DeSRL: A Linear-Time Semantic Role Labeling System

Massimiliano Ciaramita[†*]
massi@yahoo-inc.com

Giuseppe Attardi[‡]
attardi@di.unipi.it

Felice Dell'Orletta[‡]
dellorle@di.unipi.it

Mihai Surdeanu[†,⋄]
mihai.surdeanu@barcelonamedia.org

†: Yahoo! Research Barcelona, Ocata 1, 08003, Barcelona, Catalunya, Spain
‡: Dipartimento di Informatica, Università di Pisa, L. B. Pontecorvo 3, I-56127, Pisa, Italy
⋄: Barcelona Media Innovation Center, Ocata 1, 08003, Barcelona, Catalunya, Spain

Abstract

This paper describes the DeSRL system, a joined effort of Yahoo! Research Barcelona and Università di Pisa for the CoNLL-2008 Shared Task (Surdeanu et al., 2008). The system is characterized by an efficient pipeline of linear complexity components, each carrying out a different sub-task. Classifier errors and ambiguities are addressed with several strategies: revision models, voting, and reranking. The system participated in the closed challenge ranking third in the complete problem evaluation with the following scores: 82.06 labeled macro F1 for the overall task, 86.6 labeled attachment for syntactic dependencies, and 77.5 labeled F1 for semantic dependencies.

1 System description

DeSRL is implemented as a sequence of components of linear complexity relative to the sentence length. We decompose the problem into three subtasks: parsing, predicate identification and classification (PIC), and argument identification and classification (AIC). We address each of these subtasks with separate components without backward feedback between sub-tasks. However, the use of multiple parsers at the beginning of the process, and re-ranking at the end, contribute beneficial stochastic aspects to the system. Figure 1 summarizes the system architecture. We detail the parsing sub-task in Section 2 and the semantic sub-tasks (PIC and AIC) in Section 3.

2 Parsing

In the parsing sub-task we use a combination strategy on top of three individual parsing models, two developed in-house –DeSR$_{left-to-right}$ and DeSR$^{revision}_{right-to-left}$– and a third using an off-the-shelf parser, Malt 1.0.0[1].

2.1 DeSR$_{left-to-right}$

This model is a version of DeSR (Attardi, 2006), a deterministic classifier-based $Shift/Reduce$ parser. The parser processes input tokens advancing on the input from left to right with $Shift$ actions and accumulates processed tokens on a stack with $Reduce$ actions. The parser has been adapted for this year's shared task and extended with additional classifiers, e.g., Multi Layer Perceptron and multiple SVMs.[2]

The parser uses the following features:

1. SPLIT_LEMMA: from tokens -1, 0, 1, $prev(0)$, $leftChild(0)$, $rightChild(0)$
2. PPOSS: from -2, -1, 0, 1, 2, 3, $prev(0)$, $next(-1)$, $leftChild(-1)$, $leftChild(0)$, $rightChild(-1)$, $rightChild(0)$
3. DEPREL: from $leftChild(-1)$, $leftChild(0)$, $rightChild(-1)$
4. HDIST: from -1, 0

In the above list negative numbers refer to tokens on the stack, positive numbers to tokens in the input queue. We use the following path operators: $leftChild(x)$ refers to the leftmost child of token x, $rightChild(x)$ to the rightmost child of token x, $prev(x)$ and $next(x)$ respectively to the token preceding or following x in the sentence.

[*] All authors contributed equally to this work.

[*] © 2008. Licensed under the *Creative Commons Attribution-Noncommercial-Share Alike 3.0 Unported* license (http://creativecommons.org/licenses/by-nc-sa/3.0/). Some rights reserved.

[1] http://w3.msi.vxu.se/~nivre/research/MaltParser.html

[2] This parser is available for download at: http://sourceforge.net/projects/desr/.

