OpenNyAI

Release 0.1

OpenNyAI Team

Sep 01, 2023

GETTING STARTED:

1 About OpenNyAI Mission 3

2 Installation 5

3 Run All 3 AI models on Input Judgment Texts 7

4 Try on Google Colab 9

5 Hosted webapp and API 11

6 Preprocessing Judgment Text 13

7 Which Legal Named Entities are extracted? 15

8 Extract Named Entities from Judgment Text 17

9 Post Processing of extracted Named Entities 19

10 Visualization of extracted Named Entities 21

11 Getting unique provisions,statutes and precedents 23

12 Storing extracted Named Entities to a file 25

13 Huggingface Models 27

14 Structuring Court Judgments using Sentence Rhetorical Roles 29

15 Predict Rhetorical Roles 33

16 Automatic Summarization of Court Judgments 35

17 Structure of Judgment Summary 37

18 Create summary of input judgment text 39

OpenNyAI library is a framework for natural language processing on Indian legal texts. This python library provides unified access to the inference of following 3 AI models developed by OpenNyAI which focus on Indian court judgments.

- Named Entity Recognition (NER)
- Judgment structuring using sentence Rhetorical Role prediction
- Extractive Summarizer

This library is mainly for running the pretrained models on your custom input judgments text. For more details about data and model training, please refer to individual git repo links.

CHAPTER

ONE

ABOUT OPENNYAI MISSION

1.1 What is OpenNyAI mission?

OpenNyai is a mission aimed at developing open source software and datasets to catalyze the creation of AI-powered solutions to improve access to justice in India.

1.2 Innovation, accelerated by collaboration

OpenNyai is an open mission enabling everyone to contribute. Our community is engaged in building datasets, models, educational materials, holding community spaces and evangelising better design and ethics for AI for Justice solutions. If you are intrigued by AI for Justice solutions and want to explore ways in which you can contribute, then please get in touch! We would love to speak to you and figure out pathways that works best for you.

1.3 Founding Collaborators

Agami, EkStep, Thoughtworks, National Law School Banglore, Rohini Nilekani Philanthropies

1.4 Contact Us

Slack

Email: support@agami.in

LinkedIn

CHAPTER

TWO

INSTALLATION

To get started using opennyai first create a new conda environment:

```
conda create -n opennyai python=3.8
conda activate opennyai
```

Install it using pip by running the following line in your terminal

```
pip install opennyai
```

2.1 For GPU support

If you want to utilize spacy with GPU please install Cupy and cudatoolkit dependency of appropriate version.

```
conda install cudatoolkit==<your_cuda_version> #### E.g. cudatoolkit==11.2
pip install cupy-cuda<your_cuda_version> ##### E.g. cupy-cuda112
```

In case of any issue with installation please refer to spacy installation with cupy

Remember you need spacy of 3.2.4 version for models to work perfectly.

CHAPTER

THREE

RUN ALL 3 AI MODELS ON INPUT JUDGMENT TEXTS

To run the 3 OpenNyAI models on judgment texts of your choice please run following python code

```
from opennyai import Pipeline
from opennyai.utils import Data
import urllib

###### Get court judgment texts on which to run the AI models
text1 = urllib.request.urlopen('https://raw.githubusercontent.com/OpenNyAI/Opennyai/
↪master/samples/sample_judgment1.txt').read().decode()
text2 = urllib.request.urlopen('https://raw.githubusercontent.com/OpenNyAI/Opennyai/
↪master/samples/sample_judgment2.txt').read().decode()
texts_to_process = [text1,text2] ### you can also load your text files directly into this
data = Data(texts_to_process)  #### create Data object for data  preprocessing before␣
↪running ML models

use_gpu = True #### If you have access to GPU then set this to True else False
###### Choose which of the AI models you want to run from the 3 models 'NER', 'Rhetorical_
↪Role','Summarizer'
pipeline = Pipeline(components = ['NER', 'Rhetorical_Role','Summarizer'],use_gpu=use_
↪gpu) #E.g. If just Named Entity is of interest then just select 'NER'
results = pipeline(data)
```

Extra parameters for Pipeline:

