INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY

HYDERABAD

Project Report

(CS7.401 - Introduction to NLP)

Topic: Dependency Parsing

Submitted by

Team: NSU_NLP

Member 1: Nikhil Chawla (2022201045)

Member 2: Shubham Deshmukh (2022201076)

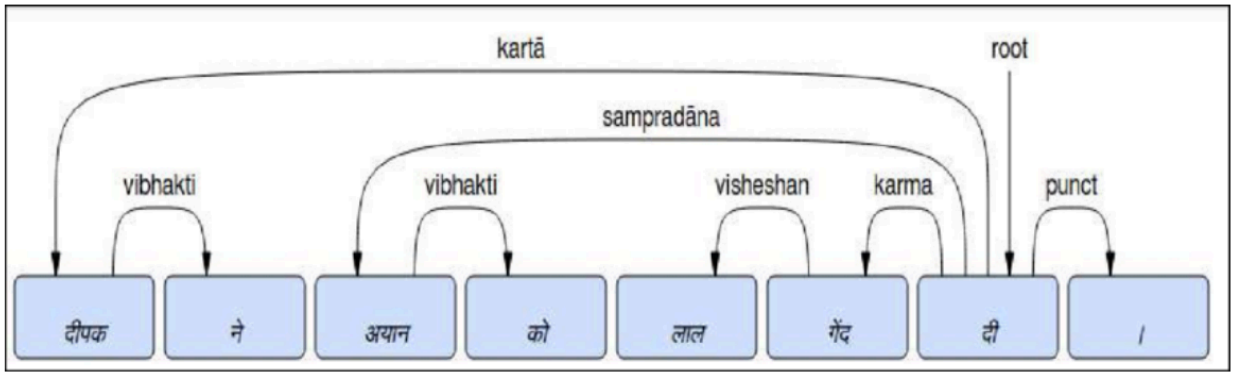
Member 3: Udrasht Pal (2022201020)

Problem Objective

In this project, we are developing a dependency parser for the **Hindi language** using a Hindi Dependency Treebank (HDTB). A dependency parser is used to recognize the syntactic relationships between words in a sentence. Despite abundant literature on dependency parsing in the NLP community, there are limited sophisticated tools available specifically for Indian languages.

Dependency Parser

A dependency parser analyzes the grammatical structure of a sentence, establishing relationships between "head" words and words that modify those heads. The figure below shows a dependency parse of a short sentence. The arrow from the word “दी” to the word “दीपक” indicates that “दीपक” modifies “दी”, and the label “Karta” assigned to the arrow describes the exact nature of the dependency.



A dependency parsing mechanism can give a parsed dependency tree for a given sentence, which involves a set of relations over its words such that one is a ‘dependent’ of the other.

Our Methodology

We are using a **Transition-based parsing approach**
Transition-based Dependency Parsing: This parser creates sentence structures by reading words one by one. It uses a stack to track words being processed and a buffer for words to process. It applies three actions—LEFT-ARC, RIGHT-ARC, and SHIFT—to build the structure. A neural network decides which action to take based on the current state. The training uses an oracle to determine the correct actions for each step in the training data.

Advantages of Transition-based Parsing:
- Faster processing.
- Easier implementation.
- Works well for incremental parsing.

Dataset

We are using **Hindi Dependency Treebank (HDTB)**:
Link to the dataset: https://ltrc.iiit.ac.in/treebank_H2014/
The Hindi Dependency Treebank (HDTB) is a linguistic resource created to facilitate the study of the syntactic structure of Hindi sentences. It consists of manually annotated sentences with syntactic dependencies, providing a detailed analysis of the grammatical relationships between words in Hindi. The HDTB was developed by the Language Technology Research Center (LTRC) at the International Institute of Information Technology, Hyderabad (IIIT-H), India. Creating the HDTB aims to support research and development in natural language processing (NLP) tasks specific to Hindi, such as parsing, machine translation, information retrieval, and sentiment analysis.

