ParsiPardaz: Persian Language Processing Toolkit

Zahra Sarabi, Hooman Mahyar, Mojgan Farhoodi
Cyber Space Research Institute (ex: ITRC)
Information Technology Faculty
Tehran, Iran
Z_sarabi@csri.ac.ir , Mahyar@csri.ac.ir , Farhoodi@csri.ac.ir

***Abstract*— ParsiPardaz Toolkit (Persian Language Processing Toolkit), which is introduced in this paper, is a comprehensive suite of Persian language processing tools, providing many computational linguistic applications. This system can process and advance all fundamental tasks required for different layers of Persian language processing from its initial layer which is lexical layer up to upper layer which are syntax and semantics. ParsiPardaz Toolkit performs a combination of normalization, tokenization, Spell checker, part of speech tagger, morphological analysis includes lemmatizing and stemming, Persian dependency parser and finally semantic role labeling (SRL). The results show high performance and accuracy.**

Keywords- ParsiPardaz Toolkit; Persian Language Processing; Tokenizer; Morphological analysis; Lemmatizer, Stemmer, POS Tagger, Dependancy Parsing;

I. INTRODUCTION

Persian Language raises many challenges for natural language processing and suffers from the lack of fundamental tools for processing raw Persian text. ParsiPardaz is a utility that, given raw Persian text, performs lexical and morphological analysis, including normalization, tokenization, POS tagging, lemmatizing and stemming. Consequently by adding this information to the raw text, we can generate dependency parse tree for Persian sentences by utilizing ParsiPardaz Dependency parser. Finally, at the highest level of language processing we perform semantic role labeling (SRL) in order to representing the full meaning of the sentences that are parsed.

This paper is organized as follows. First, some difficulties and problems of processing Persian are explained. Next, we describe the general strategy used by ParsiPardaz Tools for overcoming these challenges. In the ParsiPardaz section, we detail the operational aspects of each component of the toolkit. Finally we show the experimental results.

II. CHALLENGES OF PERSIAN PROCESSING

Persian is among the languages with complex and challenging preprocessing tasks. Some of the most important of these challenges are described below.

A. Space and half space

One of the most important problems for Persian which in computational linguistic exists, is the problem of spaces between or within a word. Many Persian words must have half space within them but putting half space is not a common approach during type Persian and users often use space instead of half space. In some other cases it's not necessary to put a space even such as prefix "نا" as "non" which it should be joined to the word it modifies without any hyphen. For example for the word "ناامید" which means "disappointed", three different cases can occur:

TABLE I. THREE DIFFERENT METHODS FOR WRITING THE WORD "ناامید"

د	ی	م	ا	space	ا	ن	**First method**
د	ی	م	ا	Half space	ا	ن	**Second method**
	د	ی	م	ا	ا	ن	**Third method**

The standard style introduced by Academy of Persian Language and Literature (APLL), is third method.

B. Unicode ambiguity

There are different Unicode for a special character. This is because Persian and Arabic have the same characters with various Unicode. For example "ی" (i) has about 56 codes which all have the same shape. Every NLP applications must unify these characters before start any high level language processing.

C. Uncommon characters

Some special characters like {إ ، أ ، ء ، ؤ}, sometimes don't write even in handwritings. The writing system shows different levels of specificity in spelling these letters, e.g. Alef-Hamza-Above (أ A) can be spelled without the Hamza (ء) as Alef (ا A); These various spelling increase ambiguity for any NLP applications. In most cases if we replace these character with one of its famous one, it will not leads to any different word with different meanings. Thus we can replace these characters with one character and unify the words. For example we can replace {إ ، أ} characters with {ا} in order to unify and normalize text.

D. diacritical mark

The Persian language has six vowel phonemes and twenty-three consonant phonemes. These vowel phonemes don't write in text except when lacks of them cause some ambiguity and they act as some diacritical mark. For example "سَر", ("Sar" , head), "سُر"("Sor", slippy), "سِر"("Ser", secret) without vowel are all forms of "سر".

978-1-4799-2093-8/13/$31.00 ©2013 IEEE

III. RELATED WORKS

Less integrated work has been done in the field of Persian language processing and most of them are sporadic tasks which tend to target a specific NLP application. There is just one integrated package for Persian language processing which is STeP-1[1]. We state some of these related works and compare them with ParsiPardaz Toolkit.

