Machine Learning manuscript No.
(will be inserted by the editor)

Alignment Based Kernel Learning with a Continuous Set of Base Kernels

Arash Afkanpour · Csaba Szepesvári · Michael Bowling

The date of receipt and acceptance will be inserted by the editor

Abstract The success of kernel-based learning methods depends on the choice of kernel. Recently, kernel learning methods have been proposed that use data to select the most appropriate kernel, usually by combining a set of base kernels. We introduce a new algorithm for kernel learning that combines a continuous set of base kernels, without the common step of discretizing the space of base kernels. We demonstrate that our new method achieves state-of-the-art performance across a variety of real-world datasets. Furthermore, we explicitly demonstrate the importance of combining the right dictionary of kernels, which is problematic for methods that combine a finite set of base kernels chosen *a priori*. Our method is not the first approach to work with continuously parameterized kernels. We adopt a two-stage kernel learning approach. We also show that our method requires substantially less computation than previous such approaches, and so is more amenable to multi-dimensional parameterizations of base kernels, which we demonstrate.

Arash Afkanpour
Department of Computing Science
University of Alberta
Tel.: +1 (780) 934-9872
Fax: +1 (780) 492-6393
E-mail: afkanpou@ualberta.ca

Csaba Szepesvári
Department of Computing Science
University of Alberta
Tel.: +1 (780) 492-8581
Fax: +1 (780) 492-6393
E-mail: szepesva@ualberta.ca

Michael Bowling
Department of Computing Science
University of Alberta
Tel.: +1 (780) 492-1766
Fax: +1 (780) 492-1071
E-mail: mbowling@ualberta.ca

Keywords Two-stage kernel learning · Continuous kernel sets

1 Introduction

A well known fact in machine learning is that the choice of features heavily influences the performance of learning methods. Similarly, the performance of a learning method that uses a kernel function is highly dependent on the choice of kernel function. The idea of *kernel learning* is to use data to select the most appropriate kernel function for the learning task.

In this paper, we consider kernel learning in the context of supervised learning. In particular, we consider the problem of learning positive-coefficient linear combinations of base kernels, where the base kernels belong to a parameterized family of kernels, $(\kappa_\sigma)_{\sigma\in\Sigma}$. Here, Σ is a "continuous" parameter space, i.e., some subset of a Euclidean space. A prime example (and extremely popular choice) is when κ_σ is a Gaussian kernel, where σ can be a single common bandwidth or a vector of bandwidths, one per coordinate. One approach then is to discretize the parameter space Σ and then find an appropriate non-negative linear combination of the resulting set of base kernels, $\mathcal{N} = \{\kappa_{\sigma_1}, \ldots, \kappa_{\sigma_p}\}$. The advantage of this approach is that once the set $\mathcal{N}$ is fixed, any of the many efficient methods available in the literature can be used to find the coefficients for combining the base kernels in $\mathcal{N}$ (see the papers by Lanckriet et al 2004; Sonnenburg et al 2006; Rakotomamonjy et al 2008; Cortes et al 2009a; Kloft et al 2011 and references therein). One potential drawback of this approach is that it requires an appropriate, *a priori* choice of $\mathcal{N}$. This might be problematic, e.g., if Σ is contained in a Euclidean space of moderate, or large dimension (say, a dimension over 20) since the number of base kernels, p, grows exponentially with dimensionality even for moderate discretization accuracies. Furthermore, independent of the dimensionality of the parameter space, the need to choose the set $\mathcal{N}$ independently of the data is at best inconvenient and selecting an appropriate resolution might be far from trivial. In this paper we explore an alternative method which avoids the need for discretizing the space Σ.

We are not the first to realize that discretizing a continuous parameter space might be troublesome: The method of Argyriou et al (2005) proposes to combine continuously parameterized kernels. They present a greedy coordinate descent-type approach to kernel learning that handles sets with infinitely many kernels. Empirically, they found this approach to have excellent and robust performance, showing the potential of the idea of avoiding discretizations.

Our new method is similar to that of Argyriou et al (2005). In fact, both methods are instances of the so-called forward-stagewise additive modeling (FSAM) procedure. However, as opposed to the method of Argyriou et al (2005), our method belongs to the group of *two-stage kernel learning* methods. The decision to use a two-stage kernel learning approach was motivated by the recent success of the two-stage method of Cortes et al (2010). In fact, our kernel learning method uses the centered kernel alignment metric of Cortes

et al (2010) (derived from the uncentered alignment metric of Cristianini et al (2002)) in its first stage as the objective function of the FSAM procedure, while in the second stage a standard kernel-based supervised learning technique is used.

In the response to the reviewers we should probably mention that we have reorganized the discussion of the relation between our method and that of Argyriou et al (2005, 2006). Arash, for your thesis, please add a statement with a proof that shows that the method of these guys is also an instance of FSAM.

The technical difficulty of implementing FSAM is that one needs to compute the functional gradient of the chosen objective function. We show that in our case this problem is equivalent to solving an optimization problem over $\sigma \in \Sigma$ with an objective function that is a linear function of the Gram matrix derived from the kernel κ_σ. Because of the nonlinear dependence of this matrix on σ, this is the step where we need to resort to local optimization: this optimization problem is in general non-convex. However, as we shall demonstrate empirically, even if we use local solvers to solve this optimization step, the algorithm still shows an overall excellent performance as compared to other state-of-the-art methods. This is not completely unexpected: One of the key ideas underlying boosting (a close relative of FSAM) is that it is designed to be robust even when the individual "greedy" steps are imperfect (cf., Chapter 12, Bühlmann and van de Geer 2011). Given the new kernel to be added to the existing dictionary, we give a computationally efficient, closed-form expression that can be used to determine the coefficient on the new kernel to be added to the previous kernels.

does this property of boosting carry over to FSAM? a more appropriate reference would be good.

The empirical performance of our proposed method is explored in a series of experiments. Our experiments serve multiple purposes. Firstly, we explore the potential advantages, as well as limitations of the proposed technique. In particular, we demonstrate that the procedure is indeed reliable (despite the potential difficulty of implementing the greedy step) and that it can be successfully used even when Σ is a subset of a multi-dimensional space. Secondly, we demonstrate that in some cases, kernel learning can have a very large improvement over simpler alternatives, such as combining some fixed dictionary of kernels with uniform weights. Whether this is true is an important issue that is given weight by the fact that just recently it became a subject of dispute (Cortes, 2009). Finally, we compare the performance of our method, both from the perspective of its generalization capability and computational cost, to its natural, state-of-the-art alternatives, such as the two-stage method of Cortes et al (2010) and the algorithm of Argyriou et al (2005). For this, we compared our method on datasets used in previous kernel learning work. To give further weight to our results, we compare on more datasets than any of the previous papers that proposed new kernel learning methods.

Our experiments demonstrate that our new method is competitive in terms of its generalization performance, while its computational cost is significantly less than that of its competitors that enjoy similarly good generalization per-

formance as our method. In addition, our experiments also revealed an interesting novel insight into the behavior of two-stage methods: we noticed that two-stage methods can "overfit" the performance metric of the first stage. In some problem we observed that our method could find kernels that gave rise to better (test-set) performance on the first-stage metric, while the method's overall performance degrades when compared to using kernel combinations whose performance on the first metric is worse. The explanation of this is that metric of the first stage is a surrogate performance measure and thus just like in the case of choosing a surrogate loss in classification, better performance according to this surrogate metric does not necessarily transfer into better performance in the primary metric as there is no monotonicity relation between these two metrics. We also show that with proper capacity control, the problem of overfitting the surrogate metric can be overcome. Finally, our experiments show a clear advantage to using kernel learning methods as opposed to combining kernels with a uniform weight, although it seems that the advantage mainly comes from the ability of our method to discover the right set of kernels. This conclusion is strengthened by the fact that the closest competitor to our method was found to be the method of Argyriou et al (2005) that also searches the continuous parameter space, avoiding discretizations. Our conclusion is that it seems that the choice of the base dictionary is more important than how the dictionary elements are combined and that the *a priori* choice of this dictionary may not be trivial. This is certainly true already when the number of parameters is moderate. Moreover, when the number of parameters is larger, simple discretization methods are infeasible, whereas our method can still produce meaningful dictionaries.

