Supervised Lexical Acquisition for Persian from a Web Corpus

Nick Pendar
Iowa State University
355 Ross Hall
Ames, IA USA 50011
pendar@iastate.edu

Serge Sharoff
University of Leeds
B05, Michael Sadler Building
Leeds, UK LS2 9JT
s.sharoff@leeds.ac.uk

Abstract

This paper reports on the compilation of a large Persian Web corpus and the cyclic supervised development of a lexicon and lemmatizer. We discuss the strategies adopted in compiling the corpus as well as some of the challenges in processing and tokenizing it. We also present the word patterns developed for the lemmatizer and the algorithms designed for the supervised lexical acquisition.

1 Introduction

One major bottleneck in computational and data-driven linguistic studies on Persian is the lack of large-scale and readily available annotated corpora and tools. The main goal of our larger project is to remedy this situation. This paper reports on the first phase of our main project, the goal of which is creating a set of open-domain computational linguistic resources for Persian. These resources, once complete, will include a 21-million word Web corpus of Persian texts (already compiled), plus a lemmatizer (developed), and POS tagger distributed as a Python module named `persian.py`.

Our Web corpus is designed to be a random snapshot of the Persian Internet. Its main advantage over traditional corpora (such as those based on newspapers) is that it reflects a large variety of uses of modern Persian and can be freely distributed in the form of a list of URLs. The completed stages of the project, so far, include the compilation of the web corpus and development of a lexicon-based lemmatizer, which involved the supervised acquisition of an appropriate lexicon.

The next section provides some background information and literature review. Section 3 provides a review of the corpus compilation procedure. Section 4 discusses some of the challenges of the tokenization of Persian texts. Section 5 presents our approach to developing a lemmatizer and lexicon, and section 6 presents the results.

2 Background and Motivation

Nouns, verbs, adjectives and adverbs in Persian inflect, which gives rise to a large number of possible forms for each lemma. NLP applications that rely on vector space models of language, such as information retrieval (IR) or text categorization (TC) systems, will find this situation problematic because of the sheer number of features that they have to deal with, as well as the high level of interdependence among text features. Despite the fact that stemming results in only small improvements in IR applications in large English documents (Krovetz 1993), Taghva et al. (2005) and Mokhtaripour & Jahanpour (2006) reported significant improvements in precision and recall when they used a stemmer for Persian IR systems. However, all the available stemmers for Persian are based on Porter's algorithm, which has been shown to introduce errors (Krovetz 1993). This is especially true in Persian, which has a much higher level of inflection increasing the chances of error. In addition, when it comes to verbs, stemming is not always the best solution because many Persian verbs, like English phrasal verbs, comprise a particle and a light verb. For example, the verb

برگرفتن /bær-gereft-æn/, "to include", "to cover"

is made up of the particle بر (/bær/, "over") and the light verb (/gereft-æn/, "to take", "to get"). Stemming this verb will result in گرفت (/gereft/, "s/he took", "s/he got"), which is a different verb.

Similarly, stemming many verbs will result in stems that are indistinguishable from other homographic words with very different meanings. This means that instead of increasing the words' information content, a stemmer frequently reduces it by making words more ambiguous than they actually are. For example, stemming the verb

دویدن /dæv-id-æn/, "to run"

results in the word دو (/do/), which is ambiguous between the verb "run" and the number "two".

Since purely form-based stemming introduces errors, we need to couple our stemmer with a lexicon to avoid spurious analyses. Also, we need to allow our system to not simply stem Persian verbs, but produce their infinitive form including any preceding particles in order to keep the verbs distinct from other words that are homographic to the verbs' stems. We refer to such a system as a *lemmatizer* here. One feature of `persian.py` module is that, unlike most other systems, it is Unicode-enabled and operates directly on UTF-8 encoded Persian script. Most other systems rely on Romanizing the text before processing and encoding it back in Persian script after processing. Our system eliminates these two passes on the input and output strings.

