Fast Linearization of Tree Kernels over Large-Scale Data

Aliaksei Severyn[1]
[1]University of Trento, DISI
38123 Povo (TN), Italy
severyn@disi.unitn.it

Alessandro Moschitti[1,2]
[2]Qatar Computing Research Institute
Qatar Foundation, Doha, Qatar
amoschitti@qf.org.qa

Abstract

Convolution tree kernels have been successfully applied to many language processing tasks for achieving state-of-the-art accuracy. Unfortunately, higher computational complexity of learning with kernels w.r.t. using explicit feature vectors makes them less attractive for large-scale data. In this paper, we study the latest approaches to solve such problems ranging from feature hashing to reverse kernel engineering and approximate cutting plane training with model compression. We derive a novel method that relies on reverse-kernel engineering together with an efficient kernel learning method. The approach gives the advantage of using tree kernels to automatically generate rich structured feature spaces and working in the linear space where learning and testing is fast. We experimented with training sets up to 4 million examples from Semantic Role Labeling. The results show that (i) the choice of correct structural features is essential and (ii) we can speed-up training from weeks to less than 20 minutes.

1 Introduction

Structural kernels are an important aspect of applied machine learning, since rich feature sets can be generated automatically, thus alleviating the need for tedious manual feature engineering. They have been successfully used in many NLP tasks, e.g. [Kate and Mooney, 2006; Daumé III and Marcu, 2004]. According to the current research in kernel methods, the optimization can be carried out either in the dual space, where the scaling is in the order of $O(n^2)$, or (similar to linear-time algorithms with linear kernel) can also be performed in the primal [Chapelle, 2007], which, in turn, requires to precompute and keep the entire Gram matrix in the memory. Both methods pose computational problems for applying kernels on datasets with millions of examples. Consequently, several approaches that attempt to trade-off accuracy for speed have been proposed: (i) linearization of the kernel space, i.e., extraction of the most important features to build explicit feature vectors, e.g., [Kudo and Matsumoto, 2003; Cumby and Roth, 2003]. (ii) Feature selection in kernel spaces [Kudo *et al.*, 2005], where the implicit computation of feature weights allows one to avoid full computation of the kernel functions; (iii) hybrid approaches mixing kernel computations with on-line linearization of trees [Kazama and Torisawa, 2005]; (iv) feature hashing [Shi *et al.*, 2009; Ganchev and Dredze, 2008], which can highly decrease the size of the linearized space; (v) linearization using reverse-kernel engineering (RKE) approach [Pighin and Moschitti, 2009], which exploits SVM models to extract the most important features from the kernel space; and (vi) recent advances in fast SVM learning with structural kernels [Severyn and Moschitti, 2011] (SDAG), which, thanks to the approximate cutting plane algorithm and its model compression via directed acyclic graphs, provides significant speedups over conventional SVM training methods. While kernel-based learning have been shown to deliver more accurate models for complex NLP tasks, none of the methods above seem to be optimal when dealing with large-scale datasets without compromising the accuracy of their exact counterparts.

In this paper, we study the latest approaches to large-scale learning with convolution tree kernels (TKs) by deriving the following findings: first, it is computationally prohibitive to naïvely enumerate all structural features, which is often mandatory for tackling high-level semantic tasks such as the extraction of predicate argument structures in Semantic Role Labeling (SRL). Indeed, (i) enumerating all possible substructures quickly becomes intractable as their number grows exponentially with the size of the input structure; (ii) hashed feature vectors tend to be very dense; and (iii) the feature collisions due to hashing cause the model to underperform w.r.t. structural kernels.

Secondly, we show that reverse-kernel engineering is a principled way to linearize exponentially large tree kernel spaces for tasks where models encoding complex structural features yield better accuracy. However, this approach inherits the computational burden of training an SVM model, which is required by its greedy mining procedure. We attack the major computational bottleneck of this approach by (i) replacing the slow SVM training with a much faster SDAG algorithm, which produces a more compact model in a form of a directed acyclic graph (DAG); and (ii) we define a linearization approach based on fragment extraction directly from the DAG model. This approach achieves the same accuracy as the traditional SVM learning with structural kernels. We provide an explanation of such surprising result by showing that the SDAG and the SVM models are rather similar, as their most important features have a large overlap. Therefore, we can use fast algorithms to learn a tree kernel model and work in the linear space where learning and testing is fast.

