A Unified View of Localized Kernel Learning*

John Moeller† Sarathkrishna Swaminathan† Suresh Venkatasubramanian†

Abstract

Multiple Kernel Learning, or MKL, extends (kernelized) SVM by attempting to learn not only a classifier/regressor but also the best kernel for the training task, usually from a combination of existing kernel functions. Most MKL methods seek the combined kernel that performs best over *every* training example, sacrificing performance in some areas to seek a global optimum. *Localized* kernel learning (LKL) overcomes this limitation by allowing the training algorithm to match a component kernel to the examples that can exploit it best. Several approaches to the localized kernel learning problem have been explored in the last several years. We unify many of these approaches under one simple system and design a new algorithm with improved performance. We also develop enhanced versions of existing algorithms, with an eye on scalability and performance.

1 Introduction

Kernel-based learning algorithms require the user to specify a kernel that defines the *shape* of the underlying data space. *Kernel learning* is the problem of learning a kernel from the data, rather than providing one by fiat. Most approaches to kernel learning assume some structure to the kernel being learned: either as an explicit linear Mahalanobis representation, or as some finite combination of set of fixed kernels. This latter class of approaches is often called *multiple kernel learning*.

While multiple kernel learning has been studied extensively and has had success in identifying the right kernel for a given task, it is expressively limited because each kernel has influence over the entire data space. Consider an example of a binary classification task, depicted in Figure 1. On the left side we show the results of classifying the data with a global MKL method (here, the UNIFORM method of Cortes et al. [5]) and on the right side we show the results of classification with our new proposed method LD-MKL. Because the global method requires that each kernel be used to classify each point in the same way, the decision boundary is not as flexible and many more support points are required.

Motivated by this, a few directions have been proposed to build *localized* kernel learning solutions. Gönen and Alpaydin [7] introduced the idea of a learned *gating* function that modulated the influence of a kernel on a point (LMKL). Lei et al. [14] observed that LMKL uses a non-convex optimization and suggested using a probabilistic clustering to generate part of the gating function beforehand, in order to obtain a convex optimization and thus prevent overfitting and yield generalization bounds (C-LMKL). Kannao and Guha [10] suggested a different approach to find a gating function by looking at individual features of the input, and uses successes of the individual kernels to learn the gating function through support vector regression (SwMKL).

All of the above approaches invoke a fixed-kernel SVM subroutine as part of the algorithm. This is inefficient, and prevents these methods from scaling. C-LMKL does argue for a convex formulation of the problem, but does not directly address the problem of scaling.

1.1 Our Contributions.

We present a *unified* interpretation of localized kernel learning that generalizes all of the approaches described above, as well as the general multiple kernel learning formulation. This interpretation yields a new algorithm for LKL that is superior to all existing methods. In addition, we make use of prior work on scalable multiple kernel learning [17] as a subroutine to make existing methods for LKL scale well, improving their performance significantly in some cases.

Our interpretation relies on a geometric interpretation of gating functions in terms of *local* reproducing kernel Hilbert spaces acting on the data. This interpretation also helps explain the observation above (only empirically observed thus far) that local kernel learning methods appear to produce good classifiers with fewer support points than global methods.

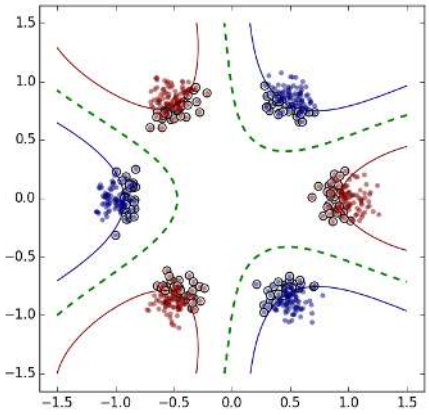
2 Background

We use the notation $[a \dots b]$ to indicate a sequence of integers i such that $a \leq i \leq b$. We use bold Roman letters to indicate vectors ($\mathbf{x}$) and matrices ($\mathbf{A}$). Matrices are capitalized. Because we discuss several approaches to localized MKL, and each uses a different set of notations, we choose our own convention:

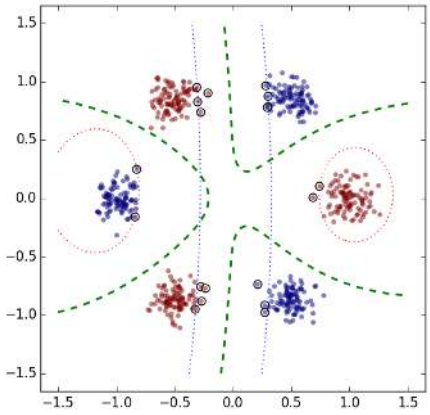
- i indexes kernel functions/spaces and the number of individual kernel spaces is m.

*This research was funded in part by the NSF under grants IIS-1251049 and CNS-1302688.

†School of Computing, University of Utah. **Email:** {*moeller, sarath, suresh*}*@cs.utah.edu*



(a) Classifier produced by global MKL with 118 support points.



(b) Classifier produced by LD-MKL with 20 support points.

Figure 1: Illustration of the difference between local and global multiple kernel learning. In each example, the classifier is built from two kernels, one quadratic and one Gaussian. Points from the two classes are colored blue and red (with transparency as a hint towards density). The decision boundary is marked in green and the margin boundaries are in the appropriate colors for the global case. For the local case, the margins of each kernel are plotted with dotted lines, red for Gaussian and blue for quadratic. Support points are indicated by black circles around points. Note that the classifier uses a soft-margin loss and so support points may not be exactly on the margin boundary. The global version has 118 support points, while the local version has only 20.

