Bachelors Of Technology Project Report

# MORPHOLOGICAL ANALYZER FOR HINDI


Faculty Mentor : Dr Gaurav Harit

Semester 4

Students: Kunal Jangid (B17CS030)

Abhinav Pandey (B17CS001)

# Contents

# Acknowledgements

The success and final outcome of this project required a lot of guidance and assistance from many people and we are extremely privileged to have got this all along the completion of my project. All that we have done is with help of such supervision and assistance. We respect and thank Dr  Gaurav Harit, for providing us an opportunity to do the project work and giving us all support and guidance his innovative ideas, excellent guidance, constant encouragement, invaluable suggestions, and support during our BTP which made us complete the project duly. We are extremely thankful to him for providing such a nice support and guidance, although he had busy schedule managing the Institute affairs.

We are thankful to and fortunate enough to get constant encouragement, support and guidance from all Teaching staffs of CSE Department which helped us in successfully completing our project work. Also, I would like to extend our sincere esteems to all staff in laboratory for their timely support.

Last, but not the least, we express gratitude to our families. Without their support we would never have been able to venture into all the activities that we did. We will always be grateful to our family for their constant prayers and immense support when we needed them the most.

Kunal Jangid(B17CS030) Abhinav Pandey(B17CS001)

# 1.Introduction

Since the beginning of time, NLP, ie Natural Language Processing has been at the very core of deriving meaning and context out of textual representation of information and data.

NLP makes it way everywhere, even if we don't realize, it makes our email application automatically correct itself when trying to send emails.

It assists beyond our expectations in Machine Translation tasks, as the amount of data available to our disposal is growing online. Overload of presented information is posed as a real problem, when we need to access a particular valuable information source from a huge knowledge base.

Morphology (being a big part of NLP) is the study of processes of word formation and also the linguistic units such as morphemes, affixes in a given language.

It consists of two branches: inflectional morphology and derivational morphology.

**Inflectional morphology**

Inflectional morphemes do not change the parts of speech of the word.

For example : plurals: bus –> buses, car –> cars

past tense: die –> died, fill –> filled

aspect: footstamp –> footstamping, microcode –> microcoding

**Derivational morphology**

Derivational morphemes are involved in changing the part of speech of the word.

For example : adjective to noun: happy –> happiness

adjective to verb: commercial –> commercialize

adjective to adjective: green –> greenish

noun to adjective: success –> successful

verb to adjective: sail –> sailable

# Problem Statement

The main objective of our work is to develop a tool which executes both inflectional and derivational morphological analysis of Hindi. Before understanding the working of our Hindi morphological analyzer, one needs to understand the working of a common English derivational morphological analyzer. Let us consider a few examples from English. Example: consider the word *happiness*. If I give this word as an input to a common English derivational analyzer, first the tool should be able to tell me that the word happiness is a noun. This segment is called inflectional morphological analysis. Then it should tell me that *happiness* (noun) = *happy* (adjective) + *ness*. It implies that the tool must provide the information about the word (root word) from which the main word is derived. Here it must tell us that it is derived from the word *happy* which is an adjective. It must also give the information about the derivational suffix. A derivational suffix is the suffix with which a root interacts to form a derivation. Here the derivational suffix is *ness*. This whole second segment is called derivational morphological analysis. Our aim is to build a tool which executes the derivational morphological analysis (along with inflectional analysis) of Hindi. The problem with Hindi morphological analysis is that the root word of a word can not be simply stripped, there are some variations in the root word after stripping like changing of matra ी to ि, eg - शारीरिक has a root word as शरीर.

# Abstract

Hindi is an Indian language which is relatively rich in morphology. Few morphological analyzers of this language have been developed. However, they do only inflectional analysis. None of them give the derivational analysis of the language. In this thesis, we present our Hindi derivational morphological analyzer which gives us derivational morphological analysis of Hindi. Our Morphological

Analyzer has been divided into three parts- Root word extraction, Identification as Singular/Plural, Masculine/Feminine, and last part as Part Of Speech Tagger.

# Motivation

Morphological analysis is an important step for any linguistically informed natural language processing task. In general, morphological analyzers of many languages perform only inflectional analysis. However, derivational analysis is highly crucial for better performance of several systems. Derivational morphological analyzers are used in a wide variety of NLP applications.

The following are some of the prominent areas where the use of derivational morphology is important :

• Machine translation

• Word sense disambiguation

• Spell-Checkers

• Search engines

• Annotation of corpora

• Lexicon building

• Terminology acquisition and in detecting term variation

Hence derivational processes can often be productive in a language. The development of an effective morphological analyzer will prove beneficial in several aspects. It will be even more helpful considering the rich morphology of the Hindi language. This was the most important factor that motivated us to build a morphological analyzer for Hindi.

# 2. Literature Analysis on previous works

The following works have already been done in the field of Morphological analysis for Hindi language :

### 1) Niraj's Morphological Analyzer

A rule-based Hindi morphological analyzer was developed by Niraj Aswani and Robert Gaizauskas . The rules of this morphological analyzer were acquired semi-automatically from corpora. This analyzer can handle both prefixes and suffixes. The system returns the root form when an inflected Hindi word is given as input. It makes use of a dictionary, and a corpus to obtain suffix-replacement rules.

### 2) HCU's Morphological Analyzer

An inflectional morphological analyzer based on the paradigm model was developed by a consortium of IIIT Hyderabad and HCU. It uses the combination of paradigms and a root word dictionary to provide inflectional analysis. Given an inflected Hindi word, this inflectional analyzer returns its root form and other grammatical features such as gender, number, person, etc. For example: if the input word to the morphological analyzer is बागवानों (gardeners). The output will be बागवान (gardener), noun, m, pl, etc. Here the word बागवान is the root word of the input word. Noun is the category of the input word, m means masculine and pl means that the input word is plural in number.

# 3) Primary Paper : Hindi Derivational Morphological Analyzer

- A rule-based Hindi morphological analyzer.
- The rules of this morphological analyzer were acquired semi-automatically from corpora.
- This analyzer can handle both prefixes and suffixes.
- The system returns the root form when an inflected Hindi word is given as input.
- It makes use of a dictionary, and a corpus to obtain suffix-replacement rules.
- Most of the rules used in this approach are learnt automatically. However, a few of them were built manually.

