# Joint Training of Dependency Parsing Filters through Latent Support Vector Machines

**Colin Cherry**
Institute for Information Technology
National Research Council Canada
colin.cherry@nrc-cnrc.gc.ca

**Shane Bergsma**
Center for Language and Speech Processing
Johns Hopkins University
sbergsma@jhu.edu

## Abstract

Graph-based dependency parsing can be sped up significantly if implausible arcs are eliminated from the search-space before parsing begins. State-of-the-art methods for arc filtering use separate classifiers to make pointwise decisions about the tree; they label tokens with roles such as *root*, *leaf*, or *attaches-to-the-left*, and then filter arcs accordingly. Because these classifiers overlap substantially in their filtering consequences, we propose to train them jointly, so that each classifier can focus on the gaps of the others. We integrate the various pointwise decisions as latent variables in a single arc-level SVM classifier. This novel framework allows us to combine nine pointwise filters, and adjust their sensitivity using a shared threshold based on arc length. Our system filters 32% more arcs than the independently-trained classifiers, without reducing filtering speed. This leads to faster parsing with no reduction in accuracy.

## 1 Introduction

A dependency tree represents syntactic relationships between words using directed arcs (Mel'čuk, 1987). Each token in the sentence is a node in the tree, and each arc connects a *head* to its *modifier*. There are two dominant approaches to dependency parsing: graph-based and transition-based, where graph-based parsing is understood to be slower, but often more accurate (McDonald and Nivre, 2007).

In the graph-based setting, a complete search finds the highest-scoring tree under a model that decomposes over one or two arcs at a time. Much of the time for parsing is spent scoring each **potential arc** in the complete dependency graph (Johnson, 2007), one for each ordered word-pair in the sentence. Potential arcs are scored using rich linear models that are discriminatively trained to maximize parsing accuracy (McDonald et al., 2005). The vast majority of these arcs are bad; in an $n$-word sentence, only $n$ of the $n^2$ potential arcs are correct. If many arcs can be filtered before parsing begins, then the entire process can be sped up substantially.

Previously, we proposed a cascade of filters to prune potential arcs (Bergsma and Cherry, 2010). One stage of this cascade operates one token at a time, labeling each token $t$ according to various roles in the tree:

- **Not-a-head** ($NaH$): $t$ is not the head of any arc
- **Head-to-left** ($HtL\{1/5/*\}$): $t$'s head is to its left within 1, 5 or any number of words
- **Head-to-right** ($HtR\{1/5/*\}$): as head-to-left
- **Root** ($Root$): $t$ is the root node, which eliminates arcs according to projectivity

Similar to Roark and Hollingshead (2008), each role has a corresponding binary classifier. These **token-role classifiers** were shown to be more effective than **vine parsing** (Eisner and Smith, 2005; Dreyer et al., 2006), a competing filtering scheme that filters arcs based on their length (leveraging the observation that most dependencies are short).

In this work, we propose a novel filtering framework that integrates all the information used in token-role classification and vine parsing, but offers a number of advantages. In our previous work, classifier decisions would often overlap: different token-role classifiers would agree to filter the same arc. Based on this observation, we propose a joint training framework where only the most confident

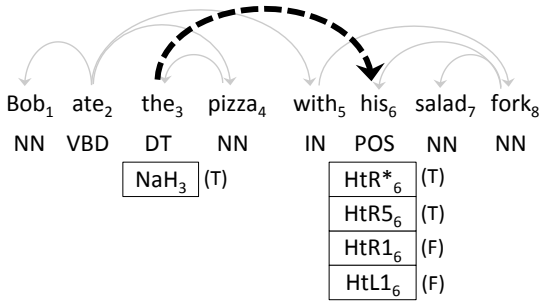


Figure 1: The dotted arc can be filtered by labeling any of the boxed roles as True; i.e., predicting that the head $the_3$ is not the head of any arc, or that the modifier $his_6$ attaches elsewhere. Role truth values, derived from the gold-standard tree (in grey), are listed adjacent to the boxes, in parentheses.

classifier is given credit for eliminating an arc. The identity of the responsible classifier is modeled as a latent variable, which is filled in during training using a latent SVM (LSVM) formulation. Our use of an LSVM to assign credit during joint training differs substantially from previous LSVM applications, which have induced latent linguistic structures (Cherry and Quirk, 2008; Chang et al., 2010) or sentence labels (Yessenalina et al., 2010).