- j and k index examples and the number of training points is n.
- t is used to indicate iterations in an algorithm.
- The Greek letter κ is used to indicate a kernel function. $\kappa_i(\mathbf{x}_j, \mathbf{x}_k)$ is the ith kernel function applied to training examples $\mathbf{x}_j$ and $\mathbf{x}_k$.

The softmax operator is a map $\mathbb{R}^d \to (0,1)^d$ that normalizes the input vector to the range $(0,1)$:

$$\text{softmax}(\mathbf{x}) = \left(\frac{\exp x_i}{\sum_{k=1}^{d} \exp x_k} \right)_{i=1}^{d}$$

2.1 Prior work on localized kernel learning Since our work unifies a number of different approaches to performing localized multiple kernel learning, we start with a self-contained review of these methods.

The core idea (somewhat simplified) of kernel learning for classification is to fix a space $\mathscr{K}$ of kernel functions $\kappa(\cdot,\cdot)$ and learn a kernel $\kappa \in \mathscr{K}$ that best classifies the training data. The term *multiple kernel learning* comes from the fact that the space $\mathscr{K}$ is often expressed as the set of all positive combinations of a fixed set of kernels, and thus the search for a specific kernel turns into a search for a set of parameters η_i, one per kernel, so that the resulting discriminant function can be written as

$$f(x) = \sum_{i=1}^{m} \eta_i \langle \mathbf{w}_i, \phi_i(\mathbf{x}) \rangle + b$$

The rationale for *localized* kernel learning (as illustrated in Section 1) is to allow the weight assigned to different kernels to vary in different parts of the data space to incorporate any local structure in the data.

Localized Multiple Kernel Learning (LMKL). Gönen and Alpaydin [6] were the first to propose an algorithm to solve this problem. They called their method *localized multiple kernel learning* (LMKL). The idea was generalize the η_i to be functions of the data $\mathbf{x}$ as well as a set of *gating parameters* $\mathbf{V} \in \mathbb{R}^{d \times m}$.

They defined a gating function as:

$$\eta(\mathbf{x}|\mathbf{V}) = \text{softmax}(\mathbf{x}^\top \mathbf{V} + \mathbf{v}_0),$$

where $\mathbf{v}_0$ is an m-dimensional vector of offsets[1].

Given such a gating function, they then defined a generalized discriminant function:

$$f(\mathbf{x}) = \sum_{i=1}^{m} \eta_i(\mathbf{x}|\mathbf{V}) \langle w_i, \phi_i(\mathbf{x}) \rangle + b,$$

[1] In later works they proposed other gating functions that employed sigmoids and Gaussian functions [7].

Expressing the classifier function leads to a non-convex optimization involving the parameters V. They then proposed solving this problem using a two-step alternating optimization algorithm, summarized in Algorithm 1.

Algorithm 1 `LMKL`

1: **repeat**
2: Calculate $\mathbf{K}_\eta$, the Gram matrix of the combined kernel, with the gating functions η_i:
3: $(K_\eta)_{jk} \leftarrow \kappa_\eta(\mathbf{x}_j, \mathbf{x}_k) = \sum_{i=1}^m \eta_i(\mathbf{x}_j)\kappa_i(\mathbf{x}_j, \mathbf{x}_k)\eta_i(\mathbf{x}_k)$
4: Solve canonical SVM with $\mathbf{K}_\eta$
5: Update gating parameters $\mathbf{V}$ using gradient descent
6: **until** convergence

The complexity of the overall algorithm is dominated by the time to perform the canonical SVM. Other variants of this basic framework include Yang et al. [27], which allows gating functions to operate on groups of points, and Han and Liu [8] which incorporates a gating function based on pairwise similarities inferred from a kernel density estimate for each kernel.

Convex LMKL (`C-LMKL`). More recently, Lei et al. [14] noted the non-convex nature of the above objective function. In order to avoid the tendency of such functions to overfit to the training data, they proposed an alternate *convex* formulation of the localized multiple kernel learning problem. The central idea of their approach is to first construct a soft clustering of the data, represented by a soft assignment function $c_\ell(x_j)$ that associates point x_j with cluster ℓ. Next, they define parameters $\beta_{\ell i}$ that associate each of m kernels with each cluster ℓ: in effect, the soft clustering fixes the locality they wish to exploit, and the $\beta_{\ell i}$ then allow them to use different kernel combinations.

The resulting optimization is convex, assuming that the loss function is convex. This allows them to obtain generalization bounds as well as good prediction accuracy in practice. The optimization itself proceeds as a two-stage optimization: the first stage invokes a standard SVM solver to find the best weight vectors given the $\beta_{\ell i}$ and the second stage optimizes $\beta_{\ell i}$ for given weights. This latter stage can in fact be solved in closed form. Thus, as with `LMKL`, the term dominating the computation time is the use of an SVM solver.

Success-Based Locally-Weighted Kernel Combination (`SwMKL`) Kannao and Guha [10] introduced `SwMKL` as a way to localize kernel learning in a different manner. Their method is to analyze each kernel for its success on the input data, then construct a gating function based on smoothing the success with a regression, summarized in Algorithm 2.

