Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

EasyMKL: a scalable multiple kernel learning algorithm

Fabio Aiolli, Michele Donini

University of Padova - Department of Mathematics Via Trieste, 63, 35121 Padova - Italy

A R T I C L E I N F O

Article history:
Received 1 July 2014
Received in revised form
18 October 2014
Accepted 5 November 2014

Keywords:
Multiple kernel learning
Kernel methods
Feature selection
Feature learning

A B S T R A C T

The goal of Multiple Kernel Learning (MKL) is to combine kernels derived from multiple sources in a data-driven way with the aim to enhance the accuracy of a target kernel machine. State-of-the-art methods of MKL have the drawback that the time required to solve the associated optimization problem grows (typically more than linearly) with the number of kernels to combine. Moreover, it has been empirically observed that even sophisticated methods often do not significantly outperform the simple average of kernels. In this paper, we propose a time and space efficient MKL algorithm that can easily cope with hundreds of thousands of kernels and more. The proposed method has been compared with other baselines (random, average, etc.) and three state-of-the-art MKL methods showing that our approach is often superior. We show empirically that the advantage of using the method proposed in this paper is even clearer when noise features are added. Finally, we have analyzed how our algorithm changes its performance with respect to the number of examples in the training set and the number of kernels combined.

1. Introduction

Large margin kernel based algorithms are recognized state-of-the-art algorithms for data mining applications. Besides the good performance they generally offer, the definition of a kernel allows a convenient way to easily inject into the classifier any available background knowledge one can have about a particular domain. Any positive definite kernel matrix implicitly specifies an inner product in a Hilbert space where large-margin techniques can be used for learning.

Recently, there has been a growing interest of researchers devoted to investigate on how these kernels can be learned from data. This is a big challenge and can potentially lead to significant improvements on a classifier. It is in fact quite established that the choice of the right features (i.e. the kernel) dramatically influences the performance of a classifier more than the classifier itself.

Kernel learning is a paradigm which is often adopted within a semi-supervised learning setting [1,2]. The goal of kernel learning is to learn the kernel matrix using available data (labeled and possibly unlabeled examples) optimizing an objective function that enforces the agreement between the kernel and the set of i.i.d. labeled data, e.g., by maximizing their alignment [3]. On the other hand, unlabeled data are typically used to regularize the generated models by constraining the discriminant function to be smooth (that is, it should not vary too much on similar examples).

Multiple Kernel Learning (MKL), see for example [4] for a recent and quite exhaustive survey, is a popular paradigm used to learn kernels. The kernel computed by these techniques are (generally linear) combinations of previously defined *weak* kernels. The main rationale behind this kind of methods is that they can alleviate the effort of the user on defining good kernels for a given problem. Using the MKL framework, the algorithm itself can be able to select the best combination among a battery of predefined and reasonable weak kernels.

In this paper, we focus on multiple kernel learning with positive and linear combination parameters, that is, MKL in the form

$$\mathbf{K} = \sum_{r=1}^{R} \boldsymbol{\eta}_r \mathbf{K}_r, \quad \boldsymbol{\eta}_r \geq 0.$$

This typology of algorithm is based on 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 bounds* for the linear combination of kernels [5,6].

According to [4], the majority of existing MKL approaches can be divided into the following two categories. *Fixed or Heuristic rule* techniques apply some fix rule, like simple summation or product of the kernels, or simple heuristic to find the combination parameters. The result usually obtained by these methods is scalable with respect to the number of kernels combined but their effectiveness will critically depend on the domain at hand. On the other side, *Optimization based* approaches learn the combination parameters by solving an optimization problem that can be integrated in the learning machine (e.g. structural risk based target function) or formulated as a different model (e.g. alignment, or other kernel similarity maximization).

http://dx.doi.org/10.1016/j.neucom.2014.11.078
0925-2312/© 2015 Elsevier B.V. All rights reserved.

Generally speaking, best performing approaches to MKL necessitate of complex optimization including semidefinite programming (SDP), or quadratically constrained quadratic programming (QCQP), just to name a few, thus making these approaches unpractical when the number of kernels to combine is large, say in the order of hundreds. Further, these methods often need to keep in memory the whole set of weak kernels, or at least the submatrices corresponding to the training data pairs. This will exhaust the memory soon when coping with hundreds of examples and hundreds of weak kernels.

The same MKL idea can be used in two different scenarios. In the first and more popular case, a small number of strong kernels have to be combined each representing a different (possibly orthogonal) view of the same task. Typically 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. Alternatively, the MKL paradigm can be exploited to combine a very large set of weak kernels aiming at boosting their accuracy. In this way, the final combination will represent a weighted combination of the different subsets of features. In this paper, we focus on this second approach because, we think, at least in principle, it can really boost the performance as it basically performs a data-driven feature learning/selection/weighting.

With this second view in mind, we see that for MKL to be effective we need to combine many kernels. For this reason, being non-scalable becomes a stringent issue for a MKL method. On the other side, simpler fixed rule algorithms are definitely more scalable in general but they are far less flexible and tend to become ineffective when coping with many kernels and noise. In this paper, we propose a very efficient algorithm which is able to cope with thousands of kernels efficiently and we empirically demonstrate the effectiveness of the methods proposed that almost always is more accurate than the baselines especially when noise is present in the weak kernels.

A summary of the paper is the following. In Section 2 we show the notation used in this paper for classification problems. In Section 3, the *Kernel Optimization of the Margin Distribution (KOMD)* classification algorithm, is presented. In Section 4 the proposed multiple kernel learning algorithm, namely EasyMKL, is presented. In Section 5, experiments are presented with respect to different dimensions, namely, the effectiveness, the scalability, and the stability of the method proposed against different baselines and state-of-the-art methods. Finally, in Section 6 we draw conclusions.

2. Notation

Throughout this paper, we consider a classification problem with training examples defined by $\{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_l, y_l)\}$, and test examples defined by $\{(\mathbf{x}_{l+1}, y_{l+1}), \ldots, (\mathbf{x}_L, y_L)\}$, $\mathbf{x}_i \in \mathbb{R}^m$, $y_i \in \{-1, +1\}$. We use $\mathbf{X} \in \mathbb{R}^{L \times m}$ to denote the matrix where examples are arranged in rows and $\mathbf{y} \in \mathbb{R}^L$ the vector of labels. The matrix $\mathbf{K} \in \mathbb{R}^{L \times L}$ denotes the complete kernel matrix containing the kernel values of each (training and test) data pair. Further, we indicate with an hat, like for example $\hat{\mathbf{X}} \in \mathbb{R}^{l \times m}, \hat{\mathbf{y}} \in \mathbb{R}^l$, and $\hat{\mathbf{K}} \in \mathbb{R}^{l \times l}$, the submatrices (or subvectors) obtained considering training examples only.

Given 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. More formally:

$$\hat{\Gamma} = \left\{ \boldsymbol{\gamma} \in \mathbb{R}^l_+ \mid \sum_{i \in \oplus} \gamma_i = 1, \sum_{i \in \ominus} \gamma_i = 1 \right\}.$$

Note that any element $\boldsymbol{\gamma} \in \hat{\Gamma}$ 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.

3. Playing with margin and the KOMD algorithm

In [7] a game theoretic interpretation as a two-player zero-sum game has been proposed for the problem of margin maximization in a classification task. Specifically, the classification task has been split into two phases. Firstly, in the ranking phase, a total order of the examples is introduced. Secondly, the pure binary classification is performed by applying a threshold to the ranking of the examples obtained in the first phase.

