

# Chapter 2

## Latent Variable Grammars for Natural Language Parsing\*

### 2.1 Introduction

As described in Chap. 1, parsing is the process of analyzing the syntactic structure of natural language sentences and will be fundamental for building systems that can understand natural languages. Probabilistic context-free grammars (PCFGs) underlie most high-performance parsers in one way or another (Charniak 2000; Collins 1999; Charniak and Johnson 2005; Huang 2008). However, as demonstrated by Charniak (1996) and Klein and Manning (2003a), a PCFG which simply takes the empirical rules and probabilities off of a treebank does not perform well. This naive grammar is a poor one because its context-freeness assumptions are too strong in some places (e.g., it assumes that subject and object NPs share the same distribution) and too weak in others (e.g., it assumes that long rewrites are not decomposable into smaller steps). Therefore, a variety of techniques have been developed to both enrich and generalize the naive grammar, ranging from simple tree annotation and category splitting (Johnson 1998; Klein and Manning 2003a) to full lexicalization and intricate smoothing (Collins 1999; Charniak 2000).

In this chapter, we investigate the learning of a grammar consistent with a treebank at the level of evaluation categories (such as NP, VP, etc.) but refined based on the likelihood of the training trees. Klein and Manning (2003a) addressed this question from a linguistic perspective, starting with a Markov grammar and manually refining categories in response to observed linguistic trends in the data. For example, the category NP might be split into the subcategory NP<sup>S</sup> in subject position and the subcategory NP<sup>VP</sup> in object position. Matsuzaki et al. (2005) and also Prescher (2005) later exhibited an automatic approach in which each category is split into a fixed number of subcategories. For example, NP would be split into NP-1 through NP-8. Their exciting result was that, while grammars quickly grew

---

\*The material in this chapter was originally presented in Petrov et al. (2006) and Petrov and Klein (2007).

too large to be managed, a 16-subcategory induced grammar reached the parsing performance of Klein and Manning (2003a)’s manual grammar. Other work has also investigated aspects of automatic grammar refinement; for example, Chiang and Bikel (2002) learn annotations such as head rules in a constrained declarative language for tree-adjointing grammars.

We present a method that combines the strengths of both manual and automatic approaches while addressing some of their common shortcomings. Like Matsuzaki et al. (2005) and Prescher (2005), we induce refinements in a fully automatic fashion. However, we use a more sophisticated split-merge approach that allocates subcategories adaptively where they are most effective, like a linguist would. The grammars recover patterns like those discussed in Klein and Manning (2003a), heavily articulating complex and frequent categories like NP and VP while barely splitting rare or simple ones (see Sect. 2.6 for an empirical analysis).

Empirically, hierarchical splitting increases the accuracy and lowers the variance of the learned grammars. Another contribution is that, unlike previous work, we investigate smoothed models, allowing us to refine grammars more heavily before running into the oversplitting effect discussed in Klein and Manning (2003a), where data fragmentation outweighs increased expressivity. Our method is capable of learning grammars of substantially smaller size and higher accuracy than previous grammar refinement work, starting from a simpler initial grammar. Because our latent variable approach is fairly language independent we are able to learn grammars directly for any language that has a treebank. We exhibit the best parsing numbers that we are aware of on several metrics, for several domains and languages, without any language dependent modifications. The performance can be further increased by combining our parser with non-local methods such as feature-based discriminative reranking (Charniak and Johnson 2005; Huang 2008).

Unfortunately, grammars that are sufficiently complex to handle the grammatical structure of natural language are often challenging to work with in practice because of their size. To address this problem, we introduce an approximate coarse-to-fine inference procedure that greatly enhances the efficiency of our parser, without loss in accuracy. Our method considers the refinement history of the final grammar, projecting it onto its increasingly refined prior stages. For any projection of a grammar, we give a new method for efficiently estimating the projection’s parameters from the source PCFG itself (rather than a treebank), using techniques for infinite tree distributions Corazza and Satta (2006) and iterated fixpoint equations. We then use a multipass approach where we parse with each refinement in sequence, much along the lines of Charniak et al. (2006), except with much more complex and automatically derived intermediate grammars. Thresholds are automatically tuned on held-out data, and the final system parses up to 100 times faster than the baseline PCFG parser, with no loss in test set accuracy.

We also consider the well-known issue of inference objectives in refined PCFGs. As in many model families (Steedman 2000; Vijay-Shankar and Joshi 1985), refined PCFGs have a derivation/parse distinction. The refined PCFG directly describes a generative model over derivations, but evaluation is sensitive only to the coarser treebank categories. While the most probable parse problem is NP-complete

Sima'an (2002), several approximate methods exist, including n-best reranking by parse likelihood, the labeled bracket algorithm of Goodman (1996), and a variational approximation introduced in Matsuzaki et al. (2005). We present experiments which explicitly minimize various evaluation risks over a candidate set using samples from the refined PCFG, and relate those conditions to the existing non-sampling algorithms. We demonstrate that minimum risk objective functions that can be computed in closed form are superior for maximizing  $F_1$ , yielding significantly higher results.

### 2.1.1 *Experimental Setup*

In this and the following chapter we will consider a supervised training regime, where we are given a set of sentences annotated with constituent information in form of syntactic parse trees, and want to learn a model that can produce such parse trees for new, previously unseen sentences. Such training sets are referred to as treebanks and consist of several 10,000 sentences. They exist for a number languages because of their large utility, and despite being labor intensive to create due to the necessary expert knowledge. In the following, we will often refer to the Wall Street Journal (WSJ) portion of the Penn Treebank, however, our latent variable approach is language independent and we will present an extensive set of additional experiments on a diverse set of languages ranging from German over Bulgarian to Chinese in Sect. 2.5.1.

As it is standard, we give results in form of labeled recall (LR), labeled precision (LP) and exact match (EX). Labeled recall is computed as the quotient of the number of correct nonterminal constituents in the guessed tree and the number of nonterminal constituents in the correct tree. Labeled precision is the number of correct nonterminal constituents in the guessed parse tree divided by the total number of nonterminal constituents in the guessed tree. These two metrics are necessary because the guessed parse tree and the correct parse tree do not need to have the same number of nonterminal constituents because of unary rewrites. Often times we will combine those two figures of merit by computing their harmonic mean ( $F_1$ ). Exact match finally measure the percentage of complete correct guessed trees.

It should be noted that these figures of merit are computed on the nonterminals excluding the preterminal (part of speech) level. This is standard practice and serves two purposes. Firstly, early parsers often required a separate part of speech tagger to process the input sentence and would focus only on predicting pure constituency structure. Secondly, including the easy to predict part of speech level would artificially boost the final parsing accuracies, obfuscating some of the challenges.

Finally a note on the significance of the results that are to follow. Some of the differences in parsing accuracy that will be reported might appear negligible, as one might be tempted to attribute them to statistical noise. However, because of the large number of test sentences (and therefore even larger number of evaluation constituents), many authors have shown with paired t-tests that differences as small

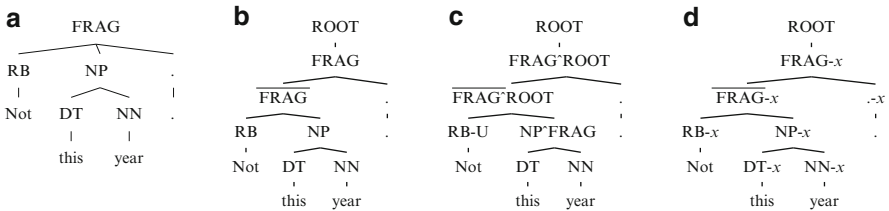
as 0.1% are statistically significant. Of course, to move science forward we will need larger improvements than 0.1%. One of the contributions of this work will therefore indeed be very significantly improved parsing accuracies for a number of languages, but what will be even more noteworthy, is that the same simple model will be able to achieve state-of-the-art performance on all tested languages.

## 2.2 Manual Grammar Refinement

The traditional starting point for unlexicalized parsing is the raw  $n$ -ary treebank grammar read from training trees (after removing functional tags and null elements). In order to obtain a cubic time parsing algorithm (Lari and Young 1990), we first binarize the trees as shown in Fig. 2.1. For each local tree rooted at an evaluation category  $A$ , we introduce a cascade of new nodes labeled  $\bar{A}$  so that each has two children. We use a right branching binarization, as we found the differences between binarization schemes to be small.

This basic grammar is imperfect in two well-known ways. First, many rule types have been seen only once (and therefore have their probabilities overestimated), and many rules which occur in test sentences will never have been seen in training (and therefore have their probabilities underestimated – see Collins (1999) for an analysis).<sup>1</sup> One successful method of combating this type of sparsity is to *markovize* the right-hand sides of the productions (Collins 1999). Rather than remembering the entire horizontal history when binarizing an  $n$ -ary production, horizontal markovization tracks only the previous  $h$  ancestors.

The second, and more major, deficiency is that the observed categories are too coarse to adequately render the expansions independent of the contexts. For example, subject noun phrase (NP) expansions are very different from object NP expansions: a subject NP is 8.7 times more likely than an object NP to expand as just a pronoun. Having separate symbols for subject and object NPs allows this variation



**Fig. 2.1** The original parse tree (a) gets binarized (b), and then either manually annotated (c) or refined with latent variables (d)

<sup>1</sup>Note that in parsing with the unsplit grammar, not having seen a rule doesn't mean one gets a parse failure, but rather a possibly very weird parse (Charniak 1996).

to be captured and used to improve parse scoring. One way of capturing this kind of external context is to use parent annotation, as presented in Johnson (1998). For example, NPs with S parents (like subjects) will be marked NP<sup>S</sup>, while NPs with VP parents (like objects) will be NP<sup>VP</sup>. Parent annotation is also useful for the pre-terminal (part-of-speech) categories, even if most tags have a canonical category. For example, NNS tags occur under NP nodes (only 234 of 70,855 do not, mostly mistakes). However, when a tag somewhat regularly occurs in a non-canonical position, its distribution is usually distinct. For example, the most common adverbs directly under ADVP are *also* (1599) and *now* (544). Under VP, they are *n't* (3779) and *not* (922). Under NP, *only* (215) and *just* (132), and so on.

### 2.2.1 Vertical and Horizontal Markovization

Both parent annotation (adding context) and RHS markovization (removing it) can be seen as two instances of the same idea. In parsing, every node has a vertical history, including the node itself, parent, grandparent, and so on. A reasonable assumption is that only the past  $v$  vertical ancestors matter to the current expansion. Similarly, only the previous  $h$  horizontal ancestors matter. It is a historical accident that the default notion of a treebank PCFG grammar takes  $v = 1$  (only the current node matters vertically) and  $h = \infty$  (rule right hand sides do not decompose at all). In this view, it is unsurprising that increasing  $v$  and decreasing  $h$  have historically helped.

Table 2.1 presents a grid of horizontal and vertical markovizations of the grammar. The raw treebank grammar corresponds to  $v = 1$ ,  $h = \infty$  (the upper right corner), while the parent annotation in Johnson (1998) corresponds to  $v = 2$ ,  $h = \infty$ , and the second-order model in Collins (1999), is broadly a smoothed version of  $v = 2$ ,  $h = 2$ . Table 2.1 also shows number of grammar categories resulting from each markovization scheme. These counts include all the intermediate categories which represent partially completed constituents. The general trend is that, in the absence of further annotation, more vertical annotation is better – even exhaustive grandparent annotation. This is not true for horizontal markovization, where the second-order model was superior. The best entry,  $v = 3$ ,  $h = 2$ , has an  $F_1$  of 81.0, already a substantial improvement over the baseline.

**Table 2.1** Horizontal and vertical Markovization:  $F_1$  parsing accuracies and grammar sizes (number of nonterminals)

Vertical order		Horizontal Markov order			
		$h = 0$	$h = 1$	$h = 2$	$h = \infty$
$v = 1$	No annotation	63.6 (98)	72.4 (575)	73.3 (2243)	73.4 (6899)
$v = 2$	Parents	72.6 (992)	79.4 (2487)	80.6 (5611)	79.5 (11259)
$v = 3$	Grandparents	75.0 (4001)	80.8 (7137)	81.0 (12406)	79.9 (19139)

### 2.2.2 Additional Linguistic Refinements

In this section, we will discuss some of linguistically motivated annotations presented in Klein and Manning (2003a). These annotations increasingly refine the grammar categories, but since we expressly do not smooth the grammar, not all splits are guaranteed to be beneficial, and not all sets of useful splits are guaranteed to co-exist well. In particular, while  $v = 3$ ,  $h = 2$  markovization is good on its own, it has a large number of categories and does not tolerate further splitting well. Therefore, we base all further exploration in this section on the  $v = 2$ ,  $h = 2$  grammar. Although it does not necessarily jump out of the grid at first glance, this point represents the best compromise between a compact grammar and useful markov histories.

In the raw grammar, there are many unaries, and once any major category is constructed over a span, most others become constructible as well using unary chains. Such chains are rare in real treebank trees: unary rewrites only appear in very specific contexts, for example S complements of verbs where the S has an empty, controlled subject. It would therefore be natural to annotate the trees so as to confine unary productions to the contexts in which they are actually appropriate. This annotation was also particularly useful at the preterminal level. One distributionally salient tag conflation in the Penn treebank is the identification of demonstratives (that, those) and regular determiners (the, a). Splitting DT tags based on whether they were only children captured this distinction. The same unary annotation was also effective when applied to adverbs, distinguishing, for example, as well from also. Beyond these cases, unary tag marking was detrimental.

