# Latent SVMs for Semantic Role Labeling using LTAG Derivation Trees[*]

**Yudong Liu** and **Gholamreza Haffari** and **Anoop Sarkar**
School of Computing Science
Simon Fraser University, British Columbia, Canada
{yudongl,ghaffar1,anoop}@cs.sfu.ca

## Abstract

A phrase structure parse tree for a sentence can be generated by many different Lexicalized Tree-Adjoining Grammar (LTAG) derivation trees. In this paper, we use multiple LTAG derivations as latent features for semantic role labeling (SRL). We hypothesize that positive and negative examples of individual semantic roles can be reliably distinguished by possibly different latent LTAG-based features. We use latent support vector machines (LSVM) for the SRL task using these latent LTAG features. Our experiments on the PropBank-CoNLL'2005 dataset show that our method obtains 89.59% F1 score which is significantly better than the state of the art F1 score of 80.32% on this task.

## 1 Introduction

Semantic Role Labeling (SRL) aims to identify and label all the arguments for each predicate occurring in a sentence. It involves identifying constituents in the sentence that represent the predicate's arguments and assigning pre-specified semantic roles to them. The following example shows the semantic roles for the predicate *sell*, taken from the PropBank corpus (Palmer et al., 2005).

[A0$_{seller}$ *Ports of Call Inc.*] reached agreements to [V *sell*] [A1$_{\text{thing}}$ *its remaining seven aircraft*] [A2$_{\text{buyer}}$ *to buyers that weren't disclosed*] .

As in other natural language learning tasks, feature selection plays an important role in high performance SRL. SRL feature extraction has relied on various syntactic representations of input sentences, such as syntactic chunks (Hacioglu et al., 2004) and full syntactic parses (Gildea and Jurafsky, 2002), CCG derivations (Gildea and Hockenmaier, 2003) and dependency trees (Hacioglu, 2004) or integrated features from different parsers (Pradhan et al., 2005). Lexicalized Tree Adjoining Grammar (LTAG) derivation trees have been used for SRL (Chen and Rambow, 2003; Liu and Sarkar, 2007) who show that LTAG can localize predicate-argument dependencies to improve performance on the SRL task.

The LTAG derivations used in (Liu and Sarkar, 2007) were obtained by applying Magerman-Collins head percolation rules to Penn treebank (PTB) trees. However, more than one plausible LTAG derivation can be extracted from a single PTB tree. In this paper, we consider multiple LTAG derivations as a source of features for the SRL task. Since there are multiple LTAG derivations available for the single PTB tree, any single derivation tree extracted using one approach is not necessarily the best feature source for the SRL instance in question. By treating LTAG derivations as latent structures of a PTB tree, we use latent SVMs (LSVM) to learn which is the most suitable LTAG derivation for each SRL prediction instance.

## 2 Lexicalized Tree Adjoining Grammars

We assume some familiarity with Lexicalized Tree-Adjoining Grammar (LTAG); (Joshi and Schabes, 1997) is a good introduction to this formalism. A LTAG is defined to be a set of lexicalized *elementary trees* (*etree* for short), of which there are two types, *initial trees* and *auxiliary trees*. Etrees are composed through two operations, *substitution* and *adjunction*. The tree produced by composing the etrees is the *derived/parse tree* and the tree that records the history of composition is the *derivation tree* as shown in Fig. 1.

### 2.1 Generation of latent LTAG derivations

As in previous work on LTAG derivation tree extraction from the Penn TreeBank (Shen, 2004), we face two kinds of ambiguities in LTAG elementary tree extraction: *head choice* and *argument-adjunct*

[*] This research was partially supported by NSERC, Canada (RGPIN: 264905). Thanks to Mirella Lapata, Kristina Toutanova, Luke Zettlemoyer and the anonymous reviewers for their helpful comments.

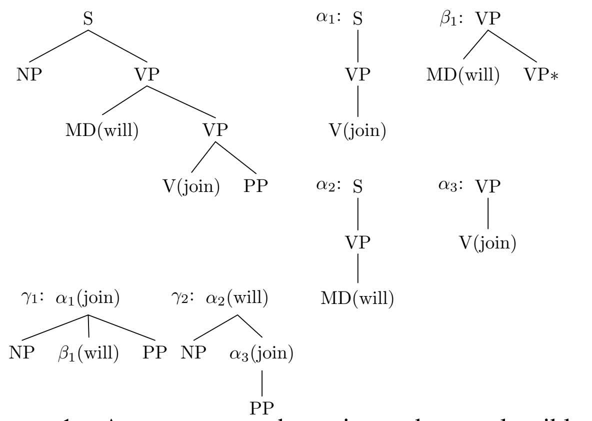

Figure 1: A parse tree schematic, and two plausible LTAG derivation trees for it: derivation tree $\gamma_1$ uses elementary trees $\alpha_1$ and $\beta_1$ while $\gamma_2$ uses $\alpha_2$ and $\alpha_3$.