Its complexity is controlled by the initial SVM computations, the different support vector regression operations, as well as the final SVM calculation on the combined kernel function. The experimental approach in [10] is to separate each kernel by feature – essentially creating individual kernels for each combination of kernel and feature and then combining them. When testing with this algorithm, we had much better success when using a kernel on all features.

Algorithm 2 `SwMKL`

1: **for all** $i \in [1..m]$ **do**
2: Train classifier $f_i : \mathbb{R}^d \to \{-1, 1\}$ with kernel κ_i
3: Train regressor $g_i : \mathbb{R}^d \to (0, 1)$ with $(\mathbf{X}, \delta(\mathbf{y}, f_i(\mathbf{X})))$
4: Train classifier using

$$\kappa(\mathbf{x}_j, \mathbf{x}_k) = \frac{\sum_{i=1}^m g_i(\mathbf{x}_j)\kappa_i(\mathbf{x}_j, \mathbf{x}_k)g_i(\mathbf{x}_k)}{\sum_{i=1}^m g_i(\mathbf{x}_j)g_i(\mathbf{x}_k)}$$

Sample-Adaptive Multiple Kernel Learning (`SAMKL`). An alternate approach employed by Liu et al. [15] is to separate out the assignment of kernels to points and the weights associated with the kernels. In their formulation, which they describe as *sample-adaptive multiple kernel learning*, they introduce latent *binary* variables to decide whether a particular kernel should operate on a particular point or not. Each point is therefore mapped to a single point in the product of the feature spaces defined by the given kernels. Now they run a two-stage alternating optimization: in the first stage, given fixed values of the latent variables, they solve a multiple kernel learning problem for the different subspaces simultaneously, and then they run an integer program solver to obtain new values of the latent variables. Note that each step of the iteration here involves costly operations (an MKL solver and an integer program solver) in comparison with the SVM solvers in the other approaches.

3 A unified view of localized kernel learning

One of the contributions of this work is a unified perspective that integrates these different approaches and also helps explain the somewhat paradoxical fact that localized multiple kernel often yields classifiers with *fewer* support points than standard multiple kernel learning methods.

3.1 Localization via Hilbert subspaces Consider the following generalized and gated kernel κ_γ defined as:

$$\kappa_\gamma(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^m \gamma_i(\mathbf{x}, \mathbf{x}')\kappa_i(\mathbf{x}, \mathbf{x}'),$$

where $\gamma_i : \mathbb{R}^d \times \mathbb{R}^d \to [0, 1]$ is a "gating function."

We call γ_i *separable* if it decomposes into a product of a function with itself, i.e. if $\gamma_i(\mathbf{x}, \mathbf{x}') = \eta_i(\mathbf{x})\eta_i(\mathbf{x}')$, where $\eta_i : \mathbb{R}^d \to [0, 1]$. For the rest of this section, we only consider separable gating functions. We also make two additional assumptions for all $\mathbf{x} \in \mathbb{R}^d$: (1) $\sum_{i=1}^m \eta_i(\mathbf{x}) = 1$, and (2) $\eta_i(\mathbf{x}) \geq 0 \; \forall i \in [1..m]$.

The RKHS of a localized kernel. Consider the Gram matrix $\mathbf{H}_i$ of γ_i: specifically the $n \times n$ matrix $\mathbf{H}_i$ whose $(j,k)^{\text{th}}$ entry is $\gamma_i(\mathbf{x}_j, \mathbf{x}_k)$ (we will refer to this later as the *gating matrix*). If γ_i is separable, then we know that $\mathbf{H}_i$ is positive definite, because it can be expressed as the outer product of a vector with itself ($\mathbf{H}_i = \eta^\top \eta$). Defining $\mathbf{K}_i$ as the Gram matrix of the kernel κ_i, it is now easy to see that we can write the Gram matrix of the kernel κ_γ as the matrix $\sum_i \mathbf{H}_i \circ \mathbf{K}_i$ (where $\circ$ denotes the Schur product).

In the separable case, since both $\mathbf{H}_i$ and $\mathbf{K}_i$ are positive definite, so is $\mathbf{H}_i \circ \mathbf{K}_i$ by the Schur product theorem. Therefore $\gamma_i(\mathbf{x}, \mathbf{x}')\kappa_i(\mathbf{x}, \mathbf{x}')$ is a kernel function, and the corresponding lifting map is $\eta_i(\mathbf{x})\Phi_i(\mathbf{x})$.

We know that a positive linear combination of kernel functions is itself a kernel function and induces a product reproducing kernel Hilbert space (RKHS) that is a simple Cartesian product of all the individual Hilbert spaces. The inner product of this space is just the sum of all the individual inner products. Thus the kernel κ_γ has a natural feature space as the product of the individual feature spaces.

Localization. This framework now allows us to provide a geometric intuition for why localized kernel learning might be able to reduce the number of required support points. Suppose that $\eta_i(\mathbf{x}) = 0$. This implies that $\langle \eta_i(\mathbf{x})\Phi_i(\mathbf{x}), \eta_i(\mathbf{x}')\Phi_i(\mathbf{x}') \rangle$ is always 0. Because the i^{th} RKHS is one component of the product RKHS, this means that $\eta_i(\mathbf{x})\Phi_i(\mathbf{x})$ lies in some subspace perpendicular to this RKHS.

Furthermore, suppose that $\eta_i(\mathbf{x}) = 1$. By our assumptions that $\sum_{i=1}^m \eta_i(\mathbf{x}) = 1$ and that η_i is non-negative, this means that $\eta_i(\mathbf{x})\Phi_i(\mathbf{x})$ is absent from *every other* RKHS in the product. Therefore $\eta_i(\mathbf{x})\Phi_i(\mathbf{x})$ lies exclusively in the i-th RKHS.