Some works which have done in each individual part of ParsiPardaz are: For Tokenizer this work [2] was used as part of Shiraz project. In this tokenizer, the input words first split words and then some specific suffixes reattach to the word. In contrast ParsiPardaz tokenizer performs tokenization with the combination of lemmatizing and some Persian derivational rules and our algorithm is more general. Some of the previous successful Persian POS Taggers is [3]. This tagger is Stanford POS Tagger which is trained on First version of Bijankhan corpus(contains 2.5 million words),while our Stanford POS Tagger train on Second version of Bijankhan corpus(contains 10 million words) which is merged with Dadegan corpus and also based on some syntax and semantic features such as POS tagging of clitics. Another successful POS Tagger is [4] which is trained TnT on Bijankhan corpus. In contrast with this tagger, our POS Tagger has more coarse grain and fine grain tags while it shows higher accuracy too. For Morphological Analyser part these two works have been developed[5][6]. In compare with these morphological analyzers, our system has increased coverage and performance and we also develop a lemmatizer in addition to stemmer .

STeP-1 is a comprehensive package that contains tokenizer, morphological analyzer and POS tagger. In order to compare these two packages we should mention that, while ParsiPardaz Toolkit performs all those preliminary tasks, it also covers higher level of language processing layers which are syntax and semantic. In addition there are some differences in each individual part of both toolkits. For example our morphological analyzer performs both lemmatizing and stemming while STeP-1 just does stemming. ParsiPardaz tokenizer performs its tokenization based on some constraint and rules of dependency syntax theory and also some syntax and semantic features such as tokenization of clitics. ParsiPardaz POS Tagger, shows a higher accuracy and performance although it has much more labels with detailed information.

In addition most of previous works actually happened in a laboratory setting and hasn't been tested on real applications. In comparison, all components of ParsiPardaz toolkit are experienced and used in "QuranJooy Question Answering System" which is a Quranic question answering project and is being conducted in Cyber Space Research Institute(CSRI). All modules and subsystem of this QA System, perform their required NLP tasks with ParsiPardaz Toolkit. This fact has led to test multiple part of ParsiPardaz Toolkit repeatedly and consequently enhance the performance of it significantly.

IV. PARSI PARDAZ TOOLKIT

ParsiPardaz (Persian Language Processing Toolkit) present here provides one solution to specific natural language processing tasks for Persian, from initial layers like lexical Analysis or morphological Analysis up to semantic layers. Figure 1 shows the levels of linguistic analysis to understand the meaning of texts.[7]

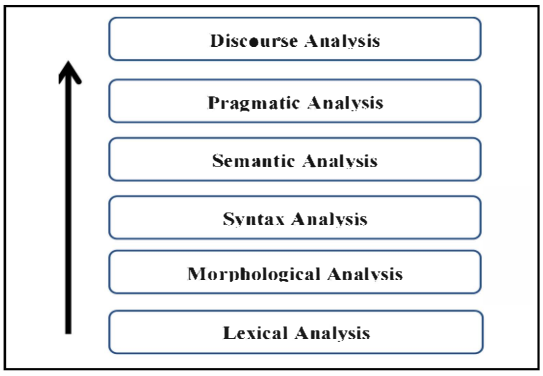

Figure 1. Levels of Linguiestic Analysis

Figure 2 shows ParsiPardaz tools in each corresponding level of linguistic analysis. As shown in Figure 2, ParsiPardaz Toolkit in its first layer performs the initial and base language processing. These principal processing are Normalization, Tokenization, Spell checker and Edition. Users can select arbitrary combination of these tools in different depths for their own application.

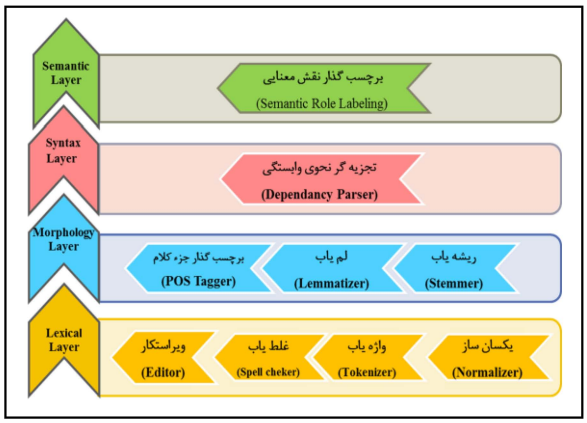

Figure 2. Parsi Pardaz Routins in reaching concept

In second layer morphological tools are exist. These tools are Stemmer, Lemmatizer and POS Tagger which the first one is stemming just based on the word structure independent of its context. In contrast Lemmatizer and POS Tagger are both context dependent and sentence dependent.