In our framework, each classifier learns to focus on the cases where the other classifiers are less confident. Furthermore, the integrated approach directly optimizes for arc-filtering accuracy (rather than token-labeling fidelity). We trade-off filtering precision/recall using two hyperparameters, while the previous approach trained classifiers for eight different tasks resulting in sixteen hyperparameters. Ultimately, the biggest gains in filter quality are achieved when we jointly train the token-role classifiers together with a dynamic threshold that is based on arc length and shared across all classifiers.

## 2 Joint Training of Token Roles

In our previous system, filtering is conducted by training a separate SVM classifier for each of the eight token-roles described in Section 1. Each classifier uses a training set with one example per treebank token, where each token is assigned a binary label derived from the gold-standard tree. Figure 1 depicts five of the eight token roles, along with their truth values. The role labelers can be tuned for high precision with label-specific cost parameters; these are tuned separately for each classifier. At test time, each of the eight classifiers assigns a binary label to each of the $n$ tokens in the sentence. Potential arcs are then filtered from the complete dependency graph according to these token labels. In Figure 1, a positive assignment to any of the indicated token-roles is sufficient to filter the dotted arc.

In the current work, we maintain almost the same test-time framework, but we alter training substantially, so that the various token-role classifiers are trained jointly. To do so, we propose a classification scheme focused on **arcs**.[1] During training, each arc is assigned a filtering **event** as a latent variable. Events generalize the token-roles from our previous system (e.g. $NaH_3$, $HtR{*}_6$). Events are assigned binary labels during filtering; positive events are said to be **detected**. In general, events can correspond to any phenomenon, so long as the following holds: For each arc $a$, we must be able to deterministically construct the set $\mathrm{Z}_a$ of all events that would filter $a$ if detected.[2] Figure 1 shows that $\mathrm{Z}_{\mathit{the_3 \rightarrow his_6}} = \{NaH_3, HtR{*}_6, HtR5_6, HtR1_6, HtL1_6\}$.

To detect events, we maintain the eight token-role classifiers from the previous system, but they become **subclassifiers** of our joint system. For notational convenience, we pack them into a single weight vector $\bar{w}$. Thus, the event $z = NaH_3$ is detected only if $\bar{w} \cdot \bar{\Phi}(NaH_3) > 0$, where $\bar{\Phi}(z)$ is $z$'s feature vector. Given this notation, we can cast the filtering decision for an arc $a$ as a maximum. We filter $a$ only if:

$$\mathrm{f}(\mathrm{Z}_a) > 0 \;\text{ where }\; \mathrm{f}(\mathrm{Z}_a) = \max_{z \in \mathrm{Z}_a} \left[\bar{w} \cdot \bar{\Phi}(z)\right] \quad (1)$$

We have reformulated our problem, which previously involved a number of independent token classifiers, as a single arc classifier $\mathrm{f}()$ with an inner $\max$ over latent events. Note the asymmetry inherent in (1). To filter an arc, $\left[\bar{w} \cdot \bar{\Phi}(z) > 0\right]$ must hold for at least one $z \in \mathrm{Z}_a$; but to keep an arc, $\left[\bar{w} \cdot \bar{\Phi}(z) \leq 0\right]$ must hold for all $z \in \mathrm{Z}_a$. Also note that tokens have completely disappeared from our formalism: the classifier is framed only in terms of events and arcs; token-roles are encapsulated inside events.

To provide a large-margin training objective for our joint classifier, we adapt the latent SVM (Felzen-

[1] A joint filtering formalism for CFG parsing or SCFG translation would likewise focus on hyper-edges or spans.