The Penn tag set also conflates various grammatical distinctions that are commonly made in traditional and generative grammar, and from which a parser could hope to get useful information. For example, subordinating conjunctions (*while*, *as*, *if*), complementizers (*that*, *for*), and prepositions (*of*, *in*, *from*) all get the tag IN. Many of these distinctions are captured by parent annotation (subordinating conjunctions occur under S and prepositions under PP), but some are not (both subordinating conjunctions and complementizers appear under SBAR). Also, there are exclusively noun-modifying prepositions (*of*), predominantly verb-modifying ones (*as*), and so on. The annotation SPLIT-IN does a We therefore perform a linguistically motivated 6-way split of the IN tag.

The notion that the head word of a constituent can affect its behavior is a useful one. However, often the head tag is as good (or better) an indicator of how a constituent will behave. We found several head annotations to be particularly effective. Most importantly, the VP category is very overloaded in the Penn treebank, most severely in that there is no distinction between finite and infinitival VPs. To allow the finite/non-finite distinction, and other verb type distinctions, we annotated all VP nodes with their head tag, merging all finite forms to a single tag VBF. In particular, this also accomplished Charniak’s gerund-VP marking (Charniak 1997).

These three annotations are examples of the types of information that can be encoded in the node labels in order to improve parsing accuracy. Overall, Klein and Manning (2003a) were able to improve test set  $F_1$  is 86.3%, which is already higher than early lexicalized models, though of course lower than state-of-the-art lexicalized parsers.

## 2.3 Generative Latent Variable Grammars

Alternatively, rather than devising linguistically motivated features or splits, we can use latent variables to automatically learn a more highly articulated model than the naive CFG embodied by the training treebank. In all of our learning experiments we start from a minimal X-bar style grammar, which has vertical order  $v = 0$  and horizontal order  $h = 1$ . Since we will evaluate our grammar on its ability to recover the treebank’s nonterminals, we must include them in our grammar. Therefore, this initialization is the absolute minimum starting grammar that includes the evaluation nonterminals (and maintains separate grammar categories for each of them).<sup>2</sup> It is a very compact grammar: 98 nonterminals (45 part of speech tags, 27 phrasal categories and the 26 intermediate categories which were added during binarization), 236 unary rules, and 3,840 binary rules. This grammar turned out to be a very good starting point for our approach despite its simplicity, because adding latent variable refinements on top of a richer grammar quickly leads to an overfragmentation of the grammar.

Latent variable grammars then augment the treebank trees with latent variables at each node, splitting each treebank category into unconstrained subcategories. For each observed category  $A$  we now have a set of latent subcategories  $A_x$ . For example, NP might be split into  $NP_1$  through  $NP_8$ . This creates a set of (exponentially many) *derivations* over split categories for each of the original *parse trees* over unsplit categories, see Fig. 2.1.

The parameters of the refined productions  $A_x \rightarrow B_y C_z$ , where  $A_x$  is a subcategory of  $A$ ,  $B_y$  of  $B$ , and  $C_z$  of  $C$ , can then be estimated in various ways; past work on grammars with latent variables has investigated various estimation techniques. Generative approaches have included basic training with expectation maximization (EM) (Matsuzaki et al. 2005; Prescher 2005), as well as a Bayesian nonparametric approach (Liang et al. 2007). Discriminative approaches (Henderson 2004) and Chap. 3 are also possible, but we focus here on a generative, EM-based split and merge approach, as the comparison is only between estimation methods, since Smith and Johnson (2007) show that the model classes are the same.

---

<sup>2</sup>If our purpose was only to model language, as measured for instance by perplexity on new text, it could make sense to erase even the labels of the treebank to let EM find better labels by itself, giving an experiment similar to that of Pereira and Schabes (1992).

To obtain a grammar from the training trees, we want to learn a set of rule probabilities  $\beta$  over the latent subcategories that maximize the likelihood of the training trees, despite the fact that the original trees lack the latent subcategories. The Expectation-Maximization (EM) algorithm allows us to do exactly that. Given a sentence  $w$  and its parse tree  $T$ , consider a nonterminal  $A$  spanning  $(r, t)$  and its children  $B$  and  $C$  spanning  $(r, s)$  and  $(s, t)$ . Let  $A_x$  be a subcategory of  $A$ ,  $B_y$  of  $B$ , and  $C_z$  of  $C$ . Then the inside and outside probabilities  $P_{\text{IN}}(r, t, A_x) \stackrel{\text{def}}{=} P(w_{r:t} | A_x)$  and  $P_{\text{OUT}}(r, t, A_x) \stackrel{\text{def}}{=} P(w_{1:r} A_x w_{t:n})$  can be computed recursively:

$$P_{\text{IN}}(r, t, A_x) = \sum_{y,z} \beta(A_x \rightarrow B_y C_z) P_{\text{IN}}(r, s, B_y) P_{\text{IN}}(s, t, C_z) \quad (2.1)$$

$$P_{\text{OUT}}(r, s, B_y) = \sum_{x,z} \beta(A_x \rightarrow B_y C_z) P_{\text{OUT}}(r, t, A_x) P_{\text{IN}}(s, t, C_z) \quad (2.2)$$

$$P_{\text{OUT}}(s, t, C_z) = \sum_{x,y} \beta(A_x \rightarrow B_y C_z) P_{\text{OUT}}(r, t, A_x) P_{\text{IN}}(r, s, B_y) \quad (2.3)$$

Although we show only the binary component here, of course there are both binary and unary productions that are included. In the Expectation step, one computes the posterior probability of each refined rule and position in each training set tree  $T$ :

$$P(r, s, t, A_x \rightarrow B_y C_z | w, T) \propto P_{\text{OUT}}(r, t, A_x) \beta(A_x \rightarrow B_y C_z) P_{\text{IN}}(r, s, B_y) P_{\text{IN}}(s, t, C_z) \quad (2.4)$$

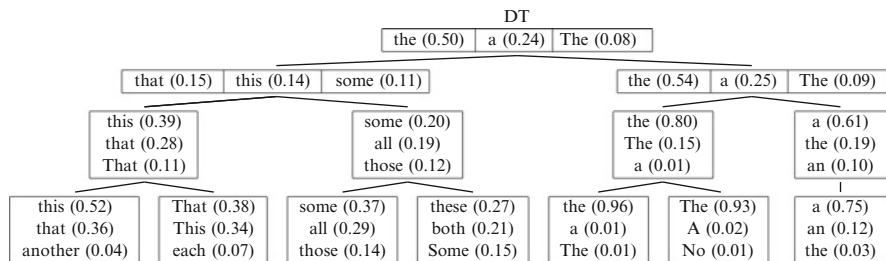
In the Maximization step, one uses the above probabilities as weighted observations to update the rule probabilities:

$$\beta(A_x \rightarrow B_y C_z) := \frac{\#\{A_x \rightarrow B_y C_z\}}{\sum_{y',z'} \#\{A_x \rightarrow B_{y'} C_{z'}\}} \quad (2.5)$$

Note that, because there is no uncertainty about the location of the brackets, this formulation of the inside-outside algorithm is linear in the length of the sentence rather than cubic (Pereira and Schabes 1992).

### 2.3.1 Hierarchical Estimation

In principle, we could now directly estimate grammars with a large number of latent subcategories, as done in Matsuzaki et al. (2005). However, EM is only guaranteed to find a local maximum of the likelihood, and, indeed, in practice it often gets stuck in a suboptimal configuration. If the search space is very large, even restarting may not be sufficient to alleviate this problem. One workaround is to manually specify some of the subcategories. For instance, Matsuzaki et al. (2005) start by refining



**Fig. 2.2** Evolution of the DT tag during hierarchical splitting and merging. Shown are the top three words for each subcategory and their respective probability

their grammar with the identity of the parent and sibling, which are observed (i.e., not latent), before adding latent variables.<sup>3</sup> If these manual refinements are good, they reduce the search space for EM by constraining it to a smaller region. On the other hand, this pre-splitting defeats some of the purpose of automatically learning latent subcategories, leaving to the user the task of guessing what a good starting grammar might be, and potentially introducing overly fragmented subcategories.

Instead, we take a fully automated, hierarchical approach where we repeatedly split and re-train the grammar. In each iteration we initialize EM with the results of the smaller grammar, splitting every previous subcategory in two and adding a small amount of randomness (1%) to break the symmetry. The results are shown in Fig. 2.3. Hierarchical splitting leads to better parameter estimates over directly estimating a grammar with  $2^k$  subcategories per observed category. While the two procedures are identical for only two subcategories ( $F_1$ : 76.1%), the hierarchical training performs better for four subcategories (83.7% vs 83.2%). This advantage grows as the number of subcategories increases (88.4% vs 87.3% for 16 subcategories). This trend is to be expected, as the possible interactions between the subcategories grows as their number grows. As an example of how staged training proceeds, Fig. 2.2 shows the evolution of the subcategories of the determiner (DT) tag, which first splits demonstratives from determiners, then splits quantificational elements from demonstratives along one branch and definites from indefinites along the other.

Because EM is a local search method, it is likely to converge to different local maxima for different runs. In our case, the variance is higher for models with few subcategories; because not all dependencies can be expressed with the limited number of subcategories, the results vary depending on which one EM selects first. As the grammar size increases, the important dependencies can be modeled, so the variance decreases.

<sup>3</sup>In other words, in the terminology of Klein and Manning (2003a), they begin with a (vertical order = 2, horizontal order = 1) baseline grammar.

### 2.3.2 *Adaptive Refinement*

It is clear from all previous work that creating more (latent) refinements can increase accuracy. On the other hand, oversplitting the grammar can be a serious problem, as detailed in Klein and Manning (2003a). Adding subcategories divides grammar statistics into many bins, resulting in a tighter fit to the training data. At the same time, each bin gives a less robust estimate of the grammar probabilities, leading to overfitting. Therefore, it would be to our advantage to split the latent subcategories only where needed, rather than splitting them all as in Matsuzaki et al. (2005). In addition, if all categories are split equally often, one quickly (four split cycles) reaches the limits of what is computationally feasible in terms of training time and memory usage.

Consider the comma POS tag. We would like to see only one sort of this tag because, despite its frequency, it always produces the terminal comma (barring a few annotation errors in the treebank). On the other hand, we would expect to find an advantage in distinguishing between various verbal categories and NP types. Additionally, splitting categories like the comma is not only unnecessary, but potentially harmful, since it needlessly fragments observations of other categories' behavior.

It should be noted that simple frequency statistics are not sufficient for determining how often to split each category. Consider the closed part-of-speech classes (e.g., DT, CC, IN) or the nonterminal ADJP. These categories are very common, and certainly do contain subcategories, but there is little to be gained from exhaustively splitting them before even beginning to model the rarer categories that describe the complex inner correlations inside verb phrases. Our solution is to use a split-merge approach broadly reminiscent of ISODATA, a classic clustering procedure (Ball and Hall 1967). Alternatively, instead of explicitly limiting the number of subcategories, we could also use an infinite model with a sparse prior that allocates subcategories indirectly and on the fly when the amount of training data increases. We formalize this idea in Sect. 2.3.4.

To prevent oversplitting, we could also measure the utility of splitting each latent subcategory individually and then split the best ones first, as suggested by Dreyer and Eisner (2006) and Headden et al. (2006). This could be accomplished by splitting a single category, training, and measuring the change in likelihood or held-out  $F_1$ . However, not only is this impractical, requiring an entire training phase for each new split, but it assumes the contributions of multiple splits are independent. In fact, extra subcategories may need to be added to several nonterminals before they can cooperate to pass information along the parse tree. Therefore, we go in the opposite direction; that is, we split every category in two, train, and then measure for each subcategory the loss in likelihood incurred when removing it. If this loss is small, the new subcategory does not carry enough useful information and can be

removed. What is more, contrary to the gain in likelihood for splitting, the loss in likelihood for merging can be efficiently approximated.<sup>4</sup>

Let  $T$  be a training tree generating a sentence  $w$ . Consider a node  $n$  of  $T$  spanning  $(r, t)$  with the label  $A$ ; that is, the subtree rooted at  $n$  generates  $w_{r:t}$  and has the label  $A$ . In the latent model, its label  $A$  is split up into several latent subcategories,  $A_x$ . The likelihood of the data can be recovered from the inside and outside probabilities at  $n$ :

$$P(w, T) = \sum_x P_{\text{IN}}(r, t, A_x) P_{\text{OUT}}(r, t, A_x) \quad (2.6)$$

where  $x$  ranges over all subcategories of  $A$ . Consider merging, at  $n$  only, two subcategories  $A_1$  and  $A_2$ . Since  $A$  now combines the statistics of  $A_1$  and  $A_2$ , its production probabilities are the sum of those of  $A_1$  and  $A_2$ , weighted by their relative frequency  $p_1$  and  $p_2$  in the training data. Therefore the inside score of  $A$  is:

$$P_{\text{IN}}(r, t, A) = p_1 P_{\text{IN}}(r, t, A_1) + p_2 P_{\text{IN}}(r, t, A_2) \quad (2.7)$$

Since  $A$  can be produced as  $A_1$  or  $A_2$  by its parents, its outside score is:

$$P_{\text{OUT}}(r, t, A) = P_{\text{OUT}}(r, t, A_1) + P_{\text{OUT}}(r, t, A_2) \quad (2.8)$$

Replacing these quantities in (2.6) gives us the likelihood  $P^n(w, T)$  where these two subcategories and their corresponding rules have been merged, around only node  $n$ . The summation is now over the subcategory considered for merging and all the other original subcategories.