Preprocessing the Data

Steps:

1. Manual Sourcing of Data: Data was manually sourced and selected from from the "news_articles_and_heritage" section of LTRC (Language Technologies Research Centre), a division of LDC.The linguistic data obtained from LTRC was in the Shakti Standard Format (SSF) encoded in UTF (Unicode Transformation Format), which is a structured format designed for the representation of linguistic annotations and analyses. SSF facilitates the encoding of various linguistic attributes such as part-of-speech tags, syntactic structures, and named entities, ensuring consistency and machine readability.

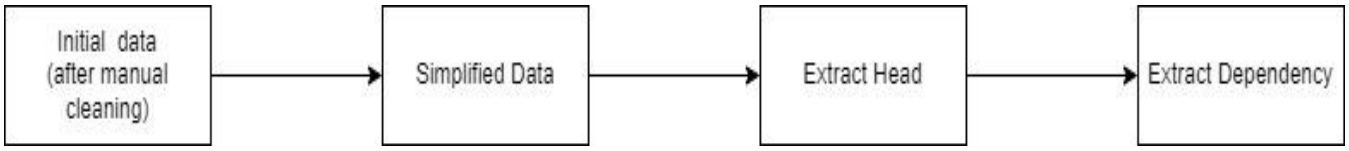
2. Data Extraction: The next involves extraction of information about constituents, terminal nodes, sentences, parts of speech and other relevant information.

3. Extraction of head: The corpus is divided into sentences and each sentence is divided into chunks. This step involves determining chunk heads based on their part-of-speech tags based on the predefined mapping.

4. Extraction of Dependency: Extraction of Dependency from the processed data for the data based on its chunk and removal of unknown and non-parsable dependency from our processed data. Then the processed data is parsed based on a transition based parsing approach.

After that we extract Dependency Parse for that we use below steps

- Simplifying data
- Head Extraction from Chunks
- Generating Parse from Head Extraction data



Initial Data looks like:

initialy Total sentence in training data :13638
initialy Total sentence in testing data :13638

```
<Sentence id='1'>
1 (( NP <fs name='NP' drel='r6:NP2'>
1.1 यह DEM <fs af='यह,pn,any,sg,3,d,,' posn='10' name='यह'>
1.2 एशिया NNP <fs af='एशिया,n,m,sg,3,o,0,0' posn='20' name='एशिया'>
1.3 की PSP <fs af='का,psp,f,pl,,o,,' posn='30' name='की'>
))
2 (( NP <fs name='NP2' drel='k7:VGF'>
2.1 सबसे INTF <fs af='सबसे,avy,,,,,,' posn='40' name='सबसे'>
2.2 बड़ी JJ <fs af='बड़ा,adj,f,pl,,o,,' posn='50' name='बड़ी'>
2.3 मस्जिदों NN <fs af='मस्जिद,n,f,pl,3,o,0,0' posn='60' name='मस्जिदों'>
2.4 में PSP <fs af='में,psp,,,,,,' posn='70' name='में'>
2.5 से PSP <fs af='से,psp,,,,,,' posn='80' name='से'>
))
3 (( NP <fs name='NP3' drel='k1:VGF'>
3.1 एक QC <fs af='एक,num,any,any,,any,,' posn='90' name='एक'>
))
```

```
4 (( VGF <fs name='VGF' voicetype='active' stype='declarative'>
4.1 है VM <fs af='है,v,any,sg,3,,है,hE' posn='100' name='है'>
))
5 (( BLK <fs name='BLK' drel='rsym:VGF'>
5.1 । SYM <fs af='।,punc,,,,,,' posn='110' name='।'>
))
</Sentence>
```

Sentence Structure:

Each sentence is enclosed within <Sentence id='x'> and </Sentence> tags, where x is the identifier for the sentence. This structure signifies the beginning and end of a sentence within the dataset.

Chunking:
The sentence is segmented into chunks or pieces, each identified with a unique tag (NP, VGF, BLK, etc.).
Each chunk is defined with additional attributes and annotations (<fs> tags) that provide detailed linguistic information about its components.

