.', 'Versus') to indicate two parties in a case.

- **Searching and replacing abbreviations :** Before the actual search, the query is checked for any acronyms used. A dictionary of commonly used abbreviations in government, financial, legal and other organizations was made. The acronyms were searched using fuzzywuzzy API to handle the errors in acronyms. The acronyms are then replaced with their full forms.

- **Searching for a party :**
  A mapping from a party name to cases in which that party name exists was prepared.
  To improve the speed of searching the list of party names in which a party is searched is reduced.
  Every alphabet contains a list of party names which contains that alphabet as the first letter of one of the words in the party name.
  The first character of each word in a party is extracted and the party name is searched in the list corresponding to that character using fuzzywuzzy. This greatly increases the search speed. The search results in a list of possible party names that match the party in the query along with a score of similarity.

- **Combining the two results :**
  Now that we have got a list of possible party names for both the parties 1 and 2. We extract the corresponding list of case names for each party in the above list, from the file consisting of a map between party name to the list of cases with that party name. Now we take the intersection of the two lists, containing the list of possible files with a score of relevance. The cases are ranked according to the score assigned.

- **Indexing of cases :**
  To reduce the number of cases to search from, cases have been indexed according to the first character of all the important word of both the party names.
  And hence when a user gives a query, we take out the first character of each word and then look for the cases under those characters' only.