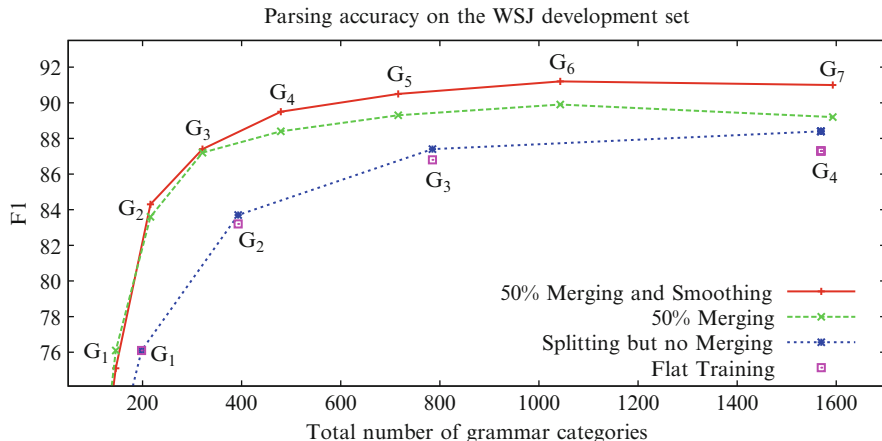
We approximate the overall loss in data likelihood due to merging  $A_1$  and  $A_2$  everywhere in all sentences  $w^i$  by the product of this loss for each local change:

$$\Delta_{\text{MERGE}}(A_1, A_2) = \prod_i \prod_{n \in T_i} \frac{P^n(w^i, T_i)}{P(w^i, T_i)} \quad (2.9)$$

This expression is an approximation because it neglects interactions between instances of a subcategory at multiple places in the same tree. These instances, however, are often far apart and are likely to interact only weakly, and this simplification avoids the prohibitive cost of running an inference algorithm for each tree and subcategory. Note that the particular choice of merging criterion is secondary, because we iterate between splitting and merging: if a particular split is (incorrectly) re-merged in a given round, we will be able to learn the same split

---

<sup>4</sup>The idea of merging complex hypotheses to encourage generalization is also examined in Stolcke and Omohundro (1994), who used a chunking approach to propose new productions in fully unsupervised grammar induction. They also found it necessary to make local choices to guide their likelihood search.



**Fig. 2.3** Hierarchical training leads to better parameter estimates. Merging reduces the grammar size significantly, while preserving the accuracy and enabling us to do more SM cycles. Parameter smoothing leads to even better accuracy for grammars with high complexity. The grammars range from extremely compact (an  $F_1$  of 78% with only 147 nonterminal categories) to extremely accurate (an  $F_1$  of 90.2% for our largest grammar with only 1,140 nonterminals)

in the next round again. Many alternative merging criteria could be used instead, and some might lead to slightly smaller grammars, however, in our experiments we found the final accuracies not to be affected. We refer to the operation of splitting subcategories and re-merging some of them based on likelihood loss as a split-merge (SM) cycle. SM cycles allow us to progressively increase the complexity of our grammar, giving priority to the most useful extensions.

In our experiments, merging was quite valuable. Depending on how many splits were reversed, we could reduce the grammar size at the cost of little or no loss of performance, or even a gain. We found that merging 50% of the newly split subcategories dramatically reduced the grammar size after each splitting round, so that after 6 SM cycles, the grammar was only 17% of the size it would otherwise have been (1,043 versus 6,273 subcategories), while at the same time there was no loss in accuracy (Fig. 2.3). Actually, the accuracy even increases, by 1.1% at 5 SM cycles. Furthermore, merging makes large amounts of splitting possible. It allows us to go from 4 splits, equivalent to the  $2^4 = 16$  subcategories of Matsuzaki et al. (2005), to 6 SM iterations, which takes a day to run on the Penn Treebank. The numbers of splits learned turned out to not be a direct function of category frequency; the numbers of subcategories for both lexical and nonlexical (phrasal) tags after 6 SM cycles are given in Figs. 2.9 and 2.10.

### 2.3.3 Smoothing

Splitting nonterminals leads to a better fit to the data by allowing each subcategory to specialize in representing only a fraction of the data. The smaller this fraction,

the higher the risk of overfitting. Merging, by allowing only the most beneficial subcategories, helps mitigate this risk, but it is not the only way. We can further minimize overfitting by forcing the production probabilities from subcategories of the same nonterminal to be similar. For example, a noun phrase in subject position certainly has a distinct distribution, but it may benefit from being smoothed with counts from all other noun phrases. Smoothing the productions of each subcategory by shrinking them towards their common base category gives us a more reliable estimate, allowing them to share statistical strength.

We perform smoothing in a linear way (Lindstone 1920). The estimated probability of a production  $p_x = P(A_x \rightarrow B_y C_z)$  is interpolated with the average over all subcategories of  $A$ .

$$p'_x = (1 - \alpha)p_x + \alpha\bar{p}, \quad \text{where} \quad \bar{p} = \frac{1}{n} \sum_x p_x \quad (2.10)$$

Here,  $\alpha$  is a small constant: we found 0.01 to be a good value, but the actual quantity was surprisingly unimportant. Because smoothing is most necessary when production statistics are least reliable, we expect smoothing to help more with larger numbers of subcategories. This is exactly what we observe in Fig. 2.3, where smoothing initially hurts (subcategories are quite distinct and do not need their estimates pooled) but eventually helps (as subcategories have finer distinctions in behavior and smaller data support).

Figure 2.3 also shows that parsing accuracy increases monotonically with each additional split-merge round until the sixth cycle. When there is no parameter smoothing, the additional seventh refinement cycle leads to a small accuracy loss, indicating that some overfitting is starting to occur. Parameter smoothing alleviates this problem, but cannot further improve parsing accuracy, indicating that we have reached an appropriate level of refinement for the given amount of training data. We present additional experiments on the effects of varying amounts of training data and depth of refinement in Sect. 2.5.1.

We also experimented with a number of different smoothing techniques, but found little or no difference between them. Similar to the merging criterion, the exact choice of smoothing technique was secondary: it is important that there is smoothing, but not how the smoothing is done.

### 2.3.4 An Infinite Alternative

In the previous sections we saw that a very important question when learning a PCFG is how many grammar categories ought to be allocated to the learning algorithm based on the amount of available training data. So far, we used a split-merge approach in order to explicitly control the number of subcategories per observed grammar category, and to use parameter smoothing to additionally counteract

overfitting. The question of “how many clusters?” has been tackled in the Bayesian nonparametrics literature via Dirichlet process (DP) mixture models (Antoniak 1974). DP mixture models have since been extended to hierarchical Dirichlet processes (HDPs) and infinite hidden Markov models (HDP-HMMs) (Teh et al. 2006; Beal et al. 2002) and applied to many different types of clustering/induction problems in NLP (Johnson et al. 2006; Goldwater et al. 2006).

In Liang et al. (2007) we present the *hierarchical Dirichlet process PCFG* (HDP-PCFG), a nonparametric Bayesian model of syntactic tree structures based on Dirichlet processes. Specifically, an HDP-PCFG is defined to have an infinite number of symbols; the Dirichlet process (DP) prior penalizes the use of more symbols than are supported by the training data. Note that “nonparametric” does not mean “no parameters”; rather, it means that the effective number of parameters can grow adaptively as the amount of data increases, which is a desirable property of a learning algorithm.

As models increase in complexity, so does the uncertainty over parameter estimates. In this regime, point estimates are unreliable since they do not take into account the fact that there are different amounts of uncertainty in the various components of the parameters. The HDP-PCFG is a Bayesian model which naturally handles this uncertainty. We present an efficient variational inference algorithm for the HDP-PCFG based on a structured mean-field approximation of the true posterior over parameters. The algorithm is similar in form to EM and thus inherits its simplicity, modularity, and efficiency. Unlike EM, however, the algorithm is able to take the uncertainty of parameters into account and thus incorporate the DP prior.

On synthetic data, our HDP-PCFG can recover the correct grammar without having to specify its complexity in advance. We also show that our HDP-PCFG can be applied to full-scale parsing applications and demonstrate its effectiveness in learning latent variable grammars. For limited amounts of training data, the HDP-PCFG learns more compact grammars than our split-merge approach, demonstrating the strengths of the Bayesian approach. However, its final parsing accuracy falls short of our split-merge approach when the entire treebank is used, indicating that merging and smoothing are superior alternatives in that case (because of their simplicity and our better understanding of how to work with them). The interested reader is referred to Liang et al. (2007) for a more detailed exposition of the infinite HDP-PCFG.

## 2.4 Inference

In the previous section we introduced latent variable grammars, which provide a tight fit to an observed treebank by introducing a hierarchy of refined subcategories. While the refinements improve the statistical fit and increase the parsing accuracy, they also increase the grammar size and thereby make inference (the syntactic analysis of new sentences) computationally expensive and slow.

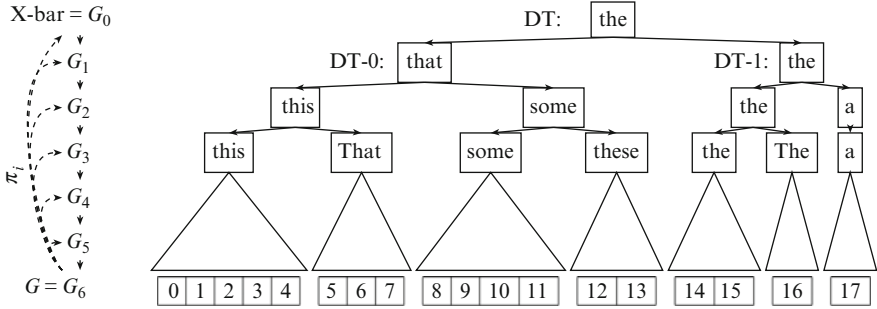
In general, grammars that are sufficiently complex to handle the grammatical structure of natural language will unfortunately be challenging to work with in practice because of their size. We therefore compute pruning grammars by projecting the (fine-grained) grammar of interest onto coarser approximations that are easier to deal with. In our multipass approach, we repeatedly pre-parse the sentence with increasingly more refined pruning grammars, ruling out large portions of the search space. At the final stage, we have several choices for how to extract the final parse tree. To this end, we investigate different objective functions and demonstrate that parsing accuracy can be increased by using a minimum risk objective that maximizes the expected number of correct grammar productions, and also by marginalizing out the hidden structure that is introduced during learning.

### 2.4.1 Hierarchical Coarse-to-Fine Pruning

At inference time, we want to use a given grammar to predict the syntactic structure of previously unseen sentences. Because large grammars are expensive to work with (in terms of memory requirements but especially in terms of computation), it is standard to prune the search space in some way. In the case of lexicalized grammars, the unpruned chart often will not even fit in memory for long sentences. Several proven techniques exist. Collins (1999) combines a punctuation rule which eliminates many spans entirely, and then uses span-synchronous beams to prune in a bottom-up fashion. Charniak et al. (1998) introduces best-first parsing, in which a figure-of-merit prioritizes agenda processing. Most relevant to our work are Goodman (1997) and Charniak and Johnson (2005) which use a *pre-parse* phase to rapidly parse with a very coarse, unlexicalized treebank grammar. Any item  $X:[i, j]$  with sufficiently low posterior probability in the pre-parse triggers the pruning of its lexical variants in a subsequent full parse.

Charniak et al. (2006) introduces *multi-level coarse-to-fine* parsing, which extends the basic pre-parsing idea by adding more rounds of pruning. In their work, the extra pruning was with grammars even coarser than the raw treebank grammar, such as a grammar in which all nonterminals are collapsed. We propose a novel multi-stage coarse-to-fine method which is particularly natural for our hierarchical latent variable grammars, but which is, in principle, applicable to any grammar. As in Charniak et al. (2006), we construct a sequence of increasingly refined grammars, reparsing with each refinement. The contributions of our method are that we derive sequences of refinements in a new way (Sect. 2.4.1.1), we consider refinements which are themselves complex, and, because our full grammar is not impossible to parse with, we automatically tune the pruning thresholds on held-out data.

It should be noted that other techniques for improving inference could also be applied here. In particular, A\* parsing techniques (Klein and Manning 2003b; Haghighi et al. 2007) appear very appealing because of their guaranteed optimality. However, Pauls and Klein (2009) clearly demonstrate that posterior pruning methods typically lead to greater speedups than their more cautious A\* analogues, while producing little to no loss in parsing accuracy.



**Fig. 2.4** Hierarchical refinement proceeds top-down while projection recovers coarser grammars. The top word for the first refinements of the determiner tag (DT) is shown where space permits

### 2.4.1.1 Projections

In our method, which we call *hierarchical coarse-to-fine* parsing, we consider a sequence of PCFGs  $G_0, G_1, \dots, G_n = G$ , where each  $G_i$  is a refinement of the preceding grammar  $G_{i-1}$  and  $G$  is the full grammar of interest. Each grammar  $G_i$  is related to  $G = G_n$  by a *projection*  $\pi_{n \rightarrow i}$  or  $\pi_i$  for brevity. A projection is a map from the nonterminal (including preterminal) category of  $G$  onto a reduced domain. A projection of grammar categories induces a projection of rules and therefore entire non-weighted grammars (see Fig. 2.4).

