A Project Report

On

Institutional Grammar

BY

Dhruv Merchant

2020A7PS2063H

Under the supervision of

Dr. Prajna Devi Upadhyay

SUBMITTED IN COMPLETE FULFILMENT OF THE REQUIREMENTS OF

CS F266: STUDY PROJECT

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI (RAJASTHAN)

HYDERABAD CAMPUS

(May 2023)

ACKNOWLEDGMENTS

I am indebted to Dr. Prajna Devi Upadhyay, my project mentor, for her constant guidance and feedback on my work.

I sincerely thank the Department of Computer Science and Information Systems (CSIS), BITS Pilani Hyderabad Campus, for providing me with this excellent opportunity to gain valuable insights and learn new things via this Formal Study Project.

I am indebted to my parents, teachers, and friends, as without their prayers and best wishes, I wouldn’t have reached this far.

My thanks and appreciations also extend to my colleagues in developing the project and people who have willingly helped me out with their abilities.

Birla Institute of Technology and Science-Pilani,

Hyderabad Campus

CERTIFICATE

This is to certify that the project report entitled "*Automating tagging using Institutional Grammar for Indian Law*" submitted by Mr. Dhruv Merchant (ID No. 2020A7PS2063H) in complete fulfilment of the requirements of the course CS F266, Study Project Course, embodies the work done by him under my supervision and guidance.

Date: 20/05/2023

(Dr. Prajna Devi Upadhyay)

BITS- Pilani, Hyderabad Campus

ABSTRACT

Institutional statements are primarily of two types – Regulatory and Non-regulatory.

Regulative statements describe actors' duties and discretion linked to specific actions within certain contextual parameters.

(Institutional Grammar 2.0 Code Book Christopher K. Frantz Saba N. Siddiki)

Regulatory statements for Indian Laws can be broken down into several parts such as Attribute, Aim, Deontic and Context, a set of syntactic components that each describe some unique information to give the sentence a meaning.

Our aim in this project is to use natural language processing and information retrieval techniques to automate the task of tagging statements from Indian Law. The tagging follows the principles of Institutional Grammar **on regulatory statements**.

We test several heuristics to obtain results that can give accurate results over all the Indian Laws based on certain set of assumptions and conditions explained further.

Thus, each law will be split into multiple entities as mentioned above (Aim, Attribute, Deontic and Context).

We have used tools such as Spacy to find patterns using their dependency graphs to explain the relationship between the entities of importance.

CONTENTS

INTRODUCTION

The project explained in the report is to work on Institutional grammar for Indian Law statements and extract syntactic value from the labels being used.
For our purpose we have worked on regulatory statements, composed of several components, although the labels of importance are the *"Aim", "Attribute", and "Deontic".*

"An institutional statement describes expected actions for actors within the presence or absence of particular constraints, or parameterizes features of an institutional system."
(Institutional Grammar 2.0 Code Book Christopher K. Frantz Saba N. Siddiki page 3)

Our dataset consists of Indian Law statements, which constitute of a give syntactical format according to rules provided in the "*Institutional Grammar 2.0 Codebook*".
Since we are extracting information by performing multiple heuristics on the given dataset of Indian laws, we make use of regulatory statements since they can be used for coding purposes, to identify and understand the relations between different labels mentioned above.

We have mainly performed two different heuristics, each of which will be explained further on in the report.

An example to show how the statements are annotated or tagged:
Annotated examples –

Statement:
The Central Government may, by notification in the Official Gazette establish one or more environmental laboratories:

Tags after classification:
Aim: establish
Attribute: The Central Government
Deontic: may

(https://labour.gov.in/)

RELATED WORK

Working on regulatory statements to label them with certain “specific” tags is essentially a sequence labelling task. Since the goal of the sequence labelling task is to find the inherent structure or properties of the input, it provides meaningful information from the input data.

The python library spaCy has an in-built Named Entity Recognition (NER) tool which is trained on large amounts of data and thus can be used as an important tool to recognise a variety of labels from its pretrained models. Although, one downside to it is it detects labels such as person, organizations, locations, dates which is too general.

What the project aims, is to be more specific to the domain of Indian Law statements. The pre-trained models provided by spaCy are trained on a combination of general-purpose text data from various sources, including news articles, web pages and more. Some of these might add irrelevant information the one needed for our own use as it is too general and not domain specific thus using spaCy directly might not give us useful results.

We have implemented a heuristic based approach towards this problem as training a model will require huge amount of data which is unavailable to us. So, we use pretrained models from spaCy and build certain heuristics based on our observations and multiple patterns.

We generate silver data for training by applying our heuristics then use that for re-training or fine tuning spaCy based on observed results.

Some well-known sequence labelling methods and uses:

1. “Bidirectional LSTM-CRF Models for Sequence Tagging" by Zhiheng Huang, Wei Xu, and Kai Yu: https://arxiv.org/abs/1508.01991

2. “End-to-End Sequence Labeling via Bi-directional LSTM-CNNs-CRF" by Xuezhe Ma and Eduard Hovy: https://arxiv.org/abs/1603.01354

3. "Neural Architectures for Named Entity Recognition" by Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer:

   https://arxiv.org/abs/1603.01360

OUR METHODS - HEURISTICS AND PROCEDURE

1. General Overview

We have several tags, each relating to a specific entity described in the Law statement.

(Refer page 6 of the report)

The "*Attribute*" corresponds to the actor in the statement. This generally is a noun that occurs in our Law statement thus is the subject in the statement, to whom the Law is related to.

The "*Aim*" is the action which is associated with the Attribute or actor. Thus, the focal action of this statement is identified by this particular entity of the statement.

The "*Deontic*" is an auxiliary verb that is related to the Aim or the action of the statement. It indicates varying level of prescriptive force of the Aim of the statement. Common examples include – shall, must, may.

The "Context" more generally explains the setting of the statement. It further elaborated on the context clauses and explains the Execution Constraint on its Activation Condition.

The Activation Condition is a set of rules or conditions that are adhered to, in order for the aim to work or function.

(Institutional Grammar 2.0 Code Book Christopher K. Frantz Saba N. Siddiki page 3 and 42)

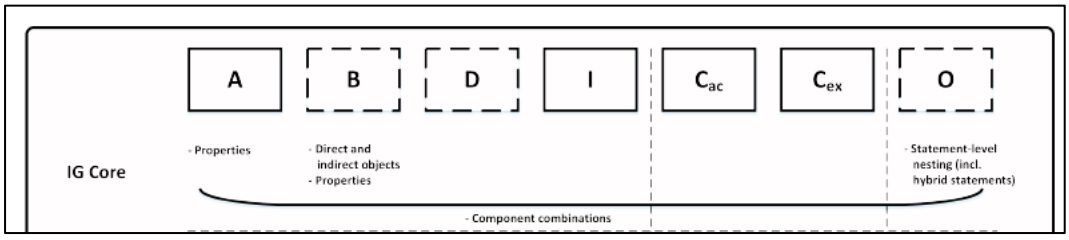


Image 1.1 (Institutional Grammar 2.0 Code Book Christopher K. Frantz Saba N. Siddiki page 42)

Image 1.1 indicates the tags that a statement can be broken down into.

For our project, we are focussing on tags “A”, “D”, “I” and “C” each corresponding to Attribute, Deontic, Aim and Context respectively.

Note that deontic is highlighted in a dashed box, implying it is a sufficient component (optional). The rest three are regulative components and must be identified.

In Image 1.1, the context is broken into two parts as well, as mentioned before – the activation condition and the execution constraint for the given activation condition.

