MORPHOLOGICAL ANALYSIS FOR HINDI LANGUAGE

SAQEEB [22111053]
SANKET SANJAY KALE [22111052]
ABHINAV KURUMA [22111401]

Emails : saqeeb22@iitk.ac.in,sanketsk22@iitk.ac.in,akuruma22@iitk.ac.in

Department of Computer Science,

Indian Institute of Technology Kanpur (IIT Kanpur)

1. Abstract

Morphological analysis is an important part of Natural Language Processing. With this, we can translate the language easily. A morphological analyzer can be implemented effectively for a language rich in morphemes. Hindi is a morphologically rich language. In this paper, we focus on the design of a morphological analyzer that includes POS tagging, lemmatizer, predicting gender, case, and number of Hindi languages. The analyzer takes a Hindi sentence or a word as input and analyzes it to generate its necessary features with its root words. Here we have used a rule-based for finding root words. For pos tagging, we have used the statistical approach. For predicting gender, case, and number we have used both the deep learning model and the Rule-based model. We compared statistics of the rule-based approach to the deep learning based approach.

2. Introduction

We live in a translingual society, and in order to communicate with people from different parts of the world we need to have to know their respective languages. Learning all these languages is not possible, so we need a mechanism that can do this task for us. Machine translators have emerged as a tool that can perform this task. In order to develop a machine translator we need to develop several different rules. The very first module that comes in the machine translation pipeline is morphological analysis.

Morphological analysis is one of the most important parts of linguistic analysis where we study the structure of words. This analysis is used to segment the words into morphemes. For the analysis of text in any language morphological analyzer is one of the foremost steps. When we talk about the language the first thing that comes to our mind is a “Word”. So in order to have knowledge about any language we need to know about its word structure.

Language like Hindi is morphologically rich and needs a keen analysis of words so that we can acquire their meaning and grammatical information. Lemmatization is the most important part of morphological analysis. With lemmatization, we come to know the root word.

Through this analysis we come to know the category, gender, number, case, etc. thus we get all the information about a word which tends us to understand the language.

3. Related Work

3.1 Lemmatization

A lot of research work has been done and is still going on for the development of a stemmer as well as a lemmatizer. The first stemmer was developed by Julie Beth Lovins[1] in 1968. There is so much research done on lemmatizing English and other European languages. But For Hindi little has been done and there is no such great work done by anyone till now for analyzing Hindi words or basically Indian languages.
A rule-based approach proposed by Plisson et al. [2] is one of the most accepted lemmatizing algorithms. It is based on the word endings where the suffix should be removed or added to get the normalized form. It emphasizes on two-word lemmatization algorithm which is based on if-then rules and the ripple-down approach. The work proposed by Goyal et al. [3] focuses on the development of a morphological analyzer and generator. They aimed to develop a translation system, especially from Hindi to Punjabi.

3.2 POS Tagging

Miikka Silfverberg et al.[4] created the pos tagger using conditional random fields and it performed well on the English corpus. Nisheet Joshi et al.[5] did an analysis on the Hindi corpus by creating a pos tagger using HMM. We took motivation from them for our pos tagger using HMM.

3.3 Gender, Case and Number

Ankita Agarwal et al.[6] performed rule-based analysis of the Hindi language from whom we took inspiration for our rule-based analysis for gender, case and number.

4. Methodology

4.1 Lemmatization

For successfully analysing, we first studied and identified the inflectional and derivational suffixes. Different rules were made to extract features from given input words. Here, the root word is extracted by using lemmatize which is also rule-based and other categories are extracted by using the rules made. For this reason, some commonly used words were taken for the development of our corpus. Corpus has been designed from the raw data that we gathered from the internet and books. The corpus is aligned in a proper way so that we can study each word individually without any errors.

We have a training "conllu" file that contains an analysis of words and their lemma and from that, we are getting root words. We first accessed the file and make a list of lemma which contains root words and make a dictionary of words and lemma so that we can use this when testing a word. Now let's see the details of the algorithm and flowchart of how rule-based architecture is done.

