Journal of AI and Data Mining
Published online:

Feature Engineering in Persian Dependency Parser

S. Lazemi and H. Ebrahimpour-Komleh*

Department of Computer Eng., University of Kashan, Kashan, Iran.

Received 16 October 2017; Revised 30 January 2018; Accepted 10 March 2018
**Corresponding author: ebrahimpour@kashanu.ac.ir (H. Ebrahimpour-Komleh).*

Abstract
Dependency parser is one of the most important fundamental tools in the natural language processing, which extracts the structure of sentences and determines the relations between words based on the grammar dependency. The dependency parser is proper for free-order languages such as Persian. In this work, data-driven dependency parser is developed with the help of phrase-structure parser for Persian. The defined feature space in each parser is one of the important factors involved in its success. Our goal is to generate and extract appropriate features to dependency parsing of Persian sentences. In order to achieve this goal, new semantic and syntactic features are defined and added to the MSTParser by the stacking method. Semantic features are obtained using word clustering algorithms based on syntagmatic analysis, and the syntactic features are obtained using the Persian phrase-structure parser, and are used as bit-string. Experiments are conducted on the Persian Dependency Treebank (PerDT) and the Uppsala Persian Dependency Treebank (UPDT). The results obtained indicate that the definition of new features improves the performance of the dependency parser for Persian. The achieved unlabeled attachment scores for PerDT and UPDT are 89.17% and 88.96%, respectively.

Keywords: *Dependency Parser, Phrase-structure parser, MSTParser, Stacking, Persian.*

1. Introduction

Parsing forms the syntactic layer of NLP layers. The purpose of this processing layer is to consider a sentence as a linguistic unit. In other words, the sentence grammatical analysis is done in this layer. Breaking a sentence into its components is done by the parser. The aim of this analysis is to determine the grammatical role of words in the sentence. In order to determine the syntax structures, we need the grammar and parsing techniques (sentence analysis method for specifying its syntax structure based on linguistic grammar) [1].

In general, parsers are divided into two categories: "phrase-structure parsers and dependency parsers" [2].

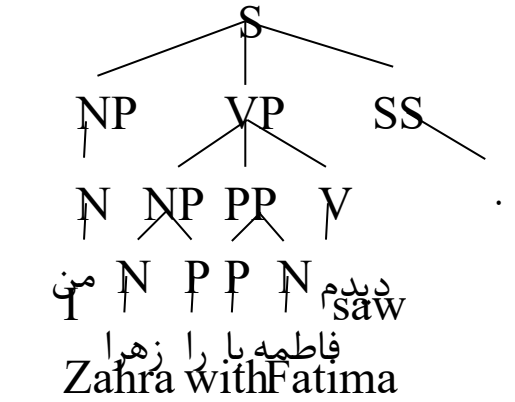
Phrase-Structure parsers: Phrase-structure parsers do the sentence structure extraction due to structural grammar (phrasal structure grammar) [3]. Structural grammars are used to describe formal languages.

The phrase-structure parsers are divided into two categories [4]:

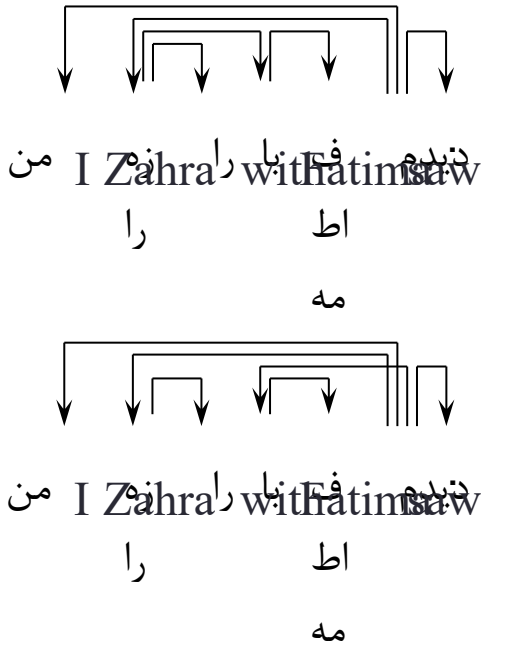
1- Rule-based parser: Sentences are broken down on the basis of the pre-defined rules. Procurement of rules manually is difficult and time-consuming; it is impossible to provide comprehensive rules that have a high coverage power. This type of analysis is not efficient due to the complexity of a natural language, and will be failed in the face of statements outside the defined rules.
2- Statistical-based parser: It tries to extract grammar automatically using statistical techniques and linguistic corpora. The problem of this method is the requirement for annotated treebank.

