A Bottom up Approach to Persian Stemming

Amir Azim Sharifloo
NLP Research Lab,
Department of Electrical &
Computer Engineering,
Shahid Beheshti University,
Tehran, Iran
a.sharifloo@mail.sbu.ac.ir

Mehrnoush Shamsfard
NLP Research Lab,
Department of Electrical &
Computer Engineering,
Shahid Beheshti University,
Tehran, Iran
m-shams@sbu.ac.ir

Abstract

Stemmers have many applications in natural language processing and some fields such as information retrieval. Many algorithms have been proposed for stemming. In this paper, we propose a new algorithm for Persian language. Our algorithm is a bottom up algorithm that is capable to reorganize without changing the implementation. Our experiments show that the proposed algorithm has a suitable result in stemming and flexibility.

1 Introduction

In linguistics, *stem* is a form that unifies the elements in a set of morphologically similar words (Frakes and Yates, 1992), therefore *stemming* is the operation which determines the stem of a given word. In other words, the goal of a stemming algorithm is to reduce variant word forms to a common morphological root, called "stem" (Bacchin et al., 2002).

There are three common approaches that are used in stemming: affix stripping, lookup table and statistical methods (Bento et al., 2005). Affix stripping dependends on the morphological structure of the language. The stem is obtained by removing some morphemes from the one or both sides of the word. Porter algorithm (Porter, 1980) is an example of this kind of algorithms. This stemmer is made up of five steps, during which certain rules are applied to the words and the most common suffixes are removed.

In lookup table approach, each word and its related stem are stored in some kind of structured form. Consequently, for each stored word, we find its stem. However, the approach needs more space. Also, for each new word, table must be updated manually.

In statistical methods, through a process of inference and based on a corpus, rules are formulated regarding word formation. Some of the methodologies adopted are: frequency counts, n-gram (Mayfield and McNamee, 2003), link analysis (Bacchin et al., 2002), and Hidden Markov Models (Melucci and Orio, 2003). This approach does not require any linguistic knowledge whatsoever, being totally independent of the morphological structure of the target language.

In this paper, we propose a new algorithm for stemming in Persian. Our algorithm is rule based and in contrast with affix stripping approach, it is a stem based approach. That means, at first we find possible stems in the word, after that we check which stems are matched with rules.

Our algorithm is bottom up while affix stripping methods are top down. In other words, we try to generate the word using candidate stems of the word which we call *cores* of the word. If the word is generated, the stem is correct. On the other hand, affix stripping approaches try to removing affixes until reaching to any stem in the word.

Some stemming methods have been presented for Persian (Taghva et al., 2005) which use affix stripping approach. Our proposed method tries to reach better precision rather than previous methods. Also, this method tokenizes the word to morphemes which could employ in other morphological methods.

The paper is organized as follows: section 2 presents a brief review of Persian from morphological perspective; in section 3, we describe the proposed

algorithm in details; section 4 is about our experiments.

2 Persian from a Morphological Perspective

Persian is an Indo-European language, spoken and written primarily in Iran, Afghanistan, and a part of Tajikistan. It is written from right to left in the Arabic-like alphabet.

In Persian, verbs involve tense, number and person. For example[1], the verb "می خوانم" (*mi-xän-am:* I read) is a present tense verb consisting of three morphemes. "م" (*am*) is a suffix denoting first single person "خوان" (*xän*) is the present tense root of the verb and "می" (*mi*) is a prefix that expresses continuity.

If a verb has any object pronoun, it can be attached to the end of the verb such as "می‌خوانمش" (*mi-xän-am-aš*: I read it) in which "ش" (*aš*: it) is an object pronoun. Also, negative form of verbs is produced with adding "ن" (*ne*) to the first of them. For example, "نمی‌خوانم" (*ne-mi-xän-am* - I don't read) is the negative form of the verb "می‌خوانم" (*mixänam* - I read). We have gathered 43 rules for verbs, some of them are shown in Table .1.

