Hidden–Variable Models for Discriminative Reranking

Terry Koo
MIT CSAIL
maestro@mit.edu

Michael Collins
MIT CSAIL
mcollins@csail.mit.edu

Abstract

We describe a new method for the representation of NLP structures within reranking approaches. We make use of a conditional log–linear model, with hidden variables representing the assignment of lexical items to word clusters or word senses. The model learns to automatically make these assignments based on a discriminative training criterion. Training and decoding with the model requires summing over an exponential number of hidden–variable assignments: the required summations can be computed efficiently and exactly using dynamic programming. As a case study, we apply the model to parse reranking. The model gives an F–measure improvement of $\approx 1.25\%$ beyond the base parser, and an $\approx 0.25\%$ improvement beyond the Collins (2000) reranker. Although our experiments are focused on parsing, the techniques described generalize naturally to NLP structures other than parse trees.

1 Introduction

A number of recent approaches in statistical NLP have focused on *reranking* algorithms. In reranking methods, a baseline model is used to generate a set of candidate output structures for each input in training or test data. A second model, which typically makes use of more complex features than the baseline model, is then used to rerank the candidates proposed by the baseline. Reranking approaches have given improvements in accuracy on a number of NLP problems including parsing (Collins, 2000; Charniak and Johnson, 2005), machine translation (Och and Ney, 2002; Shen et al., 2004), information extraction (Collins, 2002), and natural language generation (Walker et al., 2001).

The success of reranking approaches depends critically on the choice of *representation* used by the reranking model. Typically, each candidate structure (e.g., each parse tree in the case of parsing) is mapped to a feature–vector representation. Previous work has generally relied on two approaches to representation: explicitly hand–crafted features (e.g., in Charniak and Johnson (2005)) or features defined through kernels (e.g., see Collins and Duffy (2002)).

This paper describes a new method for the representation of NLP structures within reranking approaches. We build on the intuition that lexical items in natural language often fall into word clusters (for example, *president* and *chairman* might belong to the same cluster) or fall into distinct word senses (e.g., *bank* might have two distinct senses). Our method involves a hidden–variable model, where the hidden variables correspond to an assignment of words to either clusters or word–senses. Lexical items are automatically assigned their hidden values using unsupervised learning within a discriminative reranking approach.

We make use of a conditional log–linear model for our task. Formally, hidden variables within the log–linear model consist of *global* assignments, where a global assignment entails an assignment of every word in the sentence to some hidden cluster or sense value. The number of such global assignments grows exponentially fast with the length of the sentence being processed. Training and decoding with the model requires summing over the exponential number of possible global assignments, a major technical challenge in our model. We show that the required summations can be computed efficiently and exactly using dynamic–programming methods (i.e., the belief propagation algorithm for Markov random fields (Yedidia et al., 2003)) under certain restrictions on features in the model.

Previous work on reranking has made heavy use of lexical statistics, but has treated lexical items as atoms. The motivation for our method comes from the observation that statistics based on lexical items are critical, but that these statistics suffer considerably from problems of data sparsity and word–

sense polysemy. Our model has the ability to alleviate data sparsity issues by learning to assign words to word clusters, and can mitigate problems with word–sense polysemy by learning to assign lexical items to underlying word senses based upon contextual information. A critical difference between our method and previous work on unsupervised approaches to word–clustering or word–sense discovery is that our model is trained using a discriminative criterion, where the assignment of words to clusters or senses is driven by the reranking task in question.

As a case study, in this paper we focus on syntactic parse reranking. We describe three model types that can be captured by our approach. The first method emulates a clustering operation, where the aim is to place similar words (e.g., *president* and *chairman*) into the same cluster. The second method emulates a *refinement* operation, where the aim is to recover distinct senses underlying a single word (for example, distinct senses underlying the noun *bank*). The third definition makes use of an existing ontology (i.e., WordNet (Miller et al., 1993)). In this case the set of possible hidden values for each word corresponds to possible WordNet senses for the word.

In experimental results on the Penn Wall Street Journal treebank parsing domain, the hidden–variable model gives an F–measure improvement of $\approx 1.25\%$ beyond a baseline model (the parser described in Collins (1999)), and gives an $\approx 0.25\%$ improvement beyond the reranking approach described in Collins (2000). Although the experiments in this paper are focused on parsing, the techniques we describe generalize naturally to other NLP structures such as strings or labeled sequences. We discuss this point further in Section 6.1.