A lemmatizer can also infer some information about the words' parts of speech by just looking at their morphology. Therefore if we allow our lemmatizer to make educated guesses on the words' POSes, then we can build a lexicon that contains the parts of speech of its entries. Such a lexicon will also be useful for POS tagging and parsing applications. The lexicon that we compiled in this project was developed with the above goals and concerns in mind. Of course, the lexicon needs to be manually edited to remove errors, but the number of words to be manually processed has decreased substantially.

3 Corpus Compilation

Given the lack of large representative Persian corpora, we compiled a corpus from the Web, following the procedure presented in Sharoff (2006). The procedure involved the steps described in the following paragraphs.

First we constructed a list of frequent and topic-neutral word forms such as:

چشم /čæšm/, "eye"
نشسته /nešæst-e/, "s/he is seated"
آتش /ataš/, "fire"
شوهرش /šohær-æš/, "her husband".

Typically, such lists are composed of translations of English forms used for constructing the English Internet Corpus (Sharoff 2006). However, we had to make some extra considerations for making the initial word list for Persian. Many common words in Persian are borrowed from Arabic, which uses the same basic scripts. In order to avoid the risk of conflating the corpus with Arabic pages, we first made a list of the most frequent words of Persian using the Web interface of the Persian Language Database hosted by the Institute for Humanities and Cultural Studies in Tehran.[1] We then built a sublist of the content words that were unambiguously Persian.

The second step was making a list of 10,000 three-word queries from the word list. Third, we fed these queries to Google and collected the URLs of the top ten pages returned for each query. The next step was to download the available pages and sort them based on their encodings. The last step was cleaning the pages from boilerplate and navigation frames.

Persian pages use several different encodings, such as UTF-8, Windows-1256, ISO-8859-6, or MacFarsi. In some cases the encoding is not set explicitly or set incorrectly to Latin1. Since almost 75% of the retrieved pages were in correct UTF-8, only these pages were used to build the corpus. For removal of boilerplate and navigation frames we used Aidan Finn's heuristics: Connected text, which is relevant for linguistic purposes (such as the main body of a newspaper article or the content of a blog message), typically uses far fewer tags per token than navigation to other news sections or blog entries, links to related articles or standard "Search" or "Other languages" links, which do not constitute connected text and are not relevant for corpus compilation (Finn 2002; Sharoff 2006). The compiled corpus contains 21,626,774 words according to our tokenization procedure (see the next section).

[1] http://pldb.ihcs.ac.ir

4 Tokenization

One challenge in tokenizing Persian texts is the correct recognition of complex tokens and detached inflectional morphemes (see Megerdoomian 2004, 2006 for a discussion). These problems are caused by two sources: (i) orthography, and (ii) style.

The letters in Persian alphabet, the Perso-Arabic right-to-left writing system, often have more than one form. Each form is licensed based on the letter's position within the word and/or its surrounding letters. Letters may or may not attach to the letter that immediately follows them. For example, the letter ع (/'eyn/) attaches to its following letter, while the letter د (/dal/) does not. Letters that do attach to their surrounding letters then have up to four different forms: (i) *attaching initial form* (e.g., عـ), (ii) *attaching middle form* (e.g., ـعـ), (iii) *attaching final form* (e.g., ـع), and (iv) *non-attaching final form* (e.g., ع). The attaching initial form appears in the word-initial position and always attaches to the following letter e.g.,

علمی /'elm-i/, "scientific".

The attaching middle form occurs in the word-medial position after letters that can attach to their following letters. For example, the letter ع appears as ـعـ after م (/mim/), e.g.,

معبد /mæ'bæd/, "temple"

but the same letter appears as ـع after د e.g.,

دعوت /dæ'væt/, "invitation".

The attaching final form appears word-finally after letters that have an attaching form, e.g.,

بلع /bæl'/, "swallow",

and the non-attaching final form appears word-finally after letters that do not have an attaching form, e.g.,

ممنوع /mæmnu'/, "forbidden".