Thirdly, we propose a distributed system to further speed up RKE of SDAG, which includes: (a) splitting the training set into smaller subsets and (b) using multiple CPUs to learn individual SDAG models. In particular, we show that: (i) the merged feature space can be used to learn the final linear model whose accuracy is just few tens of a point lower than that of the traditional SVMs using tree kernels; and (ii) the entire process takes less than an hour. For example, on the 4 million examples from SRL dataset, the entire linearization process takes less than 20 minutes achieving an F1 of 84.5% vs. 84.8% achieved by SDAG.

Finally, we could carry out large-scale experiments, e.g., using the entire 4 million instances of the SRL dataset (boundary detection from CoNLL 2005 [Carreras and Màrquez, 2005]), in a few hours. This does not produce any loss in accuracy w.r.t. the traditional SVMs.

2 Related Work

Linearization methods aim to convert exponentially large feature spaces implicitly generated by kernels into explicit vector spaces of a reasonable size (e.g., ranging from a few thousands to a tens of millions of features). For this purpose two important assumptions should hold: (i) it is possible to define greedy algorithms that constructively extract features working in the top-down fashion starting from the most relevant fragments and progressing to the least relevant ones; and (ii) only a small percentage of the features suffices to train accurate models.

The approach in [Kudo and Matsumoto, 2003] is based on the first hypothesis. The authors suggest that a similar method to the PrefixSpan algorithm [Pei *et al.*, 2001] can be applied to tree kernels. However, it is efficient only on a limited size of support examples. When the number of support vectors becomes very large, e.g., millions of instances, mining the most frequent substructures [Zaki, 2002] becomes slow. Additionally, the assumption of point (i), the most frequent tree is the most relevant, is problematic. Suzuki and Isozaki [2005] present a feature selection method for convolution kernels based on a distribution-driven relevance assessment. The kernel function is extended to embed substructure mining and techniques for the evaluation of a fragment's χ^2.

Another recent set of approaches is based on feature hashing, e.g. [Shi *et al.*, 2009]. It enables a large number of features to be generated and efficiently stored in feature vectors of limited size (i.e., millions of dimensions). The main idea is that features hashing to the same value will contribute to the same component of a feature vector. The resulting information loss due to the collisions is supposed to be backed up by the assumption in point (ii).

A rather comprehensive overview of feature selection techniques is carried out in [Guyon and Elisseeff, 2003]. However, most of them cannot be applied to the large-scale learning in implicit kernel spaces.

3 Learning with Structural Kernels

A Tree Kernel (TK) function is a convolution kernel [Haussler, 1999] defined over pairs of trees. Given two trees it evaluates the number of substructures (or *fragments*) they have in common, i.e., it is a measure of their overlap. The function can be computed recursively in closed form and its efficient implementation is available [Moschitti, 2006]. Different TK functions are characterized by alternative fragment definitions, e.g. [Collins and Duffy, 2002b]. In this paper we focus on the Syntactic Tree kernel (STK) described in [Collins and Duffy, 2002b], since this kernel is particularly suitable for extracting relevant features from the syntactic parse trees. For each node in a given a tree it generates all possible substructures rooted at that node with the constraint that production rules can not be broken (i.e. any node in a fragment must include either all or none of its children).

Implicitly, a TK function establishes a correspondence between distinct tree fragments and dimensions in some *fragment space*, i.e., the space of all the possible tree fragments. To simplify, a tree t can be represented as a vector whose attributes count the occurrences of each fragment within the tree. The kernel between two trees is then equivalent to the scalar product between pairs of such vectors.

3.1 Kernel Machines

Kernelized SVMs learn a decision function which is defined by the inner product between the weight vector of the learned model and a test example:

$$f(\vec{x}) = \sum_{i=1}^{n} \alpha_i y_i \phi(\vec{x_i}) \cdot \phi(\vec{x}) = \sum_{i=1}^{n} \alpha_i y_i K(\vec{x_i}, \vec{x}) \quad (1)$$

where $\phi(\cdot)$ is a feature mapping defining a kernel function K. The major bottleneck in the application of SVMs with kernels to large data stems from the necessity to carry out learning in the dual space, which makes the learning time scale quadratically with the number of training examples.