Phase 1:

- Identifying the derivational variants in hindi by building a list of verbal nouns.
- Semi-automatic process used
- tagged a few instances manually and fed it to a Support Vector Machines (SVM) classifier.
- constructed a training data of size of 454 pairs of words where an individual pair (A, B) of this training data consists of a noun A derived from a verb B.

Phase 2:

- main aim is identifying derivational variants in Hindi based on the three main properties of hindi derivational variants:
- semantic relatedness
- phonological similarity
- presence of a suffix
- They built a graph whose nodes are all the words in Wikipedia, the pages in which these words occur, the bigrams of these words and a set of derivational suffixes
- Finally, they identified derivational variants by running the page rank algorithm on this graph.
- Overall, the approach was unsupervised in nature.

Phase 3:

- A derivational morphological analyzer was developed using the already existing inflectional analyzer (HCU's morphological analyzer) and the results obtained in first two phases by devising a rule based algorithm.

# Linguistica

## A Python Library

Linguistica 5 is a Python library for unsupervised learning of linguistic structure developed by Jackson Lee and John Goldsmith.

This library is based on analysis of morphemes for many language, but it did not give adequate output.

## Methods with a Linguistica object

**(1) word trigrams**

```
('u', '.', 's') 235
(',', 'it', 'is') 234
(',', 'and', 'he') 225
('of', 'course', ',') 220
(',', 'of', 'course') 189
```

**(2) signatures to stems**

```
('d', 's') 175
('ies', 'y') 173
('NULL', 'ed', 'ing', 's') 151
('NULL', 'ed') 134
```

And many more like stem to words, triphones, bigrams etc.

# Data

Two types of data are recognized:

1)raw corpus text

2)wordlist

While using Linguistica 5 via the Graphical user interface (GUI) or the Command line interface (CLI), then data is a file on local drive. However, if we use it as a Python library instead, data can either be a file from the local drive or an in-memory Python object.

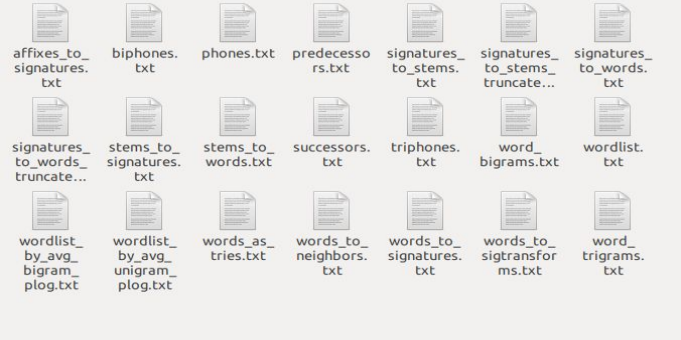
## Raw corpus text

A raw corpus text is simply a plain text file. An example is the Brown corpus (Kučera and Francis 1967) with about one million word tokens (for about 50,000 word types). This corpus is a built-in dataset that comes with Linguistica 5 – its file path is accessible as **Brown.**

## Linguistica on Hindi Data

We tried linguistica command line interface on Hindi corpus.

The following results were obtained :




These 21 features for the corpus were generated. E.g. - word_trigrams.txt looked like this:

```
73      & पीले , लाल                    \\
73      & पीले , लाल                    & 3          \\
74      & पीले रंग का                   \\
74      & पीले रंग का                   & 3          \\
75      & पोथी के चार                   \\
75      & पोथी के चार                   & 3          \\
76      & फलक के ऊपरी                   \\
76      & फलक के ऊपरी                   & 3          \\
77      & में दाहिनी तरफ                \\
77      & में दाहिनी तरफ                & 3          \\
78      & में मिलते हुए                 \\
78      & में मिलते हुए                 & 3          \\
```

# 3. OUR WORK

Different phases of our project:-

1) Preparing The Data- Root Word Database

2) Training classifier - verification of root word

3) Identifying list of suffixes and prefixes for generating root word using UNICODE (utf-8) ENCODING

4) Using k Means clustering algorithm to classify in singular/ plural and masculine/feminine

5) Using SVM classification algorithm to classify in singular/ plural and masculine/feminine

5) Identifying orthographic rules for classifying in singular/ plural

6) POS Tagging (part of speech tagging ) - Future Work

## Preparing Root Word Database :-

We found list of lemmas on the wiktionary, the hindi dictionary, website and decided to use this data in implementing the lemmatization algorithm.

Link for the Website : https://en.wiktionary.org/wiki/Category:Hindi_lemmas

Here is a screenshot of some of the words from the website :