This partitioning behavior is advantageous, because it is much simpler to find decision boundaries within the individual RKHS components rather than trying to find one that will work for all at the same time. The decision hyperplane in the product RKHS will be the unique hyperplane that intersects all the subspaces in their respective decision boundaries.

Depending on the gating function, there will of course be some training examples that are "confused" about what subspace to lie in. Therefore we wish to pick a set of gating functions that reduces this confusion. The *crucial* property of the gating function γ_i and the gating matrix $\mathbf{H}_i$ is that they are separable. With the separability constraint, we need only find a set of one-dimensional functions that works for the training data[2].

3.2 Gating and optimization The localized MKL algorithms described above (and in fact virtually all localized kernel learning algorithms) can be placed in the framework we have just described, thus explaining in a broader context how their localization works. The specifics differ on how the function κ_γ is generated:

1. **Gating**: Each algorithm has a gating function $\gamma_i(\mathbf{x}, \mathbf{x}')$ for every kernel function κ_i. Recall that the gating function simply controls the degree to which a kernel responds to a particular point.
2. **Optimization**: Each algorithm also has an optimization behavior, that either generates or tunes each γ_i.

`LMKL`**:**

- **Gating**: The gating function is separable, and $\eta(\mathbf{x}) = (\eta_1(\mathbf{x}), \ldots, \eta_i(\mathbf{x}), \ldots) = \text{softmax}(\mathbf{x}^\top \mathbf{V} + \mathbf{v}_0)$.
- **Optimization**: Alternating optimization using an SVM solver to find the kernel support points and stochastic gradient descent to find the parameters $\mathbf{V}$, $\mathbf{v}_0$.

`C-LMKL`**:**

- **Gating**: The gating function is separable, but not directly. It is equal to $\sum_{r=1}^{\ell} \beta_{ir} c_r(\mathbf{x}) c_r(\mathbf{x}')$, where $\beta_{ir} \geq 0$ is the weight with which kernel i influences points associated with cluster r, and c_r is the (pre-computed) likelihood of $\mathbf{x}$ falling into cluster r.

  Since γ_i decomposes into a linear combination $\beta_{ir} c_r(\mathbf{x}) c_r(\mathbf{x}')$, we can apply Section 3.1 to `C-LMKL`. In `C-LMKL` we replicate each kernel ℓ times (once for each c_r) and give each its own weight $\sqrt{\beta_{ir}}$.
- **Optimization**: The parameters β_{ir} are learned through (convex) optimization and the functions c_r are generated through ℓ different clusterings.

`SwMKL`**:**

- **Gating**: The gating function is not separable in this case, because the γ_i are normalized *pairwise*. $\gamma_i(\mathbf{x}, \mathbf{x}') = g_i(\mathbf{x}) g_i(\mathbf{x}') / Z(\mathbf{x}, \mathbf{x}')$, where $Z(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^m g_i(\mathbf{x}) g_i(\mathbf{x}')$, and g_i are the SVR-generated functions.

  Note While κ_γ may be positive definite, its individual terms are very unlikely to be so. It is therefore not clear whether this algorithm in its unmodified form can be placed in our unified context. We explore this issue in greater depth in the next section.
- **Optimization**: The gating functions g_i are generated using SVR from $\mathbf{X} \times \delta(\mathbf{y}, \hat{\mathbf{y}}_i)$.

`SAMKL`

- **Gating:** $\eta_i(\mathbf{x})$ is a *binary-valued* function that decides if kernel i should be used for point $\mathbf{x}$.

[2] If the gating function is not separable, but is decomposable into a positive linear combination of a fixed-size set of separable functions, then the partitioning is still possible – see Section 3 below, under "`C-LMKL`".

- **Optimization:** The optimization is an alternating optimization between the gating function and the kernel parameters. Because the η_i are binary-valued, a further *multiple* kernel learning step is required to determine kernel weights and support vectors for the classifier, and the gating parameters are learned with an *integer programming* solver.

Global ("classic") MKL:

- **Gating**: $\eta_i(\mathbf{x}) = \sqrt{\mu_i}$, where $\mu_i \geq 0$ is constant for every kernel, that is, does not change relative to each point.
- **Optimization**: The μ_i can be optimized using several methods including stochastic gradient descent, multiplicative weight updates and alternation.

4 LD-MKL: A new algorithm for localized kernel learning

Viewing the algorithms for localized kernel learning in a common framework illustrates both their commonalities and their weaknesses. With the exception of SwMKL, all the approaches make use of a two- (or three-) stage optimization of which LibSVM is one component. As we shall see in our experiments, this renders these methods quite slow and not easy to scale. SwMKL on the other hand avoids this problem by doing single SVM calculations for each kernel and then combining them into a single larger kernel. This improves its running time, but makes it incur a large memory footprint in order to build a classifier for the final kernel.

We now present a new approach, inspired by SwMKL, that addresses these concerns. Our method, which we call LD-MKL (*localized decision-based multiple kernel learning*), fits into the unified framework for localized kernel learning via the use of local Hilbert spaces, avoids the large memory footprint of SwMKL, and also scales far more efficiently than the other multi-stage optimizations.