3.2 Faster SVMs with Structural Kernels

Recently, a number of efficient cutting plane algorithms have been proposed [Joachims, 2006; Franc and Sonnenburg, 2008]. Unfortunately, these algorithms scale well only when linear kernels are used. To address slow learning with nonlinear kernels [Yu and Joachims, 2008] proposed to use sampling to reduce the number of kernel evaluations. [Severyn and Moschitti, 2011] showed that same algorithm can be successfully applied to SVM learning with structural kernels on very large data obtaining speedup factors up to 10 over conventional SVMs. Furthermore, the approach uses Directed Acyclic Graphs (DAGs) to efficiently represent a set of trees by including only the unique subtrees and accounting for the frequency of the repeated substructures. It is shown that when using DAGs, evaluating the decision function in Eq. 1 is reduced to a single kernel evaluation:

$$f(\vec{x}) = \sum_{i=1}^{n} \alpha_i y_i K(\vec{x_i}, \vec{x}) = K_{dag}(\vec{dag}, \vec{x}_i) \quad (2)$$

where $\vec{dag}$ is an equivalent representation of the learned model. It compactly encodes a collection of support vectors in the model, s.t. repeating sub-structures are uniquely represented in the DAG.

In this paper, we make use of the DAG approach to show that SVMs with structural kernels, e.g., tree kernels, can be efficiently trained on large data.

4 Linearization

4.1 Feature enumeration aka feature hashing

Recently, much attention has been drawn to feature hashing approaches, e.g. [Shi *et al.*, 2009; Ganchev and Dredze, 2008], that aim at transforming high dimensional kernel spaces into the linear space, where fast learning methods can be applied. The approach forces multiple features to collide, thus achieving drastic reduction in the effective dimension of the feature space. Still, such approaches may introduce several problems: (i) enumerating a huge number (exponential in the input size) of substructures encoded by the structural kernels may become intractable; and (ii) the noise introduced by feature collisions[1] may have a negative effect on problems where few complex structural features are essential for the learning system.

4.2 Reverse Kernel Engineering

A more principled feature extraction algorithm for Tree Kernel (TK) spaces has been proposed in [Pighin and Moschitti, 2009]. Its greedy mining algorithm relies on the model learned by an SVM to extract the most relevant features (tree fragments).

Different from the naïve feature enumeration method, the support vectors of the model along with their weights are used to guide the greedy mining to extract the most relevant tree fragments. In particular, the greedy mining approach extracts the fragments f_j from support vectors (trees) t_i based on the relevance score defined as follows:

$$\sum_{i=1}^{n} \alpha_i y_i t_{i,j} \lambda^{l(f_j)} / ||t_i|| \qquad (3)$$

where α_i are the Lagrange multipliers of the learned model, λ - kernel decay factor, $l(f_j)$ - depth of the fragment f_j, y_i - example label and $t_{i,j}$ is the frequency of the fragment j in the t_i.

While originally proposed to extract features from SVs of the SVM model, our Alg. 1 presents a simplified version of the greedy mining algorithm in [Pighin and Moschitti, 2009] that works directly on a DAG model. Each node in the DAG is associated with a weight defined as follows: $\nu_i = \alpha_i y_i f_i / ||t_i||$. This definition of the node weights is convenient to simplify the mining procedure, since all the components to compute the relevance weights from Eq. 3 are already provided by the DAG. We iterate over all nodes in the DAG calling the function $frag(n)$, which generates the smallest fragment rooted at node n. These base fragments are further expanded by calling the function $expand(f)$ which enlarges the current fragment by adding direct children to some of its nodes. This process is repeated until the index is unchanged and the top K features are returned.

[1]This poses little to no penalty for some tasks where less complex features suffice to build accurate models.

Algorithm 1 mineDAG($\vec{dag}$, K)

```
 1: for all $\langle \nu, t \rangle \in \vec{dag}$ do
 2:     for all $n \in \mathcal{N}_t$ do
 3:         $f \leftarrow frag(n); w = \lambda * \nu$
 4:         $updateIndex(\langle f, w \rangle)$
 5: while $Changed(Index)$ do
 6:     for all $\langle f, w \rangle \in Index$ do
 7:         for all $\chi \in expand(f)$ do
 8:             $updateIndex(\langle \chi, \lambda * w \rangle)$
 9:     $\mathcal{F}_K \leftarrow top(K)$
10: return $\mathcal{F}_K$
```

The information about the most relevant fragments stored in the index is then used to linearize both training and test datasets. Each tree in the input data can the be explicitly represented by a feature vector, where each attribute corresponds to one of the tree fragments as generated by TK.

5 Experiments

To assess the efficacy of the proposed approach, we focus on the task of Semantic Role Labeling. The choice of SRL task is motivated by the following: (i) it is currently a well-defined task with a substantial amount of work and comparative evaluation, (ii) it represents a complex NLP problem with enough room for improvement w.r.t. to the current state-of-the-art systems, and (iii) large amount of training data is available to learn more accurate models.