In particular, in the ranking phase, the task is to learn a unitary norm vector $\mathbf{w}$ such that

$$\mathbf{w}^\top(\phi(\mathbf{x}_\oplus) - \phi(\mathbf{x}_\ominus)) > 0$$

for most of positive $(\mathbf{x}_\oplus)$ and negative $(\mathbf{x}_\ominus)$ instance pairs in the training data. The game basically consists of one player that has to choose one vector of unitary norm $\mathbf{w}$ and the other that picks pairs of positive–negative examples according to two distributions $\boldsymbol{\gamma}^+$ and $\boldsymbol{\gamma}^-$ over the positive and negative examples, respectively. The value of the game is the expected margin obtained, that is $\mathbf{w}^\top(\phi(\mathbf{x}_p) - \phi(\mathbf{x}_n)), \mathbf{x}_p \sim \boldsymbol{\gamma}^+, \mathbf{x}_n \sim \boldsymbol{\gamma}^-$. The goal of the first player is to maximize this value while the second player wants to minimize it. This problem is equivalent to the hard SVM and can be solved efficiently by optimizing a simple linearly constrained convex function on variables $\boldsymbol{\gamma} \in \hat{\Gamma}$, namely,

$$\underset{\boldsymbol{\gamma} \in \hat{\Gamma}}{\text{minimize}} \ \underbrace{\boldsymbol{\gamma}^\top \hat{\mathbf{Y}}\hat{\mathbf{K}}\hat{\mathbf{Y}}\boldsymbol{\gamma}}_{D(\boldsymbol{\gamma})}.$$

It can be seen that the vector $\boldsymbol{\gamma}^* \in \hat{\Gamma}$ minimizing $D(\boldsymbol{\gamma})$ identifies the two nearest points in the convex hulls of positive and negative examples, respectively, in the feature space of the kernel $\mathbf{K}$.

Furthermore, a quadratic regularization over $\boldsymbol{\gamma}$ is introduced, namely, $R(\boldsymbol{\gamma}) = \boldsymbol{\gamma}^\top\boldsymbol{\gamma}$, that makes the player to prefer optimal distributions (strategies) with low variance. In fact, let p (resp. n) be the number of positive examples (resp. negative examples) in the training set, then

$$\mathbb{E}[\boldsymbol{\gamma}_+] = 1/p \quad \text{and} \quad \mathbb{E}[\boldsymbol{\gamma}_-] = 1/n$$

is always true by construction, and $\mathbb{E}[\mathbf{v}]$ denotes the expected value of elements in a vector $\mathbf{v}$. It follows that

$$\text{Var}(\boldsymbol{\gamma}_+) = \mathbb{E}[\boldsymbol{\gamma}_+^2] - \mathbb{E}[\boldsymbol{\gamma}_+]^2 = \|\boldsymbol{\gamma}_+\|^2 - p^{-2}$$
$$\text{Var}(\boldsymbol{\gamma}_-) = \mathbb{E}[\boldsymbol{\gamma}_-^2] - \mathbb{E}[\boldsymbol{\gamma}_-]^2 = \|\boldsymbol{\gamma}_-\|^2 - n^{-2},$$

obtaining $R(\boldsymbol{\gamma}) \propto \text{Var}(\boldsymbol{\gamma}_+) + \text{Var}(\boldsymbol{\gamma}_-)$.

The final best strategy for $\boldsymbol{\gamma}$ will be given by solving the optimization problem

$$\min_{\boldsymbol{\gamma} \in \hat{\Gamma}} (1-\lambda)D(\boldsymbol{\gamma}) + \lambda R(\boldsymbol{\gamma}).$$

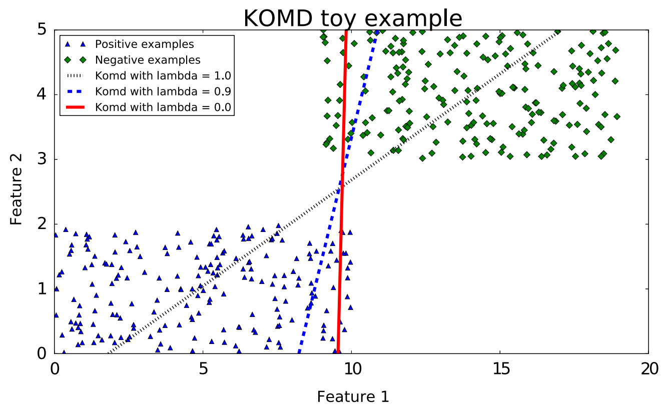


Fig. 1. KOMD solutions of the first phase found using different λ in a simple toy classification problem. The ranking among the examples is performed evaluating the *orthogonal projection* of the examples over the line defined by the solution.

The regularization parameter λ has two critical points: $\lambda = 0$ and $\lambda = 1$. When $\lambda = 0$, as we have shown before, there is no regularization, so the solution is the same as the hard SVM one, whereas when $\lambda = 1$ the minimization problem reduces to

$$\min_{\gamma \in \hat{\Gamma}} R(\gamma)$$

and the optimal solution is analytically defined by the vector of uniform distributions over positive and negative examples, that is, $\gamma_i^u = 1/p$ when $y_i = +1$, and $\gamma_i^u = 1/n$ when $y_i = -1$. In this case, the objective solution is the squared distance from the positive and negative centroids in feature space. The external parameter $\lambda \in (0,1)$ allows us to select the correct trade-off. Clearly, a correct selection of this parameter is fundamental if we are interested in finding the best performance for a classification task and this is usually made by validating on training data. In Fig. 1, an example of the solutions found by the above algorithm for a toy problem varying the value of λ is depicted.

Once the model is learned from training data, the evaluation on a new generic example $\mathbf{x}$ is obtained by

$$f(\mathbf{x}) = \mathbf{w}^\top \phi(\mathbf{x}) = \sum_i y_i \gamma_i \mathbf{K}(\mathbf{x}_i, \mathbf{x}) = \mathbf{K}_{tr}(\mathbf{x}) \hat{\mathbf{Y}} \gamma,$$

where $\mathbf{K}_{tr}(\mathbf{x}) = [\mathbf{K}(\mathbf{x}_1, \mathbf{x}), \ldots, \mathbf{K}(\mathbf{x}_l, \mathbf{x})]^\top$, i.e. the vector containing the kernel values with the training examples for $\mathbf{x}$.

When a pure binary classification is required, which is not the case in our work, then the second phase of the algorithm is also performed. The threshold is set corresponding to the point standing in the middle between the optimal point in the convex hull of positive examples and the one in the convex hull of negative examples, that is

$$\theta = \frac{1}{2}\left(\sum_{i \in \oplus} \gamma_i f(\mathbf{x}_i) + \sum_{i \in \ominus} \gamma_i f(\mathbf{x}_i) \right) = \frac{1}{2} \sum_i \gamma_i f(\mathbf{x}_i) = \frac{1}{2} \gamma^\top \hat{\mathbf{K}} \hat{\mathbf{Y}} \gamma.$$

Note that, when $\lambda = 0$ this choice corresponds exactly to the optimal hyperplane of SVM. Finally, a new example $\mathbf{x}$ will be classified according to $\text{sign}(f(\mathbf{x}) - \theta)$.

In the following we will refer to the first phase of the algorithm discussed in this section as KOMD (Kernelized Optimization of the Margin Distribution).

