# DadmaTools: a Natural Language Processing Toolkit for the Persian Language

**Romina Etezadi, Mohammad Karrabi, Najmeh Zare Maduyieh, Mohammad Bagher Sajadi**, and **Mohammad Taher Pilehvar**
Dadmatech, Tehran, Iran
{roetezadi,karrabi,n_zare,sajadi}@dadmatech.ir

## Abstract

We introduce DadmaTools, an open-source Python Natural Language Processing toolkit for the Persian language. The toolkit is a neural pipeline based on spaCy for several text processing tasks, including normalization, tokenization, lemmatization, part-of-speech, dependency parsing, constituency parsing, chunking, and *ezafe* detecting. DadmaTools relies on fine-tuning of ParsBERT using the PerDT dataset for most of the tasks. Dataset module and embedding module are included in DadmaTools that support different Persian datasets, embeddings, and commonly used functions for them. Our evaluations show that DadmaTools can attain state-of-the-art performance on multiple NLP tasks. The source code is freely available at https://github.com/Dadmatech/DadmaTools.

## 1 Introduction

With the increased accessibility of open-source natural language processing toolkits, users are now able to more easily develop tools that perform sophisticated linguistic tasks. There are several Persian NLP toolkits, such as Stanza (Qi et al., 2020), Hazm[1], Parsivar[2], and jPTDP (Nguyen and Verspoor, 2018). However, they suffer from several limitations. First, most Persian toolkits are based on conventional non-neural models which prevents them from taking full advantage of the recent developments in the filed. Examples include Parsivar (Mohtaj et al., 2018), STeP1 (Shamsfard et al., 2010), Virastar[3], Virastyar[4], and ParsiAnalyzer[5]. Second, most Persian toolkits either do not cover all the basic processing tools (e.g., Perstem[6] and farsiNLPTools) or are not open-source (e.g., FarsiYar[7]). Third, there is no single toolkit that provides state-of-the-art results across different basic tasks. Table 1 lists the toolkits available for Persian NLP along with their task coverage.

We introduce DadmaTools, an open-source Python Natural Language Processing toolkit for Persian. DadmaTools provides the following advantages compared to existing toolkits:

- Using the DadmaTools framework, users can easily download various standard Persian **datasets** and perform a variety of operations on them.
- Several pre-trained static **word embeddings** exits for the Persian language. Many of these embeddings are integrated in the DadmaTools toolkit.
- We evaluated DadmaTools on different Persian datasets, reporting **state-of-the-art** or competitive performance at each step of the pipeline.

Moreover, DadmaTools is based on the spaCy framework which allows users to integrate our toolkit with other piplelines implemented in spaCy. We note that Stanza is a widely used Persian toolkit. Hence, we mainly compare DadmaTools to Stanza. Many of the standard tasks, such as constituency parsing, chunking, and *ezafe* detection[8], are not supported by Stanza for Persian. In addition to covering these tasks, our toolkit also provides support for datasets and word embeddings. DadmaTools is

[1] https://github.com/sobhe/hazm
[2] https://github.com/ICTRC/Parsivar
[3] https://github.com/aziz/virastar
[4] https://github.com/alishakiba/virastyar
[5] https://github.com/NarimanN2/ParsiAnalyzer
[6] https://github.com/jonsafari/perstem
[7] https://www.text-mining.ir/
[8] Ezafe is a grammatical particle in Pesrsian language that links two words together



*Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: System Demonstrations*, pages 124 - 130
July 10-15, 2022 

| Toolkit | Normalizer | Lemma | POS | dependency | Constintuency | Chunker | Ezafe |
|---|---|---|---|---|---|---|---|
| Stanza | ✗ | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ |
| spaCy | ✗ | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ |
| Hazm | ✓ | ✓ | ✓ | ✓ | ✗ | ✓ | ✗ |
| farsiNLPTools (Feely et al., 2014) | ✗ | ✗ | ✓ | ✓ | ✗ | ✗ | ✗ |
| Perstem | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ |
| persianp Toolbox | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ |
| UM-wtlab pos tagger | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ |
| RDRPOSTagger | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ |
| jPTDP | ✗ | ✗ | ✓ | ✓ | ✗ | ✗ | ✗ |
| Parsivar | ✓ | ✗ | ✓ | ✓ | ✗ | ✓ | ✗ |
| text_mining | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ |
| DadmaTools | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

Table 1: Persian NLP toolkits and the corresponding tasks they support.

fully open-source. We hope it can facilitate NLP research and application for the Persian language.

