Hindi Dependency Parser using Transition-based System

Shweta Singhal and Nishant Agarwal

April 2017

Abstract

All current works on dependency parsers have been on languages other than Indian languages. The most widely spoken among them is Hindi. There is no current baseline dependency parser for Hindi and our work aims to provide an initial attempt at creating a dependency parser for Hindi using existing works done on other languages to provide a baseline system. The work shows the learning of a classifier for use in a greedy, transition-based dependency parser.

1 Introduction

In the recent times there has been a lot of work done in creating dependency parser for many languages. Dependency parsing can be done using feature based deterministic transition-based algorithms and also using graph based algorithm.

The choice of any of the parsing algorithms is dependent on the type of language which are classified as free order or not, thus termed as projective and non-projective respectively. Motivation for dependency parsing comes from their usefulness in understanding bi-lexical relations. Works done in these area have been data-driven and grammar-driven. Our motivation for dependency parsing comes from the need for dependency parsing for Indian languages which have not been ventured into by the community due to lack of dataset, not present earlier. We attempt to showcase the use of ideas from recent works, to create a dependency parser for Hindi which in turn could be used to expand over other Indian languages as well.

2 Dependency Graph

Dependency graphs are syntactic structures over sentences. A *sentence* is a sequence of tokens denoted by: $S = w_0\ w_1\ w_2 \dots w_n$ where, n is sentence length. $w_0 = \text{ROOT}$ is an artificial root token inserted at the beginning of the sentence and does not modify any other token in the sentence. Each token w_i typically represents a word (or token).

A dependency graph $G = (V, A)$ is a labeled directed graph and consists of nodes, V, and arcs, A, such that for sentence $S = w_0 w_1 w_2 \dots w_n$ and label set L the following holds [9]:

- $V \subseteq \{w_0,\ w_1, \ldots, w_n\}$
- $A \subseteq V \times L \times V$
- Arc (w_i, l, w_j) connects head w_i to dependent w_j with label l.

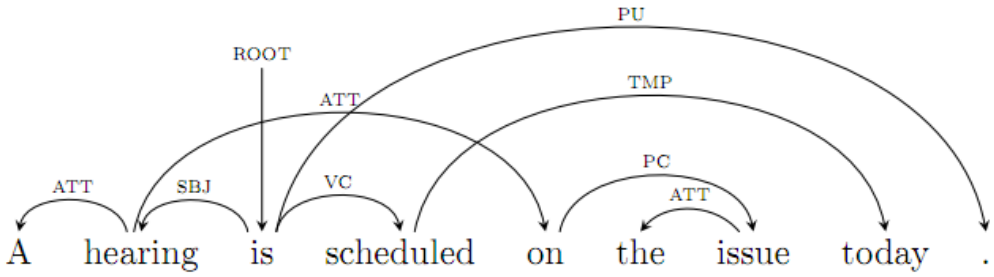

Figure 1: A dependency graph for English sentence [1]

3 Transition-based Dependency Parsing

Transition-based dependency parsing aims to predict a transition sequence from an initial configuration to some terminal configuration, which derives a target dependency parse tree. In the process, the system learn a model for scoring possible transitions and parse by searching for the optimal transition.

We use the arc-standard system from the MaltParser library, slightly different from the StanfordCoreNLP arc-standard system as used in (Chen and Manning, 2014)[7]. In the arc-standard system: A configuration c is terminal if the buffer is empty and the stack contains the single node $ROOT$, and the parse tree is given by A_c.

1. A parser configuration is a triplet $c = (S, Q, A)$, where
   - S = a stack $[\ldots, w_i]$, S of partially processed nodes.
   - Q = a queue $[w_j, \ldots]$, Q of remaining input nodes.
   - A = a set of labeled arcs (w_i, l, w_j).
2. Initialization: $([w_0]_S,\ [w_1, \ldots, w_n]_Q,\ \{\})$ NB: w0 = ROOT
3. Termination: $([w_0]_S,\ []_Q,\ A)$

The arc-standard system defines three types of transitions:

- LEFT-ARC (l) adds an arc $w_j \rightarrow s_1$ with label l and removes w_i from the stack. $([w_0, w_i]_S,\ [w_j, \ldots, w_n]_Q,\ \{\}) \rightarrow ([w_0]_S,\ [w_j, \ldots, w_n]_Q,\ A \bigcup \{(w_j, l, w_i)\})$

- RIGHT-ARC (l) adds an arc $w_i \rightarrow w_j$ with label l. Pops w_i from the stack and puts it into the queue Q. Pops w_j from queue.
  $([w_0, w_i]_S,\ [w_j, \ldots, w_n]_Q,\ \{\}) \rightarrow ([w_0]_S,\ [w_i, \ldots, w_n]_Q,\ A \bigcup \{(w_i, l, w_j)\})$
- SHIFT moves w_i from the queue to the stack. Precondition: $|b| \geq 1$.
  $([w_0, w_i]_S,\ [w_j, \ldots, w_n]_Q,\ \{\}) \rightarrow ([w_0, w_i, w_j]_S,\ [w_{j+1}, \ldots, w_n]_Q,\ A)$

In the labeled version of parsing, there are in total $|T| = 2N_l + 1$ transitions, where N_l is number of different arc labels.The essential goal of a greedy parser is to predict a correct transition from T, based on one given configuration. The classifier uses the features from the Table 1 of (Chen and Manning, 2014)[7].

