Nepali Stemming Algorithm

Draft Report

Sanjay Subedi

St. Xavier's College

Kathmandu, Nepal

(jangedoo@hotmail.com)

1 October 2011

Stemming in Nepali

Nepali is a morphologically rich language. To understand the constructs of words, one must possess the knowledge of building of words. With the knowledge of word construction process, it will be easier to study various properties of the words. Although much research has been done for stemming in other languages, not much has been done in Nepali.

Words are basically of two types

- Root word (मुल शब्द)
- Derived word (व्युत्पन्न शब्द)

Root words are the words which cannot be further divided. Sometimes they don't give any meaning but are very important part of any language. For example

बाबा, घर, जल (नाम)

म, तँ, ऊ (सर्वनाम)

Root words can be of two types

- Inflectional root: A word without its inflectional endings is called an inflectional root. For example, केटाहरु is inflectional root of केटाहरुले.
- Lexical root: It is the primary lexical unit of the word. For example, केटा is lexical root of केटाहरुले.

An inflectional root is known as a stem.

Derived words are usually generated by one of the following

1. When the root word is combined with affix. In Nepali grammar, prefix is called (उपसर्ग) and suffix is called (प्रत्यय). For example,

   उप + वास = उपवास

   टिप् + ओट = टिपोट
2. When two different words are mixed together (समास).
3. When a word or a part of the word is repeated (दित्वप्रक्रिया).
4. When there are phonetic similarities (सन्धि र आगम).

The most difficult one to decode is the word generated from phonetic similarities. There are no rules defined for the generation of words and is completely dependent on the phonetic similarities between the words involved.

Basically, stemming a word means removing affixes to get to the root word. As it may seem obvious to us, it is difficult to implement it. We have to specify the rules for all possible affixes and the changes they make to the root word. Sometimes the root word is not affected at all and in this case

we can easily separate root word and the affix easily but in most of the case, the root word is changed drastically when introduced to an affix.

In this paper, we introduce an algorithm for stemming Nepali words. This algorithm is based on stripping suffixes and hence deriving the root word. Using this algorithm we can identify Nepali कृत् and taditanta suffixes and get to the root. Since Sanskrit suffixes are also frequently used in Nepali we also need to properly identify those but the words change very drastically after using them, in most of the cases, and we cannot decode it with general algorithm that we are going to propose. So we are going to ignore those types of suffixes. Also, when the suffixes change the structure of the word and the changes are known to be consistent then we will make the rules for them.

Process

Because our algorithm is suffix stripping we will work from the end of the word to the front. First of all we need to build a database of all suffixes along with their associated rules. The rules contain information about the changes that the suffix will make to the root word when applied. In the database the suffixes are stored in ascending order according to their length of characters. Sorting is necessary for the algorithm to work and is explained later.

Unlike English, which has vowels and consonants only, Nepali has two additional constructs called symbols (चिन्ह e.g. ◌्, ◌्र, ◌ृ, ◌ं, ◌ँ) and modifiers (मात्रा). Modifiers are symbolic representation of vowels and are used with consonants only.

आ	◌ा
इ	◌ि
ई	◌ी
उ	◌ु
ऊ	◌ू
ए	◌े
ऐ	◌ै
ओ	◌ो
औ	◌ौ

Table 1: Vowels and their equivalent modifiers.

For this algorithm to work properly and to remove ambiguities and difficulties, we will substitute the modifiers with their equivalent vowels. For example, the word उपवास will be converted to उपअवआसअ and can be read as उ+प+अ+व+आ+स+अ. This conversion is done to all the parameters in the algorithm including the input words and our database. The main reason for this conversion is to find the suffix starting with the letter अ. For example, in case of the word बहुदलीय, it is easy to know that this word if formed by बहुदल + ईय because the suffix, ईय, is just appended to the root word. But in case of words that form after applying the suffixes starting with letter अ, it is not clearly visible. For example, बोलक्कड is बोल्+अक्कड. So unless we convert the word बोलक्कड into

बओलअक्कअडअ, we cannot find the valid suffix in the word. After conversion we can now break it into बओल + अक्कअडअ and then contract it into बोल्+अक्कड.

One thing to note in above conversion is the insertion of the letter अ. Our conversion is not actually correct because प is not प+अ but प्+अ. However, this does not affect the accuracy of the algorithm; instead it simplifies the process of parsing the letters because प् is a single entity but in computer representation it consists of प and ्. So we have to deal with two characters and it introduces great overhead. Although the letter अ has its importance, we will ignore it as a suffix.

