Efficient Online Learning for Mapping Kernels on Linguistic Structures

Giovanni Da San Martino
Qatar Computing Research Institute,
Hamad Bin Khalifa University
Doha, Qatar
gmartino@hbku.edu.qa

Alessandro Sperduti, Fabio Aiolli
Department of Mathematics,
University of Padova
via Trieste, 63, Padova, Italy
{sperduti, aiolli}@math.unipd.it

Alessandro Moschitti
Amazon
Manhattan Beach, CA, USA
amosch@amazon.com

Abstract

Kernel methods are popular and effective techniques for learning on structured data, such as trees and graphs. One of their major drawbacks is the computational cost related to making a prediction on an example, which manifests in the classification phase for batch kernel methods, and especially in online learning algorithms. In this paper, we analyze how to speed up the prediction when the kernel function is an instance of the Mapping Kernels, a general framework for specifying kernels for structured data which extends the popular convolution kernel framework. We theoretically study the general model, derive various optimization strategies and show how to apply them to popular kernels for structured data. Additionally, we derive a reliable empirical evidence on semantic role labeling task, which is a natural language classification task, highly dependent on syntactic trees. The results show that our faster approach can clearly improve on standard kernel-based SVMs, which cannot run on very large datasets.

1 Introduction

Many data mining applications involve the processing of structured or semi-structured objects, e.g. proteins and phylogenetic trees in Bioinformatics, molecular graphs in Chemistry, hypertextual and XML documents in Information Retrieval and parse trees in NLP. In all these areas, the huge amount of available data jointly with a poor understanding of the processes generating them, typically enforces the use of machine learning and/or data mining techniques.

The main complexity on applying machine learning algorithms to structured data resides in the design of effective features for their representation. Kernel methods seem a valid approach to alleviate such complexity since they allow to inject background knowledge into a learning algorithm and provide an implicit object representation with the possibility to work implicitly in very large feature spaces. These interesting properties have triggered a lot of research on kernel methods for structured data (Jaakkola, Diekhans, and Haussler 2000; Haussler 1999) and kernels for Bioinformatics (Kuang et al. 2004). In particular, tree kernels have shown to be very effective for NLP tasks, e.g., parse re-ranking (Collins and Duffy 2002), Semantic Role Labeling (Kazama and Torisawa 2005; Zhang et al. 2007), Entailment Recognition (Zanzotto and Moschitti 2006), paraphrase detection (Filice, Da San Martino, and Moschitti 2015), computational argumentation (Wachsmuth et al. 2017).

One drawback of using tree kernels (and kernels for structures in general) is the time complexity required both in learning and classification. Such complexity can sometimes prevent the kernel application in scenarios involving large amount of data. Typical approaches devoted to the solution of this problem relate to: (1) the use of fast learning/classification algorithms, e.g. the (voted) perceptron (Collins and Duffy 2002); (2) the reduction of the computational time required for computing a kernel (see e.g. (Shawe-Taylor and Cristianini 2004) and references therein).

The first approach is viable since the use of a large number of training examples can fill the accuracy gap between fast *on-line algorithms*, which do not explicitly maximize margin, and more expensive algorithms like Support Vector Machines (SVMs), which have to solve a quadratic optimization problem. Therefore, although the latter is supported by a solid theory and state-of-the-art accuracy on small datasets, learning approaches able to work with much larger data can outperform them.

The second approach relates to either the design of more efficient algorithms based on more appropriate data structures or the computation of kernels over set of trees, e.g., (Kazama and Torisawa 2006), (Moschitti and Zanzotto 2007). The latter techniques allow for the factorization/reuse of previously evaluated subparts leading to a faster overall processing.

Our proposal is in line with the second approach. However, it will be applicable to both online and batch learning algorithms. One of the most popular approaches to the definition of kernel functions for structured data is the convolution kernel framework (Haussler 1999). A convolution kernel can be defined by decomposing a structured object into substructures and then summing the contributions of kernel evaluations on the substructures. We exploit the idea that identical substructures appearing in different kernel evaluations can be factorized and thus computed only once, by postulating a necessary condition of optimality for a data structure and deriving results from it on an extension of the convolution kernel framework, the Mapping Kernels (Shin and Kuboyama 2010). Besides the general theoretical result, we show how to apply our results to a number of popular kernels for trees. Moreover, we give empirical evidence that such optimal struc-

tures yield in practice a significant speed up and reduction of memory consumption for the learning algorithms for the task of Semantic Role Labelling: we show that the learning time for a perceptron on a 4-million-instances dataset reduces from more than a week to 14 hours only. Since an SVM could only be trained on a significantly smaller number of examples, a simple voted perceptron significantly outperforms an SVM also with respect to classification performances.

2 Notation

A tree T is a directed and connected graph without cycles for which every node has one incoming edge, except a single node, i.e. the root, with no incoming edges. A leaf is a node with no outgoing edges. A descendant of a node v is any node connected to v by a path. A production at a node v is the tree composed by v and its direct children. A partial tree (PT) t is a subset of nodes in the tree T, with corresponding edges, which forms a tree (Moschitti 2006). A proper subtree (ST) rooted at a node t comprises t and all its descendants (Viswanathan and Smola 2003). A subset tree (SST) of a tree T is a structure which: *i)* is rooted in a node of T; *ii)* satisfies the constraint that each of its nodes contains either all children of the original tree or none of them (Collins and Duffy 2002). Finally, an elastic tree (ET) only requires the nodes of the subtree to preserve the relative positioning in the original tree (Kashima and Koyanagi 2002). This allows, for example, to connect a node with any of its descendants even if there is no direct edge in the original tree.

