

# Fast Hyperparameter Selection for Graph Kernels via Subsampling and Multiple Kernel Learning

Michele Donini<sup>1</sup>, Nicolò Navarin<sup>2</sup>, Ivano Lauriola<sup>2</sup>, Fabio Aioli<sup>2</sup> and Fabrizio Costa<sup>3</sup>

1- Computational Statistics and Machine Learning (CSML)  
Istituto Italiano di Tecnologia, Via Morego 30, Genova, Italy

2- Department of Mathematics, University of Padova  
via Trieste 63, Padova, Italy

3- Department of Computer Science, University of Exeter  
Exeter EX4 4QF, UK

## Abstract.

Model selection is one of the most computationally expensive tasks in a machine learning application. When dealing with kernel methods for structures, the choice with the largest impact on the overall performance is the selection of the feature bias, i.e. the choice of the concrete kernel for structures. Each kernel in turn exposes several hyper-parameters which also need to be fine tuned. Multiple Kernel Learning offers a way to approach this computational bottleneck by generating a combination of different kernels under different parametric settings. However, this solution still requires the computation of many large kernel matrices. In this paper we propose a method to efficiently select a small number of kernels on a subset of the original data, gaining a dramatic reduction in the runtime without a significant loss of predictive performance.

## 1 Introduction

Many application domains can be naturally encoded in a structured form. In chemo- and bioinformatics for example, molecules and polymers can be encoded as graphs with atoms as nodes and chemical bonds as edges. After a segmentation pre-processing step, images can also be represented as networks of adjacent objects or regions. Working on these domains employing traditional feature engineering is possible, but it is generally time-consuming (every different application requires a novel feature design by some expert) and information loss is often unavoidable. Instead, a direct processing of structures can be obtained using kernelized approaches, where a kernel function can be devised for each generic data type (for example sequences, trees or graphs). This in turn allows to use standard learning algorithms (e.g. SVM) to solve the desired predictive tasks.

Model selection is one of the most computationally expensive tasks in machine learning applications. When dealing with kernel methods for structures, the choice with the largest impact on the overall performance is the selection of the feature bias, i.e. the choice of the concrete kernel for structures. Each kernel in turn exposes several hyper-parameters which also need to be fine tuned. Finally, the downstream learning algorithm has its own hyper-parameters. A

common approach to hyper-parameters optimization is to fix a-priori a *parameters grid*, i.e. a set of possible values for each parameter. Then, the predictive performance of the learning algorithm is evaluated for each possible parameters combination (*grid-search* strategy), which is possibly a very time consuming step. A more efficient approach has been proposed in [1, 2], where instead of selecting the best parameters configuration, Multiple Kernel Learning (MKL) was used for combining different kernel matrices in a single learning procedure. While more efficient, this approach does not scale to very large dataset since a kernel matrix has to be computed for each kernel and for each of its hyper-parameters. In this paper we propose a new approach that, based on sub-sampling and MKL, is able to speedup the parameter selection phase by orders of magnitude on real-world datasets without loss in predictive performance.

## 2 Background

A kernel method is composed by the following two modules. A learning algorithm, that expresses the solution only via dot products between training examples. A kernel function  $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$  that is a symmetric positive semi-definite function that corresponds to a dot product in a Reproducing Kernel Hilbert Space (RKHS), i.e. there exists a  $\phi : \mathcal{X} \rightarrow \mathcal{K} \subseteq \mathbb{R}^D$ , where  $\mathcal{X}$  is the input space and  $\mathcal{K}$  is an Hilbert space (commonly referred to as *feature space*), such that  $k(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle$  with  $x_i, x_j \in \mathcal{X}$ . A kernel (or *Gram*) matrix for a set of examples  $Tr = \{x_1, \dots, x_l\}$  is an  $l \times l$  matrix  $G$  with entries  $\{G\}_{ij} = k(x_i, x_j)$ .

When dealing with structured data, the choice of the kernel to adopt is a key step. In this paper we focus on graphs, i.e. we consider each example to be a different graph. In this setting, there are several graph kernels defined in literature based on random walks [3, 4], shortest paths [5], or subgraphs up to a fixed size  $h$  [6]. The problem with these kernels is the high computational complexity, that makes them inapplicable to several real-world datasets. Recently, different almost linear time graph kernels have been defined in literature [7, 8, 9, 10], and these are the ones we focus on in this paper. Each one of these kernels considers only specific kinds of subgraphs, up to a certain dimension (that, in general, is a kernel parameter). Note that it is out of the scope of this paper to give an extensive comparison among the various graph kernels.