अ

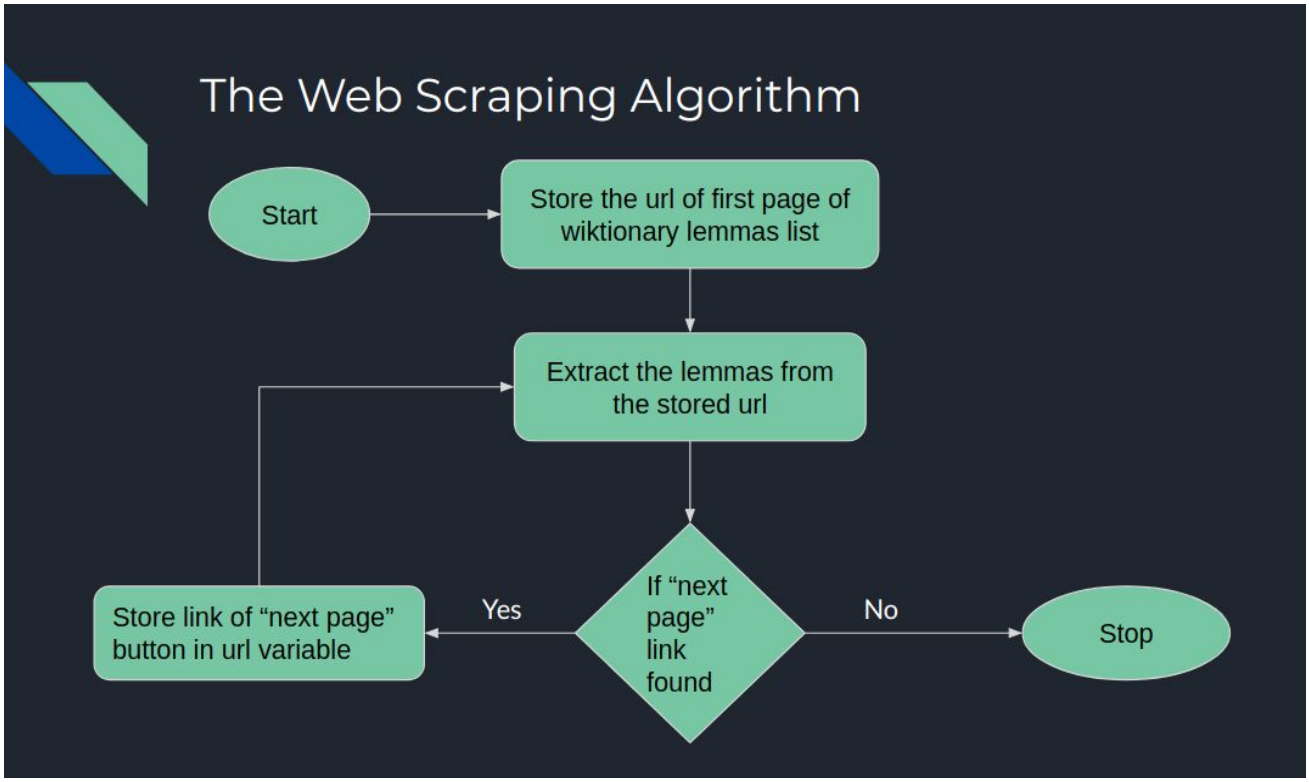
- अ
- अ-
- अँ
- अँकवैया
- अँग्रेज़ी
- अँटना
- अँधेरा
- अं
- अंक
- अंकगणित
- अंकगणितीय
- अंकन
- अंकन करना
- अंकल
- अंकारा
- अंकित
- अंकुर
- अंकुरित
- अंकुश
- अंग
- अंगकोर वाट
- अंगच्छेद
- अडा
- अंडे का शहजादा
- अंडोरा
- अंत
- अंत करना
- अंतःक्षेप
- अंतड़ी
- अंतरंगता
- अंतरजाल
- अंतरण
- अंतरराष्ट्रीय
- अंतरराष्ट्रीयता
- अंतरा
- अंतराल
- अंतरिक्ष
- अंतरिक्ष यान
- अंतरिम
- अंतर्गत
- अंतर्जातीय
- अंतर्जाल
- अंतर्देशीय
- अंतर्निहित
- अंतर्राष्ट्रीय
- अकार
- अकारण
- अकारथ
- अकाल
- अकाली
- अकूत
- अकेला
- अकेलापन
- अकेले
- अक्खड़
- अक्खा
- अक्टूबर
- अक्तूबर
- अक्लमंद
- अक्ष
- अक्षत
- अक्षम
- अक्षय
- अक्षर
- अक्षरारंभ
- अक्स
- अक्सर
- अक्साई चिन

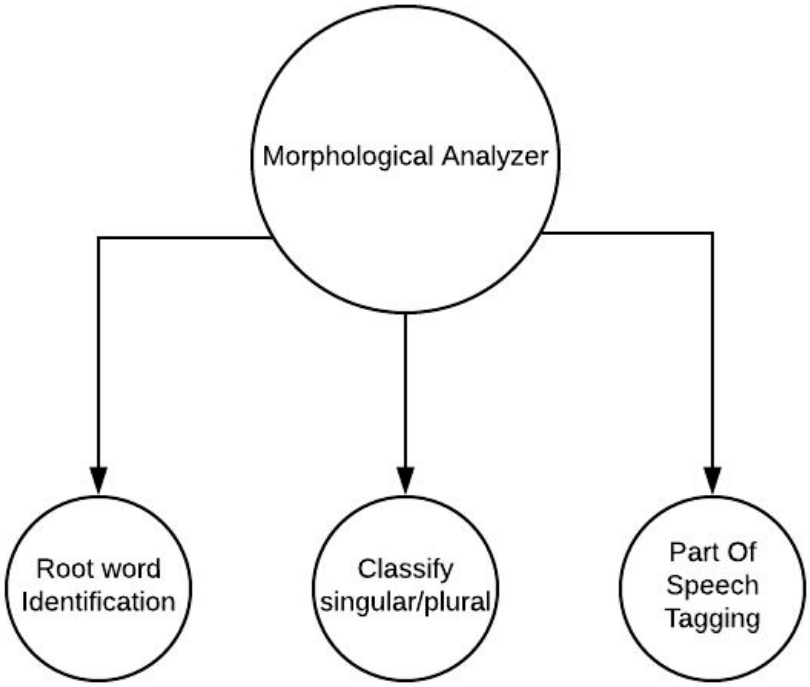
We extracted around 12,000 words present on this website distributed across 60 pages by web scraping using BeautifulSoup, a python library.

## Web Scraping ( BeautifulSoup)

Steps involved in web scraping:
Sending a HTTP request to the URL of the webpage you want to access. The server responds to the request by returning the HTML content of the webpage. For this task, we used a third-party HTTP library for python requests.

Once we had accessed the HTML content, we were left with the task of parsing the data. We used html5lib for creating a parsing tree. Now, all we needed to do was navigating and searching the parse tree that we created, i.e. tree traversal. For this task, we used another third-party python library, **Beautiful Soup**.
We successfully parsed 200 pages and extracted a data set of 11,576 words.






# Root Word Identification:-

1) Verification of Root word
2) Identifying Suffixes
3) Designing a stemmer
4) Using Stemmer To Design Lemmatizer for Unicode (utf-8) Encoding

## Verification Of Root Word:-

### Using Word2vec Approach:-

### What is word2vec?

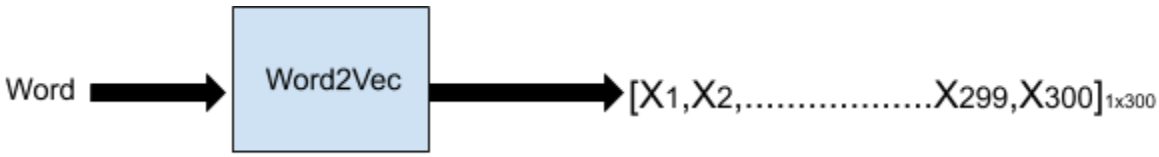
Word2vec is a two-layer neural net that processes text. Its input is a text corpus and its output is a set of vectors: feature vectors for words in that corpus. While Word2vec is not a deep neural network, it turns text into a numerical form that deep networks can understand.

