2015 IEEE Symposium Series on Computational Intelligence

Multiple Graph-Kernel Learning

Fabio Aiolli, Michele Donini, Nicolò Navarin and Alessandro Sperduti
Department of Mathematics
University of Padova
via trieste 63, Padova, Italy
Email:{aiolli,mdonini,nnavarin,sperduti}@math.unipd.it

***Abstract*—Kernels for structures, including graphs, generally suffer of the diagonally dominant gram matrix issue, the effect by which the number of sub-structures, or features, shared between instances are very few with respect to those shared by an instance with itself. A parametric rule is typically used to reduce the weights of largest (more complex) sub-structures. The particular rule which is adopted is in fact a strong external bias that may strongly affect the resulting predictive performance. Thus, in principle, the applied rule should be validated in addition to the other hyper-parameters of the kernel. Nevertheless, for the majority of graph kernels proposed in literature, the parameters of the weighting rule are fixed a priori.**

The contribution of this paper is two-fold. Firstly, we propose a Multiple Kernel Learning (MKL) approach to learn different weights of different bunches of features which are grouped by complexity. Secondly, we define a notion of kernel complexity, namely Kernel Spectral Complexity, and we show how this complexity relates to the well-known Empirical Rademacher Complexity for a natural class of functions which include SVM.

The proposed approach is applied to a recently defined graph kernel and evaluated on several real-world datasets. The obtained results show that our approach outperforms the original kernel on all the considered tasks.

I. Introduction

Historically, the machine learning community has been focused on data represented as vectors, because they are a flexible and mathematically convenient representation. However, in the last few years, there has been an increasing interest for data represented in a structured form. Indeed, in many application domains, it is easy to define a representation for the data as a structure. For example, XML files have a natural representation as trees, as well as the tree of a sentence in natural language processing. Unfortunately, it is not straightforward to define vectorial representations that are general enough and that do not force a loss of information. Graphs are a very convenient way to store information, and they can encode concepts and relationships from very different application domains in a natural way. Recently, different ambitious projects have started with the aim to build a graph database that encodes (some of the) human knowledge, such as the Google knowledge graph [23] or NELL [19]. In Chemoinformatics a chemical compound can be easily represented as a graph, with nodes that represent atoms and edges that represent bonds between atoms. In computer vision, recent advances in image segmentation and semantic parsing gave a viable way to construct a graph representation of an image.

Kernel methods are a class of learning methods that are theoretically grounded by statistical learning theory and have been successfully applied on countless real-world problems. They are composed by a learning algorithm (that is commonly stated as a convex optimization problem) and the kernel function, that is a symmetric positive semidefinite function that defines a similarity measure between examples. A kernel function $k(x_1, x_2)$ implicitly maps two input examples in a feature space (namely, Reproducing Kernel Hilbert Space or RKHS) via a function commonly referred as ϕ, and then computes the dot product $\langle \phi(x_1), \phi(x_2) \rangle$ between the two representations. Both these steps are performed implicitly, and the computational complexity of the kernel evaluation depends only on the input space dimension, and not on the dimension of the feature space which can be even infinite-dimensional.

Kernel methods are widely applied in machine learning for structured data because, unlike the majority of machine learning techniques, their application to any type of data is painless as long as a kernel function for such data is defined. Dealing with data represented as graphs is not simple, since even the basic operations can be computationally expensive (see for example the graph isomorphism problem [21]). The approach that most graph kernels follow is to compare two graphs with respect to the substructures they share. In order to be computationally efficient, the considered substructures are limited to particular types, for which the computation of the isomorphism is efficient. This comes at the cost of a reduced discriminative ability of the kernel. Moreover, in order to control the complexity, the kernels usually depend on a parameter that regulates the size of the considered structures. In this way, they balance the tradeoff between efficiency and expressiveness of the kernel (i.e. how a kernel is able to discriminate between similar examples). An open problem, when we have to deal with kernels for structured data, is the weight given to each feature. The particular weighting policy to apply can strongly influence the predictive performance of the kernel. At the moment, the policy is usually integrated in the kernel definition, possibly including also strong regularization terms, and not much work has been done to evolve this methodology. In addition to that, among the different graph kernels that have been proposed in literature, usually it is not straightforward to decide which one will be the most suited for a specific task.

The Multiple Kernel Learning (MKL) framework of-

978-1-4799-7560-0/15 $31.00 © 2015 IEEE
DOI 10.1109/SSCI.2015.226

fers a technique to learn the kernel directly from data. Specifically, MKL learns a kernel consisting of a weighted combination of different user-specified (or weak) kernels. In this way, the user is no longer required to decide a priori which kernel to use, but the MKL algorithm hopefully finds the right combination of the available kernels, given a task.

Starting from the state-of-the-art ODD$_{ST}$ kernel [8], in our approach, a partition of the original feature space of the kernel is created and a weak kernel is defined on each partition. Features are grouped on the basis of their complexity. Finally, the MKL paradigm is applied on the weak kernels.

The benefits of our approach are twofold. Firstly, with our approach, we do not need a parametric weighting rule for the graph kernel. Instead, the MKL algorithm will automatically tune the weight of different kernels. Secondly, the performance of the resulting kernel empirically improves with respect to the base kernel. Moreover, the computational complexity of the resulting learning procedure is the same as the classical approach (consisting in the kernel computation and the application of an SVM).

The paper is organized as follows. In Section II we introduce the kernels for graphs, with a focus on the one we decided to start from. In Section III we introduce the Multiple Kernel Learning framework, and the recently defined MKL algorithm we adopted. Section IV details how to generate the weak kernels starting from the original graph kernel, and analyze the situation from a statistical learning theory point of view. We show in Section V experimental results on several real world graph datasets. Finally, we draw some conclusions and the basics for future works in Section VI.