3 Background

Kernelized Perceptron

Kernel methods rely on kernel functions, which are symmetric positive semidefinite similarity functions defined on pairs of input examples. The kernelized version (Kivinen, Smola, and Williamson 2004) of the popular online learning perceptron algorithm (Rosemblatt 1958), can be described as follows. Let X be a stream of example pairs (x_i, y_i), $y_i \in \{-1, +1\}$, then the prediction of the perceptron for a new example x is the sign of the score function: $S(x) = \sum_{i=1}^{n-1} \alpha_i K(x_i, x)$, where $\alpha_i \in \{-1, 0, +1\}$ is 0 whenever $sign(S_i(x_i)) = y_i$, and y_i otherwise. We consider the set of examples $M = \{(x_i, y_i) \in X : \alpha_i \in \{-1, +1\}\}$ as the *model* of the perceptron and slightly redefine the kernel-perceptron algorithm as in the following. Let $M = \emptyset$ be an initial empty model, a new input example x is added to the model M whenever its score

$$S(x) = \sum_{(x_i, \alpha_i) \in M} \alpha_i K(x_i, x) \tag{1}$$

has different sign from its classification y. Thus the update and the insertion of the new example follow the rule:

$$\textbf{if } (yS(x) \leq 0) \textbf{ then } M \leftarrow M \cup \{(x, y)\}.$$

Eq. (1), besides different definitions for the α values, is shared among many online learning algorithms (Freund and Schapire 1999), (Crammer et al. 2006) and it is also the most expensive operation in the prediction phase of a batch learning kernel method, such as the SVM (Cristianini and Shawe-Taylor 2000). It is trivial to show that, for online learning algorithms, the cardinality of M, and consequently the memory required for its storage, grows up linearly with the number of examples in the input stream; thus the efficiency in the evaluation of the function $S(x)$ linearly decreases as well. It is therefore of great importance to be able to speed up the computation of eq. (1), which is our goal in this paper. We study the case in which the input examples are structured data, specifically trees.

Mapping Kernels (MKs)

The main idea of mapping kernels (Shin and Kuboyama 2010) is to decompose a structured object into substructures and then sum the contributions of kernel evaluations on a subset of the substructures. More formally, let us assume a "local" kernel on the subparts, $k : \chi' \times \chi' \rightarrow \mathbb{R}$, and a binary relation $R \subseteq \hat{\chi} \times \chi$ are available, with $(\hat{x}, x) \in R$ if $\hat{x}$ "is a substructure" of x. The definition of $\hat{\chi}$, χ' and R varies according to the kernel and the domain, an example of a common setting is the following: χ, $\hat{\chi}$ and χ' are sets of trees and $(\hat{x}, x) \in R$ if $\hat{x} \in \hat{\chi}$ is a subtree of $x \in \chi$. Let $\hat{\chi}_x = \{\hat{x} \in \hat{\chi} | (\hat{x}, x) \in R\}$ be the set of substructures associated with x and let $\gamma_x : \hat{\chi}_x \rightarrow \chi'$ a function mapping a substructure of x into its representation in the input space of the local kernel (see (Shin and Kuboyama 2010) for some examples). Then the mapping kernel is defined as follows:

$$K(x_i, x_j) = \sum_{(\hat{x}_i, \hat{x}_j) \in \mathcal{M}_{x_i, x_j}} k(\gamma_{x_i}(\hat{x}_i), \gamma_{x_j}(\hat{x}_j)), \tag{2}$$

where $\mathcal{M}$ is part of a mapping system $\mathbb{M}$ defined as

$$\mathbb{M} = \Big(\chi, \{\hat{\chi}_x | x \in \chi\}, \Big\{\mathcal{M}_{x_i, x_j} \subseteq \hat{\chi}_{x_i} \times \hat{\chi}_{x_j} | (x_i, x_j) \in \chi \times \chi\Big\}\Big). \tag{3}$$

$\mathbb{M}$ is a triplet composed by the domain of the examples, the space of the substructures, and a binary relation $\mathcal{M}$ specifying for which pairs of substructures the local kernel has to be computed. $\mathcal{M}$ is assumed to be finite and symmetric, i.e. $\forall x_i, x_j \in \chi . |\mathcal{M}_{x_i, x_j}| < \infty$ and $(\hat{x_j}, \hat{x_i}) \in \mathcal{M}_{x_j, x_i}$ if $(\hat{x_i}, \hat{x_j}) \in \mathcal{M}_{x_i, x_j}$. The mapping kernel extends the convolution kernel framework in two aspects: *i)* the pairs of substructures on which the local kernel has to be computed is restricted according to $\mathcal{M}$; *ii)* the functions $\gamma_x()$ are introduced so that the space of the substructures, $\hat{\chi}$, and the input space of the local kernel, χ', do not necessarily have to be identical. In (Shin and Kuboyama 2010) it is proved that the kernel K of eq. (2) is positive semidefinite *if and only if* the mapping system $\mathcal{M}$ is *transitive*. A mapping system is transitive if and only if $\forall x_1, x_2, x_3 \in \chi . (\hat{x}_1, \hat{x}_2) \in \mathcal{M}_{x_1, x_2} \wedge (\hat{x}_2, \hat{x}_3) \in \mathcal{M}_{x_2, x_3} \Rightarrow (\hat{x}_1, \hat{x}_3) \in \mathcal{M}_{x_1, x_3}$.

Note that $\mathcal{M}$ is assumed to be finite. Such assumption may not hold in an on-line learning scenario. However, the mapping kernel has been extended to the case in which $\mathcal{M}$ is countably infinite (Shin 2011).

4 Efficient Score Computation for MKs

This section describes how to speed up the computation of eq. (1) when $K()$ is a mapping kernel.

Main Result

In order to speed up the computation of the score, we seek for an optimal representation of the model. Note that our purpose is to compute the exact value of the score, not to approximate it. We postulate the following necessary condition: a representation is optimal if it avoids recomputing the local kernel for the same substructures appearing in different examples. More formally, let us consider the set $\tilde{\chi} = \bigcup_{x \in \chi} \left\{ (\gamma_x(\hat{x}), x) | \hat{x} \in \hat{\chi}_x \right\}$. A relation $\overset{\mathcal{M}}{\sim}$ between elements of $\tilde{\chi}$ can be defined as follows:

$$\begin{aligned} &(x'_1, x_1) \overset{\mathcal{M}}{\sim} (x'_2, x_2) \Leftrightarrow \forall (x', x) \in \tilde{\chi}, \\ &[k(x'_1, x') = k(x'_2, x')] \wedge \{\hat{x}_1 \in \hat{\chi}_{x_1} | \gamma_{x_1}(\hat{x}_1) = x'_1\} \\ &\times \{\hat{x}_2 \in \hat{\chi}_{x_2} | \gamma_{x_2}(\hat{x}_2) = x'_2\} \subseteq \mathcal{M}_{x_1, x_2}. \end{aligned} \quad (4)$$

Relation (4) states that two elements of χ' are equivalent if their contribution to the computation of the score is identical for any $x \in \chi$. Then the idea is to seek for a representation that groups together those elements of χ' which are equivalent according to (4) in order to satisfy our optimality criterion.