2. Examples describing methodology

Annotated examples –

Statement:

The Central Government may, by notification in the Official Gazette establish one or more environmental laboratories:

Tags after classification:

Aim: establish

Attribute: The Central Government

Deontic: may

Statement:

When an offence under this Act has been committed by any Department of Government, the Head of Department shall be deemed to be guilty of the offence and shall be liable to proceeded against and punished accordingly.

Tags after classification:

Aim: deemed

Attribute: Head of Department

Deontic: shall

The Central Government may by notification in the Official : Gazette, appoint or recognise such persons as it thinks fit and having the prescribed qualifications to be Government Analysts for the purpose of analysis-of samples of air, water, soil or other substance sent for analysis to any environmental laboratory established or recognised under sub-section (1) of section 12.	The Central Government	appoint or recognise	may
Subject to the provisions of sub-section (2), the provisions of this Act and the rules or orders made there in shall have effect not withstanding anything inconsistent there with contained in any enactment other than this Act.	provisions of Act	have effect	shall
Where any offence under this Act has been committed by a company, very person who, at the time the offence was committed, was directly in charge of, and was responsible to, the company for the conduct of the business of the company, as well as the company, shall be deemed to be guilty of the offence and:shall be liable to be proceeded against. and punished accordingly: Provided that nothing contained in this sub-section shall render any-such person liable to any punishment provided in this Act, if he proves that the offence was committed without his knowledge or that he exercised all due diligence to prevent the commission of such offence	The company	be deemed to be guilty	shall
Every rule made under this Act shall be laid, as soon as maybe after it is made, before each House of Parliament, while it is in session, for a total period of thirty days which may be comprised in one Session or in two or more successive sessions, and if, before the expiry of the session immediately following the session or the successive sessions aforesaid. both Houses agree in making any modification in the rule or both Houses agree that the rule should not be made, the rule shall thereafter have effect only in such modified form or be of no effect, as the case may be; so, however, that any such modification or annulment shall be without prejudice to the validity of anything previously done under that rule.	Rule in act	be laid	shall

Image 1.2 from Gold Standard

Image 1.2 shows more examples of tagging the Law statements.

The tags are in order of Attribute, Aim and Deontic respectively. Thus, our aim is to rightly classify these tags based on certain parameters and assumptions which will be discussed.

As it can be seen, the sentence structure is a complex sentence involving multiple clauses in a single statement. Thus, we identify if a sentence is clausal and break it into different clauses and run our heuristics on each one of the given sentences or Laws.

3. Clausal and Non-clausal sentence structures

In our first heuristic, we classified statements as clausal and non-clausal based on the activation condition in their Context tag.

Clausal sentences contain an activation condition such as an “if / when” as opposed to non-clausal sentence structures. Although there could be other complex sentence structures consisting of more than one (independent clause) and thus be clausal too without an activation condition of “if / when”.

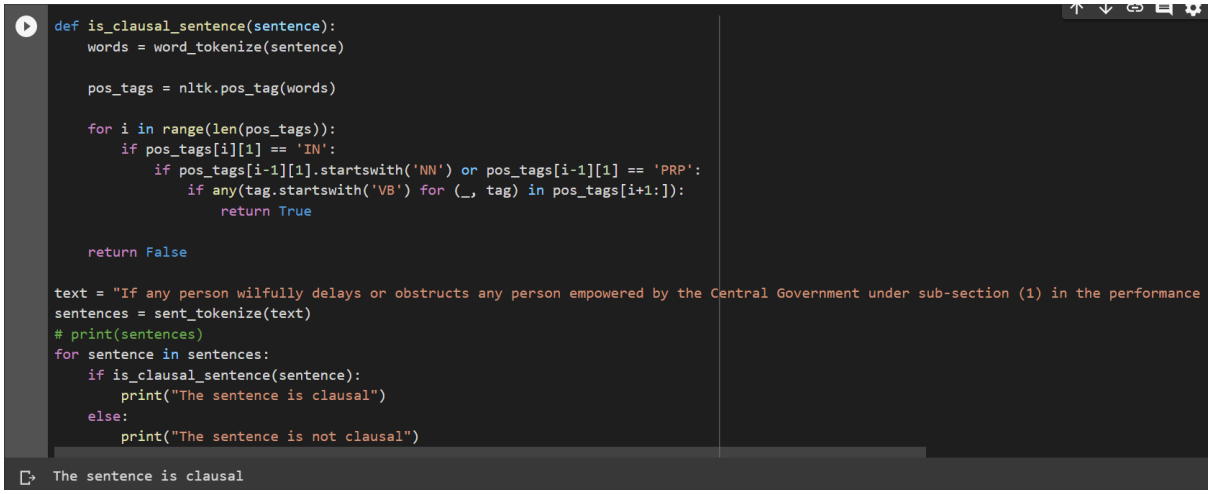


Image 1.3 – code snippet for clausal analysis part

The input statement for image 1.3 is “If any person wilfully delays or obstructs any person empowered by the Central Government under sub-section (1) in the performance of his functions, he shall be guilty of an offence under this act”.

Note that the “if” clause in the statement is followed by an activation condition that makes the statement clausal. This is also our underlying assumption that “if / when” statements lead to a clausal sentence structure.

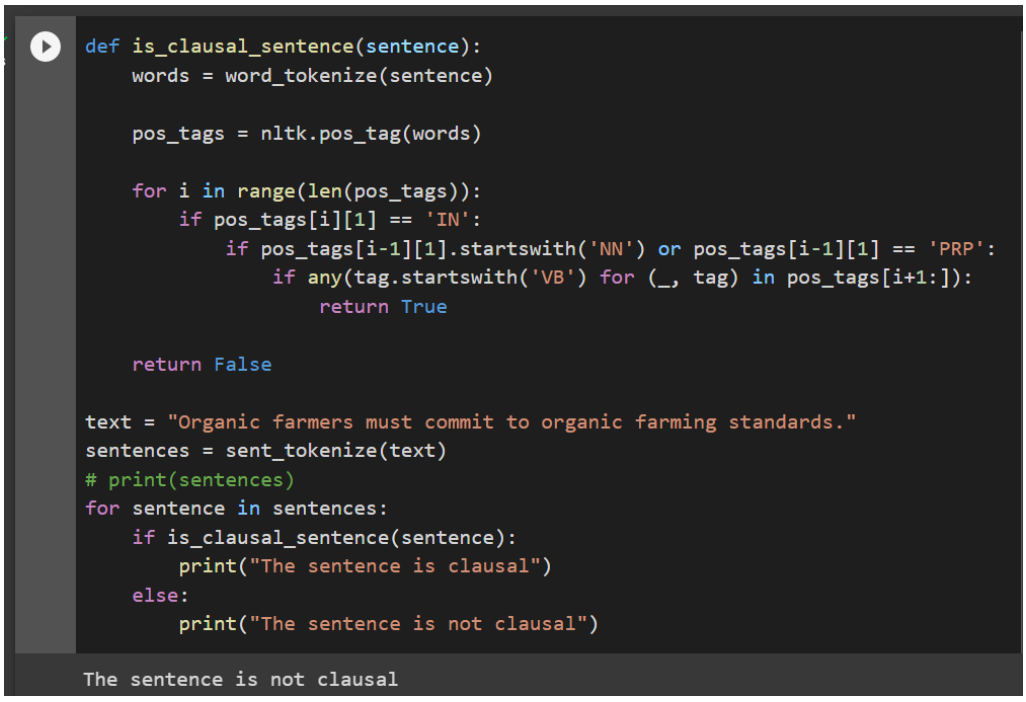


Image 1.4

The input statement for image 1.4 is “Organic farmers must commit to organic farming standards.”.

Note that the given sentence neither consists of an if or when precondition nor does it correspond to a more complex structure of multiple clauses. Thus, we consider it as a non-clausal sentence structure.