- **components (list):** Models that you want to run over your input judgements
- **use_gpu (bool):** Functionality to give a choice whether to use GPU for inference or not. Setting it True doesn't ensure GPU will be utilized it need proper support libraries as mentioned in documentation
- **verbose (bool):** Set it to True if you want to see progress bar/updates while processing happens
- **ner_model_name (string):** Accepts a model name of spacy as InLegalNER that will be used for NER inference available models are 'en_legal_ner_trf', 'en_legal_ner_sm'. 'en_legal_ner_trf' has best accuracy but can be slow, on the other hand 'en_legal_ner_sm' is fast but less accurate.
- **ner_mini_batch_size (int):** This accepts an int as batch size for processing of a document, if length of document is bigger that given batch size it will be chunked and then processed.
- **ner_do_sentence_level (bool):** To perform inference at sentence level or not, at sentence level it better accuracy. We recommend setting this to True.
- **ner_do_postprocess (bool):** To perform post-processing over processed doc. We recommend to set this to True.
- **ner_statute_shortforms_path (path):** It is the path of the csv file if the user wants to provide predefined short-forms to create statute clusters.The csv should have 2 columns namely 'fullforms' and 'shortforms' where 'full-

forms' contain the full name of the statute eg. 'code of criminal procedure' and shortforms contain the acronym that can be present in the judgment eg.'crpc'.Each row represents a fullform,shortform pair.

- **summarizer_summary_length (float):** Give you the functionality to choose the length of generated summary. Default is 0 which will set it to adaptive length selection. Valid input lie in range(0-1)

The predictions of each of the models is added at the sentence level.

For each of the sentence in an output,

- 'labels' provide predicted rhetorical role.
- 'in_summary' denoted if this sentence is selected in the summary.
- 'entities' provide the list of extracted named entities in that sentence

```
results[0]['annotations']
```

The AI generated summary by rhetorical roles can be accessed via

```
results[0]['summary']
```

The extracted named entities can be visualized using

```
from spacy import displacy
from opennyai.ner.ner_utils import ner_displacy_option
displacy.serve(pipeline._ner_model_output[0], style='ent', port=8080, options=ner_
↪displacy_option)
```

CHAPTER

FOUR

TRY ON GOOGLE COLAB

Open In Colab

CHAPTER FIVE

HOSTED WEBAPP AND API

All the 3 models can be run on a given judgment text and visualized using our webapp

We provide access to the hosted prediction API of the 3 models upon request.

CHAPTER

SIX

PREPROCESSING JUDGMENT TEXT

Judgment texts need to be preprocessed before running the AI models

6.1 Preprocessing Activities:

Following preprocessing activities are performed using spacy pretrained model

1. Separating preamble from judgment text
2. Sentence splitting of judgment text
3. Convert upper case words in preamble to title case
4. Replace newline characters within a sentence with space in judgment text

The preprocessing is done using Data object.

```
texts_to_process = [text1,text2]
data = Data(texts_to_process,preprocessing_nlp_model='en_core_web_trf')
```

The preprocessing is lazy evaluated.

6.2 Trade-Off between Preprocessing Accuracy and Run Time

One can choose which spacy pretrained model to use for preprocessing while creating Data object using parameter *preprocessing_nlp_model*. The choice of preprocessing model critically determines the performance of AI models. We recommend using 'en_core_web_trf' for preprocessing of the data, but it can be slow. Available preprocessing models are 'en_core_web_trf' (slowest but best accuracy), 'en_core_web_md', 'en_core_web_sm'(fastest but less accurate)

6.3 Additional Parameters while creating Data object

- mini_batch_size (int): This accepts an int as batch size for processing of a document, if length of document is bigger that given batch size it will be chunked and then processed.
- use_gpu (bool): Functionality to give a choice whether to use GPU for processing or not Setting it True doesn't ensure GPU will be utilized it need proper support libraries as mentioned in documentation
- use_cache (bool): Set it to true if you want to enable caching while preprocessing. Always set this to True.
- verbose (bool): Set it to if you want to see progress bar while processing happens
- file_ids (list): List of custom file ids to use for documents