4.1.1 Algorithm

Taking input a word or a sentence if it is a sentence it can have other English letters or numbers so we have to clean the data and then append words of that sentence into the input list so that we can give the input word by word. Sometimes a word itself is a root word so by checking our root words we can say whether the input is in root form or not. If we found it as a root word then display that root word. And If the given input is not a root word then we go to checking the rules. Here is the flow chart :

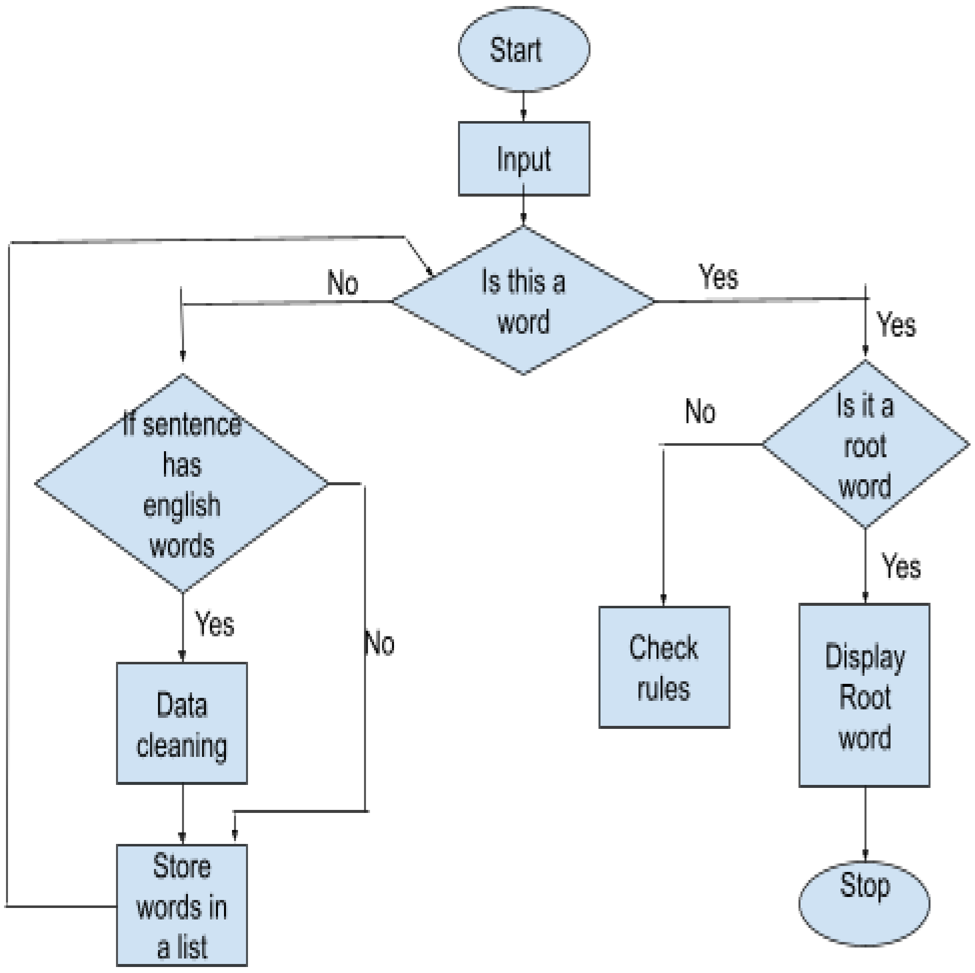


Fig: Flowchart to find lemma if given input is in root form

4.1.2 Algorithm for checking rules

We have a list of suffixes which were commonly used in the Hindi language and which we are storing in a list so that we can use them in our Rules checking method. And the suffixes which we used are below:

```
suffixes =["ो", "े", "ू", "ु", "ी", "ि", "ा","तृ","ान","ैत","ने","ाऊ","ाव","कर", "ाओ", "िए", "ाई", "ाए", "नी",
"ना", "ते", "ीं", "ती","ता", "ाँ", "ां", "ों", "ें","ीय", "ति","या", "पन", "पा","ित","ीन","लु","यत","वट","लू",
"ेरा","त्व","नीय","ौनी","ौवल","ौती","ौता","ापा","वास","हास","काल","पान","न्त","ौना","सार","पोश","नाक","ियल",
"ैया", "ौटी","ावा","ाहट","िया","हार", "ाकर", "ाइए", "ाईं", "ाया", "ेगी", "वान", "बीन","ेगा", "ोगी", "ोगे", "ाने",
"ाना", "ाते", "ाती", "ाता", "तीं", "ाओं", "ाएं", "ुओं", "ुएं", "ुआं","कला","िमा","कार","गार", "दान","खोर",
"ावास","कलाप","हारा","तव्य","वैया", "वाला", "ाएगी", "ाएगा", "ाओगी", "ाओगे", "एंगी", "ेंगी", "एंगे", "ेंगे", "ूंगी",
"ूंगा", "ातीं", "नाओं", "नाएं", "ताओं", "ताएं", "ियाँ", "ियों", "ियां","त्वा","तव्य","कल्प","िष्ठ","जादा","क्कड़", "ाएंगी",
"ाएंगे", "ाऊंगी", "ाऊंगा", "ाइयाँ", "ाइयों",
"ाइयां","अक्कड़","तव्य:","निष्ठ""ो","े","ू","ु","ी","ि","ा","कर","ाओ","िए","ाई","ाए","ने","नी","ना","ते","ीं","ती