We start by observing that the first steps of Algorithm 2 give us a classifier f_i and a gating function g_i. The function f_i, since it is an SVM decision function, can be formulated as

$$f_i(\mathbf{x}) = \sum_{j=1}^{n} \alpha_{ij} y_j \kappa_i(\mathbf{x}_j, \mathbf{x}).$$

Note that α has an additional index to indicate which kernel we trained the classifier against. Suppose we modify this function to incorporate the gating function g_i[3]:

$$\overline{f}_i(\mathbf{x}) = \sum_{j=1}^{n} \alpha_{ij} y_j g_i(\mathbf{x}_j) \kappa_i(\mathbf{x}_j, \mathbf{x}). \tag{4.1}$$

$\overline{f}_i$ is the SVM prediction function, but where each support point α_{ij} is weighted by its gating value. We can now construct a weighted vote using these functions. We combine the output of each $\overline{f}_i$, apply tanh[4], and weight by g_i:

$$f(\mathbf{x}) = \sum_{i=1}^{m} g_i(\mathbf{x}) \tanh(\overline{f}_i(\mathbf{x})) \tag{4.2}$$

Algorithm 3 contains the listing of this procedure. Note that we *retrain* each classifier on the subset of the data where the corresponding gating function is significant (i.e. is greater than $1/m$). This reduces the support points considerably because the classifier is retrained only on points that it classified well.

Algorithm 3 LD-MKL

1: **for all** $i \in [1..m]$ **do**
2: Train classifier $f_i : \mathbb{R}^d \to \{-1, 1\}$ with kernel κ_i
3: Train regressor $g_i : \mathbb{R}^d \to (0, 1)$ with $(\mathbf{X}, \delta(\mathbf{y}, f_i(\mathbf{X})))$
4: Normalize regressors g_i with softmax
5: **for all** $i \in [1..m]$ **do**
6: Retrain classifier f_i on $(\mathbf{X}, \mathbf{y})_{g_i(\mathbf{x})>1/m}$
7: Compute each decision function using (4.1)
8: Classify inputs using sign of (4.2)

If commonly-used kernels are employed (such as linear, polynomial, or Gaussian kernels), then this method can take advantage of optimizations that exist in, e.g., LibSVM to train the classifiers and regressors quickly. The training step is over after the regressors are computed and normalized.

It is easy to see that LD-MKL has the desired *gating* behavior with separable gating functions. The optimization step is as before, but without needing to consult a final SVM solver.

5 Experiments

Our experiments will seek to validate two main claims: first that LD-MKL is indeed superior to prior localized kernel learning methods, and secondly that there is demonstrable reduction in the number of support points when using localized methods.

Scalability. In addition, we will also investigate ways to make existing localized methods more scalable. As noted, with the exception of SwMKL, all approaches use a multi-stage iterative optimizer of which one step is an SVM solver. We instead make use of a multiplicative-weight-update-based solver developed by Moeller et al. [17]. This method has a much smaller memory footprint and uses a lightweight iteration that also yields sparse support vectors. While this

[3] As discussed in the previous section, we assume that the gating functions have been normalized so that (1) $\sum_{i=1}^{m} g_i(\mathbf{x}) = 1$ and (2) $g_i(\mathbf{x}) \geq 0 \; \forall i \in [1..m]$.

[4] We use $\tanh(\overline{f}_i(\mathbf{x}))$ instead of the sign of $\overline{f}_i(\mathbf{x})$ so that uncertain classifications (i.e., kernels with resulting values of $\overline{f}_i(\mathbf{x})$ near 0) don't pollute the vote with noise.

solver was designed for *multiple* kernel learning, it is easily adapted as an SVM solver.

Data Sets. Table 1 contains information about the various datasets that we test with. All of these sets are taken from the `libsvm` repository at `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html`.

Dataset	Examples	Features
Breast Cancer	683	10
Diabetes	768	8
German-Numeric	1000	24
Liver	345	6
Mushroom	8192	112
Gisette	6000	5000
Adult	32561	123

Table 1: Datasets for comparison of `LMKL`, `SwMKL`, and `C-LMKL`

Methodology. In each of the experiments we partition the data randomly between 75% train and 25% test examples. Unless otherwise indicated, we repeat each partition 100 times and average the run time and the accuracy. In all experiments where we measure accuracy, we use the proportion of correctly classified points. Where possible, we also report the standard deviation of all measured values in parentheses. Superior values are presented in bold when the value minus the standard deviation is greater than all the other values plus their respective standard deviations.

In each experiment where we used a standard SVM solver, we used LibSVM [2] via `scikit-learn` [20]. We use the default LibSVM parameters (e.g., tolerance), and vary them only for changing specific kernels and passing specific kernel parameters. We use $C = 1.0$ and for Gaussian kernels, a range of γ from 2^{-4} to 2^4 are tried and the best accuracy observed is used.

Implementations. For `LMKL`, we took MATLAB code provided by Gonen[5] and converted it to `python` to have a common platform for comparison. This code included an SMO-based SVM solver which we converted as well. We verified correctness of intermediate and final results between the two platforms before running our experiments. For `SwMKL` and `LD-MKL`, we used the SVM and SVR solvers from `scikit-learn`. For `C-LMKL`, as prescribed by Lei et al. [14], we used a kernel k-means preprocessing step with a uniform kernel and three clusters. For large data sets, kernel k-means is very slow, and so we used a streaming method proposed by Chitta et al. [3] that runs the clustering algorithm on a sample (of size 1000 in our experiments) and then estimates probabilities for the remaining points. The *global* kernel learning methods we used were UNIFORM, which merely averages all kernels, `SPG-GMKL` [9][6] and **MWUMKL** [17].