2 Related Work

Various machine–learning methods have been used within reranking tasks, including conditional log–linear models (Ratnaparkhi et al., 1994; Johnson et al., 1999), boosting methods (Collins, 2000), variants of the perceptron algorithm (Collins, 2002; Shen et al., 2004), and generalizations of support–vector machines (Shen and Joshi, 2003). There have been several previous approaches to parsing using log–linear models and hidden variables. Riezler et al. (2002) describe a discriminative LFG parsing model that is trained on standard (syntax only) treebank annotations by treating each tree as a full LFG analysis with an observed c-structure and hidden f-structure. Clark and Curran (2004) present an alternative CCG parsing approach that divides each CCG parse into a dependency structure (observed) and a derivation (hidden). More recently, Matsuzaki et al. (2005) introduce a probabilistic CFG augmented with hidden information at each nonterminal, which gives their model the ability to tailor itself to the task at hand. The form of our model is closely related to that of Quattoni et al. (2005), who describe a hidden–variable model for object recognition in computer vision.

The approaches of Riezler et al., Clark and Curran, and Matsuzaki et al. are similar to our own work in that the hidden variables are exponential in number and must be handled with dynamic–programming techniques. However, they differ from our approach in the definition of the hidden variables (the Matsuzaki et al. model is the most similar). In addition, these three approaches don't use reranking, so their features must be restricted to local scope in order to allow dynamic–programming approaches to training. Finally, these approaches use Viterbi or other approximations during decoding, something our model can avoid (see section 6.2).

In some instantiations, our model effectively clusters words into categories. Our approach differs from standard word clustering in that the clustering criteria is directly linked to the reranking objective, whereas previous word–clustering approaches (e.g. Brown et al. (1992) or Pereira et al. (1993)) have typically leveraged distributional similarity. In other instantiations, our model establishes word–sense distinctions. Bikel (2000) has done previous work on incorporating the WordNet hierarchy into a generative parsing model; however, this approach requires data with word–sense annotations whereas our model deals with word–sense ambiguity through unsupervised discriminative training.

3 The Hidden–Variable Model

In this section we describe a hidden–variable model based on conditional log–linear models. Each sentence s_i for $i = 1 \ldots n$ in our training data has a set of n_i candidate parse trees $t_{i,1}, \ldots, t_{i,n_i}$, which are the output of an N–best baseline parser. Each candidate parse has an associated F–measure score,

indicating its similarity to the gold–standard parse. Without loss of generality, we define $t_{i,1}$ to be the parse with the highest F–measure for sentence s_i.

Given a candidate parse tree $t_{i,j}$, the hidden–variable model assigns a domain of hidden values to each word in the tree. For example, the hidden–value domain for the word *bank* could be $\{bank_1, bank_2, bank_3\}$ or $\{\mathsf{NN}_1, \mathsf{NN}_2, \mathsf{NN}_3\}$. Detailed descriptions of the domains we used are given in Section 4.1. Formally, if $t_{i,j}$ spans m words then the hidden–value domains for each word are the sets $H_1(t_{i,j}), \ldots, H_m(t_{i,j})$. A global hidden–value assignment, which attaches a hidden value to every word in $t_{i,j}$, is written $\mathbf{h} = (h_1, \ldots, h_m) \in \mathbf{H}(t_{i,j})$, where $\mathbf{H}(t_{i,j}) = H_1(t_{i,j}) \times \ldots \times H_m(t_{i,j})$ is the set of all possible global assignments for $t_{i,j}$.

We define a feature–based representation Φ such that $\Phi(t_{i,j}, \mathbf{h}) \in \mathbb{R}^d$ is a vector of feature occurrence counts that describes candidate parse $t_{i,j}$ with global assignment $\mathbf{h} \in \mathbf{H}(t_{i,j})$. We write Φ_k for $k = 1 \ldots d$ to denote the k^{th} component of the vector Φ. Each component of the feature vector is the count of some substructure within $(t_{i,j}, \mathbf{h})$. For example, Φ_{12} and Φ_{101} could be defined as follows:

$$\begin{aligned} \Phi_{12}(t_{i,j}, \mathbf{h}) &= \text{Number of times the word } \textit{the} \text{ occurs with hidden value } \textit{the}_3 \text{ and part of speech tag DT in } (t_{i,j}, \mathbf{h}). \\ \Phi_{101}(t_{i,j}, \mathbf{h}) &= \text{Number of times } \textit{CEO}_1 \text{ appears as the subject of } \textit{owns}_2 \text{ in } (t_{i,j}, \mathbf{h}) \end{aligned} \quad (1)$$

We use a parameter vector $\Theta \in \mathbb{R}^d$ to define a log–linear distribution over candidate trees together with global hidden–value assignments:

$$p(t_{i,j}, \mathbf{h} \mid s_i, \Theta) = \frac{e^{\Phi(t_{i,j}, \mathbf{h}) \cdot \Theta}}{\sum_{j', \mathbf{h}' \in \mathbf{H}(t_{i,j'})} e^{\Phi(t_{i,j'}, \mathbf{h}') \cdot \Theta}}$$

By marginalizing out the global assignments, we obtain a distribution over the candidate parses alone:

$$p(t_{i,j} \mid s_i, \Theta) = \sum_{\mathbf{h} \in \mathbf{H}(t_{i,j})} p(t_{i,j}, \mathbf{h} \mid s_i, \Theta) \quad (2)$$