In our case, we also require the projections to be sequentially compatible, so that  $\pi_{i \rightarrow j} = \pi_{k \rightarrow j} \circ \pi_{i \rightarrow k}$ . That is, each projection is itself a coarsening of the previous projections. In particular, we take the projection  $\pi_{i \rightarrow j}$  to be the map that refined categories in round  $i$  to their earlier identities in round  $j$ .

It is straightforward to take a projection  $\pi$  and map a CFG  $G$  to its induced projection  $\pi(G)$ . What is less obvious is how the probabilities associated with the rules of  $G$  should be mapped. In the case where  $\pi(G)$  is more coarse than the treebank originally used to train  $G$ , and when that treebank is available, it is easy to project the treebank and directly estimate, say, the maximum-likelihood parameters for  $\pi(G)$ . This is the approach taken by Charniak et al. (2006), where they estimate what in our terms are projections of the raw treebank grammar from the treebank itself.

However, treebank estimation has several limitations. First, the treebank used to train  $G$  may not be available. Second, if the grammar  $G$  is heavily smoothed or otherwise regularized, its own distribution over trees may be far from that of the treebank. Third, we may wish to project grammars for which treebank estimation is problematic, for example, grammars which are more refined than the observed treebank grammars. Fourth, and most importantly, the meanings of the refined categories can and do drift between refinement stages, and we will be able to prune more without making search errors when the pruning grammars are as close as possible to the final grammar. Our method effectively avoids all of these problems by rebuilding and refitting the pruning grammars on the fly from the final grammar.

### 2.4.1.2 Estimating Projected Grammars

Fortunately, there is a well worked-out notion of estimating a grammar from an infinite distribution over trees (Corazza and Satta 2006). In particular, we can estimate parameters for a projected grammar  $\pi(G)$  from the tree distribution induced by  $G$  (which can itself be estimated in any manner). The earliest work that we are aware of on estimating models from models in this way is that of Nederhof (2005), who considers the case of learning language models from other language models. Corazza and Satta (2006) extend these methods to the case of PCFGs and tree distributions.

The generalization of maximum likelihood estimation is to find the estimates for  $\pi(G)$  with minimum KL divergence from the tree distribution induced by  $G$ . Since  $\pi(G)$  is a grammar over coarser categories, we fit  $\pi(G)$  to the distribution  $G$  induces over  $\pi$ -projected trees:  $P(\pi(T)|G)$ . Since the math is worked out in detail in Corazza and Satta (2006), including questions of when the resulting estimates are proper, we refer the reader to their excellent presentation for more details. The proofs of the general case are given in Corazza and Satta (2006), but the resulting procedure is quite intuitive.

Given a (fully observed) treebank, the maximum-likelihood estimate for the probability of a rule  $A \rightarrow BC$  would simply be the ratio of the count of  $A$  to the count of the configuration  $A \rightarrow BC$ . If we wish to find the estimate which has minimum divergence to an infinite distribution  $P(T)$ , we use the same formula, but the counts become expected counts:

$$P(A \rightarrow BC) = \frac{E_{P(T)}[A \rightarrow BC]}{E_{P(T)}[A]} \quad (2.11)$$

with unaries estimated similarly. In our specific case,  $A$ ,  $B$ , and  $C$  are categories in  $\pi(G)$ , and the expectations are taken over  $G$ 's distribution of  $\pi$ -projected trees,  $P(\pi(T)|G)$ . Corazza and Satta (2006) do not specify how one might obtain the necessary expectations, so we give two practical methods below.

### 2.4.1.3 Calculating Projected Expectations

Concretely, we can now estimate the minimum divergence parameters of  $\pi(G)$  for any projection  $\pi$  and PCFG  $G$  if we can calculate the expectations of the projected categories and productions according to  $P(\pi(T)|G)$ . The simplest option is to sample trees  $T$  from  $G$ , project the samples, and take average counts off of these samples. In the limit, the counts will converge to the desired expectations, provided the grammar is proper. However, we can exploit the structure of our projections to obtain the desired expectations much more simply and efficiently.

First, consider the problem of calculating the expected counts of a category  $A$  in a tree distribution given by a grammar  $G$ , ignoring the issue of projection. These expected counts obey the following one-step equations (assuming a unique *root* category):

$$c(\text{root}) = 1 \quad (2.12)$$

$$c(A) = \sum_{B \rightarrow \alpha A \beta} P(\alpha A \beta | B) c(B) \quad (2.13)$$

Here,  $\alpha$ ,  $\beta$ , or both can be empty, and a production  $A \rightarrow \gamma$  appears in the sum once for each  $A$  it contains. In principle, this linear system can be solved in any way.<sup>5</sup> In our experiments, we solve this system iteratively, with the following recurrences:

$$c_0(A) \leftarrow \begin{cases} 1 & \text{if } A = \text{root} \\ 0 & \text{otherwise} \end{cases} \quad (2.14)$$

$$c_{i+1}(A) \leftarrow \sum_{B \rightarrow \alpha A \beta} P(\alpha A \beta | B) c_i(B) \quad (2.15)$$

Note that, as in other iterative fixpoint methods, such as policy evaluation for Markov decision processes (Sutton and Barto 1998), the quantities  $c_k(A)$  have a useful interpretation as the expected counts ignoring nodes deeper than depth  $k$  (i.e., the roots are all the root category, so  $c_0(\text{root}) = 1$ ). This iteration may of course diverge if  $G$  is improper, but, in our experiments this method converged within around 25 iterations; this is unsurprising, since the treebank contains few nodes deeper than 25 and our base grammar  $G$  seems to have captured this property.

Once we have the expected counts of the categories in  $G$ , the expected counts of their projections  $A' = \pi(A)$  according to  $P(\pi(T) | G)$  are given by  $c(A') = \sum_{A: \pi(A)=A'} c(A)$ . Rules can be estimated directly using similar recurrences, or given by one-step equations:

$$c(A \rightarrow \gamma) = c(A) P(\gamma | A) \quad (2.16)$$

This process very rapidly computes the estimates for a projection of a grammar (i.e., in a few seconds for our largest grammars), and is done once during initialization of the parser.

#### 2.4.1.4 Hierarchical Projections

Recall that our final, refined grammars  $G$  come, by their construction process, with an ontogeny of grammars  $G_i$  where each grammar is a (partial) splitting of the preceding one. This gives us a natural chain of projections  $\pi_{i \rightarrow j}$  which projects backwards along this ontogeny of grammars (see Fig. 2.4). Of course, training also gives us parameters for the grammars, but only the chain of projections is needed.

---

<sup>5</sup>Whether or not the system has solutions depends on the parameters of the grammar. In particular,  $G$  may be improper, though the results of Chi (1999) imply that  $G$  will be proper if it is the maximum-likelihood estimate of a finite treebank.

Note that the projected estimates need not (and in general will not) recover the original parameters exactly, nor would we want them to. Instead they take into account any smoothing, subcategory drift, and so on which occurred by the final grammar. In Sect. 2.4.1.5, we show that the reconstructed projections are better than the original intermediate grammars, both at parsing and at pruning.

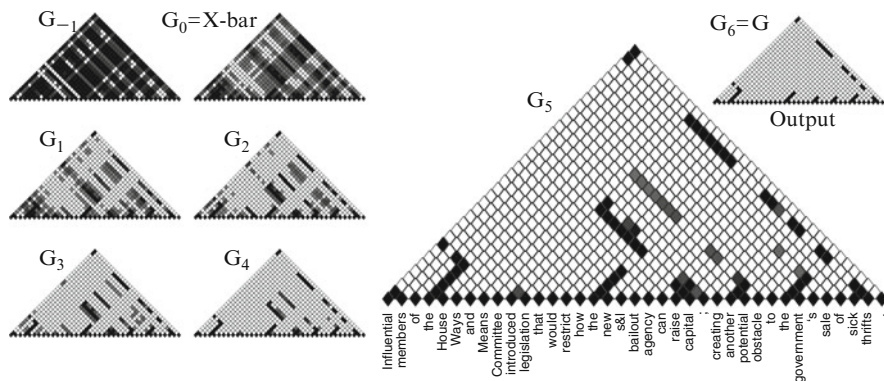
Starting from the base grammar, we run the projection process for each stage in the sequence, calculating  $\pi_i$  (chained incremental projections would also be possible). For the remainder of the paper, except where noted otherwise, all coarser grammars' estimates are these reconstructions, rather than those originally learned.

### 2.4.1.5 Pruning Experiments

As demonstrated by Charniak et al. (2006) parsing times can be greatly reduced by pruning chart items that have low posterior probability under a simpler grammar. Charniak et al. (2006) pre-parse with a sequence of grammars which are coarser than (parent-annotated) treebank grammars. However, we also work with grammars which are already heavily split, up to half as split as the final grammar, because we found the computational cost for parsing with the simple X-bar grammar to be insignificant compared to the costs for parsing with more refined grammars.

For a final grammar  $G = G_n$ , we compute estimates for the  $n$  projections  $G_{n-1}, \dots, G_0 = \text{X-Bar}$ , where  $G_i = \pi_i(G)$  as described above. Additionally we project to a grammar  $G_{-1}$  in which all nonterminals, except for the preterminals, have been collapsed. During parsing, we start off by exhaustively computing the inside/outside scores with  $G_{-1}$ . At each stage, chart items with low posterior probability are removed from the chart, and we proceed to compute inside/outside scores with the next, more refined grammar, using the projections  $\pi_{i \rightarrow i-1}$  to map between grammar categories in  $G_i$  and  $G_{i-1}$ . In each pass, we skip chart items whose projection into the previous stage had a probability below a stage-specific threshold, until we reach  $G = G_n$  (after seven passes in our case). For  $G$ , we do not prune but instead return the minimum risk tree, as will be described in Sect. 2.4.2.

Figure 2.5 shows the (unlabeled) bracket posteriors after each pass and demonstrates that most constructions can be ruled out by the simpler grammars, greatly reducing the amount of computation for the following passes. The pruning thresholds were empirically determined on a held out set by computing the most likely tree under  $G$  directly (without pruning) and then setting the highest pruning threshold for each stage that would not prune the optimal tree. This setting also caused no search errors on the test set. We found our projected grammar estimates to be significantly better suited for pruning than the original grammar estimates which were learned during the hierarchical training. Table 2.2 shows the tremendous reduction in parsing time (all times are cumulative) and gives an overview over grammar sizes and parsing accuracies. In particular, in our Java implementation on a 3 GHz processor, it is possible to parse the 1,578 development set sentences (of length 40 or less) in less than 900 s with an  $F_1$  of 91.2% (no search errors), or, by pruning more, in 600 s at 91.1%. For comparison, the Feb. 2006 release of the



**Fig. 2.5** Bracket posterior probabilities (black = high) for the first sentence of our development set during coarse-to-fine pruning. Note that we compute the bracket posteriors at a much finer level but are showing the unlabeled posteriors for illustration purposes. No pruning is done at the finest level  $G_6 = G$  but the minimum risk tree is returned instead

**Table 2.2** Grammar sizes, parsing times and accuracies for latent variable grammars PCFGs with and without hierarchical coarse-to-fine parsing on our development set (1,578 sentences with 40 or less words from section 22 of the Penn Treebank)

	$G_0$	$G_1$	$G_2$	$G_3$	$G_4$	$G_5$	$G_6$
Nonterminals	98	147	219	329	498	742	1140
Productions	3,700	8,300	19,600	52,400	126,100	298,200	531,200
No pruning	52 min	79 min	99 min	187 min	288 min	864	1612 min
X-bar pruning	8 min	11 min	14 min	22 min	30 min	68 min	111 min
C-to-F (original)	8 min	10 min	13 min	17 min	22 min	29 min	35 min
C-to-F (projected)	6 min	8 min	10 min	11 min	12 min	13.5 min	15 min
$F_1$ for above	64.8	78.0	85.2	87.7	89.7	90.6	91.2
C-to-F (lossy)	6 min	7 min	8 min	8.5 min	9 min	9.5 min	10 min
$F_1$ for above	64.3	77.8	84.9	87.5	89.4	90.4	91.1

Charniak and Johnson (2005) parser runs in 1,150 s on the same machine with an  $F_1$  of 90.7%.

### 2.4.2 Objective Functions for Parsing

A refined PCFG is a grammar  $G$  over nonterminal categories of the form  $A-x$  where  $A$  is an evaluation category (such as  $NP$ ) and  $x$  is some indicator of a subcategory, such as a parent annotation.  $G$  induces a *derivation distribution*  $P(T|G)$  over trees  $T$  labeled with refined categories. This distribution in turn induces a *parse distribution*  $P(T'|G) = P(\pi(T)|G)$  over (projected) trees with unsplit evaluation categories, where  $P(T'|G) = \sum_{T: T'=\pi(T)} P(T|G)$ . We now have several choices of how

to select a tree given these posterior distributions over trees. In this section, we present experiments with the various options and explicitly relate them to parse risk minimization (Titov and Henderson 2006).

### 2.4.2.1 Minimum Bayes Risk Parsing

The decision-theoretic approach to parsing would be to select the parse tree which minimizes our expected loss according to our beliefs:

$$T_P^* = \operatorname{argmin}_{T_P} \sum_{T_T} P(T_T|w, G) L(T_P, T_T) \quad (2.17)$$

where  $T_T$  and  $T_P$  are “true” and predicted parse trees. Here, our loss is described by the function  $L$  whose first argument is the predicted parse tree and the second is the gold parse tree. Reasonable candidates for  $L$  include zero-one loss (exact match), precision, recall,  $F_1$  (specifically EVALB here), and so on. Of course, the naive version of this process is intractable: we have to loop over all (pairs of) possible parses. Additionally, it requires parse likelihoods  $P(T_P|w, G)$ , which are tractable, but not trivial, to compute for refined grammars. There are two options: limit the predictions to a small candidate set or choose methods for which dynamic programs exist.

