International Journal of Computing and Applications | **ISSN: 0973-5704**
Volume 13, Number 1, (January-June 2018), pp 39-44
© Serials Publications, New Delhi (India)

A HYBRID APPROACH FOR LEMMATIZER FOR HINDI LANGUAGE

Suman Chaudhary[1] and Kunal Chakma[2]
[1-2]Computer Science & Engineering Department, NIT Agartala, Tripura, India.
Email: [1]suman.the.sageh@gmail.com, [2]kchakma@nita.ac.in

Abstract: Hindi is highly inflectional language. This makes the lemmatization process more complex and challenging. In this paper we proposed a different approach for developing lemmatizer for Hindi language in Devanagari script. We will break the inflected word into smallest units and then tried to form the root words from these units. Using this hybrid approach for lemmatization, a system can also able to handle the inflected words with suffixes which are not known. Hence increasing the efficiency and accuracy of the lemmatizer.

Keywords: Lemmatization, morphological analysis, Hindi, lemmatizer, hybrid, inflected.

1. INTRODUCTION

Natural language processing, as the name reflects, is processing of those language which are naturally evolved in humans without any practice. It is a field that comes under combination of computer science, artificial intelligence, and computational linguistics. It is concerned with the interactions between computers and human (natural) languages.

For development of any natural language processing system, first and foremost step is Morphological analysis. The aim of this phase of language processing is to separate strings of language input into tokens sets corresponding to discrete words, sub-words and punctuation forms. For example a word like *"uncomputable"* can be broken into three sub-word tokens as:

un – compute - able.

Morphology could be either derivational or inflectional. Derivational morphology processes the words and form new lexemes from the existing ones. For example, in गर्म + ई = गर्मी(Summer), the word गर्म(hot) with a suffix ई has changed from adjective to noun गर्मी(summer). Inflectional morphology processes the words by producing various inflections without changing the word class. For example, करिण + ें = करिणें where करिण(a ray) is noun/singular while करिणें (rays) is noun/plural. In this case, the class remains same.

In NLP, stemming and lemmatization are used to get the root form of a word. Though stemming is an important and useful NLP task, it always does not produce the correct root words for a morphologically rich language like Hindi when written in the Devanagari script. *Stemming* usually refers to a heuristic process that chops off the ends of words and often includes the removal of derivational affixes. For example if following inputs will be given to stemmer the output will be the corresponding words

Input → Output of stemmer

is → is

the going → the go

coming → com

On the other side, *Lemmatization* usually refers to obtaining the root word properly with the use of a vocabulary and morphological analysis of words and to return the base or dictionary form of a word, which is known as the lemma. For example the inputs previously given to stemmer, will given to lemmatizer the output will be following

Input → Output of Lemmatizer

is → be

the going → the go

coming → come

The main disadvantage of stemmers is that they do not give legitimate lemma in the result. Therefore, lemmatizers are more appropriate than stemmers for document processing tasks such as retrieval of documents. Hence we mostly prefer lemmatizer over stemmer.

Our main objective is to develop such approach that will give a conceptual root form of the corresponding word. This includes all the inflected words of th root and derivational roots. Example

Inflected word → Root word

शिषताएँ (Speciality) → वशिष (Special)

नकली (Fake) → नकल (copy)

रचनाएँ (Composition) → रचना (Creation)

2. RELATED WORK

Many Stemmers and lemmatizers have already been developed especially for English and European languages[9]. Huge improvemens have been achieved in last few years. A well known algorithm for stemming given by Martin Porter [10] is the foundation algorithm. The basic rule based approach for lemmatization for English language proposed by Plisson[2] is based on the word endings where the suffix should be added or removed based on word to get the normalized form. It mainly emphasizes on two word lemmatization algorithm which is based on simple if-then rules and the ripple down approach. A context sensitive lemmatizer for German language was developed by Wolfgang Lezius, Reinhard Rapp and Manfred Wettler[15]. Authors created an integrated tool for German Morphology which involved POS taggers and lemmatizer. In the first step, the morphology module delivers all possible lemma corresponding to each word form. Secondly, the tagger determines the grammatical categories of the word forms. If, for any of the lemma, the inflected form corresponding to the word form in the text does not agree with this grammatical category, the respective lemma is discarded. Many other lemmatizer for languages like English, German, Arabic, Spanish etc are available. In Stanford core NLP tool [16], fully developed lemmatizer for English, Spanish, Arabic, Chinese, French and German languages are available with advance versions.