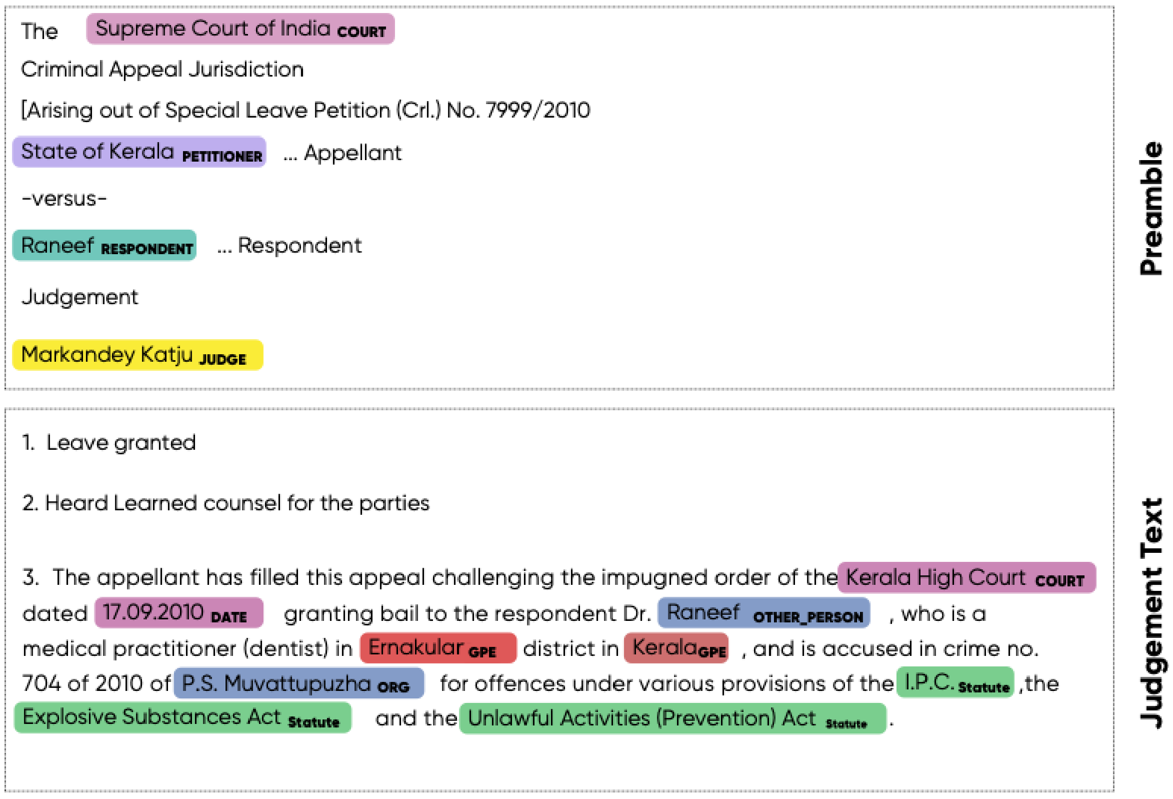
CHAPTER

SEVEN

WHICH LEGAL NAMED ENTITIES ARE EXTRACTED?

Named Entities Recognition is commonly studied problem in Natural Language Processing and many pre-trained models are publicly available. However legal documents have peculiar named entities like names of petitioner, respondent, court, statute, provision, precedents, etc. These entity types are not recognized by standard Named Entity Recognizer like spacy. Hence there is a need to develop a Legal NER model. Due to peculiarity of Indian legal processes and terminologies used, it is important to develop separate legal NER for Indian court judgment texts.

Some entities are extracted from Preamble of the judgements and some from judgement text. Preamble of judgment contains formatted metadata like names of parties, judges, lawyers,date, court etc. The text following preamble till the end of the judgment is called as the “judgment”. Below is an example



The Supreme Court of India COURT

Criminal Appeal Jurisdiction

[Arising out of Special Leave Petition (Crl.) No. 7999/2010

State of Kerala PETITIONER ... Appellant

-versus-

Raneef RESPONDENT ... Respondent

Judgement

Markandey Katju JUDGE

1. Leave granted

2. Heard Learned counsel for the parties

3. The appellant has filled this appeal challenging the impugned order of the Kerala High Court COURT dated 17.09.2010 DATE granting bail to the respondent Dr. Raneef OTHER_PERSON , who is a medical practitioner (dentist) in Ernakular GPE district in Kerala GPE , and is accused in crime no. 704 of 2010 of P.S. Muvattupuzha ORG for offences under various provisions of the I.P.C. Statute ,the Explosive Substances Act Statute and the Unlawful Activities (Prevention) Act Statute .

Following legal entities are extracted from input court judgment.

