An Unsupervised Hindi stemmer with heuristic improvements

Amaresh Kumar Pandey
Indian Institute of Information Technology- Allahabad
Deoghat, Jhalwa
Allahabad, India-211012
+91-532-2922240
amareshk.p@gmail.com

Tanveer J Siddiqui
Indian Institute of Information Technology- Allahabad
Deoghat, Jhalwa
Allahabad, India-211012
+91-532-2922240
tanveer@iiita.ac.in

ABSTRACT
Stemmers are used to convert inflected words into their root or stem. Stem does not necessarily correspond to linguistic root of a word. Stemming improve performance by reducing morphologically variants into same words. This paper presents an approach is to develop unsupervised Hindi stemmer. This paper focus on the development of an unsupervised stemmer for Hindi and evaluation of approach using manually segmented words. We evaluate our approach on 1000-1000 words randomly extracted words (only) from Hindi WordNet[1] data base. The training data has been constructed by extracting 106403 words extracted from EMILLE[2] corpus. The observed accuracy was found to be 89.9% after applying some heuristic measures. The F-score was 94.96%. As the algorithm does not require any language specific information, it can be applied to other Indian languages as well. We also evaluate the effect of stemmer in terms of reducing size of index for Hindi information retrieval task. The results have been compared with light weight stemmer [10] and UMass stemmer [17]. Test run shows that our stemmer outperforms both the stemmer.

Keywords
Unsupervised morphological analyzer, Hindi, heuristics and optimal word segment.

1. INTRODUCTION
A large volume of text is now available in electronic form in multiple languages including Hindi which is one of the most widely spoken languages in India. In order to provide access to this information the need of multi-lingual text retrieval system is increasingly felt. This has made language processing an active area of research. Early work in this area was mainly focused on English. However, the past decade has witnessed proliferation of research on Asian language processing. Still, the scarce availability of tools and other lexical resources for languages other than English and major European languages put major obstacle on the work in this direction. This is especially true for languages from Indian sub-continent. This work focuses on development of one such tool, namely stemmer for Hindi language. Almost all of the search and retrieval system use stemmer to reduce morphological variants to their stem.

Stemmers are simplest morphological systems that collapse morphological variants of a given word to one lemma or stem. The stems need not be a valid linguistic root of the word; it is usually enough that related words map to the same stem. For example, the words लडका, लडको, लडकें are reduced to same stem, लडक. Stemmers are used in text retrieval systems to reduce index size. Reducing morphological variants to same stem eliminates the need of using all variants for indexing purpose. This helps in improving recall. However, it may sometimes degrade performance.

Hindi is inflectionally rich language. Hindi words may have many morphological variants. These variants are mostly generated by adding suffixes to the root word For example, Hindi plural nouns are generated by adding suffixes like, ें,ों, ियाँ, ियों. A word may have a number of morphological variants. For example, अधिकार, अधिकारिक, अधिकारियों, अधिकारी, it will reduce into अधिकार. Similarly following inflected words खरीद, खरीदता, खरीदने, खरीदी, खरीदेगा, खरीदेगी are reduced to same stem खरीद.

Spelling variations are more subtle in Hindi than in English. For example, the Hindi word "अंग्रेज़ी" can have following possible spelling variations: अंग्रेज़ी, अंग्रेज, अंग्रेजों

Variability in Unicode encoding is another problem. Both spelling Unicode variations create problem in stemming. We have used UTF-8 code. However, no attempt to spelling normalization has been made.

Standard stemmers are available for English and other European languages. However, no such standard stemmers are available for Hindi and most of other Indian languages. Two widely known stemmers for English have been developed by Lovins [13] and Porter [14]. Both Porter's and Lovin's stemming algorithms rely on linguistic rules for stemming. It is possible to devise such riles for Hindi that would reduce inflectional variant to its stem.

[1] http://www.cfilt.iitb.ac.in/

[2] http://bowland-files.lancs.ac.uk/corplang/emille/[2]

AND '08, July 24, 2008 Singapore

However, this is a time consuming process. Further, the efforts need to be duplicated for a different language as the rules are highly language dependent.