On the other hand, not much work has been done regarding the development of lemmatizer for South Asian languages. Nearly no reliable lemmatizers available for Indian languages, particularly for Hindi language as it does for national languages like English language or other European languages. Hindi is fourth most spoken language after std. M. Chinese, Spanish and English. A rule based approach is used by Snighdha Paul, Nisheeth Joshi and Iti Mathur[6] in the development of Hindi Lemmatizer. Authors concentrated on time optimization by creating a database consisting of commonly used Hindi words. Regarding Hindi Stemmer, there are number of approaches used by authors. A rule based approach was proposed by Ramanathan & Rao [1] for stemming in Hindi. This proposed stemmer is both computationally inexpensive and domain independent. The approach is based on stripping off suffixes by generating rules concentrating on noun, adjective and verb inflections in Hindi language. Another stemmer named as "Maulik" is proposed by Upendra Mishra and Chandra Prakash[13] which is based on combination of brute force and suffix removal approaches. The proposed stemmer is light weight stemmer which is domain independent and computationally inexpensive. It also reduces the problem of under-stemming and over-stemming.

There are another tools also proposed for the morphological analysis of Hindi language. Nikhil K VS[4] built a Hindi derivational analyzer by creating a SVM classifier which he used to identify the derivational variants. A different approach is used by Shashi Pal Singh, Ajay kumar, Dr. Hemant Darbari and Anshika Gupta[5]. They build a Rule based tense synthesizer for Hindi which derived translated output in correct tense using HMM tagging. A morphological analyzer for Oriya language is proposed by TJena et. al., [7] using the paradigm approach. Using various paradigm tables, they classified nouns, adjectives and finite verbs of Oriya language. The approach proposed by Goyal et. al., [3] concentrates on the creation of a morphological analyzer and generator that translate from Hindi to Punjabi. Their main objective was to develop a translation system especially from Hindi to Punjabi.

3. CORPUS

Standard data set is taken which is available as Hindi Corpora at Center for Indian Language Technology. 9000 unique words were extracted from 20000 sentences and then cleaning process was applied on the corpus collected so far. Cleaning process is a process of manually removal of words from the dataset which are not required for the analysis and development of the project. Following types of words were removed in cleaning process

1. Root words with same inflections
2. Stop words
3. Words already in their dictionary form.
4. Words in Unstructured form like "समापनकरजारहे" (are going after concluding). This type of words are manually corrected and then stored in the list. For example correct form the given words is समापन कर जा रहे (are going after end of function). Thus we got four different words that are समापन (concluding), कर (after), जा (going) and रहे. These words were stored separately.

Above types of words were removed from the dataset. The number of words decreased to 5000 after cleaning process. Out of them, 3500 words are in their unique inflected form We break the inflected words into their root form, prefixes and suffixes as shown in Table 1.

Table 1
Structure of Data set

शब्द (inflected word)	*मूल शब्द (root word)*	*उपसर्ग (prefix)*	*प्रत्यय (suffix)*
पाठक (reader)	पाठ (lesson)		अक
सफल (successful)	फल (result)	स	
अंग्रेजी (English)	अंग्रेज (English men)		ई
समझदार (understanding)	समझ (understand)		दार
वेदों (Vedas)	वेद (Veda)		ओं

4. PROPOSED WORK

Hindi language is highly inflectional language and hence it is always difficult to develop a NLP tool for these types of languages. In this paper a hybrid approach is proposed using which will increase the accuracy of the lemmatizer for Hindi language in Devanagari script. From the dataset, we have observed that 85.7% of the inflected words have suffixes and 19.04% of words have prefixes and only 6% of the words have both suffixes and prefixes.

Therefore, on the basis of above observations from dataset a hybrid approach which is combination of dictionary based and rule based approaches is suggested for the lemmatization of Hindi language. Proposed approach is present in form of following Algorithm 0. Algorithm 0 is constitute of further 2 algorithms, Algorithm1 corresponding to dictionary based approach and Algorithm2 corresponding to rule based approach.

Algorithm 0 (string input word, string output word)

Step 1: Break the input inflected word into corresponding uni-codes (smallest units) and save it in inflected word[ ].

Step 2: Execute Algorithm1 with inflected word [ ] as its input.

Step 3: If Algorithm1 gives output, then restart from step 1 of this algorithm with next input word.

Step 4: Else execute Algorithm2 with inflected word [ ] as its input.

Step 5: Restart from step 1 with next input word.