A. Notation

In this paper, we consider the classical classification problems and we define the training examples as $\{(x_i, y_i)\}_{i=1}^{l}$, and test examples as $\{(x_i, y_i)\}_{i=l+1}^{L}$, x_i in a generic set $\mathcal{X}$, y_i with values $+1$ or -1. The matrix $\mathbf{K} \in \mathbb{R}^{L\times L}$ is the complete kernel matrix containing the values of the kernel of each (training and test) data pair. Further, we indicate with an hat, like for example $\hat{\mathbf{y}} \in \mathbb{R}^{l}$ or $\hat{\mathbf{K}} \in \mathbb{R}^{l\times l}$, the submatrices (or subvectors) obtained considering training examples only.

Fixed a training set, $\hat{\Gamma}$ will denote the domain of probability distributions $\boldsymbol{\gamma} \in \mathbb{R}^{l}_{+}$ defined over the sets of positive and negative training examples:

$$\hat{\Gamma} = \{\boldsymbol{\gamma} \in \mathbb{R}^{l}_{+} \; | \sum_{i\in\oplus} \gamma_i = 1, \sum_{i\in\ominus} \gamma_i = 1\},$$

where $\oplus$ (resp. $\ominus$) is the set of the indices of the positive examples (resp. negative examples). Note that any element $\boldsymbol{\gamma}$ of the set $\hat{\Gamma}$ corresponds to a pair of points, the first in the convex hull of positive training examples and the second in the convex hull of negative training examples. In fact, $\boldsymbol{\gamma}$ defines the two points as $x^{+} = \sum_{i\in\oplus} \gamma_i x_i$ and $x^{-} = \sum_{i\in\ominus} \gamma_i x_i$, respectively for the positive and negative convex hulls of training examples.

In this paper, we consider the case where training examples are graphs. A graph $G = (V_G, E_G, L_G)$ is a triplet where V_G is the set of vertices, E_G the set of edges and $L_G()$ a function mapping nodes to labels. A graph is undirected if $(v_i, v_j) \in E_G \Leftrightarrow (v_j, v_i) \in E_G$, otherwise it is directed. A path $p(v_i, v_j)$ of length n in a graph G is a sequence of nodes $v_1, \ldots, v_n$, where $v_1 = v_i$, $v_n = v_j$ and $(v_k, v_{k+1}) \in E_G$ for $1 \le k < n$. A cycle is a path for which $v_1 = v_n$. A graph is acyclic if it has no cycles. A DAG is a directed acyclic graph. A tree is a directed acyclic graph where each node has at most one incoming edge. The root of a tree T is represented by $r(T)$. The children of a node $v \in V_T$ are all the nodes v' s.t. $(v, v') \in E_T$. $ch_v[j]$ refers to the j-th child of v. ρ is the maximum number of children in a tree. A proper subtree rooted at node v comprises v and all its descendants.

II. Graph kernels

The vast majority of kernels for structured data are based on the convolution framework. The idea is to decompose a structure into a set of simpler structures (where a *base* kernel between these structures is given) and to define the kernel as a function of the base kernels. More formally, let $\mathcal{X}$ be a space of objects (that are our data points) and let each object x be associated with a finite subset $\mathcal{X}'_x$ of a space $\mathcal{X}'$. Furthermore, assume that a kernel over this domain $k : \mathcal{X}' \times \mathcal{X}' \to \mathbb{R}$ is defined. To define an $R-convolution$ kernel, Haussler [12] assumed a *finite* relation $R \subseteq \mathcal{X}' \times \mathcal{X}$, and let

$$K(x, y) = \sum_{(x',x)\in R} \sum_{(y',y)\in R} k(x', y').$$

Many recently defined graph kernels are members of this general framework.

Many graph kernels depend on a parameter h that defines the maximum depth of the considered substructures. Usually, too small values of h do not allow the kernel to capture enough global information about the structure of the graphs, resulting in poor predictive performances. In the same way, too high values of h tend to obfuscate the contributions of the small substructures. Moreover, the higher the h, the slower the calculation of the kernel becomes.

A. Ordered Decomposition DAG Kernels for Graphs

This section briefly describes the framework of ODD-Kernels for graphs, proposed in [8]. The idea of the ODD kernel framework is to decompose the input graph into a set of substructures for which the isomorphism can be computed efficiently, i.e. subtrees.

In order to map the graphs into trees, two intermediate steps are needed:

1) map the graph G into a multiset of DAGs $\{DD_G^{v_i} | v_i \in V_G\}$, where $DD_G^{v_i}$ is obtained by keeping each edge in the shortest path(s) connecting v_i with any $v_j \in V_G$. The Decomposition

DAGs kernel for graphs can be defined as

$$DDK_{K_{DAG}}(G_1, G_2) = \\ = \sum_{D_1 \in DD_{G_1}} \sum_{D_2 \in DD_{G_2}} K_{DAG}(D_1, D_2).$$

2) Since the vast majority of DAG kernels are extensions of kernels for ordered trees, a strict partial order between DAG nodes in $DD_G^{v_i}$ has been defined yielding Ordered Decomposition DAGs $ODD_G^{v_i}$. The ordering function considers the label of the vertex $L(v)$ and, recursively, the labels of the children of v [8].
3) Finally, any Ordered DAG (ODD) is mapped into a multiset of trees. Let us define $T(v_i)$ as the tree resulting from the visit of $ODD_G^{v_i}$ starting from node v_i: the visit returns the nodes reachable from v_i in $ODD_G^{v_i}$. Note that if a node v_j can be reached more than once, more occurrences of v_j will appear in $T(v_i)$. Also, the tree visit $T(v_i)$ can be stopped when the tree $T(v_i)$ reaches a maximum height r. Such tree is referred to as $T_r(v_i)$.