2 The New Method

The purpose of this section is to describe our new method. Let us start with the introduction of the problem setting and the notation. We consider binary classification problems, where the data $\mathcal{D} = ((X_1, Y_1), \dots, (X_n, Y_n))$ is a sequence of independent, identically distributed random variables, with $(X_i, Y_i) \in \mathbb{R}^d \times \{-1, +1\}$. For convenience, we introduce two other pairs of random variables (X, Y), (X', Y'), which are also independent of each other and they share the same distribution with (X_i, Y_i). The goal of classifier learning is to find a predictor, $g : \mathbb{R}^d \to \{-1, +1\}$ such that the predictor's risk, $L(g) = \mathbb{P}(g(X) \neq Y)$, is close to the Bayes-risk, $\inf_g L(g)$. We will consider a two-stage method, as noted in the introduction. The first stage of our method will pick some kernel $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ from some set of kernels $\mathcal{K}$ based on $\mathcal{D}$, which is then used in the second stage, using the same data $\mathcal{D}$ to find a good predictor.[1]

[1] One could consider splitting the data, but we see no advantage to doing so. Also, the methods for the second stage are not a focus of this work and the particular methods used in the experiments are described later.

Consider a parametric family of base kernels, $(\kappa_\sigma)_{\sigma\in\Sigma}$, in which Σ could have infinitely many members, such as an interval of real numbers. The kernels considered by our method belong to the set

$$\mathcal{K} = \left\{ \sum_{i=1}^{r} \mu_i \kappa_{\sigma_i} : r \in \mathbb{N}, \mu_i \geq 0, \sigma_i \in \Sigma, i = 1, \ldots, r \right\},$$

i.e., we allow non-negative linear combinations of a finite number of base kernels. For example, the base kernel could be a Gaussian kernel, where $\sigma > 0$ is its bandwidth:

$$\kappa_\sigma(x, x') = \exp\left(-\frac{\|x - x'\|^2}{\sigma^2}\right),$$

where $x, x' \in \mathbb{R}^d$. However, one could also have a separate bandwidth for each coordinate.

The "ideal" kernel underlying the common distribution of the data is

$$k^*(x, x') = \mathbb{E}\left[\, YY' \,|\, X = x, X' = x' \,\right].$$

Our new method attempts to find a kernel $k \in \mathcal{K}$ which is maximally aligned to this ideal kernel, where, following Cortes et al (2010), the alignment between two kernels $k, \tilde{k}$ is measured by the *centered alignment metric*,[2]

$$A_c(k, \tilde{k}) \stackrel{\text{def}}{=} \frac{\langle k_c, \tilde{k}_c \rangle}{\|k_c\| \|\tilde{k}_c\|},$$

where k_c is the kernel underlying k centered in the feature space (similarly for $\tilde{k}_c$), $\langle k, \tilde{k} \rangle = \mathbb{E}\left[k(X, X')\tilde{k}(X, X')\right]$ and $\|k\|^2 = \langle k, k \rangle$. A kernel k centered in the feature space, by definition, is the unique kernel k_c, such that for any x, x',

$$k_c(x, x') = \langle \Phi(x) - \mathbb{E}\left[\Phi(X)\right], \Phi(x') - \mathbb{E}\left[\Phi(X)\right] \rangle,$$

where Φ is a feature map underlying k. By considering centered kernels k_c, $\tilde{k}_c$ in the alignment metric, one implicitly matches the mean responses $\mathbb{E}[k(X, X')]$, $\mathbb{E}[\tilde{k}(X, X')]$ before considering the alignment between the kernels (thus, centering depends on the distribution of X). An alternative way of stating this is that centering cancels mismatches of the mean responses between the two kernels. When one of the kernels is the ideal kernel, centered alignment effectively standardizes the alignment by cancelling the effect of imbalanced class distributions. For further discussion of the virtues of centered alignment, see Cortes et al (2010).

Since the common distribution underlying the data is unknown, one resorts to empirical approximations to alignment and centering, resulting in the empirical alignment metric,

$$A_c(K, \tilde{K}) = \frac{\langle K_c, \tilde{K}_c \rangle_F}{\|K_c\|_F \|\tilde{K}_c\|_F},$$

[2] Note that the word metric is used in its everyday sense and not in its mathematical sense.

where,

$$K = (k(X_i, X_j))_{1 \le i,j \le n}, \quad \text{and}$$
$$\tilde{K} = (\tilde{k}(X_i, X_j))_{1 \le i,j \le n}$$

are the kernel matrices underlying k and $\tilde{k}$ respectively, and for a kernel matrix, K,

$$K_c = C_n K C_n,$$

where C_n is the so-called centering matrix defined by $C_n = I_{n\times n} - \mathbf{1}\mathbf{1}^\top/n$, $I_{n\times n}$ being the $n \times n$ identity matrix and $\mathbf{1} = (1, \ldots, 1)^\top \in \mathbb{R}^n$. The empirical counterpart of maximizing $A_c(k, k^*)$ is to maximize $A_c(K, \hat{K}^*)$, where $\hat{K}^* \stackrel{\text{def}}{=} \mathbf{Y}\mathbf{Y}^T$, and $\mathbf{Y} = (Y_1, \ldots, Y_n)^\top$ collects the responses into an n-dimensional vector. Here, K is the kernel matrix derived from a kernel $k \in \mathcal{K}$. To make this connection clear, we will write $K = K(k)$.

Define $f : \mathcal{K} \to \mathbb{R}$ by $f(k) = A_c(K(k), \hat{K}^*)$. Our goal is to find an approximate maximizer of f over $\mathcal{K}$. Before presenting our algorithm let us note that for the case when the set Σ is finite, Cortes et al (2010) show that maximizing f over the convex combination of the base kernels $(\kappa_\sigma)_{\sigma\in\Sigma}$ can be formulated as a quadratic optimization problem with $|\Sigma|$ variables. When Σ is infinite, the optimization problem would have infinitely many variables, hence one needs to follow a different approach in this case.

To find an approximate maximizer of f, we propose a steepest ascent approach to *forward stagewise additive modeling* (FSAM). FSAM (Hastie et al, 2001) is an iterative method for optimizing an objective function by sequentially adding new basis functions without changing the parameters and coefficients of the previously added basis functions. In the steepest ascent approach, in iteration t, we search for the base kernel in (κ_σ) defining the direction in which the growth rate of f is the largest, locally in a small neighborhood of the previous candidate k^{t-1}:

$$\sigma_t^* = \arg\max_{\sigma\in\Sigma} \lim_{\varepsilon\to 0} \frac{f(k^{t-1} + \varepsilon\,\kappa_\sigma) - f(k^{t-1})}{\varepsilon}. \tag{1}$$

Once σ_t^* is found, the algorithm finds the coefficient $0 \le \eta_t \le \eta_{\max}$[3] such that $f(k^{t-1} + \eta_t \kappa_{\sigma_t^*})$ is maximized and the candidate is updated using

$$k^t = k^{t-1} + \eta_t \kappa_{\sigma_t^*}.$$

The process stops when the objective function f ceases to increase by an amount larger than $\theta > 0$, or when the number of iterations becomes larger then a predetermined limit T, whichever happens earlier.

[3] In the experiments, we use the arbitrary value $\eta_{\max} = 1$. Note that the value of $\eta_{\max}$, together with the limit T acts as a regularizer. However, in our experiments, the procedure always stops before the limit T on the number of iterations is reached.

Algorithm 1 Forward stagewise additive modeling for kernel learning with a continuously parametrized set of kernels. For the definitions of f, F, F' and $K : \mathcal{K} \to \mathbb{R}^{n\times n}$, see the text.