5.1 Data and task

The dataset consists of the Penn Treebank texts [Marcus *et al.*, 1993], PropBank annotation [Palmer *et al.*, 2005] and Charniak parse trees [Charniak, 2000] as provided by the CoNLL 2005 shared task on Semantic Role Labeling [Carreras and Màrquez, 2005]. The goal is to recognize semantic roles of the target verbs in a given sentence. SRL is a complex tasks where the state-of-the-art systems achieve F1 at about 80% in CoNLL-2005 shared task [Carreras and Màrquez, 2005], which indicates the importance of extracting the best features. A common approach to tackle SRL problem involves two steps: (i) detection of the verb arguments and (ii) classification of identified arguments into their respective semantic categories. In this paper, we focus on the first task of argument identification (i.e., identification of the exact sequence of words spanning an argument). This corresponds to the binary classification of parse tree nodes into correct or not correct boundaries. The models are trained on the subsets {250k, 500k, 1mil, and 4mil}. To evaluate the learned models we report the F1[2] on two sections: 23 and 24 (used as development and test sets in CoNLL-2005 task, respectively), that contain 230k and 150k examples respectively.

5.2 Setup

For all our experiments we used Syntactic Tree Kernel (STK) [Collins and Duffy, 2002a]. Learning in the linear input space is carried out with LibLinear solver [Fan *et al.*,

[2]The reported scores correspond to the accuracy of the binary classifier, which upper bounds than the accuracy of the overall boundary detection due to errors in parsing.

2008]. Models that use tree kernels and their combination with feature vectors are trained by SDAG software[3] [Severyn and Moschitti, 2011], which couples approximate cutting plane algorithm and the DAG approach. It has an additional parameter q that controls the number of examples used to approximate cuts at each iteration. We fix q at {1k, 5k, and 10k} for training on {500k, 1mil, and 4mil} subsets of data, respectively.

To contrast the training time and accuracy obtained by SDAG, we use the SVM-Light-TK[4] [Moschitti, 2006] (henceforth, referred to as SVM), which encodes tree kernels into the SVM-Light solver [Joachims, 1999]. To linearize the kernel space using the learned SVM model we use reverse-kernel engineering framework FLINK[5] [Pighin and Moschitti, 2009]. For the kernel space mining procedure we set the minimum fragment frequency to 3 and the threshold factor to 1000.

5.3 Results

Feature-based learning. Manually designing useful feature sets for complex tasks such as SRL is a highly nontrivial task. Most of the feature-based learning systems for SRL rely on the linear features extracted from the syntactic parse trees as described in [Gildea and Jurafsky, 2002], which stresses the necessity and importance of syntactic parsing to derive the best features. We use these manually derived features (MF) as a preliminary baseline to compare and augment more feature-rich TK models.

As an alternative to manually designed features, we first study a straightforward approach to linearize the input space by simply enumerating all the STK features (henceforth, referred to as ETK). It is important to note that we work on already preselected parts of the parse trees, i.e., AST subtrees that by design contain predicate and target argument nodes. This is an essential pre-filtering step for only selecting the most relevant parts of the full parse trees. Features are generated by considering all STK fragments rooted at each node of a tree up to a given depth d, s.t. each fragment contains a given node as a root and all its descendants within the depth d. The number of generated fragments increases exponentially with d, hence to avoid long pre-processing times required to enumerate all possible STK features (which can be billions) we consider fragments up to $d = 5$. For example, for a tree with just 80 nodes for $d \in \{2, 3, 4, 5\}$ the number of generated STK features is respectively {340, 20k, 350k, 16mil}. Hence, to further speed up the feature generation for values of $d > 3$ we limit the maximum number of features each node in a tree can generate to 10k. To get a numeric representation of each tree fragment we hash its string representation and project the obtained numeric value into the linear feature space of the dimensionality 2^k. The obtained linear feature vectors are further normalized. We performed 5-fold cross-validation with different values of $k \in \{16, 18, 20, 22\}$ and found that $k = 20$ (which encodes up to 1 million features) to yield the most accurate results. Note that feature hashing approaches, e.g. [Shi *et al.*, 2009], provide an efficient way to reduce the dimensionality of the resulting feature vector, while enumerating an exponential number of all tree fragments is infeasible. In the next section we discuss a principled approach to greatly reduce the space of considered tree fragments by relying on the relevance weights derived from a pre-trained SVM model.

d	250k		500k		1mil	
	P/R	F1	P/R	F1	P/R	F1
2	82.7/70.9	76.4	83.2/70.7	76.5	83.0/77.1	79.9
3	83.5/67.4	74.6	83.6/72.6	77.7	83.5/75.0	79.0
4	84.0/57.5	68.2	83.9/63.9	72.5	83.8/68.2	75.2
5	84.9/53.5	65.6	84.1/59.6	69.8	83.7/63.6	72.3

Table 1: Accuracy of LibLinear models for ETK model on Section 24 of the SRL dataset.