Type - IV Query

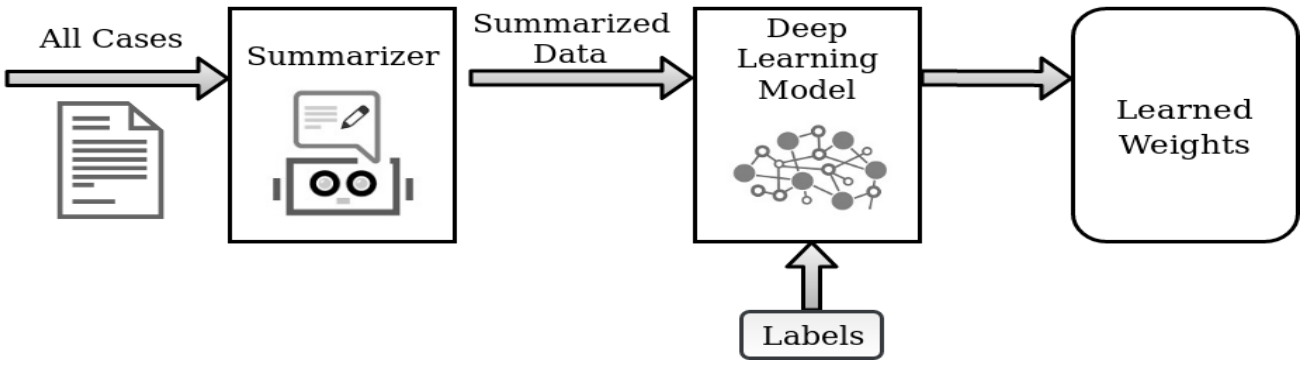
Challenges

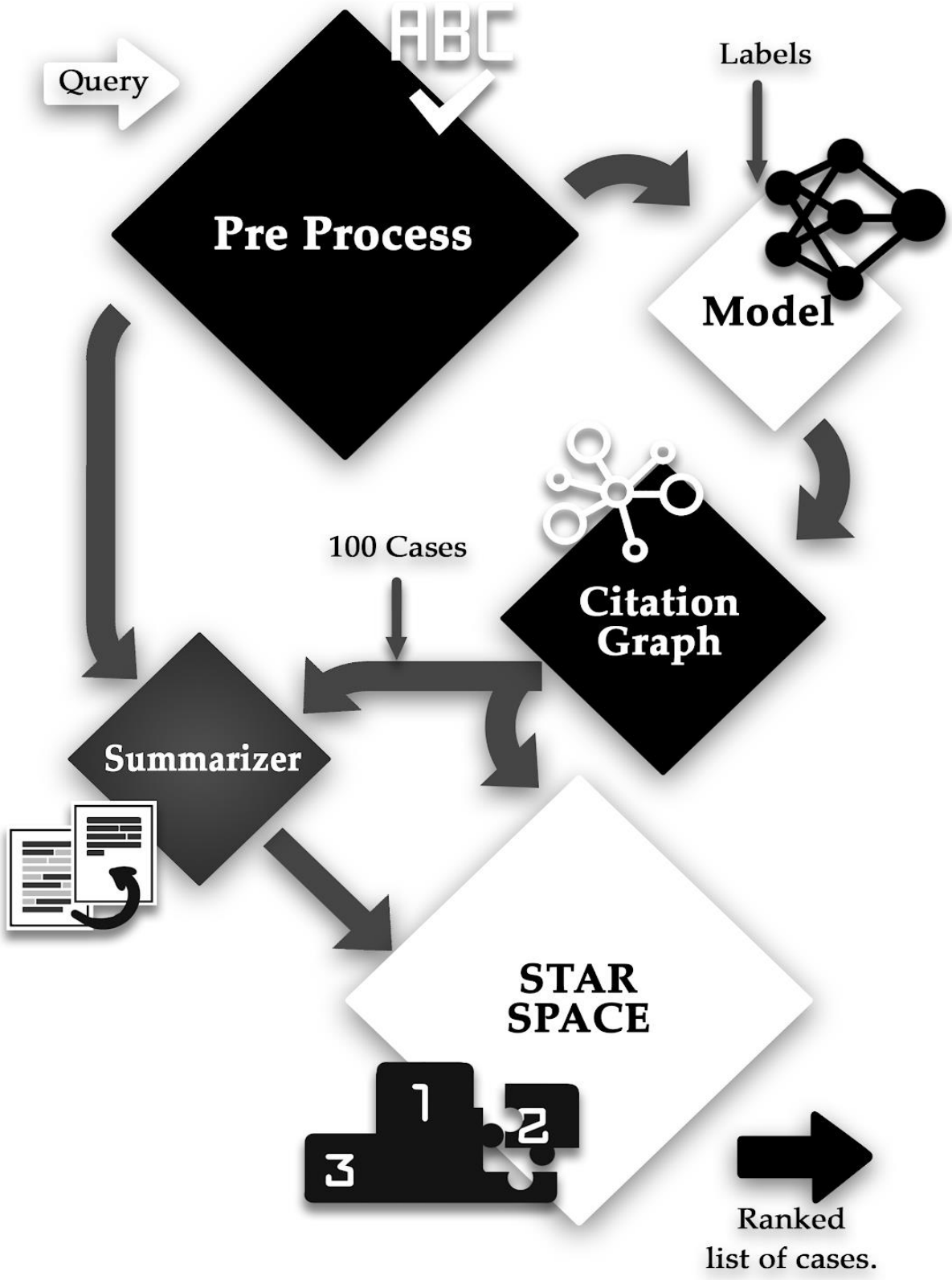
The main challenge that we face to understand the context of the user. By using the traditional machine learning methods it's very difficult to achieve that task.
Also the data in the case file is very large and complex. So, searching a query on all the 53K documents using deep learning