Inputs: data $\mathcal{D}$, kernel initialization parameter ε, the number of iterations T, tolerance θ, maximum stepsize $\eta_{\max} > 0$.
$K^0 \leftarrow \varepsilon I_n$.
for $t = 1$ **to** T **do**
  $P \leftarrow F'(K^{t-1})$
  $P \leftarrow C_n\, P\, C_n$
  $\sigma^* = \arg\max_{\sigma\in\Sigma} \langle P, K(\kappa_\sigma)\rangle_F$
  $K' = C_n\, K(\kappa_{\sigma^*})\, C_n$
  $\eta^* = \arg\max_{0\le\eta\le\eta_{\max}} F(K^{t-1} + \eta K')$
  $K^t \leftarrow K^{t-1} + \eta^* K'$
  if $F(K^t) \le F(K^{t-1}) + \theta$ **then** terminate
end for

Proposition 1 *The value of σ_t^* can be obtained by*

$$\sigma_t^* = \arg\max_{\sigma\in\Sigma} \left\langle K(\kappa_\sigma), F'\left((K(k^{t-1}))_c\right)\right\rangle_F , \tag{2}$$

where for a kernel matrix K,

$$F'(K) = \frac{\hat{K}_c^* - \|K\|_F^{-2}\langle K, \hat{K}_c^*\rangle_F\, K}{\|K\|_F \|\hat{K}_c^*\|_F} . \tag{3}$$

The proof can be found in Section A.1. The crux of the proposition is that the directional derivative in (1) can be calculated and gives the expression maximized in (2).

In general, the optimization problem (2) is not convex and the cost of obtaining a (good approximate) solution is hard to predict. Evidence that, at least in some cases, the function to be optimized is not ill-behaved is presented in Section B.1. In our experiments, an approximate solution to (2) is found using numerical methods.[4] As is usual in forward stagewise additive modeling, finding the global optimizer in (2) might not be necessary for achieving good statistical performance. Our experiments seem to confirm that this is the case.

In multi-dimensional settings, i.e., when Σ is a subset of a multi-dimensional space, we have experimentally noticed that our method can "overfit" the alignment (the evidence for this is presented later in Section 3). To prevent this, we propose that instead of (2), the regularized version of this optimization problem should be used:

$$\sigma_t^* = \arg\min_{\sigma\in\Sigma} \; -\left\langle K(\kappa_\sigma), F'\left((K(k^{t-1}))_c\right)\right\rangle_F + \lambda\, \mathrm{Reg}(\sigma). \tag{4}$$

Argyriou et al. (2006) seem to have some results that show that only finitely many kernels are needed in the optimal solution. Will these results extend to our case? At least include this in your thesis (and

[4] In particular, we use the fmincon function of Matlab, with the interior-point algorithm option. We also tried the DC-programming method of Argyriou et al (2006). The computational cost of DC-programming increases exponentially fast in the number of parameters(Argyriou et al, 2006), which renders this method inefficient for large number of parameters. Even for the one-dimensional parameter search, the DC-programming method does not uniformly perform faster than the alternative method.

Here, $\lambda > 0$ is a regularization parameter and Reg is a regularization functional that assigns higher numbers to more "complex" kernel parameters. In the experiments, we will use

$$\mathrm{Reg}(\sigma) = \|\sigma - \overline{\sigma}\|_2^2,$$

where $\overline{\sigma} = 1/d \sum_{i=1}^{d} \sigma_i$. This regularizer penalizes kernel parameter vectors with a large "variance". In particular, if $\lambda \to \infty$ then the search space is effectively reduced to a one-dimensional space.

Let us now turn to how η_t can be found. As it turns out, this parameter is easy to find. In particular, the underlying optimization problem has a closed form solution:

Proposition 2 *The value of* η_t *is given by* $\eta_t = \arg\max_{\eta \in \{0, \eta^*, \eta_{\max}\}} f(k^{t-1} + \eta \kappa_{\sigma_t^*})$, *where* $\eta^* = \max(0, (ad - bc)/(bd - ae))$ *if* $bd - ae \neq 0$ *and* $\eta^* = 0$ *otherwise,* $a = \langle K, \hat{K}_c^* \rangle_F$, $b = \langle K', \hat{K}_c^* \rangle_F$, $c = \langle K, K \rangle_F$, $d = \langle K, K' \rangle_F$, $e = \langle K', K' \rangle_F$ *and* $K = (K(k^{t-1}))_c$, $K' = (K(\kappa_{\sigma_t^*}))_c$.

The pseudocode of the full algorithm is presented in Algorithm 1. The algorithm needs the data, the number of iterations (T) and a tolerance (θ) parameter, in addition to a parameter ε used in the initialization phase and $\eta_{\max}$. The parameter ε is used in the initialization step to avoid division by zero, and its value has little effect on the performance. Note that the cost of computing a kernel-matrix, or the inner product of two such matrices is $O(n^2)$. Therefore, the complexity of the algorithm is quadratic in the number of data points. The actual cost will be strongly influenced by the computational cost of problem (2). In the description of the experiments, we include actual training times that should give a rough indication of the computational limits of the procedure.

As a final remark, note that it is straightforward to apply our algorithm to combine heterogeneous kernels too. In this case, to find the best kernel in each iteration, we first find the best kernel (parameter) for each kernel type according the objective function in line 6 of Algorithm 1. Once we determine the best kernel (parameter) for each kernel type, we iterate through these best kernels and find the best kernel, again according to the same objective function. In the next section, we evaluate our method in different problems and compare it against other algorithms.

3 Experimental Evaluation

In this section we compare our kernel learning method with several kernel learning methods on synthetic and real data; see Table 1 for the list of methods. Our method is labeled CA for Continuous Alignment-based kernel learning. In all of the experiments, we use the following values with CA: $T = 50$, $\varepsilon = 10^{-10}$, and $\theta = 10^{-3}$. The first two methods, i.e., our algorithm, and CR (Argyriou et al, 2005), are able to pick kernel parameters from a continuous set, while

Table 1: List of the kernel learning methods evaluated in the experiments. The key to the naming of the methods is as follows: CA stands for "continuous alignment" maximization, CR stands for "continuous risk" minimization, DA stands for "discrete alignment", D1, D2, DU should be obvious.

Abbr.	**Method**
CA	Our new method
CR	From Argyriou et al (2005)
DA	From Cortes et al (2010)
D1	ℓ^1-norm MKL (Kloft et al, 2011)
D2	ℓ^2-norm MKL (Kloft et al, 2011)
DU	Uniform weights over kernels

the rest of the algorithms work with a finite number of base kernels. In all of the experiments in this paper, for all methods the classifiers were trained using the soft margin SVM method, where the regularization coefficient of SVM was chosen from $10^{\{-5,-4.5,\ldots,4.5,5\}}$ using an independent validation set.

In Section 3.1 we use synthetic data to illustrate the potential advantage of methods that work with a continuously parameterized set of kernels and the importance of combining multiple kernels. We also illustrate in a toy example that multi-dimensional kernel parameter search can improve performance. These are followed by the evaluation of the above listed methods on several real datasets in Section 3.2.

3.1 Synthetic Data

The purpose of these experiments is mainly to provide empirical proof for the following hypotheses: (H1) The combination of multiple kernels can lead to improved performance as compared to what can be achieved with a single kernel. (H2) The methods that search the continuously parameterized families are able to find the "key" kernels and their combination. (H3) Our method can even search multi-dimensional parameter spaces, which in some cases is crucial for good performance.

To illustrate (H1) and (H2) we have designed the following problem: the inputs are generated from the uniform distribution over the interval $[-10, 10]$. The label of each data point is determined by the function $y(x) = \mathrm{sign}(f(x))$, where $f(x) = \sin(\sqrt{2}x) + \sin(\sqrt{12}x) + \sin(\sqrt{60}x)$. Training and validation sets include 500 data points each, while the test set includes 1000 instances. Figure 2(a) shows the functions f (blue curve) and y (red dots). For this experiment we use Dirichlet kernels of degree one, parameterized with a frequency parameter σ: $\kappa_\sigma(x, x') = 1 + 2\cos(\sigma\|x - x'\|)$.