In third layer of language processing we go beyond the level of words and reach to sentences level. In this layer which we called it syntax analysis with the help of Persian dependency parser, extract dependency structure of each sentences includes subject, object and verb and if needed, we can identify the groups of each sentence.

In fourth layer which is semantic layer, we do semantic processing based on Semantic Role Labeling (SRL), and Property-Object- Value triple (POV).

ParsiPardaz has developed completely up to third layer which is syntax layer. For the fourth layer, semantic layer, we are preparing a corpus with sentences and their SRL.

978-1-4799-2093-8/13/$31.00 ©2013 IEEE

This corpus would be a useful tool for training and testing Semantic systems. Finally we use a tool provide a pipeline of modules that carry out lemmatization, part-of-speech tagging, dependency parsing, and semantic role labeling of a sentence.

In the rest of this section we will describe the normalizer, tokenizer, morphological analyser, POS tagger and dependency parser as the four main modules of ParsiPardaz.

A. Normalizer or Unification

At the first step of language processing, we perform normalization. This task has 3 main functions. First is unifying different Unicode. It means that all characters with the same shape but different encoding, become unified and as the same. In other words if some character encode with Arabic Unicode, we convert it to Persian Unicode. Table II shows an example of this situation. Second function is omitting diacritics since experiments show that omitting diacritical marks enhance the performance of system and third function is eliminating "Tanwin" and "Hamza".

TABLE II. UNIFY DIFFERENT UNICODE

ساری	ی	ر	ا	س	word
	06CC	0631	0627	0633	code
ساري	ي	ر	ا	س	word
	064A	0631	0627	0633	code

B. Tokenizer

For second step we perform tokenization. Word Segmentation also known as tokenization focuses on recognizing word boundaries exploiting orthographic word boundary delimiters, punctuation marks, written forms of alphabet and affixes[8]. Our proposed method combines rule based methods and dictionary based and converts various forms of writing to a unique standard one. There are 4 main aspects that our approach is followed and we implement them in this module.

- Handling Space & Half Space

First is the problem of space and half space both inside and outside a word. For this reason we insert half space between different parts of a word, we separate numbers, digits, dates and English letters by adding space among their surrounding text. The most problem in spaces is caused by some common affixes which are shown in table III. All of these prefixes and postfixes are joined to the words by half space and in some cases like "نا" (non) joined to the word with no space.

TABLE III. MOST FREQUENT POSTFIXES AND PREFIXES IN PERSIAN

ای (ey)	هایی(haee)	های(haye)	ها(ha)	می(mi)
ترین (tarin)	تر(tar)	نمی(nemi)	اند(and)	بی(bi)
شان (shan)	مان(man)	نا(na)	اید(eid)	ایم (eim)
بر (bar)	هایشان (hayeshan)	هایتان (hayetan)	هایمان (hayeman)	تان (tan)

- Verb conjugation & Noun and Adjectives declension

Second is determining all verb conjugation as one token. In addition for some noun and adjectives, we replace white space by half space in some words that are written separately, according to APLL rules. Those words should be written with half space and converts to one token. Some examples of these words are shown in Table IV. We should mention to this point that some compound verbs in Persian must be written with space among its different part according to APLL rules but still is recognized as one token. Despite this we still join parts of compound verbs with half space. This is because it is not allowed to put space inside one token in parsers.

- Tokenization of Clitics

Third and the most complicated aspect is occurred when a word play a role of more than one word. In this situation some part of a word which stick to the word and has a special capacity should be separated. This token is called clitic. In morphology and syntax, a clitic is a morpheme that has syntactic characteristics of a word, but depends phonologically on another word or phrase.

TABLE IV. SOME VERB CONJUGATION AND NOUN DECLENSION IN TOKENIZATION

Input	Output
آهن ربا "Ahan roba" , "magnet"	آهن‌ربا
گفته خواهد شد "Gofte khahad shod", "Will say"	گفته‌خواهدشد
گفته شده است "Gofte shode ast" ,"has said"	گفته‌شده‌است

Sometimes clitic acts as a verb, sometimes as an object and so on. This separation is needed for higher level of Persian Language processing like parsing and consequently semantic analysis and without it such processing doesn't have appropriate performance. In these cases our tokenizer should recognize these words and separate the clitic from the remaining part of word. Some example of these words are shown in Table V.