For arbitrary loss functions, we can approximate the minimum-risk procedure by taking the min over only a set of *candidate parses*  $T_P$ . In some cases, each parse’s expected risk can be evaluated in closed form. Exact match (likelihood) has this property. In general, however, we can approximate the expectation with samples from  $P(T|w, G)$ . The method for sampling derivations of a PCFG is given in Finkel et al. (2006). It requires a single inside-outside computation per sentence and is then efficient per sample. Note that for refined grammars, a posterior parse sample can be drawn by sampling a derivation and projecting away the subcategories.

Figure 2.4 shows the results of the following experiment. We constructed 10-best lists from the full grammar  $G$  in Sect. 4.2. We then took the same grammar and extracted 500-sample lists using the method of Finkel et al. (2006). The minimum risk parse candidate was selected for various loss functions. As can be seen, in most cases, risk minimization reduces test-set loss of the relevant quantity. Exact match is problematic, however, because 500 samples is often too few to draw a match when a sentence has a very flat posterior, and so there are many all-way ties.<sup>6</sup> Since exact match permits a non-sampled calculation of the expected risk, we show this option as well, which is substantially superior. This experiment highlights that the correct procedure for exact match is to find the most probable parse.

---

<sup>6</sup>5,000 samples do not improve the numbers appreciably.

### 2.4.2.2 Alternative Objective Functions

An alternative approach to reranking candidate parses is to work with inference criteria which admit dynamic programming solutions. Table 2.3 shows three possible objective functions which use the easily obtained posterior marginals of the parse tree distribution. Interestingly, while they have fairly different decision theoretic motivations, their closed-form solutions are similar.

One option is to maximize likelihood in an approximate distribution. Matsuzaki et al. (2005) present a VARIATIONAL approach, which approximates the true posterior over parses by a cruder, but tractable sentence-specific one. In this approximate distribution there is no derivation/parse distinction and one can therefore optimize exact match by selecting the most likely derivation.

Instead of approximating the tree distribution we can use an objective function that decomposes along parse posteriors. The labeled brackets algorithm of Goodman (1996) has such an objective function. In its original formulation this algorithm maximizes the number of expected correct nodes, but instead we can use it to maximize the number of correct rules (the MAX-RULE-SUM algorithm). A worrying issue with this method is that it is ill-defined for grammars which allow infinite unary chains: there will be no finite minimum risk tree under recall loss (you can always reduce the risk by adding one more cycle). We implement MAX-RULE-SUM in a CNF-like grammar family where above each binary production is exactly one unary production (possibly a self-loop). With this constraint, unary chains are not a problem. As might be expected, this criterion improves bracket measures at the expense of exact match.

We found it optimal to use a third approach, in which rule posteriors are multiplied instead of added. This corresponds to choosing the tree with greatest chance of having all rules correct, under the (incorrect) assumption that the rules correctness are independent. This MAX-RULE-PRODUCT algorithm does not need special treatment of infinite unary chains because it is optimizing a product rather than a sum. While these three methods yield very similar results (see Fig. 2.4), the MAX-RULE-PRODUCT algorithm consistently outperformed the other two.

**Table 2.3** Different objectives for parsing with posteriors, yielding comparable results

VARIATIONAL:	$q(A \rightarrow BC, i, k, j) = \frac{r(A \rightarrow BC, i, k, j)}{\sum_x P_{OUT}(A_x, i, j) P_{IN}(A_x, i, j)}$	$T_G = \operatorname{argmax}_T \prod_{e \in T} q(e)$
MAX-RULE-SUM:	$q(A \rightarrow BC, i, k, j) = \frac{r(A \rightarrow BC, i, k, j)}{P_{IN}(root, 0, n)}$	$T_G = \operatorname{argmax}_T \sum_{e \in T} q(e)$
MAX-RULE-PRODUCT:	$q(A \rightarrow BC, i, k, j) = \frac{r(A \rightarrow BC, i, k, j)}{P_{IN}(root, 0, n)}$	$T_G = \operatorname{argmax}_T \prod_{e \in T} q(e)$

$A, B, C$  are nonterminal categories,  $x, y, z$  are latent subcategories, and  $i, j, k$  are between-word indices. Hence  $(A_x, i, j)$  denotes a constituent labeled with  $A_x$  spanning from  $i$  to  $j$ . Furthermore, we write  $e = (A \rightarrow BC, i, j, k)$  for brevity. See text for details

**Table 2.4** A 10-best list from our best  $G$  can be reordered as to maximize a given objective either using samples or, under some restricting assumptions, in closed form

Objective	LP	LR	F1	EX
Best derivation				
Viterbi derivation	89.6	89.4	89.5	37.4
Reranking				
Random	87.6	87.7	87.7	16.4
Precision (sampled)	<b>91.1</b>	88.1	89.6	21.4
Recall (sampled)	88.2	<b>91.3</b>	89.7	21.5
F1 (sampled)	90.2	89.3	<b>89.8</b>	<b>27.2</b>
Exact (sampled)	89.5	89.5	89.5	25.8
Exact (non-sampled)	90.8	90.8	90.8	<b>41.7</b>
Exact/F1 (oracle)	95.3	94.4	95.0	63.9
Dynamic Programming				
VARIATIONAL	90.7	90.9	90.8	<b>41.4</b>
MAX-RULE-SUM	90.5	<b>91.3</b>	90.9	40.4
MAX-RULE-PRODUCT	<b>91.2</b>	91.1	<b>91.2</b>	<b>41.4</b>

Overall, the closed-form options were superior to the reranking ones, except on exact match, where the gains from correctly calculating the risk outweigh the losses from the truncation of the candidate set (Table 2.4).

Note that there are two factors contributing to the improved accuracy of the MAX-RULE-PRODUCT algorithm over extracting the Viterbi derivation ( $\Delta F_1 = 1.7\%$ ): (1) the change in objective function and (2) the marginalization of the latent structure, which aggregates the probability mass that is spread out over a potentially large number of derivations that correspond to the same parse tree. To separate out those two effects, we used the MAX-RULE-PRODUCT objective function but computed the highest scoring derivation rather than parse. This gave an  $F_1$  score of 90.6, indicating that two thirds of the gains are coming from the alternative objective function (roughly 1.1%) and one third from marginalizing out the latent structure (roughly 0.6%).

## 2.5 Additional Experiments

### 2.5.1 Experimental Setup

We trained grammars for a variety of languages and ran experiments respecting the standard splits on the corpora described in Table 2.5. Starting from an X-bar grammar, we trained latent variable grammars for 6 split and merge rounds, as described in Sect. 4.2. To better deal with unknown and rare words, we extract a small number of features from the word and then compute approximate tagging probabilities. A word is classified into one of 50 unknown word categories based

**Table 2.5** Treebanks and standard setups used in our experiments

	Training set	Development set	Test set
ENGLISH-WSJ Marcus et al. (1993)	Sections 2–21	Section 22	Section 23
ARABIC Maamouri et al. (2007)	“Mona Diab” splits <sup>8</sup> (a.k.a. Johns Hopkins 2005 Workshop)		
BULGARIAN Simov et al. (2004)	11,549 sentences	Cross- validation	2,496 sentences
CHINESE Xue et al. (2002)	Articles 1–270, 400–1151	Articles 301–325	Articles 271–300
FRENCH <sup>9</sup> Abeillé et al. (2003)	Sentences 1–18,609	Sentences 18,610–19,609	Sentences 19,609–20,610
GERMAN Skut et al. (1997)	Sentences 1–18,602	Sentences 18,603–19,602	Sentences 19,603–20,602
ITALIAN Lesmo et al. (2002)	10-fold cross-validation (see also EVALITA <sup>10</sup> shared task)		

on the presence of features such as capital letters, digits, and dashes<sup>7</sup> and its tagging probability is given by:  $P'(\text{word}|\text{tag}) = k \hat{P}(\text{class}|\text{tag})$  where  $k$  is a constant representing  $P(\text{word}|\text{class})$  and can simply be dropped. Rare words are modeled using a combination of their known and unknown distributions.

At inference time, we used the MAX-RULE-PRODUCT algorithm from Sect. 2.4.2 to marginalize out the latent structure. The hierarchical coarse-to-fine pruning procedure described in Sect. 2.4.1 was applied in order to speed up parsing, as it was shown to produce very few search errors, while greatly accelerating inference. The EVALB parseval reference implementation, available from Sekine and Collins (1997), was used for scoring.

## 2.5.2 Baseline Grammar Variation

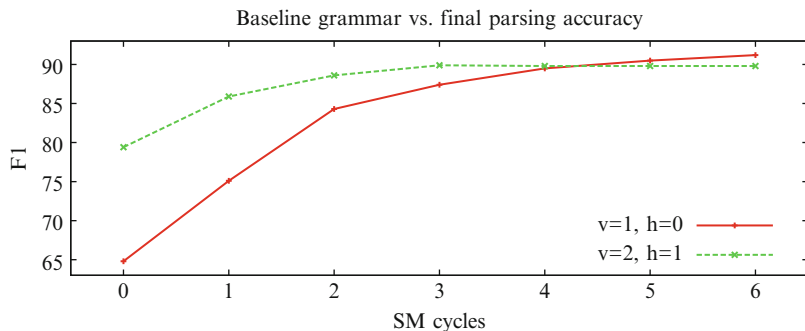
As in the case of manual grammar refinement, we can vary the level of markovization applied to the treebank before extracting the baseline grammar. Starting with a structurally annotated grammar (less markovization), gives a higher starting point, while starting with a minimal X-Bar grammar pre-imposes fewer restrictions and gives more flexibility to the learning algorithm. As Fig. 2.6 shows, the final

<sup>7</sup>For English we additionally use a list of suffixes.

<sup>8</sup>See <http://nlp.stanford.edu/software/parser-arabic-data-splits.shtml>

<sup>9</sup>Cross validation is used due to the small size of the treebank, see <http://evalita.fbk.eu/>

<sup>10</sup>This setup contains only sentences without annotation errors, as in Arun and Keller (2005).



**Fig. 2.6** Starting with a simple baseline grammar is advantageous because imposing too much initial structure causes overfragmentation in the long run

performance is more than 2% higher when starting with an X-Bar grammar (verticalmarkovization = 1, horizontalmarkovization = 0);

### 2.5.3 Final Results WSJ

By using a split-merge strategy and beginning with the barest possible initial structure, our method reliably learns PCFGs that are remarkably good at parsing. As one can see in Table 2.6, the resulting English parser ranks among the best lexicalized parsers, beating the one of Charniak and Johnson (2005), but falling short of discriminative systems that take the output of parsers as input features (Charniak and Johnson 2005; Huang 2008).

### 2.5.4 Multilingual Parsing

Most research on parsing has focused on English and parsing performance on other languages is generally significantly lower.<sup>11</sup> Recently, there have been some attempts to adapt parsers developed for English to other languages (Levy and Manning 2003; Cowan and Collins 2005). Adapting lexicalized parsers to other languages is not a trivial task as it requires at least the specification of head rules, and has had limited success. Adapting unlexicalized parsers appears to be equally difficult: Levy and Manning (2003) adapt the unlexicalized parser of Klein and

<sup>11</sup>Of course, cross-linguistic comparison of results is complicated by differences in corpus annotation schemes and sizes, and differences in linguistic characteristics.

**Table 2.6** Generative latent variable grammars achieve state-of-the-art parsing performance on a variety of languages

Parser	$\leq 40$ words			all		
	LP	LR	EX	LP	LR	EX
ENGLISH						
Charniak and Johnson (2005) <sup>12</sup>	90.3	90.1	39.6	89.7	89.6	37.2
Split-merge generative parse	<b>90.8</b>	<b>90.6</b>	<b>38.8</b>	<b>90.2</b>	<b>90.1</b>	<b>36.6</b>
ENGLISH (reranked)						
Charniak and Johnson (2005) <sup>13</sup>	92.4	91.6	<b>46.6</b>	91.8	91.0	<b>44.0</b>
Huang (2008)	<b>92.8</b>	<b>91.8</b>	46.2	<b>92.2</b>	<b>91.2</b>	43.5
ARABIC						
Bikel (2004)	76.0	75.4	–	73.4	72.5	–
Split-merge generative parse	<b>79.0</b>	<b>78.0</b>	<b>20.7</b>	<b>76.4</b>	<b>75.3</b>	<b>15.7</b>
BULGARIAN						
Chanev et al. (2007)	–			F <sub>1</sub> 80.4		
Split-merge generative parse	<b>82.4</b>	<b>81.4</b>	<b>12.8</b>	<b>82.1</b>	<b>81.1</b>	<b>12.5</b>
CHINESE <sup>14</sup>						
Bikel (2004)	82.9	79.6	–	80.6	77.5	–
Split-merge generative parse	<b>86.9</b>	<b>85.7</b>	<b>37.8</b>	<b>84.8</b>	<b>82.6</b>	<b>32.5</b>
FRENCH						
Arun and Keller (2005) <sup>15</sup>	78.2	80.1	21.2	74.6	76.6	16.4
Split-merge generative parse	<b>80.7</b>	<b>81.4</b>	<b>22.0</b>	<b>77.2</b>	<b>78.7</b>	<b>17.5</b>
GERMAN						
Dubey (2005)	F <sub>1</sub> 76.3			–		
Split-merge generative parse	<b>80.8</b>	<b>80.7</b>	<b>43.6</b>	<b>80.1</b>	<b>80.1</b>	<b>42.4</b>
ITALIAN						
Bikel (2004)	73.7	74.7	18.6	70.5	71.2	15.4
Split-merge generative parse	<b>79.0</b>	<b>79.3</b>	<b>27.4</b>	<b>75.6</b>	<b>75.7</b>	<b>22.8</b>

Manning (2003a) to Chinese, but even after significant efforts on manually choosing category splits, only modest performance gains are reported.