In [8] the Ordered Decomposition DAGs kernel is defined as:

$$ODDK(G_1, G_2) = \tag{1}$$

$$\sum_{\substack{OD_1 \in ODD_{G_1} \\ OD_2 \in ODD_{G_2}}} \sum_{j=1}^{r} \sum_{\substack{v_1 \in V_{OD_1} \\ v_2 \in V_{OD_2}}} C(r(T_j(v_1)), r(T_j(v_2))) \tag{2}$$

where $C()$ is a function defining a tree kernel. Among the kernels for trees defined in literature, the one employed in the paper is the Subtree Kernel [24], which counts the number of shared proper subtrees between the two input trees. In this case, the C function is defined as follows:

$$C_{ST}(v_1, v_2) = \begin{cases} \lambda & \text{if } L(v_1) = L(v_2) \text{ and} \\ & \rho(v_1) = \rho(v_2) = 0 \\ \lambda \Upsilon(v_1, v_2) & \text{if } L(v_1) = L(v_2) \text{ and} \\ & \rho(v_1) = \rho(v_2) \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

where λ is a weighting parameter, that will be discussed in more detail in the next section, and $\Upsilon(v_1, v_2) = \prod_{j=1}^{\rho(v_1)} C_{ST}(ch_{v_1}[j], ch_{v_2}[j])$. The resulting kernel will be referred to as ODD$_{ST}$.

Summarizing, Ordered Decomposition DAGs kernel consider as features all the proper subtrees of the trees resulting from breadth-first visits starting from each node in the graph. In [7] the authors have shown that it is possible to efficiently compute the explicit feature space representation for such a kernel. Let us assume that a bijective function h mapping labeled trees to integers is given (or equivalently, an enumeration of all possible labeled trees). Then, given a value for r, the ith component of $\phi(G)$ is defined as

$$\phi_i(G) = \\ = |\{h(T_j(v)) = i : j \leq r, v \in D, D \in ODD_G\}| \cdot \lambda^{size(h^{-1}(i))},$$

For ease of notation, $size(G) = |V_G|$. Finally, it is important to note that the property $h(s) = i$ is equivalent to the isomorphism between trees, which can be solved in polynomial time.

B. ODD kernels and feature weighting

The proposed approach exploits two handful considerations:

1) when a tree is present in the feature space representation of a graph, then all its proper subtrees are in the same space as well;
2) the bigger is the tree, the higher is the number of its subtrees that occur.

As a direct consequence, kernels that count common substructures suffer an issue of diagonally dominant gram matrix, i.e. the number of substructures shared by a graph with itself (and thus the corresponding kernel value) is usually much higher than the number of shared substructures between two different instances. This negatively affects the support vector classifier, since the classification of an instance may be dominated by the contribution of the most similar example, leading to a behavior similar to a nearest neighbor classifier. A possible solution for this issue is the downweighting of big structural features. Usually this is accomplished with a function of the size of a feature, i.e. to the number of co-occurring features. The (down)weighting function, firstly proposed for tree kernels [5], for a feature f is defined as:

$$w(f) = \sqrt{\lambda^{size(f)}} \cdot freq(f), \tag{4}$$

where $freq(f)$ is the frequency of a feature in the considered example, and $size(f)$ is the size of the sub-structure associated to the feature f. Here, the weight of a feature grows with its frequency in the example, and exponentially decreases (if $\lambda < 1$) with its size. Note that the selection of an appropriate λ parameter is a key step in the successful application of this weighting scheme. This weighting scheme intuitively makes sense, but there is no reason why other schemes cannot be more suited for a specific task. In a recent work [9], another weighting scheme for graph kernels has been proposed:

$$w(f) = tanh(\sigma \cdot freq(f) \cdot size(f)). \tag{5}$$

However, testing different weighting schemes is unpractical for different reasons. Firstly, when we test different weighting schemes we are adding a new parameter to validate. In addition, it is not clear which weight distributions one should test. Finally, in noisy datasets, where the selection for the optimal parameters is a problem, the gap in predictive performance between the different schemes may be not enough to justify the added complexity. Moreover, the kernel may be more prone to overfitting.

From a theoretical point of view, if enough examples are available, the weighting policy should be irrelevant because the weight of each feature should be adjusted by the learning algorithm. However, especially when dealing with real-world graph datasets, this is not the case, and the dependencies among features makes the learning even more difficult. For this reason, we still need to inject external knowledge into the learning algorithm via strong biases, for example providing a good feature weighting scheme. In Section IV-A, we will present a method to learn the weight associated to each feature directly from the available data.

III. Multiple Kernel Learning (MKL)

MKL [11] is one of the most popular paradigms used to learn kernels in real world applications [4]. The kernels generated by these techniques are combinations of previously defined *weak* kernels $\mathbf{K}_1, ..., \mathbf{K}_R$. The goal of MKL is to alleviate the effort of the user in defining good kernels for a given problem. We focus on MKL method with positive and linear combination parameters, that is, MKL in the form

$$\mathbf{K} = \sum_{j=1}^{R} \boldsymbol{\eta}_j \mathbf{K}_j, \qquad \boldsymbol{\eta}_j \geq 0.$$

MKL algorithms are supported by several theoretical results that bound the difference between the true error and the empirical margin error (i.e. *estimation error*). These bounds exploit the *Rademacher complexity* applied to the combination of kernels [6], [14], [17].

The MKL optimization problem turned out to be a very challenging task as, for example, doing better than the simple average of the *weak* kernels is surprisingly difficult. Moreover, the MKL algorithms have a high computational complexity. More recently, scalable methods have been proposed that can tackle thousands of kernels in a reasonable time and memory space [15], [1]. Having MKL algorithms which are scalable opens a new scenario for MKL. In fact, standard MKL algorithms typically cope with a small number of strong kernels and try to combine them. In this case, these kernels are individually well designed by experts and their optimal combination hardly leads to a significant improvement of the performance with respect to, for example, a simple averaging combination. A second perspective is also possible, where the MKL paradigm can be exploited to combine a large amount of weak kernels. The weak kernels are then exploited boosting their combined accuracy in a way similar to feature weighting. This point of view highlights the existing connection between MKL and feature learning [3]. In fact, we are able to weight the information contained into bunch of features, evaluated in different ways (i.e. using different kernels that can consider different subsets of features) [2], [16].