In order to investigate (H1), we searched through the kernel frequencies in the range $[0, 20]$. We selected 1000 kernel frequencies by discretizing this range. We then train a SVM with each kernel. The training and validation

sets consist of 1000 data points each, while the test set consists of 2000 points. We plot the test error obtained by each kernel in Figure 1. Notice that the kernel frequencies used to generate data, i.e. $\{\sqrt{2}, \sqrt{12}, \sqrt{60}\}$ obtain lowest error values. Hence, they are good choices.

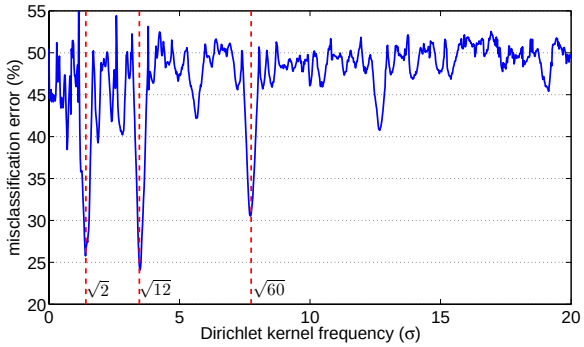


Fig. 1: Misclassification error as a function of kernel frequency.

Next, we trained SVM classifiers with pair of frequencies, i.e. $\{\sqrt{2}, \sqrt{12}\}$, $\{\sqrt{2}, \sqrt{60}\}$, and $\{\sqrt{12}, \sqrt{60}\}$. They achieved error rates of 16.4%, 20.0%, and 21.3%, respectively (the kernels were combined using uniform weights). Finally, a classifier that was trained with all three frequencies achieved an error rate of 2.3%.

Let us now turn to (H2). As shown in Figure 2(b), the CA and CR methods both achieved a misclassification error close to what was seen when the three best frequencies were used, showing that they are indeed effective. Furthermore, Figure 2(c) shows that the discovered frequencies are close to the frequencies used to generate the data. For the sake of illustration, we also tested the methods which require the discretization of the parameter space. We choose 10 Dirichlet kernels with $\sigma \in \{0, 1, \ldots, 9\}$, covering the range of frequencies defining f. As can be seen from Figure 2(b) in this example the chosen discretization accuracy is insufficient. Although it would be easy to increase the discretization accuracy to improve the results of these methods,[5] the point is that if a high resolution is needed in a one-dimensional problem, then these methods are likely to face serious difficulties in problems when the space of kernels is more complex (e.g., the parameterization is multi-dimensional). Nevertheless, we are not suggesting that the methods which require discretization are universally inferior, but merely pointing out that an "appropriate discrete kernel set" might not always be available.

To illustrate (H3) we designed a second set of problems: The instances for the positive (negative) class are generated from a $d = 50$-dimensional

[5] Further experimentation found that a discretization below 0.1 is necessary in this example.

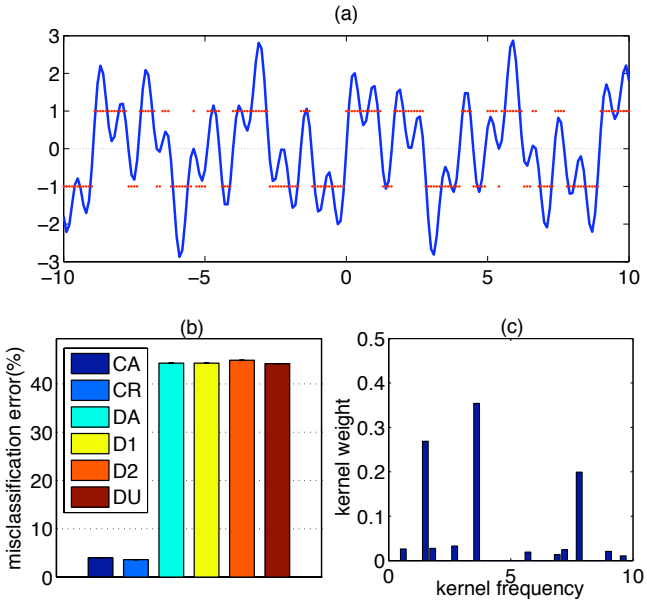


Fig. 2: (a): The function $f(x) = \sin(\sqrt{2}x) + \sin(\sqrt{12}x) + \sin(\sqrt{60}x)$ used for generating synthetic data, along with $\text{sign}(f)$. (b): Misclassification percentages obtained by each algorithm. (c): The kernel frequencies found by the CA method.

Gaussian distribution with covariance matrix $C = I_{d\times d}$ and mean $\mu_1 = \rho\frac{\theta}{\|\theta\|}$ (respectively, $\mu_2 = -\mu_1$ for the negative class). Here $\rho = 1.75$. The vector $\theta \in [0,1]^d$ determines the relevance of each feature in the classification task, e.g. $\theta_i = 0$ implies that the distributions of the two classes have zero means in the i-th feature, which renders this feature irrelevant. The value of each component of vector θ is calculated as $\theta_i = (i/d)^\gamma$, where γ is a constant that determines the relative importance of the elements of θ. We generate seven datasets with $\gamma \in \{0, 1, 2, 5, 10, 20, 40\}$. For each value of γ, the training set consists of 50 data points (the prior distribution for the two classes is uniform). The test error values are measured on a test set with 2000 instances. We repeated each experiment 10 times and report the average misclassification error and alignment measured over the test set along with the training time.

In this experiment we test two versions of our method: one that uses a family of Gaussian kernels with a common bandwidth (denoted by CA-1D), and another one (denoted by CA-nD) that searches in the space $(\kappa_\sigma)_{\sigma\in(0,\infty)^{50}}$, where each coordinate has a separate bandwidth parameter,

$$\kappa_\sigma(x, x') = \exp\left(-\sum_{i=1}^{d}\frac{(x_i - x'_i)^2}{\sigma_i^2}\right).$$

Since the training set is small,[6] one can easily overfit while optimizing the alignment. Hence, we will use the regularized version of the algorithm (cf. Equation 4) with $\text{Reg}(\sigma) = \|\sigma - \overline{\sigma}\|_2^2$, where $\overline{\sigma} = 1/d \sum_{i=1}^{d} \sigma_i$. As discussed beforehand, this regularizer causes the values of the bandwidth parameters to shrink toward their common average value.

For the sake of completeness, we also include the unregularized version of CA-nD in which $\lambda = 0$. It is labeled CA-nD(No Reg) in plots. The method of Argyriou et al (2005) has also been run with single- and multi-dimensional parameter search procedures (CR-1D and CR-nD respectively). We also include results obtained for finite kernel learning methods. For these methods, we generate 50 Gaussian kernels with bandwidths $\sigma \in mg^{\{0,\ldots,49\}}$, where $m = 10^{-3}$, and $g \approx 1.33$. Hence, the bandwidth range constitutes a geometric sequence from 10^{-3} to 10^3. Further details of the experimental setup can be found in Section B.2.

Figure 3 shows the results. Recall that the larger the value of γ, the larger is the number of nearly irrelevant features. Since methods which search only a one-dimensional space cannot differentiate between relevant and irrelevant features, their misclassification rate increases with γ. On the other hand multi-dimensional search methods are able to cope with this situation and even improve the performance. We observe that without regularization, though, for small values of γ, CA-nD(No Reg) and CR-nD drastically overfit. We also show the training time of the methods. Note that CR-1D is slower than CA-1D. [7] The same trend can be observed in their multi-dimensional counterparts, i.e. CA-nD(No Reg) and CR-nD. However the training time of CA-nD is comparable (in this experiment) to that of CR-nD since CA-nD runs cross-validation to tune λ too. Although the large training time of multi-dimensional search methods might be prohibitive, for some problems, these methods might be the only option if good performance is crucial.

3.1.1 Scalability Test

The previous experiment does not answer the question of how the methods running time and performance scale with either the number of training examples or (in the case of finite kernel learning methods) the number of kernels. Therefore, in this section we report results for experiments where these properties of the learning methods are studied. To test the methods we choose the previous synthetic dataset with $\gamma = 40$ and increase the number of training examples and the number of kernels. As with the previous experiment we report misclassification errors and training times.