Table 1 presents the results of training using ETK models on datasets containing {250k, 500k, and 1mil} examples w.r.t. the maximum depth of the generated fragments. As we can see, the recall of the classifiers goes down with larger values of d, hence negatively affecting the F1 score. This demonstrates the drawback of the feature hashing approach failing to deal with exponential feature spaces where few relevant features may get lost due to the hash collisions. Similar findings for SRL were reported in [Kudo *et al.*, 2005].

Interestingly, as demonstrated by the learning curves in Fig. 1, the features extracted by enumerating TK fragments (LibLinear$_{\text{ETK}}$)[6] still greatly outperforms carefully designed manual features (LibLinear$_{\text{MF}}$) used in many state-of-the-art SRL systems. Nevertheless, much better accuracy obtained by TK learning with SVMs (SVM$_{\text{TK}}$) is a strong indication of the high discriminative power of structural features. Unfortunately, quadratic scaling behavior of conventional algorithms to train kernelized SVMs prevented us to carry out experiments on data larger than 1 million.

In the next experiment we use SDAG to make the training with tree kernels tractable on large data. SDAG achieves its speedup due to its faster approximate cutting plane algorithm and model compression where trees are kept in an equivalent DAG.

Learning with kernels. While widely applied across many areas of NLP, purely feature-based methods are less expressive in modeling structured features, which have been shown important in complex NLP tasks such as SRL. Although learning with kernels allows for training models with higher discriminative power, it requires much larger training times. First, we contrast the accuracy obtained by learning with kernels w.r.t. to feature-based models presented previously.

Fig. 1 shows a learning curve for various models when tested on Section 23 (Section 24 demonstrated similar behavior). We first observe that the baseline MF model, indeed, is the least accurate. A better accuracy can be achieved by applying polynomial kernel (of degree 3), but this incurs considerably larger running times due to learning in kernel spaces. As confirmed before, the ETK approach works much better than MF, but still is unable to match the accuracy of learning with tree kernels. Importantly, TK models trained by SDAG

[3]We modified the software available at `http://disi.unitn.it/~severyn/code.html`

[4]http://disi.unitn.it/moschitti/Tree-Kernel.htm

[5]http://danielepighin.net/cms/software/flink

[6]limited to the maximum depth 2

match the accuracy of SVM. Furthermore, using a combination of TKs with linear or polynomial kernels applied to MF gives the highest F1 of 85.9% (when trained on 4 mil).

Another interesting dimension to compare TK models is to look at the overlap of the most relevant tree fragments extracted by each model. We sort the fragments extracted by each model according to the relevance score (Eq. (3)). Table 2 reports the overlap percentage of the top $k \in \{1\text{k}, 5\text{k}, 10\text{k}\}$ features. Features extracted by SVM serve as the baseline to compare other models. We also include an experiment with perceptron model, which shows that less accurate training algorithm extracts different features. Interestingly, SDAG, a completely different learning algorithm, has 90% feature overlap with SVM for the top 1k features. This is especially surprising considering that the size of the feature space generated with feature enumeration (depth=3) is about 5.5 mil.

k	SDAG	ETK	Perceptron
1,000	90%	17%	69%
5,000	80%	8%	50%
10,000	77%	5%	43%

Table 2: Feature overlap for the top k features w.r.t. SVM when trained on 250k instances of SRL.

Next, we consider the training times for learning with kernels. Linear models MF and ETK can be trained in linear time and took less than 5 minutes (for the latter we need to account for the time to extract TK features, which is about 1 hour with $d = 2$). The expressive power of kernelized models comes at a cost of much larger training times. However, as Table 3 shows training with SDAG algorithm (using 4 CPUs) makes learning with TK and combinations with feature vectors tractable even on 4 million, which is prohibitively expensive for SVM.