In contrast, automatically learned grammars like the ones presented here require only a treebank for training and no additional human input. One has therefore reason to believe that their performance will generalize better across languages than the performance of parsers that have been hand tailored to English. Table 2.6 shows that automatically inducing latent structure is a technique that generalizes well across language boundaries and results in state of the art performance for an array of very different languages. However, the final accuracies fall well short of the accuracy on English. Some experiments in Sect. 2.5.6 suggest that this discrepancy cannot be explained with the smaller size of the foreign treebanks, but is more likely due

<sup>12</sup>This is the performance of the lexicalized parser only.

<sup>13</sup>This is the performance of the reranking-parser from <http://www.cog.brown.edu/mj/software.htm>

<sup>14</sup>Sun and Jurafsky (2004) report better performance, however they assume gold POS tags (*p.c.*).

<sup>15</sup>Arun and Keller (2005) report results on a different test set. These figures are on the standard test set, A. Arun (*p.c.*).

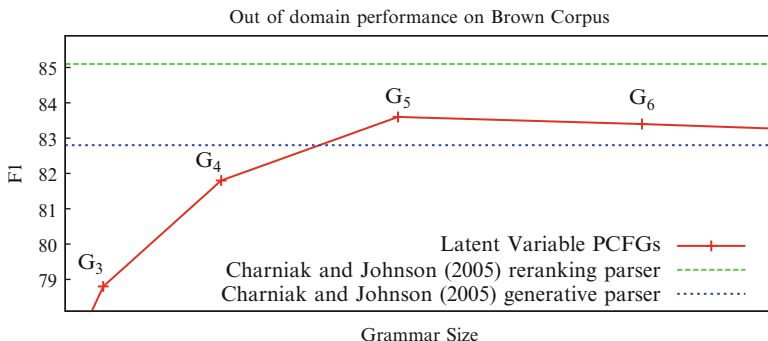
to language intrinsic characteristics, and annotation standards. We investigate the learned subcategories in Sect. 2.6.

Note that we explicitly did not attempt to adapt the parser to the new languages, to illustrate the general utility of latent variable grammars. Rather, we applied our model directly to each of the treebanks, using the same model hyperparameters (merging percentage and smoothing factor, pruning thresholds, unknown word features) as for English. This only underestimates the potential performance of our model. In fact, augmenting the unknown model with language specific suffix features can boost performance by several percentage points. Unpublished results by B. Crabbé (on French), and M. Harper (on Chinese), (*p.c.*). See also Petrov and Klein (2008c) for a set of experiments on parsing the different German treebanks, and in particular on recovering the grammatical function tags present in those treebanks in addition to pure syntactic structures.

### 2.5.5 Corpus Variation

Related to cross language generalization is the generalization across domains for the same language. It is well known that a model trained on the Wall Street Journal loses significantly in performance when evaluated on the Brown Corpus (see Gildea (2001) for more details and the exact setup of their experiment, which we duplicated here). Recently McClosky et al. (2006) came to the conclusion that this performance drop is not due to overfitting the WSJ data. Figure 2.7 shows the performance on the Brown corpus during hierarchical training. While the  $F_1$  score on the WSJ is rising we observe a drop in performance after the fifth iteration, suggesting that some overfitting might be occurring.

We observe similar trends on the Genia corpus (Tateisi et al. 2005), a corpus of abstracts from the biomedical domain, reaching our best performance of 78.9%



**Fig. 2.7** Parsing accuracy starts dropping after five training iterations on the Brown corpus, while it is improving on the WSJ, indicating overfitting

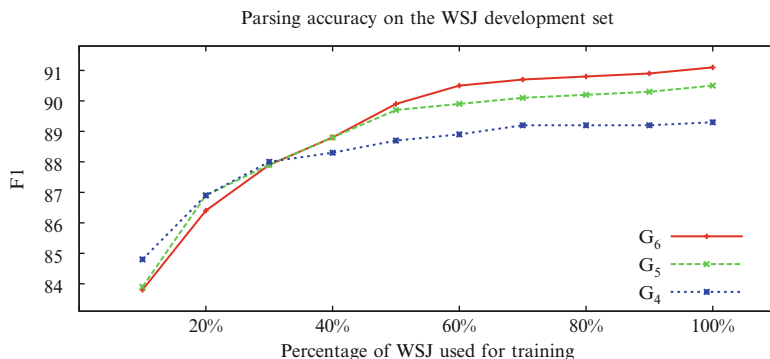
(on the full test set) after five split-merge rounds. See (Clegg and Shepherd 2007) for an extensive study comparing the parsing performance of different parsers on this domain. As in the case of language adaptation, parsing performance on out of domain data is easiest to increase by improving the part-of-speech (POS) tagging accuracy (McClosky and Charniak 2008). This is unsurprising, given the fact the POS tagging accuracy falls to below 83% on the Genia corpus (compared to above 96% on the WSJ). Future work will investigate methods of incorporating unlabeled data from the target domain for improving tagging and parsing accuracy.

### 2.5.6 Training Size Variation

It is also interesting to know how many sentences need to be manually parsed to create a training corpus that enables the learning of accurate grammars. Figure 2.8 shows how parsing accuracy varies in the presence of different amounts of training data. Surprisingly, even if we restrict ourselves to only the first 10% of the WSJ (roughly 4,000 sentences), we can achieve a parsing accuracy of almost 85%, rivaling the performance of early lexicalized parsers that were trained on the entire treebank. Parsing accuracy rises quite steeply when we add more training data, but levels off at about 70% of the training data (28,000 sentences). The last 12,000 sentences add only about 0.3% of accuracy.

It is also interesting to observe that in the presence of moderate amounts of training data (10,000 sentences), refining the grammars too heavily (5 or 6 rounds) leads to overfitting (despite smoothing and merging of the least useful splits). Only when there is sufficient empirical evidence, is it viable to refine the grammars heavily.

This experiment also partially addresses the question about the performance gap between parsing English and parsing other languages. While the WSJ is about twice



**Fig. 2.8** Parsing accuracy on the WSJ increases when more training data is used for learning the grammar. However, the last 30% of training data add only 0.3 in  $F_1$  score

as big as the treebanks for other languages, the sheer size of the treebank does not explain why parsing performance on English is so much higher. It is more likely that there are language specific explanations causing the difference in accuracy and simply labeling more data will not be sufficient to bring parsing accuracy into the 90% range for other languages. Differences in annotation standards, probably also contribute to the disparity.

## 2.6 Analysis

So far, we have presented a split-merge method for learning to iteratively refine basic categories like NP and VP into automatically induced subcategories (in the original sense of Chomsky (1965)). This approach gives parsing accuracies of up to 91.2% on the development set, substantially higher than previous category-refining approaches, while starting from an extremely simple base grammar. However, in general, any automatic induction system is in danger of being entirely uninterpretable. In this section, we examine the learned grammars, discussing what is learned. We focus particularly on connections with the linguistically motivated refinements of Klein and Manning (2003a), which we do generally recover.

Inspecting a large grammar by hand is difficult, but fortunately, our baseline grammar has less than 100 nonterminal categories, and even our most complicated grammar has only 1,043 total (sub)categories. It is therefore relatively straightforward to review the broad behavior of a grammar. In this section, we review a randomly-selected grammar for English after 4 SM cycles that produced an  $F_1$  score of 89.11 on the development set. We feel it is reasonable to present only a single grammar because all the grammars are very similar. For example, after 4 SM cycles, the  $F_1$  scores of the four trained grammars have a variance of only 0.024, which is tiny compared to the deviation of 0.43 obtained by Matsuzaki et al. (2005). Furthermore, these grammars allocate splits to nonterminals with a variance of only 0.32, so they agree to within a single latent subcategory. We also present some examples of the learned subcategories for other languages at the end of the section.

### 2.6.1 *Lexical Subcategories*

One of the original motivations for lexicalization of parsers is the fact that part-of-speech (POS) tags are usually far too general to encapsulate a word's syntactic behavior. In the limit, each word may well have its own unique syntactic behavior, especially when, as in modern parsers, semantic selectional preferences are lumped in with traditional syntactic trends. However, in practice, and given limited data, the relationship between specific words and their syntactic contexts may be best modeled at a level more fine than POS tag but less fine than lexical identity.

In our model, POS tags are refined just like any other grammar category: the subcategories for several tags are shown in Table 2.7, along with their three most frequent members. In most cases, the categories are recognizable as either classic subcategories or an interpretable division of some other kind.

Nominal categories are the most heavily refined (see Fig. 2.9), and have the splits which are most semantic in nature (though not without syntactic correlations). For example, plural common nouns (NNS) divide into the maximum number of categories (16). One category consists primarily of dates, whose typical parent is an NP subcategory whose typical parent is a root S, essentially modeling the temporal noun annotation discussed in Klein and Manning (2003a). Another category specializes in capitalized words, preferring as a parent an NP with an S parent (i.e. subject position). A third category specializes in monetary units, and so on. These kinds of syntactico-semantic categories are typical, and, given distributional clustering results like those of Schuetze (1998), unsurprising. The singular nouns are broadly similar, if slightly more homogenous, being dominated by categories for stocks and trading. The proper noun category (NNP, shown) also splits into the maximum 16 categories, including months, countries, variants of *Co.* and *Inc.*, first names, last names, initials, and so on.

Verbal categories are also heavily split. Verbal subcategories sometimes reflect syntactic selectional preferences, sometimes reflect semantic selectional preferences, and sometimes reflect other aspects of verbal syntax. For example, the present tense third person verb subcategories (VBZ) are shown. The auxiliaries get three clear categories: *do*, *have*, and *be* (this pattern repeats in other tenses), as well a fourth category for the ambiguous *'s*. Verbs of communication (*says*) and propositional attitudes (*believes*) that tend to take inflected sentential complements dominate two classes, while control verbs (*wants*) fill out another.

As an example of a less-refined category, the superlative adjectives (JJS) are split into three categories, corresponding principally to *most*, *least*, and *largest*, with most frequent parents NP, QP, and ADVP, respectively. The relative adjectives (JJR) are split in the same way. Relative adverbs (RBR) are split into a different three categories, corresponding to (usually metaphorical) distance (*further*), degree (*more*), and time (*earlier*). Personal pronouns (PRP) are well-divided into three categories, roughly: nominative case, accusative case, and sentence-initial nominative case, which each correlate very strongly with syntactic position. As another example of a specific trend which was mentioned by Klein and Manning (2003a), adverbs (RB) do contain splits for adverbs under ADVPs (*also*), NPs (*only*), and VPs (*not*).

Functional categories generally show fewer splits, but those splits that they do exhibit are known to be strongly correlated with syntactic behavior. For example, determiners (DT) divide along several axes: definite (*the*), indefinite (*a*), demonstrative (*this*), quantificational (*some*), negative polarity (*no*, *any*), and various upper- and lower-case distinctions inside these types. Here, it is interesting to note that these distinctions emerge in a predictable order (see Fig. 2.2 for DT splits), beginning with the distinction between demonstratives and non-demonstratives, with the other distinctions emerging subsequently; this echoes the result of

**Table 2.7** The most frequent three words in the subcategories of several part-of-speech tags

IN				RB			
IN-0	In	With	After	RB-0	recently	previously	still
IN-1	In	For	At	RB-1	here	back	now
IN-2	in	for	on	RB-2	very	highly	relatively
IN-3	of	for	on	RB-3	so	too	as
IN-4	from	on	with	RB-4	also	now	still
IN-5	at	for	by	RB-5	however	Now	However
IN-6	by	in	with	RB-6	much	far	enough
IN-7	for	with	on	RB-7	even	well	then
IN-8	If	While	As	RB-8	as	about	nearly
IN-9	because	if	while	RB-9	only	just	almost
IN-10	whether	if	That	RB-10	ago	earlier	later
IN-11	that	like	whether	RB-11	rather	instead	because
IN-12	about	over	between	RB-12	back	close	ahead
IN-13	as	de	Up	RB-13	up	down	off
IN-14	than	ago	until	RB-14	not	Not	maybe
IN-15	out	up	down	RB-15	n't	not	also

VBZ				DT			
VBZ-0	gives	sells	takes	DT-0	the	The	a
VBZ-1	comes	goes	works	DT-1	A	An	Another
VBZ-2	includes	owns	is	DT-2	The	No	This
VBZ-3	puts	provides	takes	DT-3	The	Some	These
VBZ-4	says	adds	Says	DT-4	all	those	some
VBZ-5	believes	means	thinks	DT-5	some	these	both
VBZ-6	expects	makes	calls	DT-6	That	This	each
VBZ-7	plans	expects	wants	DT-7	this	that	each
VBZ-8	is	's	gets	DT-8	the	The	a
VBZ-9	's	is	remains	DT-9	no	any	some
VBZ-10	has	's	is	DT-10	an	a	the
VBZ-11	does	Is	Does	DT-11	a	this	the