Dependency parsers: Dependency parser extracts the sentence structure due to dependency grammar. Dependency grammar is based upon the syntactic and conceptual relationships between words [5]. Dependency parsing analyzes the relationships between words, and therefore, can be useful in resolving

structural ambiguity. For example, the sentence "I saw Zahra with Fatima." has a phrase-structure tree (Figure 1-a) and two dependency trees (Figure 1-b); depending on the dependency parsing, the main meaning of the sentence can be obtained.



a-Phrase-structure tree.



b- Dependency trees.

Figure 1. An example of parse tree.

Dependency parsers are divided into two categories:

1- Data-Driven: The machine learning method is used in the data-driven methods, and it is assumed that the input data has a correct syntax structure. The data-driven method is divided into two categories: transition-based and graph-based. In the transition-based approach, the automata are defined, and next, predicate is predicted due to a predicate history [6]. In a graph-based approach, a space of dependency graph is created, and the scoring to them is done and the graph with the highest score will be selected [7].

2- Grammar-Driven: In the grammar-driven approach, a number of grammars are defined, and the structures that are out-of-grammar are assumed grammatically incorrect. The grammar-driven method is divided into two categories as well: "context-free and constraint-based" [7].

The tree that is obtained from dependency parsing can be projective (that has disjunct edges), non-projective (that has cross edges), and well-made (that has root, unique labels, acyclic, connected, and projective). In a projective tree, a word along with its dependents are seen as a substring of the sentence [8]. In Persian, the projective trees cover more sentences due to the free word-ordering property [9]. Also phrase-based approaches are not efficient in this language (due to the free word-ordering property), and research works in this area have been conducted to dependency-based parsing. The research works are very limited, and the proposed methods are limited to grammar-based approaches. In addition, some studies have focused on adapting the existing tools with the Persian language. The focus of this research work was to develop a graph-based dependency parser for the Persian.

One of the simplest ways to improve systems is to combine different systems. The improvement of data-driven dependency parsers is also possible by combining other parsers with the voting and stacking methods [10]. The voting method for dependency parsers was presented by Zeman and Zabokrtsky [11] for the first time. In this method, the input sentence is decomposed by independent parsers (at least three parsers), and then for the final analysis, the output of the parsers is combined based on the majority vote. In the stacking method, the output of one or more parsers is used as the input of the parser in question [10]. In other words, the output of other parsers is used to generate new features in the desired parser. In languages such as English and Czech, the voting and stacking methods have been used to improve the performance of parsers such as [12-14] but there are no research works in this regard in the Persian. In this work, we focused on improving the MSTParser [15] for Persian with the stacking method. Given the specific properties of the Persian, MSTParser is suitable for this language because in Persian:

1- Due to the SOV property, the head and the dependent are usually spaced far from each other, and MSTParser is appropriate to

determine long-distance relationships due to the creation of a sentence graph.

2- Due to the free word-order property, most Persian sentences produce non-projective trees, and MSTParser is able to produce non-projective trees.

In order to achieve our goal, firstly, the semantic features were defined by the Brown Clustering Algorithm (BCA), and then using the stacking method, we defined the syntactic features from the designed phrase-structure parser output for Persian. The rest of the paper is organized as what follows. In the second part, the conducted research works in the field of Persian parsers are discussed. In the third section, after a brief summary of MSTParser, our semantic and syntactic features are presented. The fourth section includes experiments and results. Finally, in the fifth section, a summary of the paper is presented with some future works.

2. Related works

2.1. Phrase-structure parser

The parser presented by Estiri et al. [16] is the first Persian phrase-structure parser, and has six stages. In the first step, the required tags are defined with accurate signs and a hierarchical order. the tag collection is started from groups that form sentences; and includes five groups: nominal, verbal, adverbial, adjectival, and preposition; and according to the type of group and the tags of each word that forms the group, the tags that belong to the same group are developed with more details. The number of tags contains 24 labels. The second stage involves pre-processing (unification, remove the short space), sentence segmentation (using punctuation, Persian language grammar, and the sentence initial word) and identifying words (using space and punctuation). In the third stage, the words initial tag is identified by searching in the prepared database and defined rules. The fourth step identifies the features of its verbs. For this purpose, the verb is identified by the probability of presence in a specific situation, and then the word probability to be a verb is done with regular expressions; if it matches, the verb group is constructed, and the head is considered as a verb. By considering the structure of the verb in Persian and using a dictionary, the features including tense, person, mood (imperative, indicative, subjunctive), and voice (active or passive) are extracted for verbs. In the fifth step, the noun features such as common or proper, definite or indefinite, and singular or plural are extracted. To do so, first, the stem of nouns in found, and then the features are extracted using a dictionary and considering the added affixes. In the last step, a generic label is assigned to multiple words or same label by combining tags and forming groups. Finally, the sentence parse tree is obtained by assigning labels to groups and words. Due to the lack of similar parser to Persian, the comparison and evaluation has not been done. One of the disadvantage of the parser can be the inability to deal with unstructured sentences and non-grammatical.

