SEMI-SUPERVISED APPROACH FOR HINDI STEMMING

Amit Anand, Sanjay Chatterji and Shaubhik Bhattacharya

IIIT Kalyani West Bengal, India 741235

ABSTRACT

Stemming is one of the most fundamental requirement of any Natural Language Processing tasks such as Information Retrieval. In simple words, it is the process of finding stem of a given word. This paper presents an algorithm to find the stem of a word in Hindi. The proposed algorithm uses word2vec, which is a semisupervised learning algorithm, for finding the 10 most similar words from a corpus. Then a mathematical function is proposed to achieve the above mentioned task of finding stem. Significant amount of attention need to be given to Indo-Aryan languages like Hindi, Bengali, Marathi etc. in the domain of Natural Language Processing because of their highly inflectional properties. Moreover,it is very difficult to build a rule based stemmer for such highly conflated languages. The proposed algorithm does not need any annotated corpus and does not use any hardcoded rules for finding the stem. The results are verified by selecting a set of 1000 Hindi words randomly taken from a corpus and comparing the results given by the proposed algorithm and the actual results created manually.

KEYWORDS

Inflection, Stemming, Word2Vec, Unsupervised Machine Learning

1. INTRODUCTION

Hindi comes in the fifth position in the list of most spoken languages in the world according to Ethnologue. Though it is one of the most spoken languages in the world, the linguistic resources are scarce for Hindi language. Hindi language is prominent in the list of highly inflectional languages. The number of morphological variants for a given word in Hindi can be in the order of 1000 depending on gender, number, person, tense, aspect, modality, parts-of-speech, case, etc. In Natural Language Processing (NLP) tasks such as Information retrieval, stemming is one of the fundamental tasks in preprocessing. Today most of the stemming tools for resource-poor languages are rule-based. The rule based approach is not a practical approach for regional languages like Hindi since the formation of Hindi word does not always follow it. Due to inflectional nature it is also very difficult to exhaustively define the set of rules. The words formed from the same root word are called morphological variants of the words. For example: In English language watches, watching, watched are morphological variants of word watch.

In Hindi words like: जाएँ (Jaen) [Go to], जाएं गे (jaenge) [Will go], जाय" (jaayen) [Go to], जाएँ (jaen) [Go to], जाओगे (jaoge) [Will go], जाते (jaate) [Goes], जात" (jaaten) [Go to], are different morphological variants of the word जा (ja) [Go]. The aim of this paper is to find the root word from different morphological variants of Hindi words.

In this paper, we wish to propose a novel stemming algorithm based on the context and three structural features. The context is used to form the groups of words using a semi-supervised

Natarajan Meghanathan et al. (Eds) : NLP, ARIA, JSE, DMS, ITCS - 2019
pp. 35-42, 2019. © CS & IT-CSCP 2019 DOI: 10.5121/csit.2019.91204

learning approach. This narrows down the search of root word in all the clusters to one cluster. Further, we wish to introduce a mathematical function considering the three structural features to find out the root word from each group. The features will be chosen on the basis of Size, Match and Mismatch between the group of words and input word. The three features completely exploit the structural property of the given word and we will prove that no more feature is needed. The advantages of this algorithm is that it is computationally inexpensive and can be used for different languages including low resource languages. It is helpful for other languages because of its corpus-based approach.

2. LITERATURE SURVEY

In NLP, the root words of any given language is identified by its stem. So, stemming is the first step in any NLP task. Some decades ago, people who had sound knowledge in a particular language were involved with developing rule based tools of that language. This is a tedious and rigid process. In other words, the rules formed for one language cannot be extended for the other languages. In practical scenario, a rule based stemmer can be easily made for less inflectional languages like English. But for languages that are highly inflectional like Hindi, Bengali, Nepali, etc. it is almost impossible to create a perfect rule based stemmer.

In early days, most of the stemming algorithms were based on suffix stripping and it is basically on English and some European languages. A list of appropriate suffixes were extracted by analyzing the linguistic resources of that language. Some of the famous suffix stripping Stemming algorithms of English language are Lovin's [1], Porter Stemmer [2], Paice Stemmer [3] and for Hindi language is MAULIK [4].

As discussed in [5] Hindi comes under the category of a relatively free word order language as compared to English which is a well structured language. In Hindi the relation between words is expressed by using postpositions and appropriate inflecting nouns and inflecting verbs which is used to express case information and to reflect gender, number and person information respectively. They had also categorized the inflections in Hindi language into various classes such as Noun Inflections, Verb Inflections and Adjective Inflections.