","ता","ाँ","ां","ों","ें","ाकर","ाइए","ाईं","ाया","ेगी","ेगा","ोगी","ोगे","ाने","ाना","ाते","ाती","ाता","तीं","ा
ओं","ाएं","ुओं","ुएं","ुआं","ाएगी","ाएगा","ाओगी","ाओगे","एंगी","ेंगी","एंगे","ेंगे","ूंगी","ूंगा","ातीं","नाओं","नाएं
","ताओं","ताएं","ियाँ","ियों","ियां","ाएंगी","ाएंगे","ाऊंगी","ाऊंगा","ाइयाँ","ाइयों","ाइयां"]
```

1. If the given input word is not ending with any of the suffixes then we are assuming that it is a lemma trivially.
2. If input ends with a suffix then we are removing the suffix and checking if the remaining word is a root word.
3. If that is not a root word then We add another suffix so that maybe after adding it becomes a root because for some examples we have to remove a suffix and add another suffix so that it becomes a meaningful word.
4. For example लड़कियाँ when deleting the suffix becomes लड़क but it is not a meaningful word and it is actually done by stemming not by lemmatization and we have to create a meaningful root word and so we have to add another suffix and if we add या to लड़क then we get लड़की and it is the root word. Likewise, there are so many words which can get from removing and adding suffixes and so some of them are given below.

		Rule application	
Word	**Root**	**Extraction of suffix**	**Addition of character**
लड़कियों	लड़की	ियों	ी
कहानियों	कहानी	ियों	ी
कवियों	कवि	ियों	ि (exception)
चिड़ियों	चिड़िया	ियों	िया (exception)

Fig: Some examples when extracting and adding of suffixes give root form

5. Likewise, we add another suffix and check whether the obtained word is the root word or not, and gives the output when it matches

6. When the suffix list is over and we are not getting any matching then we are just giving the output which is obtained after removing the suffix.
7. Here is the whole flow chart we are using to form a root word and display the meaningful word.

