The Leaf Projection Path View of Parse Trees: Exploring String Kernels for HPSG Parse Selection

Kristina Toutanova
CS Dept, Stanford University
353 Serra Mall
Stanford 94305, CA
USA,
kristina@cs.stanford.edu

Penka Markova
EE Dept, Stanford University
350 Serra Mall
Stanford 94305, CA,
USA,
penka@cs.stanford.edu

Christopher Manning
CS Dept, Stanford University
353 Serra Mall
Stanford 94305, CA,
USA,
manning@cs.stanford.edu

Abstract

We present a novel representation of parse trees as lists of paths (*leaf projection paths*) from leaves to the top level of the tree. This representation allows us to achieve significantly higher accuracy in the task of HPSG parse selection than standard models, and makes the application of string kernels natural. We define tree kernels via string kernels on projection paths and explore their performance in the context of parse disambiguation. We apply SVM ranking models and achieve an exact sentence accuracy of 85.40% on the Redwoods corpus.

1 Introduction

In this work we are concerned with building statistical models for parse disambiguation – choosing a correct analysis out of the possible analyses for a sentence. Many machine learning algorithms for classification and ranking require data to be represented as real-valued vectors of fixed dimensionality. Natural language parse trees are not readily representable in this form, and the choice of representation is extremely important for the success of machine learning algorithms.

For a large class of machine learning algorithms, such an explicit representation is not necessary, and it suffices to devise a kernel function $K(x, y)$ which measures the similarity between inputs x and y. In addition to achieving efficient computation in high dimensional representation spaces, the use of kernels allows for an alternative view on the modelling problem as defining a similarity between inputs rather than a set of relevant features.

In previous work on discriminative natural language parsing, one approach has been to define features centered around lexicalized local rules in the trees (Collins, 2000; Shen and Joshi, 2003), similar to the features of the best performing lexicalized generative parsing models (Charniak, 2000; Collins, 1997). Additionally non-local features have been defined measuring e.g. parallelism and complexity of phrases in discriminative log-linear parse ranking models (Riezler et al., 2000).

Another approach has been to define tree kernels: for example, in (Collins and Duffy, 2001), the all-subtrees representation of parse trees (Bod, 1998) is effectively utilized by the application of a fast dynamic programming algorithm for computing the number of common subtrees of two trees. Another tree kernel, more broadly applicable to Hierarchical Directed Graphs, was proposed in (Suzuki et al., 2003). Many other interesting kernels have been devised for sequences and trees, with application to sequence classification and parsing. A good overview of kernels for structured data can be found in (Gaertner et al., 2002).

Here we propose a new representation of parse trees which (*i*) allows the localization of broader useful context, (*ii*) paves the way for exploring kernels, and (*iii*) achieves superior disambiguation accuracy compared to models that use tree representations centered around context-free rules.

Compared to the usual notion of discriminative models (placing classes on rich observed data) discriminative PCFG parsing with plain context free rule features may look naive, since most of the features (in a particular tree) make no reference to observed input at all. The standard way to address this problem is through lexicalization, which puts an element of the input on each tree node, so all features do refer to the input. This paper explores an alternative way of achieving this that gives a broader view of tree contexts, extends naturally to exploring kernels, and performs better.

We represent parse trees as lists of paths (*leaf projection paths*) from words to the top level of the tree, which includes both the head-path (where the word is a syntactic head) and the non-head path. This allows us to capture for example cases of non-head dependencies which were also discussed by (Bod, 1998) and were used to motivate large subtree features, such as “more careful than his sister” where “careful” is analyzed as head of the adjective phrase, but “more” licenses the “than” comparative clause. This representation of trees as lists of projection