Hardware. All experiments were conducted on Intel® Xeon® E5-2650 v2 CPUs, 2.60GHz with 64GB RAM and 8 cores.

5.1 Evaluating `LD-MKL`. We start with an evaluation of `LD-MKL` in Table 2. In each row, we present accuracy and timing (numbers in parentheses are standard deviations). As we can see, for small datasets, `SwMKL` is the fastest method, but for larger datasets `LD-MKL` is the fastest. In comparison with `LMKL` and `C-LMKL`, `SwMKL` and `LD-MKL` are considerably faster. This speedup is obtained without any significant loss in accuracy: in all cases, the accuracy of `LD-MKL` is either the best or is less than optimal in a statistically insignificant way.

5.2 Scaling As we can see in Table 2, `LMKL` and `C-LMKL` run very slowly as the data complexity increases (dimensions or number of points), and the primary bottleneck is the repeated invocation of an SVM solver. As described above, we replaced the SVM solver with a single-kernel version of **MWUMKL** and studied the resulting performance.

Table 3 summarizes the results of this experiment. As we can see, for both `LMKL` and `C-LMKL`, using a scalable SVM solver greatly improves the running time of the algorithm. In fact as we can see, the methods using LibSVM fail to complete on certain inputs, whereas the methods that use MWUMKL do not. We note that **MWUMKL** uses a parameter ε which is the acceptable error in the duality gap of the SVM optimization program. Higher ε values translate to more iterations, and accuracy can often improve (up to a point) with lower ε. Unless stated otherwise, we use $\varepsilon = 0.01$. Note that for this ε, accuracy does drop significantly in certain cases.

The case of `SwMKL` is a little more interesting. For smaller data sets the basic method works quite well, and indeed outperforms any enhancement based on using **MWUMKL**. However, this comes at a price: the `SwMKL` method requires a lot of memory to solve the final kernel SVM with a kernel formed by combining the base kernels. For smaller data sets this effect does not materially affect performance, but as we move to larger data sets like Adult, the method starts to fail catastrophically. Figure 2 illustrates the memory usage incurred by the three localized methods when not using **MWUMKL** and when using it. As we can see, the memory grows polynomially with the size of input.

Stress-testing. Scaling `LD-MKL` to truly large datasets can present a challenge because we make use of kernelized support-vector regression. There are several methods to address this problem which we will not enumerate here, but

[5] `http://users.ics.aalto.fi/gonen/icml08.php`

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

	LMKL	SwMKL	LD-MKL	C-LMKL
Breast Cancer	96.58 % (1.35 %) 122 s (8.9 s)	97.1% (1.1%) 0.15 s (2.1 ms)	97.1% (1.2%) **0.14 s** (2.36 ms)	96.7% (1.1%) 28.7 s (80 ms)
Diabetes	74.71% (3.07%) 157.6 s (34 s)	77.0% (2.7%) **0.18 s** (1.4 ms)	76.7% (2.56%) 0.24 s (3.58 ms)	76.4% (2.4%) 36.8 s (32 ms)
German-Numeric	70.78% (2.85%) 216 s (22 s)	75.7% (2.4%) **0.27 s** (3.8 ms)	75.84% (2.51%) 0.38 s (3.56 ms)	76.8% (1.6%) 69.6 s (20 ms)
Liver	62.49% (5.92%) 35.4 s (7.2 s)	69.3% (5.0%) **0.1 s** (1.4 ms)	65.19% (5.81%) 0.83 s (1.3 ms)	57.7% (5.1%) 7.4 s (129 ms)
Mushroom	99.99% (0.0%) 17.27 m (1.2 m)	99.9% (0%) 14.57 s (1.2 s)	100.0% (0.0%) **3.1 s** (0.3 s)	100% (0.0%) 2.43 h (12.6 m)
Gisette	97.22% (0.34%) 48.6 m (2.8 m)	97.06% (0.35%) 4.54 m (0.72 m)	96.88% (0.46%) 4.0 m (0.06 m)	96.5% (0.28%) 3.4 h (9.24 m)
Adult	-	84.6% (0.37%)	84.78% (0.4%)	84.65% (0.83%)
Income	-	6.65 m (1.2 m)	6.52 m (0.14 m)	7.5 h (14.3 m)

Table 2: Accuracies and running times for various datasets and methods, using **LibSVM** as the SVM solver. Numbers in parentheses are standard deviations. For the first four data sets, numbers are averaged over 100 runs. For the last three larger data sets, numbers are averaged over 20 runs. Values which are significantly superior to that of other methods are typeset in bold.

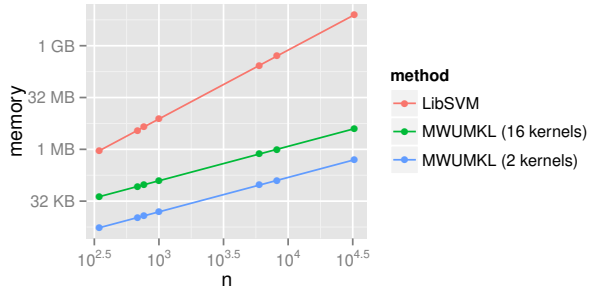


Figure 2: Minimum memory required (assuming double-precision floats) for LibSVM-based and MWUMKL-based methods. LibSVM-based methods exclude those that use only LibSVM's standard kernels, such as LD-MKL, but include those that construct a new kernel, such as LMKL, C-LMKL, and SwMKL. The values for n are taken from the "Examples" column from Table 1.

are targets for future versions of our algorithm.