Algorithm 1 (inflected word [char], string output word)

Step 1: Merge two sequential unicodes from inflected word [] to form a unigram.

Step 2: Check whether it form a root form.

Step 3: If yes then save it to the Root Word List corresponding to the input word.

Step 4: If it do not form any root then again start from step 1 of Algorithm 1 by merging next sequential unigram of input inflected word [] array to form bigram and further more by merging N-grams to form *n* + 1-grams until the longest syllable does not form.

Step 5: If words are present in the Root Word List corresponding to input word then, from the list, word with longest sequence will be returned as an output.

Step 6: If there is no word present in Root Word List, then Algorithm 2 will be executed.

Algorithm 2 (inflected word [char], string Stripped word)

Step 1: Merge two sequential unicodes from the end of the unicode list of the input word.

Step 2: Check whether it form any suffix from the list.

Step 3: If yes then chop the suffix and perform corresponding normalization process to the input word according to the specified rules and save it as Stripped word.

Step 4: Now break the Stripped word into corresponding unicode and save unicodes to inflected word [ ] array. Go back to step1. This is to check if than one suffix are present in the inflected word.

Step 5: If it does not form any suffix then we again start from step 1 of this algorithm by merging next sequential unigram, from the end, to form bigram and further more by merging bi-grams to form tri-grams.

Step 6: Stripped word will be given as output.

Now apply the proposed algorithm on some examples to get better illustration of these algorithms. Whenever any words is given to the systems, firstly Algorithm 0 will be executed. It breaks the input word and save the corresponding unicodes into the array so that it can be further used by Algorithm1 and Algorithm2 to Suppose the input word is

input word = मैदानों (Fields)

Then following steps will be performed to get the lemma:

Algorithm 0 will be executed as following:

→ Break the input word into corresponding unicodes i.e.,

inflected word [] = म, ै, द, ा, न, ो, ं

Now **Algorithm 1** will be executed for inflected word [] array. Every time the word formed by merging will be matched to root words and if it will successfully matched then the merged word will be stored in corresponding Root Word List as illustrated below:

(a) मै → This is not any root word

(b) मैद → This is not any root word

(c) मैदा → This can be a root word. It will be added to Root Word List of input word.

(d) मैदान → This can be a root word. It will be added to Root Word List of input word.

(e) मैदानो → This is not any root word.

(f) मैदानों → This is not any root word.

Now the Root Word List of input word मैदानों contains two roots मैदा (flour) and मैदान (field). But मैदान (Field) is having longer sequence of unicodes, therefore it will be given as output. Since output is given by Algorithm 1 it will not execute Algorithm 2.

Now let us take such example for which Algorithm 1 will failed to generate output and the input word will be then given to Algorithm 2. Let us suppose the word वशिष is not present in our data structure.

वशिषताएँ (Specialities)

Note that to elaborate every step of Algorithm 2, word with more than one suffix is taken as an example.

Following steps will be performed to get the lemma:

Algorithm 0 will be executed as following:

→ Break the input word into corresponding unicodes i.e.,

inflected word [] = व, ि, श, े, ष, त, ा, ए, ँ

Algorithm1 will be executed for the above word taking inflected word [] array as input but it will failed as the word is not present in our root words. Therefore, input will be send to Algorithm2 to execute.

Now **Algorithm 2** will be executed as following:

(a) Merge the last two unicodes to form a unigram from inflected word [] array. It will be एँ.

(b) Check the suffix list, if एँ is present.

(c) Yes the suffix is present hence apply the corresponding rule. That is, in case of एँ, simply chop the suffix and give the remaining words as an output.

(d) Now the Stripped word = वशिषता.

Again break it into corresponding unicodes and save it to the input array i.e.

inflected word [] = व, ि, श, े, ष, त, ा

(e) Go to step 1.

(f) Again performing step 1 that is merging last two unicodes to form unigram. It will be ता.

(g) Checking if ता is suffix or not.

(h) Yes, it is suffix and as per rule, it will be chopped and remaining word will be given as output.

(i) Stipped word = वशिेष

Again break Stripped word वशिेष into corresponding unicodes and save it into input array i.e.

inflected word [] = व, ि, श, े, ष.

(j) Go back to step 1.

(k) Merge last two unicodes े ष.

(l) Checking if this is suffix or not.

(m) No this is not suffix.

(n) Merge last last three unicodes शेष.

(o) Checking if it is suffix or not.

(p) No, it is not a suffix.

