A Devanagari Script based Stemmer

B. P. Pande[1], Pawan Tamta[2], H. S. Dhami[3]
[1, 2]SSJ Campus, Kumaun University
Almora, Uttarakhand,
[3]Vice Chancellor
Kumaun University
Nainital, Uttarakhand
India
bp.pande@rediffmail.com, pawantamta0@gmail.com, profdhami@rediffmail.com

Abstract: *Corpus based stemming has been devised to develop stemmers targeting language independent environment. These stemmers are applicable to all languages based on Latin script. In the present work, we exploit the English corpus for coded words of Devanagari script. We use the technique of Romanization and the stemmer is being tested over 100 randomly chosen Hindi words. We show that this approach has a direct application to the standardization of regional languages. For instance, we standardize the Kumauni language.*

Keywords: Information Retrieval (IR), Stemming, Script, N-gram, Devanagari, Kumauni

Received: 12 August 2014, Revised 18 September 2014, Accepted 24 September 2014

1. Introduction

Under the general Information Retrieval (IR) environment, basic goal is to retrieve documents relevant to the query seeded by an end user. This is achieved by matching semantic substance of the query with the semantic contents of the documents. With textual documents and queries, terms or words can be exploited to infer the meaning, and the IR system renders to match meaning of each query term with the meaning of each document term. Now, the *invariant* form of the word plays an important role for this term meaning. A general hypothesis is: words may have same meaning if they share a common *root* or *stem*. Stemming is the process of reducing words into their base or root form. Automatic stemming (or stemmer) is applied to words to overcome the mismatch anomalies related with text searching. By applying different techniques of stemming, additional documents can be retrieved as a response to a user query, that do not contain the exact query terms, but have word stems that matches with stems of query words. Thus stemming enhances the recall [13].

We have a rich literature of stemming algorithms- there are linguistic and statistical techniques. The former are *rule based* techniques [4] [10] [15] [23] [24] [27] [28], in which a priori knowledge of the morphology of a particular language is required. The latter are subject to exhaustive statistical analysis of the corpus, but work in a language-neutral way [8] [12] [16] [17] [18] [22] [25]. There are also some lexicon based approaches [14]. Linguistic stemmers or rule based stemmers have one major disadvantage

that they can not perform in multilingual environment. Statistical algorithms, such as one which is based on character frequencies can cope up with this limitation. The reader can have exhaustive details of stemming, its types and various techniques at [25].

Statistical stemmers can work in multilingual environment but they generally exploit English (Latin) based corpus. Therefore they can be applied to only those languages that share Latin script, for example French, Spanish, and Dutch.

1.1 Early work: Stemmers for Devnagari Script

Most of the reported work in the literature is dedicated to the English and other European languages. In grammar, inflection is a phenomenon in which a word is expended or modified to express different grammatical constructs such as tense, mood, voice, person, number, gender and case [5]. For example, in English language, *walks*, *walking* and *walked* are inflected forms of the basic verb to *walk*. An inflection exhibits one or more grammatical categories with a prefix, suffix or infix, or some other internal modification. Most of the Indian languages like Hindi (the national language of India), Tamil, Malayalam, Gujarati, Marathi, Bengali etc are highly inflected in nature. Hindi is considered as fourth most widely spoken language after Mandarin, Spanish and English [21]. Hindi language is based on a particular script, called *Devanagari* script. There are many Asian languages that are based on this script e.g. Marathi, Nepali, Kashmiri, Rajasthani, Kumauni etc. Ananthakrishnan Ramanathan and Durgesh D Rao [29] present a lightweight stemmer for Hindi. It is based on a predefined list of suffixes developed by the authors. Upendra Mishra and Chandra Prakash presented their stemmer called MAULIK [20]. They used the hybrid approach (combination of brute force and suffix removal approach). A rule based stemmer for nouns only is presented by Vishal Gupta [9]. All these techniques are linguistic technique since they depend on the morphological knowledge of Hindi language.

Some statistical techniques were developed for Hindi language as well. A statistical Hindi stemmer was developed by A. Chen and F. C. Gey [2]. An unsupervised Hindi stemmer developed by Amaresh Kumar Pandey and Tanveer J. Siddiqui [26]. But these techniques target Hindi language only, rather than a set of languages that are based on a same script.

1.2 Background and motivation behind present work

All the stemmers for Hindi language as mentioned above are based on the prior knowledge of Hindi language. Many Indo-Asian languages are based on the *Devanagari* script like, Hindi, Nepali, Kashmiri and Marathi etc. In the present work, we give the notion of a stemmer which is based on the *Devanagari* script rather than on a particular language. We hypothesize to develop a technique which is applicable to all those languages that share the *Devanagari* script, like Nepali, Angika, Kashmiri, Marathi, Hindi etc. We give a one to one mapping of each vowel and consonant of this script into an English code, called *roman* code. Our technique is based on the corpus frequencies of the coded form of the word. Our idea is to exploit English (Latin based) corpus for such roman coded words of *Devnagari* script. We demonstrate the application of the present work over Hindi language, the national language of India. We apply the technique to randomly chosen 100 Hindi words. Further we show that the same technique is applicable to the Marathi, Kumauni, Garhwali, Nepali languages also.