Chunk Relationships (“drel” attribute):
Each chunk (NP, VGF, etc.) has a **“**drel**”**attribute that specifies its relationship with other chunks in the sentence.
The **“**drel**”**attribute consists of two components separated by a colon (:):
The first component indicates the type of dependency relationship (r6, k7, k1, rsym, etc.).
The second component is the name of the related chunk, indicating the subordinate or dependent relationship.

Morphological Information (<fs> tags):
Each word within a chunk is associated with <fs> tags that provide morphological information about the word, such as its lemma (af attribute), part-of-speech (posn attribute), gender, number, and other grammatical features.

For example:
यह DEM <fs af='यह,pn,any,sg,3,d,,' posn='10' name='यह'>
Here, यह is a demonstrative pronoun (DEM) with specific morphological attributes (af='यह,pn,any,sg,3,d,' posn='10').

Simplified data:

We write the **python script** for extracting the simplified data from the initial data.

```
<Sentence id='1'>
H NP NP r6 NP2
T यह DEM यह
T एशिया NNP एशिया
T की PSP का
H NP NP2 k7 VGF
T सबसे INTF सबसे
T बड़ी JJ बड़ा
T मस्जिदों NN मस्जिद
T में PSP में
T से PSP से
H NP NP3 k1 VGF
T एक QC एक
H VGF VGF NULL ROOT
T है VM है
H BLK BLK rsym VGF
T । SYM ।
</Sentence>
```

Each line represents either a chunk (starting with H) or a token (starting with T), providing information about the part-of-speech tag, word form, and the chunk it belongs to, as well as dependency relationships where applicable.

Extract Head:

We write python script for extracting the Head of chunks form Simplified data and we use below rules for identify chunk tag

Identify Chunk Tags and Rules:
Determine the chunk tag for each chunk in the sentence based on the provided rules:

- JJP: Look for 'JJ' (adjective), otherwise 'QF', 'QC', or 'QO'.
- VGNF, VGF, VGNN: Look for 'VM' (main verb).
- NEGP: Look for 'NEG' (negation).
- RBP: Look for 'RB' (adverb), otherwise 'NN' (noun) or 'WQ' (wh-question).
- BLK: Look for 'SYM' (symbol), otherwise 'UNK', 'RP', or 'INJ'.
- CCP: Look for 'CC' (coordinating conjunction), otherwise 'SYM'.

- NP: Look for 'NN', 'PRP', 'NNP' (nouns or pronouns), otherwise 'QC', 'QF', 'QO' (quantifiers), 'NST' (demonstrative pronoun), or 'WQ' (wh-word).

Reverse Emphasis and Head Selection:
For each chunk, reverse the order of words and identify the first word that matches the preferred part-of-speech tag for that chunk:

- Start with the words within the chunk in reverse order.
- Check each word against the preferred part-of-speech tags for the chunk.
- The first matching word is selected as the head of the chunk.
- If no head word is found based on the primary preference, check the next preferences in the given order until a head word is identified.

Remove Unmatched Sentences:
If a sentence does not adhere to any of the specified rules (i.e., cannot identify a head word for any chunk based on the preferences), remove that sentence from the dataset.

Data after Extract Head

```
H एशिया एशिया NP NNP NP r6 NP2
H मस्जिदों मस्जिद NP NN NP2 k7 VGF
H एक एक NP QC NP3 k1 VGF
H है है VGF VM VGF NULL ROOT
H । । BLK SYM BLK rsym VGF
```

Dependency Extracted:

The new format of sentence annotation, using the **Arc-Eager approach ,** introduces additional details and dependencies for each chunk in the sentence. Let's break down the explanation:

The implementation was mostly based on the standard algorithm in Jurafsky. The change was only in the REDUCE operation. Initially check if a Right Arc or a Left Arc is possible. If No then check for REDUCE operation. For REDUCE operation, first check if the top of the buffer has L/R relation with any of the elements in the stack except the top. If yes, then check if the top of stack is already a child of some other element. If the answer is yes in both the cases, then apply REDUCE, else apply SHIFT.