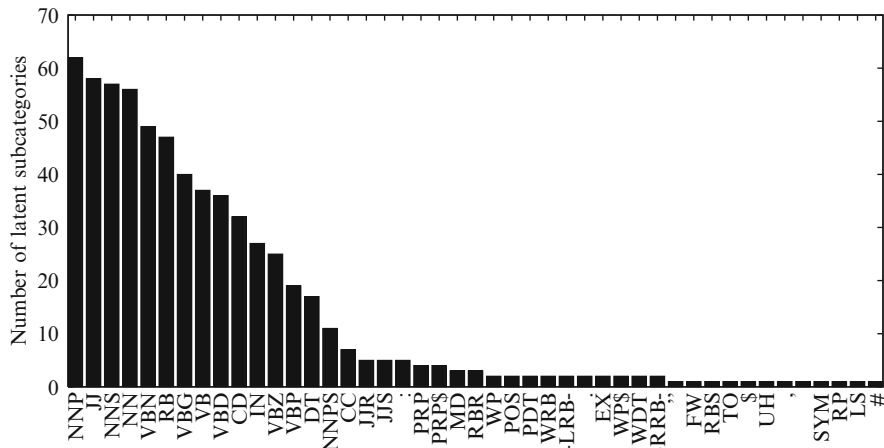
  

NNP				CD			
NNP-0	Jr.	Goldman	INC.	CD-0	1	50	100
NNP-1	Bush	Noriega	Peters	CD-1	8.50	15	1.2
NNP-2	J.	E.	L.	CD-2	8	10	20
NNP-3	York	Francisco	Street	CD-3	1	30	31
NNP-4	Inc	Exchange	Co	CD-4	1989	1990	1988
NNP-5	Inc.	Corp.	Co.	CD-5	1988	1987	1990
NNP-6	Stock	Exchange	York	CD-6	two	three	five
NNP-7	Corp.	Inc.	Group	CD-7	one	One	Three
NNP-8	Congress	Japan	IBM	CD-8	12	34	14
NNP-9	Friday	September	August	CD-9	78	58	34
NNP-10	Shearson	D.	Ford	CD-10	one	two	three
NNP-11	U.S.	Treasury	Senate	CD-11	million	billion	trillion
NNP-12	John	Robert	James	PRP			
NNP-13	Mr.	Ms.	President	PRP-0	It	He	I
NNP-14	Oct.	Nov.	Sept.	PRP-1	it	he	they
NNP-15	New	San	Wall	PRP-2	it	them	him

JJS				RBR			
JJS-0	largest	latest	biggest	RBR-0	further	lower	higher
JJS-1	least	best	worst	RBR-1	more	less	More
JJS-2	most	Most	least	RBR-2	earlier	Earlier	later

See text for discussion



**Fig. 2.9** Number of latent lexical subcategories determined by our split-merge procedure after 6 SM cycles

Klein and Manning (2003a), where the authors chose to distinguish the demonstrative contrast, but not the additional ones learned here.

Another very important distinction, as shown in Klein and Manning (2003a), is the various subdivisions in the preposition class (IN). Learned first is the split between subordinating conjunctions like *that* and proper prepositions. Then, subdivisions of each emerge: *wh*-subordinators like *if*, noun-modifying prepositions like *of*, predominantly verb-modifying ones like *from*, and so on.

Many other interesting patterns emerge, including many classical distinctions not specifically mentioned or modeled in previous work. For example, the *wh*-determiners (WDT) split into one class for *that* and another for *which*, while the *wh*-adverbs align by reference type: event-based *how* and *why* vs. entity-based *when* and *where*. The possessive particle (POS) has one class for the standard *'s*, but another for the plural-only apostrophe. As a final example, the cardinal number nonterminal (CD) induces various categories for dates, fractions, spelled-out numbers, large (usually financial) digit sequences, and others.

## 2.6.2 Phrasal Subcategories

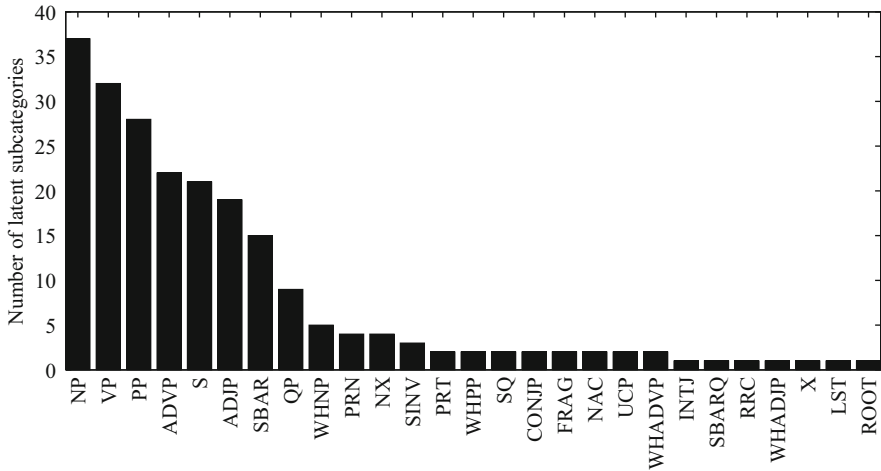
Analyzing the splits of phrasal nonterminals is more difficult than for lexical categories, and we can merely give illustrations. We show some of the top productions of two categories in Table 2.8.

A nonterminal split can be used to model an otherwise uncaptured correlation between that category's external context (e.g. its parent category) and its internal context (e.g. its child categories). A particularly clean example of a split correlating

**Table 2.8** The most frequent three productions of some latent phrasal subcategories

ADVP			
ADVP-0	RB-13 NP-2	RB-13 PP-3	IN-15 NP-2
ADVP-1	NP-3 RB-10	NP-3RBR-2	NP-3 IN-14
ADVP-2	IN-5 JJS-1	RB-8RB-6	RB-6 RBR-1
ADVP-3	RBR-0	RB-12 PP-0	RP-0
ADVP-4	RB-3 RB-6	ADVP-2 SBAR-8	ADVP-2 PP-5
ADVP-5	RB-5	NP-3 RB-10	RB-0
ADVP-6	RB-4	RB-0	RB-3 RB-6
ADVP-7	RB-7	IN-5 JJS-1	RB-6
ADVP-8	RB-0	RBS-0	RBR-1 IN-14
ADVP-9	RB-1	IN-15	RBR-0

SINV			
SINV-0	VP-14 NP-7	VP-14	VP-15 NP-7 NP-9
SINV-1	VP-14 NP-7 .-0 S-6 ,-0 VP-14 NP-7 .-0 S-11 VP-14 NP-7 .-0		



**Fig. 2.10** Number of latent phrasal subcategories determined by our split-merge procedure after 6 SM cycles

external with internal contexts is the inverted sentence category (SINV), which has only two subcategories (see Fig. 2.10), one which usually has the ROOT category as its parent (and which has sentence final punctuation as its last child), and a second subcategory which occurs in embedded contexts (and does not end in punctuation). Such patterns are common, but often less easy to predict. For example, possessive NPs get two subcategories, depending on whether their possessor is a person/country or an organization. The external correlation turns out to be that people and countries are more likely to possess a subject NP, while organizations are more likely to possess an object NP.

Nonterminal splits can also be used to relay information between distant tree nodes, though untangling this kind of propagation and distilling it into clean examples is not trivial. As one example, the subcategory S-12 (matrix clauses) occurs only under the ROOT category. S-12’s children usually include NP-8, which in turn usually includes PRP-0, the capitalized nominative pronouns, DT-{1,2,6} (the capitalized determiners), and so on. This same propagation occurs even more frequently in the intermediate categories, with, for example, one subcategory of NP category specializing in propagating proper noun sequences.

Verb phrases, unsurprisingly, also receive a full set of subcategories, including categories for infinitive VPs, passive VPs, several for intransitive VPs, several for transitive VPs with NP and PP objects, and one for sentential complements. As an example of how lexical splits can interact with phrasal splits, the two most frequent rewrites involving intransitive past tense verbs (VBD) involve two different VPs and VBDs: VP-14  $\rightarrow$  VBD-13 and VP-15  $\rightarrow$  VBD-12. The difference is that VP-14s are main clause VPs, while VP-15s are subordinate clause VPs. Correspondingly, VBD-13s are verbs of communication (*said*, *reported*), while VBD-12s are an assortment of verbs which often appear in subordinate contexts (*did*, *began*).

Other interesting phenomena also emerge. For example, intermediate categories, which in previous work were very heavily, manually split using a Markov process, end up encoding processes which are largely Markov, but more complex. For example, some classes of adverb phrases (those with RB-4 as their head) are ‘forgotten’ by the  $\overline{VP}$  intermediate grammar. The relevant rule is the very probable  $\overline{VP}$ -2  $\rightarrow$   $\overline{VP}$ -2 ADVP-6; adding this ADVP to a growing VP does not change the VP subcategory. In essence, at least a partial distinction between verbal arguments and verbal adjuncts has been learned (as exploited in Collins (1999), for example).

### 2.6.3 Multilingual Analysis

As we saw in Sect. 2.5.4, latent variable grammars achieve state-of-the-art performance on a wide array of syntactically very different languages. We analyzed and compared the learned subcategories for different languages and found many similarities. As in the case of English, the learned subcategories exhibit interesting linguistic interpretations. Tables 2.9–2.12 show selected subcategories for a number of different part-of-speech (POS) categories from randomly selected grammars after four split-merge cycles. Where applicable (most notably for Bulgarian and German) we see subcategories for different cases and genders. A particularly clean example is the determiner category (ART) for German. We should note here that some of the POS tags in the Bulgarian treebank have been annotated with gender information already (indicated by a dashed line in the table), while we automatically learn those distinction for others, for example for the personal demonstratives (PDA).<sup>16</sup> Across

---

<sup>16</sup>In a separate experiment, we removed the gender annotation and trained our model on this simplified tag set. As one might expect, many of the learned subcategories automatically recovered the gender distinctions.

**Table 2.9** The most frequent words, their translations and grammatical classification for several Bulgarian POS tags ({MASCULINE, FEMININE, NEUTER}-{SINGULAR, PLURAL})

BULGARIAN			
Pda-1 MASCULINE DEMONSTRATIVES	такъв <i>that</i> M-S	- -	- -
Pda-2 FEMININE DEMONSTRATIVES	такава <i>that</i> F-S	Такава <i>That</i> F-S	- -
Pda-3 PLURAL DEMONSTRATIVES	такива <i>these</i> F-P	Такива <i>These</i> F-P	- -
Pda-5 CAPITALIZED DEMONSTRATIVES	Такъв <i>That</i> M-S	Такова <i>That</i> N-S	Такива <i>These</i> F-P
Md-0 AUGMENTATIVE ADVERBIALS	повече <i>more</i>	много <i>a lot</i>	повечето <i>more</i>
Md-1 CAPITALIZED AUGM. ADVERBIALS	Много <i>A lot</i>	Повече <i>More</i>	Най-много <i>The most</i>
Md-4 DIMINUTIVE ADVERBIALS	малко <i>little</i>	Малко <i>Little</i>	Най-малко <i>the least</i>
Vpiicao-1 TRANSITIVE VERBS	служил <i>served</i>	потил <i>sweated</i>	надявал <i>hoped</i>
Vpiicao-3 FORMS OF “CAN”	могъл <i>could</i>	могли <i>could</i>	могла <i>could</i>
Vpiicao-4 INTRANSITIVE VERBS	вървели <i>walked</i>	прекалявала <i>overdid</i>	отивал <i>went</i>
Ncmsi-0 MONTHS	месец <i>month</i>	декември <i>decemeber</i>	януари <i>january</i>
Ncmsi-3 TYPES OF STATEMENTS	въпрос <i>question</i>	договор <i>contract</i>	отговор <i>answer</i>
Ncmsi-4 JOBS MASCULINE	студент <i>student</i> M	пенсионер <i>retiree</i> M	инженер <i>engineer</i> M
Ncfsi-0 FEMININE FAMILY MEMBERS	жена <i>woman</i>	баба <i>grandmother</i>	майка <i>mother</i>
Ncfsi-3 JOBS FEMININE	студентка <i>student</i> F	пенсионерка <i>retiree</i> F	учителка <i>teacher</i> F
Name-0 LAST NAMES	Костов <i>Kostov</i>	Буш <i>Bush</i>	Филчев <i>Filchev</i>
Name-3 CITY NAMES	София <i>Sofia</i>	Пловдив <i>Plovdiv</i>	Тексас <i>Texas</i>
Name-4 COUNTRY NAMES	Европа <i>Europe</i>	България <i>Bulgaria</i>	Югославия <i>Yugoslavia</i>
Name-7 STREET NAMES	Левски <i>Levski</i>	Дондуков <i>Dondukov</i>	Раковски <i>Rakovski</i>
Momsi-0 ROMAN NUMBERS	II	XIX	XX
Momsi-1 MASCULINE	втори <i>second</i> M	трети <i>third</i> M	пети <i>fifth</i> M
Momsi-4 DIGITS	12	11	15
Mofsi-0 YEARS	2001	2000	2002
Mofsi-3 FEMININE	втора <i>second</i> F	първа <i>first</i> F	трета <i>third</i> F
Mofsi-4 DIGITS	1	10	2

**Table 2.10** The most frequent words, their translations and grammatical classification for several Chinese POS tags ({MASCULINE, FEMININE, NEUTER}-{SINGULAR, PLURAL})