Named Entity	Extract From	Description	
COURT	Preamble, Judgment	Name of the court which has delivered the current judgement if extracted from Preamble. Name of any court mentioned if extracted from judgment sentences.	
PETITIONER	Preamble, Judgment	Name of the petitioners / appellants /revisionist from current case	
RESPONDENT	Preamble, Judgment	Name of the respondents / defendents /opposition from current case	
JUDGE	Premable, Judgment	Name of the judges from current case if extracted from preamble. Name of the judges of the current as well as previous cases if extracted from judgment sentences.	
LAWYER	Preamble	Name of the lawyers from both the parties	
DATE	Judgment	Any date mentioned in the judgment	
ORG	Judgment	Name of organizations mentioned in text apart from court. E.g. Banks, PSU, private companies, police stations, state govt etc.	
GPE	Judgment	Geopolitical locations which include names of countries,states,cities, districts and villages	
STATUTE	Judgment	Name of the act or law mentioned in the judgement	
PROVISION	Judgment	Sections, sub-sections, articles, orders, rules under a statute	
PRECEDENT	Judgment	All the past court cases referred in the judgement as precedent. Precedent consists of party names + citation(optional) or case number (optional)	
CASE_NUMBER	Judgment	All the other case numbers mentioned in the judgment (apart from precedent) where party names and citation is not provided	
WITNESS	Judgment	Name of witnesses in current judgment	
OTHER_PERSON	Judgment	Name of the all the person that are not included in petitioner,respondent,judge and witness	

More detailed definitions with examples can be found here For more details about training data and code used for training , please refer to legal_NER git repo.

CHAPTER

EIGHT

EXTRACT NAMED ENTITIES FROM JUDGMENT TEXT

Use following python to extract entities from single court judgment. For running all 3 AI models together on input text, please refer *here*.

```
import opennyai.ner as InLegalNER
from opennyai import Pipeline
from opennyai.utils import Data
import urllib

###### Get court judgment texts on which to run the AI models
text1 = urllib.request.urlopen('https://raw.githubusercontent.com/OpenNyAI/Opennyai/
↪master/samples/sample_judgment1.txt').read().decode()
text2 = urllib.request.urlopen('https://raw.githubusercontent.com/OpenNyAI/Opennyai/
↪master/samples/sample_judgment2.txt').read().decode()
texts_to_process = [text1,text2] ### you can also load your text files directly into this
data = Data(texts_to_process)  #### create Data object for data  preprocessing before␣
↪running ML models

pipeline = Pipeline(components=['NER'], use_gpu=use_gpu, verbose=True,ner_model_name='en_
↪legal_ner_trf',
             ner_mini_batch_size=40000, ner_do_sentence_level=True, ner_do_
↪postprocess=True,
             ner_statute_shortforms_path='')

results = pipeline(data)

json_result_doc_1 = results[0]

ner_doc_1 = pipeline._ner_model_output[0]

identified_entites = [(ent, ent.label_) for ent in ner_doc_1.ents]
```

Output of NER model is a spacy doc and identified_entities is list of entities extracted.

```
[(Section 319, 'PROVISION'),
 (Cr.P.C., 'STATUTE'),
 (G. Sambiah, 'RESPONDENT'),
 (20th June 1984, 'DATE')]
```

8.1 Important parameters while loading NER model

- ner_model_name (string): Accepts a model name of spacy as InLegalNER that will be used for NER inference available models are 'en_legal_ner_trf', 'en_legal_ner_sm'. 'en_legal_ner_trf' has best accuracy but can be slow, on the other hand 'en_legal_ner_sm' is fast but less accurate.
- use_gpu (bool): Functionality to give a choice whether to use GPU for inference or not. Setting it True doesn't ensure GPU will be utilized it need proper support libraries as mentioned in documentation

8.2 Important parameters while inferring NER model

- ner_do_sentence_level (bool): To perform inference at sentence level or not, at sentence level it better accuracy. We recommend setting this to True.
- ner_do_postprocess (bool): To perform post-processing over processed doc. We recommend to set this to True.
- ner_statute_shortforms_path(path):It is the path of the csv file if the user wants to provide predefined shortforms to create statute clusters.The csv should have 2 columns namely 'fullforms' and 'shortforms' where 'fullforms' contain the full name of the statute eg. 'code of criminal procedure' and shortforms contain the acronym that can be present in the judgment eg.'crpc'.Each row represents a fullform,shortform pair.
- ner_mini_batch_size (int): This accepts an int as batch size for processing of a document, if length of document is bigger that given batch size it will be chunked and then processed.
- verbose (bool): Set it to if you want to see progress bar while processing happens

CHAPTER

NINE

POST PROCESSING OF EXTRACTED NAMED ENTITIES

Since the document level context was not used duiring annotation,it is important to capture the document level context while inference. This can be done via postprocessing using rules.