Newer Persian writing styles require certain inflectional prefixes (e.g., the habitual marker می /mi/) to be detached from the verb, and the only way most word processors (especially those developed outside Iran) allow the letter ی to appear in its non-attaching form is by inserting a space after it. This means that the verb

می رود /mi-ræv-æd/, "he goes"

can only be written according to the style if it is written as two tokens separated by whitespace. The same verb, however, can be written as a single token میرود according to older styles. This is obviously a source of inconsistency.

The other source of inconsistency also relates to letter types. Because some letters do not attach to their following letter, sometimes language users type two words together without an intervening space because the words will still be easily recognizable as separate by human readers. But this cannot be done with all words. For example, to say "The man came" one may write

مرد آمد /mærd amæd/ (with the intervening space)

or

مردآمد (without the intervening space),

which makes the latter form a complex token. However, it is not possible to write "A man came" without the intervening space.

مردی آمد /mærd-i amæd/

is correct while

مردیآمد

is not. Orthographic and stylistic constraints of this sort, as well as, inconsistency on writers' part render tokenizing Persian text a challenge.

In this phase of the project, we tackled the issue of detached morphemes. First, we simply tokenized the corpus based on whitespace and non-alphanumeric characters. This resulted in 289,986 word types, from which we identified seven high-frequency detached prefixes and 56 high-frequency detached suffixes. These include simple or complex prefixes, such as می (/mi/, habitual marker), درمی (/dær-mi/, a particle and the habitual marker, and درنمی (/dær-ne-mi/, a particle, the negation marker, and the habitual marker), as well as simple

or complex suffixes, such as ها (/ha/, plural marker), هایی (/ha-yi/, plural plus indefinite marker), and هایشان (/ha-yešan/, plural plus third person plural possessive clitic). The orphaned items found also included derivational morphemes such as مند (/mænd/, adjectivizing suffix), as نا (/na/, negative prefix).

We left complex tokens to a later stage when we have a lexicon to identify the adjoining words. For this project, only a handful of very common complex tokens were dealt with.

Prior to tokenization, some preprocessing needs to be performed. The letters ی and ک sometimes appear as ي and ك which count as different characters. Also sometimes language users use a dash character known as *kashideh* (ـ) in order to lengthen words. For example, the word

باد /bad/, wind

can be written as بـاد or بـــاد etc. Before tokenization, all ي and ك characters are replaced with ی and ک respectively, and all *kashidehs* as well as diacritics are removed from the input text.

Tokenization takes place in two passes. In the first pass, the input string is turned into a list of tokens separated by word boundary characters. In the second pass, non-Persian tokens and those containing numeric characters are removed from the token list while orphan prefixes and suffixes are attached to their correct bases.

Tokenization according to this scheme over the entire corpus yielded 21,626,774 tokens, and 318,067 word types.[2] Out of this, 238,752 types were repeated four times or less throughout the corpus. Most of these infrequent items were misspellings or run-together words (complex items). In lexical acquisition, we excluded items with frequencies less than five, which means we built our lexicon based on 79,315 word types, the top 300 most frequent words of which account for almost half the tokens in the entire corpus. The 79,315 word types with frequencies of five and higher account for 21,267,403 or 98.34% of all the tokens observed.

[2]A *token* is an instance of a *word type* in a corpus; and a *word type* is a member of the set of words in a corpus.

5 Lemmatization and Lexical Acquisition

We took a cyclic approach to developing the lexicon and the lemmatizer. First we manually built a seed lexicon containing 81 of the most frequent irregular nouns together with their plural forms, 101 of the most frequent irregular verbs (observed in the corpus and complemented by a list from Bateni (1969) together with their past forms, and 3,123 other frequent headwords in Persian including pronouns, numerals, common and proper nouns, regular verbs including Persian's prevalent light verbs, and several other headwords belonging to other parts of speech. Each headword is assigned one or more part-of-speech tags from a small tagset, which is a modified and abridged version of the Penn Treebank POS tags (Marcus et al., 1993) (see Table 3 in the Appendix). We then built a lemmatizer based on our knowledge of Persian morphology and what was observed empirically during the development of the seed lexicon. Next, we used the lemmatizer and the seed lexicon to build the larger lexicon, which after manual post-editing will be used as the lemmatizer's base lexicon.

