Supplementary Material for EPIC

David Hall and Dan Klein
Computer Science Division
University of California, Berkeley
{dlwh,klein}@cs.berkeley.edu

1 Specifics on the Parsing Models

Here we present some more detail on our individual parsing models, specifically going over the features we use. As with most parsers or other complex natural language processing systems, the overall system is hard to completely describe in text, and so we are releasing our parser at `http://nlp.cs.berkeley.edu/Software.shtml`.

All of the features we use are indicator features. We use Collins head rules to determine heads for the lexicalized parser and for binarization. We describe our lexicon in section 1.4.

1.1 Lexicalized Model

Recall that we have binary rules of the form $A[h] \to B[h]\,C[d]$ and $A[h] \to B[d]\,C[h]$. The feature function φ for this model starts with several indicator features:

1. $\mathbb{I}[A \to B\,C]$
2. $\mathbb{I}[A \to B\,C, \text{HEAD} = \text{ws}_i(w_h)]$
3. $\mathbb{I}[A \to B\,C, \text{HEAD} = \text{tag}(w_h)]$
4. $\mathbb{I}[A \to B\,C, \text{DEP} = \text{ws}_i(w_d)]$
5. $\mathbb{I}[A \to B\,C, \text{DEP} = \text{tag}(w_d)]$
6. $\mathbb{I}[\text{HEAD} = \text{tag}(w_h), \text{DEP} = \text{ws}_i(w_d)]$
7. $\mathbb{I}[\text{HEAD} = \text{tag}(w_h), \text{DEP} = \text{tag}(w_d)]$
8. $\mathbb{I}[\text{HEAD} = \text{ws}_i(w_h), \text{DEP} = \text{ws}_i(w_d)]$
9. $\mathbb{I}[\text{HEAD} = \text{ws}_i(w_h), \text{DEP} = \text{tag}(w_d)]$
10. $\mathbb{I}[\text{DIST=binDistance}(h, d), *]$

where tag maps each words to its most common tag and ws_i is the i'th word shape feature. (See section 1.4 of this note.) binDistance is a function that bins words into 1 of 4 bins, based on their surface distance, and the $*$ is meant to mean that we use all the previous features with their binned variants. These features make our parser a combination dependency/monolexicalized parser.

This is not quite a complete description of our features, as these templates would produce quite a large number of features. Instead, we produce these features only for rules that occur in the treebank. For other "bad" features, we randomly hash their respective features to a number of buckets. We found that as long as there were a sufficient number of buckets (a few thousand), performance was not appreciably different. We used one fifth the number of features found in the treebank as the number of bad feature buckets.

1.2 Unlexicalized model

Our unlexicalized parser just implements the model of Klein and Manning (2003) and uses straightforward rule indicator features for annotated rules. We experimented with slightly different annotations and adding more featurization, but this seemed to work about as well as the other slight variants we tried.

1.3 Latent variable model

Our latent variable model is similarly straightforward, using just indicators on annotated rules. We tried using other features that ignored some or all of the latent annotation, but they seemed to hurt. To break symmetries, we added random noise in the range $[0, 0.001]$ to all parameters.

1.4 Lexicon and unknown word features

Our lexicon is consistent across all models. It is similar to that used by Petrov and Klein (2008), relying primarily on word shape features, but we break features into smaller "pieces" than their parser.

The function ws(w) returns a list of word shape features, including:

1. w if w occurs more than 3 times. And if the word occurs fewer than 5 times the following:
2. HasX if w has X, where X is a capital letter, many capital letters, digits, non-digits, lower case letters, a dash, initial capitalization, no lower case letters, a non-letter, or no letters at all.
3. A concatentation of many of the previous features.
4. A feature that collapses repeated character classes into a word shape feature. For example, "300-odd" becomes "dd+-xx+".
5. LongWord if length(w) < 10.
6. ShortWord if length(w) < 5.
7. Suffixes and prefixes of lengths 1, 2, and 3.

For the lexicalized parser, we only use 1, 3, and 4 as appropriate for rule features. The lexicon had access to all features.