[2] This same requirement is also needed by the previous, independently-trained filters at test time, so that arcs can be filtered according to the roles assigned to tokens.

szwalb et al., 2010; Yu and Joachims, 2009) to our problem. Given a training set $\mathcal{A}$ of $(a, y)$ pairs, where $a$ is an arc in context and $y$ is the correct filter label for $a$ (1 to filter, 0 otherwise), LSVM training selects $\bar{w}$ to minimize:

$$\frac{1}{2}||\bar{w}||^2 + \sum_{(a,y)\in\mathcal{A}} C_y \max\left[0, 1 + \mathrm{f}(\mathrm{Z}_{a|\neg y}) - \mathrm{f}(\mathrm{Z}_{a|y})\right] \quad (2)$$

where $C_y$ is a label-specific regularization parameter, and the event set Z is now conditioned on the label $y$: $\mathrm{Z}_{a|1} = \mathrm{Z}_a$, and $\mathrm{Z}_{a|0} = \{None_a\}$. $None_a$ is a **rejection event**, which indicates that $a$ is **not** filtered. The rejection event slightly alters our decision rule; rather than thresholding at 0, we now filter $a$ only if $\mathrm{f}(\mathrm{Z}_a) > \bar{w} \cdot \bar{\Phi}(None_a)$. One can set $\bar{\Phi}(None_a) \leftarrow \emptyset$ for all $a$ to fix the threshold at 0.

Though not convex, (2) can be solved to a local minimum with an EM-like alternating minimization procedure (Felzenszwalb et al., 2010; Yu and Joachims, 2009). The learner alternates between picking the highest-scoring latent event $\hat{z}_a \in \mathrm{Z}_{a|y}$ for each example $(a, y)$, and training a multiclass SVM to solve an approximation to (2) where $\mathrm{Z}_{a|y}$ is replaced with $\{\hat{z}_a\}$. Intuitively, the first step assigns the event $\hat{z}_a$ to $a$, making $\hat{z}_a$ responsible for $a$'s observed label. The second step optimizes the model to ensure that each $\hat{z}_a$ is detected, leading to the desired arc-filtering decisions. As the process iterates, event assignment becomes increasingly refined, leading to a more accurate joint filter.

The resulting joint filter has only two hyperparameters: the label-specific cost parameters $C_1$ and $C_o$. These allow us to tune our system for high precision by increasing the cost of misclassifying an arc that should not be filtered ($C_1 \ll C_o$).

Joint training also implicitly affects the relative costs of subclassifier decisions. By minimizing an arc-level hinge loss with latent events (which in turn correspond to token-roles), we assign costs to token-roles based on arc accuracy. Consequently, 1) A token-level decision that affects multiple arcs impacts multiple instances of hinge loss, and 2) No extra credit (penalty) is given for multiple decisions that (in)correctly filter the same arc. Therefore, an $NaH$ decision that filters thirty arcs is given more weight than an $HtL5$ decision that filters only one (Item 1), unless those thirty arcs are already filtered by higher-scoring subclassifiers (Item 2).

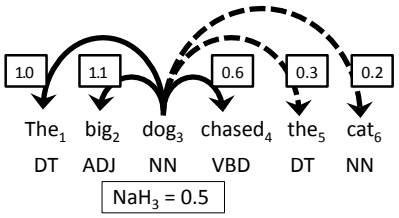


Figure 2: A hypothetical example of dynamic thresholding, where a weak assertion that $dog_3$ should not be a head $\left(\bar{w} \cdot \bar{\Phi}(NaH_3) = 0.5\right)$ is sufficient to rule out two arcs. Each arc's threshold $\left(\bar{w} \cdot \bar{\Phi}(None_a)\right)$ is shown next to its arrow.