It can be shown that (4) is in fact an equivalence relation with the assumption that the mapping is not trivial, that is, for any $x \in \chi$, and for all $\hat{x} \in \hat{\chi}_x$ it exists an $\hat{x}_3 \in \hat{\chi}_{x_3}$ such that $(\hat{x}, \hat{x}_3) \in \mathcal{M}_{x,x_3}$. In fact, while symmetry and transitivity can be proved easily, the above assumption is a sufficient condition to prove reflexivity: $(\hat{x}, \hat{x_3}) \in \mathcal{M}_{x,x_3} \overset{symmetry}{\Rightarrow} (\hat{x}_3, \hat{x}) \in \mathcal{M}_{x_3,x} \overset{transitivity}{\Rightarrow} (\hat{x}, \hat{x}) \in \mathcal{M}_{x,x}$. Note that any $\hat{x} \in \hat{\chi}_x$ not meeting the above assumption, could be ignored since it will give zero contribution to any kernel computation.

Let $Q = \tilde{\chi} / \overset{\mathcal{M}}{\sim}$ the quotient space, i.e. a partition of $\tilde{\chi}$ such that each pair of elements of $\tilde{\chi}$ belongs to the same partition if and only if they are equivalent with respect to $\overset{\mathcal{M}}{\sim}$. Given an equivalence class $\xi \in Q$, we denote by $(\xi_{\hat{\chi}}, \xi_\chi)$ a generic $(\hat{x}, x)$ such that $(\gamma_x(\hat{x}), x) \in \xi$. We call $(\xi_{\hat{\chi}}, \xi_\chi)$ the representative[1] of the class ξ. Note that the choice of $(\hat{x}, x)$ is irrelevant since any pair (y, x) such that $\gamma_x(y) = \gamma_x(\hat{x})$ is equivalent to $(\hat{x}, x)$ with respect to $\overset{\mathcal{M}}{\sim}$. Moreover, the transitivity of $\mathcal{M}$ ensures that, given $(\gamma_v(\hat{v}), v) \in \xi$, if $(\hat{x}, \hat{v}) \in \mathcal{M}_{x,v} \Rightarrow (\hat{x}, \xi_{\hat{\chi}}) \in \mathcal{M}_{x,\xi_\chi}$. The frequency of $\xi \in Q$ in a structure x is defined as

$$\psi(\xi, x) = \left| \left\{ \hat{x} \in \hat{\chi}_x | (\gamma_x(\hat{x}), x) \overset{\mathcal{M}}{\sim} (\gamma_{\xi_\chi}(\xi_{\hat{\chi}}), \xi_\chi) \right\} \right|. \quad (5)$$

If we instantiate the score function (1) for the case of mapping kernels with respect to a model M, the following is obtained:

$$\begin{aligned} S(x) = \sum_{x_i \in M} \alpha_{x_i} \sum_{(\hat{x}, \hat{x}_i) \in \mathcal{M}_{x,x_i}} k(\gamma_x(\hat{x}), \gamma_{x_i}(\hat{x}_i)) = \sum_{x_i \in M} \alpha_{x_i} \\ \cdot \sum_{\hat{x} \in \hat{\chi}_x} \sum_{\hat{x}_i \in \hat{\chi}_{x_i}} [\![(\hat{x}, \hat{x}_i) \in \mathcal{M}_{x,x_i}]\!] \, k(\gamma_x(\hat{x}), \gamma_{x_i}(\hat{x}_i)), \end{aligned} \quad (6)$$

where $[\![t]\!] = 1$ if t is true; 0 otherwise. At this point, by observing that, due to the definition of $\overset{\mathcal{M}}{\sim}$ and ξ, we have that $(\hat{x}, \hat{x}_i) \in \mathcal{M}_{x,x_i} \Leftrightarrow (\hat{x}, \xi_{\hat{\chi}}) \in \mathcal{M}_{x,\xi_\chi}$, we can introduce into the above equation a sum over all the equivalence classes without modifying the score function:

$$\begin{aligned} S(x) = &\sum_{x_i \in M} \alpha_{x_i} \sum_{\hat{x} \in \hat{\chi}_x} \sum_{\hat{x}_i \in \hat{\chi}_{x_i}} \sum_{\xi \in Q} [\![(\hat{x}, \hat{x}_i) \in \mathcal{M}_{x,x_i}]\!] \\ &[\![(\gamma_{x_i}(\hat{x}_i), x_i) \overset{\mathcal{M}}{\sim} (\gamma_{\xi_\chi}(\xi_{\hat{\chi}}), \xi_\chi)]\!] k(\gamma_x(\hat{x}), \gamma_{x_i}(\hat{x}_i)) \end{aligned} \quad (7)$$

Such addition will allow us to further compact (7). We will use $f_\xi(M) = \sum_{x_i \in M} \alpha_{x_i} \psi(\xi, x_i)$ to simplify the notation:

$$\begin{aligned} S(x) \quad &= \sum_{\hat{x} \in \hat{\chi}_x} \sum_{\xi \in Q} k(\gamma_x(\hat{x}), \gamma_{\xi_\chi}(\xi_{\hat{\chi}})) \, [\![(\hat{x}, \xi_{\hat{\chi}}) \in \mathcal{M}_{x,\xi_\chi}]\!] \\ &\quad \sum_{x_i \in M} \alpha_{x_i} \sum_{\hat{x}_i \in \hat{\chi}_{x_i}} [\![(\gamma_{x_i}(\hat{x}_i), x_i) \overset{\mathcal{M}}{\sim} (\gamma_{\xi_\chi}(\xi_{\hat{\chi}}), \xi_\chi)]\!] \\ &= \sum_{\hat{x} \in \hat{\chi}_x} \sum_{\substack{\xi \in Q \\ (\hat{x}, \xi_{\hat{\chi}}) \in \mathcal{M}_{x,\xi_\chi}}} k\big(\gamma_x(\hat{x}), \gamma_{\xi_\chi}(\xi_{\hat{\chi}})\big) \sum_{x_i \in M} \alpha_{x_i} \psi(\xi, x_i) \\ &= \sum_{\hat{x} \in \hat{\chi}_x} \sum_{\substack{\xi \in Q \\ (\hat{x}, \xi_{\hat{\chi}}) \in \mathcal{M}_{x,\xi_\chi}}} k\left(\gamma_x(\hat{x}), \gamma_{\xi_\chi}(\xi_{\hat{\chi}})\right) f_\xi(M). \end{aligned} \quad (8)$$

Note that in eq. (1), by looping over the equivalence classes and by summing the contributions of all equivalent substructures with the term $f_\xi(M)$ we satisfy the generic optimality criterion for a generic mapping kernel.