The Weisfeiler-Lehman Fast Subtree kernel (WL) [9] counts the number of identical subtree patterns obtained by subtree-walks up to height  $h$  (an hyperparameter) in the two input graphs. Note that the subtree-walks extracted by the kernel differ from subtree structures because a node can appear multiple times in the same subtree-walk.

In [8] the Neighborhood Subgraph Pairwise Distance Kernel (NSPDK) is presented. This kernel computes the exact matches between pairs of subgraphs with controlled size and distance (hyper-parameters  $r$  and  $d$ , respectively).

The Ordered Decomposition DAGs Subtree kernel ( $ODD_{ST}$ ) [10] considers as non-zero features in the RKHS the trees that appear as subtrees of the input graphs. It exploits the shortest-path (up to length  $h$ , a hyperparameter) DAG decompositions starting from each node in the graph to generate DAG structures, and then extracts tree features from them. Each tree-feature is weighted as

$f \cdot \lambda^{dim}$ , where  $f$  is the frequency of the feature,  $dim$  its dimension (the number of vertices in the tree) and  $\lambda > 0$  a weighting hyperparameter.

## 2.1 Multiple Kernel Learning

MKL [11] is one of the most popular paradigms in real world applications [12] used to learn kernels as a combination of different base kernels  $\mathbf{K}_1, \dots, \mathbf{K}_R$ . In literature exists several kernels combination mechanisms, we focused on the form:  $\mathbf{K} = \sum_{r=1}^R \eta_r \mathbf{K}_r$  with  $\boldsymbol{\eta} \in \{\boldsymbol{\eta} : \boldsymbol{\eta} \succeq 0, \|\boldsymbol{\eta}\|_q = 1\}$ . The value  $q$  being the kind of mean used, is typically fixed to 1 or 2.

Using this formulation, we are studying the family of sums of kernels in the feature space and it is well know that the sum of two kernels can be seen as the concatenation of the features contained in both the RKHS [13]. Extending the same idea, the weighted sum of a list of basic kernels can be seen as a weighted concatenation of all the features contained in all the RKHS (where the weights are the square roots of the learned weights in the vector  $\boldsymbol{\eta}$ ).

These algorithms are supported by several theoretical results that bound the *estimation error* (i.e. the difference between the true error and the empirical margin error). These bounds exploit the *Rademacher complexity* applied to the combination of kernels [14, 15].

### 2.1.1 EasyMKL

EasyMKL [16] is a recent MKL algorithm able to combine sets of basic kernels by solving a simple quadratic optimization problem. Besides its proved empirical effectiveness, a clear advantage of EasyMKL compared to other MKL methods is its high scalability with respect to the number of kernels to be combined. Specifically, its computational complexity is constant in memory and linear in time. EasyMKL finds the coefficients  $\boldsymbol{\eta}$  that maximize the margin on the training set, where the margin is computed as the distance between the convex hulls of positive and negative examples. In particular, the general problem EasyMKL tries to optimize is the following:

$$\max_{\boldsymbol{\eta} : \|\boldsymbol{\eta}\|_2 = 1} \min_{\boldsymbol{\gamma} \in \Gamma} \boldsymbol{\gamma}^\top \mathbf{Y} \left( \sum_{r=0}^R \eta_r \mathbf{K}_r \right) \mathbf{Y} \boldsymbol{\gamma} + \Lambda \|\boldsymbol{\gamma}\|_2^2, \quad (1)$$

where  $\mathbf{Y}$  is a diagonal matrix with training labels on the diagonal, and  $\Lambda$  is a regularization hyperparameter. The domain  $\Gamma$  represents two probability distributions over the set of positive and negative examples of the training set, that is  $\Gamma = \{\boldsymbol{\gamma} \in \mathbb{R}_+^\ell \mid \sum_{y_i=+1} \gamma_i = 1, \sum_{y_i=-1} \gamma_i = 1\}$ . Note that any element  $\boldsymbol{\gamma} \in \Gamma$  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. At the solution, the first term of the objective function represents the obtained margin, that is the (squared) distance between a point in the convex hull of positive examples and a point in the convex hull of negative examples, in the compounded feature space. Eq. 1 is a minimax problem that can be reduced to a simple quadratic problem with a technical derivation described in [16]. In the following sections, we will refer to this method as EasyMKL<sup>1</sup>.