Later in this paper we will describe how to train the parameters of the model by minimizing the following loss function—which is the negative log–likelihood of the training data—with respect to Θ:

$$\begin{aligned} L(\Theta) &= -\sum_i \log p(t_{i,1} \mid s_i, \Theta) \\ &= -\sum_i \log \textstyle\sum_{\mathbf{h} \in \mathbf{H}(t_{i,1})} p(t_{i,1}, \mathbf{h} \mid s_i, \Theta) \end{aligned} \quad (3)$$

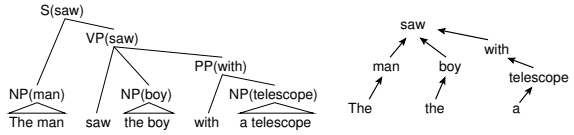


Figure 1: A sample parse tree and its dependency tree.

3.1 Local Feature Vectors

Note that the number of possible global assignments (i.e., $|\mathbf{H}(t_{i,j})|$) grows exponentially fast with respect to the number of words spanned by $t_{i,j}$. This poses a problem when training the model, or when calculating the probability of a parse tree through Eq. 2. This section describes how to address this difficulty by restricting features to sufficiently local scope. In Section 3.2 we show that this restriction allows efficient training and decoding of the model.

The restriction to local feature–vectors makes use of the dependency structure underlying the parse tree $t_{i,j}$. Formally, for tree $t_{i,j}$, we define the corresponding dependency tree $\mathbf{D}(t_{i,j})$ to be a set of edges between words in $t_{i,j}$, where $(u, v) \in \mathbf{D}(t_{i,j})$ if and only if there is a head–modifier dependency between words u and v. See Figure 1 for an example dependency tree. We restrict the definition of Φ in the following way[1]. If w, u and v are word indices, we introduce single–variable local feature vectors $\phi(t_{i,j}, w, h_w) \in \mathbb{R}^d$ and pairwise local feature vectors $\phi(t_{i,j}, u, v, h_u, h_v) \in \mathbb{R}^d$. The global feature vector $\Phi(t_{i,j}, \mathbf{h})$ is then decomposed into a sum over the local feature vectors:

$$\Phi(t_{i,j}, \mathbf{h}) = \sum_{1 \le w \le m} \phi(t_{i,j}, w, h_w) + \sum_{(u,v) \in \mathbf{D}(t_{i,j})} \phi(t_{i,j}, u, v, h_u, h_v) \quad (4)$$

Notice that the second sum, over pairwise local feature vectors, respects the dependency structure $\mathbf{D}(t_{i,j})$. Section 3.2 describes how this decomposition of Φ leads to an efficient and exact dynamic–programming approach that, during training, allows us to calculate the gradient $\frac{\partial L}{\partial \Theta}$ and, during testing, allows us to find the most probable candidate parse.

In our implementation, each dimension of the local feature vectors is an indicator function signaling the presence of a feature, so that a sum over local feature vectors in a tree gives the occurrence count

[1]Note that the restriction on local feature vectors only concerns the inclusion of hidden values; features may still observe arbitrary structure within the underlying parse tree $t_{i,j}$.

of features in that tree. For instance, define

$$\phi_{12}(t_{i,j}, w, h_w) = \left[\!\!\left[\begin{array}{l} h_w = \textit{the}_3 \text{ and tree } t_{i,j} \text{ assigns word} \\ w \text{ to part–of–speech } \mathsf{DT} \end{array} \right]\!\!\right]$$

$$\phi_{101}(t_{i,j}, u, v, h_u, h_v) = \left[\!\!\left[\begin{array}{l} (h_u, h_v) = (\textit{CEO}_1, \textit{owns}_2) \\ \text{and tree } t_{i,j} \text{ places } (u, v) \text{ in} \\ \text{a subject–verb relationship} \end{array} \right]\!\!\right]$$

where the notation $[\![\mathcal{P}]\!]$ signifies a $0/1$ indicator of predicate $\mathcal{P}$. When summed over the tree, these definitions of ϕ_{12} and ϕ_{101} yield global features Φ_{12} and Φ_{101} as given in the previous example (see Eq. 1).