The model of a kernel method is normally represented either by a set of examples (dual representation) as in eq. (1), or by a vector of features (primal representation). The former can hardly be further compacted since usually the examples tend not to be repeated in the training set; the latter groups together features belonging to different examples. However, the size of the feature space can be exponential with respect to the number of examples. When the kernel function involved is part of the mapping kernel framework, an "intermediate" representation, i.e. in terms of the parts of the examples, can be used to represent the model. This is an intermediate representation since it shares features of both the primal and the dual representation: subparts belonging to different examples can be grouped together as in the primal representation and the size of the model, if the kernel does not have exponential complexity, can not be exponential with respect to the size of the dual representation. This last statement is a direct consequence of the fact that $\forall x_1, x_2 . \mathcal{M}_{x_1,x_2}$ must be less than exponential in size for the kernel to have less than exponential complexity.

In the general case, if we want to compute eq. (8) for an input x, we need to: keep in memory $\gamma()$ to compute $\gamma_x(\hat{x})$; to represent the model, we need ξ_χ, $\xi_{\hat{\chi}}$ for each $\xi \in Q$ for computing $\mathcal{M}$. However, as we will see in Sec. 6, this is not always the case, the nature of $K()$ may induce several optimizations, e.g., if $\mathcal{M}$ is independent from ξ_χ and $\xi_{\hat{\chi}}$, then we can choose to directly represent $\gamma_{\xi_\chi}(\xi_{\hat{\chi}})$ for each $\xi_\chi, \xi_{\hat{\chi}}$ in the model. In either case, we need to be able to correctly associate $f_\xi(M)$ values with local kernel computations.

A general algorithmic procedure to derive a model representation is the following. Starting from a mapping kernel function, first identify $\xi_\chi, \xi_{\hat{\chi}}, \gamma(), \mathcal{M}$. The analysis of eq. (4)

[1] It should be noticed that the representative does not belong to Q since the first element of the pair does not belong to χ', however this choice of a representative allows us to simplify the notation.

with the current instantiations of $\xi_\chi, \xi_{\hat{\chi}}, \gamma(), \mathcal{M}$ gives indication of the elements constituting the equivalence classes and therefore the structures needed to represent the model.

Model Optimization For Recursive Local Kernel Functions

Let $k(x', \xi') = h(g(x'_1, \xi'_1) \circ k(x'_2, \xi'_2) \circ \ldots \circ k(x'_n, \xi'_n))$ be a recursive function. Then χ' is a well-founded set with respect to an inclusion relation $\subset$, with $x'_1 \subset x', \ldots, x'_n \subset x'$ and $\xi'_1 \subset \xi', \ldots, \xi'_n \subset \xi'$. By following the same postulate we stated at the beginning of Section 4, we can look for a representation that avoids to recompute any $k(x'_j, \xi'_j)$ if x'_j, ξ'_j are subparts of multiple x', ξ', respectively. We can exploit $\subset$ and represent ξ' in terms of ξ'_1 plus the information that is needed to compute the non recursive part of $k(x', \xi')$. An example of such strategy is described in Section 6.

Notice that the type of optimization described in this section is different from the one in eq. (8). The structures ξ appear as terms in a summation in eq. (8) and therefore ξ elements appearing in different examples can be grouped together. In order to get the overall contribution of such ξ structures to the computation of the score, $k(x'_j, \xi'_j)$ need to be evaluated once only (and then multiplied by $f_{\xi'_j}(M)$). On the contrary, in a recursive definition the terms $k(x'_j, \xi'_j)$ may not be assumed to appear only as terms of a summation and thus the distributive property can not be applied to group together $k(x'_j, \xi'_j)$ terms. Therefore, while we can cache $k(x'_j, \xi'_j)$ values to avoid to re-evaluate them, in general, if the same $k(x'_j, \xi'_j)$ appear in the evaluation of two different pairs $k(x', \xi'), k(x', \xi'')$, it still needs to be accessed twice to be properly composed with the other terms appearing in $k(x', \xi')$ and $k(x', \xi'')$, respectively.

5 Related Work

While SVMs have a high generalization capability, several authors pointed out that one of its drawback is the time required both in learning and classification phases (Nguyen and Ho 2006), (Anguita, Ridella, and Rivieccio 2004), (Tipping 2001), (Rieck et al. 2010). Several approaches have been pursed to tackle this problem.

Nguyen and Ho (Nguyen and Ho 2006) describe an iterative process to replace two support vectors with a new one representing both of them. The replacement operation involves the maximization of a one-variable function in the range $[0, 1]$. Anguita et al. (Anguita, Ridella, and Rivieccio 2004) replace the set of support vectors with a single one, called archetype. However the archetype is defined in the feature space and thus it is necessary to solve a further quadratic optimization problem to find an approximation in input space. The approximated model, according to the authors "maintains the ability to classify the data with a moderate increase of the error rate".

Tipping (Tipping 2001) proposed the Relevance Vector Machine, an algorithm based on Bayesian theory with a functional form similar to SVM, which tries to minimize the number of support vectors produced during learning. However the Relevance Vector Machine requires $O(N^3)$ time and $O(N^2)$ memory to train. Relevance Vector Machines not only bound the user to a particular learning algorithm, but also cannot be used for on-line settings or large datasets. Recently Croce et al. (2017) applied Nyström to obtain a vectorial representation of trees. This way, they enable the use of an approximated tree kernels in deep neural networks.

All previous papers reduce the computational burden of the classification phase by finding an approximation of the model. In most cases the approximation is found by solving an optimization problem. The computational complexity or the type of learning algorithm prevents their use in on-line settings. In this paper we describe a way to speed up the computation of the score with no approximation and without the need to solve an optimization problem. The work closest to ours is the one of Shin et al. (Shin, Cuturi, and Kuboyama 2011). Their idea is to cache repeated local kernel computations for small substructures. Their proposal is different from ours because it applies to a single kernel computation and because our theoretically guided approach helps devising an optimal representation of the model which avoids to repeat any local kernel computation. Our approach generalizes the one described in (Aiolli et al. 2007) to any convolution kernel and can be used with any kernel based algorithm.