2.2. Dependency parser

Seraji et al. [17] have proposed a parser named "ParsPer". At first, they found the best tag sets for POS tagging and depency relations using MaltParser [18], and then operated their tests with other parsers such as MSTParser [15], MateParser [19, 20], and TurboParser [21]; and they selected MateParser due to its good results for Persian. To evaluate ParsPer, first, parser was tested with Uppsala Treebank corpus, and then for the final results, parser was tested on the other corpus. MaltParser have used MaltOptimizer [22] to do the optimization processing. Based on their findings, graph-based parsers showed a better performance than transition-based parsers. The results obtained for the labeled and unlabeled attachment score was reported to be 82/58% and 86/69%.

Falavarjani and Ghassem-Sani [9] have tested two state-of-the-art parsers, "MaltParser (transition-based), and MSTParser (graph-based)" for Persian. In Persian, due to the point that non-projective trees do not cover more sentences, they tested two parsers to find the appropriate parser for projective and non-projective sentences. To reach this goal, the authors divided the sentences in the corpus into two categories: "projective sentences and non-projective sentences", and used POS tags that were from the Treebank corpus as features. To apply the MaltParser and Arc-eager algorithms, and to apply MSTParser, the second-order and non-projective settings were selected. The authors introduced MSTParser appropriate for Persian during the tests. The accuracy obtained for projective sentences by MaltParser was 87%; by MSTParser, it was 84%; for non-projective sentences by MaltParser, it was 73%; and by MSParser, it was 77%.

Seraji et at. [23] have tested the two state-of-the-art parsers "MSTParser" and "MaltParser" using the UPEDT corpus for Persian. MSTParser was performed with four different settings: first-order projective, second-order projective, first-order-non-projective, and second-order-non-projective. MaltParser was performed with different algorithms, and features such as Nivre algorithms and Covington algorithms, gold standard POS features, and Auto generated POS tags, and the MaltOptimizer tool were used for optimization.

The Nivre-eager algorithm was selected with the gold standard POS features for MaltParser due to tests and second-order feature, and projective was selected for MSTParser. During the experiments, MaltParser yielded better results.

3. Features

3.1. Default model features

The graph-based dependency parsing consists of three main stages (definition of sentence space, learning, and parsing). In the first step, a space of candidate dependency directed graphs is created for the sentence. In the learning phase, a model for scoring is determined for the dependency graph of the sentence. The parsing stage seeks to find the highest-score dependency graph (to resolve the ambiguity). There are different algorithms for graph-based parsing, and the most notable one is the arc-factored model. In the arc-factored model, the dependency graph is divided into several sub-graphs, P_1, P_2, ..., P_n; each sub-graph is individually scored, and the score of the graph is considered as the total score of the sub-graphs. In this model, each edge is represented by a feature vector. At the learning stage, each feature is weighted. Scoring to an edge in a weighted sum of all its features is done, (1).

$$S(h,d) = f_1 w_1 + ... + f_n w_n \quad (1)$$

w is the weight vector, which is calculated using the machine learning algorithms from the training samples (pairs of sentences and the corresponding dependency trees).

The next step is to find the highest score graph. MSTParser converts the issue of finding the highest score graph into a Discriminative Maximum Spanning Tree. If for the sentence $X = (x_1, x_2, ..., x_n)$ $T(G_X)$ is the spanning tree set of G_X, the MSTParser target is to find the spanning tree G' with the highest weights, (2).

$$ParseTree(X) = \arg \max_{G \in T(G_X)} \sum_{r \in G'} w.f(X,p) \quad (2)$$

w is the weight vector, and $f(X,p)$ is the feature vector for part p.