4. EasyMKL

In MKL we want to find the best combination parameters for a set of predefined kernel matrices. In our context, this is done by learning a vector of coefficients $\boldsymbol{\eta}$ that forms the combined kernel according to

$$\mathbf{K} = \sum_{r=1}^{R} \eta_r \mathbf{K}_r, \quad \eta_r \geq 0.$$

Clearly, we must restrict the possible choices of such a matrix $\mathbf{K}$ and this can be made by regularizing the learning process. So, we pose the problem of learning the kernel combination as a min–max problem over variables γ and $\boldsymbol{\eta}$. Specifically, we propose to maximize the distance between positive and negative examples with a unitary norm vector $\boldsymbol{\eta}$ as the weak kernel combination vector, that is

$$\max_{\boldsymbol{\eta}: \|\boldsymbol{\eta}\| = 1} \min_{\gamma \in \Gamma} \underbrace{(1-\lambda)\gamma^\top \hat{\mathbf{Y}} \left(\sum_r^R \eta_r \hat{\mathbf{K}}_r \right) \hat{\mathbf{Y}} \gamma + \lambda \|\gamma\|^2}_{Q(\boldsymbol{\eta}, \gamma)}.$$

Considering $\mathbf{d}(\gamma)$ the vector with the rth entry defined as

$$\mathbf{d}_r(\gamma) = \gamma^\top \hat{\mathbf{Y}} \hat{\mathbf{K}}_r \hat{\mathbf{Y}} \gamma,$$

we obtain that $Q(\boldsymbol{\eta}, \gamma)$ is equal to $(1-\lambda)\boldsymbol{\eta}^\top \mathbf{d}(\gamma) + \lambda \|\gamma\|_2^2$ and we can rewrite the original problem as

$$\min_{\gamma \in \Gamma} \max_{\boldsymbol{\eta}: \|\boldsymbol{\eta}\| = 1} Q(\boldsymbol{\eta}, \gamma) = \min_{\gamma \in \Gamma} \max_{\|\boldsymbol{\eta}\| = 1} (1-\lambda)\boldsymbol{\eta}^\top \mathbf{d}(\gamma) + \lambda \|\gamma\|_2^2. \tag{1}$$

Now, it is possible to see that the vector $\boldsymbol{\eta}^*$ maximizing the function $Q(\boldsymbol{\eta}, \gamma)$ above has a simple analytic solution:

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

Plugging this solution into the min–max problem, we obtain

$$\min_{\gamma \in \Gamma} Q(\boldsymbol{\eta}^*, \gamma) = \min_{\gamma \in \Gamma} (1-\lambda)\|\mathbf{d}(\gamma)\|_2 + \lambda \|\gamma\|_2^2.$$

The optimal γ will be the (regularized) minimizer of the 2-norm of the vector of distances. This minimizer is not difficult to find as this is a convex function though not quadratic. In order to simplify the problem further we prefer to minimize an upper-bound instead corresponding to the 1-norm of the vector of distances, thus obtaining

$$\min_{\gamma \in \Gamma} (1-\lambda)\|\mathbf{d}(\gamma)\|_1 + \lambda \|\gamma\|_2^2 = \min_{\gamma \in \Gamma} (1-\lambda)\gamma^\top \hat{\mathbf{Y}} \left(\sum_r^R \hat{\mathbf{K}}_r \right) \hat{\mathbf{Y}} \gamma + \lambda \|\gamma\|_2^2. \tag{2}$$

Interestingly, the obtained minimization problem is the same as the KOMD problem where the kernel matrix has been replaced with the simple sum of the weak kernel matrices. This will turn out to be very important for the efficiency (especially in terms of space) of the method as will be explained later on. In the following, we refer to this algorithm as *EasyMKL*.

From a more theoretical point of view, modifying the original problem using the 1-norm upper-bound, in fact, we are changing the optimal solution $\boldsymbol{\eta}^*$ in the initial problem (Eq. (1)) with a new $\tilde{\boldsymbol{\eta}^*}$, where

$$\tilde{\boldsymbol{\eta}^*} = \boldsymbol{\eta}^* \frac{\|\mathbf{d}(\gamma)\|_1}{\|\mathbf{d}(\gamma)\|_2}.$$

Consequently, the problem that we are solving is

$$\min_{\gamma \in \Gamma} (1-\lambda)\tilde{\boldsymbol{\eta}^*} \mathbf{d}(\gamma) + \lambda \|\gamma\|_2^2 = \min_{\gamma \in \Gamma} (1-\lambda)\boldsymbol{\eta}^* \frac{\|\mathbf{d}(\gamma)\|_1}{\|\mathbf{d}(\gamma)\|_2} \mathbf{d}(\gamma) + \lambda \|\gamma\|_2^2. \tag{3}$$

From this new formulation, we note that, in fact, we have just added a multiplicative coefficient $\|\mathbf{d}(\gamma)\|_1 / \|\mathbf{d}(\gamma)\|_2$ to the original optimal solution $\boldsymbol{\eta}^*$. From *Hölder's inequality* this coefficient is bounded by the number of kernels and we have that

$$1 \leq \frac{\|\mathbf{d}(\gamma)\|_1}{\|\mathbf{d}(\gamma)\|_2} \leq \sqrt{R}.$$

The value of the ratio between the 1-norm and the 2-norm tends to 1 if the vector $\mathbf{d}(\gamma)$ is very sparse, while, it tends to $\sqrt{R}$ when the values in the vector $\mathbf{d}(\gamma)$ are similar. In other words, using the proposed formulation, we are promoting sparse solutions of the vector $\mathbf{d}(\gamma)$ solving the minimum problem. Then, solving the problem in Eq. (3), the vector of the weight $\boldsymbol{\eta}^* = \mathbf{d}(\gamma)/\|\mathbf{d}(\gamma)\|_2$ will result sparser than the solution of the original problem. Clearly, Eq. (3) has the same solution as the *EasyMKL* minimum problem in Eq. (2).

A final consideration we can do here is that the quality of a kernel does not change if it is multiplied by a positive constant. However, our formulation makes particularly clear that, if kernels with different traces are present in the combination, they have unequal impact in the MKL optimization problem. In fact, most of the bounds related to the difference between the true error and the empirical margin error (i.e. estimation error) change linearly with respect to the square root of the maximal trace among the combined kernels. Moreover, if we have that $\exists T > 0$ such that

$$T^2 \geq \mathbf{K}_r(x, x) \quad \forall r = 1, \ldots, R \ \forall x$$

the estimation error (fixed a specific task and without constants) is $\mathcal{O}(T)$ [5,6]. Thus, if not differently motivated, different traces should be avoided.

5. Experiments and results

In this section, the experiments we have performed to validate the EasyMKL approach are presented. First of all, an introduction of the experimental setting is given in Section 5.1. In Section 5.2 we show some results confirming that the maximization of the separation in feature space is a good criterion to pursue to obtain effective kernels. Moreover, an overview of the baselines and state-of-the-art methods is given in Section 5.3. In Section 5.4 we compare our algorithm against other methods with respect to the AUC over ranking tasks. In Section 5.5 we perform experiments concerning time and memory performance of EasyMKL. We tested the stability of EasyMKL by adding different amounts of noise in the data in Section 5.6. In Section 5.7 we analyze how the cardinality of the training set influences our algorithm. Finally, in Section 5.8 we describe experiments that we have performed to understand the difference in performance obtained by changing the number of weak kernels.

5.1. *Experimental setting*

Experiments have been performed comparing the proposed methods with other state-of-the-art MKL algorithms with respect to accuracy and computational performance. A total of 7 benchmark datasets with different characteristics and 7 different MKL methods have been considered. Namely, the datasets considered are *Diabetes* [8] (8 features, 768 examples), *Australian* [8] (14 features, 690 examples), *German* [8] (24 features, 1000 examples), *Splice* [8] (60 features, 1000 examples), *Batch2* [9] (128 features, 1244 examples), *Mush* [8] (112 features, 2000 examples), *Gisette* [10] (5000 features, 13,500 examples), that Table 1. Data have been scaled to the $[-1, +1]$ interval and the number of training examples selected for the training part is approximately 10% of the dataset. We decided to use relatively small training sets obtaining a final accuracy less influenced by the classifier (on which the combined kernel is applied) and more influenced by the real quality of a kernel. Further, we preferred having larger test sets and many different splits of each single dataset aimed at improving the significance of the overall evaluation.

In this work we want to demonstrate that we can efficiently perform a kind of feature selection/weighting via MKL and this can be done by selecting a very large set of weak kernels each one individually defined using a small subset of the original features.