Approach

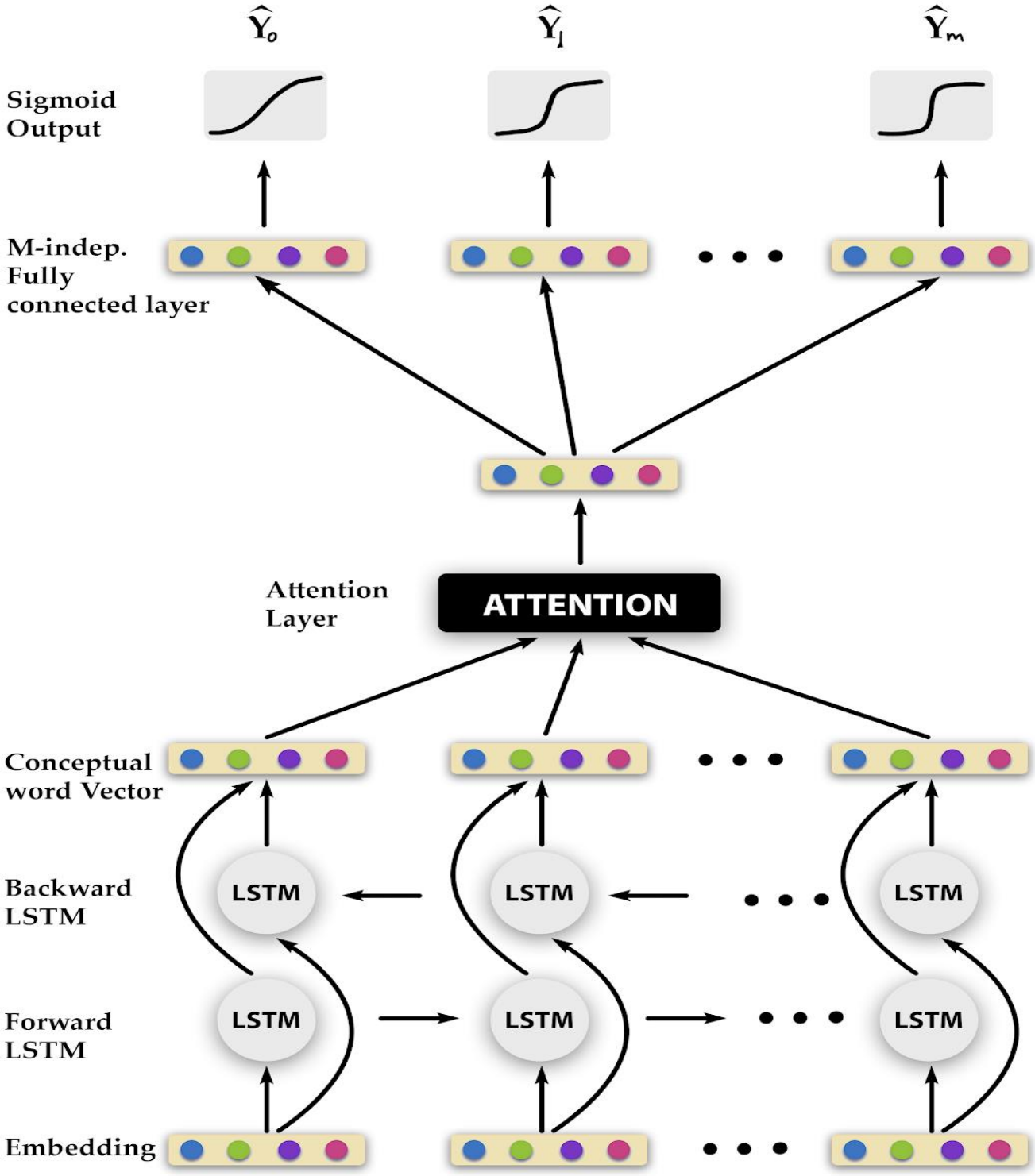
- **Training time :**
  A Attention based bi-directional multitasking model was trained on the summarize dataset.
  a) The dataset is divide as following training:validation:test :: 80:10:10
  b) The embeddings are extracted from GLoVe pre-trained embeddings. After that these are encoded using Bi-LSTM with attention and the final transformed embeddings are feed to 45 independent fully connected layer, each attached to sigmoid output.
  c) Binary categorical loss function is used.
  d) F1 score is calculated for each label and threshold is calculated.

- **Testing Time :**
  a) Using the pre-trained attention model the label for each case is calculated
  b) Use the citation graph to sample the cases based on PageRank score.
  c) Using Facebook's Startspace model we rank these output paper according to the relevance.



