Journal of Machine Learning Research 25 (2024) 1-40 Submitted 4/23; Revised 1/24; Published 6/24

Efficient Convex Algorithms for Universal Kernel Learning

Aleksandr Talitckii ATALITCK@ASU.EDU
Department of Mechanical and Aerospace Engineering
Arizona State University
Tempe, AZ 85281-1776, USA

Brendon Colbert BKCOLBE1@ASU.EDU
Department of Mechanical and Aerospace Engineering
Arizona State University
Tempe, AZ 85281-1776, USA

Matthew M. Peet MPEET@ASU.EDU
Department of Mechanical and Aerospace Engineering
Arizona State University
Tempe, AZ 85281-1776, USA

Editor: Ambuj Tewari

Abstract

The accuracy and complexity of machine learning algorithms based on kernel optimization are determined by the set of kernels over which they are able to optimize. An ideal set of kernels should: admit a linear parameterization (for tractability); be dense in the set of all kernels (for robustness); be universal (for accuracy). Recently, a framework was proposed for using positive matrices to parameterize a class of positive semi-separable kernels. Although this class can be shown to meet all three criteria, previous algorithms for optimization of such kernels were limited to classification and furthermore relied on computationally complex Semidefinite Programming (SDP) algorithms. In this paper, we pose the problem of learning semiseparable kernels as a minimax optimization problem and propose a SVD-QCQP primal-dual algorithm which dramatically reduces the computational complexity as compared with previous SDP-based approaches. Furthermore, we provide an efficient implementation of this algorithm for both classification and regression – an implementation which enables us to solve problems with 100 features and up to 30,000 datums. Finally, when applied to benchmark data, the algorithm demonstrates the potential for significant improvement in accuracy over typical (but non-convex) approaches such as Neural Nets and Random Forest with similar or better computation time.

Keywords: kernel functions, multiple kernel learning, semi-definite programming, supervised learning, universal kernels

1. Introduction

Kernels allow for a convex formulation of the nonlinear classification and regression problems – improving accuracy and robustness of data-based modeling. Specifically, every kernel defines a feature map of the data to a Reproducing Kernel Hilbert Space (RKHS) wherein a linear classification or regression problem may be solved. Furthermore, if the kernel is

universal, the RKHS will be infinite dimensional – implying that, e.g. classification data will always be linearly separable in the infinite-dimensional RKHS.

While any universal kernel (for example Gaussian or Laplacian kernel) will yield a linear separation in its associated RKHS, the robustness of that separation will be strongly influenced by the topology of the RKHS. For a poorly chosen kernel, the resulting fit will be sensitive to noise and hence the classifier or regressor may perform poorly on untrained data. However, for a well chosen kernel, the separation of data will be robust – yielding improved performance on untrained data. For example, when considering kernel selection in cancer diagnosis (See Wolberg et al., 1995) (a problem with significant variations in data collection mechanisms), lack of robustness of the classifier may result in incorrectly labelling a malignant tumour as benign. While rigorous numerical experimentation has been used to find suitable kernels for well-studied problems such as cancer classification (See Hussain et al., 2011), when the underlying data generating mechanism is new or speculative, kernel selection is itself an optimization problem known as learning the kernel. Specifically, Kernel Learning (KL) algorithms (such as those found in Xu et al., 2010; Sonnenburg et al., 2010; Yang et al., 2011) have been proposed to find the kernel, $k \in \mathcal{K}$ which optimizes an achievable metric such as the soft margin (for classification). However, the set of kernels, $k \in \mathcal{K}$, over which the algorithm can optimize strongly influences the performance and robustness of the resulting classifier or predictor.

To understand how the choice of kernel influences performance and robustness, several properties of positive kernels have been considered. For example, a *characteristic* kernel was defined in Fukumizu et al. (2007) to be a kernel whose associated integral operator is injective. Alternatively, a kernel, k, is *strictly positive definite* if the associated integral operator has trivial null-space – See, e.g. Steinwart and Christmann (2008). As stated in, e.g. Steinwart (2001), it has been observed that kernels with the characteristic and strictly positive definite properties are able to perform arbitrarily well on large sets of training data. Similar to characteristic and strictly positive definite kernels, perhaps the most well-known kernel property is that of c_0-universality, which implies that the associated RKHS is dense in $\mathcal{C}$. The relationship between these and other kernel properties (using, e.g. alternative norms) has been studied extensively in, e.g. Sriperumbudur et al. (2011). In particular, the universal property implies the kernel is both characteristic and strictly positive definite, while under certain conditions the converse also holds and all three properties are equivalent (Simon-Gabriel and Schölkopf, 2018). As a result of these studies, there is a consensus that in order to be effective on large data sets, a kernel should have the universal property.

