HMM POS Tagger with Viterbi Algorithm and A*, Discriminative and Generative model for tagging and Word Prediction

Stage-IV Submission

Submitted By:
Raksha Sharma
Shrotri Aditya Anirudh
Tengse Anamay Gurunath

POS Tagging through Generative and Discriminative model

Generative:
Formula Applied:

$$T* = \underset{T}{argmax} \prod (P(W|T) * P(T))$$

Tested Sentences:
PNP_i VDD_did XX0_not VVI_expect TO0_to VVI_see PNP_you PUN_.
PNP_i VHB_have TO0_to VVI_go TO0_to NN1_market PUN_.

Result when run on Test Corpus: Global Precision: 89.16426557077206

Discriminative
Formula Applied:

$$T* = \underset{T}{argmax} \prod (P(T|W))$$

Tested Sentences:
PNP_i VDD_did XX0_not VVI_expect TO0_to VVI_see PNP_you PUN_.
PNP_i VHB_have TO0_to VVI_go TO0_to NN1_market PUN_.

Result when run on Test Corpus: Global Precision: 89.16328350099295

Equivalence of Discriminative and Generative Models for Unigram Assumption

Discriminative Case:

$T^* = \arg\max \prod P(t_i|w_i)$
$\quad = \arg\max \prod P(t_i,w_i)/P(w_i)$
$\quad = \arg\max \prod c(t_i,w_i)/c(w_i)$
$\quad = \mathbf{\arg\max \prod c(t_i,w_i)}$

Generative Case:

$T^* = \arg\max \prod P(w_i|t_i) * P(t_i)/P(w_i)$
$\quad = \arg\max \prod P(w_i|t_i) * P(t_i)$
$\quad = \arg\max \prod P(w_i,t_i)/P(t_i) * P(t_i)$
$\quad = \arg\max \prod P(w_i,t_i)$
$\quad = \arg\max \prod c(w_i,t_i)/n$
$\quad = \mathbf{\arg\max \prod c(w_i,t_i)}$

NEXT WORD PREDICTION

$$W_{next} = argmax\, P(W_{i+1} | W_i)$$

WORD MODEL:

Accuracy achieved when run on test corpus:

Without POS: 0.2741248930782454 for K=5

Word Tag Model for Word prediction

$$W_{next} = \underset{Wnext}{argmax}\, P(W_{next} | W_{current} Tag_{current})$$

To model this probablity we built a table which maintains the count of W_{next} given $W_{current}$ and $Tag_{current}$

<u>Accuracy achieved when run on test corpus:</u>
0.29079067958484695

NOTE: For user given sentences Tags are assigned using HMM Pos tagger.

Example of word Prdiction K=5

PNP_i VVB_want TO0_to VVI_book AT0_a NN1_room PUN_.

With word+Tag
i==> have was am think do
want==> to the a you .
to==> be have make do take
book==> his
a==> new few good very lot
room==> . for and status to

With word only:
i==> have was am think do
want==> to a the him .
to==> the be a have make
book==> . is which of about
a==> new few good very lot
room==> . for and status to

A*

Heuristic: For each node we calculate the no. of steps required to reach the goal from that node and multiply it by the weight of the least cost arc in the graph.

- Once the heuristic is calculated, A Star runs much faster than Viterbi for both long and short sentences.
- Viterbi calculates the probabilities for 61*61*N possible arcs.
- However calculating the heuristic is quite expensive in A – Star which has about the same time complexity as Viterbi.
- While we could have used some random low value for the least cost arc, it is not always guaranteed that the algorithm will work in that case.

Test Case for Viterbi and A*

The most devastating storm in decades to hit the most densely populated US region cut off modern communication and left millions without power on Tuesday .

Total number of steps :58
AStar=====>

AT0_the AV0_most AJ0_devastating NN1_storm PRP_in NN2_decades TO0_to VVI_hit AT0_the AV0_most AV0_densely VVN_populated NP0_us NN1_region VVB_cut AVP_off AJ0_modern NN1_communication CJC_and VVD_left CRD_millions PRP_without NN1_power PRP_on NP0_Tuesday PUN_.