2 Unaries

The correct handling of unaries is a frequent challenge in the design of a parser. We adopt a simple two-layer transformation to trees that makes handling them much simpler Specifically, every tree consists of alternating layers of binary and unary productions. The children of every binary production rewrite as unaries (usually this is an identity rewrite), and the child of every unary rewrites as either a binary rule or a part-of-speech tag. When parsing, for each unary production $A \rightarrow B$ the most likely unary path for that production is replaced. We do this based on unannotated trees.

This approach makes sense for the anchored rule architecture we use for EP, since we do not need to worry about infinite unary chains. Finally, this choice also makes the MaxRecall algorithm (Goodman, 1996) simpler to use in practice, since the decision of how many unary rules to include is avoided.

3 Projecting to an Anchored Grammar

In this section, we discuss how to project the parse forest for an annotated grammar to an unannotated anchored grammar. This section largely derives from Matsuzaki et al. (2005)'s grammar used for decoding, but we include it here for completeness. Given a set of inside and outside scores over annotated labels $I(A[x], s, t)$ and $O(A[x], s, t)$ we can compute anchored PCFG probabilities as in Figure 1. Basically, we just marginalize over annotations, and normalize so that $\sum_{B,C,u} p({}_sA_t \rightarrow {}_sB_{uu}C_t) = 1$.

References

Joshua Goodman. 1996. Parsing algorithms and metrics. In *ACL*, pages 177–183.

Dan Klein and Christopher D. Manning. 2003. Accurate unlexicalized parsing. In *ACL*, pages 423–430.

Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. 2005. Probabilistic CFG with latent annotations. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pages 75–82, Morristown, NJ, USA. Association for Computational Linguistics.

Slav Petrov and Dan Klein. 2008. Discriminative log-linear grammars with latent variables. In J.C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20 (NIPS)*, pages 1153–1160, Cambridge, MA. MIT Press.

$$p({}_sA_t \to {}_sB_{uu}C_t) = \frac{\sum_{xyz} O(A[x],s,t)\text{score}(A[x] \to B[y]\,C[z], \mathbf{w}, s, t, u) I(B[y],s,u) I(C[z],u,t)}{\sum_x O(A[x],s,t) I(A[x],s,t)}$$

$$p({}_sA_t \to {}_sB_t) = \frac{\sum_{xy} O(A[x],s,t)\text{score}_u(A \to B, \mathbf{w}, s, t) I(B[y],s,t)}{\sum_x O(A[x],s,t) I(A[x],s,t)}$$

$$p({}_sA_{s+1} \to w_s) = 1$$

$$\text{score}(A[x] \to B[y]\,C[z], \mathbf{w}, s, t, u) = \exp\left(\theta_m^T \varphi(A[x] \to B[y]\,C[z], \mathbf{w}) + q(A \to B\,C, s, u, t)\right)$$

$$\text{score}_u(A[x] \to B[y], \mathbf{w}, s, t, u) = \exp\left(\theta_m^T \varphi(A[x] \to B[y], \mathbf{w}) + q(A \to B, s, t)\right)$$

Figure 1: Estimating an anchored grammar for a sentence **w** from the inside and outside scores of an anchored grammar. Note that we use alternating layers of unaries and binaries.

		Models		
Lat. Bits	ϕ	Lexicalized	Unlexicalized	Lex+Unlex
0	———	87.3/86.5	86.3/85.4	90.2/89.5
1	81.6/80.6	89.7/89.0	88.1/87.2	90.2/89.5
2	84.6/83.9	89.8/89.2	88.6/87.8	90.1/89.6
3	87.6/86.1	90.0/89.4	88.7/88.0	90.2/89.7
4	88.5/87.6	89.8/89.2	88.8/88.1	90.0/89.3
5	89.2/88.4	89.9/89.2	88.4/87.7	90.1/89.4
6	89.7/88.9	90.1/89.5	88.5/87.7	90.0/89.3

Table 1: Extra development set results. The values reported here are Section 22 Len 40/All of Section 22. The numbers on the left hand side are the number of factored latent bit annotations used in conjunction with the model above. The ϕ column uses only latent information.