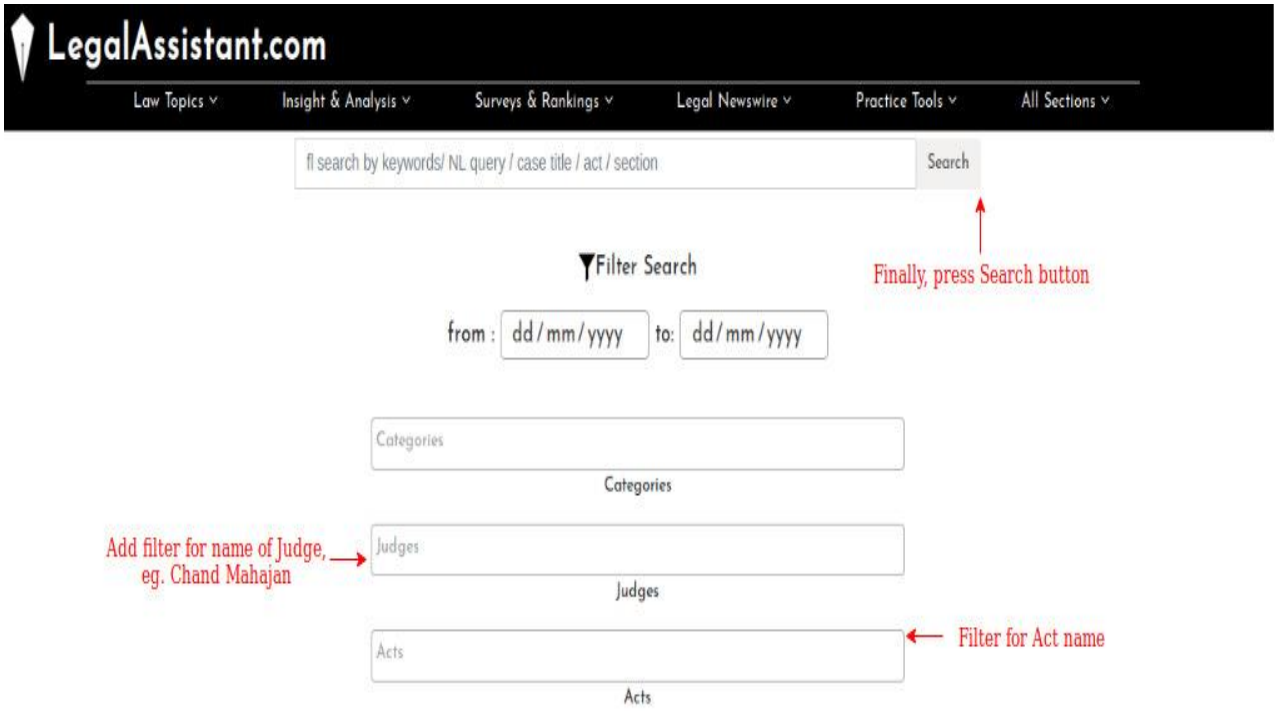


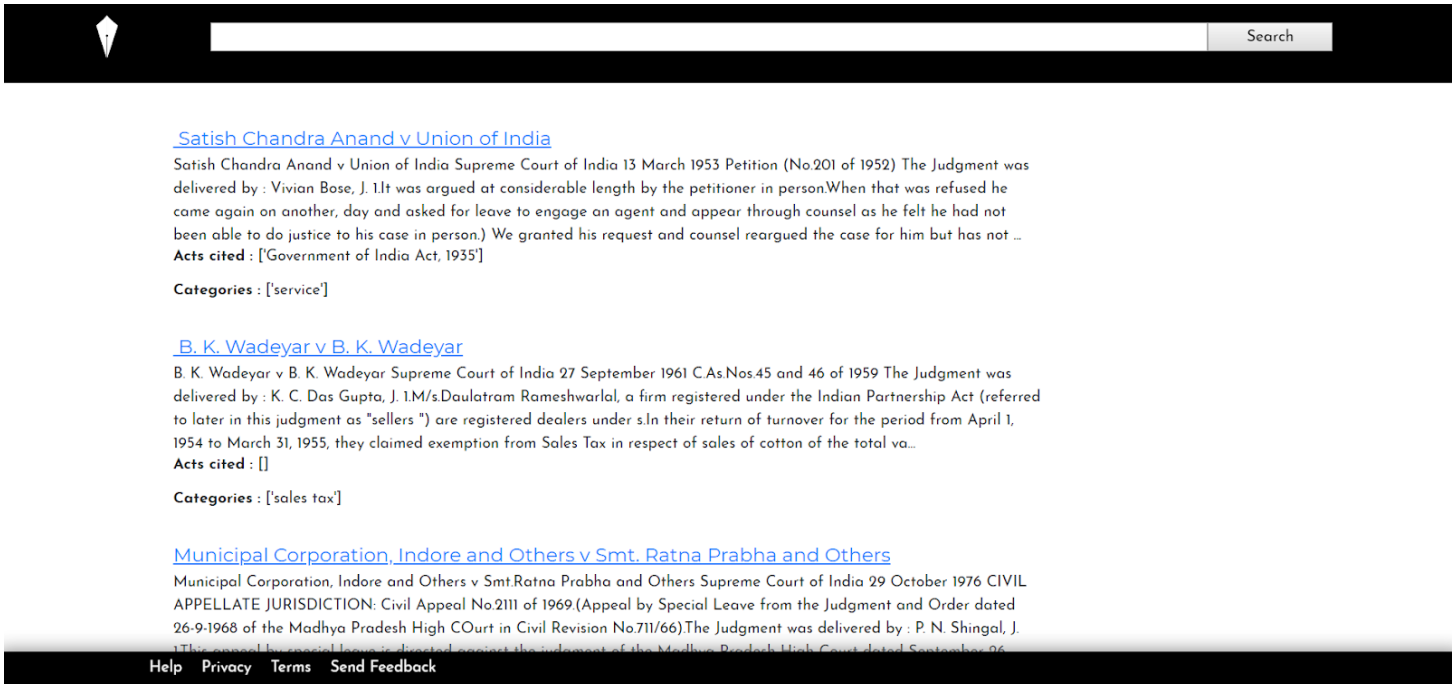
Type IV Query Data Flow

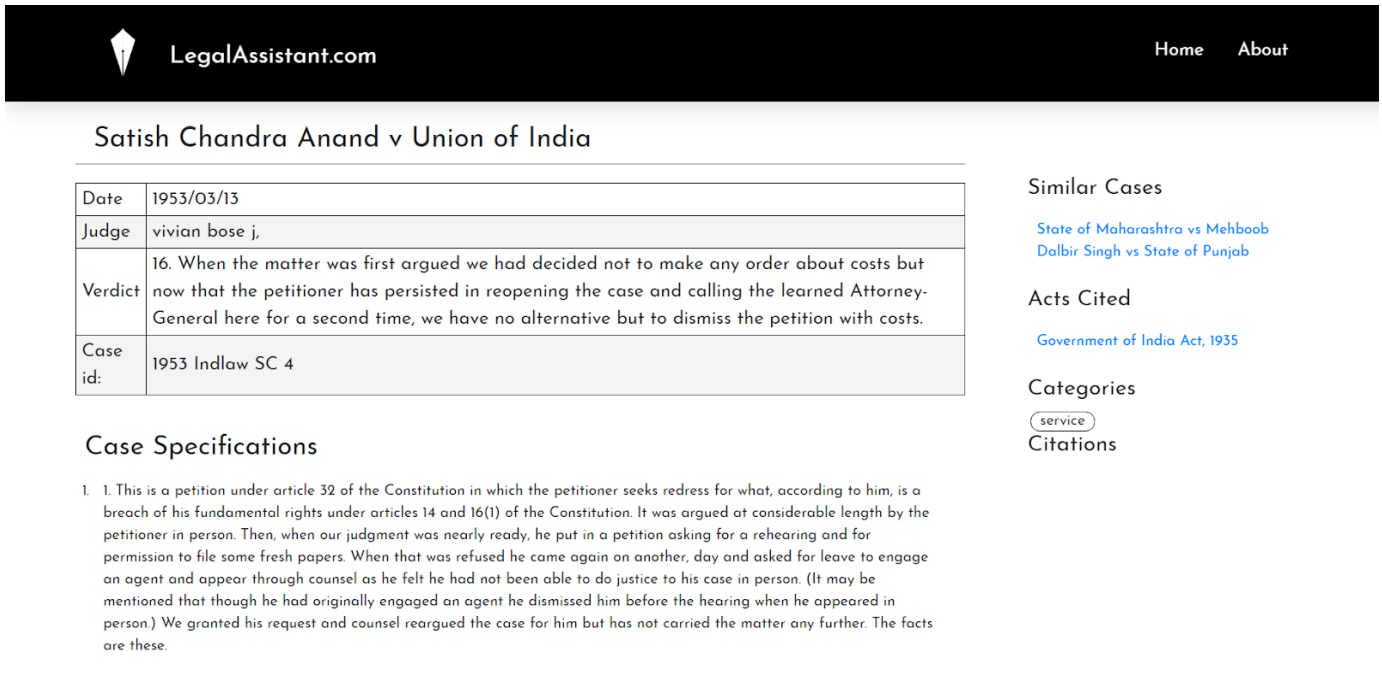


Type IV Model Architecture

6. USER INTERFACE

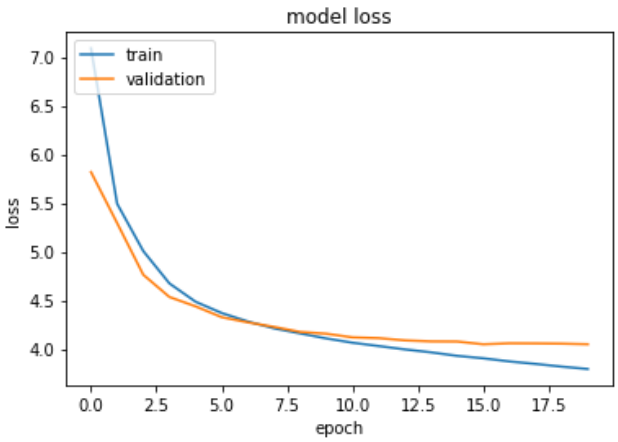






7. EXPERIMENTAL RESULTS

Attention-Based Bi-LSTM Multitasking Model :



Training Parameters :
No. of epochs = 20
Batch Size = 256
Number of Dim of LSTM Layer = 64
Number of Unique words = 10000
Learning Rate = 0.0007

8. HOW TO RUN

Unzip the code in a directory say "OpenSoft19"

```
$ pip install -r requirements.txt
```

StarWrap Installation [github.com/facebookresearch/StarSpace]
Dependency: boost, conan

```