To perform postprocessing on the extracted entities specify *ner_do_postprocess=True*. The key *normalized_name* of each entity contains the output of postprocessing.

The postprocessing is done on these entities:

1. *Precedents*: Same precedent can be written in multiple forms in a judgment. E.g. with citation,without citation,only petitioner's name supra etc.For eg. 'darambir vs state of maharashtra 2016 AIR 54','darambir vs state of maharashtra 'and'darambir's case(supra)' all refer to the same case.All the precedents referring to the same case are clustered together and the longest precedent in the cluster is the head of the cluster.The output is a dict where the keys are the head of the cluster (longest precedent) and value is a list of all the precedents in that cluster. To access the list, use

ner_doc_1.user_data['precedent_clusters']

For example

[{Madhu Limaye v. State of Mahrashtra: [Madhu Limaye v. State of Mahrashtra, Madhu Limaye v. State of Maharashtra, Madhu Limaye, Madhu Limaye, Madhu Limaye]}]

2. *Statute*: In a judgment,sometimes aconyms are used instead of the complete statute name.For eg.section 147 of IPC,section 148 of Penal code is mentioned instead of Indian Penal code.We have incorporated the acronyms for some well known statutes such as IPC,CrPC,Income Tax act,Motor vehicles act,sarfaesi etc.All the statutes which are a short form of any of these well known statute belongs to the same cluster.For eg I.P.C,IPC,Penal code will belong to the same cluster with head as "Indian Penal code". Many a times,the way a statute is referred within a judgment is explicitly mentioned .For eg. Motor Vehicle Act(herein referred as MV act). So,every mention of MV act would belong to the same cluster with head as "Motor Vehicle Act". .It can be used by:

ner_doc_1.user_data['statute_clusters']

For example: { 'Criminal Procedure Code': [Code of Criminal Procedure,Crpc] }

3. *Provision-Statute*: Every provision should have an associated statute.Sometimes the provision is followed by the statute it belongs to and sometimes the corresponding statutes are not mentioned explicitly .To find statutes for these implicit provisions,we search the judgment if the same provision is mentioned elsewhere along with the statute,if present we assign the same statute to the implicit provision.If not,the nearest statute prior to the provision is assigned to that provision after some validations.The statutes assogned are then normalised using the statute clusters The output is a list of named tuples, each tuple contains provision-statute-normalised provision-normalised statutes text eg. (362,IPC,'Section 362','Indian Penal Code') .It can be used by:

ner_doc_1.user_data['provision_statute_pairs']

For example [(Section 369, Crpc, 'Section 369','Criminal Procedure Code'), (Section 424, Crpc, 'Section 424','Criminal Procedure Code')]

4. *Other person/Org* : Same entities can be tagged with different classes in different sentences of the same judgment due to sentence level context. E.g. 'Amit Kumar' can be a petitioner in the preamble but later in the judgment is marked

as ‘other_person’. So,we reconcile these entities based on their relative importance i.e. ‘Amit Kumar’ will be marked as petitioner in the whole judgment.

CHAPTER

TEN

VISUALIZATION OF EXTRACTED NAMED ENTITIES

To visualize the NER result on single judgment text please run

```
from spacy import displacy
from opennyai.ner.ner_utils import ner_displacy_option
displacy.serve(ner_doc_1, style='ent',port=8080,options=ner_displacy_option)
```

Please click on the link displayed in the console to see the annotated entities.