<sup>1</sup>EasyMKL implementation: [github.com/jmikko/EasyMKL](https://github.com/jmikko/EasyMKL)

### 3 Proposed framework

We propose a method to reduce the number of kernels to compute on the complete dataset. We argue that the combination vector learned by EasyMKL on just a subset of the available data ( $\hat{\eta}$ ) is similar to the one obtained by the same algorithm on the whole dataset ( $\eta$ ). See Fig. 1 for an experimental assessment of this assumption. We use the values in  $\hat{\eta}$  to perform a ranking over kernels, removing those below a (user-specified) threshold. Then we compute the kernel matrix, on all the available data, for just the top kernels, and fit a model combining them. A sketch of the proposed algorithm is given in the following.

```

1 Input: examples  $Tr$ , list of kernel functions  $L$ ,
2 kernel percentage  $p_K$ , example percentage  $p_S$ 
3 Output: a list of Gram matrices on  $Tr$ 
4  $S \leftarrow \text{random\_subset\_of}(Tr, p_S)$ 
5  $K_S \leftarrow [\text{Gram}(S, k) \text{ for } k \text{ in } L]$  #Gram matrices on  $Tr_s$ 
6  $\hat{\eta} \leftarrow \text{EasyMKL}(K_S)$  #weights of the functions in  $L$ 
7  $L_{TOP} \leftarrow \text{top\_of}(L, \hat{\eta}, p_K)$  #get the top functions
8 Return  $[\text{Gram}(Tr, k) \text{ for } k \text{ in } L_{TOP}]$  #top Gram matrices on  $Tr$ 

```

In the algorithm, we adopted Python’s list comprehension notation. We used the following functions: *random\_subset\_of*( $Tr, p_S$ ) samples a percentage  $p_S$  of the examples in the set  $Tr$ ; *Gram*( $Tr, k$ ) computes the *Gram* matrix of the kernel  $k$  on the set of examples  $Tr$ ; *EasyMKL*( $K_S$ ) computes a list of weights (one for each kernel in  $K_S$ ) according to the EasyMKL algorithm; *top\_of*( $L, \hat{\eta}, p_K$ ) extracts the top  $p_K$  percentage of the elements in  $L$  according to the weights  $\hat{\eta}$ .

### 4 Experimental Assessment

We conducted our experiments on several real-world binary datasets, that are MUTAG, CPDB, GDD, AIDS, NCI1, NCI109, CAS (see [10, 9] for a more detailed description). The kernels used in our experiments are: NSPDK with  $h \in \{0, 1 \dots 9\}$  and  $d \in \{0, 1 \dots 9\}$ , ODD<sub>ST</sub> with  $h \in \{0, 1 \dots 9\}$  and  $\lambda \in \{0.5, 0.7, 0.8 \dots 1.2, 1.4, 1.6, 1.8\}$ , WL with  $h \in \{0, 1 \dots 9\}$ . In all models, we considered kernels calculated in less than 1 hour, rejecting others. The baselines used to compare our method consist on a SVM with kernels (a) $K_{VAL}$ : the kernel is the one which has the best validation score, (b) $K_{SUM}$ : the kernel is the average of all base kernels, (c) $K_{MKL}$ : the kernel is a combination of all base kernels using EasyMKL. For each baseline, the computation of all kernels is required. The hyper-parameters are chosen using a nested 10-fold CV procedure, and they are  $C \in \{2^i, i : -3 \dots 6\}$ , and  $\Lambda \in \{0, 0.1 \dots 1\}$  for  $K_{MKL}$ .