In our project we used Word2vec for hindi from source -

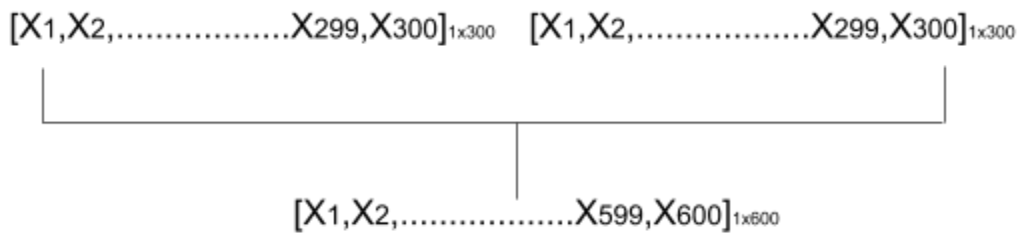
This gave us an output of numerical matrix of [300X1] . All the values of the matrix were between 0 and 1.

### Our Algorithm:-

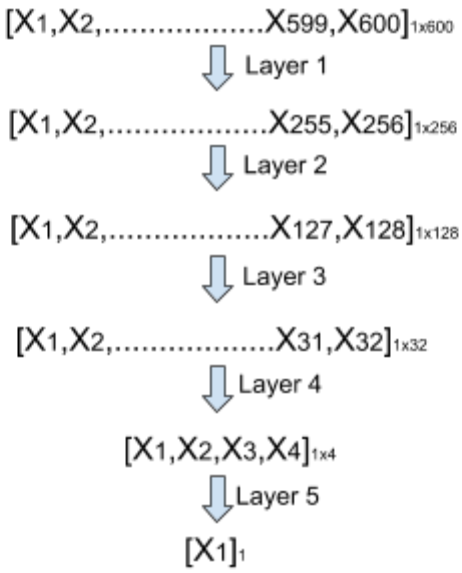
1) Converting the word and given root word to vectors of size[300X1] each.



2) Concatenating the two vectors:- to get vector of size[1X600]




3) Passing vector through neural network:- output between 1& 0.




Functions Used:-

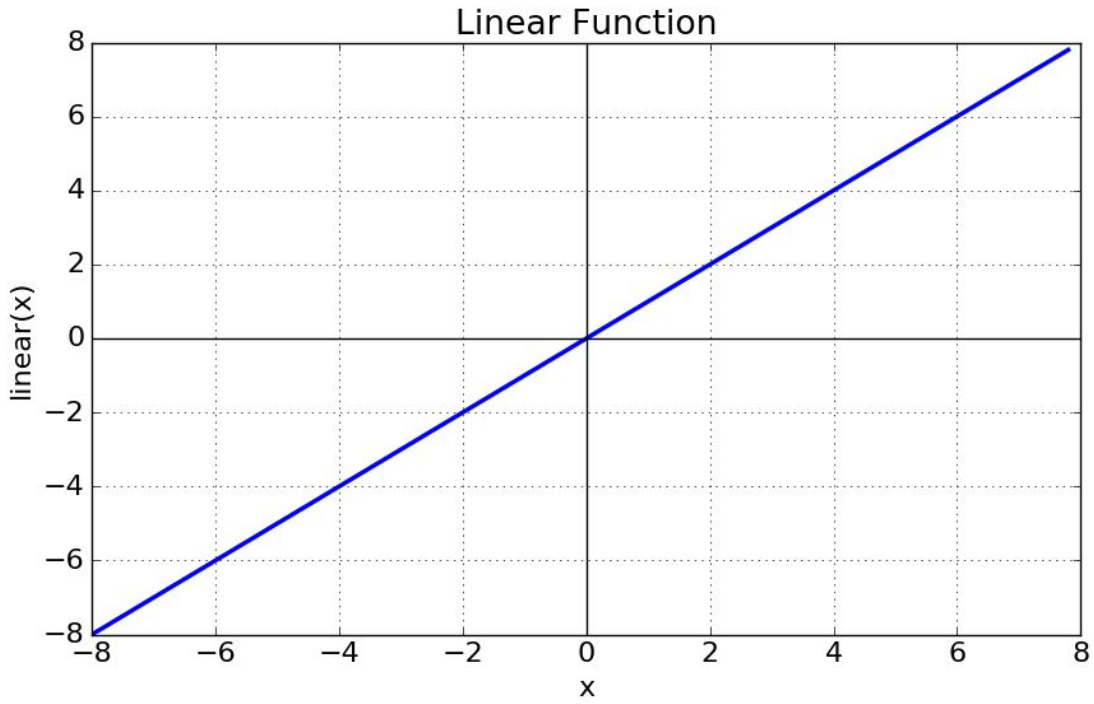
Layer 1-3 - function used -Relu

What is Relu (Rectified Linear Unit)?

Relu function is a linear function. Therefore, the output of the functions will not be confined between any range. We wanted the number of features to decrease uniformly ence we used RELU

Equation : f(x) = x

Range : (-infinity to infinity)

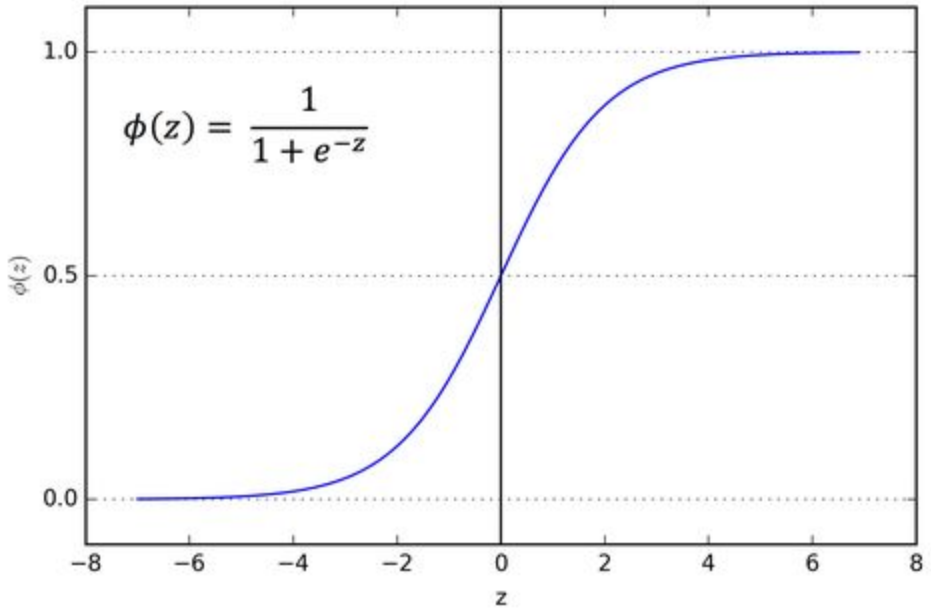



## Layer 4 - function used - Sigmoid

### What is Sigmoid & why sigmoid?

The Sigmoid Function curve looks like a S-shape.The main reason why we use sigmoid function is because it exists between (0 to 1). Therefore, it is

especially used for models where we have to predict the probability as an output.Since probability of anything exists only between the range of 0 and 1, sigmoid is the right choice.