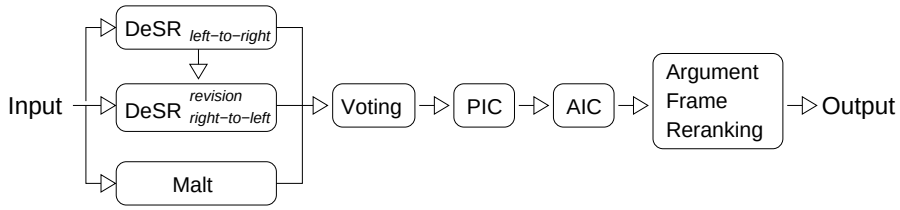

Figure 1: DeSRL system architecture.

The first three types of features are directly extracted from the attributes of tokens present in the training corpus. The fourth feature represents the distance of the token to the head of the noun phrase to which it belongs, or "O" if it does not belong to a noun phrase. This distance is computed with a simple heuristic, based on a pattern of POS tags. Attardi and Dell'Orletta (2008) have shown that this feature improves the accuracy of a shift/reduce dependency parser by providing approximate information about NP chunks in the sentence. In fact no token besides the head of a noun phrase can have a head referring to a token outside the noun phrase. Hence the parser can learn to avoid creating such links. The addition of this feature yields an increase of 0.80% in Labeled Accuracy on the development set.

2.2 Revision Parser: DeSR$^{revision}_{right-to-left}$

Our second individual parsing model implements an alternative to the method of revising parse trees of Attardi and Ciaramita (2007) (see also (Hall & Novak, 2005)). The original approach consisted in training a classifier to revise the errors of a baseline parser. The approach assumed that only local revisions to the parse tree would be needed, since the dependency parser mostly gets individual phrases correctly. The experiments showed that indeed most of the corrections can be expressed by a small set of (about 20) complex movement rules. Furthermore, there was evidence that one could get higher improvements from the tree revision classifier if this was trained on the output of a lower accuracy parser. The reason for this is that the number of errors is higher and this provides a larger amount of training data.

For the CoNLL 2008 shared task, we refined this idea, but instead of using an independent classifier for the revision, we use the parser itself. The second parser is trained on the original corpus extended with dependency information predicted by a lower accuracy parser. To obtain the base parser we use DeSR trained on half the training corpus using a Maximum Entropy (ME) classifier. The ME classifier is considerably faster to train but has a lower accuracy: this model achieved an LAS of 76.49% on the development set. Using the output of the ME-based parser we extend the original corpus with four additional columns: the lemma of the predicted head (`PHLEMMA`), the PPOSS of the predicted head (`PHPPOSS`), the dependency of the predicted head (`PHDEPREL`), and the indication of whether a token appears before or after its predicted head. A second parser is trained on this corpus, scanning sentences from right to left and using the following additional features:

1. `PHPPOSS`: from -1, 0
2. `PHLEMMA`: from -1, 0
3. `PHDEPREL`: from -1, 0
4. `PHHDIST`: from 0

Performing parsing in reverse order helps reduce several of the errors that a deterministic parser makes when dependency links span a long distance in the input sequence. Experiments on the CoNLL 2007 corpora (Dell'Orletta, 2008) have shown that this indeed occurs, especially for distances in the range from 6 to 23. In particular, the most significant improvements are for dependencies with label `COORD` (+ 6%) and `P` (+ 8%).

The revision parser achieves an LAS of 85.81% on the development set. Note that the extra features from the forward parser are indeed useful, since a simple backward parser only achieves 82.56% LAS on the development set.

2.3 Parser Combination

The final step consists in combining the outputs of the three individual models a simple voting scheme: for each token we use majority voting to select its head and dependency label. In case of ties, we chose the dependency predicted by our overall best individual model (DeSR$^{revision}_{right-to-left}$).[3]

Note that typical approaches to parser combination combine the outputs of independent parsers, while in our case one base model (DeSR$^{revision}_{right-to-left}$) is trained with

[3] We tried several voting strategies but none performed better.

information predicted by another individual model(DeSR$_{left-to-right}$). To the best of our knowledge, combining individual parsing models that are inter-dependent is novel.

3 Semantic Role Labeling

We implement the Semantic Role Labeling (SRL) problem using three components: PIC, AIC, and reranking of predicted argument frames.