The sentences which were found out to be non parsable were those sentences which had projectivity issues. If all of the arcs can be drawn without any intersection of any of the arcs (in one plane), then the sentence was parsable. Else, if at least two of the arcs

intersected, then the sentence was non-parsable and Arc Eager removed the sentence from the corpus.

Data look like after extract dependency

H एशिया एशिया NP NNP NP r6 NP2 ; H मस्जिदों मस्जिद NP NN NP2 k7 VGF ; R ; r6
H मस्जिदों मस्जिद NP NN NP2 k7 VGF ; H है है VGF VM VGF NULL ROOT ; R ; k7
H एक एक NP QC NP3 k1 VGF ; H है है VGF VM VGF NULL ROOT ; R ; k1
ROOT ; H है है VGF VM VGF NULL ROOT ; L ; ROOT
H है है VGF VM VGF NULL ROOT ; H । । BLK SYM BLK rsym VGF ; L ; rsym
H मस्जिदों मस्जिद NP NN NP2 k7 VGF ; H एक एक NP QC NP3 k1 VGF ; U ; NULL
H एशिया एशिया NP NNP NP r6 NP2 ; H है है VGF VM VGF NULL ROOT ; U ; NULL
H एक एक NP QC NP3 k1 VGF ; H । । BLK SYM BLK rsym VGF ; U ; NULL
H मस्जिदों मस्जिद NP NN NP2 k7 VGF ; H । । BLK SYM BLK rsym VGF ; U ; NULL
H एशिया एशिया NP NNP NP r6 NP2 ; H । । BLK SYM BLK rsym VGF ; U ; NULL
ROOT ; H एक एक NP QC NP3 k1 VGF ; U ; NULL
ROOT ; H मस्जिदों मस्जिद NP NN NP2 k7 VGF ; U ; NULL
ROOT ; H एशिया एशिया NP NNP NP r6 NP2 ; U ; NULL

Models:

There are two types of techniques used in the Dependency parser and they are:

1. Support Vector Machines (SVM)
2. Neural Network (FFNN)(New Work)

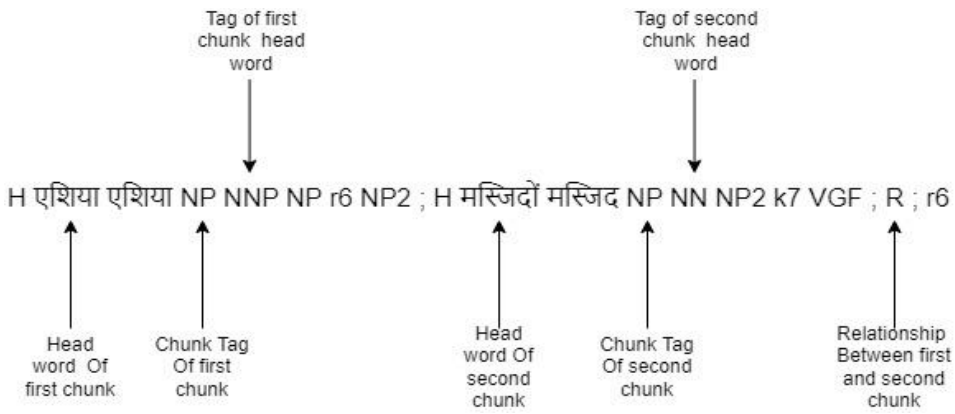
1. Support Vector Machines (SVM)

A support vector machine (SVM) is a supervised machine learning model that uses classification algorithms for two-group classification problems. A support-vector machine constructs a hyperplane or set of hyperplanes in a high- or infinite-dimensional space, which can be used for classification, regression, or other tasks like outliers detection. Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training-data point of any class (so-called functional margin), since in general the larger the margin, the lower the generalization error of the classifier. Generally SVM's are considered to be used for binary classification but here we have used a multi-class linear SVM classifier.

