Generative Kernels for Tree-Structured Data

Davide Bacciu, *Member, IEEE,* Alessio Micheli, *Member, IEEE,* and Alessandro Sperduti, *Senior Fellow, IEEE*

***Abstract*—The paper presents a family of methods for the design of adaptive kernels for tree-structured data that exploits the summarization properties of hidden states of hidden Markov models for trees. We introduce a compact and discriminative feature space based on the concept of hidden states multisets and we discuss different approaches to estimate such hidden state encoding. We show how it can be used to build an efficient and general tree kernel based on Jaccard similarity. Further, we derive an unsupervised convolutional generative kernel using a topology induced on the Markov states by a tree topographic mapping. The paper provides an extensive empirical assessment on a variety of structured data learning tasks, comparing the predictive accuracy and computational efficiency of state-of-the-art generative, adaptive and syntactical tree kernels. The results show that the proposed generative approach has a good tradeoff between computational complexity and predictive performance, in particular when considering the soft matching introduced by the topographic mapping.**

***Index Terms*—generative kernels, hidden tree Markov models, learning for structured domain, structured data processing**

I. Introduction

STRUCTURED data appear in many real-world application domains. For example, parse trees arise in natural language processing tasks where a parse tree or a semantic related tree structure is generated starting from a sentence [1], [2]; moreover, tree-like representations/patterns can be naturally derived, for example, from documents (e.g. [3]) and HTML/XML documents in information retrieval [4], [5], [6], structured network data in computer security [7], molecule structures in computational chemistry [8], [9], and image analysis. In all these application domains, learning plays a crucial role since very often the user is interested in automatic classification/regression tasks where, starting from a set of labeled instances, a classifier/regressor is pursued. Since data is naturally organized in tree-like structures, learning approaches able to directly deal with this kind of representation should be preferred. Among all possible approaches, a prominent one is the use of kernel methods [10] where kernel for trees are used (e.g., [11]). The learning performance and quality of such methods depends on the appropriateness of the underlying kernel with respect to the nature of data and learning task. This is especially true in the context of structured data, where the lack of a natural metric on the structured domain makes it difficult to select an appropriate kernel. For this reason, when dealing with a tree structured domain where a-priori information that can lead to an ad-hoc selection of a suitable tree kernel is missing, it is better to try to devise a tree kernel directly from available data, taking care to avoid overfitting. Possible approaches along this line consist in using data mining techniques to select relevant structural features [12], self-organizing neural networks for structured data [13], or a posteriori feature space pruning strategies [14]. All these approaches are mainly based on heuristics and/or are prone to overfitting.

D. Bacciu and A. Micheli are with Dipartimento di Informatica, Università di Pisa, Pisa, Italy, e-mail: bacciu,micheli@di.unipi.it

A Sperduti is with the Dipartimento di Matematica, Università di Padova, Padova, Italy, e-mail: sperduti@math.unipd.it

Manuscript received ; revised .

We think that a valuable and principled approach, in this context, is to learn the metric on the structured domain directly from available data. Our goal, in particular, is the definition of a general family of adaptive kernels for tree-structured data which can be straightforwardly applied to different application domains, without the need of costly kernel and feature engineering phases based on heuristics and prior knowledge. To this end, we put forward the use of generative probabilistic models for trees to learn to capture the structural information needed to build the tree similarity metric in the kernel. Previous works have suggested the use of generative models to define both adaptive and non-adaptive kernels for structured data. The Fisher kernel approach, for instance, allows defining an adaptive kernel on the top of a trained generative model by extracting Fisher scores from its parameters: the Fisher kernel has been originally proposed in the context of sequences by [15] and subsequently extended to trees by [16], [17]. On the other side, generative models are typically exploited in the context of graphs to define non-adaptive kernels, such as in the marginalized kernel approaches [18],[19]. Here, the generative model is used only to generate random visits (walks or tree patterns, respectively) on which a non-adaptive substructure match is computed.

This paper contributes by providing a very general way to define adaptive tree kernels on the top of generative probabilistic models for tree-structured data by efficiently exploiting the structural information summarized by the latent variables defining the hidden generative process of the structures. We provide an extensive account of the applicability of the proposed kernels to a variety of generative tree models, characterized by different underlying probabilistic assumptions and generative processes. Specifically, we focus on strategies based on generative probabilistic models involving hidden Markov states, such as top-down [3], bottom-up [20] and input-driven [21] Hidden Tree Markov Models (HTMMs), as well as generative topographic mapping approaches for structured data (GTM-SD) [22]. We explore different inference strategies to exploit the information encoded in the hidden states of these models for the definition of adaptive kernels, assessing them both in terms of kernel expressivity and computational complexity. In particular, we study two strategies for determining and weighting the contribution of the single hidden states to structural similarity: point-wisely, using Viterbi (or alike) algorithms to identify the most likely state assignments, or cumulatively, using the hidden state posterior assignment. In combination to this, we study how different feature space encodings can be defined by varying the amount of structural information allowed in the feature space representation. Specifically, we show how to allow the encoding of parent-to-child relationships appearing in the trees, by representing them in the form of couples of hidden states, like bigrams in text documents. Considering this type of information enriches the expressivity of the kernel introducing a relatively small computational overhead. The same approach can be, of course, generalized to representing more articulated structural patterns, at the cost of an increase in the kernel complexity, as discussed later in the paper.

A second contribution of the paper is the introduction of an alternative approach to compute structural similarities by allowing soft-matching among the hidden states. The approach described above allows increasing the expressivity of the kernel by increasing the size of the hidden state multisets, e.g. by a larger number of hidden states or by allowing more complex structural matches (trigram, quadrigram, etc). However, depending on the nature of the tree dataset, the introduction of a larger feature space reduces the probability that the intersection between the multisets-encodings of two trees is not empty. The introduction of soft matching avoids such problems and allows positive matches between different hidden states encoding similar structural information, which would otherwise be discarded in the hard-matching approach. We use a GTM-SD [22] model as it allows a principled approach to decide which hidden states should be considered similar, based on a neighborhood function between projections of the hidden state assignments on the generative map.

The last key contribution of this paper is of reference nature, providing a unified view over the experimental performance of the state-of-the-art syntactic and adaptive tree kernels in literature confronted with the proposed generative kernels. We propose a thorough experimental assessment comprising 7 publicly available benchmarks on tree-data classification, spanning a variety of application areas, including parse trees, structured documents, and biochemical data, with different structural characteristics. The analysis focuses on assessing both the predictive classification performance of the kernels as well as their computational requirements, providing an useful tree kernels cookbook.

The paper is organized as follows: Section II provides an overview of the background on syntactic and adaptive tree kernels in literature. Section III introduces a novel family of generative tree kernels exploiting the structural information captured by the Markov hidden states of the probabilistic tree models. Section IV presents the experimental assessment and Section V concludes the paper with a final discussion.

This article founds on two works [20], [22] concerning generative models for structured data published recently in this journal. Part of the content of this paper has appeared in two conference papers [23], [24]. We therein collect those independent contributions into a unified framework to systematically exploit them for the construction of adaptive kernels with discriminative aims. The content of the conference papers has been widely and significatively extended by considering a more general formulation of the kernel family (which previously included only point-wise hidden state multisets), a larger selection of generative models (previously considering only a single bottom-up model), a novel computational complexity analysis, and a stronger and wider experimental assessment considering a larger pool of datasets and tree kernels (over 85% of the experimental analysis is novel).

II. Background

Kernel functions define similarity measures upon which learners, e.g. support vector machines, are built to solve classification/regression problems. Several kernel functions have been proposed in the past-years to deal with structured data (see [25] for an early survey of the main approaches within a clearly defined taxonomy). Convolutional kernels are among the most popular tree kernels as they efficiently exploit the hierarchical nature of tree-structured data. The key idea is to construct a kernel for compound objects by measuring the matching between their composing substructures. Measuring such match ultimately entails defining a similarity/dissimilarity metrics for two structured pieces of information, which is not a straightforward task.

A popular approach for the definition of such tree similarity is by means of syntactic kernels, that are a class of convolutional tree kernels where the degree of matching between two trees is determined by counting the number of common substructures among the trees [11]. This amounts to seeking a match between edges, nodes and labels in all the composing substructures generated by following syntactical rules on the structure of the tree. The various approaches in literature differentiate by the way they identify the composing substructures and by how they weigh the structural matches, that is a key factor in determining the computational complexity of the kernel. The *Subset Tree kernel* (SST) by [11], for instance, counts the number of matching proper subtrees by a recursive procedure that is $O(N_T^2)$, where N_T is the maximum number of nodes among the two trees. The *Subtree kernel* (ST) [26] restricts to matching only complete subtrees, making it computationally more efficient than SST, i.e. $O(N_T \log N_T)$, but results also in a reduced expressivity. The *elastic tree* kernel [27] instead extends SST by allowing matching nodes with different labels and matching between substructures built by combining subtrees with their descendants, but at the cost of an $O(N_T^3)$ complexity [28]. The *Partial Tree kernel* (PT) [29] relaxes SST to allow partial productions of the parse-tree grammar, basically allowing to perform partial matching between subtrees at the cost of an increased computational complexity, that is $O(N_T^2 \cdot L_T^3)$, where L_T is the maximum outdegree among the trees. The *Route kernel* [30] computes the matching between two trees in terms of number of common routes, that is the shortest path linking two nodes in a tree represented by the sequence of edge indices. A similar approach is taken by [31] where the kernel function is defined in terms of subpath sets, that are routes capturing vertical structures in rooted unordered trees. The subpath kernel has also an efficient version [5] that is $O(N_T^2)$ in worst case, but can run in linear time on average. Other proposed tree kernels are reviewed by [28], [32].

Syntactic kernels are defined based on the syntax of the structured data representation, which often formally describes the semantics of the data. As such, they require some form of advanced knowledge about the relevance, or the weight, that can be assigned to the various forms of substructure match. For instance, some tasks might require to weight more a label match between two nodes u and u' over an exact match of their corresponding subtrees $\mathbf{x}_u$ and $\mathbf{x}_{u'}$. Such knowledge is not always available and is, often, application and data dependent. Within this context, adaptive kernels have gained interest as they provide a means for inferring suitable similarity measures directly from data. Adaptive kernels can be seen as a form of distance metric learning, whose objective is the acquisition of a similarity metric (with the properties of a kernel) from a given collection of training instances. Generative kernels are a popular approach to construct adaptive kernels by obtaining such similarity information from a probabilistic model describing the generative process of some sample data.

