CS 6961 : Structural Prediction

Hindi Dependency Parser using Transition-based System

Shweta Singhal and Nishant Agarwal

Introduction

There is no current baseline dependency parser for Hindi. Our main aim in this project is to provide a dependency parser for Hindi using greedy inference over learned classifier from transition-based system.

Data Description

We used the Hindi-Urdu Treebank dataset[1] of CoNLL-x format from the Hindi-Urdu Treebank Project.
Hindi data consists of Training, Testing and Development set distribution with 933, 131 & 112 documents respectively, with a range of $6 - 30$ sentences in each document .

Features of CoNLL-x format [3]:

- ID: Token counter.
- FORM: Word form or punctuation symbol.
- LEMMA: Lemma or stem of the word form.
- CPOSTAG: Coarse-grained part-of-speech tag.
- POSTAG: Fine-grained part-of-speech tag.
- FEATS: Unordered set of syntactic and/or morphological features.
- HEAD: Head of the current token, which is either a value of ID or zero (0).
- DEPREL: Dependency relation to the HEAD.
- PHEAD: Projective head of current token.
- PDEPREL: Dependency relation to the PHEAD.

Dependency Grpah

A dependency graph $G = (V, A)$ is a labeled directed graph, where V, nodes, and A, arcs.

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

Figure 1 shows a dependency graph for Hindi sentence: **rAma skUla jAkara Gara A gayA.**

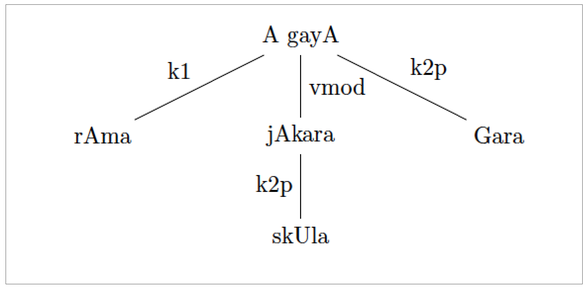

Figure 1: Dependency-graph of a Hindi sentence.

Transition-based Dependency Parser

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: $([w_0]_S,\ []_Q,\ A)$

Transition

- LEFT-ARC(l): adds an arc $b_1 \rightarrow s_1$ with label l and removes s_1 from the stack.
- RIGHT-ARC(l): adds an arc $s_1 \rightarrow b_1$ with label l. Pops s_1 from the stack and puts it into the buffer b. Pops b_1 from buffer.
- SHIFT: moves b_1 from the buffer to the stack. Precondition: $|b| \geq 1$.

Model

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.
In our model, we have used features defined in Paper [4].

Training

We generate training examples $\{(c_i, t_i)\}_{i=1}^{m}$ from the training sentences.
The gold parse dependency tree is generated from the arc-standard oracle of MaltParser [2] library.

Parsing

We use greedy inference to decode the configurations and apply the transitions.

Algorithm

We use greedy inference to decode the configurations and apply the transitions.

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.

Results

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.

Conclusions

This project was an immense learning experience for both of us. Our system can work right away on various languages for which $CoNLL - x$ format dataset is available. Hindi has a mix of projective and non-projective graphs, we need to account for non-projective as well. Our current system's accuracy suffers as we have used a projective based approach.

Future Work

- Beam-search based inference.
- Urdu data evaluation using the current model.
- Incorporate more rich features to improve accuracy.
- Current system is based upon transition based parsing, will evaluate graph-based parsing too.
- We will create word embeddings for Hindi-Urdu languages so that we can use deep learning based models to learn features from the data itself.

References

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

[2] Maltparser. `http://www.maltparser.org`.

[3] 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.

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

Advisor: Vivek Srikumar