A. EasyMKL

EasyMKL [1] is a flexible MKL algorithm that combines a set of weak kernels solving a quadratic problem. EasyMKL uses a fixed amount of memory and is linear in the computation complexity with respect to the number of kernels. The EasyMKL optimization problem is defined as:

$$\boldsymbol{\gamma}^* = \arg\min_{\boldsymbol{\gamma}\in\Gamma}(1-\Lambda)\underbrace{\boldsymbol{\gamma}^\top \hat{\mathbf{Y}}(\sum_{j=1}^{R} \hat{\mathbf{K}}_j)\hat{\mathbf{Y}}\boldsymbol{\gamma}}_{\mathcal{Q}(\boldsymbol{\gamma})} + \Lambda \underbrace{||\boldsymbol{\gamma}||_2^2}_{\mathcal{R}(\boldsymbol{\gamma})}. \qquad (6)$$

The minimization problem over the distributions in Γ is composed by two parts. The two parts are balanced using a parameter $\Lambda \in [0, 1]$ that is the only hyper-parameter of this algorithm. Specifically, EasyMKL minimizes a trade-off between the following two quantities:

- $\mathcal{Q}(\boldsymbol{\gamma}) = \boldsymbol{\gamma}^\top \hat{\mathbf{Y}}(\sum_{j=1}^{R} \hat{\mathbf{K}}_j)\hat{\mathbf{Y}}\boldsymbol{\gamma}$ and
- $\mathcal{R}(\boldsymbol{\gamma}) = ||\boldsymbol{\gamma}||_2^2$.

Considering only the function $\mathcal{Q}(\boldsymbol{\gamma})$ (i.e. $\Lambda = 0$), we have to solve the problem defined by

$$\boldsymbol{\gamma}^* = \arg\min_{\boldsymbol{\gamma}\in\Gamma} \boldsymbol{\gamma}^\top \hat{\mathbf{Y}}(\sum_{j=1}^{R} \hat{\mathbf{K}}_j)\hat{\mathbf{Y}}\boldsymbol{\gamma}. \qquad (7)$$

It can be seen that the vector $\boldsymbol{\gamma}^* \in \Gamma$ minimizing $\mathcal{Q}(\boldsymbol{\gamma})$ identifies the two nearest points in the convex hulls of positive and negative examples, respectively, in the feature space of the kernel $\hat{\mathbf{K}}_{sum} = \sum_{j=1}^{R} \hat{\mathbf{K}}_j$.

$\hat{\mathbf{K}}_{sum}$ is a particular kernel and its Reproducing Kernel Hilbert Space can be generated appending all the features contained in the feature space of all the weak kernels $\hat{\mathbf{K}}_1, \ldots, \hat{\mathbf{K}}_R$ [22].

On the other hand, considering only the quadratic regularization part $\mathcal{R}(\boldsymbol{\gamma})$ (i.e. $\Lambda = 1$), the problem becomes:

$$\boldsymbol{\gamma}^* = \arg\min_{\boldsymbol{\gamma}\in\Gamma} ||\boldsymbol{\gamma}||_2^2. \qquad (8)$$

In this case, the optimal solution $\boldsymbol{\gamma}^* \in \Gamma$ has an analytic formula. Let p (resp. n) the number of positive examples (resp. negative examples) in the training set. Then, the optimal solution of the minimization problem $\boldsymbol{\gamma}^*$ collapses in the uniform distributions over positive and negative examples: $\boldsymbol{\gamma}_i^* = 1/p, \ \forall i \in \oplus$ and $\boldsymbol{\gamma}_j^* = 1/n, \forall \ j \in \ominus$.

Fixed the hyper-parameter Λ, the optimal distribution $\boldsymbol{\gamma}^*$ for the kernel $\hat{\mathbf{K}}_{sum}$ can be exploited to define the vector $\mathbf{d}(\boldsymbol{\gamma}^*)$ as

$$\mathbf{d}_j(\boldsymbol{\gamma}^*) = \boldsymbol{\gamma}^{*\top} \hat{\mathbf{Y}}\hat{\mathbf{K}}_j\hat{\mathbf{Y}}\boldsymbol{\gamma}^* \quad \forall j = 1, \ldots, R. \qquad (9)$$

The jth entry of the vector $\mathbf{d}(\boldsymbol{\gamma}^*)$ represents the contribution to the margin between positive and negative examples given by the jth kernel. Then, the vector of the weights $\boldsymbol{\eta}^*$ has a simple analytic solution:

$$\boldsymbol{\eta}^* = \frac{\mathbf{d}(\boldsymbol{\gamma}^*)}{||\mathbf{d}(\boldsymbol{\gamma}^*)||_2}.$$

Finally, the new kernel is defined as $\mathbf{K}_{MKL} = \sum_{j=1}^{R} \boldsymbol{\eta}_j^* \mathbf{K}_j$.

IV. Hierarchical structure of the weak information

In this section, we introduce the general ideas of our proposed method to generate a hierarchical set of weak kernels from a single kernel. This is possible because of the specific definition of some kernels for structured data. In fact, we can apply these techniques directly in the Reproducing Kernel Hilbert Space, defining simple rules to group the generated features in a principled way. Several different schemes for grouping features can be defined. In the following section, we will present in more detail the approach proposed in this work. However, there is no theoretical motivation for hindering the application of the proposed technique to any arbitrary grouping of the features generated by any kernel. Our method does not fix a specific weighting scheme. However, the way in which the features are grouped defines a strong bias in the feature weighting. In fact, we are forcing features belonging to the same weak kernel to contribute with the same amount to generate the final kernel.

A. Generating the weak kernels

The general idea underpinning the generation of the weak kernels starting from an explicit RKHS representation of a kernel is quite simple: it suffices to define a function mapping features to a limited set of natural numbers, corresponding to buckets. Then, each weak kernel is defined as the dot product that considers just the features belonging to a particular bucket.

More formally, we recall that h has been defined as a bijective function mapping features (trees in our scenario) to natural numbers. Let us define a (possibly non injective) function $d : \mathcal{H} \rightarrow \Sigma$, where $\mathcal{H}$ is the RKHS and $\Sigma = \{0, 1, 2, \dots, n\}$, mapping trees to a finite set of buckets. This function assigns each feature to a specific bucket. Also, d will be used to define the weak kernels, i.e. features that are mapped to the same value will be part of the same kernel. Let us define the jth component of the ith weak kernel in the sequence as: $\phi^i_j(G) = \phi_j(G)\delta(d(h^{-1}(j)), i)$ where δ is the Kronecker's delta function.
Then we define the i-th weak kernel as $\mathbf{K}_i(G_1, G_2) = \langle \phi^i(G_1), \phi^i(G_2) \rangle$.