The *Fisher kernel* [15] denotes a general class of generative kernels that can be derived out of any parametric generative model. The underlying idea of the approach is to represent an input sample $\mathbf{x}$ in a feature space defined by the derivative of the log-likelihood $\log P(\mathbf{x}|\theta)$ of the generative model, with respect to its parameters θ. The Fisher kernel has been introduced by [15] with application to sequential data classification, using the *Hidden Markov Model* (HMM) as generative model for sequences. In [16] it has been extended to deal with tree-structured data using the *standard* HTMM [3] as a generative model for the structured samples. Note that the Fisher tree kernel is not convolutional, since the matching between two trees is performed based on the similarity between the respective Fisher scores, which does not allow direct matching between substructures. The computational complexity of the Fisher kernel depends on the parameterization of the underlying generative model. In [33] it has been discussed an alternative feature space obtained by concatenating the sufficient HTMM statistics, as well as a tree kernel based on the probability product approach by [34]. A comparative analysis by [33] shows that the Fisher Kernel has the best performance among the three in tree classification tasks and is therefore used as a baseline in the experimental assessment in Section IV.

Marginalized Kernels [35] put forward a different approach to designing kernel for structured data which can exploit the information captured in latent variables of a generative model. The role of the generative model in the definition of the marginalized kernel depends on the specific kernel instantiation. The *marginalized kernel for sequences* [35], in particular, exploits HMMs to define a joint kernel that counts the co-occurrences of hidden states and observed labels in the sequences and weights them by the posterior probabilities of the HMM hidden states computed through the forward-backward algorithm. The marginalized kernel has been later extended to graphs [18], [36], [37], using a joint kernel that counts the number of matching pairs of random walks in two graphs, where the latent variable $\mathbf{h}$ is a sequence of graph vertices generated by a first-order Markov random walk (as in HMM for sequences). A number of graph kernels have been proposed using a similar intuition of measuring graph similarity in terms of matching common subpaths [38] and subtrees [19], [39]. However, these do not exploit the statistical features of a generative model trained on the structured samples, whereas they resort to (partial) graph visits to generate the substructure features and then perform hard syntactical matching between the substructures to measure graph similarity. The kernel family introduced in this paper, on the other hand, puts forward a soft matching approach, where the structure similarity metric is data-induced thanks to the exploitation of the information captured by the underlying generative model. This approach partially resembles the early works on marginalized kernels for sequences and random walks which, nevertheless, have never been defined to handle specifically tree-structured data.

The key difference between the two approaches lies in the role and exploitation of the generative model. In particular, the proposed kernel family uses a probabilistic model that provides a distribution for the specific class of structured samples. We relax the strict syntactical sequence matching that is used to compute the marginalized kernels, introducing a measure of structural similarity based on the information summarized by the Markov hidden states, possibly complemented by some local structural properties such as parent-child relationships. Section III-D shows how such relaxation of the structure matching principle yields to kernels that are linear in the size of the structures. Note that in the marginalized approach the probabilistic model depends solely on the topology of the single sample graph for which it generates the sub-graph visits and is thus independent from the graph population it belongs to (i.e. the structured training set). In the proposed approach, on the other hand, the generative model acquires a distribution that consider the full population of training structures. In other words, the marginalized kernel is not defining an adaptive approach and, anyway, the application of the marginalized graph kernel to the specific case of tree-structured graphs would lead to a different approach (and to different results) with respect to that proposed in this paper.

The *Activation Mask* (AM) kernel [13] constructs an adaptive convolutional kernel from a trained unsupervised recursive neural network for structured data, that is the *Self Organizing Map for Structured Data* (SOM-SD) [40]. The SOM-SD extends the Self Organizing Map (SOM) approach by allowing to process structured input by learning a topological map such that similar trees tend to activate the same neurons on the map. The key intuition underlying the AM kernel is to define a feature space having one dimension associated to each neuron of the map. Then a vectorial representation for a tree can be obtained by considering which neurons are activated by the nodes of the tree. Once the above representation has been computed for any pair of trees, a kernel can be promptly defined as the dot product of these representations.

III. Generative Kernels on Hidden Markov States

We introduce scalable generative tree kernels exploiting the information captured by the hidden Markov states of the underlying probabilistic tree model. Section III-A discusses a family of generative tree kernels based on the concept of hidden state multiset and on the use of the Jaccard multiset similarity. Section III-B discusses two alternative approaches to compute the multiset encoding of a tree from a trained generative model, while Section III-C shows how a topology induced on the Markov states can be exploited to derive a convolutional generative kernel capable of computing direct matches between tree substructures.

A. A Generative Kernel Family on Hidden States Multisets

Generative approaches for trees allow modeling probability distributions over spaces of trees. This is achieved by generalizing the HMM approach for the sequential domain, through learning of an hidden generative process for labeled trees, that is regulated by hidden state variables modeling the structural context of a node and determining the emission of its label. By borrowing the nomenclature from HMM, these models are typically referred to as *Hidden Tree Markov Models* (HTMMs). Earlier on this journal [20], we have shown how different directions of the generative process result in models with different probabilistic assumptions and representational capabilities. In this context, it has been proposed the *bottom-up* HTMM (BHTMM) [20] that defines a generative process that composes the child subtrees of each node in the tree in a recursive fashion, from the leaves to the root of the tree. It has been shown [20] how this allows to capture more discriminative structural information with respect to the *top-down* HTMM [3] (i.e. the *standard* HTMM in Section I), which implements a generative process for all paths from the root to leaves of the trees. Both THTMM and BHTMM implement a *homogenous* generative process by learning an unconditional model $P(\mathbf{x}^n|\theta)$, where the input trees $\mathbf{x}^n$ are the outcome of a generative process that depends solely on the model parameters θ. Alternatively, the *Input-Output* BHTMM (IO-BHTMM) [21] defines a *non-homogenous* approach that allows learning the input-conditional model $P(\mathbf{y}^n|\mathbf{x}^n, \theta)$, where the input tree $\mathbf{x}^n$ conditions the generative process of an output structure $\mathbf{y}^n$ that, in a supervised-learning interpretation, might be understood as the target.

Notwithstanding the differences in the generative processes, such probabilistic tree models share the common intuition of introducing multinomial latent variables Q_u, associated to each node u and referred to as *hidden states*, to allow simplifying the conditional probabilities underlying the model. This is realized by introducing a set of hidden state variables associated with a state transition dynamics that follows the direction of the generative process, e.g from a node u towards its children $ch_l(u)$ for the top-down case. Specifically, an observed tree $\mathbf{x}^n$ is modeled by a set of hidden state variables $\{Q_1, \dots, Q_u, \dots\}$ following the same indexing as the observed nodes $u \in \mathcal{U}_n$, where $\mathcal{U}_n$ is the set of nodes in $\mathbf{x}^n$, and assuming values on the discrete set of hidden states $\{1, \dots, C\}$.

The hidden states variables essentially serve to summarize structural information concerning tree components, providing an adequate context, e.g., for the emission of a node label. For instance, in the bottom-up BHTMM, an hidden state Q_u can be thought of as encoding information on the substructure rooted on the u-th node. By exploiting such rich and, yet, compact representation of the structured information, we introduce an efficient generative kernel for trees founding on the concept of hidden states *multisets*. Roughly, each tree is represented in terms of its associated hidden states and structure similarity is computed on the basis of overlap in the hidden states' configurations. More specifically, given a trained HTMM, we transform a tree $\mathbf{x}$ into a *bag-of-states*, that is a vector of hidden state counts, similarly to how textual documents are represented as vectors of word counts.

The computation of the multiset encoding of a tree entails the estimation of its most likely hidden state assignment. Such an estimate can be obtained through various approaches that differ for the interpretation of what an optimal hidden state assignment is and that yield to different multiset encodings of the tree. We focus on two widely accepted formulations which are backed up by two robust and efficient inference algorithms for HTMM, that are the *Viterbi* algorithm and the *Upwards-Downwards* algorithm. The *Viterbi Algorithm* [41] is a dynamic programming approach that serves to estimate the hidden states that maximize the joint probability with the observed tree $\mathbf{x}$, i.e.

$$\max_{\mathbf{q}} P(\mathbf{X} = \mathbf{x}, \mathbf{Q} = \mathbf{q}), \tag{1}$$

where $\mathbf{q}$ is a (generic) hidden state assignment for the observed tree $\mathbf{x}$. The *Upwards-Downwards* algorithm, on the other hand, is an extension to trees of the Forward-Backward inference algorithm for HMMs on sequences [41] which allows to compute the posterior of the hidden states variables given the observed tree $\mathbf{x}$, i.e.

$$P(Q_u = j|\mathbf{x}), \text{ for } j \in 1, \dots, C. \tag{2}$$

Rather than associating a single hidden state to each node of the observed tree, the posterior allows to weight the contribution of each hidden state j to the node, yielding to a denser, yet potentially more informative multiset encoding of the tree.

Different bag-of-states encodings can be defined depending on the amount of syntactical (structural) information that we want to introduce in the kernel feature-space representation. In this work, we consider two forms of bag-of-states, shown in Fig. 1, corresponding to *unigram* and *bigram* hidden states multisets. The *unigram* is the simplest form of multiset that is based on measuring the occurrence of each hidden state independently for each node of the structure. In other words, the *unigram* encoding defines a mapping $\Phi : \mathbf{T} \rightarrow \mathbb{R}^C$ from the space of tree structures $\mathbf{T}$ to a C-dimensional feature space, such that the i-th component of the feature vector, i.e. $\Phi_i(\mathbf{x}_n)$, measures the occurrence of the i-th hidden state in structure $\mathbf{x}_n$ (see left of Fig. 1). How such occurrence is measured, depends on the type of inference algorithm used and on the weight associated to the hidden state of the specific node u (e.g. $W_u(i)$ terms in Fig. 1). Section III-B provides details on two encodings associated with *Viterbi* and *Upwards-Downwards* state inference.