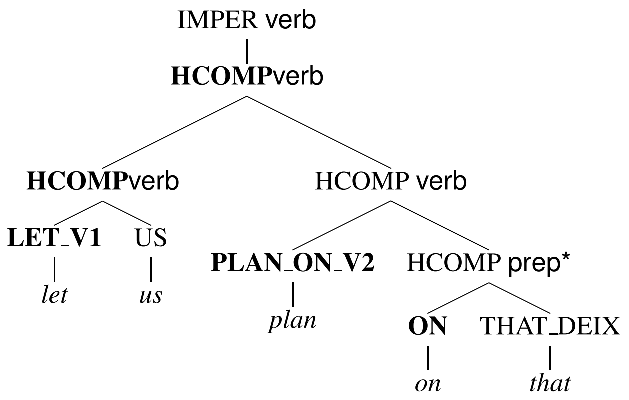

Figure 1: Derivation tree for the sentence *Let us plan on that.*

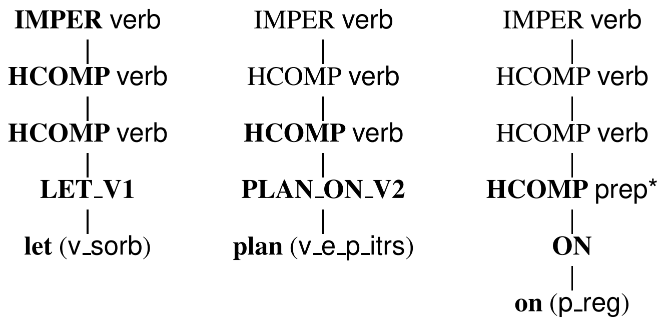

Figure 2: Paths to top for three leaves. The nodes in bold are head nodes for the leaf word and the rest are non-head nodes.

paths (strings) allows us to explore string kernels on these paths and combine them into tree kernels.

We apply these ideas in the context of parse disambiguation for sentence analyses produced by a Head-driven Phrase Structure Grammar (HPSG), the grammar formalism underlying the Redwoods corpus (Oepen et al., 2002). HPSG is a modern constraint-based lexicalist (or "unification") grammar formalism.[1] We build discriminative models using Support Vector Machines for ranking (Joachims, 1999). We compare our proposed representation to previous approaches and show that it leads to substantial improvements in accuracy.

2 The Leaf Projection Paths View of Parse Trees

2.1 Representing HPSG Signs

In HPSG, sentence analyses are given in the form of HPSG signs, which are large feature structures containing information about syntactic and semantic properties of the phrases.

As in some of the previous work on the Redwoods corpus (Toutanova et al., 2002; Toutanova and Manning, 2002), we use the derivation trees as the main representation for disambiguation. Derivation trees record the combining rule schemas of the HPSG grammar which were used to license the sign by combining initial lexical types. The derivation tree is also the fundamental data stored in the Redwoods treebank, since the full sign can be reconstructed from it by reference to the grammar. The internal nodes represent, for example, head-complement, head-specifier, and head-adjunct schemas, which were used to license larger signs out of component parts. A derivation tree for the sentence *Let us plan on that* is shown in Figure 1.[2]

[1]For an introduction to HPSG, see (Pollard and Sag, 1994).

Additionally, we annotate the nodes of the derivation trees with information extracted from the HPSG sign. The annotation of nodes is performed by extracting values of feature paths from the feature structure or by propagating information from children or parents of a node. In theory with enough annotation at the nodes of the derivation trees, we can recover the whole HPSG signs.

Here we describe three node annotations that proved very useful for disambiguation. One is annotation with the values of the feature path synsem.local.cat.head – its values are basic parts of speech such as *noun, verb, prep, adj, adv*. Another is phrase structure category information associated with the nodes, which summarizes the values of several feature paths and is available in the Redwoods corpus as Phrase-Structure trees. The third is annotation with lexical type (*le-type*), which is the type of the head word at a node. The preterminals in Figure 1 are lexical item identifiers — identifiers of the lexical entries used to construct the parse. The *le-types* are about 500 types in the HPSG type hierarchy and are the direct super-types of the lexical item identifiers. The le-types are not shown in this figure, but can be seen at the leaves in Figure 2. For example, the lexical type of **LET_V1** in the figure is *v_sorb*. In Figure 1, the only annotation performed is with the values of synsem.local.cat.head.

2.2 The Leaf Projection Paths View

The *projection path* of a leaf is the sequence of nodes from the leaf to the root of the tree. In Figure 2, the *leaf projection paths* for three of the words are shown.

We can see that a node in the derivation tree par-

[2]This sentence has three possible analyses depending on the attachment of the preposition "on" and whether "on" is an adjunct or complement of "plan".

ticipates in the projection paths of all words dominated by that node. The original local rule configurations — a node and its children, do not occur jointly in the projection paths; thus, if special annotation is not performed to recover it, this information is lost.