6 Compact Score Representation for TKs

This section discusses how eq. (8) can be instantiated to kernels for trees. Specifically, we first introduce a few representative convolution tree kernels, show how they can be represented within the mapping kernel framework and then outline the structures needed to compute the local kernel $k()$ and the $f_\xi()$ functions in eq. (8). The challenge then becomes to compactly represent all such structures in order to reduce the memory and computational burdens for computing the score.

Subset, Subtree, Partial and Elastic TKs

The evaluation of the Subtree, Subset, Partial and Elastic tree kernels is performed by implicitly extracting a set of tree fragments from the two input labelled ordered trees and then counting the matching ones. The kernels differ in the set of subtrees they match, which correspond to the ST, SST, PT and ET as defined in Section 2. All four kernels can be computed by the following recursive procedure:

$$K(T_1, T_2) = \sum_{v_1 \in V(T_1)} \sum_{v_2 \in V(T_2)} C(v_1, v_2), \qquad (9)$$

The function $C(v_1, v_2)$ computes the number of matching subtrees (subset, partial or elastic) rooted at v_1 and v_2. Its definition depends on the employed tree kernel. For example, the $C(v_1, v_2)$ for the SST kernel is defined according to the following rules: *1)* if the productions at v_1 and v_2 are different then $C(v_1, v_2) = 0$; *2)* if the productions at v_1 and v_2 are identical, and v_1 and v_2 have only leaf children (i.e. they are pre-terminals symbols) then $C(v_1, v_2) = \lambda$; *3)* if the productions at v_1 and v_2 are the same, and v_1 and v_2 are not pre-terminals, then

$$C(v_1, v_2) = \lambda \prod_{j=1}^{nc(v_1)} (\sigma + C(ch_j[v_1], ch_j[v_2])), \qquad (10)$$

where $nc(v_1)$ is the number of children of v_1, $ch_j[v]$ is the j-th child of node v and σ is set to 1. The complexity of the kernel is $O(|V(T_1)| \cdot |V(T_2)|)$. Although the Subtree kernel has $O(|V(T)| \log |V(T)|)$ complexity, where $|V(T)| = \max(|V(T_1)|, |V(T_2)|)$ (Viswanathan and Smola 2003), it can also be computed by setting $\sigma = 0$ in eq. (10). The $C()$ functions of the other tree kernels are here omitted for brevity (see references in section 2 for details). Let us just note here that both kernels are computationally more demanding than the SST kernel, but their complexity is still quadratic with respect to the number of nodes.

The four kernels above can be represented inside the mapping kernel framework: χ, χ' are sets of trees, $\hat{\chi}_x$ is the set of subtrees of x, $\forall x.\gamma_x()$ is the identity function and $\mathcal{M} \equiv \hat{\chi}_{x_1} \times \hat{\chi}_{x_2}$. The relation (4) becomes: $(x'_1, x_1) \sim (x'_2, x_2) \Leftrightarrow \forall x' \in \chi'.k(x'_1, x') = k(x'_2, x')$.

In order to represent a model as in eq. (8) we need to identify the representatives of the equivalence classes. Let us first introduce the following definition: we say that two ordered trees, T_1 and T_2, are identical if there exists an isomorphic mapping from T_1 to T_2 such that sibling ordering is preserved in both subtrees. Then, it can be shown that $(x'_1, x_1) \sim (x'_2, x_2)$ if and only if x'_1 is identical to x'_2. $\Leftarrow$) is obvious since the evalutation of the local kernel depends only on the structure of the subtree. $\Rightarrow$) can be shown as follows: if $(x'_1, x_1) \sim (x'_2, x_2)$ then $C(x'_1, x'_1) = C(x'_1, x'_2)$ and $C(x'_2, x'_2) = C(x'_2, x'_1)$, thus the set of subtrees (proper, subset, partial and elastic) rooted at x'_1 and the one rooted at x'_2 are identical. Since all four kernels consider as a feature the proper subtree rooted at x'_1 and x'_2, it implies that the subtrees x'_1 and x'_2 are identical. The property above shows that proper subtrees can be class representatives.

To compute the score for a model M the following is needed: the set of subtrees appearing in at least one tree in the model and, for each subtree, the corresponding $f_\xi(M)$ value. Fig. 1 reports an example of a model represented by two trees, T_1 and T_2 (Fig. 1-a), their decomposition into subtrees (Fig. 1-b), and the corresponding set of subtrees and $f_\xi(M)$ values (Fig. 1-c). The recursive nature of the eq. (10) implies the following inclusion relation between subtrees: $t_1 \subset t_2$ if t_1 is identical to the subtree formed by a child node of t_2 and all of its descendants. As a consequence, if two class representatives are in an inclusion relationship, $\xi' \subset \xi''$, then representing both two equivalence classes independently is redundant. Taking as an example ξ'', what is needed is only the information that is not in common with ξ' and then a reference to ξ' for the information in common between ξ' and ξ''. This is exemplified in Fig. 1: the equivalence class ξ_2 is included in $\xi_1, \xi_1 \subset \xi_3, \xi_1 \subset \xi_4, \xi_2 \subset \xi_3$ (Fig. 1-c); for example the equivalence class ξ_3 is represented by the node labelled as **a**, together with its frequency 1, and by links to the representatives of the classes ξ_1, ξ_2 (Fig. 1-d). Such compact representation results in a forest of annotated DAGs.