The *unigram* feature-space captures information on the prevalent *topics* in the tree, but does not convey any structural

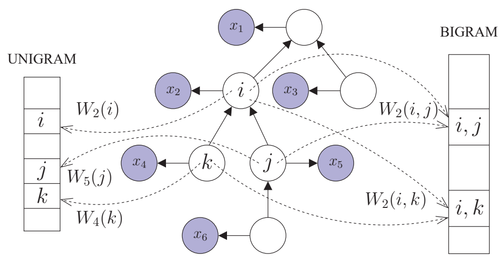


Fig. 1. Examples of unigram and bigram hidden states multisets for an example BHTMM generative model. In the unigram representation (left), the i-th vector component measures the occurrence of the i-th hidden state in the tree; the occurrence is weighted by a term (i.e. $W_2(i)$,$W_5(j)$ and $W_4(k)$ for hidden states associated to nodes $2, 5$ and 4, respectively) that depends on the inference algorithm used to estimate hidden-state assignment. In the bigram (right), there is a vector component for each pair of hidden states (e.g. i, j): the corresponding entry stores co-occurrence information concerning the first hidden state (e.g. i) being associated to a node whose child is assigned to the second hidden state (e.g. j). Occurrence is weighted by a term $W_u(i, j)$ similarly to the unigram case.

information, besides that captured by the generative model and conveyed by the hidden state assignment. To introduce some form of syntactical knowledge, we might be interested in modeling the co-occurrence of hidden-states in a parent-children relationship (see right of Fig. 1). This is similar to when, in document analysis, we model the co-occurrence of two adjacent words in a text by means of a word bigram. In analogy to this, we define an hidden state *bigram*, where an input tree $\mathbf{x}^n$ is transformed in a (C^2)-dimensional feature-vector $\Phi(\mathbf{x}^n)$, such that its i_j-th element $\Phi_{i_j}(\mathbf{x}^n)$ measures how often a node u is associated to the i-th hidden state, when its child $ch(u)$ is associated to the j-th hidden state. The bigram encoding allows to represent the co-occurrence of hidden states patterns between a parent node and each of its children taken independently, thus providing the kernel with some form of (partial) structural information. Note that such a multiset encoding approach can be taken further by introducing increasing amounts of syntax in the feature space (e.g. by considering the $(L + 1)$-gram of a node with its L children), at the cost of an increase in feature number.

Once obtained a multiset representation for the trees, we need to define an appropriate kernel for such a feature space. We propose the *Jaccard similarity* [42], that is a well known metric for comparing multisets and that, in its most general form, writes as

$$J(Z_1, Z_2) = \frac{f(Z_1 \cap Z_2)}{f(Z_1 \cup Z_2)} \tag{3}$$

where f is a suitable function (e.g. cardinality). For the purpose of this paper, we define the *Jaccard kernel* for trees as the multiset Jaccard similarity

$$k_{jac}(\mathbf{x}^1, \mathbf{x}^2) = \frac{\sum_{i=1}^{D} \min(\Phi_i(\mathbf{x}^1), \Phi_i(\mathbf{x}^2))}{\sum_{i=1}^{D} \max(\Phi_i(\mathbf{x}^1), \Phi_i(\mathbf{x}^2))} \tag{4}$$

where $\Phi(\cdot)$ is one of the multiset encodings discussed above (and the associated weighting schema) and D is the corresponding feature space size (e.g. $D = C^2$ for a bigram encoding). Our choice is motivated by the fact that Jaccard favours matching items over non-matching ones, e.g. with respect to linear/cosine product, which we expect to result in a structural similarity that favours common substructures over non-shared ones.

The proposed approach defines a broad family of generative tree kernels whose actual instantiation depends on

1) the underlying generative model: as discussed early in the section, probabilistic tree models can differ for the direction of the generative process (e.g. bottom-up, top-down) as well as for its homogeneity (e.g. input-driven vs homogenous models);
2) how hidden state occurrence is weighted: this is mostly influenced by the inference algorithms used to estimate the hidden state assignment, the most commonly used being the *Viterbi* and *Upwards-Downwards* algorithms;
3) the amount of syntactical information introduced in the feature space depending on the multiset type (e.g. unigram, bigram, etc.).

In the following, we discuss and evaluate different kernel instantiations resulting from different design choices at the level of generative tree models, inference algorithms and multiset types. We discuss how the *Viterbi* and *Upwards-Downwards* algorithms can be used to obtain multiset encodings that differ in the way they measure and weight hidden-state occurrence. In particular, we focus our analysis on the unigram and the bigram representation, and on a combination of the two, obtained by concatenating the unigram with the bigram into a $(C + C^2)$-dimensional feature space (*unibigram* in the following).

Note that the proposed generative kernel approach is not limited to dealing with tree-structured data, being enough general to be seamlessly applied to generative models and data-types other than those presented in this paper. For instance, classical HMMs for time series can be used to obtain a bag-of-states encoding for sequential data and to define a generative Jaccard kernel for sequences. The same approach can be used for any probabilistic model within the family of HMMs with discrete state-space. More generally, the proposed multiset encoding can be applied to the wide family of latent variable models with multinomial latent space, e.g. Probabilistic Latent Semantic Analysis [43].

B. Computing Multiset Encodings

The Viterbi and Upwards-Downwards algorithms address two fundamental inference problems in HTMM, providing two different forms of hidden state information associated to an observed tree. These two algorithms can be exploited to define two alternative approaches to measure hidden state occurrence, ultimately yielding to different multiset encodings for the tree. On the one hand, the Viterbi algorithm provides information on the single most likely hidden state that can be associated to each node in a tree by maximizing the joint probability in (1). In this context, it is natural to assume that a constant weight (e.g. 1) can be associated to each hidden state occurrence determined by the Viterbi algorithm. On the other hand, the Upwards-Downwards algorithm provides a node-dependent weight (i.e. the posterior in (2)) measuring the contribution of each hidden state to the single nodes of the tree. In the following, we discuss the multiset encodings associated to the two inference algorithms and we show how their computation can be embedded in the steps the respective inferential procedures.

The solution determined by the Viterbi Algorithm for the optimization problem in (1) provides the most likely hidden state assignment $Q^*_{n,u}$ for each node u in an input tree $\mathbf{x}_n$. By means of such Viterbi states, it is possible to compute a *Viterbi-unigram* encoding by counting the occurrences of the single hidden states in $Q^*_{n,u}$. Given a tree $\mathbf{x}^n$ and its associated hidden state assignments $Q^*_{n,u}$, it is transformed into a C-dimensional feature-vector $\Phi^V(\mathbf{x}^n)$ such that its i-th component is

$$\Phi^V_i(\mathbf{x}^n) = \sum_{u \in U_n} \delta(Q^*_{n,u}, i) \text{ and } i = 1, \ldots, C \quad (5)$$

where $\mathcal{U}_n$ is the set of nodes in the n-th tree and $\delta(\cdot,\cdot)$ is the Kronecker function. Similarly, we can define a *Viterbi-bigram*, where an input tree $\mathbf{x}^n$ is transformed in a (C^2)-dimensional feature-vector $\Phi^V(\mathbf{x}^n)$, such that its i_j-th element is

$$\Phi^V_{i_j}(\mathbf{x}^n) = \sum_{u \in U_n} \sum_{l \in ch(u)} \delta(Q^*_{n,u}, i)\delta(Q^*_{n,l}, j) \quad \text{and} \quad i, j = 1, \ldots, C. \quad (6)$$

where $ch(u)$ is the set of children of node u. In practice, the encodings in (5) and (6) use the Kronecker function as a weight $W_u(i)$ for hidden state occurrence, such that the counts of a multiset component are increased by one each time the corresponding hidden state configuration is found in the Viterbi states for the tree. Both feature-space encodings can be computed by a single visit of the tree which, for efficiency, can be embedded in the Viterbi recursion with only a minor modification in the (constants of the) Viterbi computational complexity. The Supplemental Material provides a procedural view of the computation of the bigram multiset for a BHTMM that exploits its Viterbi algorithm.

The Upwards-Downwards algorithm computes the posterior probability of the hidden states of the nodes of an observed tree by exploiting a decomposition of the posterior into two terms that can be computed recursively through and upwards visit of the tree followed by a downward visit. The details of the Upwards-Downwards algorithm depend on the underlying generative model and are omitted here: the reader is referred to the original papers of the various HTMM models for the details. Posteriors provide a measure of how much hidden states contribute to the generation of the observed tree: therefore, they can be exploited as weighting factors $W_u(i)$ in the multiset encoding. The *posterior-unigram*, for instance, can be computed from the posterior in (2): given a tree $\mathbf{x}^n$ and its associated single state posterior $\epsilon_{n,u}(i) = P(Q_u = i|\mathbf{x}^n)$, it is transformed into a C-dimensional feature-vector $\Phi^P(\mathbf{x}^n)$ such that its i-th component is

$$\Phi^P_i(\mathbf{x}^n) = \sum_{u \in U_n} \epsilon_{n,u}(i) \text{ and } i = 1, \ldots, C. \quad (7)$$

The *Posterior-bigram* can be computed using an higher-order posterior

$$\epsilon^l_{n,u,ch_l(u)}(i,j) = P(Q_u = i, Q_{ch_l(u)} = j|\mathbf{x}^n),$$

that is the posterior probability of a node u being in the i-th state while its l-th child is in state j. By this means, an input tree $\mathbf{x}^n$ is transformed in a (C^2)-dimensional feature-vector such that its i_j-th element is

$$\Phi^P_{i_j}(\mathbf{x}^n) = \sum_{u \in U_n} \sum_{l \in ch(u)} \epsilon^l_{n,u,ch_l(u)}(i,j) \quad \text{and} \quad i, j = 1, \ldots, C. \quad (8)$$

Similarly to the Viterbi case, the posterior encodings can be computed as part of the Upwards-Downwards procedure: the Supplemental Material exemplifies the steps needed to compute the posterior-bigram multiset using a BHTMM.

C. Inducing Topology on Markov States

The Jaccard kernel is based on an hard-matching between the hidden state labels in the two trees being considered. In fact, the encoding of a tree into an hidden state multiset essentially accounts to a relabeling of the tree, where node labels represent the hidden states assignments for the node and its neighbors. The kernel then considers two nodes having exactly the same state-label as a positive match, i.e. they are similar, while different hidden state labels mean no similarity. One drawback of this hard-matching approach is that it does not account for the possibility that two different hidden states might actually encode very similar structures, and thus assigns null similarity to their match. Another drawback is associated to the sparsity problem that may result from increasing the size of the hidden state space. Both problems can be circumvented by introducing a soft matching for the hidden states which, however, cannot be allowed between any couple of hidden states due to computational feasibility reasons.

A principled approach to determine which hidden states should be considered for the soft matching is to induce a topographical organization in the hidden states of the Markov model. By this means, distinct Markov states that are neighbors with respect to the topographical principle can be considered to be encoding similar structural knowledge. The *Generative Topographic Mapping for Structured Data* (GTM-SD) [22] implements such a constrained Markov model. By exploiting the information captured by the BHTMM hidden states, it provides a projection of a tree on the topographic map, such that similar structures are projected to nearby points on the map. The GTM-SD constrains the hidden states of a BHTMM to follow a topographical organization, by assuming that the hidden states Q_u are indexed by C latent centers of a bi-dimensional GTM map [44] (see Fig. 2). In other words, the assignment $Q_u = i$ indicates that the u-th node is assigned to the i-th hidden state which, in turn, is associated to the bi-dimensional latent point c_i in the GTM map, as in Fig. 2. Following the ideas in [44], the topologically constrained hidden states are obtained by generating the parameters of the node emission probabilities through a continuous smooth mapping $\Gamma(\cdot)$ from the GTM-SD lattice to the data space.