## 2 System Design

DadmaTools is a neural NLP pipeline, but it also includes modules for embeddings and datasets. In this section, we first describe these modules, followed by the main pipeline.

### 2.1 Dataset Module

Popular text processing libraries such as Transformers[9], NLTK (Bird and Loper, 2004), and PyTorchtext have poor support for low-resource language datasets such as Persian. The dataset module of DadmaTools provides a convenient solution for automatic downloading and utilizing of some popular Persian NLP datasets. Each available dataset can be called by a function of the same name and loaded as a generator object. For instance, the Arman (Poostchi et al., 2018) dataset can be loaded with the following lines of code:

```
import dadmatools

# load dataset
arman = dadmatools.datasets.ARMAN()

# working with dataset
len_train = len(arman.train)
test_sample = next(arman.test)
```

DadmaTools allows users to load different sets (e.g., train, test, or dev), if there are any, by using the `<DATASET>.<SET>` format (e.g., `arman.train`). Moreover, the details of the selected dataset can be viewed by using `<DATASET>.info`. DadmaTools comes with a set containing the most commonly used Persian datasets for various tasks, such as text summarization, named entity recognition (NER), spell checking, textual entailment, text classification, sentiment classification, text translation, and universal dependency. There is also a search operation in DadmaTools for finding datasets that belong to specific tasks by using `get_all_datasets_info(tasks=['<task1>', '<task2>',...])`. The list of datasets that are currently included in DadmaTools is shown in Table 2. We will keep integrating new datasets to the toolkit.

### 2.2 Embedding Module

One of the challenges in developing NLP tools for the Persian language is the lack of a library to support different pre-trained embedding models. In order to overcome this challenge, we have developed an embedding module in DadmaTools which provides a variety of public Persian embeddings. For any given embedding space, an object is created which provides a wide range of functions.

```
# download and load embedding
em_name = 'glove-wiki'
embedding = get_embedding(em_name)

# word embedding
vec = embedding(<your_word>)

# sentence embedding
text = <your_text>
t_vec = embedding.embedding_text(text)

# embedding functions
w1 = <word_1>
w2 = <word_2>
similarity_rateembedding.similarity(w1,w2)
top = embedding.top_nearest(10, w1)
```

[9] https://github.com/huggingface/transformers

| Dataset | Task |
|---|---|
| PersianNER[8] | NER |
| ARMAN (Poostchi et al., 2018) | NER |
| PEYMA (Shahshahani et al., 2018) | NER |
| FarsTail (Amirkhani et al., 2020) | Textual Entailment |
| FaSpell[9] | Spell Checking |
| PersianNews (Farahani et al., 2020) | Text Classification |
| PerDT | Universal Dependency |
| PnSummary (Farahani et al., 2021) | Text Summarization |
| SnappfoodSentiment (Farahani et al., 2020) | Sentiment Classification |
| TEP (Pilevar et al., 2011) | Text Translation(eng-fa) |
| WikipediaCorpus | Corpus |
| PersianTweets (Khojasteh et al., 2020) | Corpus |

Table 2: Persian Datasets that are currently integrated in the DadmaTools toolkit.

The details of the corresponding embeddings can be shown with `get_embedding_info(<EMBEDDING>)`. Several functions are present in DadmaTools that can be used for word embeddings, such as finding top nearest neighbours, finding similarity scores between two given words, or getting sentence embedding of a text. Word embeddings that are currently included in DadmaTools are listed in Table 3. Similarly to the datasets module, we will keep updating the list when new embedding models are available.

### 2.3 NLP Pipeline

The pipeline consists of models that range from tokenizing raw text to performing syntactic parsing and high-level task such as NER. The artchitecture of models employed in DadmaTools is mostly based on Stanza (Qi et al., 2020) and ACE (Wang et al., 2021).

**Normalization.** Each sentence can be passed to a normalizer so that different optional procedures can be applied to it, such as whitespace correction, character unification, stopwords/punctuations removal, and email/number/URL replacement. As different settings can be used, this task can be used independently of the pipeline. However, the pipeline uses a default normalizer, which only corrects whitespace and unifies the character.

**Tokenization, Sentence Splitting, and MWT.** As for tokenization and sentence splitting, DadmaTools uses a similar seq2seq architecture to that of Stanza trained on the PerDT dataset.

The tokenizer also identifies whether or not a token is a multi-word token (MWT). Once a word is detected as an MWT, the word is expanded into the underlying syntactic subwords using the MWT Expander.