3.2 Training the Model

We now describe how the loss function in Eq. 3 can be optimized using gradient descent. The gradient of the loss function is given by:

$$\frac{\partial L}{\partial \Theta} = -\sum_i F(t_{i,1}, \Theta) + \sum_{i,j} p(t_{i,j} \mid s_i, \Theta) F(t_{i,j}, \Theta)$$

where $F(t_{i,j}, \Theta) = \sum_{\mathbf{h} \in \mathbf{H}(t_{i,j})} \frac{p(t_{i,j}, \mathbf{h} \mid s_i, \Theta)}{p(t_{i,j} \mid s_i, \Theta)} \Phi(t_{i,j}, \mathbf{h})$ is the expected value of the feature vector produced by parse tree $t_{i,j}$. As we remarked earlier, $|\mathbf{H}(t_{i,j})|$ is exponential in size so direct calculation of either $p(t_{i,j} \mid s_i, \Theta)$ or $F(t_{i,j}, \Theta)$ is impractical. However, using the feature–vector decomposition in Eq. 4, we can rewrite the key functions of Θ as follows:

$$p(t_{i,j} \mid s_i, \Theta) = \frac{Z_{i,j}}{\sum_{j'} Z_{i,j'}}$$

$$F(t_{i,j}, \Theta) = \sum_{\substack{1 \le w \le m \\ h_w \in H_w(t_{i,j})}} p(t_{i,j}, w, h_w) \phi(t_{i,j}, w, h_w) + \sum_{\substack{(u,v) \in \mathbf{D}(t_{i,j}) \\ h_u \in H_u(t_{i,j}) \\ h_v \in H_v(t_{i,j})}} p(t_{i,j}, u, v, h_u, h_v) \phi(t_{i,j}, u, v, h_u, h_v)$$

where $p(t_{i,j}, w, h_w)$ and $p(t_{i,j}, u, v, h_u, h_v)$ are marginalized probabilities and $Z_{i,j}$ is the associated normalization constant:

$$Z_{i,j} = \sum_{\mathbf{h} \in \mathbf{H}(t_{i,j})} e^{\Phi(t_{i,j}, \mathbf{h}) \cdot \Theta}$$

$$p(t_{i,j}, w, x) = \sum_{\mathbf{h} \mid h_w = x} p(t_{i,j}, \mathbf{h} \mid s_i, \Theta)$$

$$p(t_{i,j}, u, v, x, y) = \sum_{\mathbf{h} \mid h_u = x, h_v = y} p(t_{i,j}, \mathbf{h} \mid s_i, \Theta)$$

The three quantities above can be computed with belief propagation (Yedidia et al., 2003), a dynamic–programming technique that is efficient[2] and exact when the graph $\mathbf{D}(t_{i,j})$ is a tree, which is the case in our parse reranking model. Having calculated the gradient in this way, we minimize the loss using stochastic gradient descent[3] (LeCun et al., 1998).

4 Features for Parse Reranking

The previous section described hidden–variable models for discriminative reranking. We now describe features for the parse reranking problem. We focus on the definition of hidden–value domains and local feature vectors in the reranking model.

4.1 Hidden–Value Domains and Local Features

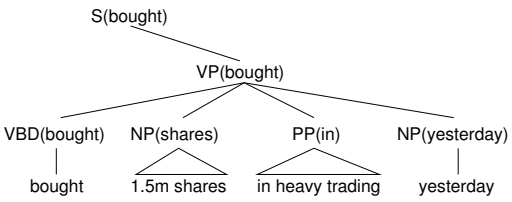
Each word in a parse tree is given a domain of possible hidden values by the hidden–variable model. Models with widely varying behavior can be created by changing the way these domains are defined. In particular, in this section we will see how different definitions of the domains give rise to the three main model types: clustering, refinement, and mapping into a pre–built ontology such as WordNet.

As illustration, consider a simple approach that splits each word into a domain of three word–sense hidden values (e.g., the word *bank* would yield the domain $\{\textit{bank}_1, \textit{bank}_2, \textit{bank}_3\}$). In this approach, each word receives a domain of hidden values that is not shared with any other word. The model is then able to distinguish several different usages for each word, emulating a refinement operation. An alternative approach is to split each word's part–of–speech tag into several sub–tags (e.g., *bank* would yield $\{\mathsf{NN}_1, \mathsf{NN}_2, \mathsf{NN}_3\}$). This approach assigns the same domain to many words; for instance, singular nouns such as *bond*, *market*, and *bank* all receive the same domain. The behavior of the model then emulates a clustering operation.

Figure 2 shows the single–variable and pairwise features used in our experiments. The features are shown with hidden variables corresponding to word–specific hidden values, such as $\textit{shares}_1$ or $\textit{bought}_3$. In our experiments, we made use of features such as those in Figure 2 in combination with the following four definitions of the hidden–value

[2] The running time of belief propagation varies linearly with the number of nodes in $\mathbf{D}(t_{i,j})$ and quadratically with the cardinality of the largest hidden–value domain.

[3] We also performed some experiments using the conjugate gradient descent algorithm (Johnson et al., 1999). However, we did not find a significant difference between the performance of either method. Since stochastic gradient descent was faster and required less memory, our final experiments used the stochastic gradient method.