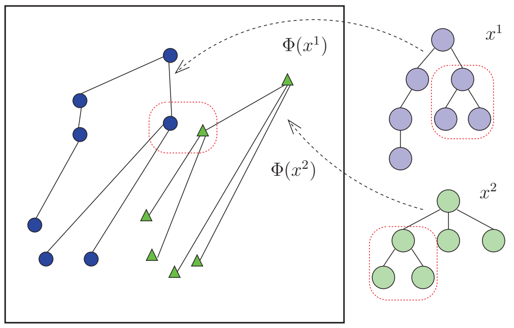


Fig. 2. Example projection of two trees (on the right-side), $\mathbf{x}^1$ and $\mathbf{x}^2$, on the bi-dimensional GTM-SD map (the square on the left-side). The smooth-mapping $\Phi(\cdot)$ projects each node to a point in the topological map. The exact projection point depends on the hidden state assignment of the node. Topographic organization ensures that similar structures are projected closely on the map. See for instance the two substructures in the dotted squares: since they have a similar structure, their roots are projected close on the map (i.e. the two nodes enclosed by the rounded square on the map).

The bottom-up generative process in GTM-SD allows to project each substructure composing a tree to a bi-dimensional latent point resulting from the hidden state assignment of the node acting as root of the substructure. This provides a distinctive fingerprint of the tree on the topographic map, corresponding to the projection of all its substructures on the map, as shown in the example in 2. Such a fingerprint can be exploited to define the soft-matching kernel by borrowing on the ideas of the Activation Mask (AM) kernel by [13]. In particular, we allow only hidden states that have a topographic distance below a user defined value to be considered for the soft-matching and we formulate a similarity measure between structures based on the distance between the points on the map resulting from their projection.

In order to compute the kernel, we need first to clarify how the tree-projection process in Fig. 2 works. To this end, consider a trained GTM-SD model: the projection of a tree $\mathbf{x}^n$ on the topological map is obtained by mapping its root onto the lattice by using its hidden state assignment Q_1. Several approaches exist to obtain such projection [22], again depending on the type of weighting given to the hidden states, as for the multisets encodings in Section III-B. The *mode* projection maps the tree to the latent point c_i corresponding to the most likely hidden state assignment $Q_1 = i$ for the tree root, resembling the Viterbi encoding approach. Clearly this projection does not exploit the continuity and smoothness properties of the map, as it collapses all projections solely on the (finite and discrete) latent centers c_i of the map. By following the posterior encoding approach, on the other hand, we are allowed to map a tree $\mathbf{x}^n$ to its *posterior mean*, i.e. the average of the latent point centers c_i, weighted by the respective posterior probabilities $P(Q_1 = i|\mathbf{x}^n)$, that is

$$\overline{X}(\mathbf{x}^n) = \sum_{i=1}^{C} P(Q_1 = i|\mathbf{x}^n) \cdot c_i. \tag{9}$$

The projections provided by (9) span the whole topographic map, with an intrinsically superior discrimination power with respect to mode projection. The posterior mean projection for each subtree $\mathbf{x}^n_u$ (rooted on node u of $\mathbf{x}^n$) can be computed as part of the *upwards recursion* in the Upward-Downwards algorithm in Section III-B, whose outcome is the posterior probability $P(Q_u = i|\mathbf{x}^n_u)$ (see [22] for details). By this means, we take a *compositional* approach where the hidden state assignment for node u is determined using only information propagated from the subtree $\mathbf{x}^n_u$, and discarding the contextual information from the rest of the $\mathbf{x}^n$ structure. This can be done very efficiently by considering u as the root node of an *isolated* tree $\mathbf{x}^n_u$. In other words, this is equivalent to projecting the subtree $\mathbf{x}^n_u$ on the map using (9), where the upwards parameter $P(Q_u = i|\mathbf{x}^n_u)$ is used in place of the contextual posterior $P(Q_u = i|\mathbf{x}^n)$.

The posterior mean mechanism provides a way to compress structural information to points on a bi-dimensional map. To define the feature space representation for our soft matching kernel, given a tree $\mathbf{x}^n$, we first obtain the hidden state activations for each composing subtree $\mathbf{x}^n_u$ using the compositional posterior $P(Q_u = i|\mathbf{x}^n_u)$. Then, we project all subtrees on the map coordinates returned by (9), by incorporating projection computation in the steps of the upwards recursion: see the Supplemental Material for an algorithmic description of this process. Figure 2 shows that this results in a feature-space where a tree is encoded by the posterior mean projection of all its nodes onto the GTM-SD map. Evaluating the similarity between two structures, in this context, becomes a matter of computing distances between points on the GTM-SD map. To this end, we define the following *weight function* between two generic points p and p' on the map, i.e.

$$T_\epsilon(p, p') = \begin{cases} \epsilon - d(p, p'), & \text{if } d(p, p') \leq \epsilon \\ 0, & \text{otherwise} \end{cases} \tag{10}$$

where $d(p, p')$ is the standard Euclidean distance. The term ϵ determines a neighborhood for the points on the map which regulates the influence of distant substructures in defining the kernel-induced similarity measure. In other words, it is the parameter regulating the soft-matching among the states, determining which hidden states configurations have to be considered sufficiently similar.

The resulting *GTM-SD Activation Mask* kernel (AM-GTM, in short) between trees $\mathbf{x}^1$ and $\mathbf{x}^2$ is defined as follows

$$k_{am-gtm}(\mathbf{x}^1, \mathbf{x}^2) = \sum_{u \in \mathcal{U}_1} \sum_{u' \in \mathcal{U}_2} T_\epsilon(p_u, p_{u'}) \tag{11}$$

where $p_u = \overline{X}(\mathbf{x}^1_u)$ and $p_{u'} = \overline{X}(\mathbf{x}^2_{u'})$ are the posterior mean projections of subtrees $\mathbf{x}^1_u$ and $\mathbf{x}^2_{u'}$ from tree $\mathbf{x}^1$ and $\mathbf{x}^2$, respectively. To demonstrate that the weight function $T_\epsilon(p, p')$ in (10) is a kernel (and so is k_{am-gtm}), we can exploit the definition of Wedland functions [45], that are a family of piecewise polynomial covariance functions with compact support. A simple family of Wendland functions is that based on univariate polynomials and it is characterized by positive definiteness on $\mathbb{R}^d$ with support on the unitary compact. Among the members of this family, we are interested in the following Wendland function

$$\phi_{1,0} = \begin{cases} (1 - r), & \text{if } r \leq 1 \\ 0, & \text{otherwise} \end{cases} \tag{12}$$

that is ensured to be positive definite in $\mathbb{R}$ with support on compact $[0, 1]$ [46]. When r is the Euclidean distance, eq. (12) is equivalent to the weight function $T_\epsilon(p, p')$ apart from the fact that (10) has support on the compact $[0, \epsilon]$. Therefore, $T_\epsilon(p, p')$ defines a positive definite kernel on the ϵ-compact $[0, \epsilon]$, yielding to Gram matrices of increasing sparseness as the radius of the hyperball ϵ approaches zero.

AM-GTM is an adaptive unsupervised kernel since it depends on a topological grouping of the tree structures that is learned by the GTM-SD directly from the data in an unsupervised fashion. Since it is completely oblivious of the task target, kernel fitting can be performed only once for different computational learning tasks, with a positive impact on computational effort. Compared to the AM kernel for the SOM-SD model [13], AM-GTM allows a finer grained exploitation of the map, where the GTM-SD smooth mapping yields to densely populated yet discriminative map by exploiting the continuous latent space. SOM-SD, on the other hand, is based on a discrete lattice and it is thus constrained to represent nodes (subtrees) through a finite set of discrete coordinates. Hence, the discriminative quality of AM-SOM kernel is strongly dependent both upon the size of the SOM-SD lattice, which directly determines the resolution of the kernel, as well as on the choice of the neighborhood parameter ϵ. Too small ϵ values can in fact induce excessive sparsity in the kernel matrix, while too large values may allow too much noise into the kernel. AM-GTM, on the other hand, naturally defines a dense kernel with a limited sensitivity to the choice of the neighborhood parameter.

D. Generative Kernels Computational Complexity

The computational complexity of the generative kernels, and of adaptive kernels in general, is the result of two operations. The former is associated with the inferential process computing the parameters/scores used to transform the input tree in its feature space representation (e.g. map projections for AM-GTM, hidden states assignments for Jaccard, etc). The latter refers to the actual kernel computation based on the feature space representation (e.g. the entries of the Gram matrix that is used by a Support Vector Classifier). The computational complexity of the syntactic kernels in Section II only depends on the second operation, since encoding is implicit in kernel computation. The presence of an encoding phase in generative kernels, on the other hand, typically makes kernel computation not directly dependent on the size of the input tree, as this is transformed into a representation that usually depends on the parametrization of the underlying generative model.

Table I summarizes the worst case complexity for the generative kernels tree kernel introduced earlier in this section as well as for the Fisher tree kernel, considering a multiclass classification task with V classes. The term C denotes the number of hidden Markov states, L is the maximum number of non-empty children in the tree dataset and N_T is the maximum tree size. The complexity of the unibigram Jaccard kernel depends on whether the squared number of hidden states C^2 is smaller than the node number N_T, i.e. $E = \min\{N_T, C^2\}$. The Jaccard kernel seems more efficient than the Fisher kernel

TABLE I
COMPUTATIONAL COMPLEXITY OF THE GENERATIVE KERNELS FOR FEATURE SPACE ENCODING (INFERENCE) AND GRAM MATRIX CALCULATION. FOR THE JACCARD KERNEL, WE REPORT RESULTS FOR THE MOST COMPLEX ENCODING, I.E. UNIBIGRAM (UBI), WHILE VARYING THE GENERATIVE MODEL USED, I.E. IO-BHTMM (IO) OR BHTMM (BU), AND THE TYPE OF INFERENCE ALGORITHM, I.E. VITERBI (V) OR UPWARD-DOWNWARDS (P).