Let $d \in \mathbb{N}$ and $\beta > 0$ be two parameters, then the rth kernel is constructed by random picking of a bag (replica is allowed) of features F_r, such that $|F_r| \leq d$, and constructing an RBF based weak kernel defined according to

$$\mathbf{K}_r(\mathbf{x}_i, \mathbf{x}_j) = \prod_{f \in F_r} e^{-\beta/|F_r|(\mathbf{x}_i^{(f)} - \mathbf{x}_j^{(f)})^2}.$$

Table 1
Datasets information: name, source, number of features, number of examples and cardinality of the training set.

Data set	Source	Features	Examples	N_{tr}
Diabetes	*UCI*	8	768	77
Australian	*Statlog*	14	690	69
German	*Statlog*	24	1000	100
Splice	*UCI*	60	1000	100
Batch2	*UCI*	128	1244	124
Mush	*UCI*	112	2000	200
Gisette	*NIPS03*	5000	13,500	1350

5.2. *Is data separation a good criterion to maximize?*

In this section, we present results showing that the maximization of the distance between positive and negative examples is a good criterion to pursue as it is positively correlated with the AUC obtained in a binary ranking task.

In particular, we designed a simple greedy algorithm which constructs a series of kernels with monotonically increasing data separation. For reader's convenience, we recall the formula of separation:

$$\min_{\gamma \in \hat{\Gamma}} \boldsymbol{\gamma}^\top \hat{\mathbf{Y}} \hat{\mathbf{K}} \hat{\mathbf{Y}} \boldsymbol{\gamma}.$$

The algorithm starts with a null kernel and considers sequentially the available weak kernels. At each iteration, the current kernel is updated with a weak kernel only if this addition increases the separation of training data. Note that, this algorithm corresponds to have $\boldsymbol{\eta}_r \in \{0, 1\}$. The algorithm is described in detail in Appendix A.

Fig. 2, an example of a plot obtained applying the greedy algorithm on the *Splice* dataset, using 10,000 weak kernels and generated as described in Section 5.1 ($d=5$), is given. This experiment clearly shows the strong correlation between the data separation and the AUC obtained in the test set using the produced kernel.

5.3. *Methods*

We have then considered three baseline methods for our experiments. The first method *SVM* is the classical SVM trained with all the features. This is only reported for the sake of completeness and just to give an idea of the difficulty of the datasets. Note that, the feature space in this case is different from MKL methods and results are not comparable. The second method, called Random Kernel (*Random*), consists of random picking from available kernels, which is equivalent to set only one $\boldsymbol{\eta}_r$ equal to 1 and all the others equal to 0. We decided to add this baseline as an indicator of how much information is brought from single kernels. Intentionally, the performance of this baseline could be really poor. Finally, the last baseline method is the Average Kernel (*Average*) where the same weight $\boldsymbol{\eta}_r$ is given to all the available weak kernels. Despite its simplicity, it is known (see [11]) that this

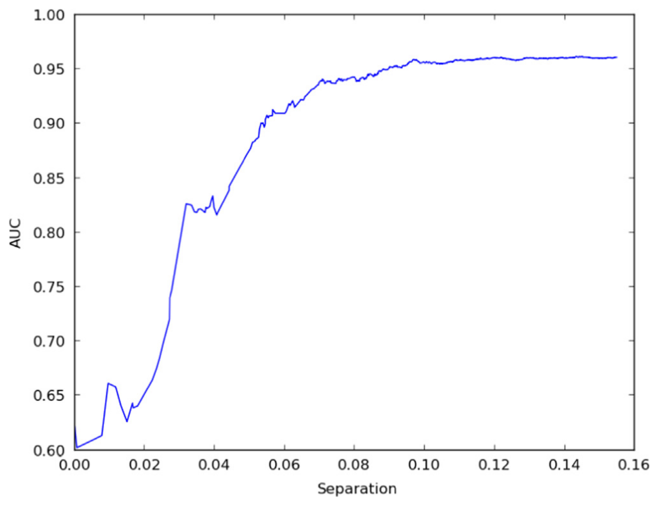


Fig. 2. Value of the quality function (separation) versus the AUC obtained using the greedy algorithm over the *Splice* dataset.

Table 2
AUC results on three datasets of SVM (RBF all features).

Average AUC (RBF all features)			
Algorithm	*Diabetes*	*Australian*	*German*
SVM	80.39%	91.37%	70.79%

Average AUC (RBF all features)			
Algorithm	*Splice*	*Batch2*	*Mush*
SVM	86.12%	95.20%	98.52%

Table 3
$AUC_{\pm std}$ results on three datasets of various MKL methods and baselines (RBF feature subset $d=5$).

Average AUC (RBF subset $d=5$)			
Algorithm	*Diabetes*	*Australian*	*German*
Random	$62.13_{\pm 6.20}\%$	$65.56_{\pm 21.32}\%$	$49.59_{\pm 11.68}\%$
Average	$63.01_{\pm 0.41}\%$	$92.09_{\pm 0.51}\%$	$\mathbf{73.04}_{\pm \mathbf{0.37}}\%$
SMKL	$77.52_{\pm 2.11}\%$	$91.64_{\pm 3.18}\%$	$69.71_{\pm 2.78}\%$
GMKL	$75.43_{\pm 3.23}\%$	$90.37_{\pm 4.01}\%$	$69.53_{\pm 3.87}\%$
GLMKL	$70.95_{\pm 2.75}\%$	$85.27_{\pm 3.56}\%$	$68.61_{\pm 3.02}\%$
EasyMKL	$\mathbf{79.91}_{\pm \mathbf{0.13}}\%$	$\mathbf{92.17}_{\pm \mathbf{0.11}}\%$	$72.71_{\pm 0.21}\%$

Average AUC (RBF subset $d=5$)			
Algorithm	*Splice*	*Batch2*	*Mushrooms*
Random	$52.61_{\pm 10.22}\%$	$81.14_{\pm 11.83}\%$	$28.58_{\pm 24.56}\%$
Average	$86.56_{\pm 0.25}\%$	$95.35_{\pm 0.43}\%$	$\mathbf{98.45}_{\pm \mathbf{0.49}}\%$
SMKL	$83.48_{\pm 2.15}\%$	$94.82_{\pm 2.18}\%$	$97.49_{\pm 3.41}\%$
GMKL	$83.70_{\pm 4.24}\%$	$92.53_{\pm 3.99}\%$	$97.39_{\pm 4.89}\%$
GLMKL	$85.36_{\pm 1.98}\%$	$94.54_{\pm 3.34}\%$	$96.38_{\pm 4.72}\%$
EasyMKL	$\mathbf{87.87}_{\pm \mathbf{0.18}}\%$	$\mathbf{98.98}_{\pm \mathbf{0.21}}\%$	$97.91_{\pm 0.24}\%$

Table 4
$AUC_{\pm std}$ results on three datasets of various MKL methods and baselines (RBF feature subset $d=10$).