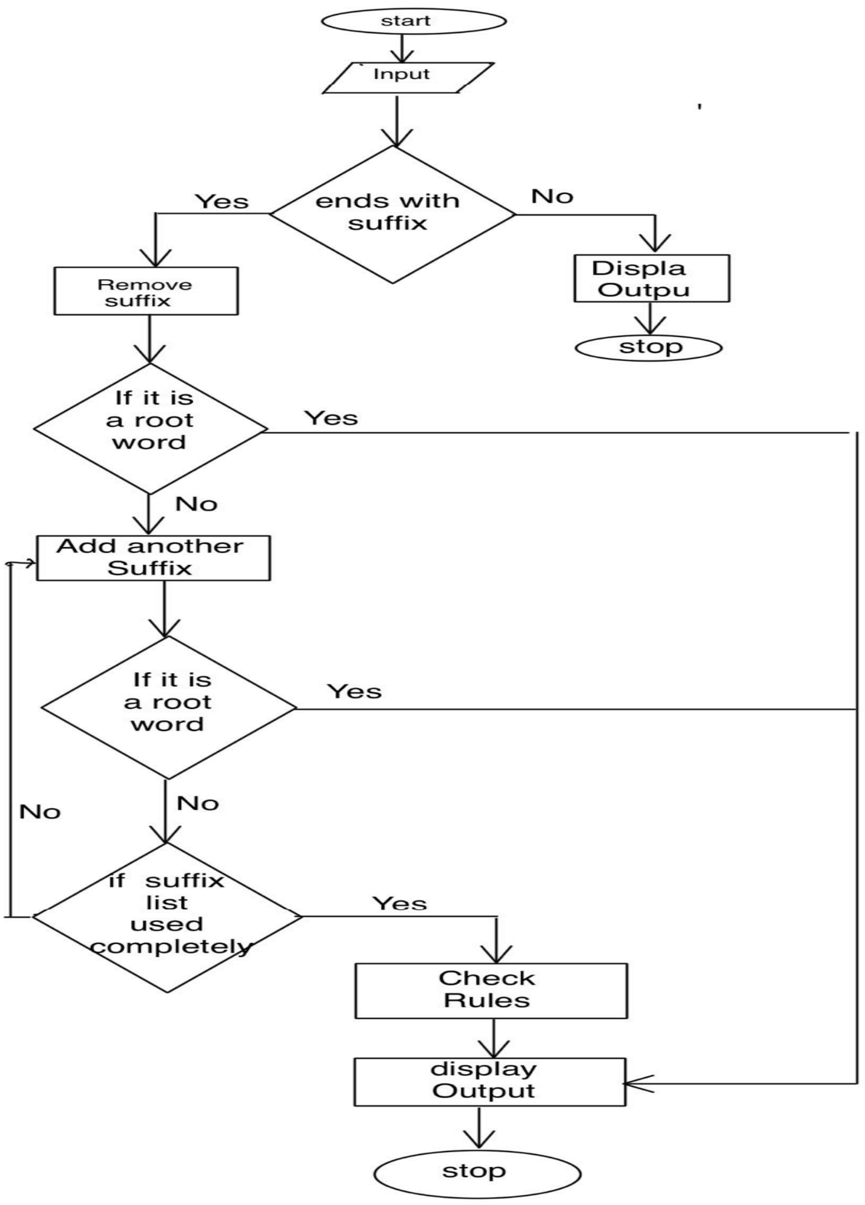


Fig: Flowchart for checking rules and giving the lemma

4.2 POS tagging

4.2.1 Algorithm (Hidden Markov Model using Viterbi algorithm)

a) Calculating Transition Probability

Storing of words and their corresponding tags is done. Then we stored the frequency of each tag. Afterwards, we stored the frequency of each combination of tags. Probability of a tag to be followed

by another tag out of all possible tags. Then added the probabilities of the combination NOT in the dictionary with minimin probability. If a tag does not occur as a start tag, then set its probability to be a start tag to the minimum value. If a particular tag combination does not exist in the dictionary, we set its probability to a minimum.
b) Calculating Emission Probabilities