Kernel	Inference	Gram
UBI-BU-V	$O(N_T \cdot C^2 \cdot V)$	$O((E+C) \cdot V)$
UBI-IO-V	$O(N_T \cdot C^2)$	$O(E+C)$
UBI-BU-P	$O(N_T \cdot C^2 \cdot L \cdot V)$	$O((E+C) \cdot V)$
UBI-IO-P	$O(N_T \cdot C^2 \cdot L)$	$O(E+C)$
AM-GTM	$O(N_T \cdot C^2)$	$O(N_T^2)$
Fisher	$O(N_T \cdot C^2 \cdot L \cdot V)$	$O((C^2 \cdot L + C + M \cdot C) \cdot V)$

when dealing with tasks with a non-trivial number of classes V and a large input vocabulary M. The posterior-weighted encodings have an the same inferential cost with respect to the Fisher kernel but have a considerably lower complexity for Gram matrix computation, due to the reduced feature space size. The Viterbi encoding can be computed with an inferential cost that is lower of a factor L with respect to the posterior-weighted encoding: as such, it might prove more adequate for dealing with trees characterized by a large outdegree. The convolutional AM-GTM kernel is the only one whose kernel computation step depends N_T as it computes matchings between substructures. Despite its worst case cost being $O(N_T^2)$, this will occur with negligible probability as it entails all the nodes from the two trees being projected in an hyperball of radius $\leq \epsilon$. The average expected computational cost is, instead, $O(c_\epsilon N_T)$ with c_ϵ being a constant depending on the choice of the neighborhood parameters ϵ.

IV. EXPERIMENTAL EVALUATION

This section provides an experimental assessment of the generative tree kernels discussed in the paper, on 7 publicly available benchmarks on tree-data classification. The benchmarks span a variety of application areas, including parse trees, structured documents, and biochemical data, with different data characteristics (e.g. tree outdegree, number of classes, sample size, size of label vocabulary, etc.).

A. Experimental Setup

Section III has discussed how different Markov state kernels can be obtained by combining alternative choices regarding the underlying generative models, the inference algorithms, the way in which hidden state information is coupled and encoded, as well as the actual kernel function. Table II shows a summarized view of the kernel configurations tested in this experimental assessment, described in terms such modeling choices. In particular, we consider both homogeneous bottom-up (BU) and top-down (TD) HTMM, as well as input driven IO-BHTMM (IO) generative models. For all the generative models under consideration, we explore the impact of a multiset encoding based on both Viterbi (V) and Upwards-Downwards (P) inference. Hidden states information is exploited considering both single state information (UNI, in Table II), as in the unigram representation, as well as

TABLE II
GENERATIVE KERNEL CONFIGURATIONS UNDER TEST AS A FUNCTION OF THE MODELING CHOICES, I.E. GENERATIVE MODELS, HIDDEN STATES FEATURES, , INFERENCE ALGORITHMS AND KERNEL TYPE.

	Model			States		Inference		Kernel		
Configuration	BU	TD	IO	UNI	BI	V	P	J	AM	F
Jaccard-BU-V	✓			✓	✓	✓		✓		
Jaccard-TD-V		✓		✓	✓	✓		✓		
Jaccard-IO-V			✓	✓	✓	✓		✓		
Jaccard-BU-P	✓			✓	✓		✓	✓		
Jaccard-TD-P		✓		✓	✓		✓	✓		
Jaccard-IO-P			✓	✓	✓		✓	✓		
AM-GTM	✓			✓			✓		✓	
Fisher		✓		✓	✓		✓			✓

information from states coupled in a parent-child relationship (BI, in Table II), as in the bigram representation. Hidden state multisets encoding are assessed using the Jaccard kernel (J), while the impact of introducing topological information in the hidden states is assessed by the AM kernel. Note that in Table II this latter kernel is characterized by a BU generative model, as the GTM-SD approach in [22] is fundamentally a constrained BU model, as well as by a UNI state encoding, as no parent-child information is used neither in the map projection nor in kernel computation. Table II also reports a configuration for a Fisher kernel based on the THTMM model as this is a state-of-the-art baseline for generative tree kernels. Clearly, the combinations of the modeling choices in Table II allow for far more kernel configurations. For the sake of compactness, we focus only on those configurations which yield to more discriminant and expressive kernels.

Table III reports the main characteristics of the datasets used for this experimental assessment. The first set of benchmarks concerns the classification of XML formatted documents from two large corpora used in the 2005 and 2006 INEX Competition [6]. These datasets are characterized by a large sample size and by a large number of unbalanced classes; trees are generally shallow, with a large outdegree. Standard splits into training and test sets are available for both datasets [6], where roughly half of the total samples are used for training. The second set of benchmarks concerns the classification of the molecular structure of glycans, that can be represented by rooted trees where nodes stand for mono-saccharides and edges stand for sugar bonds. We consider two datasets from the KEGG/Glycan database [47], referred to as the Leukemia and Cystic data [48]. These benchmarks differs considerably from INEX: the task is binary and a small number of samples is available; trees are small and have a small outdegree. The third set of experiments deals with parse trees representing English propositions from a set of Dow-Jones news articles and associated semantic information. We employ a version of the Propbank dataset [49] introduced by [50], that includes a sample from section 24 of Propbank comprising $7,000$ training trees and $2,000$ validation examples, as well as $6,000$ test samples extracted from section 23 [50]. This benchmark defines a binary classification problem with a very unbalanced class distribution, where the percentage of positive examples in each set is roughly 7%. The latter two benchmarks in Table III pertain to the classification of chemical compounds

TABLE III
CHARACTERISTICS OF THE DATASETS USED IN THE EXPERIMENTAL ANALYSIS

Name	# Trees	Classes	Outdegree	# Labels
INEX 2005 [6]	9361	11	32	366
INEX 2006 [6]	12107	18	66	65
Leukemia [48]	442	2	3	57
Cystic [48]	160	2	3	29
Propbank [50]	15000	2	15	6654
CPDB [51]	9672	2	10	23
AIDS [51]	53771	2	22	29

originally represented as more general classes of graphs and transformed into tree structures by means of the approach described by [51]. In practice, each graph is transformed into a tree, such that each direct subtree of the root represents the visit that can be performed from a vertex of the graph up to a certain depth D (set to 6 for the purpose of this analysis). These benchmarks allow to test kernel performance on trees characterized by very rich structural contexts originating from the original graph information. For the purposes of generative model training and inference, we have split each transformed tree into a forest comprising all direct subtrees of the root. The final tree encodings are computed on the aggregated trees by putting together the information from the single subtrees of the root node, yielding to a classification for the full graph.

The generative tree kernels are assessed in terms of the tradeoff between efficacy, measured in terms of classification accuracy and related metrics, and computational efficiency, i.e. time required to complete inference and kernel computation steps. Their predictive performance is compared with that of syntactic tree kernels in literature (reviewed in Section II), such as ST [26], SST [11], PT [29], subpath [31] and the Elastic Tree (ET) kernels [27], [28]. These have been chosen as a relevant sample of the available tree kernels due to their popularity, state of the art performances and availability as code: see [32] for an experimental comparison of several syntactic tree kernels (though limited to low dimensionality datasets and binary classification).

Different configurations of the generative models have been assessed by varying the number of hidden states C as follows: $C \in \{6, 8, 10\}$ for the homogenous models, $C \in \{8, 10, 16\}$ for the input-driven IO and $C \in \{81, 100, 225, 400\}$ for GTM-SD. In addition, for the Glycans tasks, we have also tested smaller state spaces, such as $C = 2$ and $C = 4$ for the homogenous and IO models, respectively, and $C = 49$ for GTM-SD. The number of tested hidden states has been determined following the guidelines in the original paper for the various generative models, i.e. [21], [22]. The AM-GTM kernel is also evaluated with respect to the choice of the neighborhood metaparameter ϵ, whose values are allowed to vary in $\{0.05, 0.1, 0.2\}$. These values ensure that the AM-GTM neighborhood covers a map area that is comparable with the coverage suggested by [30] for the original AM kernel over a SOM-SD map (i.e. about $1-2\%$ of the lattice).

Trials have been repeated multiple times for each configuration of the generative models, each time using different random initializations for the models distributions, i.e. 5 repetitions for INEX and Propbank data. For the Glycans, CPDB and

TABLE IV
TEST CLASSIFICATION ACCURACY (%) ON THE INEX 2005, INEX 2006, CPDB, AIDS AND PROPBANK BENCHMARKS. FOR PROPBANK, THE F1-SCORE IS THE REFERENCE METRIC TO BE USED DUE TO THE UNBALANCED NATURE OF THE DATA AS WELL AS FOR ALLOWING COMPARABILITY WITH LITERATURE RESULTS (ACCURACY IS ALSO REPORTED). THE MODEL PARAMETERS SELECTED IN CV ARE REPORTED IN SQUARED BRACKETS FOR THE GENERATIVE KERNELS. THE BEST TEST PERFORMANCE IS HIGHLIGHTED IN BOLD.
† VALUES OBTAINED ON APPROXIMATED SETTING AS DESCRIBED IN THE TEXT.

Kernel	Dataset					
	INEX 2005	INEX 2006	Propbank		CPDB	AIDS
	Acc (%)	Acc (%)	Acc (%)	F1	Acc (%)	Acc (%)
Jaccard-TD-V	93.40 (1.5)	44.38 (0.2)	91.00	0.577	61.72 (4.26)	78.30 (3.67)
Configuration	*[$C = 6$]*	*[$C = 8$]*	*[$C = 10$]*		*[$C = 10$]*	*[$C = 10$]*
Jaccard-BU-V	94.22 (0.9)	44.53 (0.3)	89.52	0.567	63.94 (4.88)	80.76 (4.42)
Configuration	*[$C = 8$]*	*[$C = 6$]*	*[$C = 10$]*		*[$C = 10$]*	*[$C = 10$]*
Jaccard-IO-V	95.66 (0.2)	41.51 (1.0)	91.04	0.617	68.91 (5.13)	78.84 (3.73)
Configuration	*[$C = 8$]*	*[$C = 8$]*	*[$C = 16$]*		*[$C = 8$]*	*[$C = 10$]*
Jaccard-TD-P	96.53 (0.1)	44.49 (0.3)	92.96	0.677	66.67 (4.53)	78.30 (3.04)
Configuration	*[$C = 8$]*	*[$C = 10$]*	*[$C = 10$]*		*[$C = 10$]*	*[$C = 6$]*
Jaccard-BU-P	96.12 (0.4)	**45.06 (0.2)**	93.06	0.697	66.82 (7.03)	78.44 (3.40)
Configuration	*[$C = 8$]*	*[$C = 10$]*	*[$C = 6$]*		*[$C = 10$]*	*[$C = 10$]*
Jaccard-IO-P	96.36 (0.1)	42.03 (1.3)	93.03	0.645	69.03 (3.35)	79.17 (3.46)
Configuration	*[$C = 10$]*	*[$C = 10$]*	*[$C = 16$]*		*[$C = 16$]*	*[$C = 16$]*
AM-GTM	96.71 (0.1)	43.71 (0.6)	91.16	**0.712**	75.44 (3.74)	81.33 (3.89)
Configuration	*[$C = 225$, $\epsilon = 0.05$]*	*[$C = 400$, $\epsilon = 0.2$]*	*[$C = 100$, $\epsilon = 0.2$]*		*[$C = 225$, $\epsilon = 0.05$]*	*[$C = 225$, $\epsilon = 0.05$]*
Fisher	96.82 (0.1)	39.47 (0.8)	90.85	0.542	68.87 (3.41)	76.65 (3.45)
Configuration	*[$C = 8$]*	*[$C = 6$]*	*[$C = 6$]*		*[$C = 8$]*	*[$C = 8$]*
ST	88.73	32.02	93.40	0.517	75.29 (1.64)	**82.00 (2.00)**
SST	88.79	40.41	94.28	0.542	**76.59 (2.16)**	80.17 (1.53)
PT	**97.04**	41.13	93.00	0.516	71.64 (5.88) †	76.51 (3.35) †
Subpath	82.68	39.88	92.52	0.518	75.86	71.9216