Average AUC (RBF subset $d=10$)			
Algorithm	*Diabetes*	*Australian*	*German*
Random	$71.44_{\pm 7.05}\%$	$70.36_{\pm 12.59}\%$	$56.93_{\pm 7.14}\%$
Average	$71.94_{\pm 0.34}\%$	$92.11_{\pm 0.42}\%$	$\mathbf{73.33}_{\pm \mathbf{0.36}}\%$
SMKL	$75.21_{\pm 3.41}\%$	$87.94_{\pm 2.12}\%$	$70.92_{\pm 2.56}\%$
GMKL	$72.01_{\pm 5.76}\%$	$86.88_{\pm 3.17}\%$	$70.89_{\pm 3.01}\%$
GLMKL	$71.85_{\pm 5.01}\%$	$84.80_{\pm 2.09}\%$	$69.75_{\pm 2.78}\%$
EasyMKL	$\mathbf{79.61}_{\pm \mathbf{0.16}}\%$	$\mathbf{92.17}_{\pm \mathbf{0.11}}\%$	$73.21_{\pm 0.11}\%$

Average AUC (RBF subset $d=10$)			
Algorithm	*Splice*	*Batch2*	*Mushrooms*
Random	$56.76_{\pm 8.52}\%$	$85.30_{\pm 8.24}\%$	$44.58_{\pm 37.26}\%$
Average	$90.42_{\pm 0.34}\%$	$94.70_{\pm 0.53}\%$	$\mathbf{98.86}_{\pm \mathbf{0.19}}\%$
SMKL	$86.84_{\pm 3.42}\%$	$95.07_{\pm 4.87}\%$	$97.48_{\pm 2.37}\%$
GMKL	$84.48_{\pm 6.02}\%$	$94.92_{\pm 6.97}\%$	$96.99_{\pm 4.32}\%$
GLMKL	$86.41_{\pm 5.51}\%$	$94.18_{\pm 6.09}\%$	$95.25_{\pm 3.12}\%$
EasyMKL	$\mathbf{91.19}_{\pm \mathbf{0.18}}\%$	$\mathbf{97.08}_{\pm \mathbf{0.16}}\%$	$98.29_{\pm 0.12}\%$

is a strong baseline that beats other more advanced techniques quite often and can obtain very good results. We have then considered three state-of-the-art algorithms of MKL with SVM. All these three algorithms are in the optimization based family of MKL methods. A MATLAB implementation[1] of these methods by Mehmet Gönen and Ethem Alpaydin [4] has been used.

[1] http://www.cmpe.boun.edu.tr/~gonen/mkl.

- *Simple MKL* (*SMKL*): An iterative algorithm by Rakotomamonjy [12] that implements a linear approach with kernel weights in a simplex. Basically SMKL works by repeating two main steps:
  - A SVM optimization problem defined on current weights.
  - Updating of the kernel weights using a gradient function.
- *Generalized MKL* (*GMKL*): The second algorithm, by Varma and Babu [13], is called generalized multiple kernel learning (GMKL). GMKL exploits a nonlinear approach and tackles the problem with a technique that regularizes both the hyperplane weights and the kernel combination weights. This algorithm

tries to optimize a non-convex problem, different from SMKL but, in general, with better results [4].

- *Lasso-based MKL* (*GLMKL*): The third algorithm used as baseline is called Lasso-based MKL algorithm by Kloft [14] and Xu [15]. GLMKL considers a regularization of kernel weights with 1-norm and finds the kernel weights by solving a small QP problem at each iteration.
- *SMO Projected Gradient Descent Generalized MKL* (*SPG-GMKL*): The last algorithm is inspired by the GMKL and learns simultaneously both kernel and SVM parameters. This particular algorithm [16] exploits an efficient and highly scalable implementation and is a state-of-the-art method with respect to the computational performance.

The validation of the methods was performed before each type of experiment. A subset of examples (validation set) was selected for each method and dataset and was only used to select the best hyperparameters. The kernel machines (SVM and KOMD) were validated using the same subset of data to obtain a fair comparison. The performance of SVM and KOMD was similar in most of the cases, thus confirming the findings in [7].

Table 5
Average $AUC_{\pm std}$ on the *Gisette* (NIPS2003 feature selection challenge) of EasyMKL (with RBF subset) and comparison with the Average baseline.

Algorithm	RBF $d=5$	RBF $d=10$	RBF $d=20$
Average	$95.52_{\pm 0.15}\%$	$94.72_{\pm 0.13}\%$	$93.65_{\pm 0.13}\%$
EasyMKL	$\mathbf{95.83}_{\pm \mathbf{0.12}}\%$	$\mathbf{95.40}_{\pm \mathbf{0.10}}\%$	$\mathbf{94.64}_{\pm \mathbf{0.11}}\%$

5.4. AUC comparisons

The quality of the different combined kernels has been evaluated by means of AUC (Area Under Curve), which measures how good the induced ranking order is with respect to the target classification task. In this way, we do not necessitate of a threshold setting which is an additional, and useless, degree of freedom when evaluating kernels performance. Weak kernels have been generated according to the method depicted in Section 5.1. A number of 10,000 weak kernels have been constructed for each dataset. Two different values for d, that is $d \in \{5, 10\}$, have been used in the generation algorithm with the RBF parameter β_0 obtained by model selection on a standard SVM. The experimental setting is summarized below:

1. Repeat for $r=1$ to $R=10{,}000$: pick a random $p \in \{1, \dots, d\}$ and generate a weak kernel K_r using p features picked randomly (with replacement).
2. Combine $\{\mathbf{K}_r\}_{r=1}^{R}$ with a MKL algorithm to obtain a kernel $\mathbf{K}$.
3. Rank test examples using $\mathbf{K}$ and SVM (KOMD for the proposed method) and evaluate the quality of the ranking with the AUC metric.

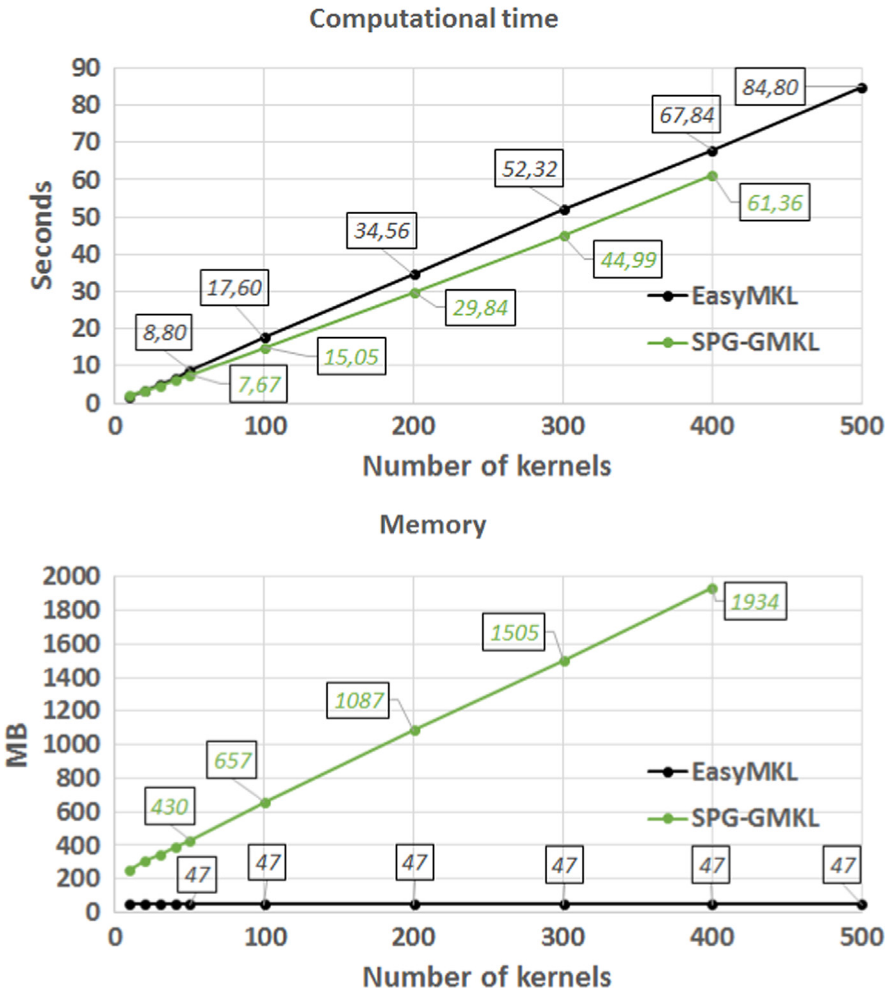


Fig. 3. Time and memory used by SPG-GMKL and EasyMKL with different amounts of kernels combined. The kernels are created using all the features of the *Splice* dataset. The experiment with 500 kernels of SPG-GMKL is not reported as it exceeds the limit of 2 GB of memory.