The default sfeature set that has been used in MSTParser is summarized as follows, used as uni-gram and bi-gram:

- POS tags of the words H_i and D_j and the label L_K
- POS tags of words surrounding and between H_i and D_j
- Number of words between H_i and D_j and their orientation
- Label features

3.2. Semantic features

At this stage, BCA has been used to create a semantic feature set. This algorithm examines the words semantically, and believes that similar words appear in the same contexts [24]. In other words, similar words have the same distribution in relation to their previous and next words. Koo et at. [25] have also used BCA to create semantic features. They believed that finding the suitable number of bits had a huge effect on parsing. Therefore, the main purpose of this stage is to find the number of bits that can show the semantic dependency of the Persian words.

The input of the algorithm is a large corpus of words, and its output is a partition of words in the hierarchical clusters. The algorithm, at first, puts each one of the words into separate clusters, and then combines the two clusters that have the maximum mutual information, (3). The output of the algorithm is a binary tree whose leaves show the words, and the root-to-leaf path represents the bit-string of words. The words that are semantically similar will have the same left bits [24].

$$v = \{w_1, w_2, ..., w_n\} \quad (3)$$
$$C : v \to \{1,2,..., k\}$$
$$Quality(C) = \sum_{c_1,c_2} P(c_1,c_2) \log \frac{P(c_1,c_2)}{P(c_1)P(c_2)} + \sum entropy(w)$$

Based upon the experiments conducted with multiple settings, the number of bits is considered to be 5-11.

3.3. Syntactic features

The information contained in the phrase-structure parser represents the structural information of the sentences, so this information can be injected as a structural feature to the dependency parser. We have constructed a feature vector for each word to extract the syntactic features using the phrase-structure trees, and we have convert the extracted features into a bit-string for injection into dependency parser. To create a bit-string of words, words are required to be clustered, so in the second step, the clustering of the words has been done. Our proposed method consists of three steps whose details are as follow:

Generating syntactic feature vector: The word parse tree path contains its structural information. In order to construct a syntactic word feature vector, we have used the modified method presented in [26]. In their proposed method, the path of the parse tree for verbs was used to semantic role labeling. In this work, the parsing tree path for all words has been expanded with some changes. To construct the syntactic feature

vector of the word "w", all sentences containing the "w" are identified, and their parse tree is created by [16]. Then all paths that start with the "w" and end up with words that are associated with "w" (head and dependent of "w") are stored. This reduces the feature vector dimensions of the "w". The syntactic feature vector dimension of the "w" is equal to the number of word's unique paths, and the amount of each dimension is obtained by dividing the number of unique path repetition into the total number of sentences. For example, the paths for the "Zahra" in figure 1-a can be saved as follows:
<NP, VP, V> <NP, P> <NP, VP, PP, P>

Clustering: At this point, the words are clustered using the K-means algorithm. The similarity of words is structurally calculated using the cosine similarity between pairs of words.

Word bits representation: To use the created syntactic feature, we display them in bits. The presented algorithm in [27] is used for this purpose. This algorithm has received clusters and creates a binary tree of words, and to each branch of the left and right is assigned a zero bit or one. Moving from the root to the leave shows the word bits of words. In this algorithm, the words that are structurally similar are placed in close places.

Our semantic and syntactic feature templates are represented in table 1. In the second dependency tree in figure 1-b, some features for (saw and Zahra) are as:
H-word: saw
D-word: Zahra
H-sem, H-syn, D-sem, D-syn: bit sting
H-sem+1, H-syn+1: bit sting to the right of head
H-sem-1, H-syn-1: bit sting to the left of head
D-sem+1, D-syn+1: bit sting to the right of dependent
D-sem-1, D-syn-1: bit sting to the left of dependent
W_k-word: with
W_k -sem, W_k -syn: bit sting
B-sem, B-syn: bit sting of word between H and D

4. Experiments and results

4.1. Corpus

For the experiments, the Persian Dependency Treebank (PerDT) [28] and the Uppsala Persian Dependency Treebank (UPDT) [29] have been used. PerDT is the first Persian dependency Treebank [30], and includes about 30,000 sentences annotated with syntactic roles and morpho-syntactic features and the corresponding dependency tree. There are 44 dependency relations, 17 types of coarse-grained, and 32 types of fine-grained POS tags.

In UPDT, the syntactic relation of words is determined by the dependency grammar. This corpus contains 6000 sentences from the Uppsala Persian Corpus (UPC-a modified version of the BijanKhan corpus [31]) with a corresponding dependency tree. In this corpus, there are 48 types of dependency relations, 15 types of coarse-grained, and 32 types of fine-grained POS tags. Both corpora are prepared based on the CoNll template and the Stanford Typed. More information about the corpora used is given in table 2.