Let us take a look at how the algorithm stems the word बहुदलीय. This word is first converted into बअहउदअलईयअ by replacing the modifiers with appropriate vowels. Then from the end, we check if that character is a suffix

Step 1: अ is considered as suffix. We first check if this character is a valid suffix. In our database it is not listed as suffix so we prepend one more character from the rear of the word.

Step 2: यअ is also not a valid suffix. So we again prepend one more character from the rear of the word.

Step 3: ईयअ (ईय only in real world) is a valid suffix so we break the word into two बअहउदअल and ईयअ.

Step 4: The word बअहउदअल is contracted (i.e. replace vowels with modifiers) which results बहुदल, and is checked in the dictionary for its existence. बहुदल is a valid word and is found in the dictionary. Hence, the stemmed form of बहुदलीय is बहुदल + ईय.

Note: We use the exploded form of word e.g. बअहउदअलईयअ *in computation and contracted form e.g.* बहुदल *for dictionary checking and returning output only.*

Now, let us look at a more complex example. Consider the word ऐतिहासिक. Its root word is इतिहास and suffix is इक.

Step 1: The word is exploded to ऐतइहआसइकअ and अ is considered as suffix. Since it is not in suffix database we continue prepending the characters from the rear until इकअ is formed.

Step 2: Break the word into ऐतइहआस and इकअ. Since, ऐतिहास (contracted form of ऐतइहआस) is not in the dictionary, we check if there are any rules associated with this suffix. The suffix इक changes the character इ (should be the first character) of the root word to ऐ in the derived word. So

we reverse the change by replacing ऐ with इ which results in इतिहास and is a valid word in the dictionary.

Step 3: Return इतिहास + इक

For the word दियालो = दियो + आलो, the algorithm actually produces दिय + आलो and then checks if the root word, in this case दिय, is in the dictionary. If it is not then a modifier is appended at the end of this word until the match is found in the dictionary or all modifiers have been appended.

दिया is checked in dictionary => not found

दियु is checked in dictionary => not found

.

.

.

दियो is checked in dictionary => found

Hence दियो is returned as root word. This approach greatly increases accuracy but at a cost of checking every possible (max 8) alternations, due to the modifiers, in the dictionary. However, this approach is more desirable than making rules for individual words and suffixes.

Algorithm

Input: Word to stem
Output: Root word and suffix

1. Find the longest valid suffix
2. If the suffix has any rules associated with it then apply those rules
3. Return root word and suffix

Here is a list of some routines that have been used in our implementation.

contractWord(wordToContract) is a routine that accepts a string and returns its *contracted* form. For example, for the word बअहउदअल, it returns बहुदल. The following rules have been used to contract a word:

- If the current letter is अ then delete that character
- If it is vowel and previous character is neither a vowel nor a modifier then replace the vowel with its equivalent modifier.

explodeWord(wordToExplode) is a routine that accepts a string and replaces the modifiers with vowels and returns it. For example, for the word बहुदल it returns बअहउदअल. Following rules have been used to *explode* a word:

- If current character is consonant and next character is neither a modifier nor a symbol then insert अ
- Else if current character is a modifier then replace it with equivalent vowel.

applyRules(word, suffixRules) is a routine that takes a word and rules for the suffix and applies those rules and returns the resultant string. In our implementation, the rules are comma separated. For example, the suffix याहा has the following rules attached to it: F+◌ा, F*◌ु◌ू, L-य, L-◌ी. Here, F means the first character (either vowel or consonant only) of the word and L means the last character of the word. Similarly, + denotes append, - denotes remove and * denotes replace.

The first rule, F+◌ा, means that we should append ◌ा to the first character of the word.

The second rule, F*◌ु◌ू, means that we should replace the modifier ◌ु of first character with ◌ू.

The third rule, L-य, means that we should remove य from the end of the word.

The fourth rule, L-◌ी, means that we should remove ◌ी from the end of the word.

The operator "+" in our rule is easy to follow since we are just appending the character specified in the rule. However, for the operators "-" and "*", we have to first check if there is the character in word specified in the rule to replace or delete. For example, L-य rule is applied only if there is य at the end of the word. Similarly F*◌ु◌ू is only applied if the first character has modifier ◌ु.

The rules are applied only once i.e. the ***F*** and ***L*** rules are applied one time each (if possible). We can use the ***F*** rule only once and ***L*** rule only once.