Supervised approach for stemmer requires a set of inflection-root pair to learn suffixes. Training set has to be created manually which is a time consuming process and requires linguistic knowledge. Therefore, we focus on the development of unsupervised approach which only requires a list of words for learning suffixes. Such a list can be easily extracted from the raw corpus.

In this paper, we describe the design of an unsupervised Hindi stemmer. Our approach uses a set of words to learn stemming rules automatically. We have extracted 106403 words from Hindi documents of EMILLE corpus. We propose the use of simple heuristics to avoid over stemming in some cases. For testing we randomly selected 1000 words from Hindi WordNet database and manually created inflection-root. Two different evaluations have been made. The first evaluation measures the performance of our stemmer in terms of accuracy, recall, precision and F-score. Two test-runs have been made. We observed an F-score of 92.7 and 94.9 respectively. Test run have also made using rule-based stemmers reported in [17 and 10] and the results are compared with our stemmer. The second evaluation measures the performance of our stemmer for indexing task. The percentage reduction in index size achieved through the use of the stemmer has been compared with the reduction achieved through the use of light weight stemmer [10] and UMass stemmer [17. Our stemmer outperforms both the stemmer. Our algorithm does not require any linguistic input. Hence, it can be easily adapted to other languages. The approach used in this work is *unsupervised as it does not require inflection-root pair* for training. This makes it easily applicable to a new language. It is *language independent* because it does not require any language specific rules as input. The approach does not require any domain specific knowledge; hence it is *domain independent as* well. The algorithm is computationally inexpensive. The number of suffixes in the final list is 51 and longest suffix strippig is used to perform stemming. Some post-processing heuristic repairs have been performed to further refine the learned suffixes.

The rest of the paper is organized as follows: Section 2 reviews earlier work reported on morphology. Section 3 presents the detail of the approach used in this work. Experimental investigation has been made in section 4. Section 5 discusses the results of our experiment. Conclusions have made in section 6.

2. RELATED WORK

The development and use of stemming algorithms are not new initiatives. First ever published stemmer was written by Julie Beth Lovins in1968 [13]. This paper was remarkable for its early date and had great influence on later work in this area. Another widely known stemmer was developed by Martin Porter [14]. These early stemmers were rule-based. Formulating such rules for a new language is time consuming. The Word Frame model proposed by Wicentowski [25] is a supervised approach for stemming. It requires a set of inflection-root pairs to learn a set of string transduction. These transductions are then used to reduce unseen inflections into their root.

A lot of work on morphology focuses on unsupervised approaches (23, 22, 3, 4, 8). [23] Brent et al. [23] proposed an information-theoretic approach based on Minimum Description Length(MDL) framework. Brent [22] and Snover and Brent [23] later proposed a Bayesian model for MDL for English and French. Goldsmith's [3] work is based on minimum description length principle which prefers a model that gives rise to minimal length coding of observed data set whereas the work in [4] uses prior distribution of morpheme length and frequency to find a good morpheme boundary. Yet another work on stemming was reported by Freitag [24]. His work is based on automatic clustering of words using co-occurrence information. The suffixes have been identified using orthographic dissimilarity between the words in different clusters.

Earlier work on Indian language morphology includes [17, 10, 12, 5]. Larkey et al [17] used a light weight stemmer in their work that uses a list of 27 common suffixes to perform stemming. This list was formulated manually. A similar approach was used by Ramanathan and Rao [10]. Their manually extracted list consists of 65 inflectional suffixes. Chen and Gey [12] used a statistical stemmer for Hindi and evaluate it s performance for Hindi text retrieval. They found that stemming fails to improve retrieval. Dasgupta and Ng [5] present an unsupervised morphological analyzer for Bengali. They reported a maximum accuracy of 64.6% when tested on 5000 words.

Other reported work include [1, 2]. [1] Proposed an unsupervised algorithm to improve existing morphological analysis and generation of Hindi. Their approach is based on 'observable paradigm' and requires existence of paradigm for different morphological class. A Telugu morphological generator, TelMore, is outlined by [2]. TelMore is a rule-based approach. It uses the results of analysis of Telugu performed by [6, 7].

The work presented in this paper focuses on the development of an unsupervised Hindi stemmer to be used in retrieval task. This work is a part of our ongoing work on cross-lingual information retrieval.