5.1 Lemmatizer Patterns

As mentioned above, the lexical acquisition algorithm runs on a lemmatization back end, which in turn is based on 11 regular expression patterns as follows:

Verb Patterns

Four patterns were defined for verbs, three for tensed verbs and one for infinitives. The following rewrite rules represent the regular expressions defined, in which the items in parentheses are optional and | represents disjunction.

(1) VB → (PRE) (NEG) (HAB) VSTEM (CAUS PAST | PAST_CAUS) SUBJ (OBJ)
(2) VB → (V_PRE) (SUB) VSTEM (CAUS) SUBJ (OBJ)
(3) VB → (V_PRE) (NEG) (HAB) VSTEM (CAUS PAST | PAST_CAUS) PASTP SUBJ
(4) VB → (V_PRE) (NEG) VSTEM (CAUS PAST | PAST_CAUS) INF

In the above patterns PRE stands for one the 11 common verbal particles, which together with light verbs create new derivations. These particles are

باز /baz/, بر /bær/, پا /pa/, پیش /piš/, خود /xod/, در /dær/, سر /sær/, فرا /færa/, فرو /foru/, نا /na/, and وا /va/. NEG is the negation particle ن /næ/. HAB is the habitual marker می /mi/. VSTEM is the verb stem, which non-greedily matches one or more letters. CAUS is the causative marker ان /an/. PAST stands for the regular past tense marker ید /id/. PAST_CAUS matches cases in which one morpheme اند /and/ stands for both causative and past. SUBJ is one of the subject markers م /æm/, ی /i/, یم /im/, ید /id/, ین /in/, ن /æn/, ند /ænd/. OBJ stands for the object markers م /æm/, ت /æt/, ش /æš/, مان /eman/, تان /etan/, شان /ešan/, مون /mun/, تون /tun/, شون /šun/. The markers include colloquially used markers as well as those common in formal language. SUB matches the subjunctive marker ب /be/. PASTP is for the past participle marker ه /e/, and INF stands for the infinitive marker ن /æn/.

Pattern (1) matches such indicative verbs as

رفتم /ræft-æm/,
"I went",
برنمیخوردند /bær-ne-mi-xord-ænd/,
"they did not collide", or
نمیپراندمش /ne-mi-pær-and-æm-eš/,
"I didn't use to make it fly".

Pattern (2) matches imperative or subjunctive verbs like

بخورید /be-xor-id/,
"you (pl.) eat" or
بپرانیمش /be-pær-an-im-æš/,
"we should make it fly".

Pattern (3) matches past participles like

پریده ام /pær-id-e-æm/,
"I have jumped".

And finally, pattern (4) matches infinitive forms such as

گفتن /goft-æn/,
"to say" or
نگفتن /næ-goft-æn/,
"not to say".

Notice that in patterns (1)-(3) SUBJ is designated as obligatory. This is in fact inaccurate. The third person singular verbs in Persian do not have an overt subject marker. However, without that, most word types that matched NN would also match VB. Making subject markers obligatory is strictly a work-around for this problem.

VSTEM extracted with the above patterns may be a present stem or an irregular past stem. Before generating the infinitive form of the verb, the lemmatizer checks VSTEM against the list of irregular verbs. If VSTEM is an irregular past form, then only the letter ن /n/ is added to it. If on the other hand, VSTEM is a present form, then its corresponding past form (regular or irregular) is used before ن /n/ to generate the infinitive form of the verb being lemmatized.