1.3 Application: Standardization of the Language

A direct application of our approach could be the standardization of the regional languages. Every language or dialect exists around us took birth as a spoken one. Oral transmission was the only medium for communication that stayed for generations. With the development of humans and their civilizations, verbal communication started losing its glory and the need for writing aroused. A standardized language or standardized dialect is a language variety used by a group of people in their public discourse [6]. Standardization is a process used to create a standard for such varieties. It takes place in a specific manner that depends on the community of people and factors such as geographical, historical and social aspects. Primary requirement for a variety of language to be standard is that it can frequently be used in public places or public discourse [6]. Some common features of a standard language are: a recognized dictionary, a recognized grammar, a standard pronunciation etc. [31]. Language standardization is a broader research area. Standardization consists of the imposition of uniformity upon a class of objects [19]. We use the stemmer developed for *Devnagari* script to standardize the Kumauni language. Kumauni language is spoken in Kumaun region of Uttarakhand in India. We standardize the Kumauni language over the dictionary feature, i.e. to develop rules to find and establish a standard form of any spoken/written Kumauni word from a variety of dialects.

2. Basic concepts and setup

In *Devanagari* script, we have 12 vowels which are shown in Table 2.1. These symbols are basically called *matras*, which are used together with consonants in a word formation process. There are 36 consonants in *Devanagari* script, shown in Table 2.2. We assign a unique character (or characters) from English alphabet system to each of these *Devanagari* vowels and consonants.

We thus define a one to one mapping from the set of *Devanagari* vowels and consonants to a set of character(s) from English language alphabet system. This unique character or characters is called the *roman code* of the corresponding vowel or consonant. Thus, a given Hindi word can be coded into a string of these roman codes. We call this equivalent coded form the *Romanized* word.

Vowel	अ	आ	इ	ई	उ	ऊ	ए	ऐ	ओ	औ	अं	अः
Matra Symbol	-	ा	ि	ी	ु	ू	े	ै	ो	ौ	-	-
Code	a	aa	i	ee	u	oo	e	ai	o	au	an	aH

Table 2.1. *Devanagari* Vowels, corresponding *Matras* and codes

क् k	ख् kh	ग् g	घ् gh	ङ Nga	च् ch	छ् chh	ज् j	झ् jh	ञ् Nj	ट् T	ठ् Th
ड् D	ढ् Dh	ण् N	त् t	थ् th	द् d	ध् dh	न् n	प् p	फ् f	ब् b	भ bh
म m	य y	र r	ल l	व् v	श् sh	ष् Sh	स् s	ह् h	क्ष Ksh	त्र tr	ज्ञ Gy

Table 2.2. *Devanagari* Consonants and corresponding codes

Apart from the vowels in table 2.1, we use the code M for 'ं'(called *chandrabindu*). Note that the symbol '्' (called *halant*) beneath each consonant in Table 2.2 denotes the atomic consonant without the '*अ*' (a) sound. For example, '*क*' in the word '*क्या*'. Now, the confluence of any consonant together with each the 12 vowel sounds (or *matras*) produces the basic entity of a word. Combination of corresponding roman codes of a consonant and any vowel sound (or *matra*) gives the roman code of that basic entity. Table 2.3 shows the same for the consonant '*क*'. We give the thorough codes of all possible combinations in Appendix A.1.

Vowel Sounds	-	अ	आ	इ	ई	उ	ऊ	ए	ऐ	ओ	औ	अं	अः
Combination	क्	क	का	कि	की	कु	कू	के	कै	को	कौ	कं	कः
Resulting Code	k	ka	kaa	ki	kee	ku	koo	ke	kai	ko	kau	kan	kaH

Table 2.3. 12 possible combinations of vowel sounds (or *matras*) for the consonant '*क*'

For a given Hindi word *क्या*, we can easily identify the basic entities. The concatenation of corresponding roman codes gives the roman code of the whole word. Say for example, in the Hindi word *'क्या'*, we have two basic enties, '*क्*' (k) *and* '*या*' i.e. the consonant *'य'* with the vowel sound '*आ*' (yaa). So the roman code for the word '*क्या*' is *'kyaa'*.

3. Methodology

For a given Hindi word to be stemmed, we first convert it to the *Romanized* word or code as explained in the previous section. As the coded form (*Romanized* word) consists of the characters from the English alphabet set, we look for the corpus frequencies of different overlapping character sequences that constitute the coded roman word. These overlapping character sequences are known as *N-grams.* An N-gram contains *N* consecutive characters of any word. For example, for the Romanized word *kyaa* (equivalent of the Hindi word '*क्या*'), we have four different N-grams: the 1-gram *k*, the 2-gram *ky*, the 3-gram *kya* and the 4-gram *kyaa*.

We tend to find out the corpus frequencies of such overlapping character sequences or N-grams. That is, for the word '*क्या*', how many times the character *k* occurs, how many times the string *ky* (*k* followed by *y*) occurs, how many time the string *kya* (*ky* followed by *a*) occurs and so on in a corpus. We call frequencies of such sequential character sequences as *sequential frequencies*. If a string has zero corpus frequency, it means it does not exist in that corpus. We idea is to exploit these sequential frequencies of the coded word. From the test data, we observe that after some number of steps, we're getting the continuous 0 frequencies. We hypothesize that this 0 frequency indicate the beginning of the Hindi word. We plot the partial word (N-gram) length verses sequential frequencies, for instance take the word '*खाना*' (*khaana*). We observe that there is a continuous decrease in the frequencies as the partial word (N-gram) length increases. After some steps, the frequency approaches the value 0 and it remains constant till the end of the word. When the frequency approaches 0 first time, we get a line parallel to the axis of *X* from that point (figure 3.1). Data for the word *khaanaa* is given in Table 3.1. Sequential frequencies are taken from COCA [3].