*distinction*. For each node in a parse tree, one of the daughter nodes is picked to be the head based on a heuristic rule that depends on the node labels, e.g. *V* is picked as the head daughter for node labeled *VP*. In LTAG-based statistical parsers, high accuracy is obtained by using the Magerman-Collins head-percolation rules in order to provide the etrees. (Chiang and Bikel, 2002) describes these heuristic rules in detail for the Collins parsing model (Collins, 1996) and (Chiang, 2000) describes the algorithm for converting a head-annotated parse tree into an LTAG derivation tree. In Fig. 1 the derivation tree $\gamma_2$ was created using this process. A *spine* of an etree is the path from lexical item to the root of the etree.

Argument-adjunct ambiguity is created if arguments are optionally localized using the substitution operation into the etree that is anchored with the predicate in conventional LTAG. Since this ambiguity is a core aspect of the SRL task, we do not use substitution in our grammar. In the resulting derivations, all etrees are in the spinal form (see $\alpha_1$ in Fig. 1). These etrees are composed into LTAG derivations through *sister adjunction* which is commonly used in LTAG statistical parsers to deal with the relatively flat Penn Treebank trees (Chiang, 2000).

Given a predicate and parse tree, the latent LTAG derivations are created using the following method (illustrated by an example in Fig. 2). For each *predicate*, we apply 3 heuristic head-percolation rules to generate 3 head-annotated parse trees per predicate:

1. Magerman-Collins rules. (produces etree $p_1$ in Fig. 2)
2. Same as Rule 1, but label all consecutive VP nodes as head from the predicate until you see a non-VP node. (also produces etree $p_1$ in Fig. 2)
3. Same as Rule 1, but label all nodes as head from the predicate to the root of the PTB tree. (produces etree $p_2$ in Fig. 2)

For each head-annotated parse tree, different predicate etrees are extracted using the algorithm given in (Chiang, 2000) that converts head-annotated parse trees into LTAG derivations. For each predicate etree, we produce argument etrees which localize the constituent that may receive a semantic role (called the target, e.g. [NP] in Fig. 2). Each node from the target to the root of the parse tree is taken to be the root of an alternative argument etree. E.g. etrees $a_1, a_3, a_5$ in Fig. 2. Each target also produces additional latent derivations by choosing a different head child for the target. E.g. etrees $a_1$ and $a_2$ are identical except for different lexical anchors. When the number of intermediate etrees (defined as etrees between the predicate etree and argument etree in the LTAG derivation) is less than 3, we also consider all the possible variants for intermediate etrees based on the same strategy for generating argument etrees described above. E.g. etrees $i_1$ and $i_2$ in Fig. 2. Derivations rooted in $i_1$ do not generate the entire input tree $T$, and they don't have to because latent LTAG features only include information to link a predicate with its potential arguments. One LTAG derivation is different from another if the combination of predicate etree, argument etree, and intermediate etrees (if any) is different. Chapter 4 of (Liu, 2009) contains the pseudo-code for the algorithm that we use to generate the latent LTAG derivation trees. In our dataset on average we have $\sim$130 derivation trees generated for each SRL instance.

## 3 Latent Support Vector Machines

We hypothesize that for each SRL classification task, positive and negative examples of individual semantic roles can be reliably distinguished by possibly different latent LTAG features, and thus the latent features can be used discriminatively using a max-margin SVM classifier that only picks the most predictive features. Inspired by (Cherry and Quirk, 2008), we apply Latent SVMs (LSVM) to the SRL task. LSVM is a relatively new methodology for

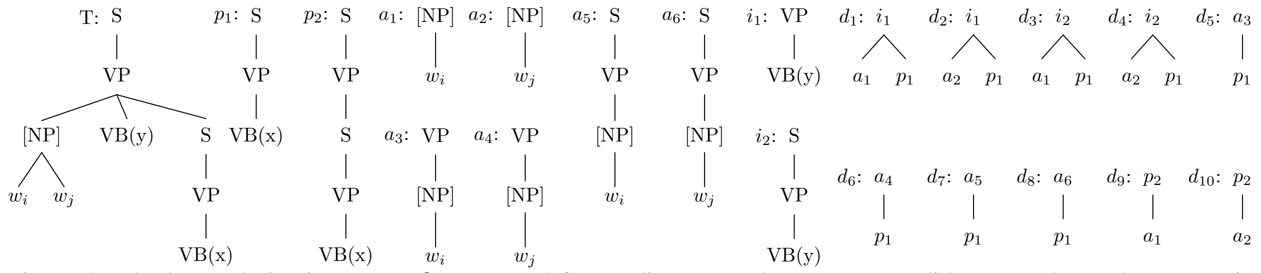

Figure 2: The latent derivation trees ($d_i$) extracted for predicate $x$ and argument candidate NP (denoted as [NP]) in a parse tree fragment $T$. $p_i$: variants of predicate etree, $a_i$: variants of argument etrees, $i_i$: variants of intermediate etrees. In contrast to (Chen and Rambow, 2003) argument nodes do not have to be the root node of an etree.

discriminative training introduced by (Felzenszwalb et al., 2008) for object detection in computer vision. LSVMs are a generalization of classic SVMs to handle classification problems with latent structures with the properties we described above.