Viterbi====>

AT0_the AV0_most AJ0_devastating NN1_storm PRP_in NN2_decades TO0_to VVI_hit AT0_the AV0_most AV0_densely VVN_populated NP0_us NN1_region VVB_cut AVP_off AJ0_modern NN1_communication CJC_and VVD_left CRD_millions PRP_without NN1_power PRP_on NP0_Tuesday PUN_.

to be or not to be is a question that has been puzzling the human mind since a long time .

AStar=====>
TO0_to VBI_be CJC_or XX0_not TO0_to VBI_be VBZ_is AT0_a NN1_question CJT_that VHZ_has VBN_been VVG_puzzling AT0_the AJ0_human NN1_mind PRP_since AT0_a AJ0_long NN1_time PUN_.

Viterbi=======>
TO0_to VBI_be CJC_or XX0_not TO0_to VBI_be VBZ_is AT0_a NN1_question CJT_that VHZ_has VBN_been VVG_puzzling AT0_the AJ0_human NN1_mind PRP_since AT0_a AJ0_long NN1_time PUN_.

PARSER PROJECTION
(Hindi Parser)

Motivation:
Since pre built open source English parsers are available,They can be used to generate Hindi Parser.

Methodology for Parser Projection:

1. Give input as Hindi Sentence .

2. Translate Hindi sentence into English sentence using standard translator.

3. Grammar used for English sentence parsing must be in CNF.

4. Generate English sentence parse tree using NLTK or Stanford Parser.

5. Parser gives bracketed grammatical structure of sentence.

6. If a non-terminal(except Start non terminal) dominates two non-terminal in the English parse tree,swap both the non-terminal's sub trees.

6. Repeat step '6' for each non terminal.

7. Use the respective Hindi lexicon in place of English lexicon .

NLTK

The NLTK modules include:

Tokenization : classes for representing and processing individual elements of text, such as words and sentences .

Tree : Classes for representing and processing hierarchical information over text.

CFG : Classes for representing and processing context free grammars.

FSA: Finite state automaton.