The parallel line contains the weakly minimal points of the region. Weakly minimal points of any set are the minimal points which lie on a straight line. For a given set *S*, a weakly minimal point can be defined as under:

Let x be an element of the set *S*. *x* is known as the minimal element of the set *S* if

$$\{x\} - cor\,(c) \cap s = \varnothing$$

Where *cor*(*C*) is the interior of the cone at x, see figure 3.2 [11].

Hindi Word	Roman Code	N-grams	Frequencies
खाना	**khaanaa**		
		k*	**421039**
		kh*	**4497**
		kha*	**3200**
		khaa*	**0**
		khaan*	**0**
		khaana*	**0**
		khaanaa*	**0**

Table 3.1. N-grams and corresponding corpus frequencies for the coded word *khaanaa*

Hindi Word	Roman Code	N-grams	Frequencies
बैठना	**baiThanaa**		
		b*	**2917024**
		ba*	**417942**
		bai*	**7031**
		baiT*	**1124**
		baiTh*	**0**
		baiTha*	**0**
		baiThan*	**0**
		baiThana*	**0**
		baiThanaa*	**0**

Table 3.2. N-grams and corresponding corpus frequencies for the coded word *baithana*

In figure 3.2 above, the region enclosed by the curve represents any set S and the lower region enclosed by two perpendicular lines represents the region $\{x\} - Cor\ C$). It is worth mentioning that the lower region does not include the boundary of *S*. Therefore the straight line section of region *S* represents the region of weakly minimal points.

We observe that the stem corresponds to one of these weakly minimal points. Out of the weakly minimal points, we identify that weakly minimal point as stem which corresponds to the first legal combination of coded characters having zero frequencies (see section 2). We say that any combination of coded characters as *legal*, if it can be decoded to form a complete word in Hindi language. A complete word always ends in a vowel. The word which ends in a consonant is not a complete word. For instance, for the word '*बैठना*' (*baithanaa*), we have 5 weakly minimal points (Table 3.2). These weakly minimal points correspond to words *baiTh*, *baiTha*, *baiThan*, *baiThana*, *baiThanaa*. Out of these words only *baiTha*, *baThana*, and *baiThanaa* are complete words as they end in a vowel. These combinations are thus legal combinations. Now, we identify the first legal combination *baitha* ('*बैठ*') as stem.