Table 1. Some rules for verbs in Persian

Rule	example
می + بن مضارع + شناسه مضارع (present person identifier + present root + mi)	می خوانم *(mi-xän-am)* (I read)
بن ماضی + ه + بود + شناسه ماضی (past person identifier + bud +eh + past root)	رفته بودم *(raft-e bud-am)* (I had gone)
ب + بن مضارع (present root + b)	بگذر *(be-gozar)* (Pass)
بن ماضی + ه + شد (shod + h + past root)	خوانده شد *(xand-e šod)* (it was read)

Nouns are more challengeable than others in Persian. We have gathered many rules for nouns that in following, we describe one of them. The plural forms of nouns are formed by adding the suffixes (ها, ان, ات, ون, ین). "ها" (*hä*) is used for all words. "ان" (*än*) is used for humans, animals and every thing that is alive. Also, "ات, ون, ین" (*ät ,un , in*) is used for some words borrowed from Arabic and some Persian words. We have another kind of plural form in Persian that is called *Mokassar* which is a derivational plural form (irregulars in Persian). Some examples of plural form are shown in Table 2.

Also, there are some orthographic rules which show the effects of joining affixes to the word. For example, consider that we have two parts of a word: A and B for joining as BA (Consider, Persian is written right to left). If the last letter of A and the first letter of B are "ا"(*ä*), one letter "ی" (*y*) is added between them. Assume A is "دانا" (*dänä* - wise) and B is "ان"(*än*), the joining result is "دانایان" (*dänä-yän:* wise people).

Table 2. Some kinds of plural form in Persian

Joining	*Result noun*
کشور + ها *(hä + kešvar)* (hä + country)	کشورها *(kešvar-hä)* (countries)
درخت + ان *(hä + deraxt)* (hä + tree)	درختان *(deraxt-än)* (trees)
کتب(Mokassar form) *(kotob)* (books)	کتب *(kotob)* (books)
آقا + ی + ان *(än + y + äghä)* (än + y + mister)	آقایان *(äghä-yän)* (men)

3 The Proposed Algorithm

Our algorithm is rule based and bottom up. At first, it tries to find substrings of the word that are stems or morphemes which are derived from any stem, we call them *cores*. After that, it joins each of cores with other elements of word for generating that word according to available rules. Finally, each core with at least one correct generation is a correct core and its stem is correct stem of the word. The algorithm includes three phases: 1. Substring tagging 2. Rule matching 3. Anti rule matching (Figure 1).

[1] Through the paper, we show Persian examples by their written form in Persian alphabet between "" followed by (*their pronunciation*: translation).

In substring tagging phase, we extract morphological information for all possible substrings of the word. At the end of this phase, we know which substrings of the word are morphemes and which ones are not. Also, we know clusters that each morpheme is their member. We use clusters for rule matching phase. Accordingly, we know cores in the word before beginning the second phase. We describe substring tagging details in section 3.1.

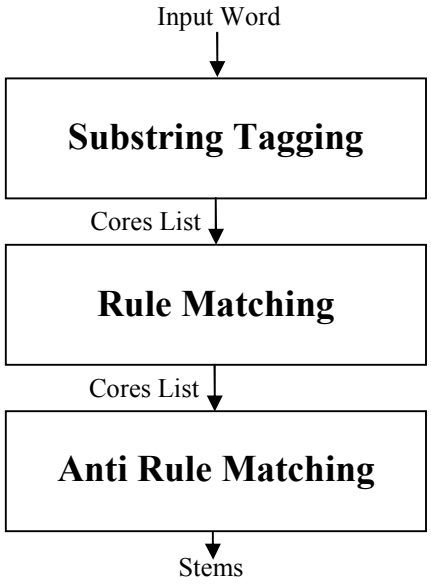

Figure1. Three phases of the proposed algorithm.

In rule matching phase, for each core that has been known in previous phase, we extract related rules. For example, "خوان" (*xän*) is one core of the word "می‌خوانم" (*mi-xän-am:* I read) and "بن مضارع" (*bone mozäre:* present root) is one of clusters that "خوان" (*xän)* is its member. Also, "م" (*am*) is a member of cluster "شناسه مضارع" (*šenase mozäre:* present person identifier) and "می" (*mi*) is a member of cluster "می" (*mi*). We have a rule in rules repository as:

(می + بن مضارع + شناسه مضارع)
(present person identifier + present root + mi)

where it is matched with the word "می‌خوانم" (mi-xän-am: I read). Therefore, we find a matched rule for "خوان" (xän). At the end of second phase, each core that has extracted any possible rule for the word, remains in cores list and other cores are removed from it.