CHINESE			
NN-4 LOCATIONS	地区 <i>region</i>	国家 <i>country</i>	省 <i>province</i>
NN-6 ECONOMIC CONCEPTS	投 <i>investment</i>	经济 <i>economy</i>	生 <i>production</i>
NN-8 MARKET	市 <i>market</i>	政府 <i>government</i>	工 <i>industry</i>
NN-14 “BEGINNING OF NEWS STORY”	by wire	者 <i>reporter</i>	picture
NT-0 YEARS	二〇〇〇年 <i>2000</i>	一九九五年 <i>1995</i>	一九九六年 <i>1996</i>
NT-5 RELATIVE YEARS	今年 <i>this year</i>	去年 <i>last year</i>	明年 <i>next year</i>
NT-6 DAYS	一日 <i>1st</i>	二日 <i>2nd</i>	十五日 <i>15th</i>
NT-12 MONTHS	十一月 <i>November</i>	六月 <i>June</i>	十月 <i>October</i>
NR-5 CHINESE CITIES	北京 <i>Beijing</i>	上海 <i>Shanghai</i>	广州 <i>Guangzhou</i>
NR-5 COUNTRY NAMES	果 <i>Congo</i>	香港 <i>Hong Kong</i>	中国 <i>China</i>
NR-7 ABBR. COUNTRY NAMES	美 <i>America</i>	英 <i>Britain</i>	日 <i>Japan</i>
VV-5 IMPROVEMENT VERBS	展 <i>develop</i>	加 <i>increase</i>	grow
VV-9 FORMS OF “MAKE”	let	使 <i>cause</i>	令 <i>force</i>
VV-6 SENTENTIAL ARGUMENT	希望 <i>wish</i>	believe	想 <i>think</i>
VV-12 DIRECTIONAL VERBS	起来 <i>come up</i>	出来 <i>come out</i>	下来 <i>come down</i>

all languages, we see subcategories for years, months, days, job titles, first and last names, locations, etc. Often times subcategories for verbs taking particular types of arguments will emerge (Tables 2.9–2.13).

2.7 Summary and Future Work

In this chapter we presented latent variable grammars, which allow fast, accurate parsing, in multiple languages and domains. Starting from an observed, but coarse, treebank we induce a hierarchy of increasingly refined grammars. We use a split-merge approach to learn a tight fit to the training data, while controlling the grammar size. Parameter smoothing is furthermore applied to overcome data fragmentation

**Table 2.11** The most frequent words, their translations and grammatical classification for several French POS tags ({MASCULINE, FEMININE}-{SINGULAR, PLURAL})

FRENCH			
D-0 DETERMINERS	les <i>the</i> M/F-P	la <i>the</i> F-S	le <i>the</i> M-S
D-2 CAPITALIZED DETERMINERS	Le <i>The</i> M-S	La <i>The</i> F-S	Les <i>The</i> M/N-P
D-2 FEMININE DETERMINERS	le <i>the</i> F-S	l' <i>the</i> F-S	cette <i>this</i> F-S
D-3 SPELLED OUT NUMBERS	deux <i>two</i>	trois <i>three</i>	un <i>one</i>
D-6 NUMBERS	1	50	40
PRO-0 RELATIVE PRONOUNS	qui <i>who</i>	que <i>that</i>	où <i>where</i>
PRO-2 INDEFINITE	un <i>one</i>	même <i>same</i>	autre <i>other</i>
PRO-3 DEMONSTATIVES	celui <i>that</i> M-S	celle <i>that</i> F-S	ceux <i>that</i> M/F-P
PRO-6 POSSESSIVE	dont <i>whose</i>	- -	- -
ADV-1 QUANTITY	plus <i>more</i>	moins <i>less</i>	environ <i>about</i>
ADV-3 SENTENCE INITIAL	Ainsi <i>Thus</i>	Enfin <i>Finally</i>	Pourtant <i>Though</i>
ADV-5 PREPOSITIONS	au <i>to</i>	à <i>to</i>	en <i>in</i>
ADV-9 NEGATION	ne	Ne	N'
C-0 COORDINATING CONJUNCTIONS	et <i>and</i>	ou <i>or</i>	- -
C-1 THAT	que <i>that</i>	- -	- -
C-6 CAPITALIZED CONJUNCTIONS	Mais <i>But</i>	Et <i>And</i>	Or <i>And yet</i>
NN-4 JOBS	président <i>President</i>	ministre <i>Minister</i>	politique <i>Politician</i>
NN-7 LOCATIONS	France <i>France</i>	Bretagne <i>Brittany</i>	Unis <i>United</i>
NN-10 FROM	de <i>from</i>	d' <i>from</i>	des <i>from</i>
NN-11 TITLES	M. <i>Mr.</i>	Mr <i>Mrs.</i>	Mme <i>Mrs.</i>
NN-13 UNITS	% %	milliards <i>billion</i>	millions <i>million</i>
NN-14 MONTH NAMES	janvier <i>January</i>	décembre <i>December</i>	juillet <i>July</i>

**Table 2.12** The most frequent words, their translations and grammatical classification for several German POS tags ({MASCULINE, FEMININE, NEUTER}-{SINGULAR, PLURAL}-{NOMINATIVE, GENITIVE, DATIVE, ACCUSATIVE})

GERMAN			
ART-1 NOMINATIVE DETERMINERS	die <i>the</i> F-S-N	der <i>the</i> M-S-N	das <i>the</i> N-S-N
ART-0 CAPITALIZED DETERMINERS	Die <i>The</i> F-S-N	Der <i>The</i> M-S-N	Das <i>The</i> N-S-N
ART-2 GENITIVE DETERMINERS	der <i>the</i> F-S-G	des <i>the</i> M/N-S-G	eines <i>a</i> M/N-S-G
ART-5 DATIVE DETERMINERS	einem <i>a</i> M/N-S-D	einen <i>a</i> M/N-P-D	einer <i>a</i> F-S-D
ART-7 ACCUSATIVE DETERMINERS	den <i>the</i> M/N-S-A	die <i>the</i> F-S-A	einen <i>a</i> M/N-S-A
PPER-0 DATIVE PRONOUNS	ihm <i>him</i> DAT	mir <i>me</i> DAT	ihr <i>her</i> DAT
PPER-1 ACCUSATIVE PRONOUNS	ihn <i>him</i> ACC	ihnen <i>them</i> ACC	uns <i>us</i> ACC
PPER-3 NOMINATIVE PRONOUNS	er <i>he</i>	sie <i>she</i>	es <i>it</i>
PPER-4 NOMINATIVE PRONOUNS	Sie <i>She</i>	Er <i>He</i>	Es <i>It</i>
ADV-0 WEEKDAYS	sonntags <i>sundays</i>	samstags <i>saturdays</i>	montags <i>mondays</i>
ADV-1 TIMES OF THE DAY	abend <i>evening</i>	morgen <i>morning</i>	oben <i>above</i>
ADV-4 CAPITALIZED ADVERBS	So <i>That</i>	Da <i>That</i>	Dann <i>Then</i>
ADV-6 ADVERBS	aber <i>but</i>	dann <i>then</i>	jedoch <i>however</i>
NN-4 JOBS	Bürgermeister <i>mayor</i>	Präsident <i>president</i>	Trainer <i>coach</i>
NN-7 LOCATIONS	Stadt <i>city</i>	Platz <i>square</i>	Strasse <i>street</i>
NN-11 WEEKDAYS	Samstag <i>Saturday</i>	Sonntag <i>Sunday</i>	Dienstag <i>Tuesday</i>
NN-14 MONTH NAMES	Juni <i>June</i>	Juli <i>July</i>	Mai <i>May</i>
NE-1 FIRST NAMES	Peter	Michael	Klaus
NE-4 CITY NAMES	Düsseldorf	Frankfurt	München
NE-7 COUNTRY NAMES	USA	Schweiz	EG
NE-15 NEWS AGENCIES	dpa	de	AP
NE-13 POLITICAL PARTIES	CDU	SPD	FDP
CARD-0 SPORTS RESULTS	6:4	2:1	1:0
CARD-1 YEAR NUMBERS	1992	1993	1991
CARD-3 SPELLED OUT NUMBERS	zwei <i>two</i>	drei <i>three</i>	fünf <i>five</i>
CARD-5 ROUND NUMBERS	100	20	50

**Table 2.13** The most frequent words, their translations and grammatical classification for several Italian POS tags ({MASCULINE, FEMININE, NEUTER}-{SINGULAR, PLURAL})

ITALIAN			
ART-2 INDEFINITE ARTICLES	una <i>a F-S</i>	un' <i>a F-S</i>	un <i>a M-S</i>
ART-4 DEFINITE ARTICLES SINGULAR	la <i>the F-S</i>	La <i>The F-S</i>	lo <i>the M-P</i>
ART-6 DEFINITE ARTICLES PLURAL	le <i>the F-P</i>	i <i>the M-P</i>	gli <i>the M-P</i>
ART-9 CONTRACTED PREPOSITIONS	della <i>from the F-S</i>	alla <i>to the F-S</i>	dalla <i>from the F-S</i>
ART-13 CONTRACTED PREPOSITIONS BEFORE VOWEL	dell' <i>from the F-S</i>	all' <i>to the F-S</i>	dall' <i>from the F-S</i>
NOU-4 UNIT NOUNS	anni <i>years</i>	giorni <i>days</i>	metri <i>meters</i>
NOU-7 THINGS	articoli <i>articles</i>	leggi <i>laws</i>	cose <i>things</i>
NOU-13 PLACES	Tirana <i>Tirana</i>	Valona <i>Valona</i>	Albania <i>Albania</i>
PREP-1 TO + ARTCLE	al <i>to the M-S</i>	alla <i>to the F-S</i>	all' <i>to the M-S</i>
PREP-3 FROM + ARTCLE	del <i>from the M-S</i>	della <i>from the F-S</i>	dell' <i>from the M-S</i>
PREP-5 SENTENCE INITIAL	In <i>In</i>	A <i>To</i>	Per <i>For</i>
VMA-1 PAST PARTICIPLE	visto <i>seen</i>	fatto <i>done</i>	stato <i>been</i>
VMA-3 PRESENT TENSE	applicano <i>apply</i>	serve <i>serve</i>	osservano <i>observe</i>
VMA-7 PRESENT PARTICIPLE	servente <i>serving</i>	dominante <i>dominating</i>	esistenti <i>existing</i>
VMA-10 INFINITIVES	vedere <i>to see</i>	fare <i>to do</i>	pagare <i>to pay</i>
NUM-0 SMALL NUMBERS	1	2	3
NUM-1 SPELLED OUT NUMBERS	due <i>two</i>	tre <i>three</i>	sei <i>six</i>
NUM-2 ROUND NUMBERS	50	20	10
NUM-3 ARBITRARY NUMBERS	832	940	874

and improve generalization performance. The resulting grammars are not only significantly more accurate, than that of previous work, but also much smaller. While all this is accomplished with only automatic learning, the resulting grammar is human-interpretable. It shows most of the manually introduced annotations discussed by previous work, but also learns other linguistic phenomena.

We also presented a coarse-to-fine inference procedure, which gives tremendous speed-ups over direct inference in the most refined model. In our multipass

approach, we repeatedly re-parse with increasingly more refined grammars, ruling out large portions of the search space. While our inference scheme is approximate, it produces very few search errors in practice because we compute a hierarchy of grammars specifically for pruning, by minimizing the KL divergence between the induced tree distributions. Finally, we investigated different objective functions for parse selection and showed that the appropriate risk-minimizing methodology significantly improves parsing accuracy.

The parser along with grammars for a number of languages is publicly available at <http://nlp.cs.berkeley.edu> and is being actively used by other researchers. It has been adapted to other languages, for example French (Crabbé and Candito 2008) and Chinese (Huang and Harper 2009), or used in systems for sentence segmentation (Favre et al. 2008), and most notably in the currently best syntactic machine translation system (Chiang et al. 2009).

But there also lots of other promising avenues worth exploring. For example, different parsers seem to be making very different errors, and it has been shown that parsing accuracy can be significantly improved by combining n-best lists from different systems (Zhang et al. 2009). However, as demonstrated by Huang (2008) n-best lists have only limited variety and the combination really should be done on the parse forests instead. Petrov (2010) presents a nice product of experts approach in which multiple grammars are combined to produce significantly improved parsing accuracies. The surprising result of that work is that the variance provided by the EM algorithm used for training is sufficient to produce latent variable grammars that vary widely in the errors they make. Rather than trying to pick the best grammar, combining all grammars in a product model produces accuracies that far exceed the accuracy of the best individual model. To further boost performance on out-of-domain text, techniques like the one presented in McClosky et al. (2006) and Huang and Harper (2009) could be extended. Finally, one way of improving parsing performance for resource-poor languages, is by exploiting parallel data and good parsers from a resource-rich language. Burkett and Klein (2008) present such a multilingual parsing systems, however, their system works only in the presence of labeled bitexts and it would be exciting to extend their work to work with less supervision. Huang et al. (2010) present self-training experiments with latent variable grammars and demonstrate the robustness of latent variable grammars across domains. Their experiments also show self-training on large amounts of unlabeled data can even further improve parsing accuracy.



<http://www.springer.com/978-3-642-22742-4>

Coarse-to-Fine Natural Language Processing

Petrov, S.

2012, XXII, 106 p., Hardcover

ISBN: 978-3-642-22742-4