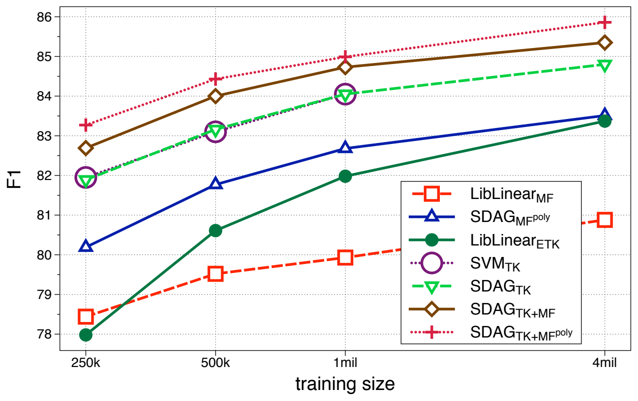

Figure 1: Learning curves for models tested on sec23. MF - manual feature vectors, poly - polynomial kernel of degree 3; ETK - enumeration of TKs (depth 2); TK+MF/MF^{poly} combination of TK with linear or polynomial kernel applied to MF. Linear feature models are learned with LibLinear, while TK models are trained with SDAG.

Model	**Time (hours)**	
	1mil	**4mil**
SDAG$_{MF^{poly}}$	7.1	13.1
SVM$_{TK}$	84.1	-
SDAG$_{TK}$	4.4	7.2
SDAG$_{TK+MF}$	4.9	8.1
SDAG$_{TK+MF^{poly}}$	9.4	15.3

Table 3: Runtimes for kernelized models.

Tree Kernel Linearization. As shown above, fast SDAG algorithm allows for efficient learning of TK models on the dataset of 4 million examples, which is impossible for conventional SVM training algorithms. Nevertheless, the training time for SDAG is still in the range of hours even when run on multiple CPUs. Hence, even using very fast approaches to learning in kernel spaces, the obtained runtimes are far from matching the speed of learning in the linear spaces with linear-time solvers such as LibLinear.

The first step towards reaching the speed of linear-time SVM solvers is to linearize the input structures. Once trees are converted into explicit feature vectors training and testing becomes linear-time. Hence, we require an efficient yet accurate method to perform linearization of TK spaces. Previous experiments with ETK have shown that applying a rather naïve technique to enumerate an exponential number of tree fragments generated by STK suffers from a large drop of accuracy, thus unable to encode complex structural features essential for the task. Moreover the pre-processing time to transform trees to explicit feature vectors exhibits exponential growth with the number of nodes in a tree and the depth of the generated tree fragments.

A more principled approach to linearization of tree kernel spaces, as described in Sec. 3.1, is to apply reverse kernel (RKE) engineering where an SVM model is used as a source of fragment weights to guide the greedy feature extraction. Hence, in this experiment our goal is to assess the accuracy of this method when linearizing TK spaces. In the following experiment we first verify that indeed, linearizing the input feature spaces generated by tree kernels does not incur substantial loss of accuracy and is in the line with the results demonstrated in [Pighin and Moschitti, 2009]. Table 4 reports the accuracy of linearized models using RKE when fed with TK models trained by SVM and SDAG. As we can see, RKE does a good job on extracting most relevant features generated by TKs when converting to the explicit feature vector representation.

While shown to be accurate, the greedy feature mining requires to train a TK model. Hence, the total runtime of the entire linearization procedure is lower-bound by the time to learn a model, which is rather large even when using SDAG. Conversely, the complexity of the successive steps in the linearization pipeline are negligibly small: mining most relevant tree fragments from the model learned on 4 million (about 250k support vectors) takes about 2 minutes. Linearizing 4 million examples took only 3 minutes. Finally, training and classification with the linearized datasets using LibLinear takes 2 minutes. Nevertheless, the bottleneck runtime imposed by learning a TK model can be overcome, which we explore in the next set of experiments.

Model	**1mil**		**4mil**	
	F1		**F1**	
	sec23	**sec24**	**sec23**	**sec24**
SVM$_{TK}$	84.1	81.8	-	-
RKE-SVM$_{TK}$	84.0	81.3	-	-
SDAG$_{TK}$	84.0	81.5	84.8	82.6
RKE-SDAG$_{TK}$	83.9	81.3	84.5	82.5

Table 4: RKE applied to TK models trained by SVM and SDAG. Linearized models are trained with LibLinear.

Distributed linearization. In the following, we take advantage of the observation (also explored in [Pighin and Moschitti, 2009]), where hefty training of the SVM model is sped up by splitting the training set into smaller subsets. Previous work in [Graf *et al.*, 2004] suggests that support vectors collected from locally learned models can encode many of the relevant features retained by models learned globally. Thus, the quadratic scaling behavior of SVMs with kernels can be conquered by carrying out learning on much smaller subsets of the data, which also allows for learning the individual models in parallel. This approach is justified by the fact that the main purpose of the SVM learning in the original kernel space is not to provide the final model that will be used for classification on the test data but is rather used to drive the feature mining process to extract the most relevant features. Hence, the sub-optimal fragment weights derived from the local models learned on the subsets of the original training set still carry enough information to extract highly discriminative features.