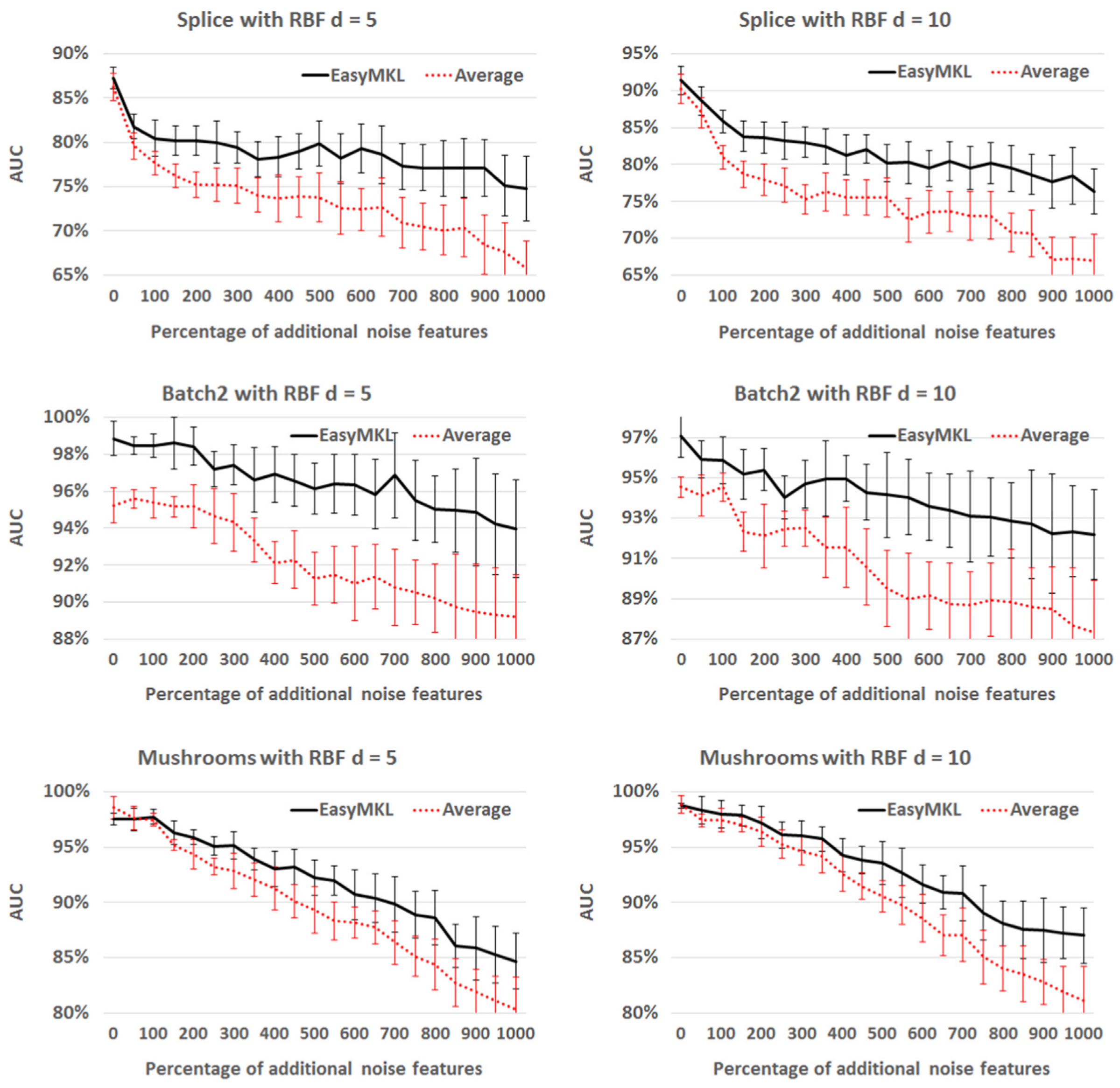


Fig. 4. AUC comparisons with standard deviation of EasyMKL against the Average Kernel algorithm with respect to different percentages of additional noise features using different datasets (*Splice*, *Batch2* and *Mush*) and two different values of *d* (5 and 10).

The same experiments have been repeated 1000 times averaging the AUC results in order to improve the significance of the evaluation. Aiming at setting our experiments as fair as possible, the KOMD algorithm has been used with the kernel produced by our MKL method, while SVM has been used on all the others. For what concerns the SVM baseline, RBF kernels with all the features have been used in this case. The obtained results are summarized in Table 2.

The results obtained with the MKL algorithms are summarized in Tables 3 and 4 (for $d=5$ and $d=10$) showing that standard MKL methods do not have performances significantly better than the simple kernel averaging. On the other side, EasyMKL has significantly better AUC in four out of six datasets. Also, EasyMKL has a small standard deviation (std) with respect to the other methods and this fact highlights the stability of our algorithm. This trend is confirmed in Table 5 where the proposed method is always significantly better than the average baseline. Unfortunately, given the relatively large number of examples of the *Gisette* dataset, it was not possible to run state-of-the-art MKL methods on this data without running out the memory.

5.5. *Performance in time and memory*

We have also performed experiments[2] to evaluate the computational time and the memory used by EasyMKL. We compared our algorithm with a state-of-the-art algorithm called SPG-GMKL [16]. A C++ implementation of SPG-GMKL provided by the authors[3] has been used. SPG-GMKL with this implementation is more than 100 times faster than SimpleMKL [16]. The *Splice* dataset has been selected for this experiment with 100 training examples. A variable number of RBF kernels have been generated with a parameter β picked randomly in $(0, 1)$ and using all the 60 features of the *Splice* dataset. We have studied the performance in time and memory, fixed an upper bound of 2 GB for the memory. Results are reported in Fig. 3.

[2] For these experiments we used a CPU Intel Core i7-3632QM @ 2.20GHz 2.20GHz.