## 3 Accounting for Arc Length

We can extend our system by expanding our event set Z. By adding an arc-level event $Vine_a$ to each $\mathrm{Z}_a$, we can introduce a **vine filter** to prune long arcs. Similarly, we have already introduced another arc-level event, the rejection event $None_a$. By assigning features to $None_a$, we learn a **dynamic threshold** on all filters, which considers properties of the arc before acting on any other event. We parameterize both $Vine_a$ and $None_a$ with the same two features, inspired by tag-specific vine parsing (Eisner and Smith, 2005):

$$\left\{ \begin{array}{ccc} \mathrm{Bias} & : & 1 \\ \mathrm{HeadTag\_ModTag\_Dir}(a) & : & \mathrm{Len}(a) \end{array} \right\}$$

where $\mathrm{HeadTag\_ModTag\_Dir}(a)$ concatenates the part-of-speech tags of $a$'s head and modifier tokens to its direction (left or right), and $\mathrm{Len}(a)$ gives the unsigned distance between $a$'s head and modifier.

In the context of $Vine_a$, these two features allow the system to learn tag-pair-specific limits on arc length. In the context of $None_a$, these features protect short arcs and arcs that connect frequently-linked tag-pairs, allowing our token-role filters to be more aggressive on arcs that do not have these characteristics. The dynamic threshold also alters our interpretation of filtering events: where before they were either active or inactive, events are now assigned scores, which are compared with the threshold to make final filtering decisions (Figure 2).[3]

[3]Because tokens and arcs are scored independently and coupled only through score comparison, the impact of $Vine_a$ and $None_a$ on classification speed should be no greater than doing vine and token-role filtering in sequence. In practice, it is no slower than running token-role filtering on its own.

## 4 Experiments

We extract dependency structures from the Penn Treebank using the head rules of Yamada and Matsumoto (2003).[4] We divide the Treebank into train (sections 2–21), development (22) and test (23). We part-of-speech tag our data using a perceptron tagger similar to the one described by Collins (2002). The training set is tagged with jack-knifing: the data is split into 10 folds and each fold is tagged by a system trained on the other 9 folds. Development and test sets are tagged using the entire training set.

We train our joint filter using an in-house latent SVM framework, which repeatedly calls a multi-class exponentiated gradient SVM (Collins et al., 2008). LSVM training was stopped after 4 iterations, as determined during development.[5] For the token-role classifiers, we re-implement the Bergsma and Cherry (2010) feature set, initializing $\bar{w}$ with high-precision subclassifiers trained independently for each token-role. $Vine$ and $None$ subclassifiers are initialized with a zero vector. At test time, we extract subclassifiers from the joint weight vector, and use them as parameters in the filtering tools of Bergsma and Cherry (2010).[6]

Parsing experiments are carried out using the MST parser (McDonald et al., 2005),[7] which we have modified to filter arcs before carrying out feature extraction. It is trained using 5-best MIRA (Crammer and Singer, 2003).

Following Bergsma and Cherry (2010), we measure intrinsic filter quality with **reduction**, the proportion of total arcs removed, and **coverage**, the proportion of true arcs retained. For parsing results, we present dependency **accuracy**, the percentage of tokens that are assigned the correct head.

### 4.1 Impact of Joint Training

Our technical contribution consists of our proposed joint training scheme for token-role filters, along with two extensions: the addition of vine filters ($Vine$) and a dynamic threshold ($None$). Using parameters determined to perform well during development,[8] we examine test-set performance as we incorporate each of these components. For the token-role and vine subclassifiers, we compare against an independently-trained ensemble of the same classifiers.[9] Note that $None$ cannot be trained independently, as its shared dynamic threshold considers arc and token views of the data simultaneously. Results are shown in Table 1.

| | Indep. | | Joint | |
|---|---|---|---|---|
| System | Cov. | Red. | Cov. | Red. |
| Token | 99.73 | 60.5 | 99.71 | 59.0 |
| + Vine | 99.62 | 68.6 | 99.69 | 63.3 |
| + None | N/A | | 99.76 | 71.6 |

Table 1: Ablation analysis of intrinsic filter quality.