AIDS tasks, on the other hand, results have been obtained by a stratified 10-fold CV using the available standard partitions [9], [51]. Node emission has been modeled by a multinomial distribution and its initialization is kept fixed, by using the prior distribution of labels estimated solely on the training trees. The tree classifiers have been realized through support vector classification, using the publicly available LIBSVM [52] software. A cross-validation (CV) procedure using validation data external to the test set has been applied to select the number of hidden states C, the AM-GTM metaparameter ϵ as well as the value of the SVM cost parameter C_{svm} from the following set of values: $0.001, 0.01, 0.1, 1, 10, 100, 1000$. We have used a 3-fold-CV applied to the training set for all datasets with the exception of Propbank, as it comes with a standard validation set.

B. Experimental Results

Table IV and Table V confront the predictive performance of the generative and syntactic tree kernels under consideration on the seven benchmarks. The first table reports the results for those datasets that are assessed in literature in terms of classification accuracy. The only exception is Propbank whose performance is to be assessed in terms of F1-score due to class unbalance in the problem (accuracy is also reported although not used for model selection). Table V reports results for Leukemia and Cystic which, in literature, are assessed in terms of the area under the ROC curve (AUC) (due to size and class balancing): nevertheless we also report accuracy to provide a uniform metric across all datasets.

The first thing to note is that the ET kernel has been tested on all the datasets in Table IV but it could not complete kernel computation due to exceeding the maximum allowed computing time (i.e. 7 days for computing at least the kernel on the training set), which happened for INEX2005, CPDB and AIDS, or due to exceeding the memory resources, which happened for INEX2006 and Propbank. Note that we have used a parallel Scala implementation of ET[1] associated to the work by [28] and that the code was run on a server comprising 48 cores and 128Gb. Therefore, the lack of results for this kernel is not due to tight resource constraints but rather to its high computational requirements (its complexity is cubic in tree size) which makes it less scalable to medium-large tree datasets, such as those in Table IV. On the other hand, ET seamlessly worked for the smaller dimensional datasets in Table V.

Table IV shows that on INEX 2005, the PT kernel achieves the best classification accuracy overall, but the Fisher and AM-GTM kernels achieve a comparable performance. The PT kernel is expressive but has computational complexity that scales quadratically with respect to the tree size and cubically with the outdegree L which, for the INEX 2005 trees, is $L = 32$. On INEX 2006, on the other hand, the best kernel results in literature (including PT) were limited to less than 42% accuracy, whereas the Jaccard and AM-GTM kernels yield to notably higher accuracies. In particular, the Jaccard-BU-P has the state of the art result with an accuracy of 45.06% (previously, was 42.62% in [9]). Interestingly, on INEX 2006, the IO model performs worse than BU and TD, while the opposite happens on the INEX 2005 data. This might be due to the fact that input labels convey little discriminative information in the former dataset, thus depleting the advantage of having an input-driven dynamics. Also, on the INEX 2006 benchmark there seems to be little difference between Viterbi and Upwards-Downwards encodings, suggesting the the hidden state space of the generative model is organized in a few

[1]http://marcocuturi.net/MT.html

specialized hidden states capturing information on a number of discriminative substructures, while the remainder of the states does not encode discriminative structural information. On Propbank, the AM-GTM kernel outperforms all the other kernels with a considerable increase in the F1 score, especially with respect to the syntactic kernels. Note that classification accuracy, on this task, leads to misleading results as a dumb model returning always the majority class prediction would achieve 93% accuracy, hence the need of using F1 score for model selection and assessment. The Jaccard kernels using the Upwards-Downwards encoding obtain the second best performance, well ahead of the syntactic kernels, in particular when using an homogenous generative model.

As regards CPDB and AIDS, Table IV reports the results for the best model in the external 10-fold CV as these are not significantly different from that of the model selected the nested 3-fold CV. Differently from results on previous benchmarks, syntactic kernels seem to be have a better performance with respect to the majority of the adaptive kernels on these data. This can be the result of the very specific nature of these two datasets. We recall that the generative models underlying the adaptive kernels are not trained on the population of visit-trees for the original graphs, but rather on the forests originating from the direct child subtrees of their roots. The same approach had been applied to the PT kernel since the large outdegree of the original visit-trees induced out-of-memory errors on the implementation available by the authors of [29][2]. Despite such an approximation, the AM-GTM kernel achieves competitive results also with respect to the syntactic kernels, yielding to accuracies that are not statistically inferior to that of the best syntactic kernels (i.e. SST for CPDB and ST for AIDS). With the same approximation, the PT kernel achieves consistently lower performances, in particular on the AIDS data.

Focusing on the results of the adaptive kernels, AM-GTM achieves, by far, the best performance on the CPDB data followed by the Jaccard-IO models. For the latter models, there seems to be a minor performance difference between Upwards-Downwards and Viterbi encoding, whereas the homogenous TD and BU models achieve consistently higher performances when posterior-unibigrams are used. This suggest that, on the one hand, there is a considerable amount of structural information that can be captured only by including the contribution of all the hidden states (note that the best performing kernel, AM-GTM, is also based on an Upwards-Downwards encoding). On the other hand, the Jaccard-IO performance suggests that the input labels are providing significant discriminative information that can be well exploited by the input-driven models. Performance on the AIDS data appears significantly more homogeneous across the different generative models and encoding schemes. The AM-GTM and Jaccard-BU kernels based on Viterbi obtain the best results, although all the Jaccard-based kernels yield to competitive accuracies.

At last, we consider the predictive performance on the Glycans datasets in Table V. For generative kernels we report both the test set performance of the model showing the best results on the 10-fold CV as well for the model selected on the nested 3-fold CV (on the training set) [9] used to determine the values of the hyperparameters. For the Leukemia dataset these two results are almost overlapping, while for the Cystic data there is a neat difference. This might be due to the small sample size of the Cystic dataset, which makes the inner 3-fold CV more noisy as it leaves the generative kernels with very few training samples. Overall, the results of the proposed generative kernels appear competitive with the performances in literature both in terms of AUC-ROC and accuracy. On Leukemia, generative kernels generally outperform syntactic kernels on the AUC-ROC, with the exception of the subpath and PT kernels. In particular, all Jaccard kernels obtain an AUC that is comparable (and in some cases higher) to that achieved by the PT kernel. On the Cystic data, the Fisher

TABLE V
TEST PERFORMANCE, AS ACCURACY (%) AND AUC-ROC (AUC), ON GLYCANS AVERAGED OVER THE 10-FOLDS (DEVIATION IS IN BRACKETS). RESULTS ARE REPORTED FOR BOTH THE BEST MODEL IN 10-FOLD CV (10-CV) AND FOR A NESTED 3-FOLD CV FOR MODEL SELECTION (NESTED). SELECTED CONFIGURATIONS ARE REPORTED IN PARENTHESIS FOR THE GENERATIVE KERNELS.

Kernel	Leukemia		Cystic	
	%	AUC	%	AUC
Jaccard-TD-V				
10-CV	91.85 (3.75)	.968 (.021)	66.88 (12.52)	.759 (.171)
	$[C = 6]$		$[C = 4]$	
Nested	92.53 (3.56)	.966 (.028)	64.38 (14.75)	.740 (.165)
	$[C = 8]$		$[C = 6]$	
Jaccard-BU-V				
10-CV	92.3 (3.6)	.966 (.035)	69.38 (8.56)	.796 (.086)
	$[C = 8]$		$[C = 10]$	
Nested	92.3 (3.6)	.966 (.035)	70.0 (1.54)	.751 (.139)
	$[C = 8]$		$[C = 4]$	
Jaccard-IO-V				
10-CV	91.85 (4.32)	.953 (.047)	71.25 (11.49)	.751 (.201)
	$[C = 16]$		$[C = 4]$	
Nested	91.85 (4.32)	.953 (.047)	56.25 (16.4)	.690 (.193)
	$[C = 16]$		$[C = 16]$	
Jaccard-TD-P				
10-CV	92.75 (3.99)	**.972 (.025)**	70.0 (13.42)	.781 (.150)
	$[C = 6]$		$[C = 10]$	
Nested	91.62 (3.42)	.962 (.037)	67.50 (7.1)	.725 (.143)
	$[C = 10]$		$[C = 8]$	
Jaccard-BU-P				
10-CV	92.29 (4.05)	.967 (.023)	73.13 (12.16)	.826 (.104)
	$[C = 8]$		$[C = 8]$	
Nested	92.74 (3.99)	.966 (.027)	73.13 (12.16)	.826 (.104)
	$[C = 10]$		$[C = 8]$	
Jaccard-IO-P				
10-CV	**94.33 (3.6)**	.971 (.028)	75.6250 (1.81)	.826 (.118)
	$[C = 16]$		$[C = 8]$	
Nested	94.33 (3.6)	.971 (.028)	75.6250 (1.81)	.826 (.118)
	$[C = 16]$		$[C = 8]$	
AM-GTM				
10-CV	92.98 (4.09)	.954 (.04)	73.75 (8.74)	.843 (.141)
	$[C = 49, \epsilon = .1]$		$[C = 225, \epsilon = .1]$	
Nested	92.53 (3.04)	.952 (.042)	72.50 (11.49)	.806 (.095)
	$[C = 225, \epsilon = .05]$		$[C = 81, \epsilon = .2]$	
Fisher				
10-CV	92.98 (3.31)	.957 (.028)	73.13 (13.84)	**.850 (.125)**
	$[C = 10]$		$[C = 6]$	
Nested	91.39 (3.37)	.930 (.062)	74.38 (11.20)	.819 (.118)
	$[C = 8]$		$[C = 8]$	
ST	92.07	.961	76.25	.798
SST	90.71	.933	58.13	.696
PT	93.44	.967	78.13	.823
Subpath	94.11	.971	**80.63**	**.850**
Elastic	92.89	.925	76.25	.763

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

kernel obtains the best AUC together with the subpath kernel, but the AM-GTM again yields to a comparable score. On the accuracy side, instead, the syntactic PT and subpath kernels have consistently higher performances. Note that on Cystic the Upwards-Downwards encoding seems more effective that the Viterbi encoding both in AUC and accuracy.