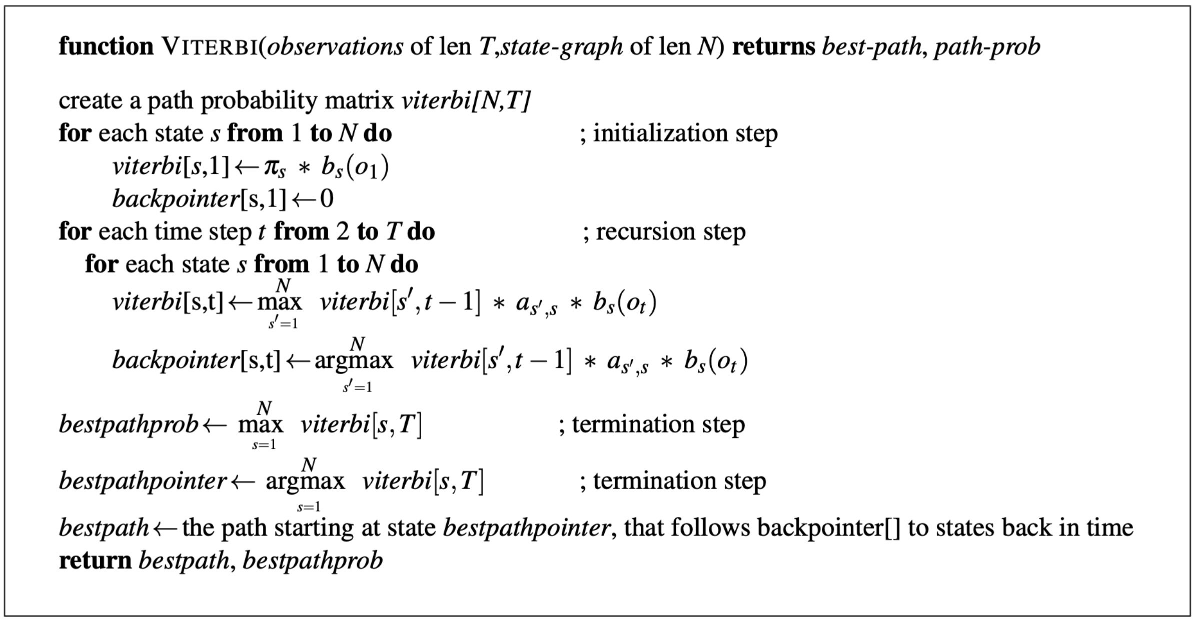
We mapped the words in the training set to their tagged POS. We calculated the probability of a word being a certain tag out of all the possible tags that it can be.

c) Viterbi Algorithm

Then we used the Viterbi algorithm for obtaining the maximum a posteriori probability estimate of the most likely sequence of hidden states.

Afterwards, we analysed algorithm predictions by calculating micro and macro precisions.

The algorithm which we have used is as follows:



```
function VITERBI(observations of len T, state-graph of len N) returns best-path, path-prob

create a path probability matrix viterbi[N,T]
for each state s from 1 to N do                      ; initialization step
      $viterbi[s,1] \leftarrow \pi_s * b_s(o_1)$
      $backpointer[s,1] \leftarrow 0$
for each time step t from 2 to T do                  ; recursion step
   for each state s from 1 to N do
      $viterbi[s,t] \leftarrow \max_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$
      $backpointer[s,t] \leftarrow \operatorname{argmax}_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$
$bestpathprob \leftarrow \max_{s=1}^{N} viterbi[s,T]$              ; termination step
$bestpathpointer \leftarrow \operatorname{argmax}_{s=1}^{N} viterbi[s,T]$        ; termination step
bestpath ← the path starting at state bestpathpointer, that follows backpointer[] to states back in time
return bestpath, bestpathprob
```

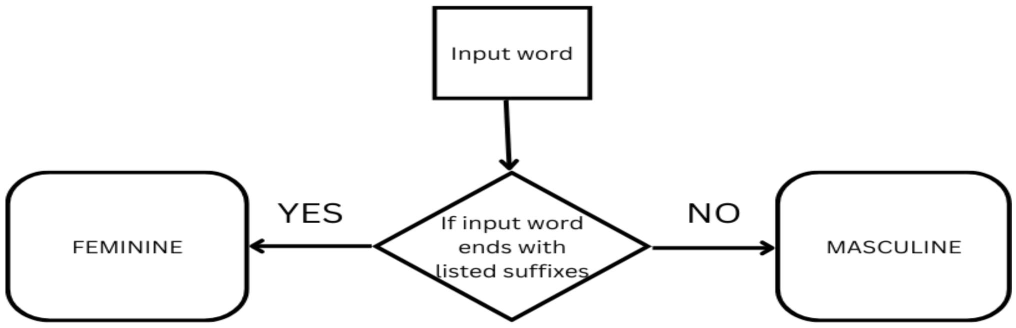
Fig: Viterbi algorithm

4.3 Gender, Case and Number Prediction

4.3.1. Gender

a) Rule-Based Approach

Each input word is going to be checked if it ends with the listed suffix below. If it ends with one of the suffixes then we are going to output feminine, otherwise, we are predicting output as masculine gender.
Suffixes = [ी, ि, ियाँ, ियां]