CHAPTER

ELEVEN

GETTING UNIQUE PROVISIONS,STATUTES AND PRECEDENTS

1. To get a list of unique precedents within a judgment:

```
from opennyai.ner import get_unique_precedent_count
precedents=InLegalNER.get_unique_precedent_count(ner_doc_1)
```

It will return a dictionary with name of the precedents as keys and number of times they occured as values.
For eg. State of Punjab v. Phil and Anr: [State of Punjab v. Phil Rani and Anr, Phil]

2. To get frequency count of all the provisions within a judgment:

```
from opennyai.ner import get_unique_provision_count
provisions=get_unique_provision_count(ner_doc_1)
```

It will return a dictionary with name of the provisions as keys and number of times they occured as values.
For eg.{'Article 226 of Constitution': 11, 'Article 227 of Constitution': 12}

3. To get frequency count of all the statutes within a judgment:

```
from opennyai.ner import get_unique_statute_count
statutes=get_unique_statute_count(ner_doc_1)
```

It will return a dictionary with name of the statutes as keys and number of times they occured as values.
For eg.{'Constitution': 30, 'Criminal Procedure Code': 77, 'Indian Penal Code': 13}

CHAPTER

TWELVE

STORING EXTRACTED NAMED ENTITIES TO A FILE

1. To save result in csv file with linked entities :

```
from opennyai.ner import get_csv
get_csv(ner_doc_1,file_name,save_path):
```

In the created csv,it will have 4 columns namely:

'file_name': name of the file/judgment

'entity': The entity found in the judgment .For eg.'section 482' ,'constiution','sibbia vs ajay'

'label': The label associated with each entity .For eg. label of 'section 482' would be 'provision'

'normalised entities': Entities including provision,statute and precedents are normalised as follows:

1.'Provision': Each provision is normalised by adding the statute associated with it alongside. For eg.'section 147' is normalised to 'Section 147 of Indian Penal Code'

2.'Statute': Each statute is normalised by adding its full form if present .For eg.'IPC' is normalised to 'Indian Penal Code'

3.'Precedent': Each precedent is normalised by checking if the particular precedent is mentioned elsewhere in the judgment and is longer than the current precent(has citations,full names etc.). For eg. normalised entity for 'amber v. State of Haryana' would be 'amber v. State of Haryana R.C.R. (Crl.)2007'

CHAPTER

THIRTEEN

HUGGINGFACE MODELS

These models are also published on huggingface

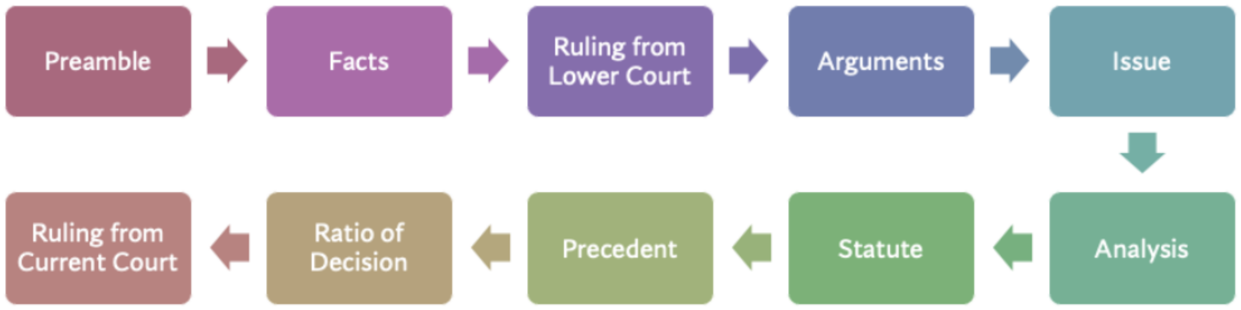
en_legal_ner_trf and en_legal_ner_sm

CHAPTER

FOURTEEN

STRUCTURING COURT JUDGMENTS USING SENTENCE RHETORICAL ROLES

Indian Court Judgements have an inherent structure which is not explicitly mentioned in the judgement text. Assigning rhetorical roles to the sentences provides structure to the judgements. This is an important step which will act as building block for developing Legal AI solutions. Though there is no prescription for writing judgement,a judgement text follows an inherent structure. For example, a judgement text would begin with preamble, state facts of the case, courts analysis of the arguments from respondents and petitioners etc. Typical structure of an Indian court judgement is as shown below. The flow is not linear and these roles can appear in any sequence.