7 Experiments

Semantic Role Labelling (SRL) is the task of automatically extracting a predicate along with its argument from a natural language sentence. Previous work has shown that Semantic

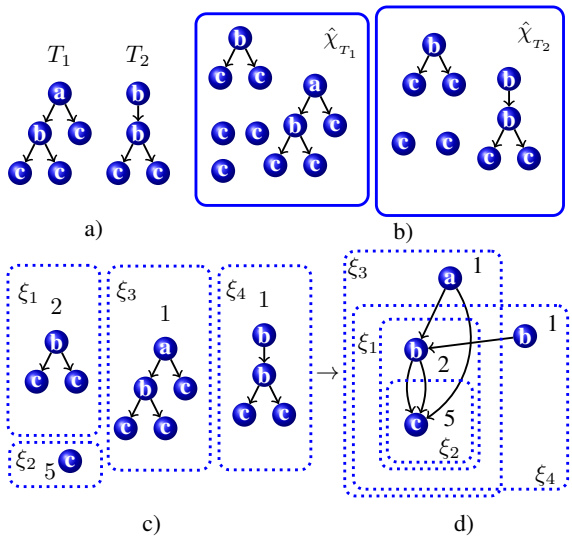

Figure 1: a) a model formed by two trees, T_1 and T_2; b) the decomposition into parts of the two trees; c) the set of equivalence classes formed according to eq. (4); each class ξ is described by a representative subtree ($\xi_{\hat{\chi}}$), and the frequency of the subtree in the model ($f_\xi(M)$); d) an inclusion relationship between subtrees is exploited to avoid redundancies when representing the same subtrees in different equivalence classes.

Role Labeling (SRL) can be carried out by applying machine learning techniques, e.g., (Gildea and Jurasfky 2002). The model proposed in (Moschitti 2004) applies tree kernels to subtrees enclosing the predicate/argument relation. More precisely, each predicate and argument pair is associated with the minimal subtree that dominates the words contained in both pair members. For example, in Fig. 2, the subtrees inside the three frames are the semantic/syntactic structures associated with the three arguments of the verb *to bring*, i.e. S_{Arg0}, S_{Arg1} and S_{ArgM}.

Unfortunately, the large size of the PropBank corpus makes learning via tree kernels rather expensive in terms of execution time, which becomes prohibitive when all the data is used. Consequently, the algorithms presented in the previous sections are very useful to speed up the learning/classification processes and make the kernel based approaches more applicable. The next section empirically shows the benefit of our DAG-based algorithms.

Space and Time Comparisons We measured the computation time and the memory allocation (in terms of input tree nodes belonging to the model) for both the traditional Perceptron algorithm and the voted Perceptron based on DAGs. The target learning tasks were those involved in Semantic Role Labelling, specifically argument boundary detection, i.e., all the nodes of the sentence parse tree are classified in *correct* or *incorrect* boundaries. The *correct* label means that the leaves (i.e., words) of the tree rooted in the target node are all and only those constituting an argument. As a refer-

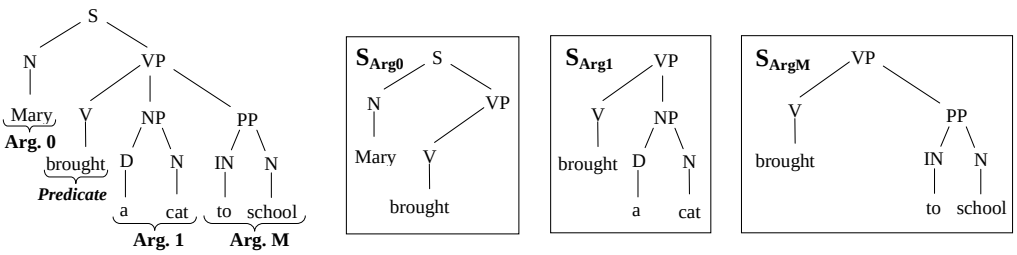

Figure 2: Parse tree of the sentence "Mary brought a cat to school" along with the PAF trees for Arg0, Arg1 and ArgM.

	Training	Validation	Test
Num. of trees	4,079,510	234,416	149,140
Num. of nodes	63,465,630	3,653,344	2,354,371
Avg num. nodes in a tree	15.56	15.58	15.79
Avg maximum outdegree	2.53	2.29	2.24
Max outdegree	59	13	15

Table 1: Statistics of syntactic trees in the boundary detection dataset.

ring dataset, we used PropBank along with PennTree bank 2 (Marcus, Santorini, and Marcinkiewicz 1993). Specifically, we used all the sections of the PennTree bank for a total of 276,039 positive and 4,187,027 negative tree examples.

The DAG performance is affected by node distribution within trees along with their maximum and average outdegree. Table 1 reports statistics about the data derived from the boundary detection dataset. Note that there is a large number of relatively small trees, which can have a large outdegree. The total amount of nodes to be processed is almost 70 millions, thus, the dataset can demonstrate the computational efficiency of our approach.

This dataset is almost prohibitive for computational expensive approaches such as SVM. It took 10,705 minutes just for processing 1 million examples with the faster SST kernel, e.g., about 7.5 days of computation. Thus we report the results obtained on the same experimental setting for the standard and voted dag perceptrons. Fig. 3 shows the execution time for the standard perceptron and the voted dag perceptron when using SST kernel combined with the polynomial kernel. In the case of the TKs and combination, we used the following parameters: $\lambda \in \{0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$ (TKs decay factor) and $\gamma \in \{0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$ (weighting the linear combination between tree and polynomial kernels). Only the values related to the faster and slower parameter settings are plotted.

Learning with the combination of kernels requires in the worst case ($\lambda = 1.0$ and $\gamma = 0.9$) $15,814.09$ seconds for the voted dag perceptron and $22,892.62$ seconds for the standard perceptron in the most favorable case ($\lambda = 0.6$ and $\gamma = 0.9$). Note that the voted perceptron, even in this unfavorable comparison, is 7078.53 seconds faster (a bit less than 2 hours). Fig. 4 shows the memory usage of standard perceptron and

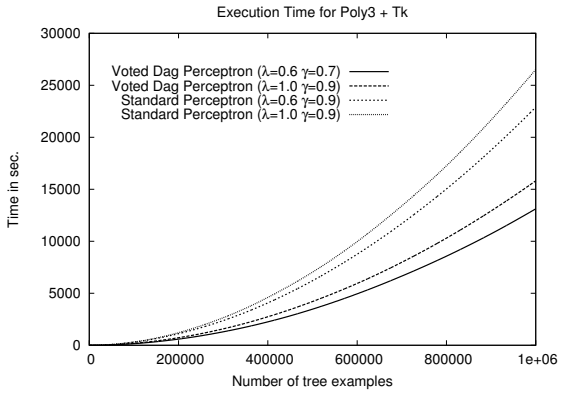

Figure 3: Execution time in seconds for the Standard Perceptron and the Voted DAG Perceptron over the training set with 1 million examples using a combination of Tk with different λ values and a polynomial kernel with degree 3 according to a parameter γ (Poly3+Tk). Only the slowest and fastest executions are reported.

voted dag perceptron algorithms.