3.1 Predicate Identification and Classification

The PIC component carries out the identification of predicates, as well as their partial disambiguation, and it is implemented as a multiclass average Perceptron classifier (Crammer & Singer, 2003). For each token i we extract the following features ($\langle,\rangle$ stands for token combination):

1. SPLIT_LEMMA: from $\langle i-1,i\rangle, i-1, i, i+1, \langle i,i+1\rangle$
2. SPLIT_FORM: from $i-2, i-1, i, i+1.i+2$
3. PPOSS: from $\langle i-2,i-1\rangle, \langle i-1,i\rangle, i-1, i, i+1, \langle i,i+1\rangle, \langle i+1,i+2\rangle$
4. WORD_SHAPE: e.g., "Xx*" for "Brazil", from $\langle i-2,i-1,i\rangle, \langle i-1,i\rangle, i-1, i, i+1, \langle i,i+1\rangle, \langle i,i+1,i+2\rangle$
5. Number of children of node i
6. For each children j of i: split_lemma$_j$, pposs$_j$, deprel$_{i,j}$, $\langle$split_lemma$_i$, split_lemma$_j\rangle$, $\langle$pposs$_i$, pposs$_j\rangle$
7. Difference of positions: $j-i$, for each child j of i.

The PIC component uses one single classifier mapping tokens to one of 8 classes corresponding to the rolesets suffixes 1 to 6, the 6 most frequent types, plus a class grouping all other rolesets, and a class for non predicates; i.e., $Y=\{0,1,2,..,7\}$. Each token classified as y_7 is mapped by default to the first sense y_1. This approach is capable of distinguishing between different predicates based on features 1 and 2, but it can also exploit information that is shared between predicates due to similar frame structures. The latter property is intuitively useful especially for low-frequency predicates.

The classifier has an accuracy in the multiclass problem, considering also the mistakes due to the non-predicted classes, of 96.2%, and an F-score of 92.7% with respect to the binary predicate identification problem. To extract features from trees (5-7) we use our parser's output on training, development and evaluation data.

3.2 Argument Identification and Classification

Algorithm 1 describes our AIC framework. The algorithm receives as input a sentence $\mathbf{S}$ where predicates have been identified and classified using the PIC component, an inference strategy $\mathcal{I}$ is used to guarantee that the generated best frames satisfy the domain constraints, plus an AIC classification model $\mathbf{w}$. We learn $\mathbf{w}$ using a multiclass Perceptron, using as output label set $\mathcal{Y}$ all argument labels that appear more than 10 times in training plus a nil label assigned to all other tokens.

Algorithm 1: AIC

```
input : sentence S; inference strategy I; model w
foreach predicate p in S do
    set frame F_in = {}
    foreach token i in S do
        if validCandidate(i) then
            ŷ = arg max_{y∈Y} score(Φ(p,i), w, y)
            if ŷ ≠ nil then
                add argument (i,ŷ) to F_in
    F_out = inference(F_in, I)
output: set of all frames F_out
```

During both training and evaluation we select only the candidate tokens that pass the $validCandidate$ filter. This function requires that the length of the dependency path between predicate and candidate argument be less than 6, the length of the dependency path between argument and the first common ancestor be less than 3, and the length of the dependency path between the predicate and the first common ancestor be less than 5. This heuristic covers over 98% of the arguments in training.

In the worst case, Algorithm 1 has quadratic complexity in the sentence size. But, on average, the algorithm has linear time complexity because the number of predicates per sentence is small (averaging less than five for sentences of 25 words).

The function Φ generates the feature vector for a given predicate-argument tuple. Φ extracts the following features from a given tuple of a predicate p and argument a:

1. $token(a)$[4], $token(\text{modifier of } a)$ if a is the head of a prepositional phrase, and $token(p)$.
2. Patterns of PPOSS tags and DEPREL labels for: (a) the predicate children, (b) the children of the predicate ancestor across VC and IM dependencies, and (c) the siblings of the same ancestor. In all paths we mark the position of p, a and any of their ancestors.
3. The dependency path between p and a. We add three versions of this feature: just the

[4] $token$ extracts the split lemma, split form, and PPOSS tag of a given token.

path, and the path prefixed with p and a's PPOSS tags or split lemmas.