3. OUR APPROACH

Our approach is partly in line with Goldsmith (2001) [3] approach. It is based on split-all method. For unsupervised learning (training), words from Hindi documents from EMILLE corpus have been extracted. These words have been split to give n-gram (n=1, 2, 3 … *l*) suffix, where *l* is length of the word. Then we compute suffix and stem probaility. These probabilities are multiplied to give split probaility. The optimal segment correspond to maximum split probabilty. Some post-processing steps have been taken to refine the learned suffixes. Figure-1 outlines the steps involved in our algorithm. The details of these steps are presented below:

3.1 Word segmentation

This step splits word j(W_j) into stem and corresponding suffixes as:

$$W_j = \{stem_{1j}, suffix_{1j}; stem_{2j}, suffix_{2j}; \ldots; stem_{ij}, suffix_{ij}\}$$

Where $stem_{ij}$ is ith stem of jth word and $suffix_{ij}$ is ith suffix of jth word.

For example, the word 'अँगुली' is split into following stem and suffix :

{अ ँगुली, अँ गुली, अँग ुली, अँगु ली, अँगुल ी, अँगुली NULL }

The suffixes correspond to n=1,2,3,…,l where l is length of the word.

Next, we have created three separate lists of stem, suffix and split and compute initial probability of stem as follows:

Input: List of words
Output: Suffix list

Step-1 Segment words as:
W_j
$= \{stem_{1j}, suffix_{1j}; stem_{2j}, suffix_{2j}; \dots; stem_{ij}, suffix_{ij}\}$
Step-2 Initialize stem, suffix and split probability using the following expression:

$$P(stem_{ij}) = \frac{f_{W_j}}{T_w * len(W_j)}$$

$$P(suffix_{ij}) = \frac{f_{W_j}}{T_w * len(W_j)}$$

$$P(split_{ij}) = P(stem_{ij}) * P(suffix_{ij})$$

Step-3 Re-compute probabilities using Naïve bays probability distributions. The exact expressions used are:

$$P_{new}(stem_{ij}) = \sum_{k=0}^{m} \{P(stem_{ij})\}$$

$$P_{new}(suffix_{ij}) = \sum_{k=0}^{m} \{P(suffix_{ij})\}$$

$$\mathrm{P_{new}}(split_{ij}) = P_{new}(stem_{ij}) * P_{new}(suffix_{ij})$$

Step-4 Repeat ste-3 until convergence
Step-5 get optimal segment.
Step-6 create equivalence classes, it is a two steps process. In first step, we group suffixes with similar stem

$$\{\{stem_i\}; \{suffix_i, suffix_j, suffix_k, \dots\}\}$$

In second step, we group stems having similar suffix list.

$$\{\{stem_i, stem_j\}; \{suffix_i, suffix_j, suffix_k, \dots\}\}$$

Step-7 Heuristics improvement in equivalence classes
Step-8 Generate suffix list using the output produced in step 7.

Fig-1 steps involved in our algorithms

$$P(stem_{ij}) = \frac{f_{W_j}}{T_w * len(W_j)}$$

Where, f_{W_j}= Frequency of word W_j in corpus.

T_w = Total number of words in corpus.

$len(W_j)$ = Length of word W_j (number of character).

Similarly we have calculated initial of suffix is as follows:

$$P(suffix_{ij}) = \frac{f_{W_j}}{T_w * len(W_j)}$$

The initial probability of each split is calculated using following distribution:

$$P(split_{ij}) = P(stem_{ij}) * P(suffix_{ij})$$

Stem and suffix probability is computed based on their frequency in the corpus. Intuitively, it seems that maximum split probability gives the optimal segments of the word. Initially we have assigned equal probability to stem and suffix.

3.2 Updating stem and suffix probability

As iteration proceeds, stem and suffix probability changes. The stem probability is recomputed based on how many times similar stem occur in whole corpus. More formally:

$$P_{new}(stem_{ij}) = \sum_{k=0}^{m} \{P(stem_{ij})\}$$

Where m= number of splits having same stem ($stem_{ij}$)
Similarly suffix probability is re-computed as:

$$P_{new}(suffix_{ij}) = \sum_{k=0}^{m} \{P(suffix_{ij})\}$$

Where m= number of splits having common suffix ($suffix_{ij}$),
The split probability is updated using new stem and suffix probabilities using the following expression:

$$\mathrm{P_{new}}(split_{ij}) = P_{new}(stem_{ij}) * P_{new}(suffix_{ij})$$

The iteration continues till convergence is achieved.