TABLE V. TOKENIZATION OF CLITICS

Input	Output	Explanation
خوبم "khobam" "I am well"	خوب + م "khob + am"	Am is verb in this word and must be separated from the adj khob.
خوردمش "khordamash" "I ate it."	خوردم + ش "khordam+ ash"	Ash is obj in this word and must be seprated from the verb Khordam.

- Recognizing compound words

978-1-4799-2093-8/13/$31.00 ©2013 IEEE

In order to perform correct tokenization we need to recognize compound words and join different parts of these words by half space and replace it with original word. An example of such words is "شرکت کننده" (Participant) which two different parts of this word must be joined by half space and therefore the two part word converts to one part "شرکت‌کننده". ParsiPardaz tokenizer performs this task too.

C. *Morphological Analyser*

Morphological analyzing of language words has many applications in natural language processing and some fields such as information retrieval. Two of the most important usage of morphological analysis is extracting stem and lemma from each word.

Stemming usually refers to a crude heuristic process that chops off the ends of words in the hope of achieving this goal correctly most of the time, and often includes the removal of derivational affixes. Stemming has been the most widely applied morphological technique for information retrieval.[9]

Lemmatization usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the lemma.[9]

The benefits of lemmatization are the same as in stemming. In addition, when basic word forms are used, the searcher may match an exact search key to an exact index key. Such accuracy is not possible with truncated, ambiguous stems.

Lemmatization is closely related to stemming. The most important difference is that a stemmer operates on a single word without knowledge of the context, and therefore cannot discriminate between words which have different meanings depending on part of speech. However, stemmers are typically easier to implement and run faster, and the reduced accuracy may not matter for some applications.

Stemmer and lemmatizer of ParsiPardaz is finding stems and lemma of each Persian word using inflectional and some derivational morphological rules in Persian and also using POS Tagger which is implemented in Parsi Pardaz Toolkit. In continue, we first explain Stemmer algorithm and next explain lemmatizing algorithm.

1) *Stemming*

The algorithm which is used for stemming uses 4 distinct date base entirely which 2 of them is prepared by team members. The first and most important database consists of Persian words with their POS tags and also frequency of each word. The second database is for some irregular plural nouns and their corresponding stems. Another data base is a collection of Persian verb stems which for each verb both present stem and corresponding past stem are linked to each other and the forth one is a pattern collection which for each syntactic category keeps their acceptable morphological rules and also their valid prefixes and affixes . The algorithm also use a structure for keeping stems and its expected tags, two lists for keeping prefixes and postfixes which were eliminated from word.

صورت واجی	صورت نوشتاری	کد مقوله نحو	بسامد	الگوی تکیه
SAdkAm	شادکام	A0	3	WS
SAdkAmi	شادکامي	N1	15	WWS
SAdemAn	شادمان	A0	2	WS
SAdemAne	شادمانه	Ad	15	WWWS
SAdemAni	شادماني	N1	4	WWS
SAdmehr	شادمهر	N3	3	WS
SAdi	شادي	N1	165	WS
SAdi	شادي	N3	15	WS
SAdi'Avar	شادي‌آور	A0	3	WWWS
SAdibaxS	شادي‌بخش	A0	9	WWS
SAdibaxSi	شادي‌بخشي	N1	1	WWWS
SAdi'Afarin	شادي‌آفرين	A0	15	WWWWS

Figure 3. part of word database (FLexicon)

The word data base which its name is "FLexicon" contains 54347 entries[10]. Figure 3 shows part of this data base. As shown in this figure for each word, its POS tag and its frequency exist too. As far as both stemming and lemmatizing need POS tags, we first convert the data base tag set to Parsi Pardaz POS tagger tag set which is defined previously. Table VI shows some of the original tags and its equivalent in Parsi Pardaz POS tagger tag set.

TABLE VI. SOME TAGS THAT ARE USED IN BOTH STEMMER AND LEMMATIZER

Original Tag	Equivalent tag in ParsiPardaz POS tagger tag set	Description
A0, A1, A2	ADJ	Adjective
Ad	ADV	Adverb
N1,N2,N3,N4	N	Noun
V1	V_H	Present verb
V2	V_G	Past verb

The verb data base consists of 6066 present and past verb stem with their possible verb prefixes[11]. We use this data base in order to validate if a stem is verb or not. Figure 3 shows part of verb data base.