b) Deep learning based approach

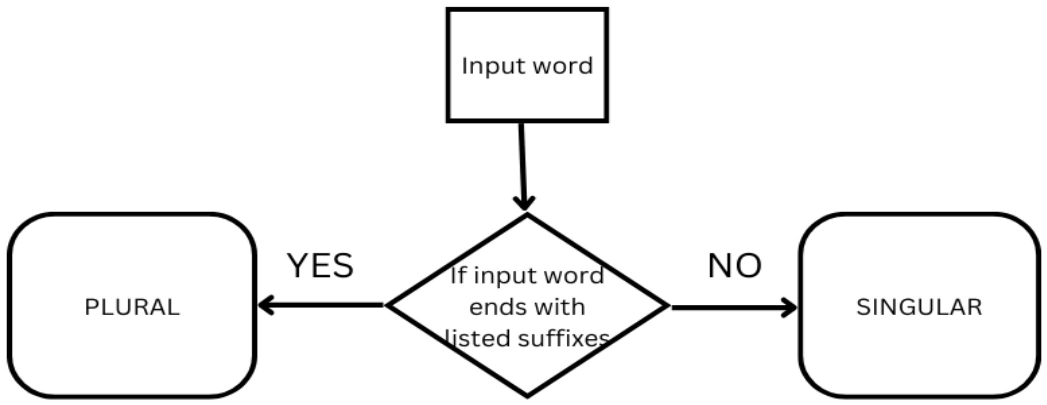
We trained a neural network model for predicting the gender of a given word. The architecture of the model is such that it contains one hidden layer whose width is 128. We used the ‘ReLu’ activation function for each node. For the output layer, we used the ‘Sigmoid’ activation function for binary classification.
Input to this model is the ‘FastText’ embedding of Hindi words which is vector embedding using the Wikipedia Hindi corpus. Labels are converted to 0 or 1 for masculine and feminine respectively.
Our dataset consists of token forms for each token and labeled them as genders of each token. We split the dataset into the train(0.8) and test(0.2) datasets.
The hyperparameters we selected are as follows :
[optimizer='adam', loss='binary_crossentropy', metrics=['accuracy']]

4.3.2 Number

a) Rule-Based Approach

Each input word is going to be checked if it ends with the listed suffix below. If it ends with one of the suffixes then we are going to output plural, otherwise, we are predicting output as a singular number.

Suffixes = [ियाँ,ियां,ियों,यो,ाओं,नों,यो] and ["ो","े","ों","ें"]



b)Deep learning based approach

We trained a neural network model for predicting the number of a given word. The architecture of the model is such that it contains one hidden layer whose width is 128. We used the ‘ReLu’ activation function for each node. For the output layer, we used the ‘Sigmoid’ activation function for binary classification.
Input to this model is the ‘FastText’ embedding of Hindi words which is vector embedding using the Wikipedia Hindi corpus. Labels are converted to 0 or 1 for singular and plural respectively.
Our dataset consists of token forms for each token and labeled them as numbers of each token. We split the dataset into the train(0.8) and test(0.2) datasets.
The hyperparameters we selected are as follows :
[optimizer='adam', loss='binary_crossentropy', metrics=['accuracy']]