Our complete system outperforms all variants in terms of both coverage and reduction. However, one can see that neither joint system is able to outperform its independently-trained counter-part without the dynamic threshold provided by $None$. This is because the desirable credit-assignment properties of our joint training procedure are achieved through duplication (Zadrozny et al., 2003). That is, the LSVM knows that a specific event is important because it appears in event sets $\mathrm{Z}_a$ for many arcs from the same sentence. Without $None$, the filtering decisions implied by each copy of an event are identical. Because these replicated events are associated with arcs that are presented to the LSVM as independent examples, they appear to be not only important, but also low-variance, and therefore easy. This leads to overfitting. We had hoped that the benefits of joint training would outweigh this drawback, but our results show that they do not. However, in addition to its other desirable properties (protecting short arcs), the dynamic threshold imposed by $None$ restores independence between arcs that share a common event (Figure 2). This alleviates overfitting and enables strong performance.

[4] As implemented at `http://w3.msi.vxu.se/~nivre/research/Penn2Malt.html`

[5] The LSVM is well on its way to convergence: fewer than 3% of arcs have event assignments that are still in flux.

[6] `http://code.google.com/p/arcfilter/`. Since our contribution is mainly in better filter training, we were able to use the arcfilter (testing) code with only small changes. We have added our new joint filter, along with the Joint P1 model to the arcfilter package, labeled as `ultra filters`.

[7] `http://sourceforge.net/projects/mstparser/`

[8] $C_0$=1e-2, $C_1$=1e-5

[9] Each subclassifier is a token-level SVM trained with token-role labels extracted from the training treebank. Using development data, we search over regularization parameters so that each classifier yields more than 99.93% arc-level coverage.

| | Filter Intrinsic | | | MST-1 | | MST-2 | |
|---|---|---|---|---|---|---|---|
| Filter | Cov. | Red. | Time | Acc. | Sent/sec* | Acc. | Sent/sec* |
| None | 100.00 | 00.0 | 0s | 91.28 | 16 | 92.05 | 10 |
| B&C R+L | 99.70 | 54.1 | 7s | 91.24 | 29 | 92.00 | 17 |
| **Joint P1** | **99.76** | 71.6 | 7s | 91.28 | 38 | **92.06** | 22 |
| B&C R+L+Q | 99.43 | **78.3** | 19s | 91.23 | 35 | 91.98 | 22 |
| **Joint P2** | 99.56 | 77.9 | 7s | **91.29** | **44** | 92.05 | **25** |

Table 2: Parsing with jointly-trained filters outperforms independently-trained filters (R+L), as well as a more complex cascade (R+L+Q). *Accounts for total time spent parsing and applying filters, averaged over five runs.

### 4.2 Comparison to the state of the art

We directly compare our filters to those of Bergsma and Cherry (2010) in terms of both intrinsic filter quality and impact on the MST parser. The B&C system consists of three stages: rules (R), linear token-role filters (L) and quadratic arc filters (Q). The Q stage uses rich arc-level features similar to those of the MST parser. We compare against independently-trained token-role filters (R+L), as well as the complete cascade (R+L+Q), using the models provided online.[10] Our comparison points, Joint P1 and P2 were built by tuning our complete joint system to roughly match the coverage values of R+L and R+L+Q on development data.[11] Results are shown in Table 2.

Comparing Joint P1 to R+L, we can see that for a fixed set of pointwise filters, joint training with a dynamic threshold outperforms independent training substantially. We achieve a 32% improvement in reduction with no impact on coverage and no increase in filtering overhead (time).

Comparing Joint P2 to R+L+Q, we see that Joint P2 achieves similar levels of reduction with far less filtering overhead; our filters take only 7 seconds to apply instead of 19. This increases the speed of the (already fast) filtered MST-1 parser from 35 sentences per second to 44, resulting in a total speed-up of 2.75 with respect to the unfiltered parser. The improvement is less impressive for MST-2, where the overhead for filter application is a less substantial fraction of parsing time; however, our training framework also has other benefits with respect to R+L+Q, including a single unified training algorithm, fewer hyper-parameters and a smaller test-time memory footprint. Finally, the jointly trained filters have no impact on parsing accuracy, where both B&C filters have a small negative effect.