4. Introducing Spacy and Dependency graphs

We will be using SpaCy library extensively in the project alongside dependency visualisers for dependency graphs.

SpaCy is designed to be fast, efficient, and production-ready for building advanced NLP applications. It performs Tokenization that splits raw text into individual words or tokens, considering factors like punctuation, contractions, and special characters.

It also has an in-built feature to perform dependency parsing to analyse the grammatical structure of sentences by assigning syntactic dependencies between words, forming a parse tree that represents the relationships between tokens.

Moreover, applications such as information extraction, text classification, named entity recognition, sentiment analysis can also be performed by the library (some of which are not in the scope of this project).

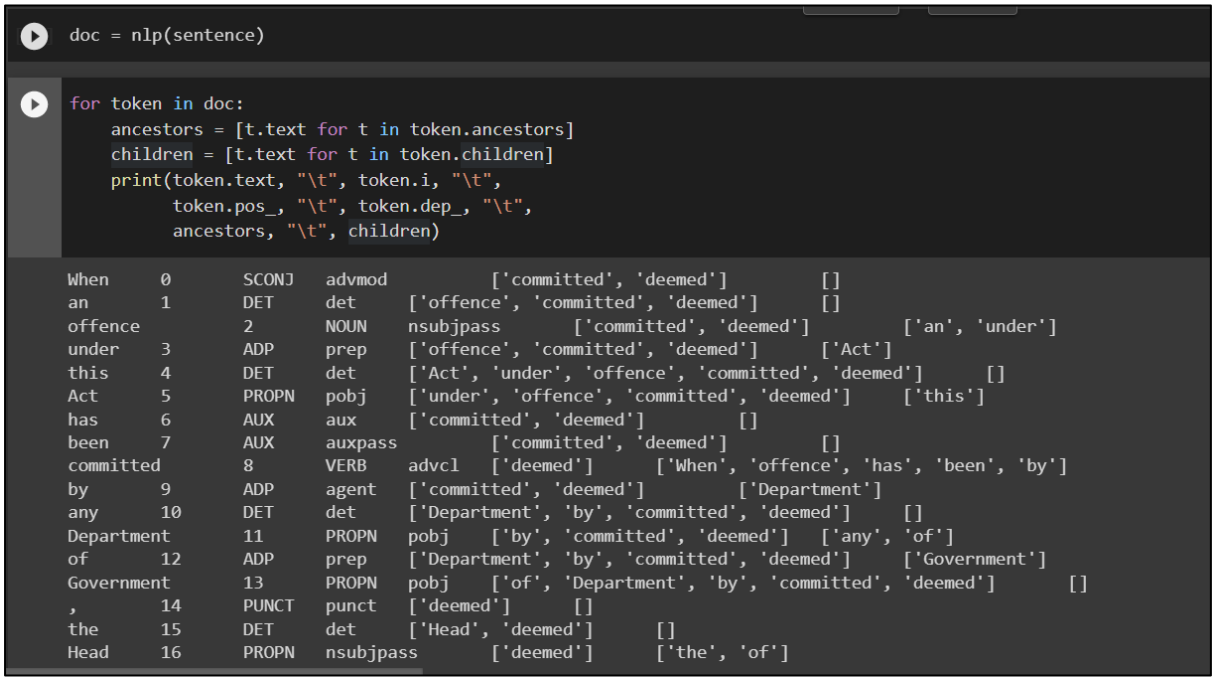
(https://spacy.io/usage/spacy-101)

The dependency graphs created using spacy that are used in our code are implemented using “en_core_web_sm” model. It allows us to analyse the grammatical structure of the sentence.

In a dependency graph, each word in the sentence is represented as a node where every node is a token, and the relationships between the words are represented as directed edges or arcs. The ROOT node has no dependencies flowing into it. The arcs indicate the grammatical dependencies, such as subject, object, modifier, or conjunction, among others. Refer Image 1.6 for more clarity.

5. Handling Conjunctive and Complex Sentences

We have handled the cases of conjunctive sentences and breaking multi clausal sentences at the same time. Thus, the code covers both conjunctive as well as complex sentence structure separation. We have used the spacy library for implementing the same by tokenizing each keyword, and forming a dependency graph between each of the entities.



```
doc = nlp(sentence)
```

```
for token in doc:
    ancestors = [t.text for t in token.ancestors]
    children = [t.text for t in token.children]
    print(token.text, "\t", token.i, "\t",
          token.pos_, "\t", token.dep_, "\t",
          ancestors, "\t", children)
```

```
When     0       SCONJ   advmod          ['committed', 'deemed']         []
an       1       DET     det     ['offence', 'committed', 'deemed']      []
offence          2       NOUN    nsubjpass       ['committed', 'deemed']         ['an', 'under']
under    3       ADP     prep    ['offence', 'committed', 'deemed']      ['Act']
this     4       DET     det     ['Act', 'under', 'offence', 'committed', 'deemed']      []
Act      5       PROPN   pobj    ['under', 'offence', 'committed', 'deemed']     ['this']
has      6       AUX     aux     ['committed', 'deemed']         []
been     7       AUX     auxpass         ['committed', 'deemed']         []
committed        8       VERB    advcl   ['deemed']      ['When', 'offence', 'has', 'been', 'by']
by       9       ADP     agent   ['committed', 'deemed']         ['Department']
any      10      DET     det     ['Department', 'by', 'committed', 'deemed']     []
Department       11      PROPN   pobj    ['by', 'committed', 'deemed']   ['any', 'of']
of       12      ADP     prep    ['Department', 'by', 'committed', 'deemed']     ['Government']
Government       13      PROPN   pobj    ['of', 'Department', 'by', 'committed', 'deemed']       []
,        14      PUNCT   punct   ['deemed']      []
the      15      DET     det     ['Head', 'deemed']      []
Head     16      PROPN   nsubjpass       ['deemed']      ['the', 'of']
```

Image 1.5 describing Dependency Graph of spacy

Image 1.5 represents a dependency graph as a list. This can be better visualised using Image 1.6 which shows a graphical representation of the same. Here the edges represent the dependencies and each word in the sentence is assigned with a token (such as a VERB or a NOUN).