People were successful in building and showing that probabilistic model [6] and statistical model [7] are as effective as linguistic knowledge. N-Gram Stemmer [8] and YASS [9] are two famous stemmers which uses statistical methods for finding stem. They are also corpus based stemmers. The main idea behind the N-Gram Stemmer is based on the fact that similar words have a high proportion of common n-grams. The YASS (Yet Another Suffix Stripper) also does not rely on linguistic expertise. They have experimentally shown that the stemmer is more effective than the suffix stripping for languages like: English, Hindi and Bengali.

3. PROPOSED METHOD

Our proposed algorithm comprises of two steps. In the first step we are using word2vec for finding similarity between the words in a corpus. It is used for finding the Ten Most Similar Words (TMSW). In the second step we are defining a new cost function by taking three structural features.

Word2vec returns a set of related models which are used to create word embeddings. Word embedding is basically a vector of features that define a word uniquely. The word embedding so formed depends on the words surrounding the given word. Using the cosine similarity the similar words from the corpus is collected. Using a corpus of English language, we found the 10 topmost

similar words of tries are: trys, try, attempts, trying, scrambles, attempting, tried, seeks, wants and attempts.

Likewise using a Hindi corpus we have developed a word2vec model for Hindi language. Using this model we found TMSW for a given Hindi input word. We can say that these words are contextually and semantically similar to the given input word. [9] had briefly explained that their approach can return some cluster of words which are semantically different. So using word embeddings obtained from word2vec, we can overcome this problem to some extent.

Then, in the second step, to extract root word from a cluster of words, we defined a cost function in such a way that it will give minimum cost for the word which is the stem of the input word. We have defined the cost function by taking into account three structural features as follows.

Feature 1: $Size_i$ is calculated based on the difference of size between the input word and the i^{th} word from the TMSW.

Feature 2: $Matches_i$ is calculated based on the number of positions where the alphabets of the input word and the i^{th} word from TMSW are matched.

Feature 3: $Mismatches_i$ is calculated based on the number of positions where the alphabets of the input word and the i^{th} word from TMSW are mismatched.

We will calculate all the above mentioned features separately and then combine them in a single mathematical function to incorporate the effect of all the three features. Let us calculate these three features first. For that let us consider the input word size is m and the word size of the i^{th} word from TMSW is n.

i. Calculation of $Size_i$: We want to penalize the stem words if it is of large size. This is to assure that the root word will always be of smaller size than the input word. Also to the word which has higher size difference with the input word we wish to assign them exponentially higher penalty. Mathematically, we have defined it as shown below.

$$Size_i = 2^{n-m} \quad (1)$$

ii. Calculation of $Matches_i$: For finding the reward for matches, we start matching the characters of both the input word and the words in TMSW. We stop at the position where the words' character are first mismatched. The percentage of characters matches from first character is defined as 't'.

Consider the English input word 'attacked' and the couple of words attacker and attack in the corresponding TMSW. The value of t is 7 for attacker and is 6 for attack. It is to be noted that the inflected word from TMSW generally has more matches in comparison to the root word. Therefore we consider negative of these matches as penalty. Similar to the size penalty, we consider exponential penalty for the number of matches also.

But what happens to the words which have no character match- ing with the input word or very less number of characters are matching? We should give higher penalty to them. We have taken 1000 inflections of different Hindi root words. Then we have calculated the value of t is average 60%, minimum 0% (in case of root ya and inflected word gayaa) and maximum 94% (in case of root Kinkartavyavimudha and inflected word Kinkartavyavimudhon). See the plot of number of words versus the value of t in Fig 1.

If we take the minimum case then we will be able to consider all the words. But then we will also consider many unrelated words. Therefore, we consider the words which have minimum 40% matches.

And then if the word has higher number of matches then exponentially we have assigned higher penalty. Here, it is to be noted that for the word which has t value = 65 will have high penalty. But as there is less chance of occurring a word who has t value between 40 and 65 in TMSW and as we have not considered the t value between 0 and 40 therefore no small words will be there with less penalty. However, if there is a word who has t value between 40 and 65 in TMSW then that word has higher chance of becoming the root which also seems practical. Mathematically, we have defined it as shown below.