Past Verb stem	Present Verb stem	Prefix	Prefix
رفت (raft)	رو (rav)	-	-
رفت (raft)	رو (rav)	-	راه (rah)
رفت (raft)	رو (rav)	فرو (foru)	-
افتاد (oftad)	افت (oft)	در (dar)	-

Figure 4. Some part of verb stem data base

The algorithm also uses a data base for some irregular plural nouns which don't obey any rules. Such plural nouns and their corresponding stems keep in this data base. This data base contains about 800 Entries and was collected by the team members.

The forth data base which is prepared by authors is Pattern collection which for each syntactic category keeps their acceptable morphological rules and also their valid prefixes and affixes. We classify these patterns according

978-1-4799-2093-8/13/$31.00 ©2013 IEEE

to 8 syntactic categories which are: verbs, nouns, adjectives, adverbs, pronouns, numbers, prepositions, and gerund form of verbs which is called Masdar. For each syntactic category we collect all possible rules and also all possible prefix and postfixes which is valid for that category. Classification according to syntactic category prevents from invalid stem extraction. For example for Verb category we collect 22 patterns that cover all conjugation of Persian verb patterns. For Adjective category there are 4 patterns and a list of valid postfixes which "تر"(er) and "ترین"(est) is two of the most important ones. Table VII shows some rules or patterns that used in morphological analyzer.

TABLE VII. SOME RULES IN PATTERN DATA BASE

Rule	Example
mi+ present root +First Single Personal Postfix	می‌روم
Khah + Second Single personal postfix + Past root	خواهی رفت
Adjective+ Adjective postfix	زیباتر
Noun+ Plural postfix+ personal postfix	کتاب‌هایم

The Stemming algorithm define as implementation of *getStem(word)* function. In this function for all input words, we first search the word in irregular plural nouns data base and if it is found then the algorithm returns its corresponding stem as a result. Otherwise we start eliminating affixes from word and if we find the remaining part in the word lexicon (FLexicon), we go to second step. In the second step, by using pattern data base, we should check if the cutting affix is a valid affix for the remaining word with its corresponding POS tag. If the result becomes true, then we return the remaining word as a valid stem. If a word is matched with any verb patterns we should validate if the outcome is an accurate verb stem. In this case we search the outcome in verb stem data base and if it finds then the outcome saves as a valid stem.

Sometime we need to eliminate more than one affix to find the stem. In these cases for each resulted stem, we keep its POS Tag and also the stem frequency and the pattern which stem and affixes are matched with that. Eventually the algorithm returns all resulted stems with its frequency which is mentioned in FLexicon, and its relative pattern. The frequency can be used for some disambiguation of stems. For example for each word the most frequent stem can be obtained with this option.

2) Lemmatizing

Lemmatization is closely related to stemming. The difference is that a stemmer operates on a single word without knowledge of the context, and therefore cannot discriminate between words which have different meanings depending on part of speech. For instance the word "مردم" (people) can be either the base form of a noun or a form of a verb "مردن" (to die) depending on the context. In this case, unlike stemming, lemmatization can in principle select the appropriate lemma depending on the context.

We define the task of lemmatizing a Persian word as the one of implementing the function *getLemma(word , tag)*. This definition implies that for a given word form and tag, there exists one and only one correct lemma and it also leaves all issues of context sensitivity to the task of determining the tag. Thus the input to lemmatization consists of a word and a part of speech tag, whose format is described in Section POS tagger of this paper. Therefore the tag should be obtained by using the ParsiPardaz POS tagger.

In this function we first obtain all possible stems of the word by calling *getStem(word)* function which is described in stemming section. Thus every stem with its corresponding POS tag are obtained. In parallel we run our POS tagger over the sentence and acquire the word POS tag in that sentence. Finally the lemmatizer goes for the most specific matching identity. We select a stem from stems list which its corresponding POS tag is identical to the word POS tag in the sentence. This stem is specified as the word lemma.

In some insignificant cases the tag which is obtained by POS tagger cannot find in stems list. In these situations the lemmatizer would just pick the most frequent stem for the word according to each stem frequency and return it as lemma. Table VIII shows some examples of lemmatizer.