3.3 Obtaining optimal segment

The split with maximum probability is considered to be optimal segments of word. The optimal breakpoint largely depends upon the probability distribution of split of the word, language and corpus. Some heuristics have been considered while taking optimal split. First, we restrict the minimum length of stem to three. This heuristic avoids reducing words like khata, khas, khara, etc to same stem 'kha'. The second heuristic is based on the probability distribution of splits of word. The rough characterization of split probability is shown in fig-2. This figure is not a plot of actual dat. It is just a conceptual representation of probability distribution of splits within words. As evident from the fig. 2, the probability of split first decreases and then starts increasing again. The point where the probability starts increasing again, shown in bold in fig.2, is considered as optimal segmentation point. As mentioned earlier, split probability is product of stem and suffix probability. As the suffix length increases, suffix probability decreases because suffix of length 1 are more probable than of length 2, 3, etc. After attaining a minimum value it starts increasing again. This is due to the increase in stem probability due to the reduction of otherwise dissimilar words into same stem. This is actually an indication of over-stemming.

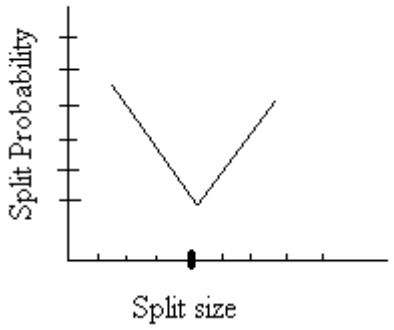

Fig-2. probability distribution of split of word

3.4 Deriving equivalence classes of word

After computing optimal splits, we group suffixes with similar stem together. This process is shown below:

$$\{\{stem_i\} : \{suffix_i, suffix_j, suffix_k, \ldots\}\}$$

Where, $suffix_j$, $suffix_k$ … have the same stem, $stem_i$, as the $suffix_i$. For example, ◌ाँ, ◌ां, ◌ों have similar stem पद्धतियFirst . step yields the group {पद्धतिय#◌ाँ ◌ां ◌ों}.

Next, we group stems with equal set of suffixes. For example, if $stem_i$ and $stem_j$ both have the same set of suffix then we group them together as shown below:

$$\{\{stem_i, stem_j\} : \{suffix_i, suffix_j, suffix_k, \ldots\}\}$$

Here $stem_i$ and $stem_j$ both contains same set of suffixes say $\{suffix_i, suffix_j, suffix_k, \ldots\}$.

For example, छुट्टिय and पद्धतिय consist similar set of suffix {◌ाँ, ◌ां, ◌ों }. So we group them to yield {{छुट्टिय पद्धतिय} : { ◌ाँ ◌ां ◌ों}}.

Table 1 shows sample classes derived after this step. We observed some error in grouping of suffix and stem. Hence, we apply some heuristics to further improve these classes as discussed in the following section.

Table1: Sample classes

S.N.	Classes (left side of # is Stems and right side of # is Suffixes)
1.	जावे सोचे#NULL ं ंगे गा गी
2.	खाये सुने#NULL ं ंगे गा
3.	अंतत मूलत#NULL ः
4.	चौथा चौड़ा रंगा रोपा लंबा#NULL ई
5.	उपजा टिका#NULL ऊ ने
6.	जमीं ठेके#NULL दार दारी दारों
7.	ईमान चौकी#NULL दार दारी
8.	अछूत छोकर ठण्ड तख्त बरतन#NULL ा ी े
9.	लिखत लिपट#NULL ा ी
10.	अँग्रेज अँग्रेज़ गवाह चाकर डकैत दलाल#NULL ी ों

3.5 Post-processing repair with heuristics

In order to improve the performance of our algorithm, we apply a heuristic that attempts to concatenate stems with common prefix string of suffixes. It is observed that in many cases the set of suffixes having similar stem involve common prefix as depicted by the following expression:

$$suffix_i = c_1c_2x_1y_1 \ldots$$
$$suffix_j = c_1c_2x_2y_2 \ldots\ldots$$
$$suffix_k = c_1c_2x_3y_3 \ldots\ldots$$