[6] We keep the size of training set small in this experiment to emphasize the importance of regularization when one deals with high-dimensional kernel parameters.

[7] Obviously, reported training times are implementation-dependent and the comparisons should be taken with a grain of salt. We implemented CA and CR methods using Matlab's fmincon function. Finite kernel learning algorithms can either be found or implemented using the SHOGUN machine learning toolbox available at http://www.shogun-toolbox.org/.

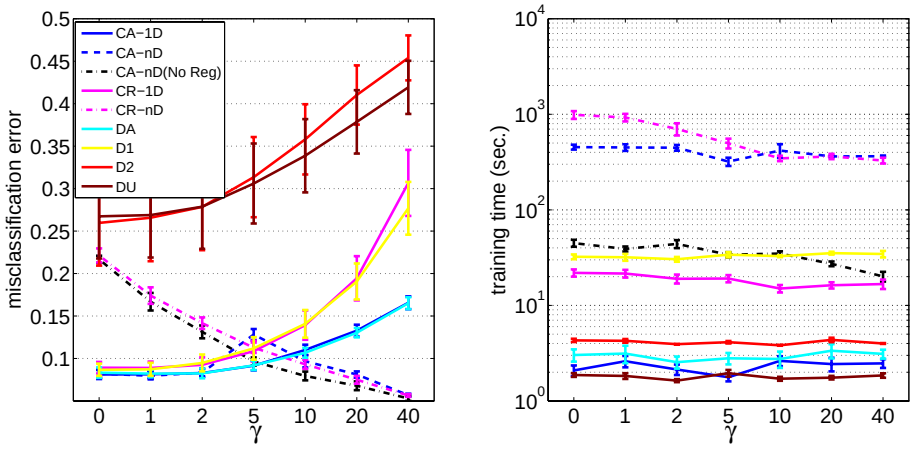


Fig. 3: Misclassification error and training time of various methods in a 50-dimensional synthetic problem as a function of the relevance parameter γ. Note that the number of irrelevant features increases with γ. For details of the experiments, see the text.

In the first experiment we examine the methods when they are provided with different number of training examples between 50 and 1600. Figure 4(top) shows the results. While the performance of the methods becomes similar as we increase the size of training set, for small training sets CA-1D and DA have the lowest misclassification error among methods that consider one-dimensional kernel parameters. In terms of training time, CA-1D is the fastest method overall. Its training time is almost comparable to that of DU, which performs no kernel learning. The misclassification error rate of the multi-dimensional methods is the best which is unsurprising given that at $\gamma = 40$ only a few variables are expected to play a significant role. In terms of their training time, CA-nD(No Reg) and CA-nD perform better than CR-nD. In particular, CA-nD(No Reg) is almost 10 times faster than its counterpart, CR-nD.

In the second experiment, we again use the 50-dimensional synthetic dataset with $\gamma = 40$ and study how the finite kernel learning methods scale as the number of kernels is increased. We provide 50 training examples to all methods to be able to make the differences between the methods easier to demonstrate. Admittedly, with these choices, we create a learning problem that is difficult to handle with the discretization approach. In particular, if one uses only 2 different values for each dimension, we already get $2^{50} \approx 10^{15}$ kernels, which is at present out of reach even for the best finite kernel learning algorithms. What one may still try is to clump all parameters together and thus reduce the dimensionality of the parameter space to one. The difficulty then is that at $\gamma = 40$ many dimensions are irrelevant, which means that even the best isotropic kernel is expected to improve moderately only.

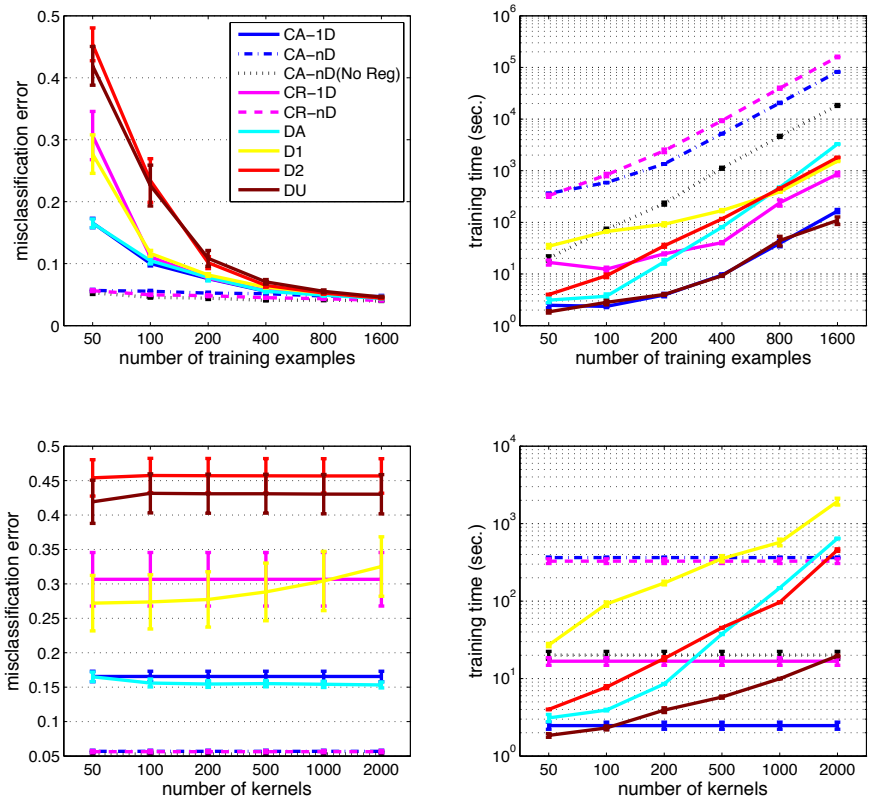


Fig. 4: Scalability of various methods with respect to the number of training examples (top) and the number of kernels (bottom). Results for the methods that search over a continuous range are included for comparison purposes in the bottom figures. Note that the curves of these methods are flat as they in fact do not rely on the provided finite kernel sets.

To test this hypothesis, we choose r Gaussian kernels where r is one of $50, 100, 200, 500, 1000$ or 2000. The parameters of the kernels are chosen from the interval $[10^{-3}, 10^{3}]$ in a geometric fashion. We provide these kernels to the finite kernel learning methods and measure misclassification error and training time. The results are shown in the bottom part of Figure 4. As can be seen, the performance of the finite kernel learning methods does not improve when the number of kernels increases: the kernels added in a blind fashion are not helpful in predicting the response. Further, as expected, their training time increases with the number of kernels. The computational complexity of D1 and D2 depends linearly on the number of kernels. The optimization problem underlying DA is an instance of quadratic programming (Cortes et al, 2010),

which typically is solved in cubic time using interior-point methods (Boyd and Vandenberghe, 2004). The DU method does not perform any kernel learning. Yet, its computation time is increased due to the extra load of computing kernel matrices.

The attentive reader might have noticed that the performance of method D1 (the ℓ^1-norm MKL method of Kloft et al 2011) on Figure 4 gets worse as the number of kernels is increased. Note that the experiment in this case are set up in such a way that the additional kernels are not helpful in predicting the response – the features corresponding to the additional kernels can be considered as "noise". Should not properly executed regularization prevent the "overfitting effect" seen on this figure? To understand why this is not the case notice that when some "noisy" features are added, the learning algorithm will reduce the training error by assigning non-zero weights to these noisy features. However, then the validation error increases. Thus, cross-validation will prefer to choose a larger value of the regularization coefficient. However, a larger regularization coefficient will also shrink the weights assigned to the "important" features. As a result, even with the optimal choice of the regularization coefficient, the predictor is performing worse than when the noise features were not introduced, which is in fact well in line with the theory available for ℓ^1-regularization.[8]

Future work: Would random projection work better?