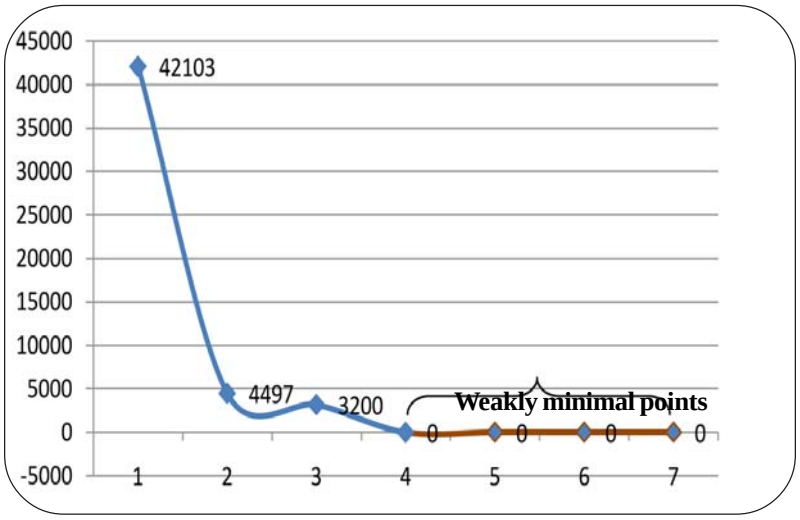

Figure 3.1. N-gram lengths vs. Frequencies

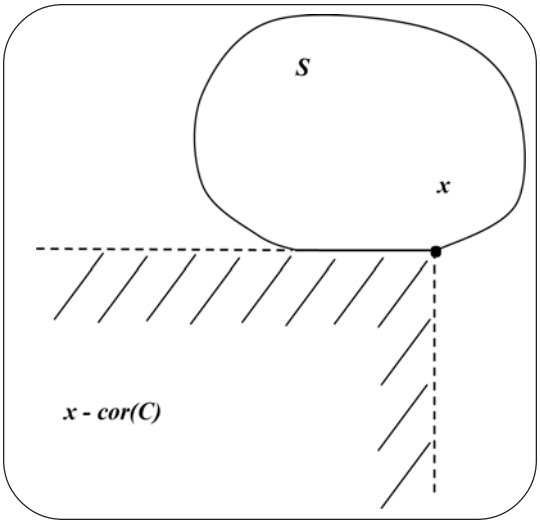

Figure 3.2. The region of weakly minimal points

4. Proposition of Algorithm

The crunch of the present method is the coding as described in section 2. We then look for the N-gram frequencies from some universal corpus and finally we decode the legal combinations back into the *Devanagari* script. The given word in *Devanagari* script is coded according to the coding scheme developed in section 2. The acquisition of sequential frequencies of the different N-grams is straight forward. We choose to obtain these frequencies from the COCA [3] corpus. This corpus has two benefits, first it is very rich and second, it is easy to calculate the N-gram frequencies using wild card [*].We then find the legal combinations. For example, take the word '*खाना*' (*khaana*), Table 3.1. For given coded word *khaana*, we measure the COCA [3] frequencies of various N-grams. The technique starts at first N-gram *k* (1-gram) and it proceeds to 2-gram viz. *kh*. We repeat the same process for higher N-grams unless we get the first legal combination. Now, this legal combination may or may not have zero frequency. We thus consider following two cases:

Case 1: A legal combination has a zero frequency. In our example, *khaa* is the first legal combination which has zero frequency. The zero COCA [3] frequency of this 4-gram implies that the string *kha* followed by *a* does not exist in the English corpus. Therefore, as mentioned in section 3, we recognize the string *khaa* as the stem of the input word. This legal combination *khaa* is then decoded into *Devanagri* script to get the stem '*खा*'.

Case 2: A legal combination has a non zero frequency. In this case, we decode the legal combination into the *Devnagari* script

and obtain the next legal combination. The next legal combination is also decoded into the *Devnagari* script. This new legal combination is now appended to the previously computed legal combination thereby giving updated legal combination. Simultaneously, the new decoded combination is appended to previously decoded combination in *Devnagri* script to get the updated decoded combination. This process is repeated again unless corresponding frequency of the current legal combination turns out to be zero.

We explain this case by means of an example. Consider the word '*बैठना*' (*baithanaa*), see Table 3.2. It is evident from Table 3.2 that the first legal coded combination is *bai* which is decoded into *Devnagari* as (*बै*). In Table 3.2, the legal combination *bai* has frequency 7031 therefore we do not consider it as a stem and proceed further. We obtain the second legal coded combination *Tha*. It is decoded into *Devanagari* as (*ठ*). We append *Tha* to *bai* to get the updated legal combination *baiTha*. Simultaneously we append (*ठ*) to (*बै*) to get the updated decoded combination'*बैठ*". Form table 3.2 the updated legal combination *baitha* corresponds to zero frequency therefore it is taken as the final stem and decoded into Devanagari as '*बैठ*'.

It is worth mentioning that if for any given word we get at most two combinations corresponding to zero frequency, then we do not follow the procedure mentioned above and take whole word as stem. For example, consider the words '*लेटना*' (*leTanaa*) and '*रोटी*' (*rotee*). Data for the coded string *leTanaa* and *rotee* is given in Table 4.1. Observe that, for the coded string *leTanaa*, there are exactly two entries with zero frequencies. Therefore, we do not apply the above stemming procedure and take the whole string *leTanaa* i.e. '*लेटना*' as the stem. Similarly for the coded string *rotee*, since we have only one entry with zero frequency, we take the whole string *rotee* i.e. '*रोटी*' as the stem.

4.1 The stemming Algorithm

In this section we propose an algorithm for *Devnagrai* script namely "*Stem-Hindi*". The steps are mentioned as under:

Procedure: Stem-Hindi

1. The given word in *Devanagri* script is coded into the *Romanized* code according to the coding explained in section 2.
2. The sequential frequencies of different N-grams of this *Romanized* code are taken from the COCA (Corpus of Contemporary American English) [3].
3. If we have at most two zero frequencies, then the whole word is considered as the stem, and stop. Else, move to next step.
4. We find the first legal combination. If this legal combination corresponds to zero frequency, we decode this combination into *Devaganari* script to get the stem and stop. Else, move to next step.
5. Find the next legal combination and decode it into *Devaganari* script. Append this current legal combination to the previously

Hindi Word ~	Roman Code	N-grams	Frequencies
लेटना	leTanaa		
		l*	1579176
		le*	452985
		leT*	46177
		leTa	31
		leTan	20
		leTana	0
		leTanaa	0
रोटी	rotee		
		r*	1155249
		ro*	275704
		rot*	6366
		rote*	159
		rotee*	0

Table 4.1. N-grams and corresponding corpus frequencies for the coded word *leTanaa* and *rotee*

calculated legal combination. Also, append the current decoded combination to the previously decoded combination.

6. If the current legal combination corresponds to zero frequency then the updated decoded combination is stem and stop. Else repeat step 6. unless we get the current legal combination corresponding to zero frequency or till the whole input word (all N-grams) is exhausted.