**Lemmatization.** Lemmatization is the task of converting the input word to its canonical form. For this purpose, we also utilize the seq2seq model presented in the Stanza for lemmatization. However, we manually validate the training dataset and remove wrong or empty instances which results in a better performance (cf. Table 5).

**Part of Speech Tagging.** For each word in a given input text DadmaTools assign a POS tag. To predict the POS tag, we used the ACE model based on ParsBERT pre-trained model (Mehrdad Farahani, 2021) fine-tunned on the PerDT dataset for the sequence labeling task.

**Dependency Parsing.** Similarly to POS tagging, our dependency parsing module is based on the ACE model. In this case, we fine-tuned ParsBERT for dependency parsing on the PerDT dataset.

**Constituency Parsing.** A constituency parse tree breaks a text into sub-phrases. Non-terminals in

[8] https://github.com/Text-Mining/Persian-NER

[9] https://lindat.mff.cuni.cz/repository/xmlui/handle/11372/LRT-1547

[10] https://github.com/Text-Mining/Persian-Wikipedia-Corpus/tree/master/models/glove

[11] https://fasttext.cc/docs/en/crawl-vectors.html

[12] http://vectors.nlpl.eu/repository/

[13] https://commoncrawl.org/

| Embedding | Model | Training corpus |
|---|---|---|
| glove-wiki[10] | GloVe | Wikipedia |
| fasttext-commoncrawl-bin[11] | FastText | CommonCrawl[12] |
| fasttext-commoncrawl-vec[11] | FastText | CommonCrawl |
| word2vec-conll[13] | word2vec | Persian CoNLL17 corpus |

Table 3: Persian word embeddings currently supported by DadmaTools.

| Dataset | # of Instances | Lemma | POS Tags | Dependency Tags | Constituency parses |
|---|---|---|---|---|---|
| Seraji | 6,000 | ✓ | ✓ | ✓ | ✗ |
| PerDT (Rasooli et al., 2020) | 29,107 | ✓ | ✓ | ✓ | ✗ |
| Bijankhan | 83,991 | ✗ | ✓ | ✗ | ✗ |
| Dorsa Treebank (Dehghan et al., 2018) | 30,000 | ✗ | ✗ | ✗ | ✓ |

Table 4: Persian datasets that are currently included in the toolkit.

the tree are types of phrases, the terminals are the words in the sentence, and the edges are unlabeled. To construct the constituency parser, we used the Supar library CRFConstituencyParser[14] using the constituency dataset provided by Dorsa Treebank (Dehghan et al., 2018). It is worth mentioning that the output parse tree tags are different from the tags produced by the POS tagger.

**Chunking.** Chunking is the process of separating and segmenting a sentence into its sub constituents, such as nouns, verbs, etc. We implemented a rule-based chunker. The rules have been written based on words, POS tags, and Dependency tags. The chunking module functions based on around sixty rules.

**Ezafe Detecting.** Ezafe is a grammatical particle in Pesrsian language that links two words together. Ezafe carries valuable information about the grammatical structure of sentences. However, it is not explicitly written in Persian scripts. To create a model to detect ezafe we used the Bijankhan corpus, as it includes ezafe as one of its POS tags. Therefore, we trained the sequence labeling model, ParsBERT, on reprocessed Bijankhan corpus for detecting the ezafe.

## 3 System Usage

It is possible to use the Normalizer directly without triggering the pipeline using the `normalizer` class. The DadmaTools pipline can be also initialized with `pipline` class. By default, only the tokenizer with sentence splitting and MWT are loaded. However, it is possible to load custom processors by adding their names as arguments. The `pipeline` will generate a `Doc` instance, which contains all the raw text's properties regarding the processes that have been called, in the form of the spaCy `Doc`. The following code snippet shows a minimal usage example of DadmaTools.

```
import datamatools.pipeline as pipe

# pipes gets the models e.g.
# pips = 'lem' will only contain
# lemmatizer in pupeline
pips = 'lem,pos,dep,cons'
nlp = pipe.language.Pipeline(pips)

# you can see the pipeline
# with this code
info = nlp.analyze_pipes(pretty=True)
print(info)
# doc is an SpaCy pbject
doc = nlp(<your_text>)
```

DadmaTools is designed to run on different types of hardware (CUDA and CPU). Priority is given to CUDA devices, if available.