TABLE VIII. LEMMATIZER OUTPUT

Input Sentence	Input (word, POS Tag)	Output lemma
"شادمانی در میان مردم موج می‌زد."	("مردم", N)	مردم
"از کار زیاد داشتم می‌مردم ."	("می‌مردم", V)	مرد
"به این بخش خبری توجه کنید."	("خبری", ADJ)	خبری
"دیروز خبری به دستم رسید ."	("خبری", N)	خبر

D. POS Tagger

Part of speech (POS) tagging usually appears as a fundamental task in many high level applications of Natural Language Processing like syntax parsing and semantic analysis and the POS tag of words used as a elementary feature of such processing. Thus having taggers with high precision is of high importance. The algorithm is based on data driven(machine learning) techniques which in the field of POS tagging have resulted in some successful part of speech tagger such as Stanford POS Tagger[12], TnT[13], HunPoS[14] and so on. Stanford is an open source software that is based on Maximum Entropy methods. HunPoS is also another open source tagger which is a reimplementation of TnT that is based on Hidden Markov Models (HMM). Both taggers allowing the user to tune the taggers by using different feature settings. Although we applied both taggers but since Stanford POS tagger has resulted to more accurate and reliable outputs and also is a Java implementation software we reports our experiment with this tagger as ParsiPardaz final POS tagger.

The first and most important step in this implementation was preparing a corpus with appropriate POS tags. For this reason we merge 2 corpuses which are Dadegan corpus[15] and Bijankhan corpus[16]. The reason of merging these corpuses is acquiring a large enough corpus with the majority of words that can act as a train set for POS tagger. Dadegan is a Persian treebank which is produced as a training set for Persian dependency parser. In this treebank all words has a coarse grain and a fine

978-1-4799-2093-8/13/$31.00 ©2013 IEEE

grain POS tag. Since ParsiPardaz parser train on this treebank, we choose it for training set of POS tagger too. Bijankhan corpus is a first manually tagged Persian corpus which its first version consists of nearly 2.6 million words but in the last release it developed to near 10 million words. Dadegan Treebank has 17 coarse grain tag set while Bijankhan corpus has 14 coarse grain tag set and about 500 fine grain tag set. The integrated new corpus has 11 coarse grain and 45 fine grain tag set.

In order to merge these 2 corpuses, we should first define a new tag set. As far as our dependency parser will be trained on Dadegan corpus, this corpus tag set is used as the baseline and the Bijankhan tag set is mapped to it. Although in some restricted tags the Dadegan tags is converted to Bijankhan tags. The main approach for this integration is as follows:

- Base the original Dadegan tag set as the desired tag set
- Decrease the number of coarse grain Dadegan tag set from 17 to 11 tags
- Add "N_INFI" tag for gerunds (Masdar) to Dadegan corpus
- Map original Dadegan Tag set to the new defined Tag set
- Map Bijankhan Tag set to the new defined Tag set
- Merged total words of both corpuses with unified tags and produce a new integrated corpus with more than 10 million words.

Table IX shows the list of our new coarse grain tags, the corresponding Bijankhan tags and also corresponding Dadegan tags.

TABLE IX. COARSE GRAIN TAG SET IN EACH CORPUS

New Integrated corpus Tag set	Dadegan Tags	Bijankhan Tags
ADJ	ADJ	AJ
N	N, IDEN	N, RES, CL
ADR	ADR	INT
ADV	ADV	ADV
CONJ	CONJ, PART, PSUS, SUBR	CONJ
NUM	POSNUM, PRENUM	NUM
POSTP	POSTP	POSTP
PR	PR	PRO
PREP	PREP, PREM	DET, P
PUNC	PUNC	PUNC
V	V	V

We have 45 fine grained tags, which the POS tagger is learned them and the final precision of the tagger is measured by these fine grain tags not by coarse grain tags. Table X Shows some of our final fine grain tags which the output of the tagger will be like them.

TABLE X. FINE GRAIN TAG SET FOR PARSIPARDAZ POS TAGGER

Tag	Description	Tag	Description
N_SING	Single Noun	V_MOD_H	Present modal verb
N_PLUR	Plural Noun	V_G_1_PLUR	Past 1th person plural verb
N_INFI	Gerunds	V_AY_2_SING	Future 2th person sing verb
PR_INTG	Question word	PUNC	Punctuation

For implementation of the POS tagger we divided the new corpus by 5 fold cross validation techniques to train and test set and apply the Stanford POS tagger on them. The architectures which we used for Stanford POS tagger are the most 2 common architectures which are "bidirectional" and "left3words". In order to customized this tagger for Persian language we added some detailed information to Stanford POS Tagger.