4) Running it on training data which had 500 words successfully parsed.

While testing we got an accuracy of 79%.

| Number of layers | 3 | 4 | 5 |
|---|---|---|---|
| Percentage Accuracy | 67% | 73% | 79% |

# The Unicode Encoding (utf-8)

## What is Unicode?

Unicode is a computing industry standard for the consistent encoding, representation, and handling of text expressed in most of the world's writing systems. The Unicode Standard consists of a set of code charts for visual reference, an encoding method and set of standard character encodings**.**

## Unicode For HINDI:-

Devanagari is a Unicode block containing characters for writing languages such as Hindi, Marathi, Sindhi, Nepali, and Sanskrit, among others. It has its range from U+0900 to U+097F.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| U+090x | □ | ॑ | ॒ | ॓ | ऄ | अ | आ | इ | ई | उ | ऊ | ऋ | ऌ | ऍ | ऎ | ए |
| U+091x | ऐ | ऑ | ऒ | ओ | औ | क | ख | ग | घ | ङ | च | छ | ज | झ | ञ | ट |
| U+092x | ठ | ड | ढ | ण | त | थ | द | ध | न | ऩ | प | फ | ब | भ | म | य |
| U+093x | र | ऱ | ल | ळ | ऴ | व | श | ष | स | ह | □ | □ | ़ | ऽ | ा | ि |
| U+094x | ी | ु | ू | ृ | ॄ | ॅ | े | ै | ॉ | ॊ | ो | ौ | ् | □ | □ | |
| U+095x | ॐ | ॕ | ॖ | ॗ | क़ | ख़ | ग़ | ज़ | ड़ | ढ़ | फ़ | य़ | | | | |
| U+096x | ॠ | ॡ | ॢ | ॣ | । | ॥ | ० | १ | २ | ३ | ४ | ५ | ६ | ७ | ८ | ९ |

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| U+097x | ॰ | ॱ | ॲ | ॳ | ॴ | ॵ | ॶ | ॷ | ॸ | ॹ | ॺ | ॻ | ॼ | ॽ | ॾ | ॿ |

# Algorithm for stemmer:-

List of Hindi suffixes were collected from the following sources :

| | | | | |
|---|---|---|---|---|
| *A* | *AeM* | *awA* | *Ane* | *egA* |
| *i* | *AoM* | *awI* | *UMgA* | *egI* |
| *I* | *iyAM* | *IM* | *UMgI* | *AegA* |
| *u* | *iyoM* | *awIM* | *AUMgA* | *AegI* |
| *U* | *AiyAM* | *awe* | *AUMgI* | *AyA* |
| *e* | *AiyoM* | *AwA* | *eMge* | *Ae* |
| *o* | *AMh* | *AwI* | *eMgI* | *AI* |
| *eM* | *iyAMh* | *AwIM* | *AeMge* | *AIM* |
| *oM* | *AiyAMh* | *Awe* | *AeMgI* | *ie* |
| *AM* | *awAeM* | *anA* | *oge* | *Ao* |
| *uAM* | *awAoM* | *anI* | *ogI* | *Aie* |
| *ueM* | *anAeM* | *ane* | *Aoge* | *akara* |
| *uoM* | *anAoM* | *AnA* | *AogI* | *Akara* |

Pages in category "Hindi suffixes"

The following 45 pages are in this category, out of 45 total.

**आ**
- - आ
- - आऊ
- - आत्मक
- - आना
- - आवा

**इ**
- - इक
- - इका
- - इयत
- - इया

**ई**
- - ई
- - ईकरण
- - ईच
- - ईय
- - ईला

**ए**
- - ए
- - एद
- - एय

**क**
- - क
- - कर
- - कार
- - कारक

**ग**
- - गर्दी
- - गिरी

**ज**
- - जनक
- - जी

**त**
- - तः
- - ता
- - त्व

**द**
- - दान
- - दार

**न**
- - ना

- - नेर

**प**
- - पन

**ब**
- - बाज

**ल**
- - ल

**व**
- - व
- - वां
- - वाद
- - वादी
- - वाला
- वाले
- - वाले

**श**
- - शाली
- - शुदा

**स**
- - सा

These suffixes were stored into list based upon their lengths. Then the stemmer algorithm was designed so as to strip off the largest suffix present at the end of the given word. If none of the suffixes are present at the end of the word then the input word is returned as possibly it has no suffixes and therefore the word itself is its stem.

## Algorithm for lemmatizer:-

The algorithm to find the smallest root word of a given hindi word as input, makes use of the following things :

1) A root word database, which we have already extracted from the internet as explained on page number 15 in Data set section of this report.
2) A stemmer, which we have already implemented as shown in the above section, algorithm for stemmer.
3) Weights for consonant and vowels, which we derived by minimizing the difference between two word (The ‘difference’ of two words is defined later in this section)

The algorithm for finding smallest root word is based on following steps :

1) Stem the word by using the stemmer. Here we have assumed that a maximum of two suffixes can be present at the end of a word and hence the stemmer function might be called twice . This ensures smallest possible stem is returned.

2) Extract each word one by one from the root word database and for each word extracted and stem obtained, a value is assigned and stored in a list. The value is assigned in the following manner :Here the values list is the list which stores values corresponding to all the words present in the root word database.
3) The word in the root word database with the maximum value is the final lemma,i.e. Smallest root word of the input word.