As seen in Figure 2, and as is always true for a grammar that produces non-crossing lexical dependencies, there is an initial segment of the projection path for which the leaf word is a syntactic head (called *head path* from here on), and a final segment for which the word is not a syntactic head (called *non-head path* from here on). In HPSG non-local dependencies are represented in the final semantic representation, but can not be obtained via syntactic head annotation.

If, in a traditional parsing model that estimates the likelihood of a local rule expansion given a node (such as e.g (Collins, 1997)), the tree nodes are annotated with the word of the lexical head, some information present in the word projection paths can be recovered. However, this is only the information in the head path part of the projection path. In further experiments we show that the non-head part of the projection path is very helpful for disambiguation.

Using this representation of derivation trees, we can apply string kernels to the leaf projection paths and combine those to obtain kernels on trees. In the rest of this paper we explore the application of string kernels to this task, comparing the performance of the new models to models using more standard rule features.

3 Tree and String Kernels

3.1 Kernels and SVM ranking

From a machine learning point of view, the parse selection problem can be formulated as follows: given m training examples $(s_i, \Phi(t_{i,0}) \ldots \Phi(t_{i,p_i-1}))$, where each s_i is a natural language sentence, m is the number of such sentences, $i = 1 \ldots m$, $t_{i,j}$ is a parse tree for s_i, p_i is the number of parses for a given sentence s_i, $\Phi(t_{i,j})$ is a feature representation for the parse tree $t_{i,j}$, and we are given the training information which of all $t_{i,j}$ is the correct parse – learn how to correctly identify the correct parse of an unseen test sentence.

One approach for solving this problem is via representing it as an SVM (Vapnik, 1998) ranking problem, where (without loss of generality) $t_{i,0}$ is assumed to be the correct parse for s_i. The goal is to learn a parameter vector $\vec{w}$, such that the score of $t_{i,0}$ ($\vec{w} \cdot \Phi(t_{i,0})$) is higher than the scores of all other parses for the sentence. Thus we optimize for:

$$\min \frac{1}{2}\vec{w} \cdot \vec{w} + C \sum \zeta_{i,j}$$

$$\forall i \forall j > 0 : \vec{w} \cdot (\Phi(t_{i,0}) - \Phi(t_{i,j})) \geq 1 - \zeta_{i,j}$$
$$\forall i \forall j > 0 : \zeta_{i,j} \geq 0$$

The $\zeta_{i,j}$ are slack variables used to handle the non-separable case. The same formulation has been used in (Collins, 2001) and (Shen and Joshi, 2003).

This problem can be solved by solving the dual, and thus we would only need inner products of the feature vectors. This allows for using the kernel trick, where we replace the inner product in the representation space by inner product in some feature space, usually different from the representation space. The advantage of using a kernel is associated with the computational effectiveness of computing it (it may not require performing the expensive transformation Φ explicitly).

We learn SVM ranking models using a tree kernel defined via string kernels on projection paths.

3.2 Kernels on Trees Based on Kernels on Projection Paths

So far we have defined a representation of parse trees as lists of strings corresponding to projection paths of words. Now we formalize this representation and show how string kernels on projection paths extend to tree kernels.

We introduce the notion of a keyed string — a string that has a key, which is some letter from the alphabet Σ of the string. We can denote a keyed string by a pair (a, x), where $a \in \Sigma$ is the key, and x is the string. In our application, a key would be a word w, and the string would be the sequence of derivation tree nodes on the head or non-head part of the projection path of the word w. Additionally, for reducing sparsity, for each keyed string (w, x), we also include a keyed string (le_w, x), where le_w is the le-type of the word w. Thus each projection path occurs twice in the list representation of the tree – once headed by the word, and once by its le-type. In our application, the strings x are sequences of annotated derivation tree nodes, e.g. x="*LET_V1:verb HCOMP:verb HCOMP:verb IMPER:verb*" for the head projection path of *let* in Figure 2. The non-head projection path of *let* is empty.

For a given kernel K on strings, we define its extension to keyed strings as follows: $K((a, x), (b, y)) = K(x, y)$, if $a = b$, and $K((a, x), (b, y)) = 0$, otherwise. We use this construction for all string kernels applied in this work.