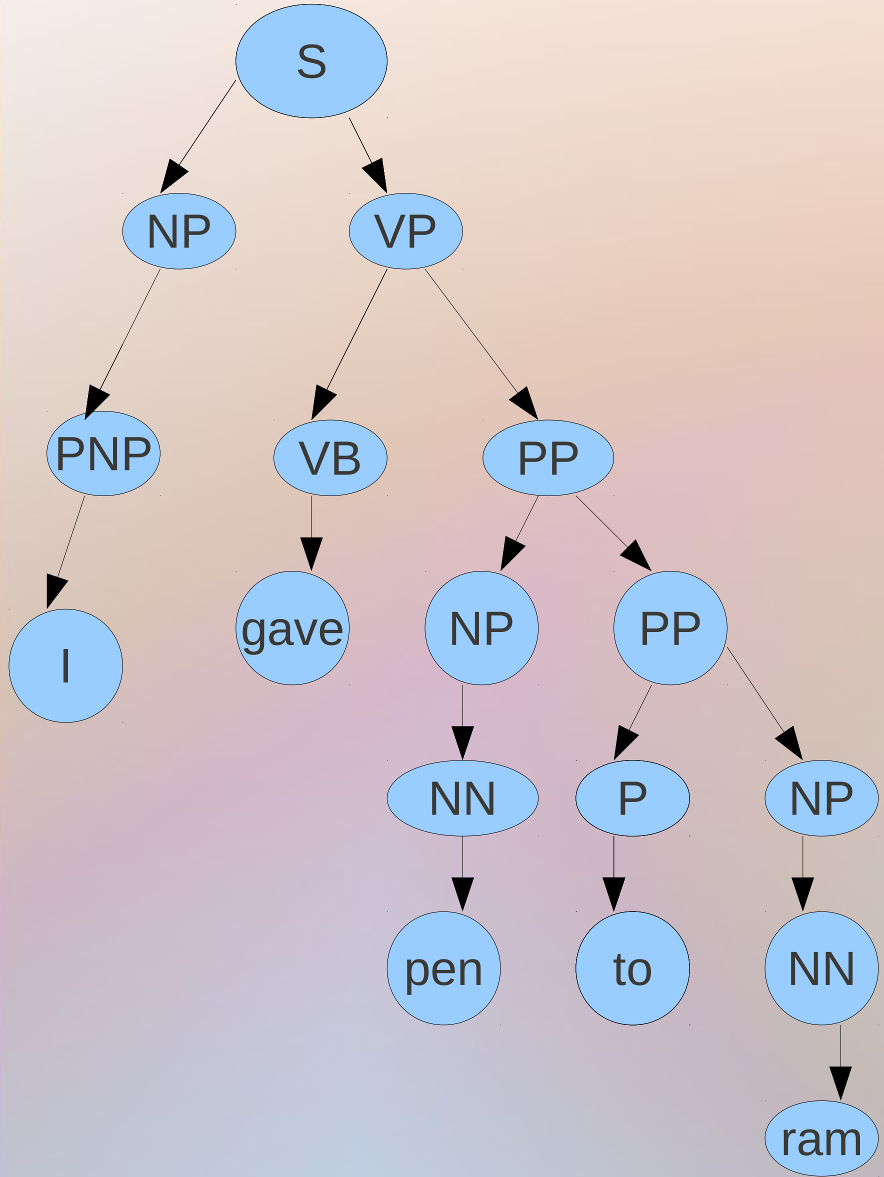
Tagger: Tagging each word with a part-of-speech, a sense, etc .

Parser: Building trees over text (includes chart, chunk and probabilistic parsers) .

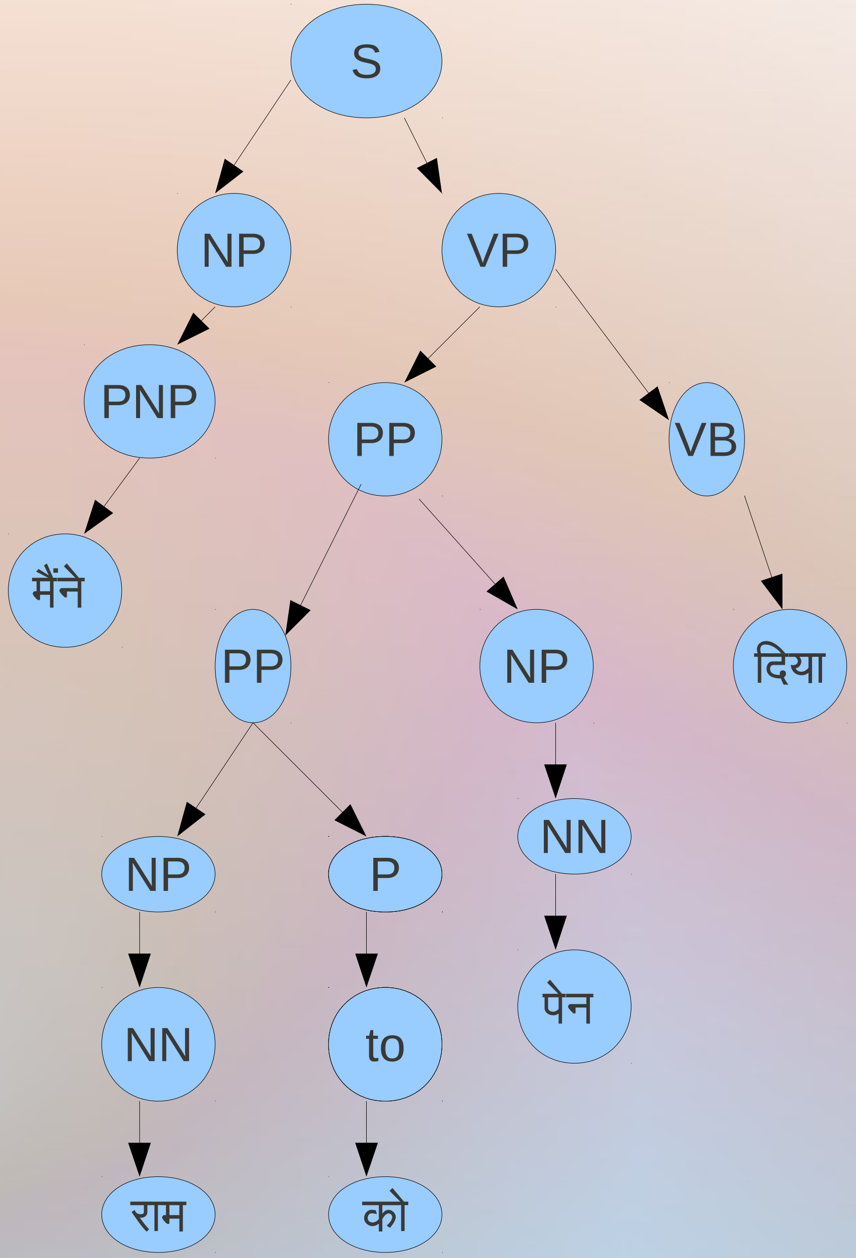
Corpus: Access (tagged) corpus data.