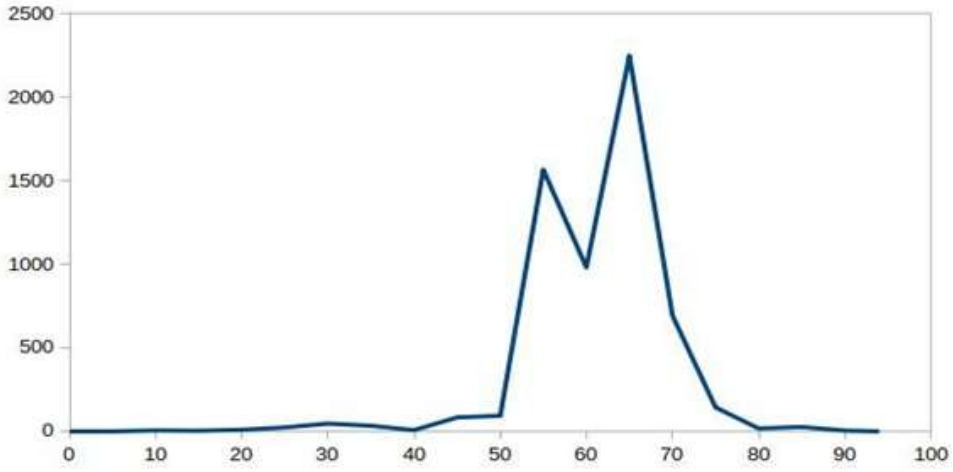

Figure 1: Number of Words Vs Percentage of Matches (t)

$$Matches_i = \sum_{j=0}^{t} 2^{-j} \qquad (2)$$

iii. Calculation of $Mismatches_i$: This cost is defined by taking into consideration the extent of mismatch of input word and the words in TMSW. Starting from 1^{st} mismatched character till the minimum length of input word and the TMSW words is considered. Here, as the length of input word is m and length of a contextually similar word is n and if the mismatch starts at q^{th} position then the cost for mismatch is mathematically defined as follows.

$$Mismatches_i = 2^{min(m,n)-q} \qquad (3)$$

There are several methods of calculating the similarity and difference of two words. Consider the following four basic distance measurement techniques.

- Euclidean Distance
- Hamming Distance
- Cosine Similarity
- Jaccard Similarity

In Euclidean and Hamming Distance we consider the place of mismatch and do some processing (insertion, deletion or substitution) on the mismatched characters. The number of processing required depends on the size difference between the words, number of matches and number of mismatches. We are not considering Cosine and Jaccard Similarity while calculating the cost of the input word and TMSW words. This is because, these corpus based equations are already been taken care by the word2vec while selecting the TMSW words. So it is obvious that it is adequate to consider these three features and the corresponding processing required to calculate the cost of distance between two independent words.

4. EXPERIMENTS, RESULTS AND DISCUSSIONS

As told before, there may be some processing requires for each of the three features. Some feature may need more processing than the other.

After finding the three different costs we have done some experiment to find the best way to combine them. The combining should be done in such a way that it will assign minimum value for the word which is the actual root. Following are the three possible ways we tried to combine to calculate the Total Cost.

i) First Case: Addition of the three different costs
$TotalCost = Size_i + Matches_i + Mismatches_i$

ii) Second Case: Multiplication of the three different costs
$TotalCost = Size_i \times Matches_i \times Mismatches_i$

iii) Third Case: Weighted addition of the three different costs
$TotalCost = w1 \times Size_i + w2 \times Matches_i + w3 \times Mismatches_i$

Let us consider that there is a root word in the TMSW. Our target is to minimize the Total Cost for root word and maximize the Total Cost for the non-root words. The first one is same as the third one with weights being assigned to 1. But, the weights may not be 1. One may use a training corpus to learn the weight values. But as we do not want to use use any training corpus as this is a unsupervised technique.

Therefore, in these three different cases of Total Cost the second one fulfills the requirement i.e- by multiplying three costs. In Third case we do not have to assign any weight because the effects of the functions for $Size_i$, $Matches_i$, and $Mismatches_i$ will be same.

For a given root word we store the TMSW words and their corresponding Total Costs. After finding the Total Cost the row corresponding to the Minimum Total Cost (MTC) will be selected provided that the second cost (M) of that row is greater then 1.5. If the second cost ($Matches_i$) of the row of MTC is less than or equals to one and half then the input word is itself the root word.

We had also done experiment in order to find the optimal number of size of cluster formed. In Table 1 we had shown that number of correct root words by taking five to fifteen most similar words. We had experimentally observed that by forming a cluster of 10 or more words, the stemmer gives highest accuracy. Thousand unique words were taken for calculation of accuracy.

Table 1: size of cluster vs accuracy of stemmer

List Size of SW	Correct Prediction
5	51.5
6	69.5
7	77.0
8	83.5
9	85.5
10	95.3
11	95.3
12	95.3
13	95.3
14	95.3

We will first explain the proposed algorithms using an English word and then using a Hindi word. Experiment with English word: In previous section we have talked about the Ten Most Similar Word (TMSW) for English word - tries. Now we will take one by one the TMSW words and then find their Total Cost. Below is an example which shows how to calculate the costs.