Table 1. List of semantic and syntactic features for head word (H) and its dependent (D), words between H and D (B) and sibling words of D between H and D (w_k).

#	Semantic Feature	Syntactic Feature
1	H-word, H-sem	H-word, H-syn
2	D-word, D-sem	D-word, D-syn
3	H-sem	H-syn
4	D-sem	D-syn
5	H-word, H-sem, D-word, D-sem	H-word, H-syn, D-word, D-syn
6	H-sem, D-word, D-sem	H-syn, D-word, D-syn
7	H-word, D-word, D-sem	H-word, D-word, D-syn
8	H-word, H-sem, D-sem	H-word, H-syn, D-syn
9	H-word, H-sem, D-word	H-word, H-syn, D-word
10	H-word, D-word	H-word, D-word
11	H-sem, D-sem	H-syn, D-syn
12	H-sem, B-sem, D-sem	H-syn, B-syn, D-syn
13	H-sem, H-sem+1, D-sem-1, D-sem	H-syn, H-syn+1, D-syn-1, D-syn
14	H-sem-1, H-sem, D-sem-1, D-sem	H-syn-1, H-syn, D-syn-1, D-syn
15	H-sem, H-sem+1, D-sem, D-sem+1	H-syn, H-syn +1, D-syn, D-syn+1
16	H-sem-1, H-sem, D-sem, D-sem+1	H-syn-1, H-syn, D-syn, D-syn+1
17	H-sem, W_k-sem, D-sem	H-syn, W_k-syn, D-syn
18	W_k-sem, D-sem	W_k-syn, D-syn
19	W_k-word, D-word	W_k-word, D-word
20	W_k-sem, D-word	W_k-syn, D-word
21	W_k-word, D-sem	W_k-word, D-syn

4.2. Evaluation metrics and results

For evaluation, the Unlabeled Attachment Score and the Labeled Attachment Score are used and defined as (4) and (5).

The corpora are split into standard train and test sets, 90% of the corpora are used for training and 10% are used for testing. In the stage of creating syntactic features, we are facing with the sparsity problem, so to reduce the problem, the sentences that contain full-frequent words of corpora are selected.

$$\text{UAS} = \frac{\text{Number of identical edges in two trees regardless of the label}}{\text{Total number of two tree edges}} \quad (4)$$

$$\text{LAS} = \frac{\text{Number of identical edges in two trees with the label}}{\text{Total number of two tree edges}} \quad (5)$$

MSTParser is used with four settings: projective-first-order, projective-second-order, non-projective-first-order, and non-projective-second-order. MIRA is used to estimate the weight vector. In order to determine the optimal number of clusters and bits, the number of clusters has been changed from 100 to 1000 by step 100, and the number of bits has been changed from 5 to 14 by step 3, and finally, for the first corpus in the semantic phase, 400 clusters and 11 bits, in the syntactic phase, 500 clusters and 8 bits, and for the second corpus in the semantic phase, 300 clusters and 11 bits, and in the syntactic phase, 400 clusters and 5 bits have been selected.

Table 2. Statistical Properties of PerDT and UPDT.

	Persian Dependency Treebank	Uppsala Persian Dependency Treebank
Number of sentences	29982	6000
Number of words	498081	151671
Number of distinct words	37618	15692
Average sentence length	16.61	25.28
	is freely available in CoNLL format	is freely available in CoNLL format

To evaluate the quality of the defined features, we have tested the effects of each one separately. Tables 3, 4, 5, and 6 show the accuracy obtained for both corpora. As it is evident, non-projective settings for both corpora had good results; also by increasing the order, the performance of the parser has been improved. Based on the results obtained, the syntactic and semantic features have improved the performance of the parser. The syntactic features often perform better in comparison with the semantic features, and in some cases, are similar to basic settings. In all settings, the linear combination of all features shows better results.

The feature vector dimensions are reported in table 7. According to tables 6 and 7, although the addition of defined features does not increase the parser accuracy for non-projective first-order setting very much, it has fewer feature dimensions. The combination of features has increased the dimension of feature vector very much. To reduce the feature vector dimensions, the features that have acquired the weight less than threshold, α, in the training stage after the implementation of the MIRA algorithm are removed. Figure 2-a shows the feature removing effect on UAS for PerDT. Figure 2-b shows the feature removing effect on UAS for UPDT, and table 8 shows the feature dimensions by considering the threshold. As shown in figure 2, removing the features with zero-weight did not change the performance of the parser but it had fewer feature dimensions. According to figure 2, by removing the features that had the weigh less than 0.04 for PerDT and 0.02 for UPDT, the precision was not reduced significantly; however, as shown in table 8, the dimensions of the feature vector were reduced.