The detailed definitions of each of the rhetorical roles is specified below

Rhetorical Role	Rhetorical Roles (sentence level)
Preamble (PREAMBLE) A	typical judgement would start with the court name, the details of parties, lawyers and judges' names, Headnotes. This section typically would end with a keyword like (JUDGEMENT or ORDER etc.) Some supreme court cases also have HEADNOTES, ACTS section. They are also part of Preamble.
Facts(FAC)	This refers to the chronology of events (but not judgement by lower court) that led to filing the case, and how the case evolved over time in the legal system (e.g., First Information Report at a police station, filing an appeal to the Magistrate, etc.) Depositions and proceedings of current court Summary of lower court proceedings
Ruling by Lower Court (RLC)	Judgments given by the lower courts (Trial Court, High Court) based on which the present appeal was made (to the Supreme Court or high court). The verdict of the lower Court, Analysis & the ratio behind the judgement by the lower Court is annotated with this label.
Issues (ISSUE)	Some judgements mention the key points on which the verdict needs to be delivered. Such Legal Questions Framed by the Court are ISSUES. E.g. "he point emerge for determination is as follow:- (i) Whether on 06.08.2017 the accused persons in furtherance of their common intention intentionally caused the death of the deceased by assaulting him by means of axe ?"
Argument by Petitioner (ARG_PETITIONER)	Arguments by petitioners' lawyers. Precedent cases argued by petitioner lawyers fall under this but when court discusses them later then they belong to either the relied / not relied upon category. E.g. "learned counsel for petitioner argued that . . . "
Argument by Respondent (ARG_RESPONDENT)	Arguments by respondents lawyers. Precedent cases argued by respondent lawyers fall under this but when court discusses them later then they belong to either the relied / not relied upon category. E.g. "learned counsel for the respondent argued that . . . "
Analysis (ANALYSIS)	Courts discussion on the evidence,facts presented,prior cases and statutes. These are views of the court. Discussions on how the law is applicable or not applicable to current case. Observations(non binding) from court. It is the parent tag for 3 tags: PRE_RLEIED, PRE_NOT_RELIED and STATUTE i.e. Every statement which belong to these 3 tags should also be marked as ANALYSIS E.g. "Post Mortem Report establishes that .. " E.g. "In view of the abovementioned findings, it is evident that the ingredients of Section 307 have been made out . . . ."
Statute (STA)	Text in which the court discusses Established laws, which can come from a mixture of sources – Acts , Sections, Articles, Rules, Order, Notices, Notifications, Quotations directly from the bare act, and so on. Statute will have both the tags Analysis + Statute E.g. "Court had referred to Section 4 of the Code, which reads as under: "4. Trial of offences under the Indian Penal Code and other laws.– (1) All offences under the Indian Penal Code (45 of 1860) shall be investigated, inquired into, tried, and otherwise dealt with according to the provisions hereinafter contained"
Precedent Relied (PRE_RELIED)	Sentences in which the court discusses prior case documents, discussions and decisions which were relied upon by the court for final decisions. So Precedent will have both the tags Analysis + Precedent E.g. This Court in Jage Ram v. State of Haryana3 held that: "12. For the purpose of conviction under Section 307 IPC, ….. "
Precedent Not Relied (PRE_NOT_RELIED)	Sentences in which the court discusses prior case documents, discussions and decisions which were not relied upon by the court for final decisions. It could be due to the fact that the situation in that case is not relevant to the current case. E.g. This Court in Jage Ram v. State of Haryana3 held that: "12. For the purpose of conviction under Section 307 IPC,
Ratio of the deci-	Main Reason given for the application of any legal principle to the legal issue. This is the result of the analysis by the court. This typically appears right before the final decision. This is not the same as "Ratio Decidendi" taught in the Legal Academic cur-

For more details about how the data was collected and model training , please refer to the paper and git repo.

CHAPTER

FIFTEEN

PREDICT RHETORICAL ROLES

Use following python to get structure of 2 court judgments using sentence rhetorical roles. For running all 3 AI models together on input text, please refer *here* .

```
from opennyai import RhetoricalRolePredictor
from opennyai.utils import Data
import urllib

###### Get court judgment texts on which to run the AI models
text1 = urllib.request.urlopen('https://raw.githubusercontent.com/OpenNyAI/Opennyai/
↪master/samples/sample_judgment1.txt').read().decode()
text2 = urllib.request.urlopen('https://raw.githubusercontent.com/OpenNyAI/Opennyai/
↪master/samples/sample_judgment2.txt').read().decode()
texts_to_process = [text1,text2] ### you can also load your text files directly into this
data = Data(texts_to_process)  #### create Data object for data  preprocessing before␣
↪running ML models

pipeline = Pipeline(components=['Rhetorical_Role'], use_gpu=use_gpu, verbose=True)

results = pipeline(data)

json_result_doc_1 = results[0]
```