Word	Stem	Word	Stem	Word	Stem
खाना	*खा*	घूमना	घू	दामाद	दामा
काटना	*काट*	दिशाऐ	*दिशाऐ*	नौकरी	नौक
पीना	*पीना*	शब्दकोश	*शब्द*	व्यापारी	व्या
प्यासा	प्या	निशाचर	*निशा*	गीतकार	*गीत*
सोना	*सोना*	खिडकी	खिड	वैज्ञानिक	वैज्ञा
लेटना	*लेटना*	समय	*समय*	पुजारी	पुजा
बैठना	*बैठ*	खनिज	खनि	मजदूरी	मज
मारना	*मार*	वास्तविक्ता	वास्	आविष्कारक	आवि
उडना	*उडना*	असत्य	असत	लेखक	*लेख*
दौडना	*दौड*	पुरखा	*पुरखा*	विचारक	विचा
गणितज्ञ	*गणित*	नदी	*नदी*	ज्वालामुखी	ज्वा
किसान	किसा	महासागर	*महा*	नास्तिक्ता	नास्
प्रबंधक	*प्रबंध*	शहर	*शहर*	मानवता	मान
रोटी	*रोटी*	स्वतंत्रता	स्वतं	करना	*करना*
जानवर	जान	गुलामी	गुला	उतरना	*उतर*
शरीर	*शरीर*	आतंकवाद	आतं	सुलाना	*सुला*
आत्मा	आत	गर्भपात	*गर्भ*	बैठाना	*बैठा*
वस्तु	*वस्तु*	शांति	शा	समझाना	*समझा*
मरज़ी	*मरज़ी*	हिन्दू	*हिन्दू*	खोजना	*खोज*
चेतना	चेत	इसलाम	इसला	बेचना	*बेच*
देनदारी	देनदा	धरना	*धरना*	स्वामी	स्वा
मानदेय	मान	सेना	*सेना*	पारायण	पारा
तापक	*ताप*	तपोवन	*तपो*	चौकन्ना	*चौक*
कुहरा	*कुहरा*	खोखलापन	खोख	घमंडी	घमं
मान्यता	मान्	गिरगिट	गिरगि	कोपभाजन	कोपभा
ईश्वराधीन	ईश	रवाना	रवा	गुरुकुल	*गुरु*
वकालतनामा	वका	जानकार	जान	अतिथि	अति
तमाचा	तमा	परिधि	*परिधि*	शब्द	*शब्द*
बहुलता	बहु	चिकित्सालय	चिकि	समीकरण	समीक
छीनना	छी	शालीन	शाली	युवराज	युव
गरिमा	*गरिमा*	बालकपन	*बाल*	विकसित	विकसि
त्यागपत्र	त्या	लहराना	*लहरा*	जातीवाद	*जाती*
तरकश	तरक	सदाचार	*सदा*	उम्मीदवार	उम्मी
रईस	*रईस*				

Table 5.1. Hindi words and their stems

Following table 5.2 exhibits stems resulted with our script based technique. Six different clusters are being taken into consideration. First three clusters are of Hindi language, and remaining are of Marathi, Nepali and Kumauni languages respectively. All these languages are based on *Devanagari* script. N-gram corpus frequencies are again taken from COCA [3].

Cluster	Word	Stem	Cluster	Word	Stem
1 (Hindi)	प्रबंध	प्रबंध	4 (Marathi)	करा	करा
	प्रबंधक	प्रबंध		करावे	करा
	प्रबंधन	प्रबंध		करावी	करा
2 (Hindi)	जाल	जाल	5 (Nepali)	दिन	दिन
	जालक	जाल		दिनहुं	दिन
	जालदार	जाल		दिनका	दिन
	जालरघ्रं	जाल		दिनदिनै	दिन
3 (Hindi)	प्रभु	प्रभु	6 (Kumauni)	लेख	लेख
	प्रभुता	प्रभु		लेखन	लेख
	प्रभुत्व	प्रभु		लेखनु	लेख

Table 5.2. Clusters and their stems

6. Standardization of Kumauni Language

In this section, we develop a mathematical tool for the standardization of Kumauni language. The Hindi stemmer developed by us gives the root of the Hindi word. Naturally the Kumauni dialect which is closest to the associated Hindi stem is a standard one. The question is how to decide the closest word? The technique of *Romanization* makes it much easier. Consider the coding developed in section 2. It is evident from Table 2.3 that the word with more phonetic complexity will have more English alphabets when it is Romanized i.e. when its roman code is calculated. For example, the *Devnagari* consonant '*क*' has code *ka* when it is Romanized, whereas the basic elements '*कौ*','*कं*', '*कः*' have codes *kau*, *kan* and *kaH* respectively. By this idea we select that dialect of the Kumauni word as standard, whose Romanized form contains the least number of English alphabets when compared with the Hindi stem.

The algorithm for the standardization of Kumauni language is given under

6.1 *The Algorithm*

In this section we propose an algorithm for standardizing Kumauni words namely *Standard-KU*. The steps of the algorithm are mentioned as under:
Procedure: Standard-KU
1. Find the stem of the Hindi word by Stem-Hindi
2. Code the dialects of Kumauni language into Roman code.
3. Find that dialect as standard word which is closest to the stem.
4. Decode the standard word into Kumauni language.

6.2 *Application and Results*

Kumauni language is divided into two parts, western Kumauni and eastern Kumauni. These are further divided into six and four dialects respectively [30]. In the following table (Table 6.2.1), we take 10 different forms of the transitive verb '*खा*' (*खाना*) and intransitive verb '*जा*' (*जाना*). These words are taken from [30].