Given a tree $t_1 = ((a_1, x_1), \ldots, (a_n, x_n))$ and a tree $t_2 = ((b_1, y_1), \ldots, (b_m, y_m))$, and a kernel K on keyed strings, we define a kernel KT on the trees as follows:

$$KT(t_1, t_2) = \sum_{i=1}^{n} \sum_{j=1}^{m} K((a_i, x_i), (b_j, y_j))$$

This can be viewed as a convolution (Haussler, 1999) and therefore KT is a valid kernel (positive definite symmetric), if K is a valid kernel.

3.3 String Kernels

We experimented with some of the string kernels proposed in (Lodhi et al., 2000; Leslie and Kuang, 2003), which have been shown to perform very well for indicating string similarity in other domains. In particular we applied the N-gram kernel, Subsequence kernel, and Wildcard kernel. We refer the reader to (Lodhi et al., 2000; Leslie and Kuang, 2003) for detailed formal definition of these kernels, and restrict ourselves to an intuitive description here. In addition, we devised a new kernel, called Repetition kernel, which we describe in detail.

The kernels used here can be defined as the inner product of the feature vectors of the two strings $K(x,\ y)$= $\Phi(\mathrm{x})^T\ \Phi(y)$, with feature map from the space of all finite sequences from a string alphabet Σ to a vector space indexed by a set of subsequences from Σ. As a simple example, the 1-gram string kernel maps each string $x \in \Sigma^*$ to a vector with dimensionality $|\Sigma|$ and each element in the vector indicates the number of times the corresponding symbol from Σ occurs in x. For example, $\Phi_a^{1-gram}(abcadaf) = 3$.

The *Repetition kernel* is similar to the 1-gram kernel. It improves on the 1-gram kernel by better handling cases with repeated occurrences of the same symbol. Intuitively, in the context of our application, this kernel captures the tendency of words to take (or not take) repeated modifiers of the same kind. For example, it may be likely that a ceratin verb take one PP-modifier, but less likely for it to take two or more.

More specifically, the *Repetition kernel* is defined such that its vector space consists of all sequences from Σ composed of the same symbol. The feature map obtains matching of substrings of the input string to features, allowing the occurrence of gaps. There are two discount parameters λ_1 and λ_2. λ_1 serves to discount features for the occurrence of gaps, and λ_2 discounts longer symbol sequences.

Formally, for an input string x, the value of the feature vector for the feature index sequence $\alpha = a \ldots a$, $|\alpha| = k$, is defined as follows: Let s be the left-most minimal contiguous substring of x that contains α, $s = s_1 \ldots s_l$, where for indices $i_1 = 1, i_2, \ldots i_k = l$, $s_{i_1} = a = s_{i_2} = \ldots = s_{i_k}$. Then $\Phi_\alpha^{Repetition}(x)$=$\lambda_1^{l-k}\lambda_2^k$.

For our previous example, if $\lambda_1 = .5, \lambda_2 = 1$, $\Phi_a(abcadaf) = 1$, $\Phi_{aa}(abcadaf) = .25$, and $\Phi_{aaa}(abcadaf) = .125$.

The weighted *Wildcard kernel* performs matching by permitting a restricted number of matches to a wildcard character. A (k, m) wildcard kernel has as feature indices k-grams with up to m wildcard characters. Any character matches a wildcard. For example the 3-gram aab will match the feature index $a * b$ in a (3,1) wildcard kernel. The weighting is based on the number of wildcard characters used – the weight is multiplied by a discount λ for each wildcard.

The *Subsequence kernel* was defined in (Lodhi et al., 2000). We used a variation where the kernel is defined by two integers (k, g) and two discount factors λ_1 and λ_2 for gaps and characters. A subseq(k,g) kernel has as features all n-grams with $n \leq k$. The g is a restriction on the maximal span of the n-gram in the original string – e.g. if $k = 2$ and $g = 4$, the two letters of a 2-gram can be at most $g - k = 2$ letters apart in the original string. The weight of a feature is multiplied by λ_1 for each gap, and by λ_2 for each non-gap. For the example above, if $\lambda_1 = .5, \lambda_2 = 3, k = 2, g = 3$, $\Phi_{aa}(abcadaf) = 3 \times 3 \times .5 = .45$. The feature index aa matches only once in the string with a span at most 3 – for the sequence ada with 1 gap.

The details of the algorithms for computing the kernels can be found in the fore-mentioned papers (Lodhi et al., 2000; Leslie and Kuang, 2003). To summarize, the kernels can be implemented efficiently using tries.