Table 3. Projective-First-Order.

	PerDT		UPDT	
	UAS	LAS	UAS	LAS
Baseline-Features	80.40	78.53	78.20	71.67
Semantic-Features	75.92	73.15	78.50	71.95
Syntactic-Features	80.54	77. 26	76.35	70.55
All-Features	81.36	80.89	80.00	72.57

Table 4. Non-Projective-First-Order.

	PerDT		UPDT	
	UAS	LAS	UAS	LAS
Baseline-Features	82.34	80.68	84.2	77.86
Semantic-Features	83.90	82.67	83.11	80.16
Syntactic-Features	82.95	81.20	82.21	77.98
All-Features	85.83	81.39	84.20	77.89

Table 5. Projective-Second-Order.

	PerDT		UPDT	
	UAS	LAS	UAS	LAS
Baseline-Features	82.34	80.68	84.2	77.86
Semantic-Features	83.90	82.67	83.11	80.16
Syntactic-Features	82.95	81.20	82.21	77.98
All-Features	85.83	81.39	84.20	77.89

Table 6. Non-Projective-Second-Order.

	PerDT		UPDT	
	UAS	LAS	UAS	LAS
Baseline-Features	85.16	82.99	85.31	83.80
Semantic-Features	85.94	81.12	85.99	84.68
Syntactic-Features	87.147	82.52	83.70	79.89
All-Features	89.17	85.83	88.96	86.25

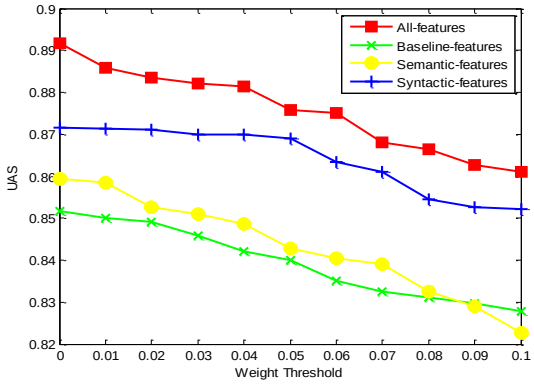
Table 7. Feature Space Dimension.

	PerDT	UPDT
Baseline-1st order	4368125	938649
Baseline-2nd order	6014935	1697835
Our features-1st order	4092103	789641
Our features-2nd order	5869415	1236874
All features-1st order	12578161	4896238
All features-2nd order	17025128	5694502

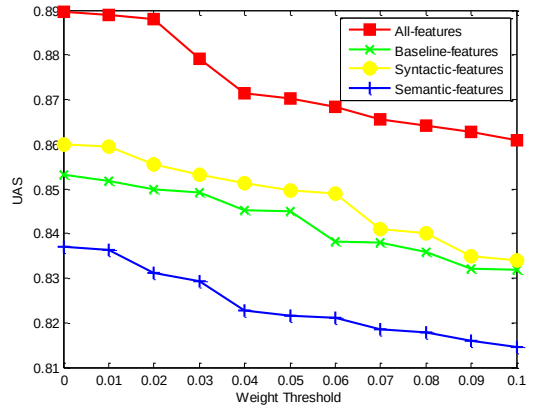
5. Conclusion

In this work, we have stacked the Persian phrase-structure parser for the dependency parser. The phrase-structure parser contains the syntactic information of the sentences that we have shown, in this paper; this information is useful in Persian dependency parser. The parse tree path in the phrase-structure parser was extracted as a syntactic feature, and after clustering, was used as a bit string in MSTParser. To extract the semantic features, the number of proper bits for the string of words derived from the BCA was set for the Persian. The syntactic and semantic features were used as first

and second order in the MSTParser parser and their effects were investigated. Experiments showed that the injection of new features improved the performance of MSTParser for the Persian but this led to make an enormous increase in the feature dimensions. As future research works, the extraction and addition of other features such as the morphological features of word and the provision of methods to reduce the feature vector dimensions, the use of other methods of word clustering based on syntagmatic analysis is proposed.



a. Features removing effect on UAS for PerDT.



b. Features removing effect on UAS for UPDT.

Figure 2. Results of features removing.

Table 8. Effect of feature removing on feature Space Dimension.