3.2 Real Data

We evaluate the methods listed in Table 1 on several binary classification tasks from MNIST and the UCI Letter recognition dataset, along with several other datasets from the UCI machine learning repository (Frank and Asuncion, 2010) and Delve datasets (see, http://www.cs.toronto.edu/~delve/data/datasets.html). In all experiments the results are averaged over 10 runs.

MNIST. In the first experiment, following Argyriou et al (2005), we choose 8 handwritten digit recognition tasks of various difficulty from the MNIST dataset (LeCun and Cortes, 2010). This dataset consists of 28×28 images with pixel values ranging between 0 and 255. In these experiments, we used Gaussian kernels with parameter σ: $G_\sigma(x, x') = \exp(-\|x - x'\|^2/\sigma^2)$. Due to the large number of attributes (784) in the MNIST dataset, we do not evaluate multi-dimensional versions of CA and CR (they are evaluated on a similar dataset, of smaller scale, see below). For the algorithms that work with a finite kernel set, we pick 20 kernels with the value of σ picked from an equidistant discretization of the interval $[500, 50000]$. In each experiment, the training and validation sets consist of 500 and 1000 data points, while the test set has 2000 data points. The test-set error plots for all of the problems are shown in Figure 5. In order to give an overall impression of the algorithms' performance,

[8] Under some technical conditions, theory predicts that the performance degradation is proportional to the logarithm of the number of features (e.g., Bühlmann and van de Geer, 2011).

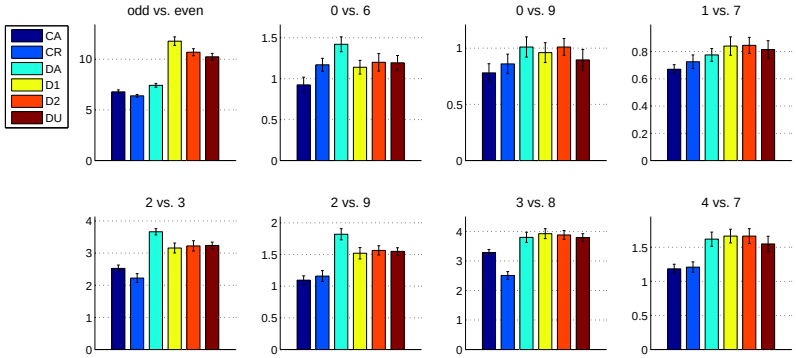


Fig. 5: Misclassification percentages in different tasks of the MNIST dataset.

Table 2: Median rank and training time (seconds) of various kernel learning methods evaluated in experiments.

	Rank			**Time**	
	MNIST	Letter	Misc. data	MNIST	Letter
CA-1D	1	1	3	12 ± 1	9 ± 1
CA-nD	N/A	5.5	2	N/A	770 ± 80
CR-1D	2	2	3	377 ± 56	590 ± 21
CR-nD	N/A	3	5	N/A	1550 ± 200
DA	4.5	4.5	3	31 ± 1	11 ± 1
D1	4.5	8	4	57 ± 6	21 ± 1
D2	5	7	7	58 ± 3	22 ± 1
DU	4	6	7	10 ± 1	5 ± 1

we ranked them based on the results obtained in the above experiment. Table 2 reports the median ranks of the methods for the experiment just described.

Overall, methods that choose σ from a continuous set outperformed their finite counterparts. This suggests again that for the finite kernel learning methods the range of σ and the discretization of this range is important to the accuracy of the resulting classifier.

UCI Letter Recognition. In another experiment, we evaluated these methods on 12 binary classification tasks from the UCI Letter recognition dataset. This dataset includes 20000 data points, each with 16 features, of the 26 capital letters in the English alphabet. For each binary classification task, the training and validation sets include 300 and 200 data points, respectively. The misclassification error is measured over 1000 test points. As with MNIST, we used Gaussian kernels. However, in this experiment, we ran both the one- and multi-dimensional search versions of CA and CR. The rest of the methods learn a single parameter and the finite kernel learning methods were provided with

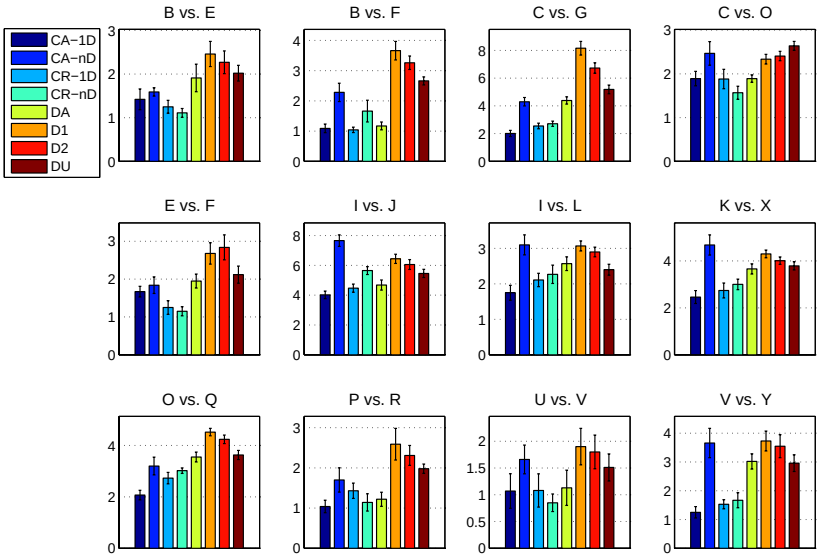


Fig. 6: Misclassification percentages in different tasks of the UCI Letter recognition dataset.

20 kernels with σ's chosen from the interval $[1, 200]$ in an equidistant manner. The number of kernels for these methods is chosen to make their training time comparable to that of CA-1D. The plots of misclassification error are shown in Figure 6. We report the median rank of each method in Table 2. While the one-dimensional version of our method outperforms the rest of the methods, the classifier built on the kernel found by the multi-dimensional version of our method did not perform well. We examined the value of alignment between the learned kernel and the target label kernel on the test set achieved by each method. The results are shown in Figure 7. The multi-dimensional version of our method achieved the highest value of alignment in every task in this experiment. This experiment demonstrates that high alignment values between the learned kernel and the ideal kernel do not necessarily translate into a more accurate classifier. Aside from this observation, the same trend observed in the MNIST data can be seen here. The continuous kernel learning methods outperform the finite kernel learning methods.

Miscellaneous datasets. In the last experiment we evaluate all methods on 11 datasets chosen from the UCI machine learning repository and Delve datasets. Most of these datasets were used previously to evaluate kernel learning algorithms (See, e.g. Lanckriet et al, 2004; Cortes et al, 2009a,b, 2010; Rakotomamonjy et al, 2008). The specification of these datasets are shown in Table 3.

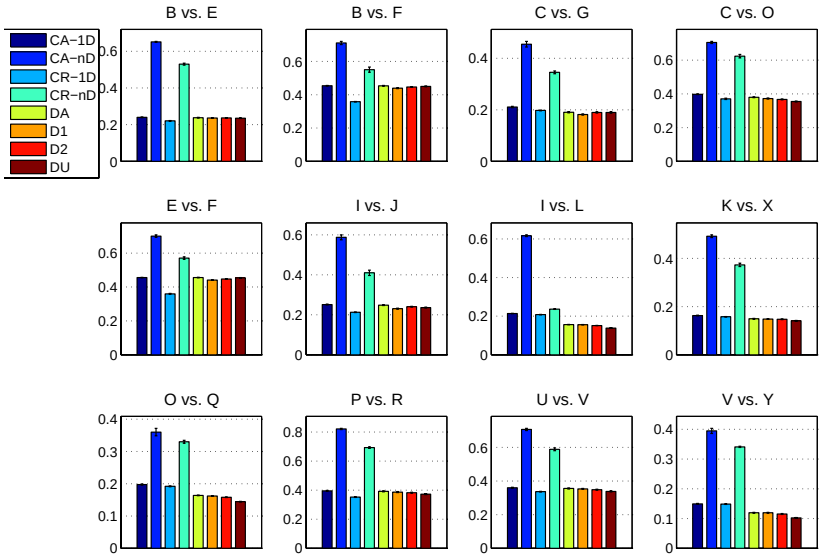


Fig. 7: Alignment values in different tasks of the UCI Letter recognition dataset.