4 Experiments

In this section we describe our experimental results using different string kernels and different feature annotation of parse trees. We learn Support Vector Machine (SVM) ranking models using the software package SVM^{light} (Joachims, 1999). We also normalized the kernels:

$$K'(t_1, t_2) = \frac{K(t_1,t_2)}{\sqrt{K(t_1,t_1)}\sqrt{K(t_2,t_2)}}.$$

For all tree kernels implemented here, we first extract all features, generating an explicit map to the space of the kernel, and learn SVM ranking models using SVM^{light} with a linear kernel in that space. Since the feature maps are not especially expensive for the kernels used here, we chose to solve the problem in its primal form. We were not aware of the existence of any fast software packages that could solve SVM ranking problems in the dual formulation. It is possible to convert the ranking problem into a classification problem using pairs of trees as shown in (Shen and Joshi, 2003). We have taken this approach in more recent work using string kernels requiring very expensive feature maps.

We performed experiments using the version of the Redwoods corpus which was also used in the work of (Toutanova et al., 2002; Osborne and Baldbridge, 2004) and others. There are 5307 annotated sentences in total, 3829 of which are ambiguous. The average sentence length of the ambiguous sentences is 7.8 words and the average number of parses per sentence is 10.8. We discarded the unambiguous sentences from the training and test sets. All models were trained and tested using 10-fold cross-validation. Accuracy results are reported as percentage of sentences where the correct analysis was ranked first by the model.

The structure of the experiments section is as follows. First we describe the results from a controlled experiment using a limited number of features, and aimed at comparing models using local rule features to models using leaf projection paths in Section 4.1. Next we describe models using more sophisticated string kernels on projection paths in Section 4.2.

4.1 The Leaf Projection Paths View versus the Context-Free Rule View

In order to evaluate the gains from the new representation, we describe the features of three similar models, one using the leaf projection paths, and two using derivation tree rules. Additionally, we train a model using only the features from the *head-path* parts of the projection paths to illustrate the gain of using the *non-head* path. As we will show, a model using only the head-paths has almost the same features as a rule-based tree model.

All models here use derivation tree nodes annotated with only the rule schema name as in Figure 1 and the synsem.local.cat.head value. We will define these models by their feature map from trees to vectors. It will be convenient to define the feature maps for all models by defining the set of features through templates. The value $\Phi_\alpha(t)$ for a feature α and tree t, will be the number of times α occurs in the tree. It is easy to show that the kernels on trees we introduce in Section 3.2, can be defined via a feature map that is the sum of the feature maps of the string kernels on projection paths.

As a concrete example, for each model we show all features that contain the node [HCOMP:verb] from Figure 1, which covers the phrase *plan on that*.

Bi-gram Model on Projection Paths (2PP)

The features of this model use a projection path representation, where the keys are not the words, but the *le-types* of the words. The features of this model are defined by the following template : $(leType, node_i, node_{i+1}, isHead?)$. $isHead?$ is a binary variable showing whether this feature matches a head or a non-head path, $leType$ is the le-type of the path leaf, and $node_i, node_{i+1}$ is a bi-gram from the path.

The node [HCOMP:verb] is part of the head-path for *plan*, and part of the non-head path for *on* and *that*. The le-types of the words *let, plan, on*, and *that* are, with abbreviations, *v_sorb, v_e_p, p_reg*, and *n_deic_pro_sg* respectively. In the following examples, the node labels are abbreviated as well; EOP is a special symbol for end of path and SOP is a special symbol for start of path. Therefore the features that contain the node will be:

```
(v_e_p,[PLAN_ON:verb],[HCOMP:verb],1)
(v_e_p,[HCOMP:verb],EOP,1)
(p_reg,SOP,[HCOMP:verb],0)
(p_reg,[HCOMP:verb],[HCOMP:verb],0)
(n_deic_pro_sg,[HCOMP:prep*],[HCOMP:verb],0)
(n_deic_pro_sg,[HCOMP:verb],[HCOMP:verb],0)
```

Bi-gram Model on only Head Projection Paths (2HeadPP)

This model has a subset of the features of Model 2PP — only those obtained by the *head path* parts of the projection paths. For our example, it contains the subset of features of 2PP that have last bit 1, which will be only the following:

```
(v_e_p,[PLAN_ON:verb],[HCOMP:verb],1)
(v_e_p,[HCOMP:verb],EOP,1)
```