	PerDT (Baseline, Our, All)	UPDT (Baseline,Our, All)
0.00	5817596,5348967,15459702	1085647,647510,4458106
0.01	5236894,5187946, 14126307	1024861,506840,3845687
0.02	4678902,4780345, 12945182	947513,464712,3358419
0.03	4387956,4187469,12079466	921312,312509,3114251
0.04	3697581, 3845610, 10844878	913556,289985,2895167
0.05	3284615, 3710769, 9584600	693955,259415,2441963
0.06	3058458, 3187946, 7648521	502574,234169,2094658
0.07	2679428, 2879451, 6487010	328694,198635,1749284
0.08	2567948, 2536473, 5864318	328125,175602,1710208
0.09	1487518, 2140317, 5032875	278130,143692,1364776
0.10	978415, 1125749, 4326151	236115,96152,864822

In the stage of creating syntactic features, we are facing with the sparsity problem, which by providing a solution to solve, it can have a significant effect on reducing the parsing time. Also in this research work, the number of clusters and the number of bits were empirically obtained so the optimal determination of these parameters with suitable algorithms could lead us to better results.

References

[1] Cambria, E. V White, B. (2014). Jumping NLP curves: A review of natural language processing research. IEEE Computational intelligence magazine, vol. 9, pp. 48-57.

[2] Jurafsky, D. C James, H. (2000). Speech and language processing an introduction to natural language processing, computational linguistics, and speech.

[3] Seraji, M. Ginter, F. & Nivre, J. (2016). Universal Dependencies for Persian. Proceedings of Language Resources and Evaluation Conference, pp. 2361-2365.

[4] Ridge, T. (2011). Simple, functional, sound and complete parsing for all context-free grammars. International Conference on Certified Programs and Proofs, pp. 103-118.

[5] Nivre, J. (2005). Dependency grammar and dependency parsing. MSI report, vol. 5133, pp. 1-32.

[6] de Kok, D. & Hinrichs, E. (2016). Transition-based dependency parsing with topological fields. Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, pp. 1-7.

[7] Kübler, S. McDonald, R. & Nivre, J. (2009). Dependency parsing. Synthesis Lectures on Human Language Technologies, vol. 1, pp. 1-127.

[8] Grella, M. (2015). Notes About a More Aware Dependency Parser. arXiv preprint arXiv:1507.05630.

[9] Falavarjani, S. A. M. & Ghassem-Sani, G. (2015). Advantages of Dependency Parsing for Free Word Order Natural Languages. International Conference on Current Trends in Theory and Practice of Informatics, pp. 511-518.

[10] Fishel, M. & Nivre, J. (2009) .Voting and stacking in data-driven dependency parsing.

[11] Zeman, D. & Žabokrtský, Z. (2005). Improving parsing accuracy by combining diverse dependency parsers. Proceedings of the Ninth International Workshop on Parsing Technology, pp. 171-178.

[12] Sagae, K. & Lavie, A. (2006). Parser combination by reparsing. Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers, pp. 129-132.

[13] Nivre, J. & McDonald, R. T. (2008). Integrating Graph-Based and Transition-Based Dependency Parsers. Annual Meeting of the Association for Computational Linguistics, pp. 950-958.

[14] Samuelsson, Y. Täckström, O. Velupillai, S. Eklund, J. Fišel, M. & Saers,M. (2008). Mixing and blending syntactic and semantic dependencies. Proceedings of the Twelfth Conference on Computational Natural Language Learning, pp. 248-252.

[15] McDonald, R. Pereira, F. Ribarov, K. & Hajič, J. (2005). Non-projective dependency parsing using spanning tree algorithms. Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing, pp. 523-530.

[16] Estiri, A. Kahani, M. Hoseini, M. & Asgarian, E. (2012). Designing Persian language parser tool. International Conference on Asian Language Processing.

[17] Seraji, M. Bernd, B. & Nivre,J. (2015). ParsPer: A Dependency Parser for Persian. International Conference on Dependency Linguistics (DepLing 2015), August 24-26, 2015, Uppsala, Sweden, pp. 300-309.

[18] Nivre, J. Hall, J. & Nilsson, J. (2006). Maltparser: A data-driven parser-generator for dependency parsing. Proceedings of Language Resources and Evaluation Conference, pp. 2216-2219.

[19] Bohnet, B. & Kuhn, J. (2012). The best of both worlds: a graph-based completion model for transition-based parsers. Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics, pp. 77-87.

[20] Bohnet, B. & Nivre, J. (2012). A transition-based system for joint part-of-speech tagging and labeled non-projective dependency parsing. Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, pp. 1455-1465.