Table 3: Datasets used in the experiments

Dataset	**# features**	**# instances**	**Train size**	**Valid. size**	**Test size**
Banana	2	5300	500	1000	2000
Breast Cancer	9	263	52	78	133
Diabetes	8	768	153	230	385
German	20	1000	200	300	500
Heart	13	270	54	81	135
Image Seg.	18	2086	400	600	1000
Ringnorm	20	7400	500	1000	2000
Sonar	60	208	41	62	105
Splice	60	2991	500	1000	1491
Thyroid	5	215	43	64	108
Waveform	21	5000	500	1000	2000

The performance of each method is shown in Figure 8. The median rank of each method is shown in Table 2. Contrary to the Letter experiment, in this case the multi-dimensional version of our method outperforms the rest of the methods.

Training Times. We measured the time required for each run and each kernel learning method in the MNIST and the UCI Letter experiments. In each

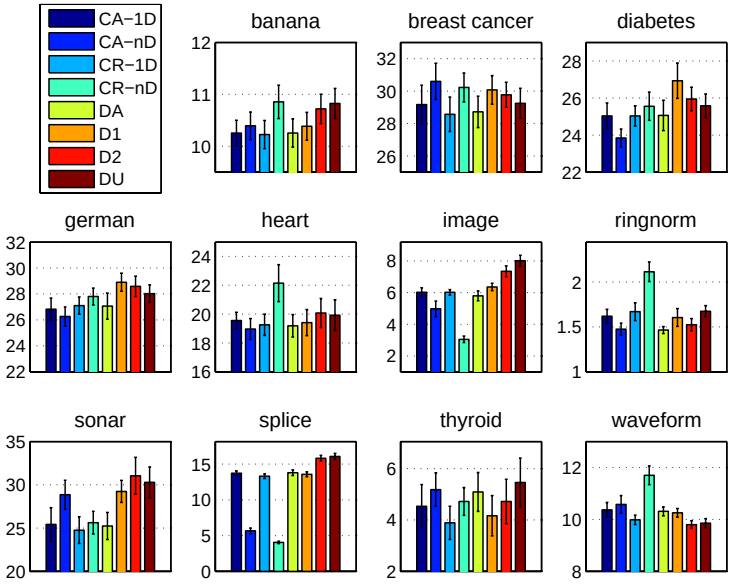


Fig. 8: Misclassification percentages obtained in 11 datasets.

case we took the average of the training time of each method over all tasks. The average required time along with the standard error values are shown in Table 2. Among all methods, the DU method is fastest, which is expected, as it requires no additional time to compute kernel weights. The CA-1D is the fastest among the rest of the methods. In these experiments our method converges in less than 10 iterations (kernels). The general trend is that one-stage kernel learning methods, i.e., D1, D2, and CR, are slower than two-stage methods, CA and DA. In these experiments CR is slower than its counterpart, CA, since it usually requires more iterations (around 50) to converge. The multi-dimensional search methods (CA-nD and CR-nD) are significantly slower than their one-dimensional counterparts. The explanation of this is that these methods introduce an additional regularization parameter that needs to be tuned based on cross-validation.

Fix this up. CR regularizes, right?

Is there any chance to speed this up? See the NIPS paper http://nips.cc/Conferences/2012/Program/event.php?ID=3183.

Make the reviewer happy by mentioning that the

4 Conclusion and Future Work

We presented a novel method for kernel learning. This method addresses the problem of learning a kernel in the positive linear span of some continuously parameterized kernel family. The algorithm implements a steepest ascent approach to forward stagewise additive modeling to maximize an empirical centered correlation measure between the kernel and the empirical approximation

to the ideal response-kernel. The method was shown to perform well in a series of experiments, both with synthetic and real data. We showed that in one-dimensional kernel parameter search, our method outperforms standard multiple kernel learning methods without the need to discretizing the parameter space. While the method of Argyriou et al (2005) also benefits from searching in a continuous space, it was seen to require significantly more training time compared to our method. We also showed that our method can successfully deal with high-dimensional kernel parameter spaces.

The main lesson of our experiments is that the methods that start by discretizing the kernel space without using the data might lose the potential to achieve good performance before any learning happens.

A secondary outcome of our experiments is the observation that although test-set alignment is generally a good indicator of good predictive performance, a larger test-set alignment does not necessarily transform into a smaller misclassification error. Although this is not completely unexpected, we think that it will be important to thoroughly explore the implications of this observation.

We think that currently our method is perhaps the most efficient method to design data-dependent dictionaries that provide competitive performance. However, for now this remains an empirical observation. Thus, it remains an interesting problem to find methods that enjoy strong bounds on both of their performance and computational costs. Another important question is to scale up the method to work with large datasets. The current algorithm scales quadratically with the number of kernels because of the need to compute the data Grammian. However, for large datasets, it might be sufficient to compute an approximation of the Grammian, e.g., by subsampling the data. Determining the best approach to subsampling however is left for future work.

A Proofs

A.1 Proof of Proposition 1

First, notice that the limit in (1) is a directional derivative, $D_{\kappa_\sigma} f(k^{t-1})$. By the chain rule,

$$D_{\kappa_\sigma} f(k^{t-1}) = \langle\, K(\kappa_\sigma),\, F_c'(K(k^{t-1}))\,\rangle_F\,,$$

where, for convenience, we defined $F_c(K) = A_c(K, \hat{K}^*)$. Define

$$F(K) = \langle K, \hat{K}_c^* \rangle_F / (\|K\|_F \|\hat{K}_c^*\|_F)$$

so that $F_c(K) = F(K_c)$. Some calculations give that

$$F'(K) = \frac{\hat{K}_c^* - \|K\|_F^{-2} \langle K, \hat{K}_c^* \rangle_F \, K}{\|K\|_F \|\hat{K}_c^*\|_F}$$

(which is the function defined in (3)). We claim that the following holds:

Lemma 1 $F'_c(K) = C_n F'(K_c) C_n$.

Proof By the definition of derivatives, as $H \to 0$,

$$F(K+H) - F(K) = \langle F'(K), H \rangle_F + o(\|H\|).$$

Also,

$$F_c(K+H) - F_c(K) = \langle F'_c(K), H \rangle_F + o(\|H\|).$$

Now,

$$\begin{aligned} F_c(K+H) - F_c(K) &= F(C_n K C_n + C_n H C_n) - F(C_n K C_n) \\ &= \langle F'(K_c), C_n H C_n \rangle_F + o(\|H\|) \\ &= \langle C_n F'(K_c) C_n, H \rangle_F + o(\|H\|), \end{aligned}$$

where the last property follows from the cyclic property of trace. Therefore, by the uniqueness of derivative, $F'_c(K) = C_n F'(K_c) C_n$.

Now, notice that $C_n F'(K_c) C_n = F'(K_c)$. Thus, we see that the value of σ_t^* can be obtained by

$$\sigma_t^* = \arg\max_{\sigma \in \Sigma} \left\langle K(\kappa_\sigma), F'\left((K(k^{t-1}))_c \right) \right\rangle_F ,$$

which was the statement to be proved.