## 4 Training Datasets

There are multiple Persian datasets that provide training data for various NLP tasks. Table 4 provides details about some of these dataset. We experimented with these datasets, both in isolation and when combined. Taking these results as our basis, we chose the best models as default for DadmaTools. The merging of different datasets for POS tags and dependencies was carefully evaluated by linguists.

- **Seraji**. This dataset has 6K instances. Our manual validation revealed noisy lemmas in

[14] https://parser.yzhang.site/en/latest/parsers/const.html

| Toolkit | Seraji | | | PerDT | | | PerDT + Seraji | | |
|---|---|---|---|---|---|---|---|---|---|
| | **Dependency** | **POS** | **Lemm.** | **Dependency** | **POS** | **Lemm.** | **Dependency** | **POS** | **Lemm.** |
| Stanza | 87.20 / 83.89 | 97.43 | _ | 93.34 / 91.05 | 97.35 | 98.97 | 84.95 / 80.55 | 88.53 | 96.92 |
| jPTDP | – / 84.07 | 96.66 | – | – | – | – | | | |
| Hazm | – | – | 86.93 | – | – | 89.01 | – | – | 87.95 |
| DadmaTools | **92.5 / 89.23** | **97.83** | – | **95.36 / 92.54** | **97.52** | **99.14** | **92.3 / 88.79** | **96.15** | **97.86** |

Table 5: F1 score percentage for various models on different Persian datasets. For dependency we report two scores, for UAS (Unlabeled Attachment Score) and LAS (Labeled Attachment Score), as UAS/LAS.

the dataset which might be due to its automatic construction procedure.

- **PerDT**. This dataset has almost 30K instances. Thanks to its manual curation by linguists, the dataset is relatively free of annotation errors and mistakes.

- **PerDT + Seraji**. Combining these two datasets was challenging in the case of dependency parsing and POS tagging. We tried to unify tags based on some rules. However, the dataset was not fully unified. Therefore, we did not train the final DadmaTools model based on this combination.

- **Bijankhan**. This dataset has almost 80K instances. The tokenized sentences in this dataset is completely different from the previous ones. However, it has the ezafe tage in its tagset which we used for training the ezafe detection.

- **Dorsa Treebank**. This dataset is a Persian constituency treebank. The treebank was developed by using a bootstrapping approach which converts a dependency structure tree to a phrase structure tree. The annotations are then manually verified. The treebank consists of approximately 30,000 sentences.

## 5 Experiments

Models presented in Table 5 are separately trained on Seraji, PerDT, and combination of both. The evaluations are carried out on corresponding test sets of each dataset. We carried out a set of experiments to compare DadmaTools with other popular toolkits. We opted for Stanza as our main competitor, given that it is the most widely used toolkit for the Persian language. We were limited to compare our toolkit only with those models that were trained on the same datasets (or had public source codes allowing us to train and test on specific datasets). It is worth mentioning that the final models (lemmatizer, POS tagger, and dependacny parser) presented in DadmaTools is based on the PerDT dataset only.

**Lemmatizer.** We compare DadmaTools against Stanza and hazm on the lemmatization task. As shown in Table 5, DadmaTools outperforms the other two tools.

**POS Tagger.** For this experiment, we compared DadmaTools against Stanza given that both models use the same dataset for their training. As shown in Table 5, DadmaTools achieved a better result in predicting the universal tags.

**Dependency Parser.** Similarly to the previous setting, Stanza is our main baseline for dependency parsing, given their training on the same dataset. As shown in Table 5, DadmaTools outperforms Stanza (+1.5% for LAS and +2% in UAS) in predicting the universal tags.

**Constituency Parser.** To train the constituency parser model we used the Dorsa Treebank. One fifth of the treebank was split for testing and the F-score percentage on the test dataset was 82.88. There is no other similar constituency parser model to compare the results with.

**Chunker.** Our chunker is rule-based model that employs nearly eighty rules. There is no gold dataset in the Persian language that would allow us to evaluate the chunking module.

### 5.1 Speed

In order to test the speed of DadmaTools, we combined PerDT and Seraji (PerDT + Seraji) to have a large dataset which would allow us to draw reliable conclusions. Table 6 shows the average run time of models on PerDT + Seraji, computed based on running the models on GPU. Hazm is faster for POS tagging and lemmatizing due to its rule-based nature (as opposed to our neural model).

| Toolkit | Dependency parser | POS tagger | Lemmatizer |
|---|---|---|---|
| Stanza | 0.065 | 0.051 | 0.032 |
| Hazm | 2.404 | 0.001 | 0.000 |
| DadmaTools | 0.027 | 0.029 | 0.038 |

Table 6: Average run time (in seconds) per instance in the PerDT test set, using an Nvidia GeForce RTX 3090 GPU

.