Noun Patterns

We have also defined four noun patterns as follows:

(5) NN → NSTEM (PL) (POSS | Y) (RA)
(6) NN → $NSTEM_h$ (PL_h) (POSS | Y) (RA)
(7) NN → $NSTEM_v$ (PL_v) (POSS | Y) (RA)
(8) NN → ا LL ا L

NSTEM matches the noun stem, which could be any sequence of two or more letters. $NSTEM_h$ matches a noun stem that ends in ه /e/, and NSTEMv matches nouns that end in ا /a/ or و /u/. These patterns are also non-greedy. PL matches the plural markers ها /ha/, های /ha-ye/ ان /an/, این /in/. PL_h stands for the plural markers ات /at/, and گان /gan/. PL_v stands for the plural markers یان /yan/ and یون /yun/. POSS matches the possessive markers م /æm/, ت /æt/, ش /æš/ مان /man/ تان /tan/ شان /šan/, مون /mun/, تون /tun/, شون /šun/, یم /yæm/, یت /yæt/, یش /yæš/, یمان /yeman/, یتان /yetan/, یشان /yešan/, یمون /yemun/, یتون /yetun/, یشون /yešun/. Y matches the indefinite marker ی /i/, یی /yi/, or ئی /'i/. RA matches the oblique/topic marker را /ra/, رو /ro/, or و /o/. L in the last pattern stands for any letter. Patterns (5)-(7) match Persian common nouns; whereas, pattern (8) matches plural forms for certain nouns borrowed from Arabic.

Adjective Pattern

Adjectives *per se* do not follow any special patterns in Persian unless they are derivations from other parts of speech; however, they are easily recognizable when they appear in the comparative or

superlative form. We define an adjective stem, JJSTEM, just like a NSTEM, that is, as any sequence of two or more Persian letters. The comparative marker is تر /tær/, and the superlative marker is ترین /tærin/. Therefore, the adjective pattern is defined as in (9) below:

(9) JJ → JJSTEM COMP | SUP

Again in order to prevent almost all word types to match this pattern, COMP | SUP is designated as obligatory.

Adverb Pattern

At present, our lemmatizer recognizes any sequence of three letters or more followed by ا /a/, وار /var/, انه /ane/, and گانه /gane/, (RBSUF), which are derivational suffixes, as adverbs. (10) represents this pattern.

(10) RB → JJSTEM RBSUF

Ordinal Numbers

Ordinal numbers in Persian are made with cardinal numbers plus م /om/, می /mi/, or مین /omin/ (ODSUF). (11) shows this pattern.

(11) OD → CD ODSUF

At this point, the irregular ordinal numbers, such as اولین /ævval-in/, "first" and سومین /sevvomin/, "third" are not lemmatized and are simply tagged as OD. This will be fixed with a simple table lookup in the future.

5.2 The Lemmatizer Algorithm

Table 1 below shows the pseudo-code of the lemmatization algorithm used for this project. Given a word *w*, the lemmatizer will return *w* as lemma and *pos(w)* as the part of speech of *w* iff *w* is present in the seed lexicon. Otherwise, it will try all the patterns in (1)-(11) to find all the patterns that may match. If a VB pattern matches, the algorithm will try to build the verb's infinitival form together with its particle PRE if it is present. The infinitive form of verbs in Persian is made with the verb stem in the past tense plus the infinitive marker ن /æn/. Therefore, to build an infinitive, it is necessary to check to see if the verb is irregular, in which case, the irregular past form is used. The constructed infinitival form is then returned as lemma *l*, and *pos(l)* is set to VB.

If an OD pattern matches, then the lemmatizer checks to see if the corresponding CD exists in the seed lexicon. If it does not, that pattern is rejected; otherwise, the corresponding cardinal number is returned as lemma, and *pos(l)* is set to CD.

If any other pattern matches, then the hypothetical word stem is extracted and returned as *l*. The *pos(l)* is set to the POS associated with the pattern only if at least one prefix and suffix in the pattern has matched with some part of *w*. This is to avoid spurious POS assignments due to a word matching only with the stem patterns.

If the lemma hypothesized according to any of the patterns exists in the seed lexicon and the POS associated with that pattern belongs to the list of that lemma's POSes, then that analysis is returned as the sole analysis without consideration of further patterns; otherwise, a list of analyses is created. The lemmatizer then ranks the analyses from the one with the shortest lemma to the one with the longest. This list of lemmas, together with their possibly assigned POSes, prefixes, and suffixes, is then returned.