The precision of "bidirectional" model is upper than "left3word" but the speed of tagging by "left3word" model is more in contrast with "bidirectional".

E. Dependency parsing

The process of manually annotating linguistic data from a huge amount of naturally occurring texts is a very expensive and time consuming task. Due to the recent success of machine learning methods and the rapid growth of available electronic texts, language processing tasks have been facilitated greatly. Considering the value of annotated data, a great deal of budget has been allotted to creating such data(Rasooli, Kouhestani, and Moloodi 2013).

For Dependency parsing we used Malt-Parser (V 1.7.2)[17]. Malt Parser is a system for data-driven dependency parsing, which can be used to induce a parsing model from Treebank data and to parse new data using an induced model. Malt Parser is developed by Johan Hall, Jens Nilsson and Joakim Nivre at Uppsala University, Sweden. We have also used Persian syntactic dependency Treebank (DADEGAN) to learn. For this reason we have changed (Edit) POS labels and morphological labels for more compatibility with other components such as POS.

TABLE XI. DADEGAN TREEBANK *STATISTICS*

1	Number of Sentences	29,982
2	Number of Words	498,081
3	Average Sentence Length	16.61
4	Number of Distinct Words	37,618
5	Number of Distinct Lemmas	22,064
6	Number of Verbs	60,579
7	Number of Verb Lemmas	4,782
8	Average Frequency of Verb Lemmas	12.67

Also the main *results* of our *evaluation* are depicted in the following table.

TABLE XII. *EVALUATION* DEPENDENCY PARSING WITH MALTPARSER (V 1.7.2).

N	evaluation field	Result New Corpus	Result Main Corpus
1	Unlabeled Relations	91.26	90~
2	Labeled Relations	87.34	85~

F. Semantic Role Labeling

At the highest level of language processing we perform semantic role labeling (SRL) in order to representing the full meaning of the sentences that are parsed. For Semantic

978-1-4799-2093-8/13/$31.00 ©2013 IEEE

Role labeling we first prepare a corpus which label the thematic roles in CoNLL format. This corpus is on Persian translation of "Taha Surah" which is one of the holy Quran Surah. This semantic corpus will be developed and increased in future. We use a language independent tool which provide a pipeline of modules that carry out lemmatization, part-of-speech tagging, dependency parsing, and semantic role labeling of a sentence[18]. Since this module is in progress during "QuranJooy Question Answering System", the result and evaluation of this module will be reported in future works.

V. Experimental Result

In order to test the different parts of ParsiPardaz Toolkit we prepare different test sets for each NLP modules. For lexical layer modules like Normalizer, tokenizer, Spell checker we use an input text with about 100 sentence and 1000 words. This input text contains many different verb conjugation, frequent postfix and prefix in Persian, abbreviation, date, number and so on. The result shows 95% in precision for tokenizer and 100% for normalizer.

The POS Tagger was tested by 5-fold cross validation over Bijankhan 10 million tokens corpus and the overall result show about 98.5% precision for ParsiPardaz POS Tagger.

The Morphological analyzer was tested by randomly selected 100 sentence of Bijankhan corpus and the result shows about 86% for lemmatizer and 88% for stemmer.

Table XIII shows the final results of each ParsiPardaz Toolkit components in an overall view.

TABLE XIII. Overall results of Parsi Pardaz Toolkit components

N	Parsi Pardaz modules	Result
1	Normalizer	100%
2	Tokenizer	95%
3	POS Tagger	98.5%
4	Lemmatizer	86%
5	Stemmer	88%
6	Dependancy Parser(labeled)	87%
7	Semantic Role Labeler	In progress

VI. Conclusion

As mentioned earlier, Persian is a language with its own challenges. We tried to overcome some of those challenges by preparing valuable Persian Language processing tools in a comprehensive suite named ParsiPardaz Toolkit. Most of individual component of this package has high accuracy and performance among similar Persian components. ParsiPardaz is the only Persian package that covers higher level of Persian Language Processing such as syntax and semantic. Furthermore we prepared some useful database such as Persian verb patterns. The most important output of this Toolkit is its Semantic analysis component which performs Semantic Role Labeling (SRL). For this purpose we build a semantic corpus over Persian translation of "Taha Surah" which is one of the holy Quran Surah. We also apply a language independent tool for SRL and train it over this corpus. We hope to develop this semantic corpus over Quran and report more details about the linguistic aspects and the findings and experiments on the semantic analysis task in future publications.