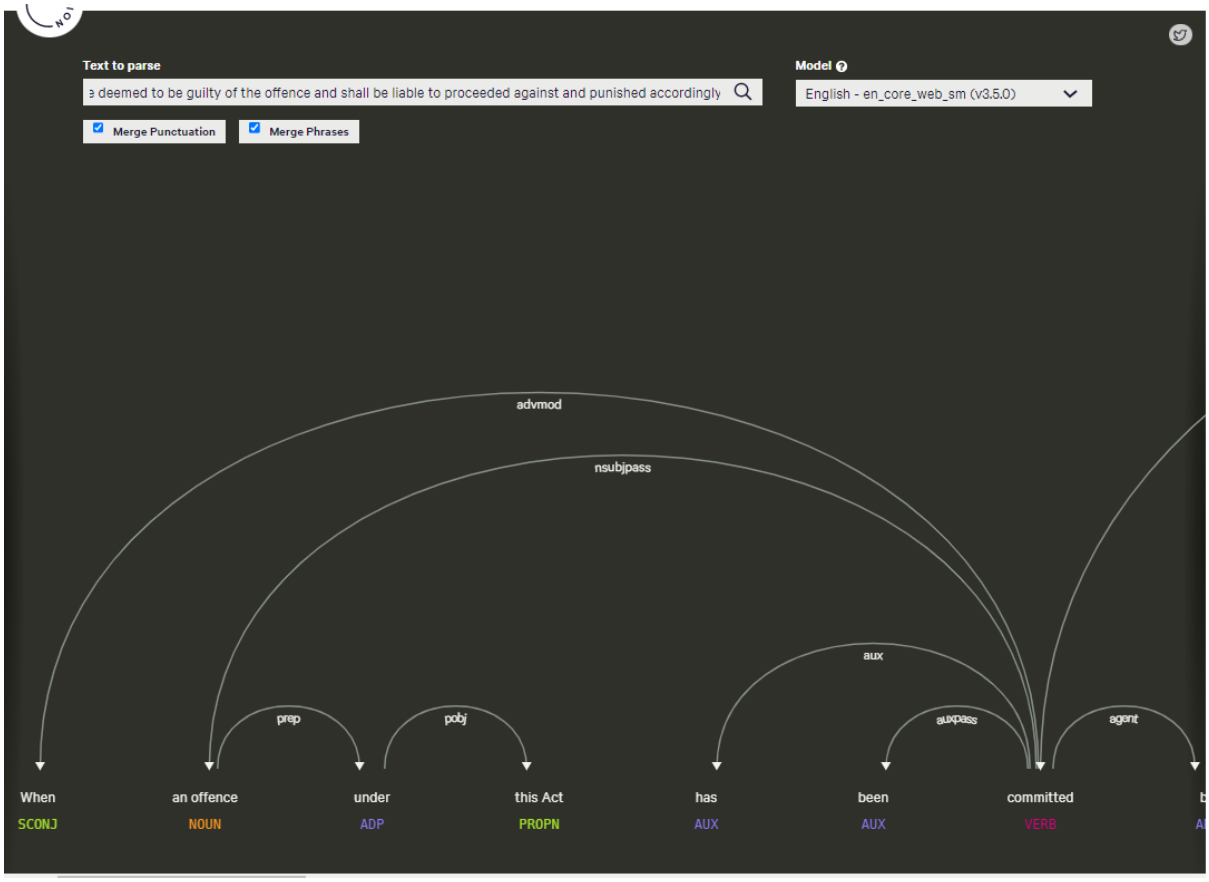


Image 1.6 (displaCy dependency visualiser)

Further on, we split these compound or complex sentences into component clauses. We ensure that each of these component clauses are still linked to each other using one of the many dependencies in the graph.

```
] clauses_text = [clause.text for clause in sentence_clauses]
  print(clauses_text)

  ['When an offence under this Act has been committed', ', the Head of Department shall be deemed to']
```

Image 1.7

Output segment for the given piece of input text. We can clearly see that the clauses are broken at “comma”, thus indicating separate clauses as required.

HEURISTIC 1

```
nlp = spacy.load('en_core_web_sm')
nlp.add_pipe('merge_noun_chunks')
# text = "Every rule made under this Act shall be laid, as soon as maybe a
# doc = nlp(text)
i = 0
noun_found = False
attribute = ""
deontic = ""
aim = ""
for text in clauses_text:
  doc = nlp(text)
  for entity in doc:
      print(entity, entity.pos_, entity.dep_)
      if (entity.pos_ == "PROPN" or entity.pos_ == "NOUN"):
          attribute = str(entity)
          aim = str(entity.head.text)
          aim_found = True
          break

  for entity in doc:
    if (entity.pos_ == "AUX") and (aim == entity.head.text):
      deontic = str(entity)
      # print(deontic + entity.head.text)

# print(doc.vocab)

# for chunk in doc.noun_chunks:
#     if str(chunk.text) == attribute:
#         aim = str(chunk.root.head.text)

print("attribute: " + attribute)
print("aim: " + aim)
print("deontic: " + deontic)
```

Image 1.8

Image 1.8 and 1.9 represent the input code and the corresponding output for a given sentence respectively.

```
When SCONJ advmod
an offence NOUN nsubjpass
, PUNCT punct
the Head PROPN nsubjpass
attribute: the Head
aim: deemed
deontic: be
```

Image 1.9

It is of prime importance to note that the sentence structures are of two types:

TYPE 1: "*The Central Government or any officer empowered by it in this behalf, shall have power to take, for the purpose of analysis, samples of air, water, soil or other substance from any factory, premises or other place in such manner as may be prescribed."*

TYPE 2: "*If any person wilfully delays or obstructs any person empowered by the Central Government under sub-section (1) in the performance of his functions, he shall be guilty of an offence under this act*"

Type 1 sentences follow a format wherein the law starts with an attribute (The Central Government). Also, this type of statement does NOT contain an "if" or a "when". As a result, we tend to get our attribute and aim in the first clause itself.

On the contrary in Type 2 sentences, the sentence is divided into two clauses. This statement starts with an "if" or a "when" thus the attribute is not the first entity in the statement and cannot be retrieved directly.

Sentence in code snippet - "*When an offence under this Act has been committed by any Department of Government, the Head of Department shall be deemed to be guilty of the offence and shall be liable to proceeded against and punished accordingly"*

Since the sentence structure is of TYPE 2, (as a when is encountered first), we know that the Attribute does NOT occur as the first token itself. As a result, we use a **nsubjpass / nsubj edge** to extract the given Attribute.

For the Aim part, we simply find the aim as with TYPE 1. Aim of the sentence is the root verb in most cases. There is an **aux / auxpass** dependency from the aim to the deontic. It is important to note that deontic is an auxiliary verb in the sentence. Since our code only identifies individual (single) tokens for the time being, the deontic is identified as "be". It is worth noting from the spacy visualiser that for a clausal sentence like this one, it has an **aux** or **aux edge** to the other token.

NOTEWORTHY:

When searching for an auxiliary verb (shall/will), there is an advcl directed edge from the verb to either the next verb or another auxiliary verb. An "aux edge" from this node leads us to our auxiliary verb. In this example committed is the verb with an advcl edge to deemed. *Deemed has an aux edge to shall.*