Input: *w,* the word to lemmatize; *P,* the regular expression patterns for inflected categories; L_s, the seed lexicon
Output: *A,* a list of candidate analyses including lemma of *w*, *pos(w)*, *prefixes(w)* and *suffixes(w)*

```
Lemmatize(w, P, Ls):
 A = [ ]
 if w in Ls:
  return [(w, pos(w), [ ], [ ])]
 else:
  for p in P:
   if w matches p:
    l = extract_or_build_lemma(w, p)
    if l in Ls and hypothetical_pos(l) in pos(l):
     A = [(l, pos(l), pre(w), suf(w))]
     return A
    else:
     add (l, hypothetical_pos(l), prefixes(w),
          suffixes(w)) to A
  if not A:
   A = [(w,[ ], [ ], [ ])]
   return A
 remove duplicate analyses from A
 sort A based on length(l)
 return A
```

Table 1: Pseudo-code of the lemmatizer algorithm

5.3 The Acquisition Algorithm

Table 2 on the next page shows the pseudo-code for the supervised lexical acquisition algorithm developed for this project. The algorithm picks each word type *w* and lemmatizes it. If the lemmatizer is able to reliably lemmatize *w* by looking up its lemma *l* in the seed lexicon, then *l* along with the POS of *l* is recorded in the acquired lexicon, which simply amounts to copying an entry from the seed lexicon to the acquired lexicon. On the other hand, if the lemmatizer has ventured one or more guesses on what *l* and *pos(l)* can be, then each guessed *l*, in the order of the shortest to the longest, is searched for in the list of word types observed in the corpus. If *l* has not been observed in the corpus, it is then assumed to be spurious and is rejected. This means that only lemmas that have occurred in the corpus are considered legitimate lemmas. This is to avoid conflating the lexicon with non-existent lemmas. If neither *w*, nor a lemma of *w* is found in the seed lexicon or in the corpus, then the *w* itself is recorded as *l* with no assigned POS.

The resulting lexicon is a mapping $\mathscr{L}$: $L \rightarrow E$, where:

- L is all the headwords (lemmas) generated, and E is a triple $<\mathbb{N}, POS, FORM>$;
- $\mathbb{N}$ is the set of natural numbers and stands for the frequency of $l \in L$ in all its forms in the corpus;
- $POS \subseteq TAGSET$ is a list of POS tags from the tagset assigned to l;
- $FORM \subseteq W$ is a list of all the forms observed for l in the set W of all the word types in the corpus with a frequency of five or higher.

6 Results

As mentioned in section 4, we ran the lexical acquisition algorithm on the 79,315 word types, with a frequency of five or higher in the corpus. The algorithm generated a lexicon with 43,346 entries, which is a 45% reduction in the size of the initial word list *W*. Each entry was given 0-3 POS tags. In total, 7,642 (i.e., 17.6%) of the entries were not given any POS tags; 33,935 (i.e., 78.3%) of the entries had only one POS tag; 1,717 (i.e., 4.0%) of the entries had two POS tags; and only 70 (i.e., 0.16%) of the entries were given three POS tags. Obviously, the more frequent lemmas tend to have more forms and tags than the less frequent ones. It is not possible to offer an exact evaluation of the accuracy of the output at this point without comparing the results with a gold standard or through manual analysis. However, an analysis of the top 100 lemmas revealed that about 10% of them contained at least one error in the lemma generated, the assigned tag(s), or the forms attributed to the lemmas. Accuracy, of course, diminishes as word frequency decreases (i) because there are fewer forms in the word list for comparison, (ii) because hypothetical lemmas are less likely to appear in the seed lexicon, and (iii) because many of the infrequent word types are errors themselves. However, post-editing this lexicon is far more efficient than building a lexicon based on *W* from scratch.