Acknowledgment

The authors thank the QuranJooy team members for their tests, reviews and comments regarding the proposed scheme: Parisa Saeedi, Ehsan Darrudi, Ehsan Sherkat, Sobhan Mosavi, Samaneh Heidari, Majid Laali, Marjan Bazrafshan, Ali Rahnama, Vali Tawosi. We also express our gratitude to QuranJooy Project Supervisors for their valuable insights: Dr. Alireza Yari, Dr. Behrouz Minaei, Dr. Kambiz Badie, Dr. Farhad Oroumchian.

References

[1] M. Shamsfard, H.S. Jafari, M. Ilbeygi, STeP-1: A Set of Fundamental Tools for Persian Text Processing., in: LREC, 2010.

[2] K. Megerdoomian, R. Zajac, Tokenization in the Shiraz project, technical report, NMSU, CRL, Memoranda in Computer and Cognitive Science, 2000.

[3] Z. Shakeri, N. Riahi, S. Khadivi, Preparing an accurate Persian POS tagger suitable for MT, (1391).

[4] M. Seraji, A statistical part-of-speech tagger for Persian, in: Proc. 18th Nord. Conf. Comput. Linguist. NODALIDA 2011, 2011: pp. 340–343.

[5] E. Rahimtoroghi, H. Faili, A. Shakery, A structural rule-based stemmer for Persian, in: Telecommun. IST 2010 5th Int. Symp., 2010: pp. 574–578.

[6] J. Dehdari, D. Lonsdale, A link grammar parser for Persian, Asp. Iran. Linguist. 1 (2008).

[7] D. Jurafsky, J.H. Martin, A. Kehler, K. Vander Linden, N. Ward, Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition, MIT Press, 2000.

[8] S. Kiani, M. Shamsfard, Word and Phrase Boundary detection in Persian Texts, in: 14th CSI Comput. Conf, 2008.

[9] A.K. Ingason, S. Helgadóttir, H. Loftsson, E. Rögnvaldsson, A Mixed Method Lemmatization Algorithm Using a Hierarchy of Linguistic Identities (HOLI), in: Adv. Nat. Lang. Process., Springer, 2008: pp. 205–216.

[10] M. Eslami, M. Sharifi, S. Alizadeh, T. Zandi, Persian ZAYA lexicon, in: 1st Work. Persian Lang. Comput., 2004: pp. 25–26.

[11] M.S. Rasooli, A. Moloodi, M. Kouhestani, B. Minaei-Bidgoli, A syntactic valency lexicon for Persian verbs: The first steps towards Persian dependency treebank, in: 5th Lang. Technol. Conf. LTC Hum. Lang. Technol. Chall. Comput. Sci. Linguist., 2011: pp. 227–231.

[12] K. Toutanova, D. Klein, C.D. Manning, Y. Singer, Feature-rich part-of-speech tagging with a cyclic dependency network, in: Proc. 2003 Conf. North Am. Chapter Assoc. Comput. Linguist. Hum. Lang. Technol.-Vol. 1, 2003: pp. 173–180.

[13] T. Brants, TnT: a statistical part-of-speech tagger, in: Proc. Sixth Conf. Appl. Nat. Lang. Process., 2000: pp. 224–231.

[14] P. Halácsy, A. Kornai, C. Oravecz, HunPos: an open source trigram tagger, in: Proc. 45th Annu. Meet. ACL Interact. Poster Demonstr. Sess., 2007: pp. 209–212.

[15] M.S. Rasooli, M. Kouhestani, A. Moloodi, Development of a persian syntactic dependency treebank, in: Proc. NAACL-HLT, 2013: pp. 306–314.

[16] F. Raja, H. Amiri, S. Tasharofi, M. Sarmadi, H. Hojjat, F. Oroumchian, Evaluation of part of speech tagging on Persian text, Univ. Wollongong Dubai-Pap. (2007) 8.

[17] J. Nivre, J. Hall, J. Nilsson, Maltparser: A data-driven parser-generator for dependency parsing, in: Proc. LREC, 2006: pp. 2216–2219.

[18] A. Björkelund, L. Hafdell, P. Nugues, Multilingual semantic role labeling, in: Proc. Thirteen. Conf. Comput. Nat. Lang. Learn. Shar. Task, 2009: pp. 43–48.

978-1-4799-2093-8/13/$31.00 ©2013 IEEE