4. Length of the dependency path.
5. Distance in tokens between p and a.
6. Position of a relative to p: before or after.

We implemented two inference strategies: *greedy* and *reranking*. The greedy strategy sorts all arguments in a frame $\mathbf{F_{in}}$ in descending order of their scores and iteratively adds each argument to the output frame $\mathbf{F_{out}}$ only if it respects the domain constraints with the other arguments already selected. The only domain constraint we use is that core arguments cannot repeat.

3.3 Reranking of Argument Frames

The reranking inference strategy adapts the approach of Toutanova et al. (2005) to the dependency representation with notable changes in candidate selection, feature set, and learning model. For candidate selection we modify Algorithm 1: instead of storing only $\hat{y}$ for each argument in $\mathbf{F_{in}}$ we store the top k best labels. Then, from the arguments in $\mathbf{F_{in}}$, we generate the top k frames with the highest score, where the score of a frame is the product of all its argument probabilities, computed as the $softmax$ function on the output of the Perceptron. In this set of candidate frames we mark the frame with the highest F_1 score as the positive example and all others as negative examples.

From each frame we extract these features:

1. Position of the frame in the set ordered by frame scores. Hence, smaller positions indicate candidate frames that the local model considered better (Marquez et al., 2007).
2. The complete sequence of arguments and predicate for this frame (Toutanova, 2005). We add four variants of this feature: just the sequence and sequence expanded with: (a) predicate voice, (b) predicate split lemma, and (c) combination of voice and split lemma.
3. The complete sequence of arguments and predicate for this frame combined with their PPOSS tags. Same as above, we add four variants of this feature.
4. Overlap with the PropBank or NomBank frame for the same predicate lemma and sense. We add the precision, recall, and F_1 score of the overlap as features (Marquez et al., 2007).
5. For each frame argument, we add the features from the local AIC model prefixed with the corresponding argument label in the current frame (Toutanova, 2005).

The reranking classifier is implemented as multilayer perceptron with one hidden layer of 5 units, trained to solve a regression problem with a least square criterion function. Previously we experimented, unsuccessfully, with a multiclass Perceptron and a ranking Perceptron. The limited number of hidden units guarantees a small computational overhead with respect to a linear model.

	WSJ + Brown	WSJ	Brown
Labeled macro F_1	82.69	83.83	73.51
LAS	87.37	88.21	80.60
Labeled F_1	78.00	79.43	66.41

Table 1: DeSRL results in the closed challenge, for the overall task, syntactic dependencies, and semantic dependencies.

	Devel	WSJ	Brown
$\text{DeSR}_{left-to-right}$	85.61	86.54	79.74
$\text{DeSR}^{revision}_{right-to-left}$	85.81	86.19	78.91
MaltParser	84.10	85.50	77.06
Voting	87.37	88.21	80.60

Table 2: LAS of individual and combined parsers.

4 Results and Analysis

Table 1 shows the overall results of our system in the closed challenge. Note that these scores are higher than those of our submitted run mainly due to improved parsing models (discussed below) whose training ended after the deadline. The score of the submitted system is the third best for the complete task. The system throughput in our best configuration is 28 words/second, or 30 words/second without reranking. In exploratory experiments on feature selection for the re-ranking model we found that several features classes do not contribute anything and could be filtered out speeding up significantly this last SRL step. Note however that currently over 90% of the runtime is occupied by the syntactic parsers' SVM classifiers. We estimate that we can increase throughput one order of magnitude simply by switching to a faster, multiclass classifier in parsing.

4.1 Analysis of Parsing

Table 2 lists the labeled attachment scores (LAS) achieved by each parser and by their combination on the development set, the WSJ and Brown test sets. The results are improved with respect to the official run, by using a revision parser trained on the output of the lower accuracy ME parser, as

			Labeled F_1			Unlabeled F_1		
Syntax	PIC	Inference	Devel	WSJ	Brown	Devel	WSJ	Brown
gold	gold	greedy	88.95	90.21	84.95	93.71	94.34	93.29
predicted	gold	greedy	85.96	86.70	78.68	90.60	90.98	88.02
predicted	predicted	greedy	79.88	79.27	66.41	86.07	85.33	80.14
predicted	predicted	reranking	80.13	79.43	66.41	86.33	85.62	80.41

Table 3: Scores of the SRL component under various configurations.