In anti-rule matching phase, we extract anti rules from anti rules repository for each core in the list. Each core which has any matched anti rule with the word morphemes, is removed from the cores list. At the end of the third phase, each stem of any core in the cores list is the correct stem for the word.

3.1 Substring Tagging

Every word with length N has $N*(N+1)/2$ substrings. Therefore, we need $N*(N+1)/2$ string matching for finding them in morphemes repository. We employ a Trie tree for storing morphemes and present an algorithm for retrieving morphological information from it that reduces the number of string matching. This algorithm needs $N(N+1)/2$ character matching (instead of string matching) at most. A simplified part of tree is shown in Figure 2.

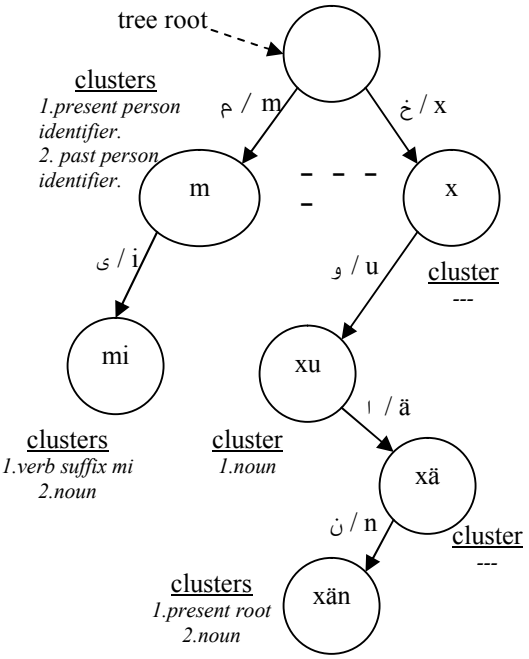

Figure 2. A simplified part of Trie tree that is used for storing morphological information.

The algorithm is described in the following:

We initiate N pointers (N is the word length) that they point to the tree root. Also, we use a counter C that is an index on the word. At first, C's value is one that means its related letter is first letter of the word. At the end of each step, C is increased by one. Therefore, in each step, C points to one letter of the word that we call this letter L.

At first step, first pointer P1 finds one edge between root edges that its letter is equal with L. P1 goes down on tree by that edge. Here, P1 extract morphological information from its new position (a node of the tree) and fills morphological information for substring (1, 2).

At the second step, L is the second letter of the word, second pointer P2 finds one edge between root edges that its letter is equal with L. P2 goes down on tree by that edge, extract morphological information from its new position (a node of the tree) and fills morphological information for substring (2, 3). Also, P1 goes down on tree by an edge contained L, from its position that it is one of root children and fills morphological information for substring (1, 3).At third step, L is third letter of the word. Third pointer P3 starts from root and goes down on tree by an edge that its letter is equal with L and fills morphological information for substring (3, 4). P1, P2 repeat this work from their positions and fill morphological information for substring (1, 4) and (2, 4) respectively.

Next steps are done like these steps. Finally, we have obtained morphological information for all substrings of the word. Also, if one pointer could not find any edge with value L, it is blocked until the end of algorithm. Figure 3 shows pseudo code of this algorithm.

```
 Word: string;
P: array of pointer with word.length size;

for C = 1  to  word.length do
{
   for i = 1  to  C do
   {
      If (P[i] <> Blocked)
      {
        edge = find_edge( P[i] , L );
        // find_edge finds a edge from its position
        // in tree that its letter is equal with L.
        if ( edge <> null )
        {
          GoDown(P[i],edge);
          substring(i, C).mInfo = P[i]-> mInfo;
          // mInfo is morphological Information
        }
         else P[i] = Blocked;
      }
   }
}
```

Figure 3. The used algorithm for obtaining morphological information from Trie tree.

3.2 Rule Matching

We use many rules to generate correct words by morphemes in Persian. We store these rules in rules repository. Some gathered rules are shown in Table 3.