Let's consider the first 10 dialects given in table 6.2.1. The associated Hindi word is *खाना* (*khaanaa*) and its stem obtained by Stem-Hindi is *खा* (*khaa*). The Romanized code of the stem is *khaa* and from Table 6.2.1 it is evident that among the 10 dialects of Kumauni language *khaaM* is closest to *khaa*. Therefore by standard- the decoded form of *khaaM* i.e. *खाँ*, is identified the standard Kumauni dialect. Similarly if we consider second 10 dialects, we get the Kumauni dialect *janoo* and *janoon* as the standard dialect. But since *jaanoo* is spoken in more dialects so we prefer to take it as standard word. The standard Kumauni word is therefore decoded as *जानू*.

S. No.	Dialect Name	Word form	Romanized form	Word form	Romanized form
1	खसपर्जिया	खाँ	khaaM	जानू	jaanoo
2	चौगर्खिया	खाँछों	khaaMchhon	जानू	jaanoo
3	गंगोली	खाँछू	khaaMchhoo	जानू	jaanoo
4	दनपुरिया	खानू	Khaanoo	जानों	jaanon
5	पछाई	खानु	Khaanu	झानु	jhaanu
6	रौ-चाबैंसी	खाँ	khaaM	झानू	jhaanoo
7	कुमय्याँ	खाँछु	khaaMchhu	जानू	jaanoo
8	सोर्याली	खाँछु	khaaMchhu	जानू	jaanoo
9	अस्कोटी	खाँछु	khaaMchhu	जानू	jaanoo
10	सीराली	खाँछु	khaaMchhu	जानू	jaanoo

Table 6.2.1. Words *'खाना'(khaanaa)* and *'जाना' (jaanaa)* in 10 different Kumauni dialects

7. Conclusion and future work

We tested our approach on 100 randomly chosen Hindi words. The aim of stemming is not to find the linguistically correct stem but it should be the root of inflected words [7]. From the list given above 49 stems are linguistically correct (which are shown italicized in Table 5.1) i.e. they belong to the Hindi dictionary. The advantage of the script based stemming is that it is applicable to all languages based on *Devnagari* script. Now the easily available English corpus such as COCA, CLEF, TREC etc. can be utilized for *Devnagari* script. Further the approach can be utilized for the standardization of those regional languages having many dialects. An effort has been made to standardize Kumauni language. The same approach can also be utilized to standardize other variants of Hindi language.

References

1. Appendix: Common Hindi Words available at < http://en.wiktionary.org/wiki/Appendix: Common_Hindi_words>, visited 16 Sep. 2014.

2. Chen, A., Gey, F. C. (2003). Generating statistical Hindi stemmers from parallel texts. ACM Trans. Asian Language Inform. Process. V. 2 (3).

3. Corpus of Contemporary American English (COCA). Available at <http://corpus.byu.edu/coca/>, visited 16 July 2014.

4. John, Dawson. (1974) Suffix removal and word conflation, *ALLC Bulletin* 2 (3) 33-46.

5. Encyclopedia Britannica < http://www.britannica.com/ EBchecked/topic /287731/inflection>

6. Finegan, Edward. (2007). *Language: Its Structure and Use (5th ed.)*. Boston, MA, USA: Thomson Wadsworth. p. 14.

7. Frakes W. B. (1984). *Term conflation for information retrieval, In:* Proceedings of the 7th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. 383-389.

8. Peng, Funchun., Ahmed, Nawaaz., Li, Xin., Lu, Yumao. (2007). Context sensitive stemming for web search, *In:* Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. 639-646

9. Vishal, Gupta. (2014). *Hindi Rule Based Stemmer for Nouns. International Journal of Advanced Research in Computer Science and Software Engineering*. 4 (1) 62-65.

10. Donna, Harman. (1991) How effective is suffixing? *Journal of the American Society for Information Science.* 42: 7-15

11. Jahn., J. (2011). *Vector Optimization*. 2ed Springer.

12. Paik, Jiaul H., et al. (2011) GRAS: An effective and efficient stemming algorithm for information retrieval. *ACM Transactions on Information Systems.* 29(4) Article No. 19

13. Wessel, Kraaij., Renee, Pohlmann. (1996). *Viewing stemming as recall enhancement*. Proceedings of the 19th annual international ACM SIGIR Conference on Research and development in Information Retrieval, 40-48.

14. Robert, Krovetz. (1993). Viewing morphology as an inference process. Proceedings of the 16th Annual International ACM

SIGIR Conference on Research and Development in Information Retrieval. 191-202

15. Lovins, J. B. (1968). Development of a stemming algorithm. Mechanical Translation and Computational Linguistics. 11. 22-31

16. Prasenjit, Majumder., et al. (2007) YASS: Yet another suffix stripper. *ACM Transactions on Information Systems*. 25 (4) Article No. 18

17. James, Mayfield., Paul, McNamee. (2003). Single N-gram stemming. Proceedings of the 26th Aannual International ACM SIGIR Conference on Research and Development in Information Rretrieval. 415-416.

18. Massimo, Melucci., Nicola, Orio. (2007). Design, Implementation, and Evaluation of a Methodology for Automatic Stemmer Generation. *Journal of the American Society for Information Science and Technology.* 58 (5) 673–686.