Single–variable features generated for $(shares_1)$ =
$(shares_1)$
$(shares_1,$ Word: *shares*$)$
$(shares_1,$ POS: NN$)$
$(shares_1,$ Word: *shares*, POS: NN$)$
$(shares_1,$ Highest NT: NP$)$
$(shares_1,$ Word: *shares*, Highest NT: NP$)$
$(shares_1,$ POS: NN, Highest NT: NP$)$
$(shares_1,$ Word: *shares*, POS: NN, Highest NT: NP$)$
$(shares_1,$ Up Path: NP, VP, S$)$
$(shares_1,$ Word: *shares*, Up Path: NP, VP, S$)$
$(shares_1,$ Down Path: NP, NN$)$
$(shares_1,$ Word: *shares*, Down Path: NP, NN$)$
$(shares_1,$ Head Rule: NP $\rightarrow$ CD, CD, $\text{NNS}_{\text{head}}$$)$
$(shares_1,$ Word: *shares*, Head Rule: NP $\rightarrow$ CD, CD, $\text{NNS}_{\text{head}}$$)$
$(shares_1,$ Mod Rule: VP $\rightarrow$ VBD_{head}, NP_{mod}, PP, NP$)$
$(shares_1,$ Word: *shares*, Mod Rule: VP $\rightarrow$ VBD_{head}, NP_{mod}, PP, NP$)$
$(shares_1,$ Head Gpar Rule: VP $\rightarrow$ NP $\rightarrow$ CD, CD, $\text{NNS}_{\text{head}}$$)$
$(shares_1,$ Word: *shares*, Head Gpar Rule: VP $\rightarrow$ NP $\rightarrow$ CD, CD, $\text{NNS}_{\text{head}}$$)$
$(shares_1,$ Mod Gpar Rule: S $\rightarrow$ VP $\rightarrow$ VBD_{head}, NP_{mod}, PP, NP$)$
$(shares_1,$ Word: *shares*, Mod Gpar Rule: S $\rightarrow$ VP $\rightarrow$ VBD_{head}, NP_{mod}, PP, NP$)$

Pairwise features generated for $(shares_1, bought_3)$ =
$(shares_1, bought_3,$ Dependency: VBD, NP, VP, Right, +Adj, -CC$)$
$(shares_1, bought_3,$ Rule: VP $\rightarrow$ VBD_{head}, NP_{mod}, PP, NP$)$
$(shares_1, bought_3,$ Gpar Rule: S $\rightarrow$ VP $\rightarrow$ VBD_{head}, NP_{mod}, PP, NP$)$

Figure 2: The features used in our model. We show the single–variable features produced for hidden value $shares_1$ and the pairwise features produced for hidden values $(shares_1, bought_3)$, in the context of the given parse fragment. Highest NT = highest nonterminal, Up Path = sequence of ancestor nonterminals, Down Path = sequence of headed nonterminals, Head Rule = rules headed by the word, Mod Rule = rule in which word acts as modifier, Head/Mod Gpar Rule = Head/Mod Rule plus grandparent nonterminal.

domains (in each case we give the model type that results from the definition—clustering, refinement, or pre–built ontology—in parentheses):

Lexical (Refinement) Each word is split into three sub–values. See Figure 2 for an example of features generated for this choice of domain.

Part–of–Speech (Clustering) The part–of–speech tag of each word is split into five sub–values. In Figure 2, the word *shares* would be assigned the domain $\{\text{NNS}_1, \ldots, \text{NNS}_5\}$, and the word *bought* would have the domain $\{\text{VBD}_1, \ldots, \text{VBD}_5\}$.

Highest Nonterminal (Clustering) The highest nonterminal to which each word propagates as a headword is split into five sub–values. In Figure 2 the word *bought* yields domain $\{\text{S}_1, \ldots, \text{S}_5\}$, while *in* yields $\{\text{PP}_1, \ldots, \text{PP}_5\}$.

Supersense (Pre–Built Ontology) We borrow the idea of using WordNet lexicographer filenames as broad "supersenses" from Ciaramita and Johnson (2003). For each word, we split each of its supersenses into three sub–supersenses. If no supersenses are available, we fall back to splitting the part–of–speech into five sub–values. For example, *shares* has the supersenses noun.possession, noun.act and noun.artifact, which yield the domain $\{\text{noun.possession}_1, \text{noun.act}_1, \text{noun.artifact}_1, \ldots$ $\text{noun.possession}_3, \text{noun.act}_3, \text{noun.artifact}_3\}$. On the other hand, *in* does not have any WordNet supersenses, so it is assigned the domain $\{\text{IN}_1, \ldots, \text{IN}_5\}$.

4.2 The Final Feature Sets

We created eight feature sets by combining the four hidden–value domains above with two alternative definitions of dependency structures: standard head–modifier dependencies and "sibling dependencies." When using sibling dependencies, connections are established between the headwords of adjacent siblings. For instance, the head–modifier dependencies produced by the tree fragment in Figure 2 are $(\textit{bought}, \textit{shares})$, $(\textit{bought}, \textit{in})$, and $(\textit{bought}, \textit{yesterday})$, while the corresponding sibling dependencies are $(\textit{bought}, \textit{shares})$, $(\textit{shares}, \textit{in})$, and $(\textit{in}, \textit{yesterday})$.