## 6 Conclusion and Future Work

We introduced DadmaTools, an open-source Python Natural Language Processing toolkit for the Persian language. DadmaTools supports different NLP tasks. Moreover, it is based on the spaCy framework which allows users to integrate the toolkit with other processors in a pipeline. As future work, we intend to extend the supported tasks by adding high-level NLP tasks, such as sentiment analysis, entailment, and summarization. We also plan to provide users with the ability to add new datasets and models to the toolkit. We hope that the toolkit can pave the way for research and development for the Persian language.

## References


Hossein Amirkhani, Mohammad AzariJafari, Zohreh Pourjafari, Soroush Faridan-Jahromi, Zeinab Kouhkan, and Azadeh Amirak. 2020. FarsTail: A Persian Natural Language Inference Dataset. *arXiv preprint arXiv:2009.08820*.

Steven Bird and Edward Loper. 2004. NLTK: The natural language toolkit. In *Proceedings of the ACL Interactive Poster and Demonstration Sessions*, pages 214–217, Barcelona, Spain. Association for Computational Linguistics.

Mohammad Hossein Dehghan, Mohammad Molla-Abbasi, and Heshaam Faili. 2018. Toward a multi-representation persian treebank. In *2018 9th International Symposium on Telecommunications (IST)*, pages 581–586. IEEE.

Mehrdad Farahani, Mohammad Gharachorloo, Marzieh Farahani, and Mohammad Manthouri. 2020. ParsBERT: Transformer-based model for persian language understanding. *ArXiv*, abs/2005.12515.

Mehrdad Farahani, Mohammad Gharachorloo, and M. Manthouri. 2021. Leveraging ParsBERT and pretrained mT5 for persian abstractive text summarization. *2021 26th International Computer Conference, Computer Society of Iran (CSICC)*, pages 1–6.

Weston Feely, Mehdi Manshadi, Robert Frederking, and Lori Levin. 2014. The cmu metal farsi nlp approach. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 4052–4055.

Hadi Abdi Khojasteh, Ebrahim Ansari, and Mahdi Bohlouli. 2020. LSCP: Enhanced large scale colloquial persian language understanding. *arXiv preprint arXiv:2003.06499*.

Marzieh Farahani Mohammad Manthouri Mehrdad Farahani, Mohammad Gharachorloo. 2021. Parsbert: Transformer-based model for persian language understanding. *Neural Processing Letters*.

Salar Mohtaj, Behnam Roshanfekr, Atefeh Zafarian, and Habibollah Asghari. 2018. Parsivar: A language processing toolkit for persian. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*.

Dat Quoc Nguyen and Karin Verspoor. 2018. An improved neural network model for joint POS tagging and dependency parsing. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 81–91, Brussels, Belgium. Association for Computational Linguistics.

Mohammad Taher Pilevar, Heshaam Faili, and Abdol Hamid Pilevar. 2011. Tep: Tehran english-persian parallel corpus. In *International Conference on Intelligent Text Processing and Computational Linguistics*, pages 68–79. Springer.

Hanieh Poostchi, Ehsan Zare Borzeshi, and Massimo Piccardi. 2018. BiLSTM-CRF for persian named-entity recognition ArmanPersoNERCorpus: the first entity-annotated persian dataset. In *LREC*.

Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D. Manning. 2020. Stanza: A Python natural language processing toolkit for many human languages. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*.

Mohammad Sadegh Rasooli, Pegah Safari, Amirsaeid Moloodi, and Alireza Nourian. 2020. The persian dependency treebank made universal. *arXiv preprint arXiv:2009.10205*.

Mahsa Sadat Shahshahani, Mahdi Mohseni, Azadeh Shakery, and Heshaam Faili. 2018. PEYMA: A tagged corpus for persian named entities. *arXiv preprint arXiv:1801.09936*.

Mehrnoush Shamsfard, Hoda Sadat Jafari, and Mahdi Ilbeygi. 2010. STeP-1: A set of fundamental tools for persian text processing. In *LREC*.

Xinyu Wang, Yong Jiang, Nguyen Bach, Tao Wang, Zhongqiang Huang, Fei Huang, and Kewei Tu. 2021. Automated Concatenation of Embeddings for Structured Prediction. In *the Joint Conference of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (**ACL-IJCNLP 2021**)*. Association for Computational Linguistics.