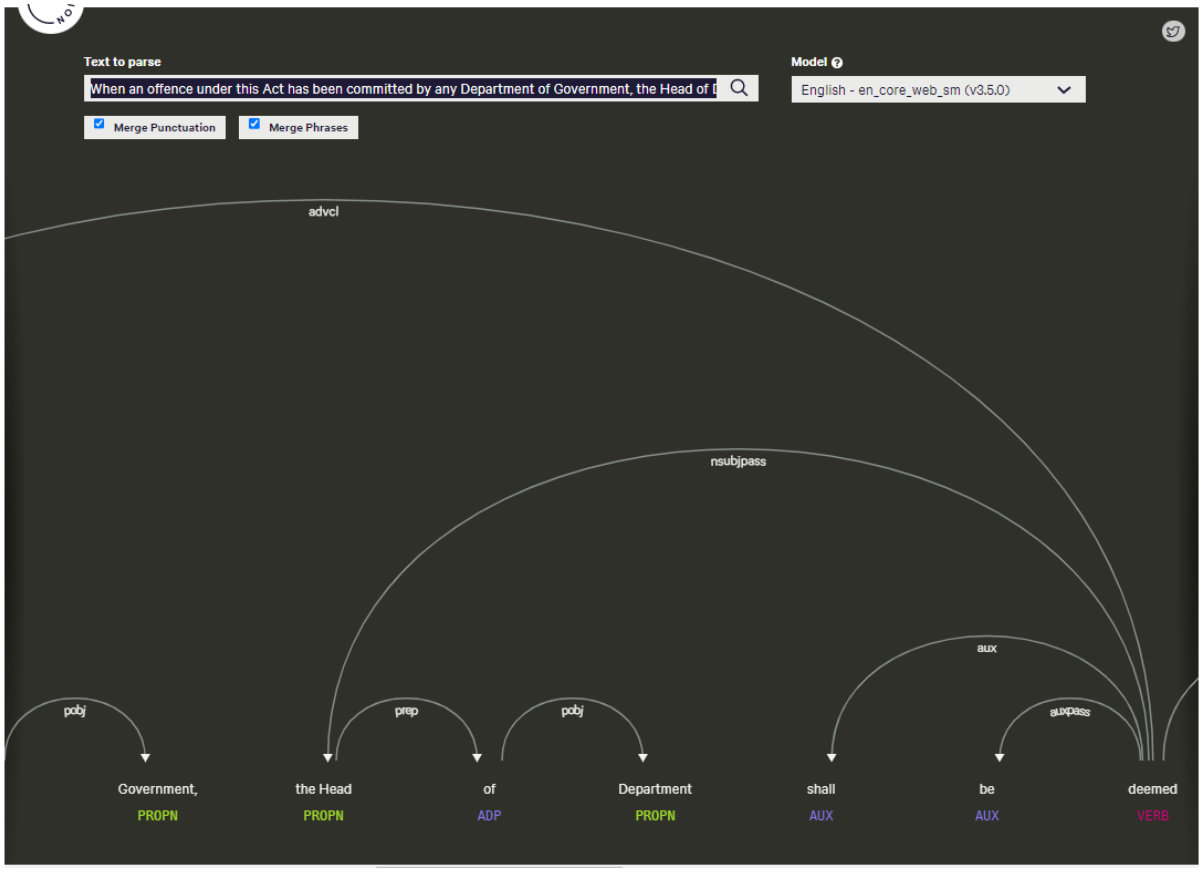


Image 1.10

The advcl edge goes from deemed to the previous verb *committed* in this case.

nsubj / nsubjpass dependency

nsubjpass / nsubj edge in the dependency graph is an edge from the attribute (noun) to its aim (verb). This gives accurate results when both have been identified in the first clause of the sentence itself. In our example, there is a nsubjpass from the Head (attribute) to deemed (verb).

This should work for both clausal and non-clausal sentences.

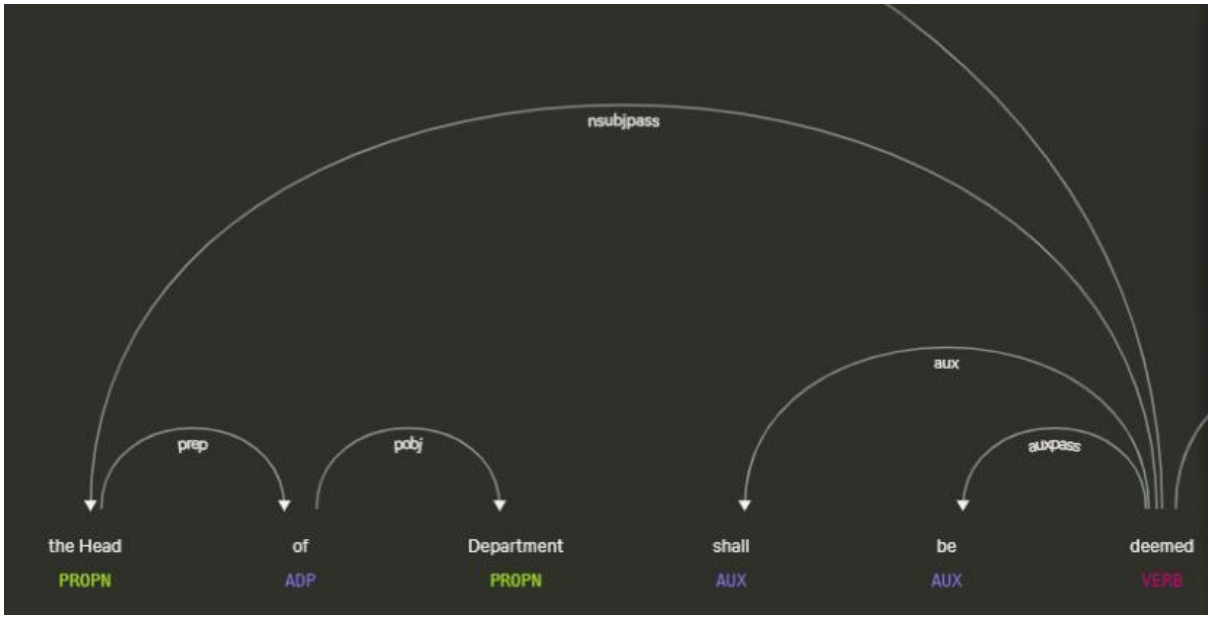


Image 1.11

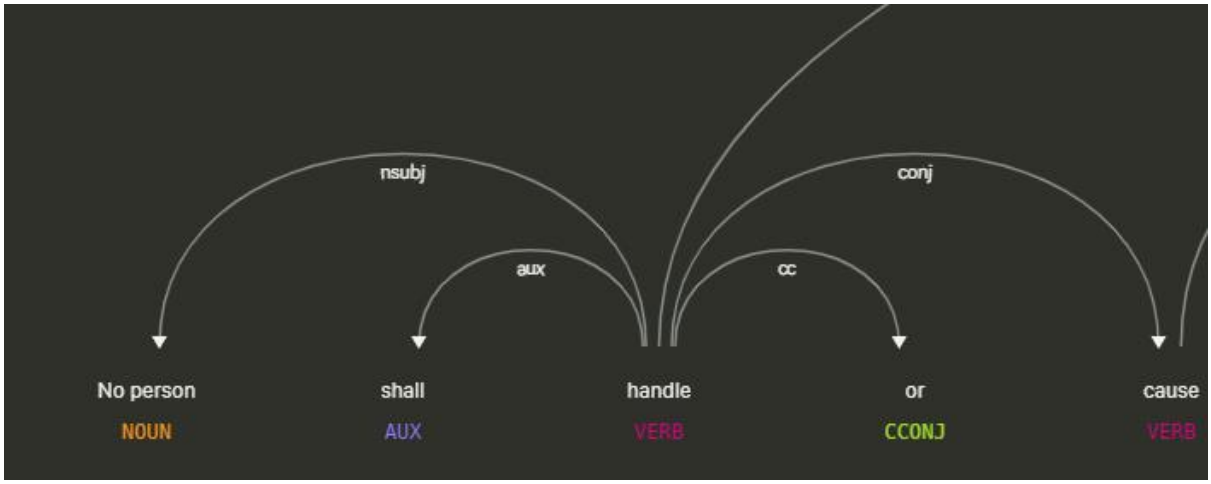


Image 1.12

Image 1.11 shows the nsubj dependency for a clausal sentence "*When an offence under this Act has been committed by any Department of Government, the Head of Department shall be deemed to be guilty of the offence and shall be liable to proceeded against and punished accordingly*".

Image 1.12 shows nsubj dependency for a non-clausal sentence "*No person shall handle or cause to be handled any hazardous substance except in accordance with such procedure and after complying with such safeguards as may be prescribed*".
(https://labour.gov.in/)

HEURISTIC 2

Another important and noticeable observation is the sentence structures with emphasis on the ROOT token of the dependency graph.

Consider the following sentences and their dependency graphs as shown.

"The authority referred to in sub-section (6) may, if it is satisfied that the appellant was prevented by sufficient cause from preferring the appeal within the period specified in sub-section (6), allow the appeal to be preferred within a further period of 30 days but not thereafter"

"Where immediately before the issue of a notification under section 5 fixing or revising the minimum rates of wages in respect of any scheduled employment, wages at a rate higher than the rate so fixed or revised, were payable either by contract or agreement, or, under any other law for the time being in force, then, notwithstanding anything contained in this Act, wages at such higher rate shall be payable to the employees in such scheduled employment and the wages so payable shall be deemed to be the minimum wages for the purposes of this Act."