Input: Features and their combination
Output: Edge labels and Edge Relationship that can be either Left, Right or Unknown

Feature are :

- Head Word: The word that represent the chunk
- POS tags : Tag of head word of chunk
- Chunk Tags: Tag of Chunk



Classifiers and their Results that are used based on Feature selected:

1. Chunk LRU
   It is a classifier that predicts the edge relationship based on the chunk tags.

	precision	recall	f1-score	support
L	0.70	0.82	0.76	30833
R	0.67	0.72	0.69	78653
U	0.84	0.78	0.80	161372
accuracy			0.77	270858
macro avg	0.73	0.77	0.75	270858
weighted avg	0.77	0.77	0.77	270858

	precision	recall	f1-score	support
L	0.71	0.83	0.76	4174
R	0.67	0.72	0.69	10448
U	0.84	0.78	0.81	21519
accuracy			0.77	36141
macro avg	0.74	0.78	0.75	36141
weighted avg	0.77	0.77	0.77	36141

2. POS LRU
   This classifier uses the POS tags(tag of head word) to predict the edge relationship

	precision	recall	f1-score	support
L	0.59	0.31	0.41	32409
R	0.67	0.69	0.68	82664
U	0.76	0.81	0.78	169393
accuracy			0.72	284466
macro avg	0.67	0.60	0.62	284466
weighted avg	0.71	0.72	0.71	284466

	precision	recall	f1-score	support
L	0.60	0.31	0.41	4174
R	0.67	0.69	0.68	10448
U	0.75	0.81	0.78	21561
accuracy			0.72	36183
macro avg	0.68	0.61	0.63	36183
weighted avg	0.71	0.72	0.71	36183

3. POS chunk LRU
   The Classifier used POS tags(tag of head word) and the tag of chunk to predict the Edge relationship

	precision	recall	f1-score	support
L	0.70	0.82	0.76	32409
R	0.67	0.70	0.68	82664
U	0.83	0.78	0.80	169393
accuracy			0.76	284466
macro avg	0.73	0.77	0.75	284466
weighted avg	0.77	0.76	0.76	284466

	precision	recall	f1-score	support
L	0.71	0.83	0.76	4174
R	0.67	0.70	0.68	10448
U	0.83	0.78	0.80	21561
accuracy			0.76	36183
macro avg	0.73	0.77	0.75	36183
weighted avg	0.77	0.76	0.76	36183

4. Word LRU
   This Classifier uses the head words to predict the edge relationship.

	precision	recall	f1-score	support
L	0.77	0.77	0.77	32409
R	0.70	0.83	0.76	82664
U	0.87	0.80	0.84	169393
accuracy			0.80	284466
macro avg	0.78	0.80	0.79	284466
weighted avg	0.81	0.80	0.81	284466

	precision	recall	f1-score	support
L	0.75	0.71	0.73	4174
R	0.66	0.77	0.71	10448
U	0.83	0.78	0.80	21561
accuracy			0.77	36183
macro avg	0.75	0.75	0.75	36183
weighted avg	0.77	0.77	0.77	36183

5. Word chunk LRU
   This classifier uses the tags of the Chunk and head word to predict edge relationship

	precision	recall	f1-score	support
L	0.77	0.83	0.80	32409
R	0.71	0.82	0.76	82664
U	0.88	0.80	0.84	169393
accuracy			0.81	284466
macro avg	0.79	0.82	0.80	284466
weighted avg	0.82	0.81	0.81	284466

	precision	recall	f1-score	support
L	0.74	0.79	0.77	4174
R	0.67	0.79	0.72	10448
U	0.86	0.77	0.81	21561
accuracy			0.78	36183
macro avg	0.75	0.78	0.77	36183
weighted avg	0.79	0.78	0.78	36183

6. Word Pos LRU
   This classifier uses the head word and POS tag(tag of head word of chunk) to predict the edge relationship.

	precision	recall	f1-score	support
L	0.77	0.77	0.77	32409
R	0.71	0.83	0.76	82664
U	0.87	0.80	0.84	169393
accuracy			0.80	284466
macro avg	0.78	0.80	0.79	284466
weighted avg	0.81	0.80	0.81	284466

	precision	recall	f1-score	support
L	0.75	0.72	0.73	4174
R	0.66	0.79	0.72	10448
U	0.85	0.77	0.81	21561
accuracy			0.77	36183
macro avg	0.75	0.76	0.76	36183
weighted avg	0.78	0.77	0.78	36183