Therefore the output will be the last stripped word which is वशिेष(Special).

Hence in this way if any inflected word with the suffix for which rules are not defined, can also handled by system and chances are high to get the correct lemma for the input.

5. CONCLUSION

The proposed approach is purely based on the observations from the standard dataset Using this approach for development of lemmatizer for Hindi language in Devanagari script, more accuracy can be achieve as compared to rule based lemmatizer. It includes not only those words with predefines prefixes and suffixes but also those words that are having unknown suffixes and are present in out data structure used for the storage scheme.

This approach emphasized on word level lemmatization. The lemmatizer can further be improved by applying the sentences level lemmatization process which include context based comparisons to give the correct lemma as an output.

Many inflected words have more than one word present in corresponding Root Word List. In the proposed algorithm, the word with longest sequence of unicodes will be given as output. It may not always give the correct lemma corresponding to the input inflected word. But if sentence level lemmatization will be applied, it will consider the neighboring words of the given input in sentence and based on that output will be given. In sentence level lemmatization, using probability models, it will check the probability of the words in the list with the neighboring words and output will be the word having highest probability.

Consider the following example with word मैदानों

बच्चे मैदानों मैं खेल रहे है |

(Children are playing in the fields.)

In above sentence, for inflected word मैदानों as an output, there are two words मैदा (flour) and मैदान (field) in Root Word List.

The probability of मैदा (flour) and मैदान(field) with बच्चे (Chidlren) may be nearly same but the probability of मैदान (field) with खेल (play) will always be higher than the probability of मैदा(flour) with खेल(play). Hence in this way always correct lemma will be given as an output.

References

[1] A. Ramnathan, D Rao, "A lightweight Stemmer for Hindi," In Proceedings of Workshop on Computational Linguistics for South Asian Languages, 10 th Conference of the European Chapter of Association of Computational Linguistcs. Pp 42-48. 2003.

[2] Plisson, J, Larc, N, Mladenic, "A Rule based approach to word lemmatization," Proceedings of the 7 th International Multiconference Information Society,

IS-2004, Institute Jozef Stefan, Ljubljana, pp.83- 86, 2008.

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

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

[5] Nikhil K V S, "Hindi derivational morphological analyzer," Language Technologies Research Center, IIIT Hyderabad, 2012.

[6] Shashi Pal Singh, Ajai Kumar, Dr. Hemant Darbari and Anshika Gupta, "Improving the quality of Machine Translation using Rule Based Tense Synthesizer for Hindi," IEEE International Advance Computing Conference (IACC), 2015.

[7] Snigdha Paul, Nisheeth Joshi and Iti Mathur, "Development of a Hindi Lemmatizer," Vol. 2 International Journal of Computational Linguistics and Natural Language Processing, Issue 5, 2013.

[8] Itisree Jena, Sriram Chaudhary, Himani Chaudhary and Dipti M.Sarma,"Developing Oriya Morphological Analyzer Using Lt-toolbox," ICISIL 2011, CCIS 139, pp. 124-129, 2011.

[9] John goldsmith, 2001, Unsupervised learning of the morphology of a Natural language, Computational Linguistics, Volume 27, No. 2 pp. 153198, 2001

[10] Martin F. Porter, An algorithm for suffix stripping, Program, Vol. 14, No. 3, pp 130-137, 1980.

[11] Deepa Gupta, Rahul Kumar Yadav, Nidhi Sajan, "Improving Unsupervised Stemming by using Partial Lemmatization Coupled with Data-Based Heuristics for Hindi, " International Journal of Computer Application(0975-8887), Vol. 38, No. 8, January 2012.

[12] Mohd. Shahid Hussain, "An unsupervised approach to develop stemmer," International Journal on Natural Language Computing, Vol. 1, No. 2, August 2012.

[13] Upendra Mishra, Chandra Prakash, "MAULIK: An Effective Stemmer for Hindi Language", International Journal on Computer Science and Engineering (IJCSE), ISSN: 0975-3397, Vol. 4 No. 05 May 2012.

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

[15] Wolfgang Lezius, Reinhard Rapp, Manfred Wettler, "A freely available morphological analyzer, disambiguator and context sensitive lemmatizer for German", COLING '98 Proceedings of the 17th international conference on Computational linguistics - Volume 2, Association for Computational Linguistics Stroudsburg, PA, USA ©1998.

[16] http://stanfordnlp.github.io/CoreNLP/

[17] https://en.wikipedia.org/wiki/Hindi