The performance of Joint-P2+MST-2 is comparable to the system of Huang and Sagae (2010), who report a parsing speed of 25 sentences per second and an accuracy of 92.1 on the same test set, using a *transition-based* parser enhanced with dynamic-programming state combination.[12] Graph-based and transition-based systems tend to make different types of errors (McDonald and Nivre, 2007). Therefore, having fast, accurate parsers for both approaches presents an opportunity for large-scale, robust parser combination.

## 5 Conclusion

We have presented a novel use of latent SVM technology to train a number of filters jointly, with a shared dynamic threshold. By training a family of dependency filters in this manner, each subclassifier focuses on the examples where it is most needed, with our dynamic threshold adjusting filter sensitivity based on arc length. This allows us to outperform a 3-stage filter cascade in terms of speed-up, while also reducing the impact of filtering on parsing accuracy. Our filtering code and trained models are available online at `http://code.google.com/p/arcfilter`. In the future, we plan to apply our joint training technique to other rich filtering regimes (Zhang et al., 2010), and to other NLP problems that combine the predictions of overlapping classifiers.

[10] Results are not identical to those reported in our previous paper, due to our use of a different part-of-speech tagger. Note that parsing accuracies for the B&C systems have improved.

[11] P1: $C_0$=1e-2, $C_1$=1e-5, P2: $C_0$=1e-2, $C_1$=2e-5

[12] The usual caveats for cross-machine, cross-implementation speed comparisons apply.

## References


Shane Bergsma and Colin Cherry. 2010. Fast and accurate arc filtering for dependency parsing. In *COLING*.

Ming-Wei Chang, Dan Goldwasser, Dan Roth, and Vivek Srikumar. 2010. Discriminative learning over constrained latent representations. In *HLT-NAACL*.

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

Michael Collins, Amir Globerson, Terry Koo, Xavier Carreras, and Peter L. Bartlett. 2008. Exponentiated gradient algorithms for conditional random fields and max-margin markov networks. *JMLR*, 9:1775–1822.

Michael Collins. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *EMNLP*.

Koby Crammer and Yoram Singer. 2003. Ultraconservative online algorithms for multiclass problems. *JMLR*, 3:951–991.

Markus Dreyer, David A. Smith, and Noah A. Smith. 2006. Vine parsing and minimum risk reranking for speed and precision. In *CoNLL*.

Jason Eisner and Noah A. Smith. 2005. Parsing with soft and hard constraints on dependency length. In *IWPT*.

Pedro F. Felzenszwalb, Ross B. Girshick, David McAllester, and Deva Ramanan. 2010. Object detection with discriminatively trained part based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(9).

Liang Huang and Kenji Sagae. 2010. Dynamic programming for linear-time incremental parsing. In *ACL*.

Mark Johnson. 2007. Transforming projective bilexical dependency grammars into efficiently-parsable CFGs with unfold-fold. In *ACL*.

Ryan McDonald and Joakim Nivre. 2007. Characterizing the errors of data-driven dependency parsing models. In *EMNLP-CoNLL*.

Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005. Online large-margin training of dependency parsers. In *ACL*.

Igor A. Meĺčuk. 1987. *Dependency syntax: theory and practice*. State University of New York Press.

Brian Roark and Kristy Hollingshead. 2008. Classifying chart cells for quadratic complexity context-free inference. In *COLING*.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *IWPT*.

Ainur Yessenalina, Yisong Yue, and Claire Cardie. 2010. Multi-level structured models for document-level sentiment classification. In *EMNLP*.

Chun-Nam John Yu and Thorsten Joachims. 2009. Learning structural SVMs with latent variables. In *ICML*.

Bianca Zadrozny, John Langford, and Naoki Abe. 2003. Cost-sensitive learning by cost-proportionate example weighting. In *Third IEEE International Conference on Data Mining*.

Yue Zhang, Byung-Gyu Ahn, Stephen Clark, Curt Van Wyk, James R. Curran, and Laura Rimell. 2010. Chart pruning for fast lexicalised-grammar parsing. In *EMNLP*.