$ git clone https://github.com/facebookresearch/StarSpace.git
$ cd StarSpace
$ make
$ cd python
$ ./build.sh

$ cp StarSpace/python/test/starwrap.so OpenSoft19/
$ cd OpenSoft19
$ export FLASK_APP=application.py
$ python application.py
```

The application would now be hosted at 0.0.0.0:8080 on the corresponding machine. To access over the network, enter IP:8080 on the browser, where IP is the public IP of the host machine.

9. REFERENCES

1. Manupatra: https://www.manupatrafast.com/
2. Westlaw india: http://www.westlawindia.com/
3. Legal crystal: https://www.legalcrystal.com/
4. Fuzzywuzzy: https://github.com/seatgeek/fuzzywuzzy
5. Bing Spell check API: https://azure.microsoft.com/en-us/services/cognitive-services/spell-check/
6. Sumy: https://pypi.org/project/sumy/
7. Case Summarizer: https://github.com/Law-AI/summarization
8. RBM: http://deeplearning.net/tutorial/code/rbm.py
9. Summarization Bot: https://www.summarizebot.com/
10. Kaggle LSTM: https://www.kaggle.com/ngyptr/lstm-sentiment-analysis-keras
11. Page rank: https://en.wikipedia.org/wiki/PageRank
12. Tf-IDF vectors for summarization: https://en.wikipedia.org/wiki/Tf-idf
13. KD tree for optimization: https://en.wikipedia.org/wiki/K-d_tree
14. Nltk: https://www.nltk.org/
15. Facebook Starspace: https://research.fb.com/downloads/starspace/
16. Network X: https://networkx.github.io/