Rule Tree Model I (Rule I)

The features of this model are defined by the two templates: $(leType, node, child_1, child_2, 1)$ and $(leType, node, child_1, child_2, 0)$. The last value in the tuples is an indication of whether the tuple contains the le-type of the head or the non-head child as its first element. The features containing the node [HCOMP:verb] are ones from the expansion at that node and also from the expansion of its parent:

```
(v_e_p,[HCOMP:verb],[PLAN_ON:verb],[HCOMP:prep*],1)
(p_reg,[HCOMP:verb],[PLAN_ON:verb],[HCOMP:prep*],0)
(v_sorb,[HCOMP:verb],[HCOMP:verb],[HCOMP:verb],1)
(v_e_p,[HCOMP:verb],[HCOMP:verb],[HCOMP:verb],0)
```

Model	Features	Accuracy
2PP	36,623	82.70
2HeadPP	11,490	80.14
Rule I	28,797	80.99
Rule II	16,318	81.07

Table 1: Accuracy of models using the leaf projection path and rule representations.

Rule Tree Model II (Rule II)

This model splits the features of model Rule I in two parts, to mimic the features of the projection path models. It has features from the following templates: $(leTypeHead, node, headChild, 1)$ and $(leTypeNonHead, node, nonHeadChild, 0)$. The features containing the [HCOMP:verb] node are:

```
(v_e_p,[HCOMP:verb],[PLAN_ON:verb],1)
(p_reg,[HCOMP:verb],[HCOMP:prep*],0)
(v_sorb,[HCOMP:verb],[HCOMP:verb],1)
(v_e_p,[HCOMP:verb],[HCOMP:verb],0)
```

This model has less features than model Rule I, because it splits each rule into its head and non-head parts and does not have the two parts jointly. We can note that this model has all the features of 2HeadPP, except the ones involving start and end of path, due to the first template. The second template leads to features that are not even in 2PP because they connect the head and non-head paths of a word, which are represented as separate strings in 2PP.

Overall, we can see that models Rule I and Rule II have the information used by 2HeadPP (and some more information), but do not have the information from the non-head parts of the paths in Model 2PP. Table 1 shows the average parse ranking accuracy obtained by the four models as well as the number of features used by each model. Model Rule I did not do better than model Rule II, which shows that joint representation of rule features was not very important. The large improvement of 2PP over 2HeadPP (13% error reduction) shows the usefulness of the non-head projection paths. The error reduction of 2PP over Rule I is also large – 9% error reduction. Further improvements over models using rule features were possible by considering more sophisticated string kernels and word keyed projection paths, as will be shown in the following sections.

4.2 Experimental Results using String Kernels on Projection Paths

In the present experiments, we have limited the derivation tree node annotation to the features listed in Table 2. Many other features from the HPSG signs

No.	Name	Example
0	Node Label	*HCOMP*
1	synsem.local.cat.head	*verb*
2	Label from Phrase Struct Tree	*S*
3	Le Type of Lexical Head	*v_sorb_le*
4	Lexical Head Word	*let*

Table 2: Annotated features of derivation tree nodes. The examples are from one node in the head path of the word *let* in Figure 1.

are potentially helpful for disambiguation, and incorporating more useful features is a next step for this work. However, given the size of the corpus, a single model can not usefully profit from a large number of features. Previous work (Osborne and Baldbridge, 2004; Toutanova and Manning, 2002; Toutanova et al., 2002) has explored combining multiple classifiers using different features. We report results from such an experiment as well.

Using Node Label and Head Category Annotations

The simplest derivation tree node representation that we consider consists of features 0 and 1 - schema name and category of the lexical head. All experiments in this subsection section were performed using this derivation tree annotation. We briefly mention results from the best string-kernels when using other node annotations, as well as a combination of models using different features in the following subsection.

To evaluate the usefulness of our Repetition Kernel, defined in Section 3.3, we performed several simple experiments. We compared it to a 1-gram kernel, and to a 2-gram kernel. The results – number of features per model, and accuracy, are shown in Table 3. The models shown in this table include both features from projection paths keyed by words and projection paths keyed by le-types. The results show that the Repetition kernel achieves a noticeable improvement over a 1-gram model (7.8% error reduction), with the addition of only a small number of features. For most of the words, repeated symbols will not occur in their paths, and the Repetition kernel will behave like a 1-gram for the majority of cases. The additional information it captures about repeated symbols gives a sizable improvement. The bi-gram kernel performs better but at the cost of the addition of many features. It is likely that for large alphabets and small training sets, the Repetition kernel may outperform the bi-gram kernel.