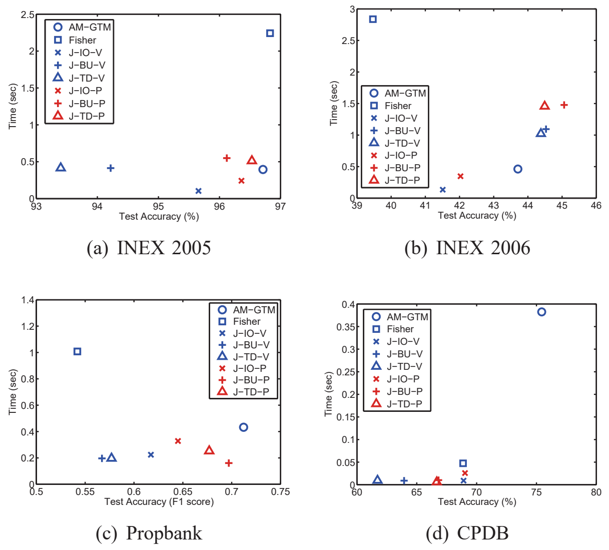
Overall, the indication provided by the experimental assessment is that no syntactic kernel seems effective on all datasets; rather each syntactic kernel can perform well on a specific dataset and badly on the others, depending on how well its predefined structure similarity function matches the characteristics of the dataset. This is the case for the subpath kernel, which is very effective on the vertical, almost sequence-like structures of the Glycans datasets, but fails on richer structures such as in INEX, Propbank and AIDS. Similarly, none of the adaptive kernels taken singularly exceed the performance of all the syntactic kernels but they show a good performance on the majority of the datasets, with significantly lower performances on CPDB and Cystic data for the Jaccard-based kernels. In the AM-GTM case, on the other hand, one can note a generally very high predictive performance, in line with the state-of-the-art. As we will discuss more in detail in the next section, the adaptive kernels are also characterized by a competitive tradeoff between computational complexity and predictive performance, whereas the most expressive syntactic kernels (in their state-of-the-art implementations), such as PT and elastic kernels, show problems in providing results due to exceeding memory limits or reasonable execution time caps.

C. Computational cost

We conclude by evaluating the tradeoff between kernel predictive accuracy and its computational efficiency, by providing an empirical assessment of the average computational effort required by inference and kernel computation on a test tree of the INEX, Propbank and CPDB datasets. Results have been obtained by Matlab implementations running on a Intel I5 Quad-core at 2.7 GHz CPU equipped with 4GBytes of RAM and they are reported in Fig. 3. It shows accuracy-efficiency plots for the CV-selected configurations where top-most areas denote models with an higher computational cost, while right-most areas denote higher predictive accuracy. As expected, the Fisher kernel yields to the worse computational effort on almost all the datasets, with as much as 2.242 seconds required, on average, to encode and compute the kernel on an INEX 2005 test tree. At the same time, the Fisher kernel attains the lowest accuracy among the generative kernels on the INEX 2006 and Propbank datasets. The AM-GTM is characterized by the best tradeoff between effort and accuracy on the first three datasets, requiring as little as 0.395 seconds on an INEX 2005 tree while yielding to accuracy results comparable, when not superior to the other kernel methods. On the other hand, AM-GTM has the worse time-efficiency on the graph benchmarks due to its computation being dominated by the square of the tree size, while the graph-induced trees are composed by a large number of nodes.

The lowest computational cost is attained, in general, by the Jaccard-IO thanks to the compactness of its kernel feature space, e.g.requiring only 0.104 seconds on an INEX 2005 tree, and even less on the INEX 2006 and Propbank data. The computational cost of Jaccard-BU and Jaccard-TD is in general higher, but comparable to that of Jaccard-IO. The only exception is on INEX 2006 where the large class number and the high outdegree result in a neat increase of its feature space size, as compared to Jaccard-IO. Note that on Propbank the Jaccard-IO-V and Jaccard-BU-V outperform the syntactic kernels while maintaining the cost for kernel prediction to 0.224 and 0.196 seconds per test tree, respectively. The difference between the computational efficiency of posterior and Viterbi encodings is minor: nevertheless, it the computational cost of Viterbi kernels can be reduced by an efficient implementation exploiting the sparsity of its encoding.

Generative and syntactic kernel can be confronted on their computational complexity. Table VI summarizes the complexities discussed in Section II and III-D by focusing on the cost of computing the kernel between 2 trees, considering as relevant complexity terms the maximum tree size N_T, the maximum outdegree L and the number of hidden states C (i.e. we have simplified all the irrelevant terms in Table I for clarity). Note how syntactic kernels have always superlinear complexity with respect to tree size N_T, where the most effective and expressive kernels in previous experiments are at



(a) INEX 2005 (b) INEX 2006 (c) Propbank (d) CPDB

Fig. 3. Accuracy-efficiency plot for the INEX 2005, INEX 2006, Propbank and CPDB datasets: on the x-axis is the test accuracy or F1 score on the CV-selected results, while the y-axis shows the corresponding average time needed to perform inference and kernel computation on a single test tree.

TABLE VI
COMPUTATIONAL COMPLEXITY OF TREE KERNEL COMPUTATION WITH RESPECT TO TREE SIZE N_T, OUTDEGREE L AND HIDDEN STATE NUMBER C (FOR GENERATIVE KERNEL ONLY).

Kernel	Complexity	Kernel	Complexity
Jaccard-UBI-V	$O(N_T \cdot C^2)$	SST	$O(N_T^2)$
Jaccard-UBI-P	$O(N_T \cdot C^2 \cdot L)$	PT	$O(N_T^2 \cdot L_T^3)$
AM-GTM	$O(N_T \cdot C^2)$	Subpath	$O(N_T^2)$
Fisher	$O(N_T \cdot C^2 \cdot L)$	Elastic	$O(N_T^3)$
ST	$O(N_T \log N_T)$		

least quadratic. In particular PT and elastic kernels are cubic with respect to outdegree and tree size, respectively, which, in some cases, can result in exceedingly long kernel computation times. The proposed generative kernels have, in general, linear complexity in N_T and the quadratic dependance on C does not seem impacting for the C values in the experimentation.

V. Discussion and Conclusions

Kernel methods offer a modular approach to build learning and pattern recognition systems for structured data. The ability to learn the kernel function from data becomes critical when dealing with such complex, non-standard information, as sufficient background knowledge might not be available to hand-code the kernel or we might simply lack a proper formalization of what a *good* structural similarity is. In this paper, we have proposed a solid methodology for the design of adaptive generative tree kernels that exploit the summarization properties of hidden states in hidden Markov models for trees.

We have introduced a compact, and yet discriminative, feature space encoding for trees, based on the concept of hidden state multisets. Such multiset representation allows to intuitively control the amount of *syntactical* information that is injected into the kernel, e.g. by measuring the co-occurrence of hidden states in parent-child relationships, while controlling the tradeoff with computational complexity. A generative Jaccard kernel has been defined on the top of the multiset encoding and it has been applied in combination to different hidden tree Markov models and tree encoding schemes to show its generality. Nevertheless, the proposed approach is much more general, as it can be employed to define generative kernels on the top of any probabilistic model using categorical latent variables, which easily extends the classes of data processable with the kernel.

The Jaccard kernel takes an hard-matching approach where structural similarity is measured based on the overlap in the hidden states distributions, thus implicitly discarding the fact that two different states might encode very similar information. We have shown how a topology induced on the Markov states by a GTM-SD [22] can be exploited to circumvent such limitations, by deriving a *convolutional* generative kernel capable of computing direct matches between substructures. This kernel defines a feature space encoding where trees are represented by their activation fingerprints on continuous topographic maps, which allow a form of computationally effective soft-matching between hidden states.

We have carried out an in-depth empirical assessment of the proposed generative kernels with a comparative analysis covering state-of-the-art generative, adaptive and syntactical tree kernels, focusing in particular on the tradeoff between predictive performance and computational efficiency. Several application domains and associated data types have also been taken into consideration, including the classification of documents, propositions and bio-molecular data. These results are intended to provide a guideline for selecting the most adequate kernel configuration for a large variety of application areas. Overall, the generative kernels taken into consideration have shown a competitive predictive performance with respect to syntactic kernels in literature coupled with contained computational requirements. The predictive performance of the generative kernels seems to generalize very well across radically different applications, whereas syntactic kernels tend to be very specific, with task-dependent performances. This is not surprising, given the adaptive nature of the generative kernels, that can learn the task-specific similarity metrics directly from the data.

The results show that the AM-GTM kernel has by far the best performance among the proposed generative kernels, also achieving the best results in literature on the INEX 2006 and Propbank [50] datasets, considerably increasing the classification performance with respect to the previous top-scoring methods. Interestingly, AM-GTM learns a truly unsupervised metric which is completely oblivious of the computational learning task, hence allowing its application to different tasks without the need of retraining the metric. As concerns the Jaccard kernels, the Upwards-Downwards approach yields to more discriminative encodings with respect to Viterbi inference, although a smart exploitation of the sparsity of the latter encoding is expected to yield to faster kernel computation routines. When compared to the Fisher kernel, Jaccard-BU and Jaccard-IO yield to a superior predictive performance that pairs with a considerable contraction of the induced feature space.

Concluding, the paper proposes a novel family of methods for building effective and computationally efficient adaptive kernels by mining the state space of latent variable generative models. With respect to the marginalized kernel solutions in literature, the proposed methodology allows to exploit the statistical features of a generative model trained on structured samples to yield to a completely data-driven structure similarity metric, rather than using a generative model only to generate the substructure features on which hard syntactical matching is then performed to measure structure similarity. In particular, the proposed approach yields to a truly adaptive kernel where the structural similarity metric is induced from the distribution over the full population of training structures acquired by the underlying generative model. The collection of adaptive kernels that can be derived from our proposal provides a rich set of tools for tree learning, reducing the burden of a syntactical definition for structure similarity, and allowing to chose the most suitable solution according to the sought tradeoff between efficacy and efficiency for each task.