The model created by the voted dag perceptron, when a combination of kernels is employed (Fig. 4), comprises from $124,333$ to $145,928$ nodes, while the model created by the standard perceptron comprises from $391,240$ to $625,501$ nodes. The amount of nodes used by the standard perceptron can be more than 4.2 times higher than the one employed by the voted dag perceptron.

Finally, in Fig. 5 we report the training times for the voted dag perceptron on the full training set (i.e., up to 4,079,510 trees) when using the parameters selected on the validation set after training 1 million of tree examples. It can be noticed that the training is completed after $129,163.98$ seconds (less than 36 hours) by using the polynomial kernel of degree 3, $51,235.9$ seconds (a bit more than 14 hours) by using the SST kernel ($\lambda = 0.4$), and $178,826.19$ seconds (less than 50 hours) when using a linear combination of the two previous kernels ($\lambda = 0.6$ and $\gamma = 0.9$). In any case, the training for the voted dag perceptron on the full training set is much faster than training an SVM on 1 million examples. We finally point out that using the entire dataset is also prohibitive for the perceptron so we could not report its running time, which is larger than one week.

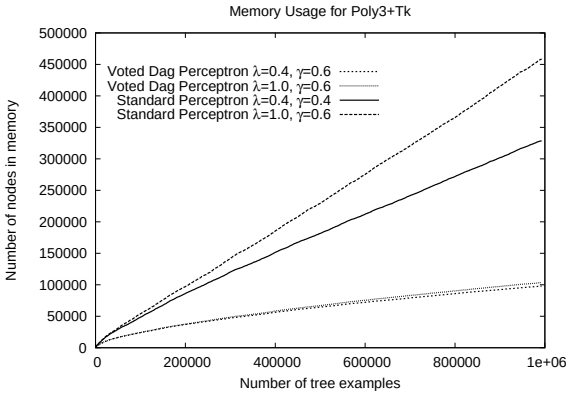

Figure 4: Evolution of the number of tree nodes stored in memory and belonging to the model developed by the Standard Perceptron and the Voted DAG Perceptron during training on the training set with 1 million examples using the SST kernel combined with polynomial kernel. Only kernel parameters with largest and lower number of nodes stored are plotted.

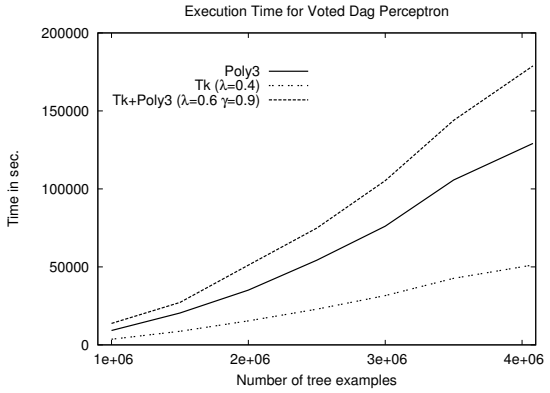

Figure 5: Execution times of voted dag perceptron when using the full training set for polynomial kernel of degree 3 (Poly3), SST kernel (Tk), and a combination of the two (Tk+Poly3).

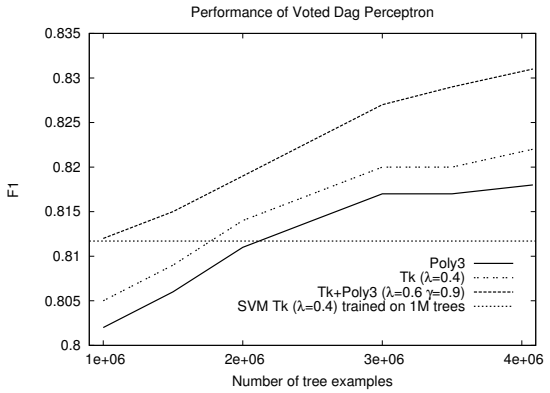

Figure 6: Accuracy performance on the test set, as measured by the F1 measure, for polynomial kernel of degree 3 (Poly3), the SST kernel (Tk), a combination of the two (Tk+Poly3) and an SVM trained on 1 million examples using the SST kernel (SVM Tk).

Accuracy Comparisons In Fig. 6 we report the accuracy measured by the F1, for the voted dag perceptron when using the kernels described above and the full training set. The figure also reports the F1 on the test set for SVM using the SST kernel (with $\lambda = 0.4$) and trained on the first million of tree examples. As expected, the voted (dag) perceptron using the SST kernel trained on 1 million examples gets lower performance (0.805) than SVM. However, the performance steadily increases with the number of training examples. With 2 millions examples the voted perceptron gets a better accuracy (0.814) in a much shorter training time, i.e., less than 4.3 hours versus about 7.5 days of computation required for training SVM. When the full training set is used, the voted perceptron takes a bit more than 14 hours to learn a model able to reach an F1 of 0.822, significantly improving the accuracy over the SVM. Just using a degree 3 polynomial kernel for numerical features improves performance (up to 0.818) as well (at the cost of a larger training time: see Fig. 5). The best performance is obtained by combining the two kernels: the training time still remains reasonable (on the full training set is less than 50 hours), while F1 increases up to 0.831. In summary, the adoption of the dag-based approach allows to improve accuracy by using more examples at a fraction of the time needed by an SVM trained on 1 million examples.

8 Conclusions

In this paper, we have proposed a general approach to speed up the computation of the score function of classifiers based on kernel methods for structured data. Our methods avoid the re-computation of kernels over identical substructures appearing in different examples. We theoretically analyzed the formulation of the score, eq. (1), for Mapping Kernels, the most comprehensive framework for specifying kernel functions for structured data, and derived various strategies to find an optimal representation of the model: this allows to reduce its memory consumption and speed up the score computation. We showed that our findings apply to most popular tree kernels. Finally, we provided empirical evidence of the benefit of our approach on Semantic Role Labelling task. We showed that the learning time of a perceptron algorithm on a large dataset of 4-million-instance decreases from more than a week to 14 hours only. It should be noted that our SVM-based model could only be trained on a significantly smaller number of examples. Thus, a simple voted perceptron could outperform the SVM models also with respect to classification accuracy.