Input: *W,* list of word types in corpus; *F,* frequencies of the words in *W*
Output: $\mathscr{L}$, the acquired lexicon

```
Acquire(W):
  ℒ = [ ]
  for w in W:
    A = Lemmatize(w)
    for a in A:
      if a comes directly from L_s:
        add l, pos(l), freq(l) to ℒ
        break
      else:
        if l in W:
          add l, pos(l), freq(l) to ℒ
          break
    if no analysis added to ℒ:
      add w, [ ], freq(w) to ℒ
  return ℒ
```

Table 2: Pseudo-code of the lexical acquisition algorithm

7 Conclusions and Future Work

The lexicon generated using this supervised acquisition algorithm will be post-edited and released as part of the `persian.py` module to be used by its lemmatizer and forthcoming POS tagger. The corpus itself together with the associated Python module will also be made available as open-domain resources. The lexicon, and the accompanying lemmatizer, can also be used in vector space models of Persian in order to significantly reduce dimensionality and interdependence among features.

References

Mohammad-Reza Bateni. 1969. *Tosif-e saxteman-e dasturi-ye zaban-e farsi.* (*Grammatical Description of the Persian Language)*. 12th reprint 2001 by Agah Publishing. Tehran: Amir Kabir.

Aidan Finn, Nicholas Kushmerick and Barry Smyth. 2002. Genre classification and domain transfer for information filtering. In *Proceedings of the European Colloquium on Information Retrieval Research*. Glasgow.

Robert Krovetz. 1993. Viewing morphology as an inference process. In *SIGIR-93*. pp. 191-202. ACM.

Mitchel P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of English: the Penn treebank. *Computational Linguistics*. *19*(2), 313-330.

Karine Megerdoomian. 2004. Finite-State morphological analysis of Persian. In *Proceedings of the Workshop on Computational Approaches to Arabic Script-based Languages*. Coling 2004, University of Geneva, Switzerland.

Karine Megerdoomian. 2006. Developing a Persian part of speech tagger. In *Proceedings of the Second Workshop on Persian Language and Computers*. Tehran University, Iran.

Alireza Mokhtaripour and Saber Jahanpour. 2006. Introduction to a New Farsi Stemmer. In *CIKM'06*. pp. 826-827. ACM

Serge Sharoff. 2006. Creating general-purpose corpora using automated search engine queries. In Marco Baroni and Silvia Bernardini (eds.) *WaCky! Working papers on the Web as Corpus.* Bologna: Gedit.

Kazem Taghva, Russell Beckley and Mohammad Sadeh. 2005. A stemming algorithm for the Farsi Language. In *Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'05)*. pp. 158-162. IEEE.

Appendix

Tag	Description	Example
CC	coordinating conjunction	و /væ/, "and"
CD	cardinal number	یک /yek/, "one"
DT	determiner	این /in/, "this"
EX	exclamation	خداحافظ /xoda-hafez/, "good-bye"
IN	preposition	در /dær/, "in"
JJ	adjective	خوب /xub/, "good"
JJR	comparative adjective	بهتر /beh-tær/, "better"
JJS	superlative adjective	بهترین /beh-tærin/, "best"
MD	modal verb	باید /bayæd/, "must"
NN	common noun	میز /miz/, "table"
NNP	proper noun	ایران /iran/, "Iran"
NNS	plural noun	آداب /adab/, "customs"
OD	ordinal number	اول /ævvæl/, "first"
PP	pronoun	من /mæn/, "I"
PPS	plural pronoun	ما /ma/, "we"
QP	question particle /aya/	آیا /aya/, (marks yes/no questions)
RA	oblique marker /ra/	را /ra/, (oblique marker postposition)
RB	adverb	بعد /bæ'd/, "next"
RBR	comparative adverb	آمرانه تر /amer-ane-tær/, "more commandingly"
RBS	superlative adverb	آمرانه ترین /amer-æne-tærin/, "most commandingly"
RP	particle	لا /la/, (Arabic negation particle)
SC	subordinating conjunction	اگر /ægær/, "if"
UH	interjection	آخ /ax/, "ouch"
VB	verb	ارز /ærz/, "cost"
VBD	past verb	ارزید /ærz-id/, "cost" (past)
VBI	infinitive verb	ارزیدن /ærz-id-æn/, "to cost"
WDT	wh-determiner	چقدر /čeqærd/, "how much"
WP	wh-pronoun	چه /če/, "what"
WRB	wh-adverb	چرا /čera/, "why"

Table 3: Tags used by the system