4.3 Mixed Models

The different hidden–variable models display varying strengths and weaknesses. We created mixtures of different models using a weighted average:

$$\log p(t_{i,j}|s_i) = \sum_{m=1}^{M} \lambda_m \log p_m(t_{i,j}|s_i, \Theta_m) - Z(s_i)$$

where $Z(s_i)$ is a normalization constant that can be ignored, as it does not affect the ranking of parses. The λ_m weights are determined through optimization of parsing scores on a development set.

5 Experimental Results

We trained and tested the model on data from the Penn Treebank (Marcus et al., 1994). Candidate parses were produced by an N–best version of the Collins (1999) parser. Our training data consists of Treebank Sections 2–21, divided into a training corpus of 35,540 sentences and a development corpus of 3,676 sentences. In later experiments, we made use of a secondary development corpus of 1,914 sentences from Section 0. Sections 22–24, containing 5,455 sentences, were held out as the test set.

For each of the eight feature sets described in Section 4.2, we used the stochastic gradient descent

	Section 22		Section 23		Section 24		Total	
	LR	LP	LR	LP	LR	LP	LR	LP
C99	89.12	89.20	88.14	88.56	87.17	87.97	88.19	88.60
MIX	90.43	90.61	89.25	89.69	88.46	89.29	89.41	89.87
C2K	90.27	90.62	89.43	89.97	88.56	89.58	89.46	90.07
MIX+	90.57	90.79	89.80	90.27	88.78	89.73	89.78	90.29

Table 1: The results on Sections 22–24. **LR** = Labeled Recall, **LP** = Labeled Precision.

method to optimize the parameters of the model. We created various mixtures of the eight models using the weighted–average technique described in Section 4.3, testing the accuracy of each mixture on the secondary development set. Our final model was a mixture of three of the eight possible models: supersense hidden values with sibling trees, lexical hidden values with sibling trees, and highest nonterminal hidden values with normal head–modifier trees.

Our final tests evaluated four models. The two baseline models are the Collins (1999) base parser, C99, and the Collins (2000) reranker, C2K. The first new model is the MIX model, which is a combination of the C99 base model with the three models described above. The second new model, MIX+, is created by augmenting MIX with features from the method in C2K. Table 1 shows the results. The MIX model obtains an F–measure improvement of $\approx 1.25\%$ over the C99 baseline, an improvement that is comparable to the C2K reranker. The MIX+ model yields an improvement of $\approx 0.25\%$ beyond C2K.

We tested the significance of 8 comparisons corresponding to the results in Table 1 using the sign test[4]: we tested MIX vs. C99 on Sections 22, 23, and 24 individually, as well as on Sections 22–24 taken as a whole; we also tested MIX+ vs. C2K on these 4 test sets. Of the 8 comparisons, all showed significant improvements at the level $p \leq 0.01$ with the exception of one test, MIX+ vs. C2K on Section 24.

6 Discussion

6.1 Applying the Model to Other NLP Tasks

In this section, we discuss how hidden–variable models might be applied to other NLP problems, and in particular to structures other than parse trees. To summarize the model, the major components of the approach are as follows:

• We assume some set of candidate structures $t_{i,j}$, which are to be reranked by the model. Each structure $t_{i,j}$ has $n_{i,j}$ words $w_1, \ldots, w_{n_{i,j}}$, and each word w_k has a set $H_k(t_{i,j})$ of possible hidden values.

• We assume a graph $\mathbf{D}(t_{i,j})$ for each $t_{i,j}$ that defines possible interactions between hidden variables in the model. We assume some definition of local feature vectors, which consider either single hidden variables, or pairs of hidden variables that are connected by an edge in $\mathbf{D}(t_{i,j})$.

The approach can be instantiated in several ways when applying the model to other NLP tasks. We have already seen that by changing the definition of the hidden–value domains $H_k(t_{i,j})$, we can derive models with widely varying behavior. In addition, there is no requirement that the hidden variables only be associated with words in the structure; the hidden variables could be associated with other units. For example, in speech recognition hidden variables could be associated with phonemes rather than words, and in Chinese word segmentation, hidden variables could be associated with individual characters rather than words.

NLP tasks other than parsing involve structures $t_{i,j}$ that are not necessarily parse trees. For example, in speech recognition candidates are simply strings (utterances); in tagging tasks candidates are labeled sequences (e.g., sentences labeled with part–of–speech tag sequences); in machine translation candidate structures may be source–language/target–language pairs, along with alignment structures specifying the correspondence between words in the two languages. Sentences and labeled sequences are in a sense simplifications of the parsing case, where a natural choice for the underlying graph $\mathbf{D}(t_{i,j})$ would be an N^{th} order Markov structure, where each word depends on the previous N words. Machine translation alignments are a more interesting type of structure, where the choice of $\mathbf{D}(t_{i,j})$ might actually depend on the alignment between the two sentences.