Firstly, we performed some experiments to check how many samples are required to make  $\eta$  and  $\hat{\eta}$  similar. In our context two weights vectors are similar if they provide the same top kernels. The metric used for comparison is the Jaccard similarity score. Fig. 1 shows the Jaccard score between the indices of top kernels selected using  $\eta$  and  $\hat{\eta}$  with different threshold and different subsampling dimension. According to preliminary results, we fix the dimension of subset used in the first step to learn  $\hat{\eta}$  as 10% and the number of top kernels as 10%, except for MUTAG (the smallest datasets), where we use 20% of training data. To learn  $\hat{\eta}$  we chose a random subsampling of training data and perform a 10-fold cross

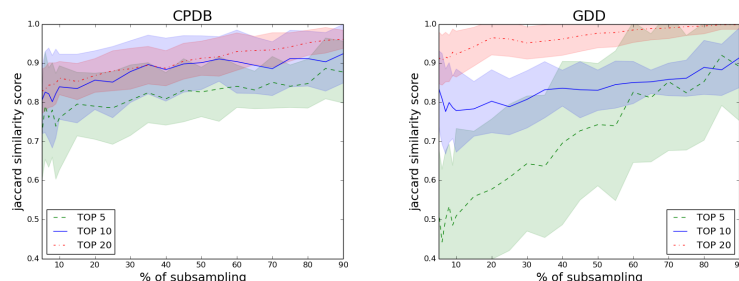


Fig. 1: Average jaccard values and standard deviation in 50 runs between the indices of top kernels using a subset of training data and using the whole set.

dataset	# train	$K_{VAL}$	$K_{MKL}$	$K_{SUM}$	$K_{MKL}^{sub}$	$K_{SUM}^{sub}$	$K_W^{sub}$
MUTAG	188	$0.90 \pm 0.08$ 18m	$0.93 \pm 0.06$ 40m	$0.93 \pm 0.06$ 8m	$0.93 \pm 0.05$ 3m	$0.93 \pm 0.05$ 4m	$0.93 \pm 0.05$ 4m
CPDB	684	$0.88 \pm 0.03$ 1h 16m	$0.88 \pm 0.03$ 8h 33m	$0.88 \pm 0.03$ 20m	$0.88 \pm 0.02$ 1h 37m	$0.89 \pm 0.03$ 5m	$0.88 \pm 0.02$ 5m
GDD	1178	$0.86 \pm 0.05$ >100h	$0.87 \pm 0.04$ >100h	$0.87 \pm 0.04$ >100h	$0.87 \pm 0.04$ 3h	$0.87 \pm 0.04$ 4h 15m	$0.80 \pm 0.04$ 4h 15m
AIDS	1503	$0.78 \pm 0.08$ 6h 39m	$0.78 \pm 0.08$ 50h	$0.78 \pm 0.08$ 2h 44m	$0.78 \pm 0.08$ 11h 10m	$0.77 \pm 0.07$ 1h 4m	$0.77 \pm 0.07$ 1h 4m
NCI1	4110	$0.88 \pm 0.04$ 39h	-	$0.89 \pm 0.04$ 7h 24m	$0.88 \pm 0.04$ 98h	$0.88 \pm 0.05$ 2h 1m	$0.88 \pm 0.04$ 2h 1m
NCI109	4127	$0.87 \pm 0.03$ 44h	-	$0.87 \pm 0.03$ 7h 36m	$0.87 \pm 0.02$ 99h	$0.87 \pm 0.03$ 1h 47m	$0.87 \pm 0.02$ 1h 47m
CAS	4337	$0.92 \pm 0.01$ 38h	-	$0.92 \pm 0.01$ 4h 26m	$0.92 \pm 0.01$ >100h	$0.92 \pm 0.01$ 1h 34m	$0.92 \pm 0.01$ 1h 34m

Table 1: AUC score, standard deviation and computational time for all baselines and all subsampling proposed methods.

validation using EasyMKL to combine kernels and SVM as base learner. Then we fit the model with the best combination and select the top kernels. In the second step we calculate the top kernel matrices using the whole training data and combine it to get the kernel. We propose 3 different methods to perform this combination: (1)  $K_{MKL}^{sub}$ : using the EasyMKL algorithm, (2)  $K_{SUM}^{sub}$ : calculating the average of kernels, (3)  $K_W^{sub}$ : using the same weights (rescaled) from the first learning step. In all of these methods we use a SVM as base learner. The hyper-parameters sets are the same used in baselines. Table 1 shows the AUC score, standard deviation and computation time<sup>2</sup>. used to make kernels and models for all problems.