Both these sentences have different sentence structures based on the "ROOT" of the dependency graph.

The above sentence has ROOT token as a "VERB" whereas the below sentence has an "AUX" as a root. We will consider both of them as different cases and develop our heuristic accordingly as Case 1 and Case 2.

NOTE: If there are cases where we obtain both types of tokens as ROOTS (multiple roots) then we give preference to VERB and classify it as Case 1.

There is also a CASE 3 where a NOUN is the ROOT token. In that case we use our heuristic 1 instead since it yields better results.

Dependency graph for statement 1 with root as VERB:

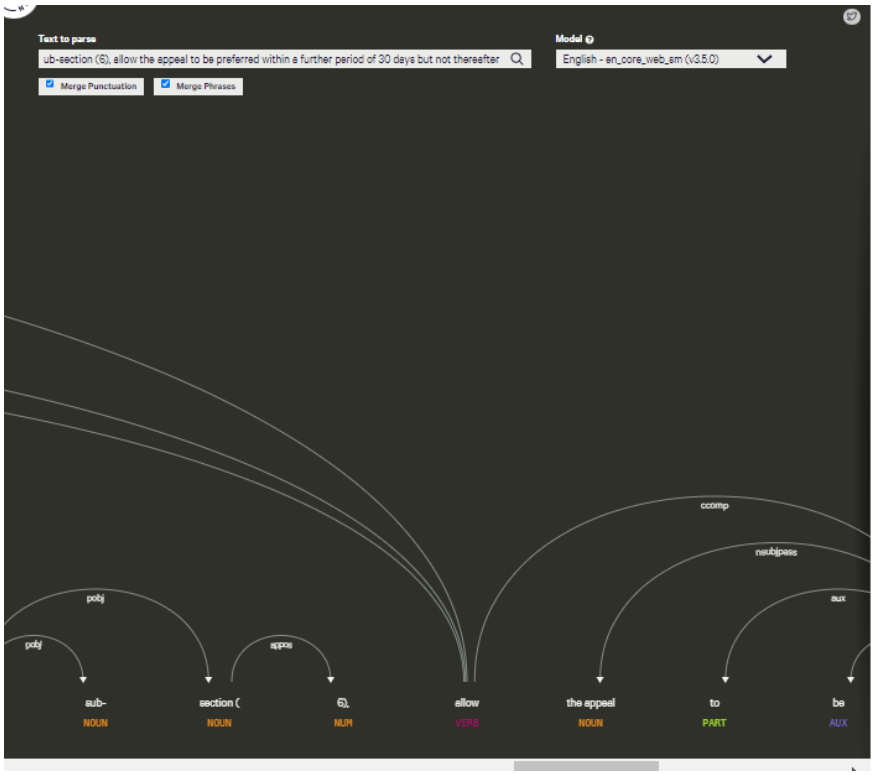


Image 1.13 showing CASE 1 with "VERB" root

Dependency graph for statement 2 with "AUX" as the root:

Image 1.14 showing CASE 2 with "AUX" root

```
keywords = ["shall","may","must"]
secondary_keywords = ["is","has","have","was","having"]


if(case==1):
    #AIM
    #working on root_verb
    flag = 0
    #first adding all VERBS with conj and ccomp to the root_verb list
    for token in root_verb:
        for children in token.children:
            if(children.dep_ == 'ccomp' or children.dep_ == 'conj'):
                root_verb.append(children)

    #in GIVEN ORDER in the list, if we find a token with both nsubj and aux then we return it
    for token in root_verb:
        children_dep = [child.dep_ for child in token.children]
        str1 = "nsubj"
        str2 = "aux"

        if(str1 in children_dep and str2 in children_dep):
            aim_token.append(token)
            flag = 1
            break #only 1 aim in this case
    #nusbjpass and aux if not aux and nsubj
    #if we dont find any such VERB then we take entire root_verb as aim (some of them might be right as there could be multi
    if(flag == 0):
        aim_token = root_verb
    #NOTE: AIM never empty since CASE 1

    #DEONTIC
    #deontic won't be empty due to keywords and secondary keywords
    for token in aim_token:
        for child in token.children:
            if(child.dep_ =='aux' and child.pos_ == 'AUX'):
                deontic_token.append(child)
                break
        if(len(deontic_token)>0):
            break

    #if no such deontic found then search in keywords and later in secondary keywords
    if(len(deontic_token) == 0):
        for keyword in keywords:
            for token in doc:
                if(keyword in token.text and token.pos_ =="AUX"):
                    deontic_token.append(token)
                    break
            if(len(deontic_token)>0):
                break

    if(len(deontic_token) == 0):
        for keyword in secondary_keywords:
            for token in doc:
                if(keyword in token.text and token.pos_ =="AUX"):
                    deontic_token.append(token)
                    break
            if(len(deontic_token)>0):
                break

    #ATTRIBUTE
    #first check for nsubj
    for token in aim_token:
        for entity in token.children:
            if(entity.dep_ =='nsubj' and (entity.pos_ == "PROPN" or entity.pos_ == "NOUN" or entity.pos_ == "PRON")):
                attr_token.append(entity)

    #else check for nsubpass
    if(len(attr_token) == 0):
        for token in aim_token:
            for entity in token.children:
                if(entity.dep_ =='nsubjpass' and (entity.pos_ == "PROPN" or entity.pos_ == "NOUN" or entity.pos_ == "PRON")):
                    attr_token.append(entity)

    #else if empty then check in entire sentence
    if(len(attr_token) == 0):
        for token in doc:
            for entity in token.children:
                if(entity.dep_ =='nsubj' and (entity.pos_ == "PROPN" or entity.pos_ == "NOUN" or entity.pos_ == "PRON")):
                    attr_token.append(entity)
                    break
            if(len(attr_token) > 0):
                break

    if(len(attr_token) == 0):
        for token in doc:
            for entity in token.children:
                if(entity.dep_ =='nsubjpass' and (entity.pos_ == "PROPN" or entity.pos_ == "NOUN" or entity.pos_ == "PRON")):
                    attr_token.append(entity)
                    break
            if(len(attr_token) > 0):
                break


elif(case==2):
    #AIM
    for token in root_aux:
        for children in token.children:
            if((children.dep_ == 'ccomp' or children.dep_ == 'conj') and children.pos_ == 'VERB'):
                aim_token.append(children)

    if(len(aim_token) == 0):
```

Image 1.15 code snippet for case 1 of HEURISTIC 2

This snippet is a part of the function “tokenise”.

The list “*keywords*” and “*secondary_keywords*” maintains the most frequent deontic for Indian Law statements.

In the above code, CASE 1 corresponds to the case where VERB is the root. In Such cases we directly add the **root token as aim** and also other directly connected verbs to aim token as well. These other verbs are connected via “**ccomp**” or “**conj**” dependencies.

For finding the deontic, we look for “**aux**” dependencies which give a direct edge to the **deontic** (an “**AUX” token**) from the aim token found in the previous step. If no such edge is found, we then traverse the “keywords” list declared earlier and find a matching entry in the sentence. Repeat the same process with “secondary_keywords” if no match found in “keywords”.

Finally for the attribute part, we find “nsubj” and “nsubjpass” dependencies from the ROOT token (VERB in this case). If a noun is found to the destination node of the edge, we add it to the attr_token. Since spaCy has nouns classified as pronouns and proper nouns as well, we even look for them. Since attribute dependencies have multi-level hierarchy of preference in our heuristic, we first look for “**nsubj**” dependencies from the aim. If not found we traverse the aim token list once again and look for “**nsubjpass**” dependencies this time since we have given higher preference to “nsubj”.