Parsing

I gave pen to ram .



मैंने राम को एक पेन दिया .



Challenges in Parser projection

- It is not always true that there will be a proposition with respect to the hindi sentence case marker .

- Ex: " राम ने सेब खाया “. “ Ram ate the apple.”

Solution: Introduce a new variable C which derive all possible case markers.

NN -> NN C
NN-> राम / सेब
C -> का / की / के / ने / को

Note : If we consider the translation of **“ram”** is **“** राम ने **"** then a huge number of lexicon are possible with respect to a signle english lexicon .

- There is no translation in Hindi for English lexicon “a,an,The”

- Ex: “Delhi is the capital of India” " दिल्ली भारत की राजधानी है.”

Solution: Drop the lexicon “the” from English parse tree.

- Unlike European languages Hindi does not have fixed location of words.

Ex. राम ने सीता को देखा .	सीता को राम ने देखा .

Challenges in parser projection (challenges with English parser)

* Multi word Name Entities

Ex - " Gupta and Sons Maruti Motor Dealers"

* One word can play multiple semantic rolls.

Ex - " Dogs dogs dog dog dogs"

* Proper intonation can disambiguate the given sentence in speaking but such sentences are difficult to be parsed by parser.

"The player kicked the ball kicked him"

```
(ROOT
  (S
    (NP (DT The) (NN player))
    (VP (VBD kicked)
      (SBAR
        (S
          (NP (DT the) (NN ball))
          (VP (VBD kicked)
            (NP (PRP him))))))))
```

Results Obtained by Yago

Enter entity 1 Sachin Tendulkar
Enter entity 2 Asha Bhosle

<Sachin_Tendulkar>---<hasWonPrize>---><Padma_Vibhushan><---<hasWonPrize>--- <Asha_Bhosle>

Enter entity 1 Brett Lee
Enter entity 2 Asha Bhosle

<Brett_Lee>---<created>---><You're_the_One_for_Me><---<created>---<Asha_Bhosle>

Work done

- Creation of Emission table and Transition table.
- Implementation of viterbi.
- Tagger is trained for given BNC corpus and BNC tag set.
- It is able to tag user given sentences.
- Pos tagger's global precision using viterbi algo is 93.7460795863103
- Implementation of Discriminative and Generative Model for POS Tagging.
- Next word prediction with or without Pos Tagging.
- Pos Tagging with A*.
- Finding relation between two entities using Yago database.

Thank You.