7. Word POS chunk LRU
   This classifier uses the head word of chunk, pos tag(tag of head word of chunk) and chunk tag to predict the edge relationship.

	precision	recall	f1-score	support
L	0.77	0.83	0.80	32409
R	0.71	0.82	0.76	82664
U	0.88	0.80	0.84	169393
accuracy			0.81	284466
macro avg	0.79	0.82	0.80	284466
weighted avg	0.82	0.81	0.81	284466

	precision	recall	f1-score	support
L	0.74	0.79	0.77	4174
R	0.67	0.79	0.72	10448
U	0.86	0.77	0.81	21561
accuracy			0.78	36183
macro avg	0.75	0.78	0.77	36183
weighted avg	0.79	0.78	0.78	36183

2. Neural Network (FFNN) (New Work)

We're developing a Feedforward Neural Network (FFNN) for dependency parsing on Hindi language data. Our approach involves leveraging pre-trained word embeddings specifically tailored for Hindi, utilizing the "IndicNLP" embedding model.

Why do we choose FFNN?

- Fixed Input Length: FFNNs work well when the length of our sentences doesn't change. In our approach the input length is fixed.
- Efficient for Fixed-Length Inputs: They handle sentences with the same length efficiently.
- Simple and Easy to Understand: FFNNs have a straightforward structure, making them easy to work with and understand.
- Works with Large Datasets: They can handle big sets of data, which is useful for projects like ours.
- Saves Memory: FFNNs need less memory during use, which is good for running our model on different devices.

1. Chunk LRU FFNN:

It is a model that predicts the edge relationship based on the chunk tags.

```
chunk_tags": ["NP", "VGF", "BLK", "CCP", "JJP", "VGNN", "VGNF", "RBP",
"NEGP", "ROOT"]
```

We have 10 different chunk tags, and we're using "one hot encoding" to represent them as input for our FFNN. Our goal is to train the model to predict the relationships between these chunks based on their tags.

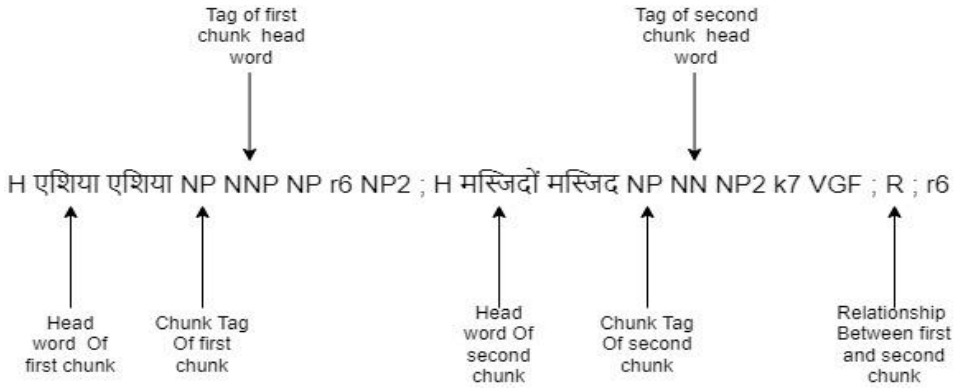


Figure 2

Input=[“NP”,”NP”]

After one hot encoding=[[1,0,0,0,0,0,0,0,0,0],[1,0,0,0,0,0,0,0,0,0]]

```
Number of feature=2
input_size = 2X10=20 # Number of features after one-hot encoding
hidden_size = 64  # Number of units in the hidden layer
num_classes = 3  # Number of output classes
num_epochs=10
```

Results

	precision	recall	f1-score	support
R	0.68	0.73	0.70	10448
L	0.68	0.90	0.77	4174
U	0.85	0.77	0.81	21519
accuracy			0.77	36141
macro avg	0.74	0.80	0.76	36141
weighted avg	0.78	0.77	0.77	36141