Let $D = \{\langle x_i, y_i \rangle\}_1^n$ be a set of labeled examples where $x_i$ is an input and $y_i$ is the output label, either $+1$ or $-1$. In classical SVMs, a weight vector $\mathbf{w}$ needs to be trained from $D$ by optimizing the following objective function:

$$\arg\min_{\mathbf{w}} \frac{\lambda}{2}\|\mathbf{w}\|^2 + \sum_{i=1}^{n} \max(0, 1 - y_i f_{\mathbf{w}}(x_i)) \quad (1)$$

where $f_{\mathbf{w}}(x_i) = \mathbf{w} \cdot \Phi(x_i)$ and $\Phi$ is a feature vector for representing the input $x_i$. Each misclassification (when $y_i f_{\mathbf{w}}(x_i) < 1$) incurs a penalty proportional to the degree to which it violates the margin constraint.

In LSVMs, we introduce an extra dimensionality of latent structures, and classify input instances while simultaneously searching for a good latent structure. The score assigned to an input $x$ is then defined as,

$$f_{\mathbf{w}}(x) = \max_{h \in H(x)} \mathbf{w} \cdot \Phi(x, h) \quad (2)$$

where $H$ is a set of latent structures. As in classical SVMs, the weight vector is trained by solving Eqn. 1, where those latent annotations are preferred which induce low training error (the second term), and at the same time, lead to a low complexity classifier (the first term). In other words, the training instances are embedded into an appropriate extended feature space specific to the given classfication task. If there is only one possible latent structure for each input ($|H(x)| = 1$), then LSVM reduces to classical SVM.

To learn the model parameter $\mathbf{w}$, we use a coordinate descent algorithm to optimize Eqn. 1:

1. Find the best structure for each training example using the current weight vector:
$$\forall i : h_i \leftarrow \arg\max_{h \in H(x_i)} \mathbf{w} \cdot \Phi(x_i, h)$$
2. Using the current values for latent structures, learn the weight vector to minimize Eqn. 1 with $f_{\mathbf{w}}(x_i) = \mathbf{w} \cdot \Phi(x_i, h_i)$.

For SVM training in the second step, we use the freely available SvmSgd package[1] which implements *stochastic gradient descent* (SGD), an online learning algorithm for SVMs which is known to scale to millions of features and a large number of training examples (as is the case in the SRL task). The SGD algorithm selects a random training example at time $t$, and sets:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \nabla F^{(t)}(\mathbf{w}^{(t)})$$

where $F^{(t)}(\mathbf{w})$ is the objective function specialized for the example chosen at time $t$, and $\eta$ is the learning rate. The gradient $\nabla F^{(t)}(\mathbf{w})$ for our case is

$$\begin{cases} \lambda\mathbf{w} - y^{(t)}\Phi(x^{(t)}, h^{(t)}) & \text{if } y^{(t)}\mathbf{w} \cdot \Phi(x^{(t)}, h^{(t)}) < 1 \\ \lambda\mathbf{w} & \text{otherwise} \end{cases}$$

For our experiments, the space of latent features is structured, namely each latent LTAG derivation tree. However, the derivation tree is mapped into an unstructured set of latent features that are used in a linear kernel within the LSVM framework. More structured latent variables can also be used in LSVMs which need more sophisticated training algorithms (Yu and Joachims, 2009; Wang and Mori, 2009).

[1] *http://leon.bottou.org/projects/sgd*

| Task | Positive Examples | Negative Examples |
|---|---|---|
| ID | 219,556 | 349,629 |
| A0 | 60,351 | 159,205 |
| A1 | 80,486 | 139,070 |
| A2 | 17,191 | 202,365 |
| A3 | 3,005 | 218,551 |
| A4 | 2,461 | 217,095 |

Table 1: The number of positive and negative training examples for the identification and classification tasks.

## 4 Semantic Role Labeling with LSVM

### 4.1 SRL Instance Generation

We use a commonly used architecture for building our SRL system. It consists of four stages: **pruning**, **argument identification**, **argument classification** and finally optionally we perform a **multi-class classification**. For SRL identification we train a binary LSVM classifier and for SRL classification we use the standard "one-vs-all" scheme. For each given training point $\langle x_i, y_i \rangle$, $x_i$ is a constituent from the derived tree, and $y_i$ is the output label. For identification, $y$ indicates whether the current constituent that dominates a span is an argument or not. For each A$_k$ classification, $y$ indicates whether the current constituent, which has been already identified as an argument, has the A$_k$ semantic role. In the pruning stage we only allow nodes that are sisters to nodes on the path from the predicate to the root *in the parse tree*. This is commonly used in the SRL community (c.f. (Xue and Palmer, 2004)). Pruning and parser errors provide an upper bound on the recall for any of our methods. We then generate the SRL training instances (positive and negative) from the pruned PTB trees. Table 1 shows the number of positive and negative instances that are generated from our training data for LSVM training. As in many SRL systems, this instance generation method makes an independence assumption among SRL instances extracted from the same sentence. However, we can obtain high performance without an explicit model of the frameset because with sufficiently discriminative features we never confuse a structural location in the parse during classification (A0, A1, etc.) and such a classifier will not predict two A0's in a row even though each prediction is independent.