Acknowledgment

The authors would like to thank Giovanni da San Martino for having provided the Glycans and Propbank experimental data and Nicoló Navarin for the CPDB and AIDS datasets. This work is partially supported by the Italian Ministry of Education, University, and Research (MIUR) under project SIR 2014 LIST-IT (grant n. RBSI14STDE).

References

[1] A. Severyn and A. Moschitti, “Structural relationships for large-scale learning of answer re-ranking,” in *SIGIR*, 2012, pp. 741–750.

[2] S. Menchetti, F. Costa, P. Frasconi, and M. Pontil, "Wide coverage natural language processing using kernel methods and neural networks for structured data," *Pattern Recognition Letters*, vol. 26, no. 12, pp. 1896–1906, 2005.

[3] M. Diligenti, P. Frasconi, and M. Gori, "Hidden tree markov models for document image classification," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 25, no. 4, pp. 519–523, 2003.

[4] K. Rieck, T. Krueger, U. Brefeld, and K.-R. Müller, "Approximate tree kernels," *Journal of Machine Learning Research*, vol. 11, pp. 555–580, 2010.

[5] D. Kimura and H. Kashima, "Fast computation of subpath kernel for trees," in *Proc. of the 29th International Conference on Machine Learning (ICML '12)*. New York, NY, USA: Omnipress, 2012, pp. 393–400.

[6] L. Denoyer and P. Gallinari, "Report on the XML mining track at INEX 2005 and INEX 2006: categorization and clustering of XML documents," *SIGIR Forum*, vol. 41, no. 1, pp. 79–90, 2007.

[7] C. Bockermann, M. Apel, and M. Meier, "Learning SQL for database intrusion detection using context-sensitive modelling," in *Proc. of the 6th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, ser. DIMVA '09. Springer, 2009, pp. 196–205.

[8] A. Micheli, A. Sperduti, and A. Starita, "An introduction to recursive neural networks and kernel methods for cheminformatics," *Current pharmaceutical design*, vol. 13, no. 14, pp. 1469–1495, 2007.

[9] C. Gallicchio and A. Micheli, "Tree echo state networks," *Neurocomputing*, vol. 101, pp. 319–337, 2013.

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

[11] M. Collins and N. Duffy, "New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron," in *Proc. of the 40th Annual Meeting on Assoc. for Comput. Ling.*, 2002, pp. 263–270.

[12] J. Suzuki and H. Isozaki, "Sequence and tree kernels with statistical feature mining," in *Advances in Neural Information Processing Systems 18*. MIT Press, 2006, pp. 1321–1328.

[13] F. Aiolli, G. D. S. Martino, M. Hagenbuchner, and A. Sperduti, "Learning nonsparse kernels by self-organizing maps for structured data," *IEEE Transactions on Neural Networks*, vol. 20, no. 12, pp. 1938–1949, 2009.

[14] D. Pighin and A. Moschitti, "Reverse engineering of tree kernel feature spaces," in *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, 2009, pp. 111–120.

[15] T. Jaakkola and D. Haussler, "Exploiting generative models in discriminative classifiers," *Advances in neural information processing systems*, pp. 487–493, 1999.

[16] L. Nicotra, A. Micheli, and A. Starita, "Fisher kernel for tree structured data," in *Proc. of the 2004 Int. Joint Conf. on Neural Netw.*, vol. 3, july 2004, pp. 1917 – 1922.

[17] L. Denoyer and P. Gallinari, "Bayesian network model for semi-structured document classification," *Inf. Process. Manage.*, vol. 40, no. 5, pp. 807–827, 2004.

[18] H. Kashima, K. Tsuda, and A. Inokuchi, "Marginalized kernels between labeled graphs," in *Proceedings of the Twentieth International Conference on Machine Learning*. AAAI Press, 2003, pp. 321–328.

[19] P. Mahé and J.-P. Vert, "Graph kernels based on tree patterns for molecules," *Machine Learning*, vol. 75, no. 1, pp. 3–35, 2009.

[20] D. Bacciu, A. Micheli, and A. Sperduti, "Compositional generative mapping for tree-structured data - part I: Bottom-up probabilistic modeling of trees," *IEEE Trans. Neural Netw. Learning Syst.*, vol. 23, no. 12, pp. 1987–2002, 2012.

[21] ——, "An input-output hidden Markov model for tree transductions," *Neurocomputing*, vol. 112, pp. 34–46, 2013.

[22] ——, "Compositional generative mapping for tree-structured data - part II: Topographic projection model," *IEEE Trans. Neural Netw. Learning Syst.*, vol. 24, no. 2, pp. 231–247, 2013.

[23] D. Bacciu, A. Micheli, and S. Alessandro, "Adaptive tree kernel by multinomial generative topographic mapping," in *Proc. of the 2011 IEEE International Joint Conference on Neural Networks (IJCNN'11)*. IEEE, 2011, pp. 1651–1658.

[24] ——, "A generative multiset kernel for structured data," in *Proc. of the 2012 International Conference on Artificial Neural Networks (ICANN'12)*, ser. LNCS, vol. 7552. Springer, 2012, pp. 57–64.

[25] T. Gärtner, "A survey of kernels for structured data," *SIGKDD Explorations*, vol. 5, no. 1, pp. 49–58, 2003.

[26] S. V. N. Vishwanathan and A. J. Smola, "Fast kernels for string and tree matching," in *Advances in Neural Information Processing Systems 15*. MIT Press, 2003, Article, pp. 569–576.

[27] H. Kashima and T. Koyanagi, "Kernels for semi-structured data," in *Proceedings of the Nineteenth International Conference on Machine Learning*, ser. ICML '02. Morgan Kaufmann, 2002, pp. 291–298.

[28] K. Shin, M. Cuturi, and T. Kuboyama, "Mapping kernels for trees," in *Proc. of the 28th International Conference on Machine Learning (ICML-11)*. New York, NY, USA: ACM, June 2011, pp. 961–968.

[29] A. Moschitti, "Efficient convolution kernels for dependency and constituent syntactic trees," in *Proc. of the 17th European conference on Machine Learning*, ser. ECML'06. Springer, 2006, pp. 318–329.

[30] F. Aiolli, G. Da San Martino, and A. Sperduti, "Route kernels for trees," in *Proc. of the 26th Annual International Conference on Machine Learning (ICML '09)*. ACM, 2009, pp. 17–24.

[31] D. Kimura, T. Kuboyama, T. Shibuya, and H. Kashima, "A subpath kernel for rooted unordered trees," in *Advances in Knowledge Discovery and Data Mining*. Springer, 2011, pp. 62–74.

[32] K. Shin and T. Kuboyama, *A Comprehensive Study of Tree Kernels*. Springer International Publishing, 2014, pp. 337–351.

[33] L. Nicotra and A. Micheli, "Modeling adaptive kernels from probabilistic phylogenetic trees," *Artificial Intelligence in Medicine*, vol. 45, no. 2-3, pp. 125–134, 2009.

[34] T. Jebara, R. Kondor, and A. Howard, "Probability product kernels," *The Journal of Machine Learning Research*, vol. 5, pp. 819–844, 2004.

[35] K. Tsuda, T. Kin, and K. Asai, "Marginalized kernels for biological sequences," *Bioinformatics*, vol. 18, no. suppl 1, pp. S268–S275, 2002.

[36] P. Mahé, N. Ueda, T. Akutsu, J.-L. Perret, and J.-P. Vert, "Extensions of marginalized graph kernels," in *Proceedings of the Twenty-first International Conference on Machine Learning*, ser. ICML '04. New York, NY, USA: ACM, 2004, pp. 70–77.

[37] S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt, "Graph kernels," *The Journal of Machine Learning Research*, vol. 11, pp. 1201–1242, 2010.

[38] K. M. Borgwardt and H.-P. Kriegel, "Shortest-path kernels on graphs," in *Proceedings of the Fifth IEEE International Conference on Data Mining*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 74–81.

[39] N. Shervashidze, P. Schweitzer, E. J. Van Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-lehman graph kernels," *The Journal of Machine Learning Research*, vol. 12, pp. 2539–2561, 2011.

[40] M. Hagenbuchner, A. Sperduti, A. Tsoi *et al.*, "A self-organizing map for adaptive processing of structured data," *IEEE Trans. Neural Networks*, vol. 14, no. 3, pp. 491–505, 2003.

[41] L. R. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," *Readings in Speech Recognition*, pp. 267–296, 1990.

[42] P. Jaccard, "The distribution of the flora in the alpine zone.1," *New Phytologist*, vol. 11, no. 2, pp. 37–50, 1912.

[43] T. Hofmann, "Probabilistic latent semantic indexing," in *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 1999, pp. 50–57.

[44] C. Bishop, M. Svensén, and C. Williams, "GTM: The generative topographic mapping," *Neural Comput.*, vol. 10, no. 1, pp. 215–234, 1998.

[45] H. Wendland, "Piecewise polynomial, positive definite and compactly supported radial functions of minimal degree," *Advances in computational Mathematics*, vol. 4, no. 1, pp. 389–396, 1995.

[46] M. S. Floater and A. Iske, "Multistep scattered data interpolation using compactly supported radial basis functions," *J. Comput. Appl. Math.*, vol. 73, no. 1-2, pp. 65–78, Oct. 1996.

[47] M. Kanehisa, S. Goto, S. Kawashima, Y. Okuno, and M. Hattori, "The kegg resource for deciphering the genome," *Nucleic Acids Research*, vol. 32, no. suppl 1, pp. D277–D280, 2004.

[48] L. Li, W.-K. Ching, T. Yamaguchi, and K. Aoki-Kinoshita, "A weighted q-gram method for glycan structure classification," *BMC Bioinformatics*, vol. 11, no. Suppl 1, p. S33, 2010.

[49] P. Kingsbury and M. Palmer, "From Treebank to PropBank," in *LREC*, 2002, pp. 1989–1993.

[50] F. Aiolli, G. Da San Martino, and A. Sperduti, "Extending tree kernels with topological information," in *Proc. of the 21th international conference on Artificial neural networks*, ser. LNCS. Springer, 2011, pp. 142–149.

[51] G. Da San Martino, N. Navarin, and A. Sperduti, "A tree-based kernel for graphs," in *Proc. of the 12th SIAM Int. Conf. on Data Mining*, 2012, pp. 975–986.

[52] C.-C. Chang and C.-J. Lin, *LIBSVM: a library for support vector machines*, 2001, http://www.csie.ntu.edu.tw/∼cjlin/libsvm.