INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY

HYDERABAD

Project Interim Report

(CS7.401 - Introduction to NLP)

Topic: Dependency Parsing

Submitted by

Team: NSU_NLP

Team No: 53

Member 1: Nikhil Chawla (2022201045)

Member 2: Shubham Deshmukh (2022201076)

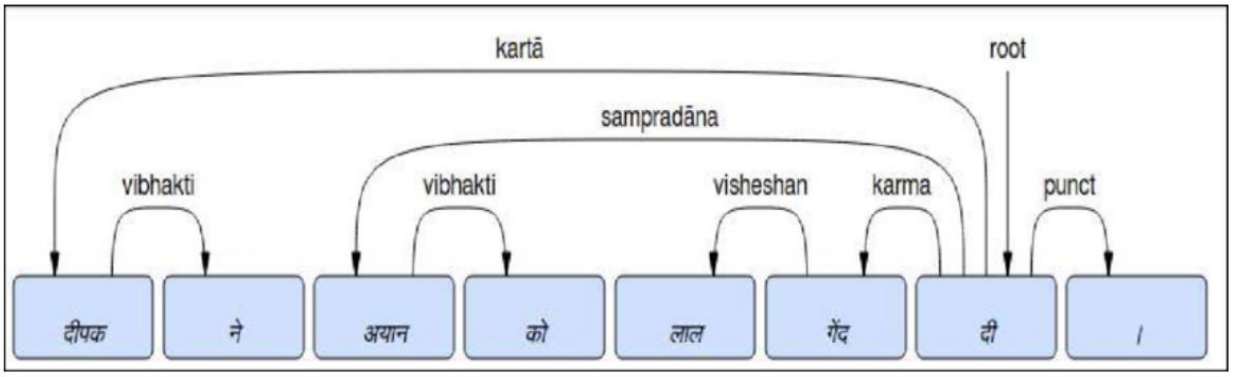
Member 3: Udrasht Pal (2022201020)

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.

Work done so far

Interim: Setup data preprocessing and Feature-Extraction pipeline. Implement a parser with greedy choice as a swappable module.

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.

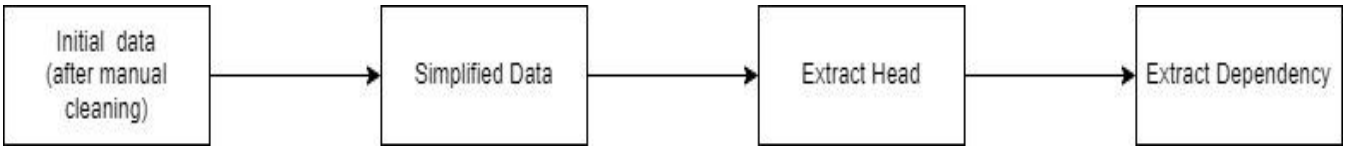
2. Data Extraction: The next involves extraction of information about constituents, terminal nodes, sentences, parts of speech and other relevant information.

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

Checking the Processed Data on Baseline SVM

Support Vector Machine (SVM) is a supervised machine learning algorithm primarily used for classification tasks. In dependency parsing, SVM can be employed to classify pairs of words or features extracted from a sentence into different dependency relations. Features which can be used are:

- Word embeddings: Distributed representations of words in a continuous vector space capture semantic information.
- Part-of-speech (POS) tags: Indicate the grammatical category of a word.
- Syntactic features: Information about the relationships between words, such as whether a word is a head or a modifier.

But here the feature used is Chunk tags only and it is tested for final processed data on the baseline SVM. The training data consists of pairs of words or word-feature pairs along with their corresponding dependency relations.

During training, the SVM learns a hyperplane that best separates the feature vectors corresponding to different dependency relations. The goal is to find the hyperplane that maximizes the margin.

SVM employs a loss function that penalizes misclassifications and aims to minimize classification errors while maximizing the margin.

Post training, testing is done to check and final performance is calculated with accuracy.

	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	21561
accuracy			0.77	36183
macro avg	0.74	0.78	0.75	36183
weighted avg	0.77	0.77	0.77	36183

While SVM may not achieve state-of-the-art performance compared to neural methods, it can provide a solid baseline for dependency parsing tasks.

Conclusion of work done so far

Pree process the data

- Extract useful data from raw data
- Find tag for the chunk and also find the dependency relation betweens chunks
- Find R/L dependency by using Arc-Eager approach
- Apply Baseline method (SVM approach) using chunk tags.

Accuracy with SVM (chunks tags) on test data 77%

Work plan till final submission:

- Apply different feature combinations like POS Tags, POS Tags + Chunk Tags, Word + POS + Chunk, , Word + POS, and so on. To train model and find accuracy
- We are also evaluating our model by use two parameters **,** Unlabeled Attachment Score (UAS), Labeled Attachment Score (LAS).
- Apart from that we are also implement one simple neural network for dependency parsing and calculate the performance on that model

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

Dataset:

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