Table 2: Features from the phrase-structure tree used in our system. Features with asterisk $*$ are not used for argument identification.

| **Basic features from (Gildea and Jurafsky, 2002)** | |
|---|---|
| • predicate lemma and voice<br>• phrase type and head word (phrase = candidate)<br>• path from phrase to predicate<br>• position: phrase relative to predicate: before or after<br>• sub-cat records the immediate structure that expands from predicate's parent | 1<br><br>2 |
| **Additional features proposed by (Surdeanu et al. 2003; Pradhan et al., 2004, 2005)** | |
| • predicate POS<br>• head word POS<br>• first/last word/POS<br>• POS of word immediately before/after phrase<br>• path length<br>• Lowest common ancestor path between phrase and predicate<br>• punctuation immediately before/after phrase$*$<br>• path trigrams$*$: up to 9 are considered<br>• head word named entity label such as "PER, ORG, LOC"$*$<br>• content word named entity label for PP parent node$*$ | 1 |
| **Additional features proposed by (Xue and Palmer, 2004)** | |
| • predicate_phrase type<br>• predicate_head word<br>• voice_position<br>• syntactic frame$*$ | |

[1] In Fig. 1 *VP↑VP↑S↓NP* is the path from the predicate to a candidate phrase with path length 4.
[2] This feature is different from the **frame** feature which usually refers to all the semantic participants for the particular predicate.

### 4.2 Features

Table 2 lists the features that are extracted from PTB trees, and Table 3 listed the features we proposed that are extracted from the LTAG derivations mostly taken from (Liu and Sarkar, 2007) but we add $n$-gram features on etree spines. Our LTAG-based SRL systems use features from both Table 2 and Table 3.

## 5 Experiments

### 5.1 Experimental Setup

Training data (PropBank Sections 02-21), development set data (Section 24) and test data (Sec-

Table 3: Features from latent LTAG derivation trees used in our system. Head related features in Table 2 will be determined by the current LTAG derivation tree when they are used with the features in this table.

| |
|---|
| • pred etree (plus POS $\rightarrow$ voice) |
| • arg etree (plus POS $\rightarrow$ NE if any) |
| • reln of arg etree to pred etree: {child, parent, sibling, grandparent, uncle, other} |
| • distance between arg etree and pred etree: {0, 1, 2} |
| • label/index of attachment node between pred etree and parent (index of root = 0) [1] |
| • position1 of pred etree to its parent etree |
| • intermediate etree #1: predicate's parent etree, POS and lemma of anchor [2] |
| • label/index of attachment point of arg etree and its parent OR [3] |
| • label/index of attachment point of arg etree and its child [4] |
| • intermediate etree #2: arg's parent etree, POS and lemma of anchor OR [5] |
| • arg's child etree, POS and lemma of anchor [4] |
| • position2 of arg etree to its parent |
| • label of attachment between pred's parent etree and arg's parent etree [6] |
| • position3 of pred's parent etree to arg's parent etree |
| • position1 + position2 + position3 + relation |
| • position1 + position2 + position3 + distance |
| • relation + distance |
| • upto 9 trigrams of arg etree spine (from the head) |
| • upto 9 trigrams of pred etree spine (from the pred) |

when relation is:
[1] parent or sibling or grandparent or uncle.
[2] sibling or grandparent or uncle.
[3] child or sibling or uncle.
[4] grandparent.
[5] sibling or uncle.
[6] uncle.

tion 23) are taken from CoNLL-2005 shared task data[2]. All the necessary annotation information such as predicates, parse trees from the Charniak parser (Charniak, 2000) as well as named entity labels are part of the data. The argument set we considered is the same as that of CoNLL-2005 shared task data, which includes {A0, A1, A2, A3, A4, A5} and 13 adjunct-like arguments such as AM-TMP and 11 referring arguments such as R-A0. The classification performance is evaluated using standard *Precision/Recall/F1* (p/r/f) scores.

We train 30 binary LSVM classifiers on the training data, one $A_x$ or not-$A_x$ binary classifier for each argument type that appears in the test set and development set.

We run LSVM training for a fixed number of iterations (39 in our experiments) and choose a stopping point based on performance on a heldout development set. There are 3 argument types (R-A3, R-A4 and R-AM-ADV) that never appear in the dev set, so we choose iteration 10 on the test set as the stopping point for them. In our experiment we use the default settings for SvmSgd. For more details, please refer to Chapter 4 of (Liu, 2009).

## 5.2 Results

Table 4 shows the results of LSVM-SRL using the CoNLL-2005 evaluation program on the CoNLL test data using the Charniak parse trees. We significantly outperform the previous best results on this dataset given in (Toutanova et al., 2008).

## 5.3 Analysis

To compare the relative utility of the LSVM for SRL, we have built two baseline systems:

- **Baseline1** is an SRL system that uses the features in Table 2 only. No LTAG derivation is introduced for each SRL instance.
- **Baseline2** is an SRL system that uses the features in Table 2 and Table 3. The LTAG derivation that is introduced for each SRL instance is generated using *Magerman-Collins* head percolation rules and *sister-adjunction* operation. This baseline replicates the SRL method in (Liu and Sarkar, 2007).

We use SvmSgd to train both baseline systems.

The weight initialization for LSVM-SRL is taken from Baseline2: the weights trained from Baseline2 are the initial weights for LSVM training. Some features may be shared between multiple latent LTAG derivations. We alternate between learning/updating the weights using SvmSgd and selecting the best latent derivation based on the updated weights. Table 5 shows the *Precision/Recall/Fscore* of LSVM-SRL and the baselines for each binary classification task. It shows that LSVM is producing very high precision with small gains in recall, which matches our intuition that the features are more predictive but do not address parser errors, and other issues that lead to low recall.