CHAPTER

SIXTEEN

AUTOMATIC SUMMARIZATION OF COURT JUDGMENTS

Court judgments can be very long and it is a common practice for legal publishers to create headnotes of judgments. E.g. sample headnote. The process of creating headnotes is manual and based on the certain rules and patterns. With advances in Artifical Intelligence, we can create automatically summaries of long text and then an expert to correct it to create final summary. This will drastically reduce the time needed for creation of headnotes and make the process more consistent. AI model can also learn from the feedback given by the expert and keep on improving the results.

CHAPTER

SEVENTEEN

STRUCTURE OF JUDGMENT SUMMARY

While standard way of writing headnotes captures the important aspects of the judgement like HELD, experts believe that it is not the best style of writing summaries. E.g. it is difficult to establish if the facts of a new case are similar to facts of an old case by reading headnotes of the old case.

So we have come up with revised structure of writing summaries. Summary will have 5 sections Facts summary, Arguments summary, Issue summary, Analysis Summary and Decision Summary. Leveraging our previous work on structuring court judgements, we can automatically predict Rhetorical Roles for each sentence and then create this sectionwise summary. The following table shows which rhetorical roles to expect in each of the summary sections.

Summary Section	Rhetorical Roles
Facts	Facts, Ruling by Lower Court
Issue	Issues
Arguments	Argument by Petitioner, Argument by Respondent
Analysis	Analysis, Statute, Precedent Relied, Precedent Not Relied, Ratio of the decision
Decision	Ruling by Present Court

We believe this structure of writing summaries is better suited for Legal Research and Infomation Extraction. This will also improve the readability of the summaries.

17.1 Extractive summarization using Rhetorical Roles

There are two styles of creating summaries viz. Extractive & Abstractive. Extractive summaries pick up important sentences as-is and put them in order for creating final summary. Abstractive summarization on the other hand paraphrases the important information to create crisp summary in its own words. While abstractive summaries are more useful, they are harder to create and evaluate. Hence, as first step we focus on extractive summarization which will pick up most important sentences and arrange them in the structure described above. Once this task is done correctly, then we can focus on the abstractive summarization

17.2 Which rhetorical roles are summarized?

We empirically found out that “Issues” and “Decision” written in original judgement are very crisp and rich in information. So we do not try to summarize them. We carry forward all the sentences with these 2 roles directly into the summary. “Preamble” is important in setting the context of case and also copied to summary. For remaining rhetorical roles, we rank the sentences in descending order of importance as predicted by the AI model and choose the top ones as described in section 5.

CHAPTER

EIGHTEEN

CREATE SUMMARY OF INPUT JUDGMENT TEXT

Summarizer model needs Rhetorical Role model output as input. Hence Rhetorical Role prediction model needs to run before Summarizer model rune.

To use Summarizer model simply execute code below. For running all 3 AI models together on input text, please refer *here* .

```
from opennyai import Pipeline
from opennyai.utils import Data
import urllib

###### Get court judgment texts on which to run the AI models
text1 = urllib.request.urlopen('https://raw.githubusercontent.com/OpenNyAI/Opennyai/
↪master/samples/sample_judgment1.txt').read().decode()
text2 = urllib.request.urlopen('https://raw.githubusercontent.com/OpenNyAI/Opennyai/
↪master/samples/sample_judgment2.txt').read().decode()
texts_to_process = [text1,text2] ### you can also load your text files directly into this
data = Data(texts_to_process)  #### create Data object for data  preprocessing before␣
↪running ML models

pipeline = Pipeline(components=['Rhetorical_Role', 'Summarizer'], use_gpu=use_gpu,␣
↪verbose=True, summarizer_summary_length=0.0)

results = pipeline(data)

json_result_doc_1 = results[0]
summaries_doc_1 = results[0]['summary']
```

Result:

```
{'id': 'ExtractiveSummarizer_xxxxxxx]',
  'summaries': {'facts': 'xxxx',
  'arguments': 'xxxx',
  'ANALYSIS': 'xxxx',
  'issue': 'xxxx',
  'decision': 'xxxx',
  'PREAMBLE': 'xxxx'}]
```