We have also added extra cases to prevent any of the labels to be empty. So, if we do not find the required dependencies (say “nsubj” **AND “**nsubjpass”) from the aim_token, we iterate through the entire sentence and look for that dependency. Since, multiple attributes are possible due to sentence having multiple clauses, we do not get erroneous results.

This heuristic is relatively more better than Heurisitc 1 since it is more specific. Breaking the cases on ROOT token enusres we do not encounter cases where AIM is empty which was a serious challenge faced in heuristic 1 due to lack of certain dependencies.

```
elif(case==2):
    #AIM
    for token in root_aux:
        for children in token.children:
            if((children.dep_ == 'ccomp' or children.dep_ == 'conj') and children.pos_ == 'VERB'):
                aim_token.append(children)

    if(len(aim_token) == 0):
        for children in token.children:
            if((children.dep_ == 'advcl') and children.pos_ == 'NOUN'):
                aim_token.append(children)
        for token in aim_token:
            for children in token.children:
                if((children.dep_ == 'ccomp' or children.dep_ == 'conj') and children.pos_ == 'VERB'):
                    aim_token.append(children)

    if(len(aim_token) == 0):
        for token in root_aux:
            for children in token.children:
                if(children.pos_ == 'VERB'):
                    aim_token.append(children)



    #DEONTIC
    #for deontic first check aux from root_aux else check from aim else from keywords and secondary keywords
    for token in root_aux:
        for children in token.children:
            if((children.dep_ == 'aux') and children.pos_ == 'AUX'):
                deontic_token.append(children)
                break
        if(len(deontic_token) > 0):
            break

    if(len(deontic_token) == 0):
        for token in aim_token:
            for child in token.children:
                if(child.dep_ =='aux' and child.pos_ == 'AUX'):
                    deontic_token.append(child)
                    break
            if(len(deontic_token)>0):
                break

    #if no such deontic found then search in keywords and later in secondary keywords
    if(len(deontic_token) == 0):
        for keyword in keywords:
            for token in doc:
                if(keyword in token.text and token.pos_ =="AUX"):
                    deontic_token.append(token)
                    break
            if(len(deontic_token)>0):
                break

    if(len(deontic_token) == 0):
        for keyword in secondary_keywords:
            for token in doc:
                if(keyword in token.text and token.pos_ =="AUX"):
                    deontic_token.append(token)
                    break
            if(len(deontic_token)>0):
                break

    #ATTRIBUTE
    for token in root_aux:
        for entity in token.children:
```

Image 1.16 code snippet for CASE 2 of HEURISTIC 2

Here the snippet shows implementation of heuristic with “AUX” as the root.

We first find “AIM” again by finding direct edges from the root token directly to a “VERB”. The edges or dependencies used here are “**ccomp**” and “**conj**”. We then add more VERBS from that token using the same type of edges.

Deontic is calculated differently in case 2. We directly find an “aux” edge to an “AUX” token. If we don not find any such token we THEN look for “aux” edges from aim_token as opposed to case 1 where we first checked edges from the “aim_token”. We then look for the tokens in “keywords” and “secondary_keywords” in given order if deontic is not found.

Attribute is calculated in exactly the same way as in case 1.

TEST DATASET

The test dataset was created by labelling sentences Indian Law statements from multiple sources listed below.

https://clc.gov.in/clc/sites/default/files/PaymentofGratuityAct.pdf

https://egazette.nic.in/WriteReadData/2019/210422.pdf

https://labour.gov.in/sites/default/files/model%20bill%20englsih%20.pdf

https://github.com/InstitutionalGrammar/IG-Inception-Layers

A list of more than 100 sentences was used to create the Gold Standard Dataset. The sentences were carefully analysed and the Aim, Attribute and Deontic were identified and mentioned to compare with the results from the heuristics later on. The Gold standard was annotated manually for each of the 100 examples. This now served as a benchmark or reference for evaluating the performance of the heuristics.

The process was more time-consuming although this could now be used as a standard for the silver data generated using the models. Thus, it provided the base for reliability of our model since it is more consistent and performed under the rules specified by the Code Book.

(Institutional Grammar 2.0 Code Book Christopher K. Frantz Saba N. Siddiki)

Certain ambiguous cases were discussed and clarified to improve the quality of the annotations. Disagreements or discrepancies were resolved through discussions, clarification of guidelines by thorough reference to the Code Book.

Image 1.2 shows some sentences along with the annotations from the Gold Standard dataset.

RESULTS

The accuracies for the heuristics are calculated such that the heuristic with more accurate results is used in the accuracy metric as discussed. So, heuristic 2 is given preference whenever we come across sentences having VERB and AUX tokens as attributes. If we fail to find any such root, we use heuristic 1 as our model to be compared against the gold standard.

The following image shows how the Gold standard and the results from the heurisitcs were converted to a list of respective tokens since we are handling the case of multiple labels for a single sentence as well.

Image 1.17

The function “convert_to_lowecase” ensures that homogenenous format is adhered to across all the tokens and that doesn’t lead to a loss in accuracies.

“calculate_accuracy” is used to find whether we find a common label in the Gold Standard and the token returned from the heuristic. IF a match is found we increment the count value which keeps the track of the number of correctly labelled tokens when compared with the Gold standard.

This process is repeated for all “Aim”, “Attribute” and “Deontic” separately and the results are tabulated below.

```
def calculate_accuracy(df,c1,c2):
    count = 0
    for index, row in df.iterrows():

        list_a = row[c1]
        list_b = row[c2]
        flag = False
    # Check if any element from List A is a substring of List B
        for a in list_a:
            for b in list_b:
                if a in b and (flag ==False):
                    flag = True
                    count += 1  # Return 1 if a match is found

    # Check if any element from List B is a substring of List A
        for b in list_b:
            for a in list_a:
                if b in a and (flag ==False):
                    count += 1  # Return 1 if a match is found
                    flag == True
        if(flag==False):
            print(index, row[0])
            print("\n")
    return count/len(df)*100
```

```
In [66]:
def convert_to_lowercase(value):
    if isinstance(value, str):
        return value.lower()
    elif isinstance(value, list):
        return [v.lower() for v in value]
    else:
        return value

# Apply the conversion function to the entire DataFrame using applymap()
scores_sheet = scores_sheet.applymap(convert_to_lowercase)
```

```
In [67]:
accuracy_deontic = calculate_accuracy(scores_sheet,3,6)
print("\n\n\n")
print(accuracy_deontic)
```

```
46 provided that nothing contained in this sub-section shall render any such person liable to any punishment if he proves that
```

Image 1.18

```
print(accuracy_attr)
print(accuracy_deontic)
print(accuracy_aim)
```

```
73.77049180327869
94.26229508196722
93.44262295081968
```

Image 1.19

ACCURACY SCORES		
AIM	**ATTRIBUTE**	**DEONTIC**
93.4426	**94.2623**	**73.7705**

The values of accuracies are up to 4 decimal places.

Clearly Aim and Attribute return high accuracies, reason being most of the ROOT tokens are VERBs which in turn are the AIMs for most of the regulatory sentences. Thus AIM gets annotated correctly since it the ROOT token itself and we do not need to traverse multiple edges to reach to it.

As for the deontic there is a direct edge from the ROOT token to the deontic in most cases which makes it simpler to extract. Moreover, the keyword and secondary_keyword list ensure we find the required AUX in most cases, if we fail to find it through dependencies.

The accuracy of attribute is relatively less because we have effectively identified the NOUN corresponding to the attribute in most cases but the usefulness of that NOUN is still sometimes questionable which is a challenge since spaCy doesn't distinguish much between different types of NOUNS through dependencies. Following are certain methods that can be used to improve the performance for the discrepancies and improve accuracies further.