[2]http://www.lsi.upc.es/∼srlconll/.

| | Our system | | | | | | (Toutanova et al., 2008) | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| label | corr. | excess | miss. | prec. | rec. | F1 | corr. | excess | miss. | prec. | rec. | F1 |
| Overall | 11832 | 506 | 2245 | 95.90 | 84.05 | **89.59** | 11094 | 2452 | 2983 | 81.90 | 78.81 | 80.32 |
| A0 | 3322 | 41 | 241 | 98.78 | 93.24 | **95.93** | 3168 | 417 | 395 | 88.32 | 88.30 | 88.31 |
| A1 | 3907 | 342 | 1020 | 91.95 | 79.30 | **85.16** | 4004 | 909 | 923 | 81.50 | 81.27 | 81.38 |
| A2 | 821 | 6 | 289 | 99.27 | 73.96 | **84.77** | 763 | 276 | 347 | 73.44 | 68.74 | 71.01 |
| A3 | 129 | 13 | 44 | 90.85 | 74.57 | **81.90** | 96 | 32 | 77 | 75.00 | 55.49 | 63.79 |
| A4 | 80 | 4 | 22 | 95.24 | 78.43 | **86.02** | 71 | 24 | 31 | 74.74 | 69.61 | 72.08 |
| A5 | 4 | 6 | 1 | 40.00 | 80.00 | 53.33 | 4 | 0 | 1 | 100.00 | 80.00 | **88.89** |
| AM-ADV | 391 | 7 | 115 | 98.24 | 77.27 | **86.50** | 275 | 149 | 231 | 64.86 | 54.35 | 59.14 |
| AM-CAU | 48 | 4 | 25 | 92.31 | 65.75 | **76.80** | 36 | 17 | 37 | 67.92 | 49.32 | 57.14 |
| AM-DIR | 58 | 0 | 27 | 100.00 | 68.24 | **81.12** | 34 | 27 | 51 | 55.74 | 40.00 | 46.58 |
| AM-DIS | 290 | 7 | 30 | 97.64 | 90.62 | **94.00** | 241 | 54 | 79 | 81.69 | 75.31 | 78.37 |
| AM-EXT | 26 | 1 | 6 | 96.30 | 81.25 | **88.14** | 13 | 7 | 19 | 65.00 | 40.62 | 50.00 |
| AM-LOC | 271 | 0 | 92 | 100.00 | 74.66 | **85.49** | 206 | 104 | 157 | 66.45 | 56.75 | 61.22 |
| AM-MNR | 263 | 1 | 81 | 99.62 | 76.45 | **86.51** | 195 | 117 | 149 | 62.50 | 56.69 | 59.45 |
| AM-MOD | 547 | 2 | 4 | 99.64 | 99.27 | **99.45** | 540 | 11 | 11 | 98.00 | 98.00 | 98.00 |
| AM-NEG | 228 | 0 | 2 | 100.00 | 99.13 | **99.56** | 225 | 5 | 5 | 97.83 | 97.83 | 97.83 |
| AM-PNC | 81 | 0 | 34 | 100.00 | 70.43 | **82.65** | 45 | 38 | 70 | 54.22 | 39.13 | 45.45 |
| AM-PRD | 2 | 0 | 3 | 100.00 | 40.00 | **57.14** | 1 | 0 | 4 | 100.00 | 20.00 | 33.33 |
| AM-REC | 1 | 0 | 1 | 100.00 | 50.00 | **66.67** | 0 | 0 | 2 | 0.00 | 0.00 | 0.00 |
| AM-TMP | 895 | 54 | 192 | 94.31 | 82.34 | **87.92** | 786 | 195 | 301 | 80.12 | 72.31 | 76.92 |
| R-A0 | 220 | 0 | 4 | 100.00 | 98.21 | **99.10** | 204 | 15 | 20 | 93.15 | 91.07 | 92.10 |
| R-A1 | 149 | 3 | 7 | 98.03 | 95.51 | **96.75** | 128 | 31 | 28 | 80.50 | 82.05 | 81.27 |
| R-A2 | 16 | 0 | 0 | 100.00 | 100.00 | **100.00** | 8 | 5 | 8 | 61.54 | 50.00 | 55.17 |
| R-A3 | 0 | 0 | 1 | 0.00 | 0.00 | 0.00 | 0 | 0 | 1 | 0.00 | 0.00 | 0.00 |
| R-A4 | 0 | 0 | 1 | 0.00 | 0.00 | 0.00 | 0 | 0 | 1 | 0.00 | 0.00 | 0.00 |
| R-AM-ADV | 2 | 1 | 0 | 66.67 | 100.00 | **80.00** | 0 | 0 | 2 | 0.00 | 0.00 | 0.00 |
| R-AM-CAU | 4 | 0 | 0 | 100.00 | 100.00 | **100.00** | 2 | 0 | 2 | 100.00 | 50.00 | 66.67 |
| R-AM-EXT | 1 | 1 | 0 | 50.00 | 100.00 | **66.67** | 0 | 0 | 1 | 0.00 | 0.00 | 0.00 |
| R-AM-LOC | 20 | 1 | 1 | 95.24 | 95.24 | **95.24** | 10 | 3 | 11 | 76.92 | 47.62 | 58.82 |
| R-AM-MNR | 5 | 12 | 1 | 29.41 | 83.33 | **43.48** | 2 | 2 | 4 | 50.00 | 33.33 | 40.00 |
| R-AM-TMP | 51 | 0 | 1 | 100.00 | 98.08 | **99.03** | 37 | 13 | 15 | 74.00 | 71.15 | 72.55 |
| V | 5126 | 141 | 141 | 97.32 | 97.32 | 97.32 | 5126 | 141 | 141 | 97.32 | 97.32 | 97.32 |