Let us consider the ODD$_{ST}$ kernel defined in Section II-A and its corresponding explicit feature space representation. Every feature in this space encodes a specific labeled tree. Intuitively, the strategy we decided to employ is to generate a weak kernel corresponding to each height of the tree features, i.e. we group in a particular weak kernel all the trees in the RKHS having the same height.

The motivation of such a choice is the following. The features of ODD$_{ST}$ are not independent because, as detailed in Section II-A, if a tree is present in the explicit feature space representation of a graph, also all its proper subtrees are. This relation describes a partial order relation between features, where $a \dot{<} b$ if and only if a is isomorphic to a sub-tree of b. Figure 1 shows an example of some features and draws the partial order relationship induced by the "has as proper subtree" relationship. A necessary condition for two features to be in the relationship is that the height of the first feature must be greater than the height of the second feature. Thus, the simplest way to define a $d()$ function in such a way that two dependent features do not end up in the same bucket is to define $d(t) = height(t)$ where $height(t)$ is a function returning the height of a tree (that is the maximum distance between the root and a leaf node).

Then, the j-th weak kernel $\mathbf{K}_j$ comprehends all the tree-features with a fixed height j. Let us recall that the original ODD$_{ST}$ kernel depends on the r parameter, determining the maximum height of the performed DAG visits, and on a λ parameter, determining the weight of the features as a function of their size. With our proposed approach, for a fixed kernel, r weak kernels will be generated, indexed with $h \in \{0, 1, \dots, r\}$. The λ parameter is no longer required, since the learning algorithm will weight the different buckets of features. This formulation is particularly convenient from a computational point of view and the weak kernels can be generated from a slight modification of the original kernel algorithm. The resulting explicit feature space representation of the i-th kernel is: $\phi^i_j(G) = \phi_j(G)\delta(height(h^{-1}(j)), i)$.

This feature grouping scheme is simple, but it is designed in order to behave in a very specific way. In fact, each sub-kernel will have a weight assigned by exploiting EasyMKL. This means that the features in the same bucket will share the same weight. Since all the proposed feature weighting schemes depend in some sense from the size of the features, it is reasonable to maintain this intuition in our bucketing function.

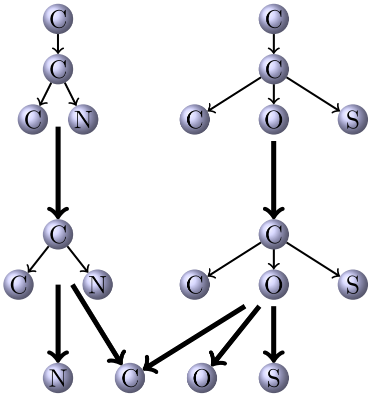

Fig. 1. Example of hierarchy among tree features. Thick arrows encodes the "has as proper subtree" relationship.

B. Complexity of the weak kernels

The hierarchical structure of the weak kernels is highlighted studying the complexity of the single kernel. Firstly, we define the set of the vectors with fixed 1-norm

$$\Psi = \{\boldsymbol{\alpha} \in \mathbb{R}^L \text{ s.t. } \|\boldsymbol{\alpha}\|_1 = 1\}. \tag{10}$$

Then, the class of functions that we consider are linear functions with bounded norm

$$\{\mathbf{x}_j \rightarrow \sum_{i=1}^{l} \alpha_i \mathbf{K}_{i,j} : \boldsymbol{\alpha}^T \mathbf{K} \boldsymbol{\alpha} \leq B^2, \ \forall \boldsymbol{\alpha} \in \Psi\} \subseteq \tag{11}$$

$$\subseteq \{\mathbf{x}_j \rightarrow \mathbf{w} \cdot \boldsymbol{\phi}(\mathbf{x}_j) : \|\mathbf{w}\|_2 \leq B\} = \mathcal{F}_B, \tag{12}$$

where $\boldsymbol{\phi}$ is the feature mapping for the kernel $\mathbf{K}$. The following theorem bounding the complexity of $\mathcal{F}_B$ holds:

Theorem 1. *([22], Theorem 4.12). Given a kernel* $\mathbf{K}$*, evaluated over a set of points* $\mathcal{X}$*, the Empirical Rademacher Complexity of the class* $\mathcal{F}_B$ *satisfies*

$$\hat{\mathcal{R}}(\mathcal{F}_B) \leq \mathcal{O}(B\sqrt{tr(\mathbf{K})}). \tag{13}$$

Equation 13 shows that the *empirical Rademacher complexity* is proportional to the trace of the kernel. Now, we introduce important observations about the value of

$\boldsymbol{\alpha}^T\mathbf{K}\boldsymbol{\alpha}$, for a general kernel $\mathbf{K}$. Specifically, the following proposition holds:

Proposition 1. *Let $\mathbf{K}$ be a kernel matrix in $\mathbb{R}^{L\times L}$ with eigenvalues $\lambda_1 \geq \cdots \geq \lambda_L \geq 0$, then*

$$\forall \boldsymbol{\alpha} \in \Psi, \quad \boldsymbol{\alpha}^T\mathbf{K}\boldsymbol{\alpha} \leq \lambda_1 \leq \|\mathbf{K}\|_F. \tag{14}$$

and λ_1 is the optimal bound.