Given an oracle o that correctly predicts the next transition $o(c)$, parsing is deterministic:

Algorithm 1 Dependency-Parsing Algorithm [5]

1. $PARSE(w_1, \ldots, w_n)$
   - $c \leftarrow ([w_0]_S, [w_1, \ldots, w_n]_Q, \{\})$
   - while $Q_c \neq []$ or $|S_c| > 1$
   - $t \leftarrow o(c)$; t is transition
   - $c \leftarrow t(c)$
   - return $T = (\{w_0, w_1, \ldots, w_n\}, A_c)$

4 Model

In this section we explain the components of the greedy, transition-based dependency parser.

4.1 Training

In earlier Section 3, we have talked about the oracle to learn the transition depending upon the current configuration. An oracle can be approximated by a (linear) multi-class classifier:

$$o(c) = argmax_t\ wf(c, t)$$

where, $f(c, t)$ are the features learned from the configuration c.Features could be over input tokens relative to S and Q, it could be the (partial) dependency tree defined by A, from the (partial) transition sequence or some combination of the current configuration words and their format. We generate training examples $\{(c_i, t_i)\}_{i=1}^{m}$ from the training sentences and their gold parse dependency trees using the arc-standard oracle of MaltParser [4] library,

where c_i is a configuration, $t_i \in T$ is the oracle transition. Each configuration is converted to the feature template from Table 1 of (Chen and Manning, 2014)[7], which are a combination of features described in Zhang and Nivre 2011[11] and Huang et. al[8].

We use the liblinear package[3] to train a multiclass classifier.

4.2 Parsing

We use greedy inference to decode the configurations and apply the transitions. The trained classifier in the previous section, predicts the transition and from the feature template of the current configuration and apply to the current configuration to get the next configuration till the exit condition is not met. In cases of exception where there is a head or dependent node is missing we catch and print the dependency graph as it is for evaluation purposes.

Therefore, in our case the Algorithm 3 will remain the same except the oracle o will get replaced by the trained model.

Algorithm 2 Dependency-Parsing Algorithm after Training [5]

1. $PARSE(w_1, \ldots, w_n)$
   - $c_0 \leftarrow ([w_0]_S, [w_1, \ldots, w_n]_Q, \{\})$
   - while $Q_c \neq []$ or $|S_c| > 1$
   - create features $f(c, t)$ from the c_0
   - Predict next transition t from the learned model from the feature set;
   - Apply transition to get the next configuration, $c \leftarrow t(c)$
   - return $T = (\{w_0, w_1, \ldots, w_n\}, A_c)$
2. write dependency graph of the sentence in the same format as the input sentence into a file.

5 Experiments

5.1 Dataset

We used the Hindi-Urdu Treebank dataset[2] from the Hindi-Urdu Treebank Project. The dataset contains both Hindi and Urdu corpus, available in CoNLL-X and SSF(Shakti Standard Format) format, with utf and wx encodings separately. The wx encoding stands for ROMAN ALPHABETIC CODING SCHEME FOR INDIAN LANGUAGES. We have decided to go with wx encoding and use only the Hindi CoNLL-X format corpus for the

current system evaluation.

Our Hindi dataset consists of Training, Testing and Development set distribution with $933, 131$ & 112 documents in each set respectively, with a range of $6 - 30$ sentences in each document.

Features of CoNLL-x format [6]:

- ID: Token counter, starting at 1 for each new sentence.
- FORM: Word form or punctuation symbol.
- LEMMA: Lemma or stem (depending on the particular data set) of the word form.
- CPOSTAG: Coarse-grained part-of-speech tag, where the tag set is language-specific. It is mapped from POSTAG.
- POSTAG: Fine-grained part-of-speech tag, where the tag set depends on the language, or identical to the coarse-grained part-of-speech tag if not available.
- FEATS: Unordered set of syntactic and/or morphological features (e.g. for some tree-banks temporal and case information).
- HEAD: Head of the current token, which is either a value of ID or zero (0). Depending on the original treebank annotation, there may be multiple tokens with ID= 0.
- DEPREL: Dependency relation to the HEAD. The set of dependency types depends on the particular language.
- PHEAD: Projective head of current token.
- PDEPREL: Dependency relation to the PHEAD.

5.2 Results

Overall in the Hindi corpus dataset we have around $533, 432$ unique features from $333, 829$ training instances. The test data gave $42, 204$ test instances.

On our dataset, we report unlabeled attachment scores (UAS) and labeled attachment scores (LAS). Table 1 shows the comparison of average accuracy after applying Algorithm 4.2 on the Training+Development dataset and using various classifiers from lib-linear partitioned for Token & Sentence level.