Table 4: Precision/Recall/Fscore% of the LSVM-SRL system compared to the state of the art on the PropBank/CoNLL-2005 shared task (Toutanova et al., 2008). Both systems use Charniak parser output.

**Learning Curves** Fig. 3 shows the learning curve evaluated on test data across the iterations of LSVM training. The largest improvement happens in the first iteration where the weights are updated for the first time; the performance levels off after a few iterations. A similar trend has been observed in the development set. To better understand the behavior of LSVMs, we analyse various aspects of the LSVM-SRL algorithm and experimental results.

**Stability in LSVM Training** In each training iteration LSVM picks the argmax latent LTAG derivation. Are the latent derivation trees stabilized during the LSVM learning iterations? Fig. 4 plots the percentage of the instances in the training data for which the latent derivation tree has changed compared to the immediately previous iteration. The model converges pretty quickly to its best latent annotation for the training data. Fig. 4 also shows the percentage of the instances for which the argmax derivation tree has not been observed in *all* previous iterations. For over 90% of the instances the derivation trees are changed in the first iteration, which is a drastic departure from the initial derivation trees (from Baseline2). The tendency to pick unseen derivation trees exists throughout the learning process, although it drops significantly after a few iterations.

**Model Complexity** Does LSVM training effectively minimize the complexity of the learned model by capturing feature sparsity? Each newly-selected previously unseen derivation tree (in Fig. 4) brings

| class | Baseline1 (p/r/f)% | | | Baseline2 (p/r/f)% | | | LSVM-SRL (p/r/f)% | | | stop iter | recall bound |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **ID** | 87.71 | 84.86 | 86.26 | 89.00 | 85.21 | 87.01 | 98.96 | 86.38 | **92.24** | 12 | 86.90 |
| **A0** | 86.46 | 85.30 | 85.87 | 87.87 | 87.50 | 87.68 | 99.26 | 90.31 | **94.57** | 18 | 94.24 |
| **A1** | 78.70 | 83.72 | 81.13 | 84.56 | 82.16 | 83.34 | 99.69 | 82.00 | **89.99** | 22 | 84.37 |
| **A2** | 85.04 | 73.91 | 79.09 | 83.00 | 74.86 | 78.72 | 99.26 | 73.35 | **84.36** | 26 | 76.24 |
| **A3** | 77.04 | 65.82 | 71.00 | 83.46 | 67.09 | 74.39 | 98.48 | 76.47 | **86.09** | 18 | 78.24 |
| **A4** | 77.42 | 79.12 | 78.26 | 90.14 | 70.33 | 79.01 | 98.77 | 79.21 | **87.91** | 20 | 80.20 |
| **AM** | 80.85 | 81.39 | 81.11 | 82.10 | 81.87 | 81.99 | 97.73 | 85.31 | **91.10** | 22 | 85.75 |

Table 5: Results for individual binary classifiers, identification (ID) and $A_x$ v.s. not-$A_x$ classification. The stopping iteration was the one that gave highest accuracy on the dev set. The upper bound on recall is due to missing constituents in the Charniak parser output and due to pruning the tree.

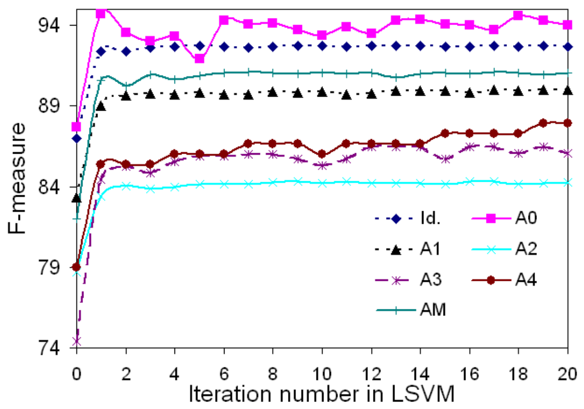

Figure 3: Fscore of identification and classification tasks on test data for each iteration of LSVM.