FUTURE WORK

1. Fine tuning Spacy

We talked about how the silver dataset is created using the two Heuristics discussed earlier.

Heuristic 2 clearly showed that ROOT Verb might not always be the main AIM of the sentence as in domain specific tasks like the one being performed.

“Provided that in relation to the State of Jammu and Kashmir, the reference to the accounting year commencing on any day in the year 1964 and every subsequent accounting year shall be construed as reference to the accounting year commencing on any day in the 1968 and every subsequent accounting year.” (Refer metrics sheet example 12)

No edges into” provided”, therefore it is the ROOT Verb (case 1 in heuristic 2). Thus, Heuristics can be used to improve performance of spaCy on more domain specific purposes. We can use different pretrained models and run heuristics on them as well to improve performance.

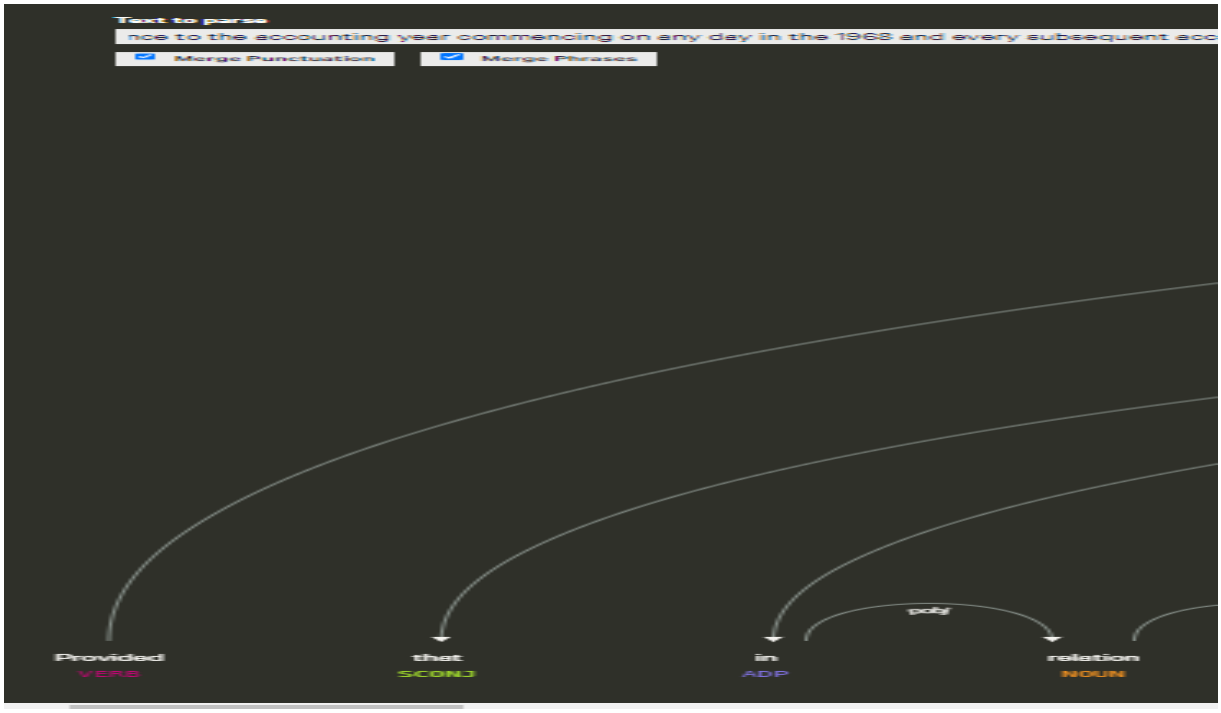


Image 1.20

2. Utilising Prompt-tuning techniques

We can extend the project to return better results by making use of Rule based prompts by defining a set of rules or patterns that trigger specific responses. We can further explore and use SpaCy in conjunction with larger language models like GPT-3 by OpenAI. There is also scope of increasing accuracies by using different available pipelines and adjust the hyperparameters of the language model during training to optimize its performance.

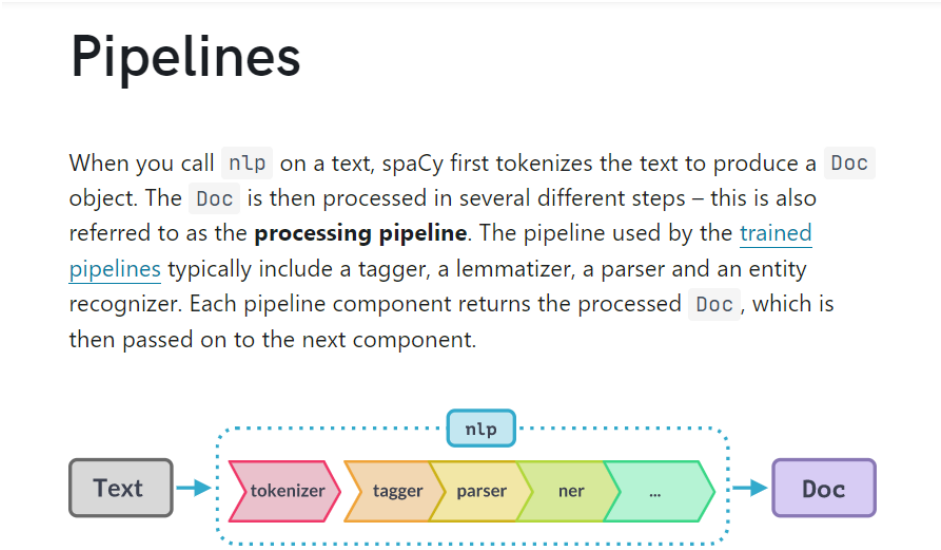


Image 1.21

(https://spacy.io/usage/spacy-101#pipelines)

CONCLUSION

Based on our assumptions and heuristics mentioned over the complete report, we have found out how different set of rules need to be followed when working with different types of sentences. We also categorised the tokens of importance into Aim, Attribute, Deontic and further wish to continue the same task with the Context of the Grammar as well.

The modularity of our code allowed us to individually look at the different steps carried out in the given order as above, and later on tokenize and identify the required entities from the entire sentence, be it Clausal or Non-clausal.

The dependencies like advcl talked so far are only the ones giving relationships between Aim, Attribute and Deontic. We further on have to add the Context to the given relationship.

To optimise our results and add reliability we have also performed Prompt Engineering using LSTM to get the required results, although it is not part of the report yet. This would be a key step in fine tuning spacy on results that were giving incorrect outputs previously.

REFERENCES

Online Spacy - https://demos.explosion.ai/displacy

Prompt Engineering - https://github.com/dair-ai/Prompt-Engineering-Guide/blob/main/guides/prompts-intro.md

Named Entity Extraction - https://github.com/prajnaupadhyay/openie-with-entities

Institutional Grammar Codebook - https://github.com/InstitutionalGrammar/IG-Inception-Layers

Breaking sentences to clauses - https://subscription.packtpub.com/book/data/9781838987312/2/ch02lvl1sec13/splitting-sentences-into-clauses

Source - https://clc.gov.in/clc/sites/default/files/PaymentofGratuityAct.pdf

Source - https://egazette.nic.in/WriteReadData/2019/210422.pdf

Source - https://labour.gov.in/sites/default/files/model%20bill%20englsih%20.pdf

Language models - https://arxiv.org/pdf/2204.06031.pdf

Bidirectional LSTM-CRF Models for Sequence Tagging - https://arxiv.org/abs/1508.01991

End-to-End Sequence Labeling via Bi-directional LSTM-CNNs-CRF- https://arxiv.org/abs/1603.01354

Neural Architectures for Named Entity Recognition - https://arxiv.org/abs/1603.01360