Table 2 shows the comparison of average accuracy after applying Algorithm 4.2 on the Training dataset and using various classifiers from lib-linear partitioned for Token & Sentence level.

Classifiers	**UAS**		**LAS**	
	Token	**Sentence**	**Token**	**Sentence**
L_2-regularized logistic regression (dual)	0.766	0.751	0.613	0.624
L_2-regularized logistic regression (primal)	0.752	0.766	0.614	0.625
L_1-regularized logistic regression	0.769	0.784	0.620	**0.629**
L_2-regularized L_2-loss SVC (dual)	0.758	0.773	0.598	0.610
L_2-regularized L_2-loss SVC (primal)	0.766	0.781	0.606	0.618
L_2-regularized L_1-loss SVC (dual)	**0.776**	**0.790**	**0.616**	0.627
L_1-regularized L_2-loss SVC	0.755	0.775	0.572	0.585

Table 1: UAS and LAS score on Training+Development dataset.

Classifiers	**UAS**		**LAS**	
	Token	**Sentence**	**Token**	**Sentence**
L_2-regularized logistic regression (dual)	0.750	0.765	0.612	0.623
L_2-regularized logistic regression (primal)	0.750	0.764	0.612	0.623
L_1-regularized logistic regression	0.762	0.777	**0.618**	**0.629**
L_2-regularized L_2-loss SVC (dual)	0.759	0.774	0.597	0.609
L_2-regularized L_2-loss SVC (primal)	0.768	0.783	0.607	0.617
L_2-regularized L_1-loss SVC (dual)	**0.773**	**0.787**	0.613	0.623
multi-class SVC by Crammer and Singer	0.730	0.746	0.582	0.594
L_1-regularized L_2-loss SVC	0.757	0.773	0.569	0.583

Table 2: UAS and LAS score on Training dataset.

6 Conclusions and Future Work

The dependency parser we present is an initial attempt at getting the ball rolling for Indian language. This was an immense learning experience. The system implemented has proven well for projective dependency graphs and our dataset set has a mix of both projective and non-projective graphs. Thus, we suspect that out system's accuracy may have suffered from it. Our system can work right away on various languages for which CoNLL-X format dataset is available.

For future work, we would like to use the arc-standard algorithm of StanfordCoreNLP since that matches well with the algorithm in [7]. We would also like to try out Beam-search algorithm for inference with various beam size. Currently, our system uses the basic features and we plan to incorporate more rich features such as direction, valency, etc as suggested in [7]. For now, our system is tightly based on (deterministic) transition-based parser MaltParser [4], we will in future work through may be graph-based dependency parser MSTParser [10]. MaltParser is more accurate on short dependencies and disambiguation of core grammatical functions with deterministic, local learning, where, MSTParser is more accurate on long dependencies and dependencies near the root of the tree with exact, global learning.

We also plan to evaluate our model on Urdu dataset. We have not found any other dependency parser for Hindi dataset as a means for comparison, but will work through the details of our system to make it more robust and compatible to all the languages which are not yet explored.

Current system is feature-based approach of the dependency parser technique called Transition-based systems, in future we will love to work with deep learning models to learn appropriate features from data by themselves. Initially, when we were doing the feasibility study of whether the parser could be designed using deep learning models, we were constrained by the dependency on word-embeddings, which for Hindi-Urdu languages are not yet available. We want to create our own word embeddings for these languages and make them publicly available for academic purposes.

References

[1] Dependency tree for english sentence. https://commons.wikimedia.org/wiki/File:Latex-dependency-parse-example-with-tikz-dependency.png.

[2] Hindi-urdu dependency treebanks (hutb). http://ltrc.iiit.ac.in/hutb_release/.

[3] liblinear. http://liblinear.bwaldvogel.de/.

[4] Maltparser. http://www.maltparser.org.

[5] Parsing algorithm. https://stp.lingfil.uu.se/~nivre/docs/BeyondMaltParser.pdf.

[6] Sabine Buchholz and Erwin Marsi. Conll-x shared task on multilingual dependency parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*, CoNLL-X '06, pages 149–164, Stroudsburg, PA, USA, 2006. Association for Computational Linguistics.

[7] Danqi Chen and Christopher D Manning. A fast and accurate dependency parser using neural networks. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

[8] Liang Huang, Wenbin Jiang, and Qun Liu. Bilingually-constrained (monolingual) shift-reduce parsing. In *Proceedings of EMNLP*, 2009.

[9] Sandra Kubler, Ryan McDonald, Joakim Nivre, and Graeme Hirst. *Dependency Parsing*. Morgan and Claypool Publishers, 2009.

[10] Ryan McDonald, Kevin Lerman, and Fernando Pereira. Multilingual dependency analysis with a two-stage discriminative parser. In *IN PROCEEDINGS OF THE CONFERENCE ON COMPUTATIONAL NATURAL LANGUAGE LEARNING (CONLL*, pages 216–220, 2006.

[11] Yue Zhang and Joakim Nivre. Transition-based dependency parsing with rich non-local features. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: Short Papers - Volume 2*, HLT '11, pages 188–193, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.