Table 3. Some gathered rules that we use.

Rule
ماضی ساده → بن ماضی + شناسه ماضی (past person identifier + past root → simple past)
امر → ب + بن مضارع (present root + b → imperative)
اسم جمع → اسم + ها (hä + noun → plural noun)
اسم جاندار جمع → اسم جاندار + ان (än + alive noun → alive plural noun)

Each rule is a sequence of clusters. A cluster represents a set of morphemes that affects role of them in the word. In other words, each morpheme could be applied as one or more roles for generating words. So, each role can be a cluster membership. For example, in English, "book" is a verb and a noun. But, As a noun, it has a plural form (books) and as a verb, it has a past form (booked).

Similarly, in Persian, the word "مرد" (mord: die) is a verb root and "مردند" (mord-and: They died) is a verb, too. Also, "مرد" (mard: man) is a noun and "مردها" (mard-hä: men) is one of its plural forms. In consequence, we put "مرد" in both of cluster "اسم" (esm: noun) and "بن ماضی" (bone mäzi: past root). We create a new cluster when a rule needs it and that cluster is not in clusters repository.

As we discussed about it, in Persian, we have several suffixes for plural form that every one is used for a set of nouns. The suffix "ها" (*hä*) is used for every noun and the suffix "ان" (*än*) is special for everything that is alive. Other suffixes are ap-

plied for some words borrowed from Arabic and some Persian words. A noun such as "پسر" (*pesar:* boy) has several plural forms (e.g. "پسرها"/*pesar-hä*, "پسران"/*pesar-än*). Therefore, we employ clusters for organizing this situation. For example, we put the morpheme "پسر" (*pesar:* boy) in cluster "اسم" (*esm:* noun) and "جاندار" (*jändär:* alive). Also, we have two rules in rules repository:

"اسم" + "ها"
(hä + noun)

and

"جاندار" + "ان"
(än + alive)

The morpheme "پسر" (*pesar:* boy) is a member of both clusters "اسم" (*esm:* noun) and "جاندار" (*jändär:* alive). Accordingly, these words "پسرها" (*pesar-hä:* boys) and "پسران" (*pesar-än:* boys) are correct form and their stem is "پسر" (*pesar:* boy). But about the morpheme "کتاب" (*ketäb:* book), it is a noun and a member of cluster "اسم" (*esm:* noun) but it is not a member of cluster "جاندار" (*jändär:* alive). So, "کتابها" (*ketäb-hä:* books) is a correct form and its stem is "کتاب" (*ketäb:* book). In contrast, "کتابان" (*ketäb-än*) is a wrong form and "کتاب" (*ketäb:* book) is not its stem. Also, we organize suffixes in similar cluster such as cluster "شناسه مضارع" (*šenase mozäre:* present person identifier), "حرف نفی فعل" (*harfe nafye fel*). Table 4 shows some clusters.

Table 4. Some clusters that we use.

Cluster	Cluster
شناسه مضارع (present person identifier)	بن مضارع (present root)
پسوند جمع ها (plural suffix hä)	بن ماضی (past root)
پسوند جمع ان (plural suffix än)	اسم (noun)

At the end of this phase, each core must have a rule that it can generate the word. Otherwise, it is removed from cores list.

3.3 Anti Rule Matching

This phase is similar previous phase with a small difference. Like previous phase, we have a rules repository, but these rules are not applied in Persian. In fact, these rules are exceptions of previous phase rules. For example, we have a rule in rules repository:

(اسم جاندار + ان)
(än + alive noun)

On the other hand, there is an exception for this rule. Every noun with the final letter "ه" (he) can not use this rule. For example, "پرنده" (parand-e: bird) is a kind of animals with the final letter "ه" (he) and the word "پرندهان" (parand-e-än) is a wrong word in Persian. We call these exceptional rules "Anti rules".

The details of this phase: Each core from cores list retrieves the anti rules that they involve it. After that, each retrieved anti rule is checked with the morphemes in the word for possibility of word generation. Until now, all things were similar previous phase, but the difference is here. If there is any anti rule related to a rule of any core, that rule is removed from candidate rule list of that core. At the end of this phase, each core must have at least one rule that it can generate the word. Otherwise, it is removed from cores list. Finally, remained cores in cores list have correct stems of the word.