The Vowel and consonant weights were estimated by using the following approach :

1) Made a dataset containing around 500 word, root_word pairs.
2) Stored sum of values of all pairs of words for a fixed consonant weight, say 1 and varying vowel weights.
3) The vowel weight corresponding to the maximum value sum gave the most optimal value of weight of vowel for a particular weight of consonant.
4) Accuracy of different models:-

| Root word extracted by removing | Only Suffixes | Only Prefixes | Both Suffixes and Prefixes |
|---|---|---|---|
| Percentage Accuracy | 94% | 87% | 78% |

**Classification into singular/Plural:-**

1) Clustering using K Means Algorithm.
2) Identifying different set of suffixes.
3) Using suffix matching clustering into singular and plural.

**What is KMeans clustering?**

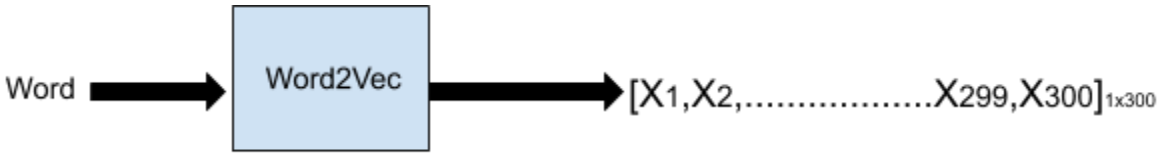
The algorithm partitions a set of N vector X = $x_j$ j=1 to N into C classes $v_i$, i =1 to c and finds a cluster centre for each class ci denotes the centroid of cluster vi such that an objective function of dissimilarity, for example a distance measure, is minimized. The objective function that should be minimized, when the Euclidean distance is selected as a dissimilarity measure, can be described as follows:-

$$P = \Sigma_{i=1}^{c}(\Sigma_{k,xk}|||xk - ci||)^2$$

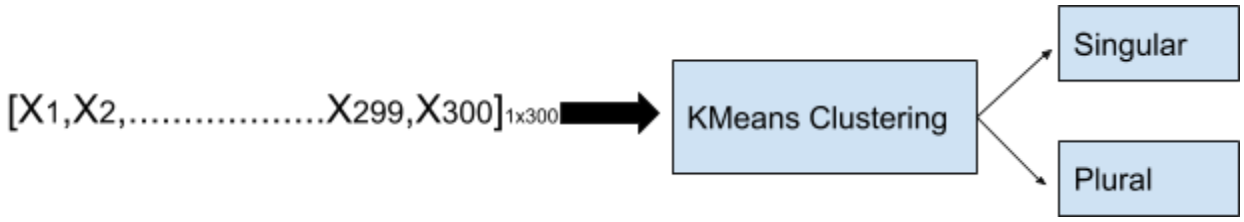
Where $||x_k - c_i||$ $||^2$ the objective is function within group i and $||x_k - c_i||^2$ is a chosen distance measure between a data point $x_k$ and the cluster centre $c_i$.

**K Means Algorithm Approach:-**

1) Reading word from dataset, converting them to their word vectors using Word2Vec approach. The vector obtained were of [1X300] dimension each.



2)KMeans algorithm Clustered it in two groups.




3) Different distances used in clustering:-

**1) Euclidean Distance-** The Euclidean distance between two data points involves computing the square root of the sum of the squares of the differences between corresponding values. It represents the Euclidean distance when q=2.

$$d = (i, j) = \sqrt{\Sigma_{k=1}^{n}[X_{ik} - X_{jk}]^2}$$

**2) Cosine Distance-** The Cosine distance between u and v, is defined as

$$d = 1 - \cos(\theta)$$

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

**3) Manhattan Distance-** It is also known as City block distance, and absolute value distance or L1 distance. Manhattan distances a distance

that follows a route along the non-hypotenuse sides of a triangle. The name refers to the grid-like layout of most American cities which makes it impossible to go directly between two points. :

$$d(i,j) = \Sigma_{k=1}^{n} |X_{ik} - X_{jk}|$$

| Different distances used | Euclidean Distance | Cosine Distance | Manhattan Distance |
|---|---|---|---|
| Percentage accuracy | 74% | 67% | 84% |

4) Using suffix matching classifying into singular and plural. We got accuracy of **94%** while we tested of suffix matching approach using unicode encoding.

## Similar Approach was used for classification of Masculine/ Feminine.

The accuracy for Masculine/Feminine clustering was

| Different distances used | Euclidean Distance | Cosine Distance | Manhattan Distance |
|---|---|---|---|
| Percentage accuracy | 60% | 61% | 76% |

4) Using suffix matching classifying into masculine/feminine. We got accuracy of **89%** while we tested of suffix matching approach using unicode encoding.

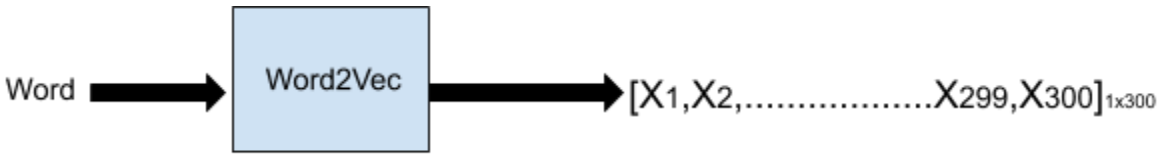
# SVM classifier:-

What is SVM classifier?

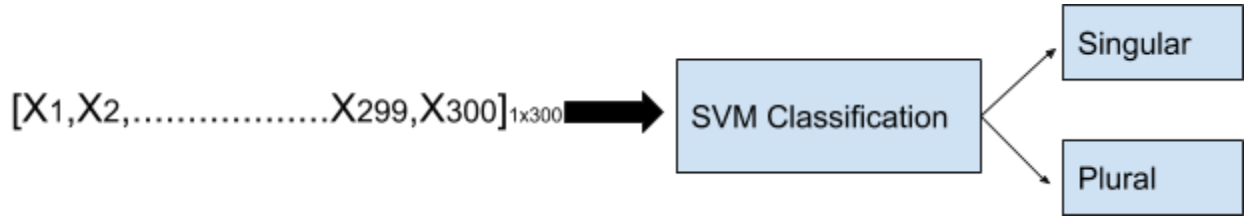
In machine learning, support-vector machines (also support-vector networks) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis.

SVM classifier Algorithm:-

1) Reading word from dataset, converting them to their word vectors using Word2Vec approach. The vector obtained were of [1X300] dimension each.