	Devel	WSJ	Brown
Unlabeled F_1	92.69	90.88	86.96
Labeled F_1 (PIC)	87.29	84.87	71.99
Labeled F_1 (Sense 1)	79.62	78.94	70.11

Table 4: Scores of the PIC component.

mentioned earlier. These results show that voting helps significantly (+1.56% over the best single parser) even though inter-dependent models were used. However, our simple voting scheme does not guarantee that a well-formed tree is generated, leaving room for further improvements; e.g., as in (Sagae & Lavie, 2006).

4.2 Analysis of SRL

Table 3 shows the labeled and unlabeled F_1 scores of our SRL component as we move from gold to predicted information for syntax and PIC. For the shared task setting –predicted syntax and predicted PIC– we show results for the two inference strategies implemented: greedy and reranking. The first line in the table indicates that the performance of the SRL component when using gold syntax and gold PIC is good: the labeled F_1 is 90 points for the in-domain corpus and approximately 85 points for the out-of-domain corpus. Argument classification suffers the most on out-of-domain input: there is a difference of 5 points between the labeled scores on WSJ and Brown, even though the corresponding unlabeled scores are comparable.

The second line in the table replicates the setup of the 2005 CoNLL shared task: predicted syntax but gold PIC. This yields a moderate drop of 3 labeled F_1 points on in-domain data and a larger drop of 6 points for out-of-domain data.

We see larger drops when switching to predicted PIC (line 3): 5-6 labeled F_1 points in domain and 12 points out of domain. This drop is caused by the PIC component, e.g., if a predicate is missed the whole frame is lost. Table 4 lists the scores of our PIC component, which we compare with a baseline system that assigns sense 1 to all identified predicates. The table indicates that, even though our disambiguation component improves significantly over the baseline, it performs poorly, especially on out-of-domain data. Same as SRL, the classification sub-task suffers the most out of domain (there is a difference of 15 points between unlabeled and labeled F_1 scores on Brown).

Finally, the reranking inference strategy yields only modest improvements (last line in Table 3). We attribute these results to the fact that, unlike Toutanova et al. (2005), we use only one tree to generate frame candidates, hence the variation in the candidate frames is small. Considering that the processing overhead of reranking is already large (it quadruples the runtime of our AIC component), we do not consider reranking a practical extension to a SRL system when processing speed is a dominant requirement.

References

G. Attardi. 2006. Experiments with a Multilanguage Non-Projective Dependency Parser. In *Proc. of CoNNL-X 2006*.

G. Attardi and M. Ciaramita. 2007. Tree Revision Learning for Dependency Parsing. In *Proc. of NAACL/HLTC 2007*.

G. Attardi, F. Dell'Orletta. 2008. Chunking and Dependency Parsing. In *Proc. of Workshop on Partial Parsing*.

K. Crammer and Y. Singer. 2003. *Ultraconservative Online Algorithms for Multiclass Problems.* Journal of Machine Learning Research 3: pp.951-991.

F. Dell'Orletta. 2008. Improving the Accuracy of Dependency Parsing. PhD Thesis. *Dipartimento di Informatica, Università di Pisa*, forthcoming.

K. Hall and V. Novak. 2005. Corrective Modeling for Non-Projective Dependency Parsing. In *Proc. of IWPT*.

L. Marquez, L. Padro, M. Surdeanu, and L. Villarejo. 2007. UPC: Experiments with Joint Learning within SemEval Task 9. In *Proc. of SemEval 2007*.

K. Sagae and A. Lavie. 2006. Parser Combination by reparsing. In *Proc. of HLT/NAACL*.

M. Surdeanu, R. Johansson, A. Meyers, L. Màrquez and J. Nivre. 2008. The CoNLL-2008 Shared Task on Joint Parsing of Syntactic and Semantic Dependencies. In *Proc. of CoNLL-2008*.

K. Toutanova, A. Haghighi, and C. Manning. 2005. Joint Learning Improves Semantic Role Labeling. In *Proc. of ACL*.