A.2 Proof of Proposition 2

Let $g(\eta) = f(k^{t-1} + \eta \kappa_{\sigma_t^*})$. Using the definition of f, we find that with some constant $\rho > 0$,

$$g(\eta) = \rho \frac{a + b\eta}{(c + 2d\eta + e\eta^2)^{1/2}}.$$

Notice that here the denominator is bounded away from zero (this follows from the form of the denominator of f). In particular, $e > 0$. Further,

$$\lim_{\eta \to \infty} g(\eta) = - \lim_{\eta \to -\infty} g(\eta) = \rho \frac{b}{\sqrt{e}}. \tag{5}$$

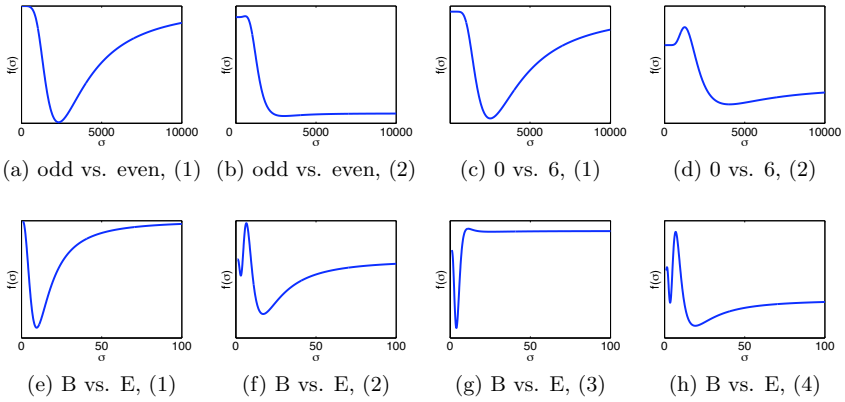
Taking the derivative of g we find that

$$g'(\eta) = \rho \frac{bc - ad + (bd - ae)\eta}{(c + 2d\eta + e\eta^2)^{3/2}}.$$

Therefore, g' has at most one root and g has at most one global extremum, from which the result follows by solving for the root of g' (if g' does not have a root, g is constant).

B Details of the numerical experiments

In this section we provide further details and data for the experimental results.



(a) odd vs. even, (1) (b) odd vs. even, (2) (c) 0 vs. 6, (1) (d) 0 vs. 6, (2)

(e) B vs. E, (1) (f) B vs. E, (2) (g) B vs. E, (3) (h) B vs. E, (4)

Fig. 9: The *flipped* objective function underlying (2) as a function of σ, the parameter of a Gaussian kernel in selected MNIST and UCI Letter problems. Our algorithm needs to find the minimum of these functions (and similar ones).

B.1 Non-Convexity Issue

As we mentioned in Section 2, our algorithm may need to solve a non-convex optimization problem in each iteration to find the best kernel parameter. Here, we explore this problem numerically, by plotting the function to be optimized in the case of a Gaussian kernel with a single bandwidth parameter. In particular, we plotted the objective function of Equation 2 with its sign flipped, therefore we are interested in the local minima of function $h(\sigma) = -\left\langle K(\kappa_\sigma), F'(\,(K(k^{t-1}))_c)\,\right\rangle_F$, see Figure 9. The function h is shown for some iterations of some of the tasks from both the MNIST and the UCI Letter experiments. The number inside parentheses in the caption specifies the corresponding iteration of the algorithm. On these plots, the objective function does not have more than 2 local minima. Although in some cases the functions have some steep parts (at the scales shown), their optimization does not seem very difficult.

B.2 Implementation details

The one-dimensional version of our algorithm, CA-1D, and the CR-1D method, employ Matlab's fmincon function with multiple restarts from the set $10^{\{-3,\ldots,5\}}$, to choose the kernel parameters. The multi-dimensional versions, i.e. CA-nD and CR-nD, use fmincon only once, since in this particular example the search method runs on a multi-dimensional search space, which makes the search an expensive operation. The starting point of multi-dimensional search methods is a vector of equal elements where this element is the weighted average of the

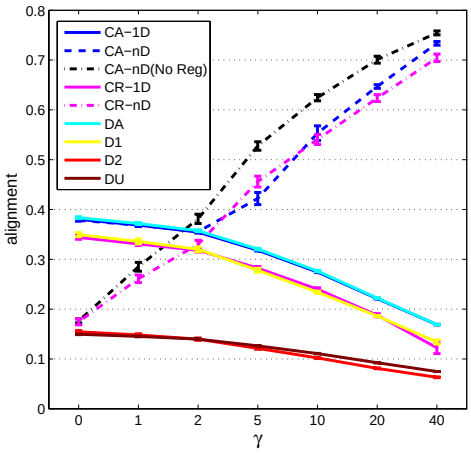


Fig. 10: Alignment values in the 50-dimensional synthetic dataset experiment.

kernel parameters found by the corresponding one-dimensional search method, weighted by the coefficient of the corresponding kernels.

We tuned the value of the regularization parameter in Equation (4) from $10^{\{-5,\ldots,14\}}$ using an independent validation set (the best value of λ is the one that achieves the highest value of alignment on the validation set). We decided to use a large validation set, following essentially the practice of Kloft et al (2011, Section 6.1), to make sure that in the experiments reasonably good regularization parameters are used, i.e., to factor out the choice of the regularization parameters. This might bias our results towards CA-nD, as compared to CA-1D, though similar results were achieved with a smaller validation set of size 200. As a final detail note that D1, D2 and CR also use the validation set for choosing the value of their regularization factor, and together with the regularizer, the weights also. Hence, their results might also be positively biased (though we do not think this is significant, in this case).

The training times reported in the paper correspond to a single run of learning kernel weights and learning a classifier. The extra time required to tune SVM regularization parameter is not included. However, for the CA-nD method the time required to tune the regularization parameter λ has been taken into account.

Figure 10 shows the (centered) alignment values for the learned kernels (on the test data) as a function of the relevance parameter γ. It can be readily seen that the multi-dimensional methods have an edge over other methods when the number of irrelevant features is large, in terms of kernel alignment. As seen in Figure 3, this edge is also transformed into an edge in terms of test-set performance. Note also that the discretization is fine enough so that the alignment maximizing finite kernel learning method DA can achieve the same alignment as the method CA-1D.

References

Argyriou A, Micchelli C, Pontil M (2005) Learning convex combinations of continuously parameterized basic kernels. In: Proceedings of the 18th Annual Conference on Learning Theory, pp 338–352

Argyriou A, Hauser R, Micchelli C, Pontil M (2006) A DC-programming algorithm for kernel selection. In: Proceedings of the 23rd international conference on Machine learning, pp 41–48

Boyd S, Vandenberghe L (2004) Convex optimization. Cambridge University Press

Bühlmann P, van de Geer S (2011) Statistics for High-Dimensional Data: Methods, Theory and Applications. Springer

Cortes C (2009) Invited talk: Can learning kernels help performance? In: ICML '09, pp 1–1

Cortes C, Mohri M, Rostamizadeh A (2009a) L2 regularization for learning kernels. In: Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence, pp 109–116

Cortes C, Mohri M, Rostamizadeh A (2009b) Learning non-linear combinations of kernels. In: Advances in Neural Information Processing Systems 22, pp 396–404

Cortes C, Mohri M, Rostamizadeh A (2010) Two-stage learning kernel algorithms. In: Proceedings of the 27th International Conference on Machine Learning, pp 239–246

Cristianini N, Kandola J, Elisseeff A, Shawe-Taylor J (2002) On kernel-target alignment. In: Advances in Neural Information Processing Systems 15, MIT Press, pp 367–373

Frank A, Asuncion A (2010) UCI machine learning repository. URL http://archive.ics.uci.edu/ml

Hastie T, Tibshirani R, Friedman J (2001) The Elements of Statistical Learning. Springer Series in Statistics, Springer-Verlag New York

Kloft M, Brefeld U, Sonnenburg S, Zien A (2011) Lp-norm multiple kernel learning. Journal of Machine Learning Research 12:953–997

Lanckriet G, Cristianini N, Bartlett P, Ghaoui L, Jordan M (2004) Learning the kernel matrix with semidefinite programming. Journal of Machine Learning Research 5:27–72

LeCun Y, Cortes C (2010) MNIST handwritten digit database. URL http://yann.lecun.com/exdb/mnist/

Rakotomamonjy A, Bach F, Canu S, Grandvalet Y (2008) SimpleMKL. Journal of Machine Learning Research 9:2491–2521

Sonnenburg S, Rätsch G, Schäfer C, Schölkopf B (2006) Large scale multiple kernel learning. The Journal of Machine Learning Research 7:1531–1565