5.3 Support Points We have argued earlier that localized multiple kernel learning methods have the potential to generate classifiers with comparable accuracy but fewer support points than global multiple kernel methods. This fact was first observed by Gönen and Alpaydin [6]. We now present detailed empirical evidence establishing this claim. We compare the different localized kernel learning methods to UNIFORM (a multiple kernel learning algorithm that merely takes an average of all the kernels in its dictionary [4]), SPG-GMKL [9] (an iterative MKL solver that uses the spectral projected gradient), and **MWUMKL**, run in its original form as a multiple kernel learning algorithm. Results are presented in Table 4. While we did not annotate the results with accuracy numbers for ease of viewing, all methods have comparable accuracy (as Table 2 also indicates).

We observe that in all cases, the classifier using the fewest support points is always one of the localized methods, and the differences are always significant. However, it is not the case that a single local method always performs best. In general, LD-MKL (and SwMKL) appear to perform slightly better, but this is not consistent. Nevertheless, the results provide a clear justification for the argument that local kernel learning indeed finds sparser solutions.

6 Related Work

The general area of kernel learning was initiated by Lanckriet et al. [13] who proposed to simultaneously train an SVM as well as learn a convex combination of kernel functions. The key contribution was to frame the learning problem as

	LMKL	SwMKL	LD-MKL	C-LMKL
Breast Cancer	97.08 % (1.1 %) 0.18 s (4.3 ms)	96.42% (1.6%) 0.18 s (1.2 ms)	93.4% (2.2%) 0.63 s (9.3 ms)	90.4% (2.4%) 5.6 s (122 ms)
Diabetes	73.19% (3.39%) **0.27 s** (18 ms)	76.63% (2.9%) 0.29 s (3.8 ms)	77.0% (3.4%) 0.48 s (46 ms)	71.1% (10%) 7.2 s (32 ms)
German-Numeric	70.07% (3.1%) 0.63 s (43 ms)	72.2% (3.29%) 0.62 s (10.2 ms)	73.0% (3.8%) 1.0 s (55 ms)	73.4% (4.1%) 16.3 s (101 ms)
Liver	56.82% (6.53%) 0.13 s (5 ms)	59.63% (10.47%) **0.11 s** (3.4 ms)	58.8% (8.0%) 0.3 s (6.5 ms)	49.7% (6.3%) 1.45 s (106 ms)
Mushroom	99.87% (0.1%) 24.4 s (0.36 s)	99.9% (0%) **21.3 s** (0.2 s)	99.9% (0.1%) 53.0 s (0.2 s)	98.8% (0.24%) 31.4 m (1.2 m)
Gisette	**97.28%** (0.4%) **8.2 m** (0.18 m)	69.96% (2.01%) 8.91 m (0.44 m)	92.2% (0.8%) 29.0 m (10 s)	90.26% (1.2%) 28.5 m (53.1 s)
Adult Income	57.4% (5.31%) 9.4 m (0.6 m)	83.96% (0.61%) 9.1 m (6.8 s)	80.2% (0.8%) 12.3 m (14.3 s)	84.65% (0.35%) 47.65 m (2.46 m)

Table 3: Accuracies and running times for various datasets and methods, using **MWUMKL** as the SVM solver. Numbers in parentheses are standard deviations. For the first four data sets, numbers are averaged over 100 runs. For the last three larger data sets, numbers are averaged over 20 runs. Values which are significantly superior to that of other methods are typeset in bold.

	Localized Methods				Global methods		
	SwMKL	LD-MKL	LMKL	C-LMKL	MWUMKL	UNIFORM	GMKL
Breast	11.4% (1%)	12.9% (1.1%)	38% (3.5%)	**10.8%** (1.1%)	21% (1.4%)	70.2% (1.9%)	15.1% (1%)
Diabetes	**55.2%** (1.3%)	56.4% (1.3%)	58% (1.7%)	73.9% (10%)	79.1% (1.7%)	70% (2%)	61.9% (1.2%)
German	52.2% (3.3%)	**43.4%** (2.4%)	89.2% (2.8%)	99.8% (0.3%)	81.7% (1.1%)	60.8% (1.6%)	68.4% (1.4%)
Liver	82.2% (1.7%)	70.2% (7.3%)	**63.1%** (2.3%)	88.1% (2.7%)	92.2% (1.6%)	89.6% (2.6%)	84.2% (1.9%)
Mushrooms	4.3% (0.2%)	8.1% (0.8%)	**1.9%** (0.1%)	4.0% (0.3%)	22.6% (0.1%)	96.4% (0.8%)	15.2% (0.2%)
Gisette	**20.8%** (0.3%)	31.9% (0.2%)	32.3% (0.8%)	26.3% (0.5%)	36.9% (0.0%)	99.4% (0.3%)	46.2% (0.3%)
Adult	35.6% (0.2%)	37.4% (0.2%)	-	35.4% (0.2%)	40.4% (0.0%)	48.2% (0.2%)	41.7% (0.1%)

Table 4: Numbers of support points computed as a percentage of the total number of points. Numbers in parentheses are standard deviations over 100 iterations. Values which are significantly superior to that of other methods are typeset in bold.

an optimization over positive semidefinite kernel matrices which in turn reduces to a QCQP. Soon after, Bach et al. [1] proposed a block-norm regularization method based on *second order cone programming* (SOCP).