4.3.3 Case

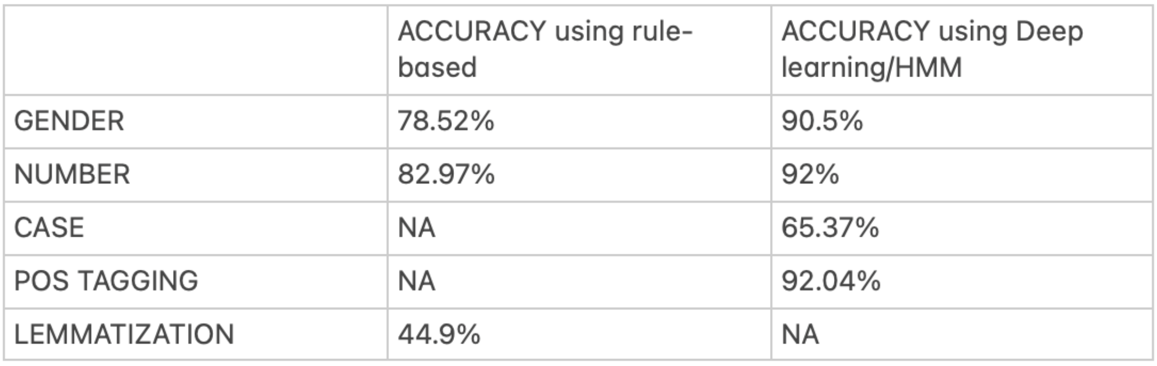
Deep Learning-Based Approach

We trained a neural network model for predicting the case of a given word. The architecture of the model is such that it contains one hidden layer whose width is 128. We used the ‘ReLu’ activation function for each node. For the output layer of size 7 representing 7 classes of cases, we used the “Softmax” activation function for multi-class classification.
Input to this model is the ‘FastText’ embedding of Hindi words which is vector embedding using the Wikipedia Hindi corpus. Labels are converted to one of 0 to 7 for representing cases and then converted each value to one hot encoding.
Our dataset consists of token forms for each token and labeled them as numbers of each token. We split the dataset into the train(0.8) and test(0.2) datasets.
The hyperparameters we selected are as follows :
[optimizer='adam', loss='binary_crossentropy', metrics=['accuracy']]

Datasets

The datasets we have used are taken from :
`https://github.com/UniversalDependencies/UD_Hindi-HDTB/tree/master`

Results



	ACCURACY using rule-based	ACCURACY using Deep learning/HMM
GENDER	78.52%	90.5%
NUMBER	82.97%	92%
CASE	NA	65.37%
POS TAGGING	NA	92.04%
LEMMATIZATION	44.9%	NA

Table: Accuracies of different attributes by different methods

Conclusion

We have discussed in this report finding the lemmas, pos tags and other morphological features such as gender, case and number for the Hindi language (Devanagari script).In the lemmatizer, we almost covered all rules for word formation to find out the lemma of a given word. We went with HMM algorithm for finding pos tags because this model gave us the maximum accuracy for finding pos tags. We compared both rule-based and deep learning based approaches for finding out gender, case and number of words. We observed that deep learning based approaches are outperforming rule-based approaches. But we still think that there is still future scope to add more morphological rules by studying more about the Hindi language. Of our limited understanding of the Hindi language, we observed that statistical and deep learning models are outperforming the rule-based models.

References

[1] Julie Beth Lovins, Development of stemming Algorithm, Mechanical Translation and Computational Linguistics, Vol. 11, No. 1, pp 22-23, 1968.

[2] Plisson, J, Larc, N, Mladenic, D.: A Rule based approach to word lemmatization, Proceedings of the 7th International Multiconference Information Society, IS-2004, Institut Jozef Stefan, Ljubljana, pp. 83-86, 2008.

[3] Vishal Goyal, Gurpreet Singh Lehal, Hindi Morphological Analyzer and Generator, IEEE Computer Society Press, California, USA pp. 1156-1159, 2008.

[4] Part-of-Speech Tagging using Conditional Random Fields: Exploiting Sub-Label Dependencies for Improved Accuracy Miikka Silfverberga, Teemu Ruokolainenb, Krister Lindéna, Mikko Kurimob

[5] Jan Zizka (Eds): HMM BASED POS TAGGER FOR HINDI by Nisheeth Joshi1, Hemant Darbari2 and Iti Mathur(2013).

[6] Morphological Analyser for Hindi – A Rule Based Implementation Ankita Agarwal1, Pramila2, Shashi Pal Singh3, Ajai Kumar4, Hemant Darbari5.

[7] Bharti Akshar, Vineet Chaitanya, Rajeev Sangal, Natural Language Processing: A Paninian Perspective. Prentice-Hall of India, 1995.