2. POS LRU FFNN:

This Model Uses the POS tags of the head words to predict the edge relationship.

```
"pos_tags": ["NNP", "NN", "VM", "QC", "SYM", "PRP", "CC", "NST", "JJ",
"WQ", "QF", "RB", "QO", "NEG", "ROOT"]
```

We have 15 different Pos tags, and we're using "one hot encoding" to represent them as input for our FFNN. Our goal is to train the model to predict the relationships between these chunks based on their Pos tags

From figure 2

Input=[“NNP”,”NP2”]

After one hot encoding=[[1,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,1,0,0,0,0,0,0,0,0,0,0,0,0,0]]

```
Number of feature=2
input_size = 2X15=30 # Number of features after one-hot encoding
hidden_size = 64  # Number of units in the hidden layer
num_classes = 3  # Number of output classes
num_epochs=10
```

Results:

	precision	recall	f1-score	support
R	0.68	0.73	0.70	10448
L	0.64	0.90	0.75	4174
U	0.85	0.75	0.80	21519
accuracy			0.76	36141
macro avg	0.72	0.79	0.75	36141
weighted avg	0.78	0.76	0.77	36141

3. POS chunk LRU:
This Model used POS tags(tag of head word) and the tag of chunk to predict the Edge relationship.

```
chunk_tags": ["NP", "VGF", "BLK", "CCP", "JJP", "VGNN", "VGNF",
"RBP", "NEGP", "ROOT"]
```

```
"pos_tags": ["NNP", "NN", "VM", "QC", "SYM", "PRP", "CC", "NST",
"JJ", "WQ", "QF", "RB", "QO", "NEG", "ROOT"]
```

We have 15 different Pos tags,and 10 chunk tags, and we're using "IndicNLP" pre-trained embedding. With the help of embedding we find the embedding of each pos tag,chunk tag of dimension 100 to represent them as input for our FFNN.

From figure 2
Input=[“NN”,“NNP”,”NN”,”NP2”]
Input size after embedding=4X100=400

```
Number of feature=4
input_size = 400 # Number of features after one-hot encoding
hidden_size = 64  # Number of units in the hidden layer
num_classes = 3  # Number of output classes
num_epochs=10
```

Results

	precision	recall	f1-score	support
R	0.68	0.73	0.71	10448
L	0.71	0.85	0.78	4174
U	0.85	0.78	0.81	21519
accuracy			0.78	36141
macro avg	0.75	0.79	0.76	36141
weighted avg	0.78	0.78	0.78	36141

4. Word LRU:
This Model uses the head words to predict the edge relationship. we're using "IndicNLP" pre-trained embedding. With the help of embedding we find the embedding of Hindi words in dimensions 100.

From figure 2
Input=["एशिया","मस्जिदों"]
Input size after embedding=2 X 100=200

```
Number of feature=4
input_size = 2 X 100=200# Number of features after one-hot encoding
hidden_size = 64  # Number of units in the hidden layer
num_classes = 3  # Number of output classes
num_epochs=10
```

Result

	precision	recall	f1-score	support
R	0.73	0.75	0.74	10448
L	0.76	0.86	0.80	4174
U	0.86	0.82	0.84	21519
accuracy			0.80	36141
macro avg	0.78	0.81	0.79	36141
weighted avg	0.81	0.80	0.80	36141

5. Word Chunk LRU :
This Model uses the head words and chunk to predict the edge relationship. We're using "IndicNLP" pre-trained embedding. With the help of embedding we find the embedding of Hindi words and chunk Tag .

From figure 2
Input=["एशिया","NP","मस्जिदों","NP"]

Input size after embedding=4X 100=400

```
Number of feature=4
input_size = 2 X 100=200# Number of features after one-hot encoding
hidden_size = 64  # Number of units in the hidden layer
num_classes = 3  # Number of output classes
num_epochs=10
```

Results

	precision	recall	f1-score	support
R	0.76	0.78	0.77	10448
L	0.75	0.93	0.83	4174
U	0.88	0.83	0.86	21519
accuracy			0.83	36141
macro avg	0.80	0.85	0.82	36141
weighted avg	0.83	0.83	0.83	36141

6. Word Pos LRU:

This Model uses the head words and pos of head word to predict the edge relationship. We're using "IndicNLP" pre-trained embedding. With the help of this we find the embedding of Hindi words and Pos Tag .