We have gathered a set of anti rules in a repository that each anti rule is related to a rule in rule repository. Some of these anti rules are shown in Table 5.

Table 5. Some gathered anti rules that we use.

Anti Rule
(اسم جاندارمنتهی به ه + ان) (an + alive noun ended with h)
(اسم منتهی به ا، و، ه، ی + ات) (at + noun ended with *ä,u,h, y*)

4 Experiments and Results

The most primitive method for assessing the performance of a stemmer is to examine its behavior when applied to samples of words - especially words which have already been arranged into 'conflation groups'. This way, specific errors (e.g., fail-

ing to merge "maintained" with "maintenance", or wrongly merging "experiment" with "experience") can be identified, and the rules adjusted accordingly.

We evaluated the proposed algorithm with a limited corpus of *Hamshahri* newspaper. We started with 252 rules and 20 anti rules. The algorithm retrieved 90.1 % of word stems correctly. The failed words are related to absence of some rules in rule repository or stems in Trie tree. Some of words in the corpus are shown in Table 6.

Table 6. Some of words in *Hamshahri* newspaper corpus

Stem	Word
ماجرا *(mäjarä)* (event)	ماجرای *(mäjarä-ye)* (event of)
رسم *(rasm)* (custom)	رسمها *(rasm-hä)* (customs)
پدیده *(padide)* (phenomenon)	پدیده های *(padid-e-hä-ye)* (phenomenons of)
بودن *(bud)* (to be)	بودند *(bud-and)* (They were)
ساعت *(säat)* (watch)	ساعتهایشان *(säat-hä-ye-šän)* (watch)
کشیدن *(kešidan)* (to draw)	بکشند *(be-keša-and)*
آخر *(äxar)* (end)	آخرین *(äxar-in)* (last)
رفتن *(raftan)* (going)	نرفته بودند *(na-raft-e budand)* (They had not gone)
سال *(säl)* (year)	امسال *(em-säl)* (this year)
مطالعه *(motäle'e)* (study)	مطالعات *(motäle-at)* (studies)
منطقه *(mantaghe)* (area)	مناطق *(manätegh)* (areas)

One of words could not be handle with our algorithm is "جابجا" (*jä-be-jä* - exchange). We discovered related rule for that and added it to rules repository. Therefore, if we evaluate the algorithm, the result will be better. Rules repository evolves and the algorithm result will be better without any change of program and code compilation.

5 Conclusion

In this paper, we proposed a bottom up method to stem Persian words. The main purpose of this method is high precision stemming based on morphological rules. The experiments show that it has suitable results in stemming and presents possibility of evolution easily.

References

Bento, Cardoso and Dias. 2005. *Progress in Artificial Intellegence*, 12th Portuguese Conference on Artificial Intelligence, pages 693 – 701.

Chris Paice. 1996. *Method for Evaluation of Stemming Algorithms Based on Error Counting*. JASIS , pages 632-649.

Frakes and Yates. 1992. *Information Retrieval: Data Structures and Algorithms*. Prentice Hall, NJ.

Mayfield and McNamee. 2003. *Single N-gram Stemming*. In Proceedings of the 26th annual international ACM SIGIR conference on Research and development in information. retrieval, pages 415-416.

Melucci and Orio. 2003. *A Novel Method for Stemmer Generation Based on Hidden Markov Models*. In Proceedings of Conference on Information and Knowledge Management (CIKM03), pages 131-138.

Michela Bacchin, Nicola Ferro, and Massimo Melucci. 2002. *Experiments to evaluate a statistical stemming algorithm*. Working Notes for CLEF 2002, pages 161-168.

Michela Bacchin, Nicola Ferro, and Massimo Melucci. 2002. *The Effectiveness of a Graph-Based Algorithm for Stemming*. Springer-Verlag Berlin Heidelberg. pages 117–128.

Porter. *An Algorithm for Suffix Stripping*. 1980. Program. pages 130-137.

Taghva, Beckley and Sadeh. 2005. *A stemming algorithm for the Farsi language*. IEEE ITCC 2005, pages 158 - 162.