[21] Martins, A. F. Smith, N. A. Xing, E. P. Aguiar, P. M. & Figueiredo, M. A. (2010). Turbo parsers: Dependency parsing by approximate variational inference. Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing, pp. 34-44.

[22] Ballesteros, M. & Nivre, J. (2012). MaltOptimizer: A System for MaltParser Optimization. Proceedings of Language Resources and Evaluation Conference, pp. 2757-2763.

[23] Seraji, M. Megyesi, B. & Nivre, J. (2012). Dependency parsers for Persian. 24th International Conference on Computational Linguistics, 8-15 December, 2012, Mumbai, India.

[24] Popat, K. (2013). Word Clustering for Data Sparsity: A Literature Survey.

[25] Koo, T. Carreras Pérez, X. & Collins, M. (2008). Simple semi-supervised dependency parsing. 46th Annual Meeting of the Association for Computational Linguistics, pp. 595-603.

[26] Gordon, A. S. & Swanson, R. (2007). Generalizing semantic role annotations across syntactically similar verbs. university of southern california marina del rey ca inst for creative technologies.

[27] Ushioda, A. (1996). Hierarchical clustering of words. Proceedings of the 16th conference on Computational linguistics-Volume 2, pp. 1159-1162.

[28] Rasooli, M. S. Kouhestani, M. & Moloodi, A. (2013). Development of a Persian syntactic dependency treebank. Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pp. 306-314.

[29] Seraji, M. Jahani, C. Megyesi, B. & Nivre, J. (2014). A Persian treebank with Stanford typed dependencies. Proceedings of Language Resources and Evaluation Conference, 2014, 26-31 May, Reykjavik, Iceland, pp. 796-801.

[30] Pakzad, A. & Mianei Bidgo;i, B. (2016). An improved joint model: POS tagging and dependency parsing, Journal of AI and Data Mining, vol. 4, no. 1, pp. 1-8.

[31] Bijankhan, M. (2004). The role of the corpus in writing a grammar: An introduction to a software. Iranian Journal of Linguistics, vol. 19.

نشریه هوش مصنوعی و داده کاوی

طراحی و ساخت ویژگی در تجزیه‌گر وابستگی فارسی

صغری لازمی و حسین ابراهیم‌پور کومله*

گروه مهندسی کامپیوتر- هوش مصنوعی، دانشگاه کاشان، کاشان، ایران

ارسال ۲۰۱۷/۱۰/۱۶؛ بازنگری ۲۰۱۸/۰۱/۳۰؛ پذیرش ۲۰۱۸/۰۳/۱۰

چکیده:

تجزیه‌گر وابستگی یکی از ابزارهای پایه مهم در پردازش زبان طبیعی است که براساس دستور وابستگی به استخراج ساختار جملات و تعیین روابط بین کلمات می‌پردازد. تجزیه‌گر وابستگی، برای زبان‌های بدون ترتیب، مانند زبان فارسی، مناسب است. در این مقاله، تجزیه‌گر وابستگی مبتنی بر داده با کمک تجزیه‌گر مبتنی بر سازه برای زبان فارسی توسعه داده شده است. یکی از فاکتورهای مهم موفقیت هر تجزیه‌گری، فضای ویژگی تعریف شده برای آن می‌باشد. هدف ما، تولید و استخراج ویژگی‌های مناسب برای تجزیه‌ی وابستگی جملات فارسی است. برای نیل به این هدف، ویژگی‌های معنایی و ساختاری جدیدی تعریف شده و با روش پشته‌سازی به تجزیه‌گر وابستگی MSTParser افزوده شده است. ویژگی‌های معنایی بااستفاده از الگوریتم‌های خوشه‌بندی کلمات-مبتنی بر تحلیل زنجیری و ویژگی‌های ساختاری بااستفاده از تجزیه‌گر ساختاری زبان فارسی بدست آمده و بصورت رشته بیت مورد استفاده قرار گرفته‌اند. آزمایشات برروی پیکره وابستگی دادگان و پیکره وابستگی اوپسالا انجام شده است. نتایج بدست آمده حاکی از آنست که تعریف ویژگی‌های جدید باعث بهبود عملکرد تجزیه‌گر وابستگی برای زبان فارسی شده است. معیار یال بدون برچسب بدست آمده برای پیکره‌های مورد استفاده به ترتیب ۸۹/۱۷٪ و ۸۸/۹۶٪ می‌باشد.

کلمات کلیدی: تجزیه‌گر وابستگی، تجزیه‌گر مبتنی بر سازه، MSTParser، پشته‌سازی، زبان فارسی.