From this point on, we will fix the string kernel for projection paths keyed by words — it will be a linear combination of a bi-gram kernel and a Rep-

Kernel	Features	Accuracy
1-gram	44,278	82.21
Repetition	52,994	83.59
2-gram	104,331	84.15

Table 3: Comparison of the Repetition kernel to 1-gram and 2-gram.

etition kernel. We found that, because lexical information is sparse, going beyond 2-grams for lexically headed paths was not useful. The projection paths keyed by le-types are much less sparse, but still capture important sequence information about the syntactic frames of words of particular lexical types.

To study the usefulness of different string kernels on projection paths, we first tested models where only le-type keyed paths were represented, and then tested the performance of the better models when word keyed paths were added (with a fixed string kernel that interpolates a bi-gram and a Repetition kernel).

Table 4 shows the accuracy achieved by several string kernels as well as the number of features (in thousands) they use. As can be seen from the table, the models are very sensitive to the discount factors used. Many of the kernels that use some combination of 1-grams and possibly discontinuous bi-grams performed at approximately the same accuracy level. Such are the wildcard(2,1,λ) and subseq(2,g,λ_1,λ_2) kernels. Kernels that use 3-grams have many more parameters, and even though they can be marginally better when using le-types only, their advantage when adding word keyed paths disappears. A limited amount of discontinuity in the Subsequence kernels was useful. Overall Subsequence kernels were slightly better than Wildcard kernels. The major difference between the two kinds of kernels as we have used them here is that the Subsequence kernel unifies features that have gaps in different places, and the Wildcard kernel does not. For example, $a * b, *ab, ab*$ are different features for Wildcard, but they are the same feature ab for Subsequence – only the weighting of the feature depends on the position of the wildcard.

When projection paths keyed by words are added, the accuracy increases significantly. subseq(2,3,.5,2) achieved an accuracy of 84.96%, which is much higher than the best previously published accuracy from a single model on this corpus (82.7% for a model that incorporates more sources of information from the HPSG signs (Toutanova et al., 2002)). The error reduction compared to that model is 13.1%. It is also higher than the best result from voting classifiers (84.23% (Osborne and Baldbridge, 2004)).

Model	Features		Accuracy	
	le	w & le	le	w & le
1gram	13K	-	81.43	-
2gram	37K	141K	82.70	84.11
wildcard (2,1,.7)	62K	167K	83.17	83.86
wildcard (2,1,.25)	62K	167K	82.97	-
wildcard (3,1,.5)	187K	291K	83.21	83.59
wildcard (3,2,.5)	220K		82.90	-
subseq (2,3,.5,2)	81K	185K	83.22	**84.96**
subseq (2,3,.25,2)	81K	185K	**83.48**	84.75
subseq (2,3,.25,1)	81K	185K	82.89	-
subseq (2,4,.5,2)	102K	206K	83.29	84.40
subseq (3,3,.5,2)	154K	259K	83.17	83.85
subseq (3,4,.25,2)	290K	-	83.06	-
subseq (3,5,.25,2)	416K	-	83.06	-
combination model				**85.40**

Table 4: Accuracy of models using projection paths keyed by *le-type* or both *word* and *le-type. Numbers of features are shown in thousands.*

Other Features and Model Combination

Finally, we trained several models using different derivation tree annotations and built a model that combined the scores from these models together with the best model subseq(2,3,.5,2) from Table 4. The combined model achieved our best accuracy of 85.4%. The models combined were:

Model I A model that uses the Node Label and le-type of non-head daughter for head projection paths, and Node Label and sysnem.local.cat.head for non-head projection paths. The model used the subseq(2,3,.5,2) kernel for le-type keyed paths and bi-gram + Repetition for word keyed paths as above. Number of features of this model: 237K Accuracy: 84.41%.

Model II A model that uses, for head paths, Node Label of node and Node Label and sysnem.local.cat.head of non-head daughter, and for non-head paths PS category of node. The model uses the same kernels as Model I. Number of features: 311K. Accuracy: 82.75%.