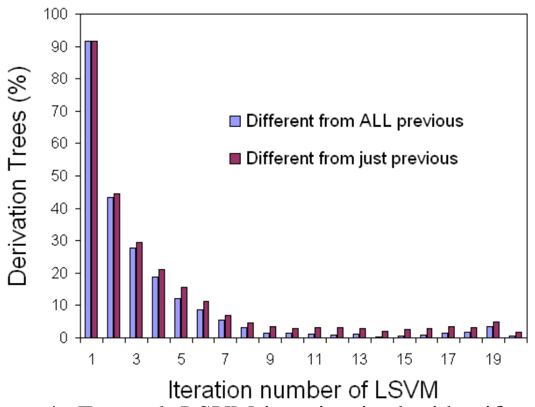

Figure 4: For each LSVM iteration in the identification task, the percentage of derivation trees which are different from the previous iteration versus those which are new and have not been observed in all previous iterations.

a set of new active features[3] into the model. Although new active features are added constantly to the model, Fig. 5 shows that the total number of active features decreases drastically during the first few learning iterations, and then stabilizes. In fact the model has picked a small set of features (8K features from the pool of 1,242,869 all possible features) having strong predictions for each individual task.

**LSVM Initialization** Does the initialization of the weight vector in LSVM matter? In our main experiments we initialize based on SVM training using linguistically plausible Magerman-Collins (MC) head rules. To compare against this initialization, for each training instance, we randomly select a derivation tree from its space of latent derivation trees, and we train a regular SVM with SvmSgd using this derivation tree (as we did in Baseline2). Over five random choices this approach gives an accuracy of 71.15±.79/86.11±.29/77.92±.36. We then use these weights as the initial setting for LSVM training. E.g. in the A0 classification task, we have observed that p/r/f% is 84.54±8.00/86.44±2.12/85.33±4.49 which is lower than initialization using MC head rules[4]. The results show that proper initialization matters a lot.

**Why is LSVM so accurate?** Fig. 6 is an example showing that LSVM finds ways to localize the predicate-argument relation in the LTAG derivation tree. In $\gamma_1$ tree, the predicate etree and the argument etree is far away from each other and their relation is not a *parent, child, sibling, grandparent* or *uncle* relationship. However in $\gamma_2$ tree these two etrees are localized as a *grandparent* relation. A more detailed analysis is shown in Table 6 where we show how the LSVM chosen derivation is different from the

[3]Active features are those which have non-zero values in the learned weight vector.

[4]The random initialization results are inflated since we chose the best results on the test set for the 5 runs rather than choosing a stopping point using the devset as in the results in Table 4.

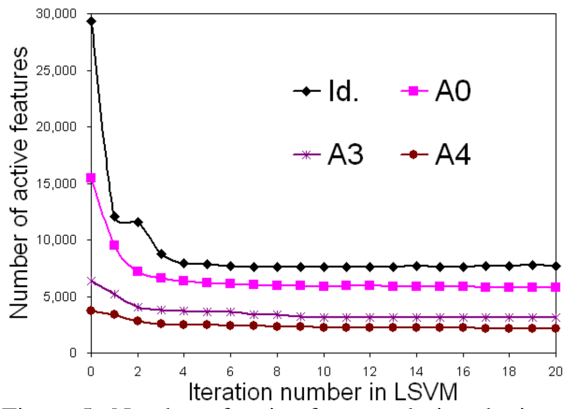

Figure 5: Number of active features during the iteration of LSVM for different tasks. These curves for the other classification tasks are similar to that of A0.

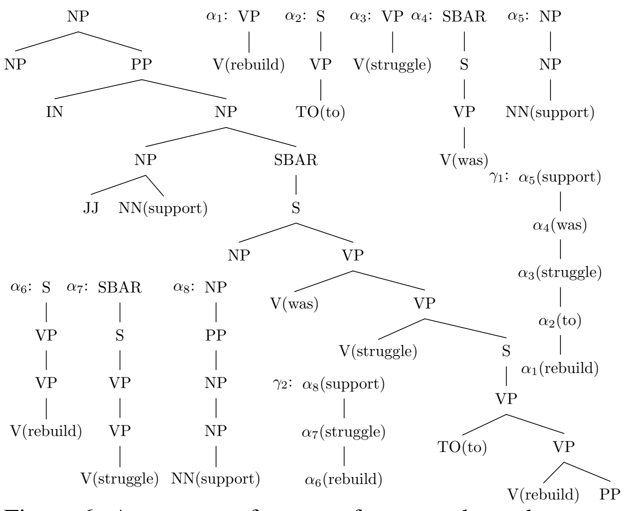

Figure 6: A parse tree fragment from test data where *rebuild* is the predicate and *support* is Arg1 of the predicate. $\gamma_1$ is the derivation tree from Baseline2, and $\gamma_2$ is the LSVM picked derivation tree. This is an example that is mis-predicted by Baseline2 but corrected by LSVM.