Input Word: *tries*
First similar word: *trys*

i) First Cost($Size_i$): Using equation 1

$$Size_1 = 2^{n-m}$$

Here *m* is number of character in Input Word i.e-tries and *n* is number of character in Similar Word i.e-trys
so, $m = 5$ and $n = 4$

$$Size_1 = 0.5$$

Similarly we will find the first cost of all 10 similar words. In 3^{rd} Column of Table 2 we had shown First cost.

ii) Second Cost**(**$Matches_i$**):** Using equation 2

$$Matches_1 = \sum_{j=0}^{t} 2^{-j}$$

Here the first and second characters are matched so, $t = 2$.

Matches1 = 1.75

This is mentioned in the 4th column of Table 2. This value is greater than 1.5.

iii) **Third Cost(*Mismatches*$_i$):** Using equation 3

Mismatches$_1$ = 2^x

Here mismatch starts after 2^{nd} character and length of input word is 5. Therefore $min(m, n) - q = min(5, 4) - 2 = 2$ Hence $Mismatches_1 = 4$

Mismatches$_1$ = 8

It is mentioned in the 5^{th} column of Table 2. The Total Cost is men- tioned in the 6^{th} Column of Table 2 .

Table 2: Cost Calculation for English word

Input Words	Similar Words	First Cost	Second Cost	Third Cost	Total Cost
tries	trys	0.5	1.75	4	3.5
tries	try	0.25	1.75	2	0.875
tries	attempts	8	1	32	256
tries	trying	16	1.75	8	28
tries	scrambles	2	1	32	64
tries	attempting	32	1	32	1024
tries	tried	1	1.9375	2	3.875
tries	seeks	1	1	32	32
tries	wants	1	1	32	32
tries	attempts	4	1	32	128

Experiment With Hindi Word: We had shown above with the help of an example the working of our stemmer for English language similarly we will show for Hindi language . As we had discussed Hindi Language is very high inflectional. In [4] they had categorized the different inflections like: Noun Inflection, Adjective Inflection and Verb Inflection but in our proposed algorithm there is no need for that.

Input Word : जाएँ गे

The cost with all similar words are in Table 3

Table 3: Cost Calculation for a Hindi Word

Input Words	Similar Words	First Cost	Second Cost	Third Cost	Total Cost
जाएँगे	जायेंगे	2	1.75	16	56.0
जाएँगे	जाएंगे	1	1.875	8	15.0
जाएँगे	जायें	0.5	1.75	8	7.0
जाएंगे	जाएँ	0.25	1.875	2	0.9375
जाएँगे	जाएं	0.25	1.875	2	0.9375
जाएँगे	जा	0.125	1.75	1	0.21875
जाएँगे	जायँ	0.25	1.75	4	1.75
जाएँगे	जाओगे	0.5	1.75	8	7.0
जाएँगे	जाते	0.25	1.75	4	1.75
जाएँगे	जातें	0.5	1.75	8	7.0

5. CONCLUSION AND FUTURE WORK

In India there are 179 regional languages with their own grammar. Due to such variety in regional languages and their inflectional properties it is practically impossible to build the rule based stemmer for every languages. Taking these factors into consideration we tried to build a stemmer which is independent of the grammar of the given language. The proposed algorithm would have been more accurate if corpus become more extensive, both in size and richness. It can also be tested for other languages specially for English to compare its performance with the standard stemmers.

REFERENCES

[1] J. B. Lovins, "Development of a stemming algorithm," Mech. Translat. & Comp. Linguistics, vol. 11, no. 1-2, pp. 22–31, 1968. [Online]. Available: http://www.mt-archive.info/ MT-1968-Lovins.pdf

[2] M. Porter, "An algorithm for suffix stripping," Program, vol. 40, no. 3, pp. 211–218, 2006. [Online]. Available: https://doi.org/10. 1108/00330330610681286

[3] C. D. Paice, "Another stemmer." SIGIR Forum, vol. 24, pp. 56–61, 11 1990.

[4] U. Mishra and C. Prakash, "Maulik: An effective stemmer for hindi language."

[5] A. Ramanathan and D. Rao, "A lightweight stemmer for hindi," 2003.

[6] M. Bacchin, N. Ferro, and M. Melucci, "A probabilistic model for stemmer generation," Inf. Process. Manage., vol. 41, no. 1, pp. 121–137, Jan. 2005. [Online]. Available: http://dx.doi.org/10. 1016/j.ipm.2004.04.006

[7] D. Sharma, "Article: Stemming algorithms: A comparative study and their analysis," International Journal of Applied Informa- tion Systems, vol. 4, no. 3, pp. 7–12, September 2012, published by Foundation of Computer Science, New York, USA.

[8] J. Mayfield and P. Mcnamee, "Single n-gram stemming," 01 2003, pp. 415–416.

[9] P. Majumder, M. Mitra, S. K. Parui, G. Kole, P. Mitra, and K. Datta, "Yass: Yet another suffix stripper," ACM Trans. Inf. Syst., vol. 25, p. 18, 2007.