Here $suffix_i$, $suffix_j$, $suffix_k$ have a similar stem, $stem_i$ and similar prefix substring c_1c_2 . The application of this heuristic concatenates $stem_i$ with suffix c_1c_2 and modifies suffixes accordingly by dropping common prefix from them. For example, suffixes यक, यकता and यकतानुसार contain the same stem आवश् and similar prefix substring यक. This heuristics will concatenate the stem आवश् with the common prefix string of suffixes, यक , to produce the stem आवश्यक and it drops the common prefix from stems to give NULL, ता, and तानुसार. Table 2 shows final suffix list.

Table 2 Suffix list of our stemmer

ुद्दीन ्रोजन इयों लैंड ियाँ ियों िस्ट ंगे करण कों गों नों पुर मान याँ यों रों एँ एफ ओं ओं कर गी गे जी डी ता ते ना ने या वर ◌ाँ ◌ां ◌ाल िक ित ीय ेंं ोंं ्स ँ ं ई ए फ ◌ा ी ु ू े

3.6 Stemming of new words

The output of our unsupervised stemming algorithm is the set of suffixes. Each suffix roughly captures some morphological variation. We have used longest suffix matching during stemming. This means that longest possible suffix of the target word matching with some suffix in the list is dropped. The suffix list is arranged in decreasing order of length to facilitate it. We start matching form first suffix to last suffix, where a match is found we segment the word in its stem and suffix. Fig. 3 shows sample output of our system.

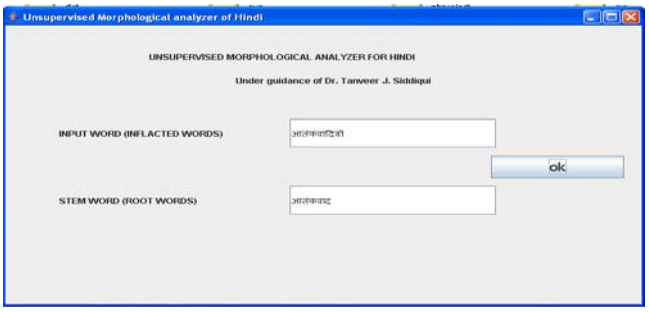

Fig-3 Sample output

4. EXPERIMENT

In order to evaluate the proposed stemmer, two different experiments have been performed. The first experiment evaluates the stemmer in terms of accuracy, precision and recall measure. The second experiment evaluates the stemmer in terms of its ability to reduce the index size in an information retrieval task. Comparisons have been made with light weight stemmer [10] and UMass stemmer [17]. The training data set has been constructed by extracting 106403 words from 225 documents taken from EMILLE corpus. Table-3 summarizes some of the statistics of these datasets. Two different test sets have been created by randomly extracting words from Hindi Word-Net database.

Table 3: Statistics of Training Data set

Feature of Data set	Training Data-1(EMILLE)
Total words	106403
Unique Words	10548
All segment	67277
Min length	4
Max Length	18

Experiment 1: First experiment evaluates the performance of the stemmer in terms of accuracy, recall, precision and F-score. Two different runs (run 1 and Run 2) have been conducted. These run correspond to two different test sets. The evaluation measures used in this experiment are described below:

Accuracy is defined as fraction of words stemmed correctly.

Recall is defined as the ratio of length of stem produced by our stemmer and the length of standard stem. If length of observed stem is less than length of standard stem then recall is considered 100% because all the characters in the observed stem are also present in the standard stem. However, the precision will be low in this case because standard stem will have some extra character.

Precision is defined as the ratio of length of standard stem and length of the stem produced by our stemmer. If the length of observed stem is greater than the length of standard stem then precision will be considered 100% because all the character present in standard output is also present in observed output

F-score is defined as the harmonic mean of recall and precision.