Proof: We can exploit the spectral decomposition of a kernel matrix and rewrite

$$\boldsymbol{\alpha}^T\mathbf{K}\boldsymbol{\alpha} = \sum_{i=1}^{L} \lambda_i(\boldsymbol{\alpha}^T\boldsymbol{u}_i)^2, \tag{15}$$

where $\boldsymbol{u}_i$ is the eigenvector with eigenvalue λ_i. Then, it is easy to see that:

$$(\boldsymbol{\alpha}^T\boldsymbol{u}_i)^2 = \cos(\theta_{\boldsymbol{\alpha},\boldsymbol{u}_i})^2\|\boldsymbol{\alpha}\|_2^2, \tag{16}$$

where $\theta_{\boldsymbol{\alpha},\boldsymbol{u}_i}$ is the angle between the vector $\boldsymbol{\alpha}$ and the eigenvector $\boldsymbol{u}_i$. Using the properties of the norms ($\|\boldsymbol{\alpha}\|_2^2 \leq \|\boldsymbol{\alpha}\|_1^2 = 1$) and the fact that the eigenvectors $\boldsymbol{u}_i$ are an orthonormal base, we can obtain the final result:

$$\boldsymbol{\alpha}^T\mathbf{K}\boldsymbol{\alpha} \leq \sum_{i=i}^{L} \lambda_i \cos(\theta_{\boldsymbol{\alpha},\boldsymbol{u}_i})^2\|\boldsymbol{\alpha}\|_1^2 \leq \lambda_1 \leq \sqrt{\sum_{i=1}^{L}\lambda_i^2} = \|\mathbf{K}\|_F. \tag{17}$$

The bound is reached when the kernel matrix $\mathbf{K}$ has only one eigenvalue different from zero, its eigenvector is parallel to the vector $\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}$ is such that $\|\boldsymbol{\alpha}\|_2 = \|\boldsymbol{\alpha}\|_1$. ■

Finally, from the previous results the following corollary holds:

Corollary 1. *Let $\mathbf{K}$ be a kernel matrix in $\mathbb{R}^{L\times L}$ and Ψ the set defined in Eq. 10, then*

$$\forall \boldsymbol{\alpha} \in \Psi, \quad \frac{\boldsymbol{\alpha}^T\mathbf{K}\boldsymbol{\alpha}}{\|\mathbf{K}\|_F} \leq 1. \tag{18}$$

From Proposition 1 and using Theorem 1, given a kernel with Frobenius norm equal to 1, we have that the upper bound of the *empirical Rademacher complexity* becomes $\hat{\mathcal{R}}(\mathcal{F}_B) \leq \mathcal{O}(\sqrt{tr(\mathbf{K})})$ when setting $B^2 = B = 1$. Then, the trace of unitary Frobenius Norm kernels is a good estimator for their *empirical Rademacher complexity.*

Now, we exploit the results above to give a principled definition for a complexity of a kernel. We call this measure *spectral complexity.*

The *Spectral complexity* is defined by the ratio between the trace norm and the Frobenius norm of a matrix. Note that, it can be also computed by resorting to the eigenvalue decomposition, that is:

$$R(\mathbf{K}) = \frac{\sum_{i=1}^{L}\lambda_i}{\sqrt{\sum_{i=1}^{L}\lambda_i^2}} = \frac{tr(\mathbf{K})}{||\mathbf{K}||_F} = \frac{tr(\mathbf{K})}{\sqrt{\sum_{ij}\mathbf{K}_{ij}^2}}, \tag{19}$$

where we used the fact that, for positive definite matrices, the Frobenius norm is equal to the 2-norm of the eigenvalues.

$h/data$	CAS $r^*=3$	CPDB $r^*=3$	AIDS $r^*=8$	NCI1 $r^*=4$	GDD $r^*=2$
0	0.002	0.008	0.004	0.001	0.005
1	0.006	0.025	0.009	0.004	0.183
2	0.021	0.080	0.023	0.012	0.999
3	0.151	0.480	0.110	0.089	
4			0.229	0.368	
5			0.306		
6			0.375		
7			0.468		
8			0.683		
$\mathbf{K}_{sum}$	0.015	0.051	0.074	0.017	0.061
$\mathbf{K}_{MKL}$	0.095	0.215	0.275	0.165	0.281

TABLE I. SPECTRAL COMPLEXITY OF THE WEAK KERNELS, FOR EACH DATASET AND FOR EACH VALUE OF $h \leq r^*$ COMPARED TO THE SPECTRAL COMPLEXITY OF $\mathbf{K}_{sum}$ AND $\mathbf{K}_{MKL}$.

The Spectral Complexity is not the first complexity measure for kernels that exploits the trace [18]. Tipically, the trace of a kernel is used as (part of the) bound for the generalization error. On the other hand, the simple trace is not a good measure for the estimation of the kernel expressivity. In particular, if two kernels differ only for a multiplicative constant, they have different traces, but equal expressivity. Our Spectral complexity solves this issue applying the normalization of the Frobenious norm.

In fact, *Spectral complexity* has the following interesting properties:

- the identity matrix $\mathbf{I}_L$ has maximum complexity with $R(\mathbf{I}) = \sqrt{L}$,
- the kernel $\mathbf{K} = \mathbf{1}\mathbf{1}^\top$ has the least complexity with $R(\mathbf{K}) = R(\mathbf{1}\mathbf{1}^\top) = 1$ and
- $R(\mathbf{K}) = R(c \cdot \mathbf{K}), \quad \forall c > 0.$

An equivalent standardized version can also be used with values in the interval $[0, 1]$, that is

$$R_n(\mathbf{K}) = \frac{R(\mathbf{K}) - 1}{\sqrt{L} - 1} \in [0, 1]. \tag{20}$$

For all the weak kernels combined using EasyMKL we have evaluated the normalized version of the *spectral complexity.* For each dataset, the weak kernels evaluated are $\mathbf{K}_1, \ldots, \mathbf{K}_{r^*}$, where r^* is the best hyper-parameter r obtained in validation. The results are shown in Table I and Figure 2. The results highlight how our segmentation of the features induces a hierarchical list of kernels with respect to the *spectral complexity.* In fact, the h parameter of the single weak kernel is monotonically connected to the spectral complexity. This result is correlated to the popularity[1] of the features among the graphs of the dataset. In general, sub-structures that are simpler (i.e with low r) exist among a larger number of graphs with respect to features that are more complex (i.e. with high r).

The kernel $\mathbf{K}_{sum}$ uses all the features contained in the weak kernels with the same weight and it obtains a very low spectral complexity. Table I shows that the difference between $\mathbf{K}_{sum}$ and $\mathbf{K}_{MKL}$ is large in terms of *spectral complexity.* Specifically, for all the datasets the *spectral complexity* of $\mathbf{K}_{MKL}$ is bigger with respect to the one

[1] The popularity is an estimation of the probability of the existence of a specific feature in a randomly picked example from the dataset.