For efficiency, researchers started using optimization methods that alternate between updating the classifier parameters and the kernel weights. Many authors then explored the MKL landscape, including Rakotomamonjy et al. [21], Sonnenburg et al. [22], Xu et al. [25, 26]. However, as pointed out in [4], most of these methods do not compare favorably (both in accuracy as well as speed) even with the simple *uniform* heuristic. More recently, Moeller et al. [17] developed a multiplicative-weight-update based approach that has a much smaller memory footprint and scales far more effectively. Other *global* kernel learning methods include [5, 16, 18, 19, 23] and notably methods using the ℓ_p-norm [11, 12, 24].

References

[1] Francis R. Bach, Gert R. G. Lanckriet, and Michael I. Jordan. Multiple kernel learning, conic duality, and the SMO algorithm. In *ICML*, 2004.

[2] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[3] Radha Chitta, Rong Jin, Timothy C Havens, and Anil K Jain. Approximate kernel k-means: Solution to large scale kernel clustering. In *KDD*, pages 895–903. ACM, 2011.

[4] Corinna Cortes. Invited talk: Can learning kernels help performance? In *ICML*, Montreal, Canada, 2009.

[5] Corinna Cortes, Mehryar Mohri, and Afshin Ros-

tamizadeh. Learning non-linear combinations of kernels. In *NIPS*, Vancouver, Canada, 2009.

[6] Mehmet Gönen and Ethem Alpaydin. Localized multiple kernel learning. In *ICML*, pages 352–359, 2008.

[7] Mehmet Gönen and Ethem Alpaydin. Localized algorithms for multiple kernel learning. *Pattern Recognition*, 46(3):795–807, 2013.

[8] Yina Han and Guizhong Liu. Probability-confidence-kernel-based localized multiple kernel learning with norm. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 42(3):827–837, 2012.

[9] Ashesh Jain, S. V. N. Vishwanathan, and Manik Varma. SPG-GMKL: generalized multiple kernel learning with a million kernels. In *KDD*, pages 750–758, 2012.

[10] Raghvendra Kannao and Prithwijit Guha. Tv news commercials detection using success based locally weighted kernel combination. *arxiv*, 2015. URL `http://arxiv.org/abs/1507.01209v1`.

[11] Marius Kloft, Ulf Brefeld, Soeren Sonnenburg, Pavel Laskov, Klaus-Robert Müller, and Alexander Zien. Efficient and accurate lp-norm multiple kernel learning. In *NIPS*, Vancouver, Canada, 2009.

[12] Marius Kloft, Ulf Brefeld, Sören Sonnenburg, and Alexander Zien. l_p-norm multiple kernel learning. *JMLR*, 12:953–997, 2011.

[13] Gert R. G. Lanckriet, Nello Cristianini, Peter Bartlett, Laurent El Ghaoui, and Michael I. Jordan. Learning the kernel matrix with semidefinite programming. *JMLR*, 5:27–72, December 2004.

[14] Yunwen Lei, Alexander Binder, Ürün Dogan, and Marius Kloft. Localized multiple kernel learning - A convex approach. *CoRR*, abs/1506.04364, 2015. URL `http://arxiv.org/abs/1506.04364`.

[15] Xinwang Liu, Lei Wang, Jian Zhang, and Jianping Yin. Sample-adaptive multiple kernel learning. In *AAAI*, 2014.

[16] Charles A. Micchelli and Massimiliano Pontil. Learning the kernel function via regularization. *JMLR*, 6: 1099–1125, December 2005.

[17] John Moeller, Parasaran Raman, Suresh Venkatasubramanian, and Avishek Saha. A geometric algorithm for scalable multiple kernel learning. In *AISTATS*, pages 633–642, 2014.

[18] Cheng Soon Ong, Alexander J. Smola, and Robert C. Williamson. Learning the kernel with hyperkernels. *JMLR*, 6:1043–1071, 2005.

[19] Francesco Orabona and Jie Luo. Ultra-fast optimization algorithm for sparse multi kernel learning. In *ICML*, Bellevue, USA, 2011.

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

[21] Alain Rakotomamonjy, Francis Bach, Stéphane Canu, and Yves Grandvalet. More efficiency in multiple kernel learning. In *ICML*, Corvalis, USA, 2007.

[22] Sören Sonnenburg, Gunnar Rätsch, Christin Schäfer, and Bernhard Schölkopf. Large scale multiple kernel learning. *JMLR*, 7:1531–1565, December 2006.

[23] Manik Varma and Bodla Rakesh Babu. More generality in efficient multiple kernel learning. In *ICML*, Montreal, Canada, 2009.

[24] S. V. N. Vishwanathan, Zhaonan Sun, Nawanol Ampornpunt, and Manik Varma. Multiple kernel learning and the SMO algorithm. In *NIPS*, Vancouver, Canada, 2010.

[25] Zenglin Xu, Rong Jin, Irwin King, and Michael R. Lyu. An extended level method for efficient multiple kernel learning. In *NIPS*, Vancouver, Canada, 2008.

[26] Zenglin Xu, Rong Jin, Haiqin Yang, Irwin King, and Michael R. Lyu. Simple and efficient multiple kernel learning by group lasso. In *ICML*, Haifa, Israel, 2010.

[27] Jingjing Yang, Yuanning Li, Yonghong Tian, Lingyu Duan, and Wen Gao. Group-sensitive multiple kernel learning for object categorization. In *ICCV*, pages 436–443. IEEE, 2009.