As a final note, there is some flexibility in the choice of $\mathbf{D}(t_{i,j})$. In the case that $\mathbf{D}(t_{i,j})$ is a tree belief propagation is exact. In the more general case where $\mathbf{D}(t_{i,j})$ contains cycles, there are alternative algorithms that are either exact (Cowell et al., 1999) or approximate (Yedidia et al., 2003).

[4]The input to the sign test is a set of sentences with judgements for each sentence indicating whether model 1 gives a better parse than model 2, model 2 gives a better parse than model 1, or models 1 and 2 give equal quality parses. When using the sign test, for each sentence in question we calculate the F–measure at the sentence level for the two models being compared, deriving the required judgement from these scores.

6.2 Packed Representations and Locality

One natural extension of our reranker is to adapt it to candidate parses represented as a packed parse forest, so that it can operate on the base parser's full output instead of a limited N-best list. However, as we described in Section 3.1, our features are locally scoped with respect to hidden–variable interactions but unrestricted regarding information derived from the underlying candidate parses, which poses a problem for the use of packed representations. For instance, the Up/Down Path features (see Figure 2) enumerate the vertical sequences of nonterminals that extend above and below a given headword. We could restrict the features to local scope on the candidate parses, allowing dynamic–programming to be used to train the model with a packed representation. However, even with these restrictions, finding $\arg\max_t \sum_{\mathbf{h}} p(t, \mathbf{h} \mid s, \Theta)$ is NP–hard, and the Viterbi approximation $\arg\max_{t,\mathbf{h}} p(t, \mathbf{h} \mid s, \Theta)$ — or other approximations — would have to be used (see Matsuzaki et al. (2005)).

6.3 Empirical Analysis of the Hidden Values

Our model makes no assumptions about the interpretation of the hidden values assigned to words: during training, the model simply learns a distribution over global hidden–value assignments that is useful in improving the log–likelihood of the training data. Intuitively, however, we expect that the model will learn to make hidden–value assignments that are reasonable from a linguistic standpoint. In this section we describe some empirical observations concerning hidden values assigned by the model.

We established a corpus of parse trees with hidden–value annotations, as follows. First, we find the optimal parameters Θ^* on the training set. For every sentence s_i in the training set, we then use Θ^* to find t_i^*, the most probable candidate parse under the model. Finally, we use Θ^* to decode $\mathbf{h}_i^*$, the most probable global assignment of hidden values, for each parse tree t_i^*. We created a corpus of $(t_i^*, \mathbf{h}_i^*)$ pairs for the feature set defined by part–of–speech hidden–value domains and standard dependency structures. The remainder of this section describes trends for several of the most common part–of–speech categories in the corpus.

As a first example, consider the hidden values for the part–of–speech VB (infinitival verb). In the majority of cases, words tagged VB either modify a modal verb tagged MD (e.g., in *the new rate will*/MD *be*/VB *payable*) or the infinitival marker *to* (e.g., in *in an effort to streamline*/VB *bureaucracy*). The statistics of our corpus reflect this distinction. In 11,546 cases of the VB_1 hidden value, 10,652 cases modified *to*, and 81 cases modified modals tagged MD. In contrast, in 11,042 cases of the VB_2 value, the numbers were 8,354 and 599 for modification of modals and *to* respectively, showing the opposite preference. This polarization of hidden values allows modifiers to the VB (e.g., *payable* in *the new rate will be payable*) to be sensitive to whether the verb is modifying a modal or *to*.

In a related case, the hidden values for the part–of–speech TO (corresponding to the word *to*) also show that the model is learning useful structure. Consider cases where *to* heads a clause which may or may not have a subject (e.g., in *it expects* ⟨*its sales to remain steady*⟩ vs. *a proposal* ⟨*to ease reporting requirements*⟩). We find that for hidden values TO_1 and TO_5 together, 946 out of 976 cases have a subject. In contrast, for the hidden value TO_4, only 29 out of 10,148 cases have a subject. This splitting of the TO part–of–speech allows modifiers to *to*, or words modified by *to*, to be sensitive to the presence or absence of a subject in the clause headed by *to*.

Finally, consider the hidden values for the part–of–speech NNS (plural noun). In this case, the model distinguishes contexts where a plural noun acting as the head of a noun–phrase is or isn't modified by a post–modifier (such as a prepositional phrase or relative clause). For hidden value NNS_3, 12,003 out of the 12,664 instances in our corpus have a post–modifier, but for hidden value NNS_5, only 4,099 of the 39,763 occurrences have a post–modifier. Similar contextual effects were observed for other noun categories such as singular or proper nouns.