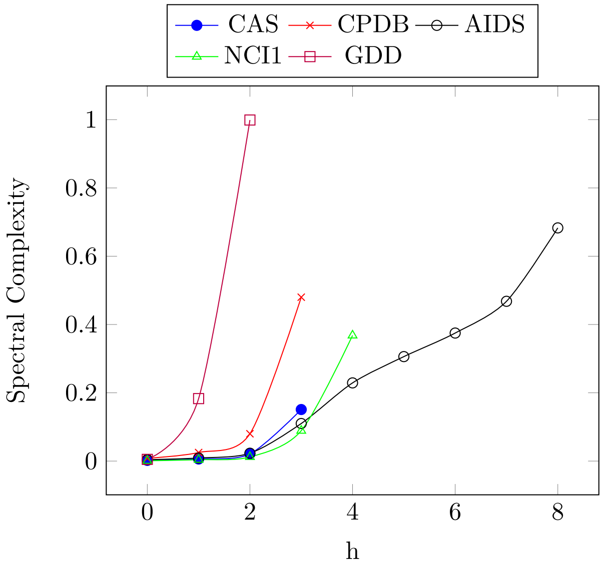

Fig. 2. Spectral complexity of the weak kernels, for each dataset and for each value of $h \leq r^*$.

of $\mathbf{K}_{sum}$ and is lower than the one of the weak kernel $\mathbf{K}_{r^*}$ (i.e. $R(\mathbf{K}_{r^*}) > R(\mathbf{K}_{MKL}) > R(\mathbf{K}_{sum})$). We have empirically proved that neither using all the weak kernels summed with the same weight nor using only the more complex features (contained in $\mathbf{K}_{r^*}$) is the optimal solution given a specific task. EasyMKL allows us to select a trade-off between the complexity, optimizing the weights with respect to the specific task.

V. Experiments

In this section, we compare our proposed ODD-MKL approach to the ODD_{ST} kernel that we considered as the base kernel. This kernel has shown state-of-the-art predictive performances [8]. Moreover, as baseline of the grouping method of the features, we created random splits of features of ODD_{ST} generating weak kernels without any hierarchical structure. We compared the MKL results obtained using this family of weak kernels with our proposed method that, on the other hand, adopts a principled way to split the features among the different weak kernels.

For the original ODD_{ST} kernel, we adopted an SVM classifier [20] given its state-of-the-art predictive performances. The C parameter has been selected in the range $\{0.001, 0.01, \ldots, 10000\}$, and the parameters of the kernel $h \in \{1, 2, \ldots, 8\}$, $\lambda \in \{0.5, 0.6, \ldots, 1.6\}$. As for the methods adopting the MKL approach, the Λ parameter of EasyMKL has been validated in $\{0, 0.1, 0.2, \ldots, 1.0\}$.

A. Datasets description

We compared the different methods on five real-world graph datasets from bioinformatics. CAS[2], CPDB [13], AIDS [26], NCI1 [25] and GDD [10]. The first four datasets represent chemical compounds: nodes represent atoms and are labeled according to the atom type, and edges represent bonds between atoms. GDD is a dataset of proteins. In this case, each node in a graph represents an amino acid and is labeled according to the amino acid type. In GDD, there is an edge connecting two nodes if the corresponding amino acids are less than 6Å apart in the 3-dimensional folding of the protein. All the datasets encode binary classification problems.

Data/Kernel	ODD_{ST}	ODD-MKL
CAS	$0.8982_{\pm 0.0017}$	$0.9049_{\pm 0.0008}$
CPDB	$0.8442_{\pm 0.0067}$	$0.8564_{\pm 0.0056}$
AIDS	$0.8262_{\pm 0.0052}$	$0.8515_{\pm 0.0031}$
NCI1	$0.9069_{\pm 0.0010}$	$0.9144_{\pm 0.0008}$
GDD	$0.8473_{\pm 0.0038}$	$0.8498_{\pm 0.0026}$

TABLE II. Average AUC results (± standard deviation) of the original ODD_{ST} kernel and the proposed ODD-MKL kernel, in nested 10-fold cross validation.

Data/Kernel	ODD_{ST}	ODD-MKL	ODD-MKL-random
CAS	0.8983	0.9049	0.8762
CPDB	0.8576	0.8676	0.8386
AIDS	0.8401	0.8706	0.8480
NCI1	0.9085	0.9159	0.8782
GDD	0.8426	0.8507	0.8299

TABLE III. Average AUC results of the original ODD_{ST} kernel, the proposed ODD-MKL kernel, and the random split MKL kernel in 10-fold cross validation.

B. Results

Table II reports the AUC results of the base kernel (ODD_{ST}) and our proposed MKL approach in *nested* 10-fold cross validation, where the hyper-parameters of the different methods are selected using the training dataset only, using an inner 10-fold cross validation for each split of the 10-fold. The whole procedure has been repeated 10 times, and we reported the average and standard deviation of the 10 runs. From the table emerges that the proposed approach is able to improve the predictive performance with respect to the baseline in all the considered datasets.

In order to investigate if the proposed bucketing function makes sense, or if the performance improvements are due just to the fact that the algorithm has more degrees of freedom, we compared the proposed feature grouping method with a random method. We set, for each dataset, the best performing kernel parameter r^* (i.e. the parameter that defines the maximum height of the extracted tree). We computed a new feature bucketing function that randomly assigns each feature to one of the buckets. We repeated the experiment with different random seeds in order to mitigate the fluctuations in performance due to the random assignments of the features. Table III reports the results of such an experiment. In all the datasets the random approach performs worse than our proposed method, and in 4 out of 5 datasets it performs worse than the base kernel. These results confirm that the hierarchical relationship in the feature grouping criterion is a key factor in the definition of the weak kernels to successfully apply MKL.