Mathematically, If length of standard stem is greater than length of actual stem than, recall is 100% and precision is calculated using below formulae:

$$precision = \begin{cases} 1, & if\ length\ of\ standard\ stem > length\ of\ actual\ stem \\ \frac{\sum_{i=0}^{m} \frac{lenght\ of\ actual\ stem}{length\ of\ standard\ stem}}{m}, & \text{otherwise} \end{cases}$$

If the length of standard stem is less than the length of actual stem then, precision is 100% and recall is calculated using below formulae,

$$recall = \begin{cases} 1, & if\ length\ of\ standard\ stem < length\ of\ actual\ stem \\ \frac{\sum_{i=0}^{m} \frac{lenght\ of\ standard\ stem}{length\ of\ actual\ stem}}{m}, & \text{otherwise} \end{cases}$$

Where m is total number of test words.

F-score is calculated as:

$$F - Score = \frac{(2 * precision * recall)}{(precision + recall)}$$

Table 4 shows the results of the test runs. Table 5 and 6 show the results using light weight stemmer and UMass stemmer respectively.

Table: 4 Test Results using our unsupervised stemmer

Test Data	Accuracy	Precision	Recall	F-Score
Run-1	**85.685**	**96.196**	**89.489**	**92.721**
Run-2	**89.979**	**96.793**	**93.186**	**94.955**

Table: 5 Test Results using Lightweight stemmer

Test Data	Exact Accuracy	Precision	Recall	F-Score
Run-1	67.067	92.892	74.174	82.48
Run-2	70.140	94.388	75.851	84.111

Table: 6 Test Results using UMass stemmer

Test Data	Exact Accuracy	Precision	Recall	F-Score
Run-1	72.572	93.893	78.678	85.615
Run-2	78.657	96.793	81.863	88.704

Experiment 2: In the second experiment, we used the stemmer to index Hindi documents from EMILLE corpus and compute the reduction in index size. The reduction is measured as the difference in number of index terms with and without stemming. The baseline in this case is index size without stemming. Table 7 compares % reduction in index size obtained using our stemmer with reduction obtained using light weight stemmer and UMass stemmer.

Table: 7 Reduction in index size

Test parameter	**Without stemming (Baseline)**	**Stemming with our stemmer**	**Stemming with Lightweight stemmer**	**Stemming with UMass stemmer**
No. of words	Reference	**18.24 %**	12.348%	15.25%
Index Size	Reference	**15.01%**	9.12%	12.41%

5. RESULTS AND DISCUSSIONS

As shown in Table 4, we observed a maximum accuracy of 89.9% and F-score of 92.9 % using test set 2. The recall and precision figures indicate that the difference in accuracy in two test runs is mainly due to under-stemming. The maximum accuracy observed with light weight stemmer and UMass stemmer is 70.1% and 78.7% as shown in Table 6 and 7 respectively. In all the test runs the performance was found better using test set 2. Our stemmer outperforms both the light weight stemmer and UMass stemmer in terms of accuracy and F-score. Though, the UMass uses only 27 suffixes, its performance was found better than the light weight stemmer.

Table 7 shows the result of comparison in terms of index size reduction. The baseline performance in this case corresponds to indexing without stemming. The maximum reduction of 18.24 %, in terms of number of index terms, was obtained using our stemmer. Again the UMass was close to this performance with an observed reduction of 15.25%. The reduction achieved using lightweight stemmer was 12.35%. The reduction in terms of index size (measured in MB) was found to be 15.01 % with our stemmer.

6. CONCLUSION AND FUTURE WORK

The observed accuracy seems to be promising. We have achieved a maximum accuracy of 89.9%. Our approach is unsupervised and language independent. It uses a set of words to learn suffixes and does not require any linguistic input. Hence, it can be used for developing stemmers of other languages as well.

We have evaluated the performance of our stemmer in terms of accuracy, recall and precision measures and compared it with two other stemmers for Hindi. The performance of our stemmer was found better than both the stemmers. Yet another evaluation has been made in terms of its ability to reduce index size. Again, the %reduction was found better for our stemmer. However, we have not evaluated retrieval performance in terms of recall and precision using reduced index. We would like to do the same in future.