## 5 Discussion and conclusion

We have presented an efficient approach to model selection for graph kernels based on the multiple kernel learning framework. We have shown how working

<sup>2</sup>We used an Intel(R) Xeon(R) CPU E5-2680 2.70GHz, 256GB (RAM).

on small subsets (up to 10%) of the data yields results that are stable and accurate enough to select the optimal combination of kernels via EasyMKL. We have investigated three methods to efficiently combine a subset of kernels while preserving at the same time an high AUC score for the downstream machine learning algorithms. Finally, we showed how we can use the simple summation of the selected kernels, avoiding the expensive step of running MKL on the full training set, to scale the MKL approach to very large datasets. In future work we will extend the library of graph kernels to obtain a comprehensive and easy to use software package.

## References

- [1] Fabio Aioli, Michele Donini, Nicolò Navarin, and Alessandro Sperduti. Multiple Graph-Kernel Learning. In *IEEE Symposium on Computational Intelligence and Data Mining*, pages 1607 – 1614, Cape Town, 2015. IEEE.
- [2] Carlo M. Massimo, Nicolò Navarin, and Alessandro Sperduti. Hyper-Parameter Tuning for Graph Kernels via Multiple Kernel Learning. In *Neural Information Processing of ICONIP, Kyoto, Japan, October 16–21, 2016, Part II*, pages 214–223. Springer, 2016.
- [3] Hisashi Kashima, Koji Tsuda, and Akihiro Inokuchi. Marginalized kernels between labeled graphs. In Tom Fawcett and Nina Mishra, editors, *ICML*, pages 321–328. AAAI Press, 2003.
- [4] Pierre Mahé, Nobuhisa Ueda, Tatsuya Akutsu, Jean-Luc Perret, and Jean-Philippe Vert. Extensions of marginalized graph kernels. In *Twenty-first international conference on Machine learning - ICML '04*, page 70, New York, New York, USA, 2004. ACM Press.
- [5] K.M. Borgwardt and Hans-Peter Kriegel. Shortest-Path Kernels on Graphs. In *ICDM*, pages 74–81, Los Alamitos, CA, USA, 2005. IEEE.
- [6] Nino Shervashidze, Kurt Mehlhorn, Tobias H Petri, S V N Vishwanathan, Karsten M Borgwardt, Tobias H Petri, Kurt Mehlhorn, and Karsten M Borgwardt. Efficient graphlet kernels for large graph comparison. In Max Welling, editor, *AISTATS*, volume 5, pages 488–495, Clearwater Beach, Florida, USA, 2009. CSAIL.
- [7] Markus Heinonen, Juho Rousu, N Välimäki, and V Mäkinen. Efficient Path Kernels for Reaction Function Prediction. In *BIOINFORMATICS 2012 - Proceedings of the International Conference on Bioinformatics Models, Methods and Algorithms*, pages 202–207, 2012.
- [8] Fabrizio Costa and Kurt De Grave. Fast neighborhood subgraph pairwise distance kernel. In *ICML*, pages 255–262. Omnipress, 2010.
- [9] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. Weisfeiler-Lehman Graph Kernels. *JMLR*, 12:2539–2561, 2011.
- [10] Giovanni Da San Martino, Nicolò Navarin, and Alessandro Sperduti. Ordered Decompositional DAG Kernels Enhancements. *Neurocomputing*, 192:92–103, 2016.
- [11] Mehmet Gonen and Ethem Alpaydin. Multiple Kernel Learning Algorithms. *Journal of Machine Learning Research*, 12:2211–2268, 2011.
- [12] Serhat Bucak, Rong Jin, and Anil Jain. Multiple Kernel Learning for Visual Object Recognition: A Review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(7):1354–1369, 2014.
- [13] John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.
- [14] Andreas Maurer and Massimiliano Pontil. Structured sparsity and generalization. *Journal of Machine Learning Research*, 13:671–690, 2012.
- [15] Corinna Cortes, Mehryar Mohri, and Afshin Rostamizadeh. Generalization bounds for learning kernels. *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 247—254, 2010.
- [16] Fabio Aioli and Michele Donini. EasyMKL: a scalable multiple kernel learning algorithm. *Neurocomputing*, pages 1–10, 2015.