Furthermore, splitting the training data into smaller subsets allows for setting up a fully distributed system, e.g., MapReduce. Each subset of the input data is mapped to individual workers. Locally learned models are recombined and used in the feature extraction and further linearization of the training and test data. Finally, fast SVM solvers are used to produce the final linear model for testing.

To test the efficacy of the distributed linearization approach we split the entire dataset of 4mil into $n \in \{10, 20, 40, \text{and } 80\}$ subsets, which corresponds to learning on the subsets of size $s \in \{400\text{k}, 200\text{k}, 100\text{k and } 50\text{k}\}$ respectively. We have found that learning local models on subsets smaller than 100k lead to a small drop of the final accuracy of the linearized model, while using larger subsets did not provide any further improvement. Hence, we fix $n = 40$. Table 5 reports the running times to train a local model on a single subset of 100k and the overall time of the linearization approach (from learning a TK model to linearization and training in the linear space) on the entire dataset of 4 million. Since learning of local models is fully independent: all 40 jobs were distributed among 10 CPUs, which cut down the total time by 4x. We see that SDAG delivers much faster training times. Interestingly, by using smaller sample sizes to approximate the cutting planes inside the SDAG algorithm, we obtain the same final accuracy while reducing the training time to only

Model	**Time (min)**	**F1**	
		sec23	**sec24**
SVM$_{TK}$	57 / 233	84.2	82.2
SDAG$_{TK}$ (q=250)	3 / 17	84.4	82.4
SDAG$_{TK}$ (q=500)	7 / 33	84.4	82.4
SDAG$_{TK}$ (q=1000)	11 / 49	84.5	82.5

Table 5: RKE with local models on 4 mil of SRL ($n = 40$ subsets). First value in the 2nd column indicates the time to learn a single model of 100k, while the second value is the overall time for 4 mil (from learning a TK model to linearization and training in the linear space).

3 minutes for learning one local model.

Hence, the proposed approach achieves remarkably low runtime of only 17 minutes for the full set of activities required by linearization process on the entire 4 million dataset.

6 Conclusions

In this paper, we experimented with learning in linear spaces using manual features, linearized kernel spaces through hashing methods, reverse kernel engineering and approximate cutting plane training with DAG model compression.

Our findings reveal that on a high-level semantic task such as SRL: (i) the naïve approach of enumerating all possible sub-structures becomes intractable and hashed feature vectors fail to achieve both the same accuracy of tree kernels and high efficiency. (ii) In contrast, SDAG allows for achieving the same accuracy as SVMs and makes learning practical. However, the classification and learning time may still not be appealing for large-scale experiments. (iii) Linearization with RKE is rather effective as again there is almost no loss in accuracy and it benefits from extracting complex and highly discriminative features derived from learning in kernel spaces.

As a result, we derive an efficient approach to kernel learning: applying reverse-kernel engineering directly on the SDAG model. This alleviates the major computational bottleneck of the original approach, where traditional SVM training was used. Interestingly, the extracted features have high overlap with the baseline SVM. Additionally, we achieve a significant speedup with almost no loss in accuracy by splitting the data into smaller subsets. This allows for more efficient kernel space learning in a fully distributed manner.

Summing up, we can train an accurate tree kernel model on 4 million instances from SRL, in less than 20 minutes using 10 CPUs. We achieved F1 of 84.5% on Section 23, which is the state-of-the-art performance of the binary classifier for boundary detection without using ensembles of learners and relying only one a single source of the syntactic information from the parse trees.

7 Acknowledgements

This research has been supported by the European Community's Seventh Framework Program (FP7/2007-2013) under the #288024 LiMoSINe project.

References

[Carreras and Màrquez, 2005] Xavier Carreras and Lluís Màrquez. Introduction to the CoNLL-2005 Shared Task: Semantic Role Labeling. In *CoNLL*, 2005.

[Chapelle, 2007] Olivier Chapelle. Training a support vector machine in the primal. *Neural Comput.*, 19(5):1155–1178, May 2007.

[Charniak, 2000] Eugene Charniak. A maximum-entropy-inspired parser. In *ANLP*, pages 132–139, 2000.

[Collins and Duffy, 2002a] M. Collins and N. Duffy. New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In *ACL*, Philadelphia, PA, USA, 2002.