2)SVM algorithm Clustered it in two groups.




The training dataset was of 1000 words

This approach gave us a accuracy of 91% for singular/plural classification, and 81% for masculine/feminine classification.

## Part Of Speech Tagging:-

Morphological analysis of a word also involves tagging word with its possible parts of speech.

Previous works in POS taggings are :

1) Manish and Pushpak researched on Hindi POS using a simple HMM-based POS tagger .
2) Nisheeth Joshi, Hemant Darbari and Iti Mathur also researched on Hindi POS using Hidden Markov Model with the frequency count of two tags seen together in the corpus divided by the frequency count of the previous tag seen independently in the corpus.

In our Project we made use of TnT tagger which is present in the nltk library of python. It is trainable on different languages and virtually any tagset.For training the tagger we needed a tagged training data of hindi language. We used the already tagged data which is present in 'indian' module in nltk.corpus .The algorithm used for part of speech tagging gave us an accuracy of 52%, We want to keep this part of our project in future works, because this required a labelled dataset of hindi words.

# 4.Results

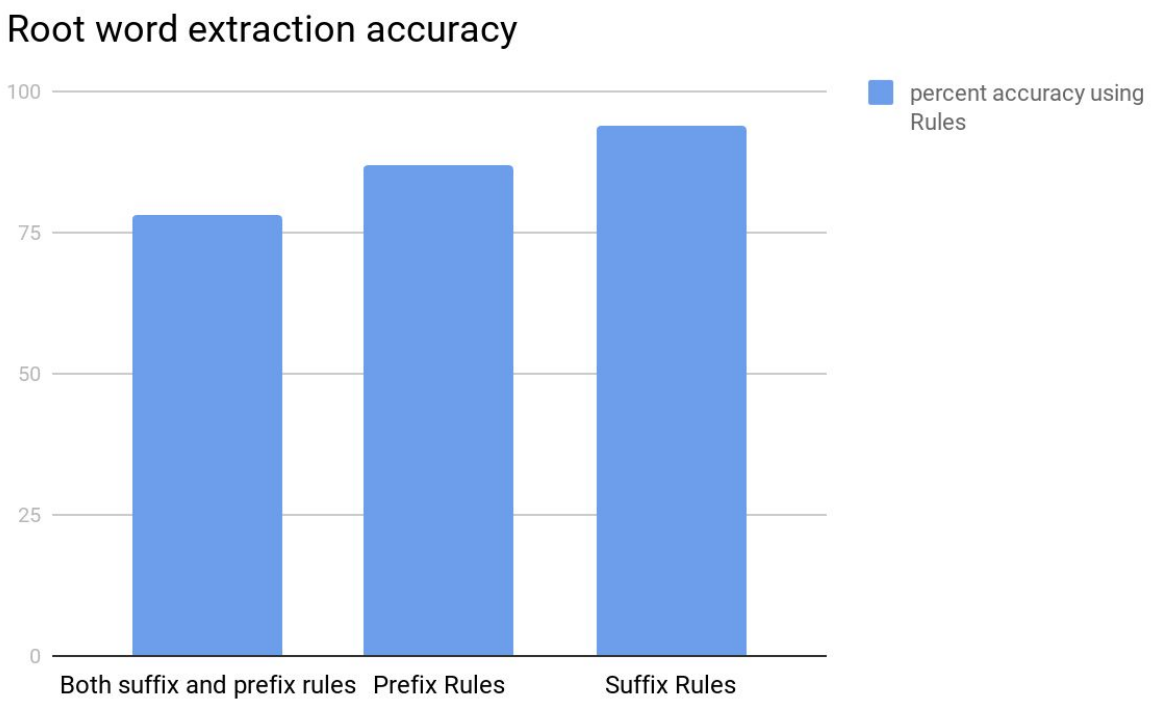
## Extraction of Root word:-

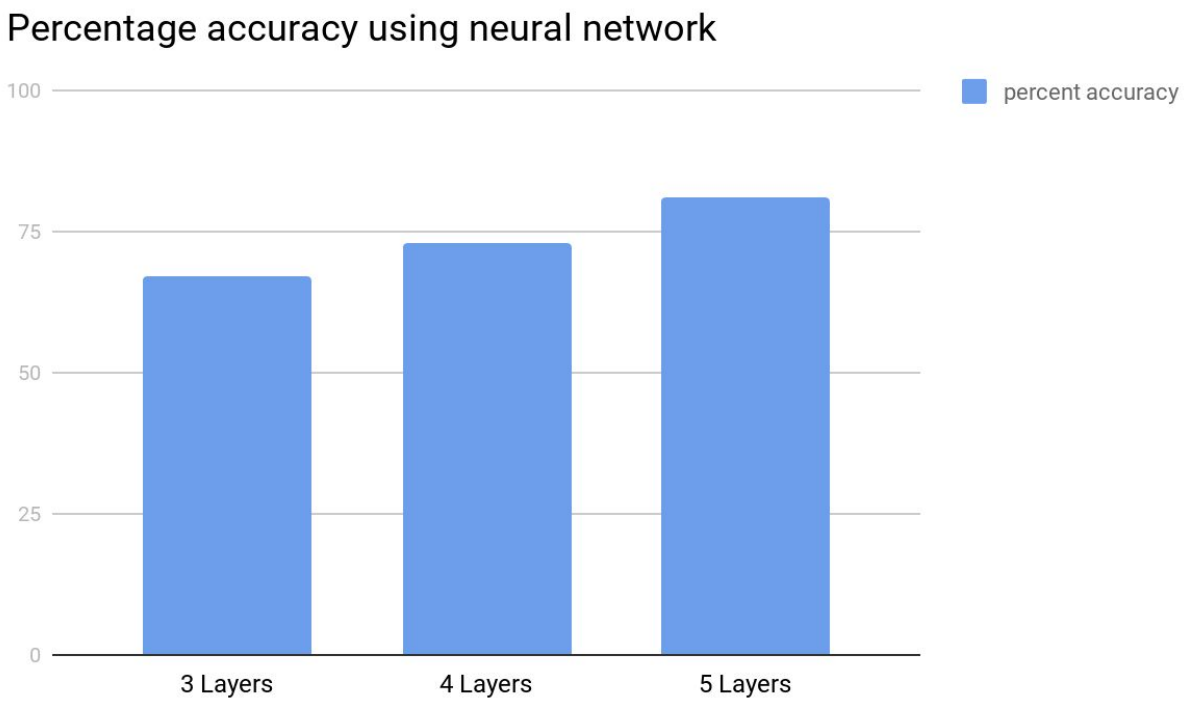
The algorithm we used to extract root word gave us an accuracy of :-

78% when both suffix and prefix rules were used

87% when only prefixes rules were used

94%  when only suffix rules were used






**The reason for this trend was- Hindi being a morphologically rich language 95% of the root words can be extracted by just applying suffix addition rules.**

Eg- words like अतिसुन्दर have a prefix अति which when stripped out gives a valid word सुन्दर, but when words like अतिथियों which also have prefix अति but when we strip off the left out word does not make sense थियों.