Model III This model uses PS label and sysnem.local.cat.head for head paths, and only PS label for non-head paths. The kernels are the same as Model I. Number of features: 165K Accuracy: 81.91%.

Model IV This is a standard model based on rule features for local trees, with 2 levels of grandparent annotation and back-off. The annotation used at nodes was with Node Label and sysnem.local.cat.head. Number of features: 78K Accuracy: 82.6%.

5 Conclusions

We proposed a new representation of parse trees that allows us to connect more tightly tree structures to the words of the sentence. Additionally this representation allows for the natural extension of string kernels to kernels on trees. The major source of accuracy improvement for our models was this representation, as even with bi-gram features, the performance was higher than previously achieved. We were able to improve on these results by using more sophisticated Subsequence kernels and by our Repetition kernel which captures some salient properties of word projection paths.

In future work, we aim to explore the definition of new string kernels that are more suitable for this particular application and apply these ideas to Penn Treebank parse trees. We also plan to explore annotation with more features from HPSG signs.

Acknowledgements

We would like to thank the anonymous reviewers for helpful comments. This work was carried out under the Edinburgh-Stanford Link programme, funded by Scottish Enterprise, ROSIE project R36763.

References

Rens Bod. 1998. *Beyond Grammar: An Experience Based Theory of Language*. CSLI Publications.

Eugene Charniak. 2000. A maximum entropy inspired parser. In *Proceedings of NAACL*, pages 132 – 139.

Michael Collins and Nigel Duffy. 2001. Convolution kernels for natural language. In *Proceedings of NIPS*.

Michael Collins. 1997. Three generative, lexicalised models for statistical parsing. In *Proceedings of the ACL*, pages 16 – 23.

Michael Collins. 2000. Discriminative reranking for natural language parsing. In *Proceedings of ICML*, pages 175–182.

Michael Collins. 2001. Parameter estimation for statistical parsing models: Theory and practice of distribution-free methods. In *IWPT*. Paper written to accompany invited talk at IWPT 2001.

Thomas Gaertner, John W. Lloyd, and Peter A. Flach. 2002. Kernels for structured data. In *ILP02*, pages 66–83.

David Haussler. 1999. Convolution kernels on discrete structures. In *UC Santa Cruz Technical Report UCS-CRL-99-10*.

Thorsten Joachims. 1999. Making large-scale SVM learning practical. In B. Scholkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*.

Christina Leslie and Rui Kuang. 2003. Fast kernels for inexact string matching. In *COLT 2003*, pages 114–128.

Huma Lodhi, John Shawe-Taylor, Nello Cristianini, and Christopher J. C. H. Watkins. 2000. Text classification using string kernels. In *Proceedings of NIPS*, pages 563–569.

Stephan Oepen, Kristina Toutanova, Stuart Shieber, Chris Manning, and Dan Flickinger. 2002. The LinGo Redwoods treebank: Motivation and preliminary apllications. In *Proceedings of COLING 19*, pages 1253—1257.

Miles Osborne and Jason Baldbridge. 2004. Ensemble-based active learning for parse selection. In *Proceedings of HLT-NAACL*.

Carl Pollard and Ivan A. Sag. 1994. *Head-Driven Phrase Structure Grammar*. University of Chicago Press.

Stefan Riezler, Detlef Prescher, Jonas Kuhn, and Mark Johnson. 2000. Lexicalized stochastic modeling of constraint-based grammars using log-linear measures and EM training. In *Proceedings of the ACL*, pages 480—487.

Libin Shen and Aravind K. Joshi. 2003. An SVM-based voting algorithm with application to parse reranking. In *Proceedings of CoNLL*, pages 9–16.

Jun Suzuki, Tsutomu Hirao, Yutaka Sasaki, and Eisaku Maeda. 2003. Hierarchical directed acyclic graph kernel: Methods for structured natural language data. In *Proceedings of the ACL*, pages 32 – 39.

Kristina Toutanova and Christopher D. Manning. 2002. Feature selection for a rich HPSG grammar using decision trees. In *Proceedings of CoNLL*.

Kristina Toutanova, Christopher D. Manning, Stuart Shieber, Dan Flickinger, and Stephan Oepen. 2002. Parse disambiguation for a rich HPSG grammar. In *Proceedings of Treebanks and Linguistic Theories*, pages 253–263.

Vladimir Vapnik. 1998. *Statistical Learning Theory*. Wiley, New York.