Acknowledgments

We would like to thank the anonymous reviewers for providing us with suggestions and insights on how to further develop the ideas of the paper.

References

Aiolli, F.; Da San Martino, G.; Sperduti, A.; and Moschitti, A. 2007. Efficient Kernel-based Learning for Trees. In *CIDM 2007*, 308–315.

Anguita, D.; Ridella, S.; and Rivieccio, F. 2004. An algorithm for reducing the number of support vectors. In *Proceedings of the WIRN04 XV Italian Workshop on Neural Networks*. Perugia, Italy.

Collins, M., and Duffy, N. 2002. New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In *ACL02*.

Crammer, K.; Dekel, O.; Keshet, J.; Shalev-shwartz, S.; and Singer, Y. 2006. Online Passive-Aggressive Algorithms. *Journal of Machine Learning Research* 7:551–585.

Cristianini, N., and Shawe-Taylor, J. 2000. *An introduction to support vector machines and other kernel-based learning methods*. Cambridge University Press, 1 edition.

Croce, D.; Filice, S.; Castellucci, G.; and Basili, R. 2017. Deep Learning in Semantic Kernel Spaces. In *ACL'17 (Volume 1: Long Papers)*, 345–354. Stroudsburg, PA, USA: Association for Computational Linguistics.

Filice, S.; Da San Martino, G.; and Moschitti, A. 2015. Structural Representations for Learning Relations between Pairs of Texts. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 1003–1013. Beijing, China: Association for Computational Linguistics.

Freund, Y., and Schapire, R. E. 1999. Large Margin Classification Using the Perceptron Algorithm. *Machine Learning* 37(3):277–296.

Gildea, D., and Jurasfky, D. 2002. Automatic labeling of semantic roles. *Computational Linguistic* 28(3):496–530.

Haussler, D. 1999. Convolution kernels on discrete structures. Technical Report UCSC-CRL-99-10, University of California, Santa Cruz.

Jaakkola, T.; Diekhans, M.; and Haussler, D. 2000. A discriminative framework for detecting remote protein homologies. *Journal of Computational Biology* 7(1,2):95–114.

Kashima, H., and Koyanagi, T. 2002. Kernels for {Semi-Structured} Data. In *Proceedings of the Nineteenth International Conference on Machine Learning*, 291–298. Morgan Kaufmann Publishers Inc.

Kazama, J., and Torisawa, K. 2005. Speeding up training with tree kernels for node relation labeling. In *HLT/EMNLP*.

Kazama, J., and Torisawa, K. 2006. Semantic role recognition using kernels on weighted marked ordered labeled trees. In *Proceedings of CoNLL-X*, 53–60. New York City: Association for Computational Linguistics.

Kivinen, J.; Smola, A. J.; and Williamson, R. C. 2004. Online learning with kernels. *Signal Processing, IEEE Transactions on* 52(8):2165–2176.

Kuang, R.; Ie, E.; Wang, K.; Wang, K.; Siddiqi, M.; Freund, Y.; and Leslie, C. S. 2004. Profile-based string kernels for remote homology detection and motif extraction. In *3rd International IEEE Computer Society Computational Systems Bioinformatics Conference (CSB 2004)*, 152–160.

Marcus, M. P.; Santorini, B.; and Marcinkiewicz, M. A. 1993. Building a large annotated corpus of english: The Penn Treebank. *Computational Linguistics* 19:313–330.

Moschitti, A., and Zanzotto, F. 2007. Fast and effective kernels for relational learning from texts. In Ghahramani, Z., ed., *Proceedings of the 24th Annual International Conference on Machine Learning (ICML 2007)*, 649–656. Omnipress.

Moschitti, A. 2004. A study on convolution kernels for shallow semantic parsing. In *Proceedings of ACL'04*.

Moschitti, A. 2006. Efficient convolution kernels for dependency and constituent syntactic trees. In Fürnkranz, J.; Scheffer, T.; and Spiliopoulou, M., eds., *ECML*, volume 4212 of *Lecture Notes in Computer Science*, 318–329. Springer.

Nguyen, D., and Ho, T. 2006. A bottom-up method for simplifying support vector solutions. *IEEE transactions on neural networks* 17(3):792–796.

Rieck, K.; Krueger, T.; Brefeld, U.; and Mueller, K.-R. 2010. Approximate tree kernels. *Journal of Machine Learning Research* 11:555–580.

Rosemblatt, F. 1958. A probabilistic model for information storage and organization in the brain. *Psychological Review* 65:386–408.

Shawe-Taylor, J., and Cristianini, N. 2004. *Kernel Methods for Pattern Analysis*. Cambridge University Press.

Shin, K., and Kuboyama, T. 2010. A Generalization of Haussler' s Convolution Kernel — Mapping Kernel and Its Application to Tree Kernels. *Journal of Computer Science and Technology* 25(5):1040–1054.

Shin, K.; Cuturi, M.; and Kuboyama, T. 2011. Mapping kernels for trees. In Lise Getoor and Scheffer, T., ed., *Proceedings of ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, 961 – 968. Omnipress.

Shin, K. 2011. Mapping kernels defined over countably infinite mapping systems and their application. *Journal of Machine Learning Research* 20:367–382.

Tipping, M. E. 2001. Sparse bayesian learning and the relevance vector machine. *Journal of Machine Learning Research* 1:211–244.

Viswanathan, S., and Smola, A. J. 2003. Fast kernels for string and tree matching. In S. Becker, S. T., and Obermayer, K., eds., *NIPS 15*. Cambridge, MA: MIT Press. 569–576.

Wachsmuth, H.; Da San Martino, G.; Kiesel, D.; and Stein, B. 2017. The Impact of Modeling Overall Argumentation with Tree Kernels. In *EMNLP 2017*, 2369–2379. Copenhagen, Denmark: Association for Computational Linguistics.

Zanzotto, F. M., and Moschitti, A. 2006. Automatic learning of textual entailments with cross-pair similarities. In *ACL*. The Association for Computer Linguistics.

Zhang, M.; Che, W.; Aw, A.; Tan, C. L.; Zhou, G.; Liu, T.; and Li, S. 2007. A grammar-driven convolution tree kernel for semantic role classification. In *Proceedings of ACL'07*, 200–207. Prague, Czech Republic: Association for Computational Linguistics.