Here is the screenshot of the root extraction algorithm. It can be seen that the word no. 25 is lemmatized wrong. The reason being that the actual root word लिख is not present in the root word database.

# Classification into (singular,plural) & (Masculine,Feminine):-

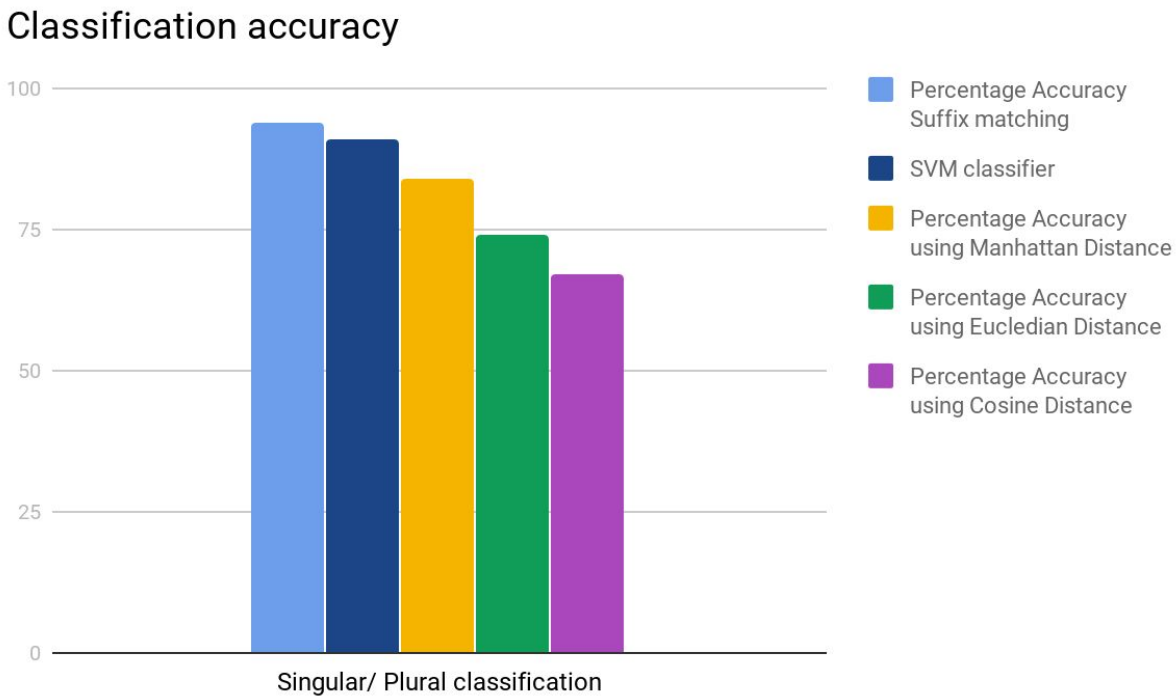
The algorithm we used to classify words into (singular and plural) & (masculine and feminine)  gave us an accuracy of :-

Suffix Matching:-

94% in the case of Singular/Plural classification

86% in the case of Masculine / Feminine.

**The reason for this trend was- Hindi being a morphologically rich language almost all the words can be classified into singular and plural but not all words can be classified into masculine and feminine**




Eg- गलतियों being a plural word can not be distinguished into masculine neither feminine.

Similarly शरीर can not be classified into masculine/feminine.

```
1  कुत्ते Plural
2  बन्दर Singular
3  पुलिसवाले Plural
4  आदमी Singular
5  महिलायें Plural
6  आम Singular
7  शरीर Singular
8  पानी Singular
9  लड़का Singular
10 लड़कों Plural
```

Here is the result of singular/plural classification algorithm

# Part of speech tagging:-

The algorithm used for part of speech tagging gave us an accuracy of 52%,

We want to keep this part of our project in future works, because this required a labelled dataset of hindi words.

# 5.Conclusion and Future Prospects

What initially started out as a identifying morphemes on a particular type of problem eventually caught our interests due to the peculiar results and findings. This motivated us to scratch the surface harder and come up with the approach of developing a slight trade-of between accuracy and efficiency by using existing word vectors. This approach negligibly affects the run-time, but does wonders to the accuracy

. Previous State of the Art(Derivational and Inflectional morphological analyzers) are also outperformed by this method. This is due to the use of unicode encoding model which essentially follows from intuition.

Designing the Word2Vec from scratch using the features which would actually mean and give a significantly good output.

This can be attributed to the future of this project.

# 6.References


1) Wiktionary for list of lemmas- https://en.wiktionary.org/wiki/Category:Hindi_lemmas
2) Linguistica (python library)- https://pypi.org/project/linguistica/
3) G. M. Singh, L. G. Singh, S.S. Joshi , “A full form lexicon based Morphological Analysis and generation tool for Punjabi”,International Journal of Cybernatics and Informatics, Hyderabad,India,October 2007
4) V. Goyal, G. S. Lehal, “Hindi Morphological Analyzer and Generator”, First International Conference on Emerging Trends in Engineering and Technology, USA, 2008.
5) A. Agarwal, Pramila, S. P. Singh, A. Kumar, H. Darbari, “Morphological Analyser for Hindi – A Rule Based Implementation” International Journal of Advanced Computer Research (ISSN (print) March-2014
6) N. Kanuparthi, A. Inumella, and D. M. Sharma, “Hindi derivational morphological analyzer”, Association for Computational Linguistics 2012.
7) KVS Nikhil, (2012), “Hindi Derivational Morphological Analyzer”, IIIT Hyderabad, Hyderabad - 500 032, INDIA November 2012

# Thank You