7 Conclusions and Future Research

The hidden–variable model is a novel method for representing NLP structures in the reranking framework. We can obtain versatile behavior from the model simply by manipulating the definition of the hidden–value domains, and we have experimented with models that emulate word clustering, word refinement, and mappings from words into an existing ontology. In the case of the parse reranking task,

the hidden–variable model achieves reranking performance comparable to the reranking approach described by Collins (2000), and the two rerankers can be combined to yield an additive improvement.

Future work may consider the use of hidden–value domains with mixed contents, such as a domain that contains 3 refinement–oriented lexical values and 3 clustering–oriented part–of–speech values. These mixed values would allow the hidden–variable model to exploit interactions between clustering and refinement at the level of words and dependencies. Another area for future research is to investigate the use of unlabeled data within the approach, for example by making use of clusters derived from large amounts of unlabeled data (e.g., see Miller et al. (2004)). Finally, future work may apply the models to NLP tasks other than parsing.

Acknowledgements

We would like to thank Regina Barzilay, Igor Malioutov, and Luke Zettlemoyer for their many comments on the paper. We gratefully acknowledge the support of the National Science Foundation (under grants 0347631 and 0434222) and the DARPA/SRI CALO project (through subcontract No. 03-000215).

References

Daniel Bikel. 2000. A statistical model for parsing and word–sense disambiguation. In *Proceedings of EMNLP*.

Peter F. Brown, Vincent J. Della Pietra, Peter V. deSouza, Jennifer C. Lai, and Robert L. Mercer. 1992. Class–based n–gram models of natural language. *Computational Linguistics*, 18(4):467–479.

Eugene Charniak and Mark Johnson. 2005. Coarse–to–fine n–best parsing and maxent discriminative reranking. In *Proceedings of the 43rd ACL*.

Massimiliano Ciaramita and Mark Johnson. 2003. Supersense tagging of unknown nouns in wordnet. In *EMNLP 2003*.

Stephen Clark and James R. Curran. 2004. Parsing the wsj using ccg and log–linear models. In *ACL*, pages 103–110.

Michael Collins and Nigel Duffy. 2002. New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In *ACL 2002*.

Michael Collins. 1999. *Head–Driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania, Philadelphia, PA.

Michael Collins. 2000. Discriminative reranking for natural language parsing. In *Proceedings of the 17th ICML*.

Michael Collins. 2002. Ranking algorithms for named entity extraction: Boosting and the voted perceptron. In *ACL 2002*.

Robert G. Cowell, A. Philip Dawid, Steffen L. Lauritzen, and David J. Spiegelhalter. 1999. *Probabilistic Networks and Expert Systems*. Springer.

Mark Johnson, Stuart Geman, Stephen Canon, Zhiyi Chi, and Stefan Riezler. 1999. Estimators for stochastic "unification–based" grammars. In *Proceedings of the 37th ACL*.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. 1998. Gradient–based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1994. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19(2):313–330.

Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. 2005. Probabilistic cfg with latent annotations. In *ACL*.

George A. Miller, Richard Beckwith, Christiane Fellbaum, Derek Gross, and Katherine Miller. 1993. Five papers on wordnet. Technical report, Stanford University.

Scott Miller, Jethran Guinness, and Alex Zamanian. 2004. Name tagging with word clusters and discriminative training. In *HLT–NAACL*, pages 337–342.

Franz Josef Och and Hermann Ney. 2002. Discriminative training and maximum entropy models for statistical machine translation. In *Proceedings of the 40th ACL*, pages 295–302.

Fernando C. N. Pereira, Naftali Tishby, and Lillian Lee. 1993. Distributional clustering of english words. In *Proceedings of the 31st ACL*, pages 183–190.

Ariadna Quattoni, Michael Collins, and Trevor Darrell. 2005. Conditional random fields for object recognition. In *NIPS 17*. MIT Press.

Adwait Ratnaparkhi, Salim Roukos, and R. Todd Ward. 1994. A maximum entropy model for parsing. In *ICSLP 1994*.

Stefan Riezler, Tracy H. King, Ronald M. Kaplan, Richard S. Crouch, John T. Maxwell III, and Mark Johnson. 2002. Parsing the wall street journal using a lexical–functional grammar and discriminative estimation techniques. In *ACL*.

Libin Shen and Aravind K. Joshi. 2003. An svm–based voting algorithm with application to parse reranking. In Walter Daelemans and Miles Osborne, editors, *Proceedings of the 7th CoNLL*, pages 9–16. Edmonton, Canada.

Libin Shen, Anoop Sarkar, and Franz Josef Och. 2004. Discriminative reranking for machine translation. In *HLT–NAACL*, pages 177–184.

Marilyn A. Walker, Owen Rambow, and Monica Rogati. 2001. Spot: A trainable sentence planner. In *NAACL*.

Jonathan S. Yedidia, William T. Freeman, and Yair Weiss, 2003. *Exploring Artificial Intelligence in the New Millennium*, chapter 8: "Understanding Belief Propagation and Its Generalizations", pages 239–236. Science & Technology Books.