From figure 2
Input=["एशिया","NNP","मस्जिदों","NN"]

Input size after embedding=4X 100=400

```
Number of feature=4
input_size = 4 X 100=400# Number of features after one-hot encoding
hidden_size = 64  # Number of units in the hidden layer
num_classes = 3  # Number of output classes
num_epochs=10
```

Results

R	0.76	0.77	0.77	10448
L	0.74	0.91	0.82	4174
U	0.88	0.83	0.85	21519
accuracy			0.82	36141
macro avg	0.79	0.84	0.81	36141
weighted avg	0.83	0.82	0.82	36141

7. Word pos chunk LRU:

This Model uses the head words of chunk, chunk tag,head word tag to predict the edge relationship.

From figure 2
Input=[“एशिया”,”NP”,”NNP”,”मस्जिदों”,”NP”,”NN”]

Input size after embedding=6X 100=600

```
Number of feature=6
input_size = 6 X 100=600# Number of features after one-hot encoding
hidden_size = 64  # Number of units in the hidden layer
num_classes = 3  # Number of output classes
num_epochs=10
```

Result

	precision	recall	f1-score	support
R	0.75	0.82	0.78	10448
L	0.75	0.92	0.82	4174
U	0.90	0.82	0.86	21519
accuracy			0.83	36141
macro avg	0.80	0.85	0.82	36141
weighted avg	0.84	0.83	0.83	36141

Conclusions

S.No.	Features	SVM Accuracy	FFNN Accuracy
1.	Chunk LRU	0.77	0.77
2.	POS LRU	0.72	0.76
3.	POS chunk LRU	0.76	0.78
4.	Word LRU	0.77	0.80
5.	Word Chunk LRU	0.78	0.83
6.	Word POS LRU	0.77	0.82
7.	Word POS Chunk LRU	0.78	0.83

In all combinations tested, including Chunk LRU, POS LRU, POS Chunk LRU, Word LRU, Word Chunk LRU, Word POS LRU, and Word POS Chunk LRU, the neural network (feed-forward network) achieved higher accuracy compared to the SVM classifier.

In the SVM case, the highest accuracy, 78%, was observed with the Word Chunk LRU combination, while the lowest accuracy, 72%, was observed with the POS LRU combination.

In contrast, the neural network (Feed-forward neural network) achieved the highest accuracy of 83% with two combinations: Word POS Chunk LRU and Word Chunk LRU, while the lowest accuracy, 76%, was observed with the POS LRU combination.

As observed in the table above, an increase in features correlates with an increase in accuracy.

Referring to reference 4, it confirms that our process is correct, as the accuracy scores align.

References:

1) Transition-based Dependency Parsing with Rich Non-local Features https://aclanthology.org/P11-2033.pdf
2) Deep Biaffine Attention For Neural Dependency Parsing https://arxiv.org/pdf/1611.01734.pdf
3) https://nlp.stanford.edu/software/nndep.html#:~:text=A%20dependency%20parser%20analyzes%20the,parse%20of%20a%20short%20sentence.&text=A%20Fast%20and%20Accurate%20Dependency%20Parser%20Using%20Neural%20Networks
4) Baseline http://web2py.iiit.ac.in/research_centres/publications/download/inproceedings.pdf
5) Transition parsing https://aclanthology.org/P11-2033.pdf
6) https://web.stanford.edu/~jurafsky/slp3/old_oct19/15.pdf
7) http://pages.di.unipi.it/attardi/Paper/LREC-PParsing.pdf
8) https://medium.com/data-science-in-your-pocket/dependency-parsing-associated-algorithms-in-nlp-96d65dd95d3e
9) https://indicnlp.ai4bharat.org/pages/home/

Dataset:

1. HDTB Data corpus taken from LTRC, IIIT Hyderabad https://ltrc.iiit.ac.in/treebank_H2014/