REFERENCE

1. Bharati, Akshar Rajeev Sangal, S.M. Bendre, Pavan Kumar, Aishwarya, Unsupervised Improvement of Morphological Analyzer for Infectionally Rich Languages, Proceedings of the NLPRS, 2001, pp 685-692
2. Ganapathiraju, Madhavi and Lori Levin: TelMore: Morphological Generator for Telugu Nouns and verbs, In the proceedings of Second International Conference on Universal Digital Library Alexandria, Egypt November 17-19, 2006
3. Goldsmith, John. (2001) Unsupervised Learning of the Morphology of a Natural Language. Computational Linguistics, Volume 27, No. 2, pp 153-198, 2001
4. Creutz, M. and Lagus, K. Unsupervised Morpheme Segmentation and Morphology Induction from Text Corpora Using Morfessor 1.0.Tech. rep. A81, Helsinki University of Technology, 2005.
5. Sajib Dasgupta, Vincent Ng, Unsupervised morphological parsing of Bengali, 2007
6. Brown, C.P., *The Grammar of the Telugu Language*.1991, New Delhi: Laurier Books Ltd.
7. Krishnamurti, B., *A grammar of modern Telugu*. 1985,Delhi ; New York: Oxford University Press.
8. Mathias Creutz (2003):Unsupervised segmentation of words using prior distributions of morph length and frequency. In Proceedings of ACL-03, the 41st Annual Meeting of the Association of Computational Linguistics, pages 280-287, Sapporo, Japan, 7-12 July.
9. Shone, Patrick and Jurafsky, Daniel, 2001: Knowledge-Free Induction of Inflectional Morphologies, In procc. Of second meeting of NACCL, pp. 183-191.
10. Ramanathan, A. and Rao, D. 2003. A lightweight stemmer for Hindi. In Proceedings of the 10th Conference of the European Chapter of the Association for Computational Linguistics (EACL), on Computatinal Linguistics for South Asian Languages (Budapest, Apr.) Workshop.
11. P. Majumder, M. Mitra, S.K. Parui, G. Kole, P. Mitra and K. Dutta, YASS: Yet another suffix stripper, ACM Transactions on Information Systems, Vol. 25, No. 4, pp. 18-38, October 2007
12. Chen, A. and Gey, F.C. 2003. Generating statistical Hindi stemmers from parallel texts. ACM Trans. Asian Language Inform. Process. Vol. 2, No. 3, Sep. 2003.
13. Julie Beth Lovins (1968).Development of a stemming algorithm. Mechanical Translation and Computational Linguistics 11:22–31. 1977

14. Porter, M. (1980). "An algorithm for suffix stripping program", Vol. 14, pp. 130-137
15. IIT-Bombay, http://www.cfilt.iitb.ac.in/
16. The EMILLE Corpus, http://bowland-files.lancs.ac.uk/corplang/emille/
17. Leah S. Larkey, Margaret E. Connell, and Nasreen Abduljaleel, "Hindi CLIR in Thirty Days", ACM Transaction on Asian Language Information Processing, Vol-2, No. 2, June 2003, Pages No. 130-142
18. W.B. Frakes and R. Baeza-Yates. 1992. *Information Retrieval: Data Structures Algorithms*. Prentice Hall, Englewood Cliffs, New Jersey, USA.
19. R.R. Korphage. 1997. *Information Storage and Retrieval*. Wiley Computer Publishing, USA.
20. Hafer, M. A., & Wess, S. F. (1974). Word segmentation by letter successor varities. Information Storage and Retrieval, 10, 371–385.
21. Harris, Z. (1955). From phoneme to morpheme. Language, 31(2), 190–222.
22. Snover, M. G., & Brent, M. R. (2001). A Bayesian model for morpheme and paradigm identification. In Proceedings of the 39th annual meeting of the ACL, pp. 482–490.
23. Brent, M. R., Murthy, S. K., & Lundberg, A. (1995). Discovering morphemic suffixes: A case study in minimum description length induction. In Proceedings of the fifth international workshop on artificial intelligence and statistics
24. Freitag, D. (2005). Morphology induction from term clusters. In Proceedings of the ninth conference on computational natural language learning (CoNLL), pp. 128–135.
25. R. Wicentowski. "Multilingual Noise-Robust Supervised Morphological Analysis using the WordFrame Model." In Proceedings of Seventh Meeting of the ACL Special Interest Group on Computational Phonology (SIGPHON), pp. 70-77, 2004.