[3] http://www.cs.cornell.edu/~ashesh/pubs/code/SPG-GMKL/download.html

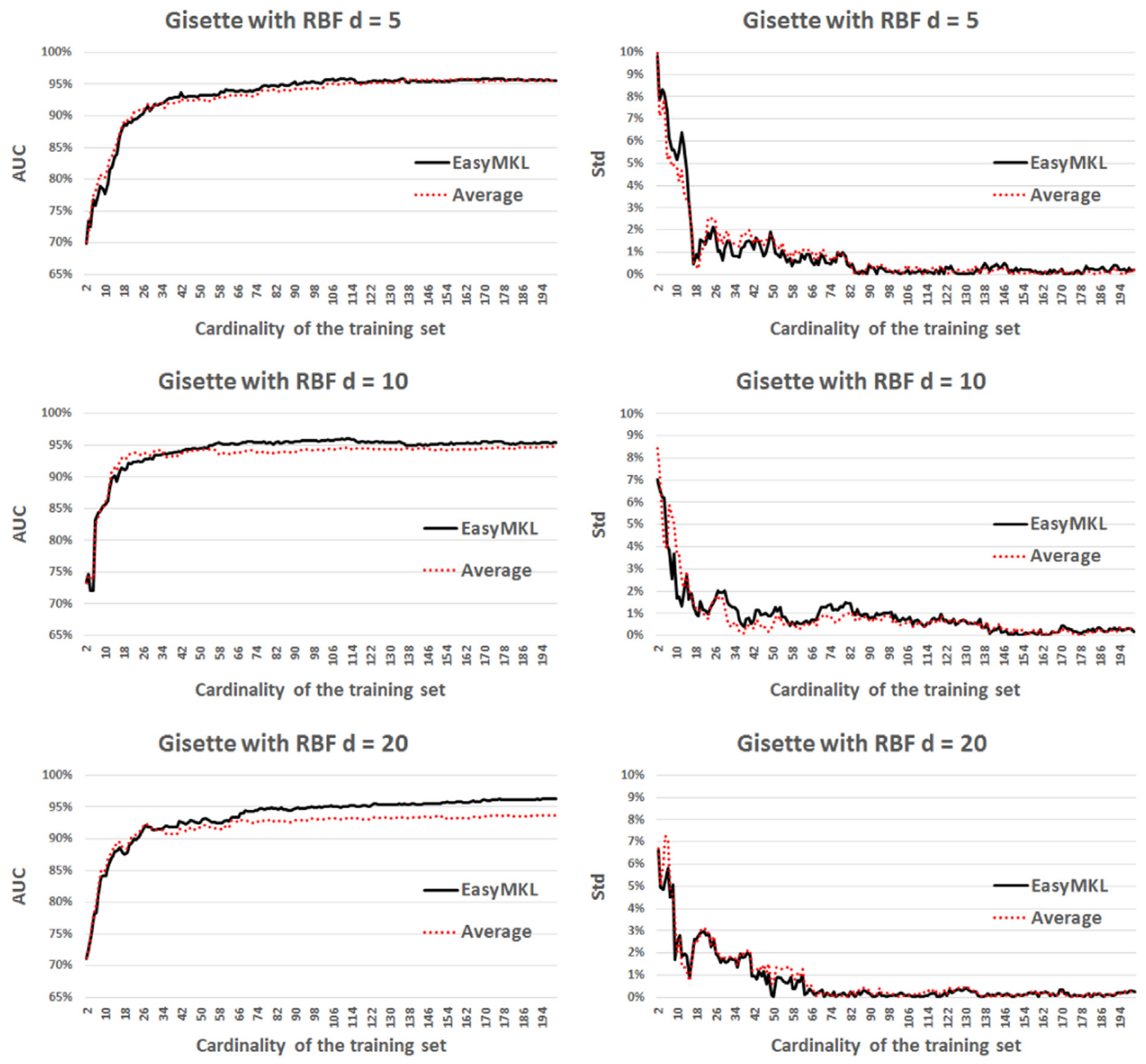


Fig. 5. AUC comparisons (left) with standard deviation (right) of EasyMKL against the Average Kernel algorithm with respect to different cardinality of the training set using the *Gisette* dataset and three different values of d (5,10 and 20).

Based on our experiments, the time complexity of SPG-GMKL is linear with the number of kernels, with a constant of 0.15 s per kernel. EasyMKL has a linear increase in time too, with 0.17 s per kernel. The memory used by SPG-GMKL has a sub-linear growth with the number of kernels and has reached 2 GB with only 400 kernels. Vice versa, the memory used by our algorithm is independent from the number of kernels to combine. This is due to the fact that the optimization in our algorithm consists of a simple KOMD optimization on the sum of kernels. The computation of this sum of matrices can be easily implemented incrementally and only two kernel matrices need to be stored in memory. It follows that, with our method, we can use an *unlimited* number of different kernels with only a small memory requirement (e.g. 47 MB for the *Splice* dataset).

5.6. Stability with noise features

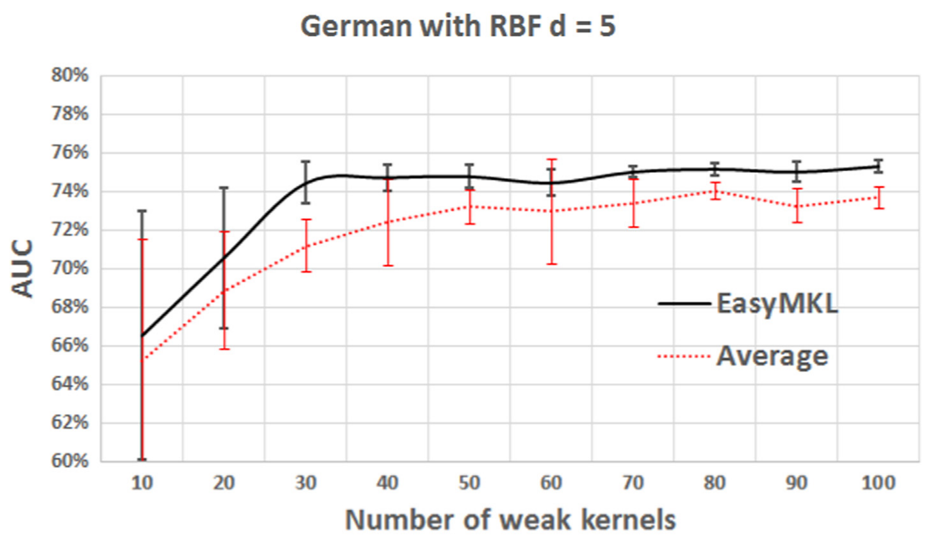
In this set of experiments we injected artificial noise features, that is, features uncorrelated with the label of the example, to study the robustness of EasyMKL with respect to the Average Kernel method.

Similar to [17], given a dataset of L examples, each new noise feature h is created using three simple steps:

1. Pick a feature f randomly from the set of the original features.
2. Let $\mathcal{F} = \{x_i^{(f)} : i = 1, \ldots, L\}$ be the list of values of f (with replica).
3. For each example x_i with $i = 1, \ldots, L$, define the value of $x_i^{(h)}$ as a randomly picked value from $\mathcal{F}$.

Using this method we have created features with a distribution that is similar to the original features but uncorrelated with the labels. We have generated noise features in different percentages of the original features. With these new datasets we have repeated the same experiments described in Section 5.4 with the RBF weak kernels created using d equals to 5 and 10 and three datasets: *Splice*, *Batch2*, and *Mush*. We have fixed the percentages of noise features $p \in \{0, 50, 100, 150, \ldots, 900, 950, 1000\}$. Finally, we compared EasyMKL against the Average Kernel algorithm.

The results are summarized in Fig. 4. We can note that our algorithm obtains good results even with a 1000% of additional noise features and it outperforms the baseline algorithm. EasyMKL increases the gap from the baseline monotonically with respect to the increase of the percentage of noise features. However, quite



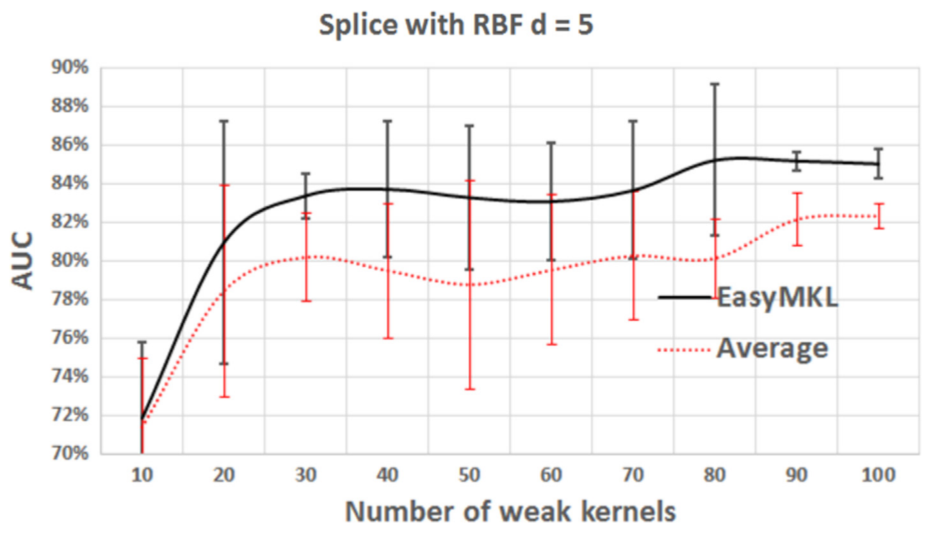


Fig. 6. AUC comparisons with standard deviation of EasyMKL against the average kernel algorithm with respect to different number of weak kernels using the *German* dataset (24 features) and the *Splice* dataset (60 features) with $d = 5$.

surprisingly, these results confirm that the Average Kernel algorithm is still a very strong baseline.