VI. Conclusions and future work

In this paper, we proposed a way to improve the predictive performance of graph kernels. We have generated a set

[2] http://www.cheminformatics.org/datasets/bursi

of weak kernels from a graph kernel (ODD_{ST}) grouping the feature space in a principled way, and then applied MKL to the weak kernels. We have proved that our weak kernels are generated in a hierarchical manner with respect to a new measure of complexity (called *spectral complexity*) strictly connected to the *empirical Rademacher complexity*.

From the experimental results, we can claim that the new kernels outperform the base kernels with respect to the AUC measure. It is important to highlight that this result is reached using our principled grouping of the features injected to the weak kernels. In fact, using random bunches of features the new kernels obtained from the MKL algorithm have unsatisfying performances (that are even worse than the base kernel).

As a side effect, our proposed method has been able to learn the weight of each bunch of features without any new hyper-parameter. Then, the proposed method has one parameter less than the original methodology.

Future research lines include, for example, the application of the proposed method to other graph kernels, a more complete analysis of the complexity of the generated kernel matrices and the generation of deeper kernels combining more shattered sets of features in order to exploit the full potentiality of the MKL.

Acknowledgments

This work was supported by the University of Padova under the strategic project *BIOINFOGEN*.

References

[1] F. Aiolli and M. Donini. Easymkl: a scalable multiple kernel learning algorithm. *Neurocomputing*, 2015.

[2] F. R. Bach. Exploring large feature spaces with hierarchical multiple kernel learning. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems 21*, pages 105–112. Curran Associates, Inc., 2009.

[3] V. Bolon-Canedo, F. Aiolli, and M. Donini. Feature and kernel learning. In *23th European Symposium on Artificial Neural Networks, ESANN 2015, Bruges, Belgium, April 22-24, 2015*, 2015.

[4] S. S. Bucak, R. Jin, and A. K. Jain. Multiple kernel learning for visual object recognition: A review. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 36(7):1354–1369, 2014.

[5] M. Collins and N. Duffy. Convolution Kernels for Natural Language. *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS*, 14:625–632, 2001.

[6] C. Cortes, M. Mohri, and A. Rostamizadeh. Generalization bounds for learning kernels. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, pages 247–254, 2010.

[7] G. Da San Martino, N. Navarin, and A. Sperduti. A memory efficient graph kernel. In *The 2012 International Joint Conference on Neural Networks (IJCNN)*. Ieee, June 2012.

[8] G. Da San Martino, N. Navarin, and A. Sperduti. A Tree-Based Kernel for Graphs. In *Proceedings of the Twelfth SIAM International Conference on Data Mining*, pages 975–986, 2012.

[9] G. Da San Martino, N. Navarin, and A. Sperduti. Exploiting the ODD framework to define a novel effective graph kernel. In *23th European Symposium on Artificial Neural Networks, ESANN 2015, Bruges, Belgium, April 22-24, 2015*, 2015.

[10] P. D. Dobson and A. J. Doig. Distinguishing Enzyme Structures from Non-enzymes Without Alignments. *Journal of Molecular Biology*, 330(4):771–783, 2003.

[11] M. Gönen and E. Alpaydin. Multiple kernel learning algorithms. *Journal of Machine Learning Research*, 12:2211–2268, 2011.

[12] D. Haussler. Convolution Kernels on Discrete Structures. Technical report, Department of Computer Science, University of California at Santa Cruz, 1999.

[13] C. Helma, T. Cramer, S. Kramer, and L. De Raedt. Data mining and machine learning techniques for the identification of mutagenicity inducing substructures and structure activity relationships of noncongeneric compounds. *Journal of Chemical Information and Computer Sciences*, 44(4):1402–1411, 2004.

[14] Z. Hussain and J. Shawe-Taylor. Improved loss bounds for multiple kernel learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011*, pages 370–377, 2011.

[15] A. Jain, S. V. N. Vishwanathan, and M. Varma. Spg-gmkl: Generalized multiple kernel learning with a million kernels. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, August 2012.

[16] P. Jawanpuria, M. Varma, and S. Nath. On p-norm path following in multiple kernel learning for non-linear feature selection. In T. Jebara and E. P. Xing, editors, *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 118–126. JMLR Workshop and Conference Proceedings, 2014.

[17] M. Kloft and G. Blanchard. The local rademacher complexity of lp-norm multiple kernel learning. In *Advances in Neural Information Processing Systems*, pages 2438–2446, 2011.

[18] G. R. G. Lanckriet, N. Cristianini, P. Bartlett, L. E. Ghaoui, and M. I. Jordan. Learning the Kernel Matrix with Semidefinite Programming. *J. Mach. Learn. Res.*, 5:27–72, 2004.

[19] T. Mitchell, W. Cohen, E. Hruschka, P. Talukdar, J. Betteridge, A. Carlson, B. D. Mishra, M. Gardner, B. Kisiel, J. Krishnamurthy, N. Lao, K. Mazaitis, T. Mohamed, N. Nakashole, E. Platanios, A. Ritter, M. Samadi, B. Settles, R. Wang, D. Wijaya, A. Gupta, X. Chen, A. Saparov, M. Greaves, and J. Welling. Never-Ending Learning, 2015.

[20] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[21] R. C. Read and D. G. Corneil. The graph isomorphism disease. *Journal of Graph Theory*, 1(4):339–363, 1977.

[22] J. Shawe-Taylor and N. Cristianini. *Kernel methods for pattern analysis.* Cambridge university press, 2004.

[23] A. Singhal. Introducing the Knowledge Graph: things, not strings, 2012.

[24] S. V. N. Vishwanathan and A. J. Smola. Fast Kernels for String and Tree Matching. In *NIPS*, pages 569–576, 2002.

[25] N. Wale, I. Watson, and G. Karypis. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and Information Systems*, 14(3):347–375, 2008.

[26] O. S. Weislow, R. Kiser, D. L. Fine, J. Bader, R. H. Shoemaker, and M. R. Boyd. New soluble-formazan assay for HIV-1 cytopathic effects: application to high-flux screening of synthetic and natural products for AIDS-antiviral activity. *Journal of the National Cancer Institute*, 81(8):577–586, 1989.