19. James, Milory. (2001). *Language Ideologies and the Consequence of Standardization. Journal of Sociolinguistics* 5/4. 530-555

20. Mishra Upendra., Chandra Prakash. (2012). *MAULIK: An Effective Stemmer for Hindi Language*. I*nternational Journal on Computer Science and Engineering* (IJCSE), 4 (5) 711-717.

21. Nationalencyklopedin (NE), 2010 estimates, <http://www.ne.se/spr%C3%A5k/v%C3%A4rldens-100-st%C3%B6rsta-spr%C3%A5k-2010>

22. Karanikolas, Nikitas, N. (2013). A methodology for building simple but robust stemmers without language knowledge: overview, data model and ranking algorithm. Proceedings of the 14th International Conference on Computer Systems and Technologies. 284-290.

23. Chris D, Paice. (1990). Another stemmer. ACM SIGIR Forum. 24 (3) 56-61.

24. Pande B. P., Dhami, H. S.(2011). Application of Natural Language Processing Tools in Stemming. *International Journal of Computer Applications*. 27 (6) 14-19.

25. Pande B. P., Tamta, P. , Dhami, H. S. (2014). *A simple algorithm for the problem of suffix stripping*. International Journal of Applied Linguistics.

26. Amaresh, Kumar., Pandey., Tanveer, J., Siddiqui. (2008). *An unsupervised Hindi stemmer with heuristic improvements*. Proceedings of the second workshop on Analytics for noisy unstructured text data, p. 99-105.

27. Porter, M. F. (1980) An algorithm for suffix stripping. *Program* 14: 130-137

28. Sirsat, Sandeep., et al. (2013). A Comparative Study of Truncating Stemming Algorithms. *International Journal of Advanced Scientific and Technical Research*. 3 (2) 411-416.

29. Ramanathan, A., Rao, Durgesh, 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, p. 42-48.

30. Bisht, Sher Singh. (2005). *कुमाउँ हिमालय की बोलियों का सर्वेक्षण.* Indian Publishers' Distributers.

31. Standard language <http://en.wikipedia.org/wiki /Standard_language>

Appendix

Table A.1: Possible combinations of Devanagari consonants and vowel sounds (Mataras)

क	क्	क	का	कि	की	कु	कू	के	कै	को	कौ	कं	कः
	k	ka	kaa	ki	kee	ku	koo	ke	kai	ko	kau	kan	kahh
ख	ख्	ख	खा	खि	खी	खु	खू	खे	खै	खो	खौ	खं	खः
	kh	kha	khaa	khi	khee	khu	khoo	khe	khai	kho	khau	khan	khahh
ग	ग्	ग	गा	गि	गी	गु	गू	गे	गै	गो	गौ	गं	गः
	g	ga	gaa	gi	gee	gu	goo	ge	gai	go	gau	gan	gahh

घ	घ्	घ	घा	घि	घी	घु	घू	घे	घै	घो	घौ	घं	घः
	gh	gha	ghaa	ghi	ghee	ghu	ghoo	ghe	ghai	gho	ghau	ghan	ghahh
ङ	-	ङ	ङा	ङि	ङी	ङु	ङू	ङे	ङै	ङो	ङौ	ङं	ङः
	-	Nga	Ngaa	Ngi	Ngee	Ngu	Ngoo	Nge	Ngai	Ngo	Ngau	Ngan	Ngahh
च	च्	च	चा	चि	ची	चु	चू	चे	चै	चो	चौ	चं	चः
	ch	cha	chaa	chi	chee	chu	choo	che	chai	cho	chau	chan	chahh
छ	छ्	छ	छा	छि	छी	छु	छू	छे	छै	छो	छौ	छं	छः
	chh	chha	chhaa	chhi	chhee	chhu	chhoo	chhe	chhai	chho	chhau	chhan	chhahh
ज	ज्	ज	जा	जि	जी	जु	जू	जे	जै	जो	जौ	जं	जः
	j	ja	jaa	ji	jee	ju	joo	je	jai	jo	jau	jan	jahh
झ	झ्	झ	झा	झि	झी	झु	झू	झे	झै	झो	झौ	झं	झः
	jh	jha	jhaa	jhi	jhee	jhu	jhoo	jhe	jhai	jho	jhau	jhan	jhahh
ञ	ञ् -	ञ	ञा	ञि	ञी	ञु	ञू	ञे	ञै	ञो	ञौ	ञं	ञः
	Nj	Nja	Njaa	Nji	Njee	Nju	Njoo	Nje	Njai	Njo	Njau	Njan	Njahh
ट	ट्	ट	टा	टि	टी	टु	टू	टे	टै	टो	टौ	टं	टः
	T	Ta	Taa	Ti	Tee	Tu	Too	Te	Tai	To	Tau	Tan	Tahh
ठ	ठ्	ठ	ठा	ठि	ठी	ठु	ठू	ठे	ठै	ठो	ठौ	ठं	ठः
	Th	Tha	Thaa	Thi	Thee	Thu	Thoo	The	Thai	Tho	Thau	Than	Thahh
ड	ड्	ड	डा	डि	डी	डु	डू	डे	डै	डो	डौ	डं	डः
	D	Da	Daa	Di	Dee	Du	Doo	De	Dai	Do	Dau	Dan	Dahh
ढ	ढ्	ढ	ढा	ढि	ढी	ढु	ढू	ढे	ढै	ढो	ढौ	ढं	ढः
	Dh	Dha	Dhaa	Dhi	Dhee	Dhu	Dhoo	Dhe	Dhai	Dho	Dhau	Dhan	Dhahh
ण	ण्	ण	णा	णि	णी	णु	णू	णे	णै	णो	णौ	णं	णः
	N	Na	Naa	Ni	Nee	Nu	Noo	Ne	Nai	No	Nau	Nan	Nahh
त	त्	त	ता	ति	ती	तु	तू	ते	तौ	तो	तौ	तं	तः
	t	ta	taa	ti	tee	tu	too	te	tai	to	tau	tan	tahh
थ	थ्	थ	था	थि	थी	थु	थू	थे	थै	थो	थौ	थं	थः