[Collins and Duffy, 2002b] Michael Collins and Nigel Duffy. New Ranking Algorithms for Parsing and Tagging: Kernels over Discrete Structures, and the Voted Perceptron. In *ACL*, 2002.

[Cumby and Roth, 2003] Chad Cumby and Dan Roth. Kernel Methods for Relational Learning. In *ICML*, 2003.

[Daumé III and Marcu, 2004] Hal Daumé III and Daniel Marcu. A tree-position kernel for document compression. In *DUC*, 2004.

[Fan *et al.*, 2008] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. Liblinear: A library for large linear classification. *JMLR*, 9:1871–1874, June 2008.

[Franc and Sonnenburg, 2008] Vojtech Franc and Sören Sonnenburg. Optimized cutting plane algorithm for support vector machines. In *ICML*, pages 320–327, 2008.

[Ganchev and Dredze, 2008] Kuzman Ganchev and Mark Dredze. Small statistical models by random feature mixing. In *In workshop on Mobile NLP at ACL*, 2008.

[Gildea and Jurafsky, 2002] Daniel Gildea and Daniel Jurafsky. Automatic labeling of semantic roles. *Computational Linguistics*, 28:245–288, 2002.

[Graf *et al.*, 2004] Hans Peter Graf, Eric Cosatto, Leon Bottou, Igor Durdanovic, and Vladimir Vapnik. Parallel support vector machines: The cascade svm. In *NIPS*, 2004.

[Guyon and Elisseeff, 2003] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003.

[Haussler, 1999] David Haussler. Convolution kernels on discrete structures. Technical report, Dept. of Computer Science, University of California at Santa Cruz, 1999.

[Joachims, 1999] T. Joachims. Making large-scale SVM learning practical. In *Advances in Kernel Methods - Support Vector Learning*, chapter 11, pages 169–184. MIT Press, Cambridge, MA, 1999.

[Joachims, 2006] T. Joachims. Training linear SVMs in linear time. In *KDD*, 2006.

[Kate and Mooney, 2006] Rohit J. Kate and Raymond J. Mooney. Using string-kernels for learning semantic parsers. In *ACL*, July 2006.

[Kazama and Torisawa, 2005] Jun'ichi Kazama and Kentaro Torisawa. Speeding up training with tree kernels for node relation labeling. In *EMNLP*, 2005.

[Kudo and Matsumoto, 2003] Taku Kudo and Yuji Matsumoto. Fast methods for kernel-based text analysis. In *ACL*, 2003.

[Kudo *et al.*, 2005] Taku Kudo, Jun Suzuki, and Hideki Isozaki. Boosting-based parse reranking with subtree features. In *ACL*, 2005.

[Marcus *et al.*, 1993] Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1993.

[Moschitti, 2006] Alessandro Moschitti. Making tree kernels practical for natural language learning. In *EACL*, Trento, Italy, 2006.

[Palmer *et al.*, 2005] Martha Palmer, Paul Kingsbury, and Daniel Gildea. The proposition bank: An annotated corpus of semantic roles. *Computational Linguistics*, 31(1):71–106, 2005.

[Pei *et al.*, 2001] J. Pei, J. Han, Mortazavi B. Asl, H. Pinto, Q. Chen, U. Dayal, and M. C. Hsu. PrefixSpan Mining Sequential Patterns Efficiently by Prefix Projected Pattern Growth. In *ICDE*, 2001.

[Pighin and Moschitti, 2009] Daniele Pighin and Alessandro Moschitti. Efficient linearization of tree kernel functions. In *CONLL*. ACL, 2009.

[Severyn and Moschitti, 2011] A. Severyn and A. Moschitti. Fast support vector machines for structural kernels. In *ECML/PKDD*, 2011.

[Shi *et al.*, 2009] Qinfeng Shi, James Petterson, Gideon Dror, John Langford, Alex Smola, and S.V.N. Vishwanathan. Hash kernels for structured data. *Journal of Machine Learning Research*, 10:2615–2637, December 2009.

[Suzuki and Isozaki, 2005] Jun Suzuki and Hideki Isozaki. Sequence and Tree Kernels with Statistical Feature Mining. In *NIPS*, 2005.

[Yu and Joachims, 2008] Chun-Nam John Yu and T. Joachims. Training structural svms with kernels using sampled cuts. In *KDD*, 2008.

[Zaki, 2002] Mohammed J. Zaki. Efficiently mining frequent trees in a forest. In *SIGKDD*, 2002.