Magerman-Collins derivation (used by Baseline2), for each label (identification or ID, and A0. . .A4). From the examples that LSVM gets right but Baseline2 gets wrong, we count the percentage of differences between the LSVM argmax derivation and the Baseline2 derivation. We count the following differences: changed lexical head of argument etree (LexHd), a shorter distance between predicate etree and argument etree (SDist), a changed predicate etree (PredSP), a changed argument etree (ArgSP), and any changed intermediate etree (IntSP). LSVM does not change the lexical head frequently for the identification classifier. In contrast, LSVM creates a shorter distance between predicate etree and argument etree for identification (often by picking a different predicate etree), but not for classification. This is consistent with insights from (Pradhan et al., 2008) who say "...the more lexical/semantic features dominating the classification task where more general structural features are more dominant in the argument identification task."

| | ID | A0 | A1 | A2 | A3 | A4 |
|---|---|---|---|---|---|---|
| LexHd | 26.1 | 44.5 | 47.7 | 74.4 | 28.6 | 18.8 |
| SDist | 79.9 | 18.4 | 15.9 | 5.3 | 3.9 | 6.3 |
| PredSP | 90.7 | 43.4 | 58.1 | 74.6 | 80.5 | 81.3 |
| ArgSP | 47.1 | 56.6 | 65.8 | 80.6 | 28.6 | 25.0 |
| IntSP | 5.7 | 17.1 | 14.7 | 2.3 | 14.3 | 6.2 |

Table 6: Percentage of LSVM corrected examples with changed substructures in LSVM chosen derivation tree.

To help understand the details of our implementation and to encourage replication of this result, the source code and the LSVM-SRL output on test data in CoNLL 2005 format can be downloaded from *http://natlang.cs.sfu.ca*.

## 6 Related and Future Work

(Cherry and Quirk, 2008) is the first application of LSVMs to NLP. They train a LSVM for language modeling which uses latent parse trees. We share the intuition that the latent features can be helpful in discrimination between positive and negative examples; in their case, latent CFG rules were used to judge (un)grammaticality of sentences.

The EM algorithm has been used in statistical parsing to explore the space of head-choice and head-percolation strategies (Chiang and Bikel, 2002; Shen, 2004). In both cases the heuristic Magerman-Collins rules always out-performed the learned rules in terms of parsing accuracy.

(Vickrey and Koller, 2008) is the only other work (that we know of) that uses latent features for SRL. They use hand-written sentence simplification rules and search for the most useful set of simplifications that can improve training error based on a log-linear model. Their model sums over the set of possible simplifications in contrast to our LSVM approach. We are currently exploring the use of log-linear models for SRL using latent LTAG features where we sum over the latent derivations.

## References

Eugene Charniak. 2000. A maximum-entropy-inspired parser. In *NAACL-2000*.

John Chen and Owen Rambow. 2003. Use of deep linguistic features for the recognition and labeling of semantic arguments. In *EMNLP-2003*.

Colin Cherry and Chris Quirk. 2008. Discriminative, syntactic language modeling through latent SVMs. In *AMTA-2008*.

David Chiang and Daniel Bikel. 2002. Recovering latent information in treebanks. In *COLING-2002*.

David Chiang. 2000. Statistical parsing with an automatically extracted tree adjoining grammars. In *ACL-2000*.

Michael Collins. 1996. A new statistical parser based on bigram lexical dependencies. In *ACL-1996*.

Pedro Felzenszwalb, David McAllester, and Deva Ramanan. 2008. A discriminatively trained, multiscale, deformable part model. In *CVPR-2008*.

Daniel Gildea and Julia Hockenmaier. 2003. Identifying semantic roles using combinatory categorial grammar. In *EMNLP-2003*.

Daniel Gildea and Dan Jurafsky. 2002. Automatic labeling of semantic roles. *Computational Linguistics*, 58(3):245–288.

Kadri Hacioglu, Sameer Pradhan, Wayne Ward, James Martin, and Dan Jurafsky. 2004. Semantic role labeling by tagging syntactic chunks. In *CoNLL-2004 Shared Task*.

Kadri Hacioglu. 2004. Semantic role labeling using dependency trees. In *COLING-2004*.

Aravind Joshi and Yves Schabes. 1997. Tree-adjoining grammars. In G. Rosenberg and A. Salomaa, editors, *Handbook of Formal Languages*, pages 69–123. Springer, Berlin.

Yudong Liu and Anoop Sarkar. 2007. Experimental evaluation of LTAG-based features for semantic role labeling. In *EMNLP-2007*.

Yudong Liu. 2009. *Semantic role labeling using lexicalized tree adjoining grammars*. Ph.D. thesis, Simon Fraser University, Burnaby, BC, Canada.

Martha Palmer, Dan Gildea, and Paul Kingsbury. 2005. The proposition bank: An annotated corpus of semantic roles. *Computational Linguistics*, 31(1).

Sameer Pradhan, Wayne Ward, Kadri Hacioglu, James Martin, and Dan Jurafsky. 2005. Semantic role labeling using different syntactic views. In *ACL-2005*.

Sameer Pradhan, Wayne Ward, and James Martin. 2008. Towards robust semantic role labeling. *Computational Linguistics*, 34(2).

Libin Shen. 2004. Nondeterministic LTAG derivation tree extraction. In *Proc. TAG+7*.

Kristina Toutanova, Aria Haghighi, and Christopher Manning. 2008. A global joint model for semantic role labeling. *Computational Linguistics*, 34(2).

David Vickrey and Daphne Koller. 2008. Sentence simplification for semantic role labeling. In *Proceedings of ACL-08: HLT*.

Yang Wang and Greg Mori. 2009. Max-margin hidden conditional random fields for human action recognition. In *CVPR*.

Nianwen Xue and Martha Palmer. 2004. Calibrating features for semantic role labeling. In *EMNLP-2004*.

Chun-Nam Yu and Thorsten Joachims. 2009. Learning structural svms with latent variables. In *ICML '09*.