	th	tha	thaa	thi	thee	thu	thoo	the	thai	tho	thau	than	thahh
द	द्	द	दा	दि	दी	दु	दू	दे	दै	दो	दौ	दं	दः
	d	da	daa	di	dee	du	doo	de	dai	do	dau	dan	dahh
ध	ध्	ध	धा	धि	धी	धु	धू	धे	धै	धो	धौ	धं	धः
	dh	dha	dhaa	dhi	dhee	dhu	dhoo	dhe	dhai	dho	dhau	dhan	dhahh
न	न्	न	ना	नि	नी	नु	नू	ने	नै	नो	नौ	नं	नः
	n	na	naa	ni	nee	nu	noo	ne	nai	no	nau	nan	nahh
प	प्	प	पा	पि	पी	पु	पू	पे	पै	पो	पौ	पं	पः
	p	pa	paa	pi	pee	pu	poo	pe	pai	po	pau	pan	pahh
फ	फ्	फ	फा	फि	फी	फु	फू	फे	फै	फो	फौ	फं	फः
	f	fa	faa	fi	fee	fu	foo	fe	fai	fo	fau	fan	fahh
ब	ब्	ब	बा	बि	बी	बु	बू	बे	बै	बो	बौ	बं	बः
	b	ba	baa	bi	bee	bu	boo	be	bai	bo	bau	ban	bahh
भ	भ्	भ	भा	भि	भी	भु	भू	भे	भै	भो	भौ	भं	भः
	bh	bha	bhaa	bhi	bhee	bhu	bhoo	bhe	bhai	bho	bhau	bhan	bhahh
म	म्	म	मा	मि	मी	मु	मू	मे	मै	मो	मौ	मं	मः
	m	ma	maa	mi	mee	mu	moo	me	mai	mo	mau	man	mahh
य	य्	य	या	यि	यी	यु	यू	ये	यै	यो	यौ	यं	यः
	y	ya	yaa	yi	yee	yu	yoo	ye	yai	yo	yau	yan	yahh
र	र्	र	रा	रि	री	रु	रू	रे	रै	रो	रौ	रं	रः
	r	ra	raa	ri	ree	ru	roo	re	rai	ro	rau	ran	rahh
ल	ल्	ल	ला	लि	ली	लु	लू	ले	लै	लो	लौ	लं	लः
	l	la	laa	li	lee	lu	loo	le	lai	lo	lau	lan	lahh
व	व्	व	वा	वि	वी	वु	वू	वे	वै	वो	वौ	वं	वः
	v	va	vaa	vi	vee	vu	voo	ve	vai	vo	vau	van	vahh
श	श्	श	शा	शि	शी	शु	शू	शे	शै	शो	शौ	शं	शः
	sh	sha	shaa	shi	shee	shu	shoo	she	shai	sho	shau	shan	shahh
ष	ष्	ष	षा	षि	षी	षु	षू	षे	षै	षो	षौ	षं	षः
	Sh	Sha	Shaa	Shi	Shee	Shu	Shoo	She	Shai	Sho	Shau	Shan	Shahh
स	स्	स	सा	सि	सी	सु	सू	से	सै	सो	सौ	सं	सः
	s	sa	saa	si	see	su	soo	se	sai	so	sau	san	sahh
ह	ह्	ह	हा	हि	ही	हु	हू	हे	है	हो	हौ	हं	हः
	h	ha	haa	hi	hee	hu	hoo	he	hai	ho	hau	han	hahh
क्ष	क्ष्	क्ष	क्षा	क्षि	क्षी	क्षु	क्षू	क्षे	क्षै	क्षो	क्षौ	क्षं	क्षः
	Ksh	Ksha	Kshaa	Kshi	Kshee	Kshu	Kshoo	Kshe	Kshai	Ksho	Kshau	Kshan	KshaH
त्र	त्र्	त्र	त्रा	त्रि	त्री	त्रु	त्रू	त्रे	त्रै	त्रो	त्रौ	त्रं	त्रः
	tr	tra	traa	tri	tree	tru	troo	tre	trai	tro	trau	tran	trahh
ज्ञ	ज्ञ	ज्ञ	ज्ञा	ज्ञि	ज्ञी	ज्ञु	ज्ञू	ज्ञे	ज्ञै	ज्ञो	ज्ञौ	ज्ञं	ज्ञः
	Gy	Gya	Gyaa	Gyi	Gyee	Gyu	Gyoo	Gye	Gyai	Gyo	Gyau	Gyan	Gyahh