5.7. *Performance with respect to different cardinalities of the training set*

In this section we have analyzed how the size of the training set influences the performance of EasyMKL. We compared our results against the Average Kernel method, that is used as baseline. We performed these experiments over the *Gisette* dataset using the same experimental setting as in Section 5.4 but with different sizes of the training set. Specifically, we have selected the size of the training set varying from 10 to 200 examples. For each size, Fig. 5 has been obtained by evaluating the average on a large number of runs.

From these results, it is clear that our algorithm outperforms the Average Kernel algorithm when the training set is sufficiently large. This is due to the fact that EasyMKL effectively uses the training set to learn which features are more important (see experiments in Section 5.6) and, when data are abundant, it can exploit this information to outperform the baseline. Not surprisingly, when the size of the training set is small the two algorithms perform similarly (and the standard deviation is too big to claim any result).

5.8. *Performance changing the number of weak kernels*

We also performed experiments analyzing the performance of EasyMKL against the Average Kernel method using different numbers of weak kernels. The kernels have been generated using the same setting described in Section 5.4. The number of generated kernels R was selected ranging from $R = 10$ to $R = 200$ (with a step of 10 kernels). We performed the experiments fixing $d = 5$ (i.e. very weak kernels) and using two datasets: *Splice* and *German*.

The experiments have been repeated several times for each value of R and the results are reported in Fig. 6. We can observe that EasyMKL outperforms the Average Kernel method and these experiments confirm that our algorithm also works well when combining a small number of weak kernels. In particular, EasyMKL obtains a steeper gain in performance with the first weak kernels (i.e. from 10 to 30 weak kernels).

6. Conclusion

In this paper we have proposed the EasyMKL algorithm which is able to cope efficiently with a very large number of different kernels. The experiments we reported have shown that the proposed method is more accurate than MKL state-of-the-art methods. We have also discussed time and memory requirements with respect to the number of combined kernels showing that EasyMKL uses only a constant amount of memory and it has only a linear time complexity. Finally, EasyMKL seems also quite robust with respect to the noise introduced by features which are not informative and works well even if used with a small number of weak kernels.

Appendix A. Greedy Algorithm

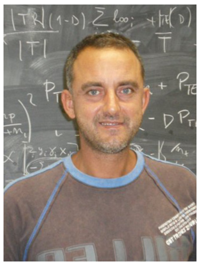
In this appendix we present in detail the greedy algorithm (Algorithm 1) used in Section 5.2.

Algorithm 1. A greedy MKL algorithm which aims at greedily maximizing the separation of positive and negative examples in the training set.

```
Require: $\lambda \in [0,1]$
Ensure: A kernel matrix $\mathbf{K}^*$ and a vector $\boldsymbol{\gamma}^*$
  $\mathbf{K}^* = \mathbf{0},\ c = 0,\ d^* = 0$
  for $r = 1$ to $r_{max}$ do
    pick a new $\mathbf{K}_r$ such that $\mathrm{tr}(\mathbf{K}_r) = k$
    set $\mathbf{K} = \frac{1}{c+1}(\mathbf{K}^* + \mathbf{K}_r)$
    set $\boldsymbol{\gamma}^* = \mathrm{KOMD}(\hat{\mathbf{K}}, \lambda)$
    compute current distance $d = \boldsymbol{\gamma}^{*\top} \hat{\mathbf{Y}}\hat{\mathbf{K}}\hat{\mathbf{Y}}\boldsymbol{\gamma}^*$
    if $d > d^*$ then
      $d^* = d,\ \mathbf{K}^* = \mathbf{K}^* + \mathbf{K}_r,\ c = c + 1$
    end if
  end for
  $\mathbf{K}^* = \frac{1}{c}\mathbf{K}^*$, now $\mathrm{tr}(\mathbf{K}^*) = k$
  return $\mathbf{K}^*$, $\boldsymbol{\gamma}^* = \mathrm{KOMD}(\hat{\mathbf{K}}^*, \lambda)$
```

References

[1] Semi-Supervised Learning, in: O. Chapelle, B. Schölkopf, A. Zien (Eds.), MIT Press, Cambridge, MA, 2006, URL ⟨http://www.kyb.tuebingen.mpg.de/ssl-book⟩.
[2] X. Zhu, A.B. Goldberg, Introduction to Semi-Supervised Learning, Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool Publishers, 2009.
[3] G.R.G. Lanckriet, N. Cristianini, P.L. Bartlett, L.E. Ghaoui, M.I. Jordan, Learning the kernel matrix with semidefinite programming, J. Mach. Learn. Res. 5 (2004) 27–72.
[4] M. Gönen, E. Alpaydin, Multiple kernel learning algorithms, J. Mach. Learn. Res. 12 (2011) 2211–2268.
[5] C. Cortes, M. Mohri, 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, 2010, pp. 247–254. URL ⟨http://www.icml2010.org/papers/179.pdf⟩.
[6] Z. Hussain, 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, 2011, pp. 370–377. URL ⟨http://www.jmlr.org/proceedings/papers/v15/hussain11a/hussain11a.pdf⟩.
[7] F. Aiolli, G.D.S. Martino, A. Sperduti, A kernel method for the optimization of the margin distribution, in: ICANN (1), 2008, pp. 305–314.
[8] K. Bache, M. lichman, Uci machine learning repository (2013). URL ⟨http://archive.ics.uci.edu/ml⟩.
[9] A. Vergara, S. Vembu, T. Ayhan, M.A. Ryan, M.L. Homer, R. Huerta, Chemical gas sensor drift compensation using classifier ensembles, 2012.
[10] I. Guyon, S. Gunn, A. Ben-Hur, G. Dror, Result analysis of the nips 2003 feature selection challenge, in: L.K. Saul, Y. Weiss, L. Bottou (Eds.), Advances in NIPS 17, MIT Press, Cambridge, MA, 2005, pp. 545–552.
[11] X. Xu, I.W. Tsang, D. Xu, Soft margin multiple kernel learning, IEEE Trans. Neural Netw. Learn. Syst. 24 (5) (2013) 749–761.
[12] A. Rakotomamonjy, F.R. Bach, S. Canu, Y. Grandvalet, Simplemkl, J. Mach. Learn. Res. 9 (2008) 2491–2521.
[13] A.P. Danyluk, L. Bottou, M.L. Littman (Eds.), Proceedings of ICML 2009, Montreal, Quebec, Canada, June 14–18, 2009, vol. 382 of ACM International Conference Proceeding Series, ACM, 2009.
[14] M. Kloft, U. Brefeld, S. Sonnenburg, A. Zien, Non-sparse regularization and efficient training with multiple kernels, CoRR abs/1003.0079.
[15] Z. Xu, R. Jin, H. Yang, I. King, M.R. Lyu, Simple and efficient multiple kernel learning by group lasso, in: ICML, 2010, pp. 1175–1182.
[16] A. Jain, S.V.N. Vishwanathan, 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, 2012.
[17] I. Guyon, A.B. Hur, S. Gunn, G. Dror, Result analysis of the nips 2003 feature selection challenge, in: Advances in Neural Information Processing Systems 17, MIT Press, Vancouver, British Columbia, Canada, 2004, pp. 545–552.



F. Aiolli received a Master's Degree and a PhD in Computer Science both from the University of Pisa. He was Post-doc at the University of Pisa, Paid Visiting Scholar at the University of Illinois at Urbana-Champaign (IL), USA, and Post-doc at the University of Padova. He is currently Assistant Professor at the University of Padova. His research activity is mainly in the area of Machine Learning and Information Retrieval.

M. Donini received his Bachelor's degree and Master's degrees in Mathematics from the University of Padova in 2010 and 2012, respectively. He is currently a PhD student in Machine Learning at the University of Padova. His research interests include kernel methods, multiple kernel learning and machine learning in general.