While the universal property is now well-established as being a desirable property in any kernel, much less attention has been paid to the question of what are the desirable properties of a set of kernels. This question arises when we use kernel learning algorithms to find the optimal kernel in some parameterized set. The assumption, of course, is that for any given data generating process, there is some ideal kernel whose associated feature map maximizes separability of data generated by that underlying process. Clearly, then, when constructing a set of kernels to be used in kernel learning, we would like to ensure that this set contains the ideal kernel or at least a kernel whose associated feature map closely approximates the feature map of this ideal kernel. With this in mind, (Colbert and Peet, 2020) proposed three desirable properties of a set of kernels $\mathcal{K}$ - tractability, density, and universality. Specifically, $\mathcal{K}$ is said to be tractable if $\mathcal{K}$ is convex (or, preferably, a

linear variety) - implying the kernel learning problem is solvable in polynomial time (e.g. Rakotomamonjy et al., 2008; Jain et al., 2012; Lanckriet et al., 2004; Qiu and Lane, 2005). The set $\mathcal{K}$ has the density property if, for any $\epsilon > 0$ and any positive kernel, k^* there exists a $k \in \mathcal{K}$ where $\|k - k^*\| \leq \epsilon$. The density property implies that kernels from this set can approximate the feature map of the ideal kernel arbitrarily well (which then implies the resulting learned kernel will perform well on untrained data). Finally, the set $\mathcal{K}$ is said to have the universal property if any $k \in \mathcal{K}$ is universal.

Having defined desirably properties in a set of kernels, the question becomes how to parameterize such a set. While there are many ways of parameterizing sets of kernels (See Gönen and Alpaydın, 2011, for a survey), not all such parameterizations result in a convex kernel learning algorithm. Furthermore, at present, there is no tractable parameterization which is dense in the set of all possible kernels. To address this problem, in Colbert and Peet (2020), a general framework was proposed for using positive matrices and bases to parameterize families of positive kernels (as opposed to positive kernel matrices as in Lanckriet et al., 2004; Qiu and Lane, 2005; Ni et al., 2006). This framework allows one to define a basis of kernels for a class of integral operators and then to use SDP to find kernels which represent squares of such operators – implying that the resulting kernels define positive operators. In particular, Colbert and Peet (2020) proposed a set of basis functions which were then used to parameterize positive integral operators of the form (using one-dimension for simplicity)

$$(\mathcal{P}\mathbf{x})(s) := k_{1a}(s)\int_0^s k_{1b}(\theta)\mathbf{x}(\theta)d\theta + k_{2a}(s)\int_s^1 k_{2b}(\theta)\mathbf{x}(\theta)d\theta, \tag{1}$$

which correspond to what are known (in one-dimension) as semiseparable kernels – See, e.g. Gohberg et al. (2012). In n-dimensions, a kernel constructed using this particular parameterization was referred to as a Tessellated Kernel (TK) – indicative of their blockwise partition of the domain (a feature resulting from the semi-separable structure and reminiscent of the activation functions used in neural nets). It was further shown that the interior of this class of kernels was universal and the set of such kernels was dense in the set of all positive kernels. Using this positive matrix parameterization of the family of Tessellated Kernels, it was shown in Colbert and Peet (2020) that the associated SVM kernel learning algorithm could be posed as an SDP and that the solution to this SDP achieves superior Test Set Accuracy (TSA) when compared with a representative sample of existing classification algorithms (including non-convex kernel learning methods such as simple Neural Nets).

Unfortunately, however, although the TSA data reported in Colbert and Peet (2020) showed improvement over existing classification algorithms, this accuracy came at the cost of significant increases in computational complexity – a factor attributable to the high complexity of primal-dual interior point algorithms for solving SDP. Unlike Quadratic Programming (QP), which is used to solve the underlying SVM or Support Vector Regression (SVR) problem for a fixed kernel, the use of SDP, Quadratically Constrained Quadratic Programming (QCQP) and Second Order Cone Programming (SOCP) for kernel learning significantly limits the amount of training data which can be processed. Note that while this complexity issue has been partially addressed in the context of MKL (which considered an efficient reduction to QP complexity in Jain et al., 2012), such a reduction to QP has not

previously been proposed for SDP-based algorithms such as kernel matrix learning. The main goal of this paper, then, is to propose a new algorithm for optimizing over families of kernels parameterized by positive matrices, but without the use of SDP and its associated computational overhead.

Fundamentally, the algorithm proposed in this paper is based on a reformulation of the SDP defined in Colbert and Peet (2020) as a saddle-point optimization (See Section 3 and Section 4). This saddle-point formulation allows us to then decompose the optimization problem into primal and dual sub-problems, OPT_A and OPT_P (Section 5). Based on this decomposition, we propose a Franke-Wolfe type algorithm for solving the kernel learning problem - an approach based on the work in Rakotomamonjy et al. (2008) and Jain et al. (2012). Critically, we then show that the SDP in the subproblem OPT_P admits an analytic solution using the Singular Value Decomposition - implying a worst-case computational complexity of $O(n_P^3)$ where $n_P^2/2 + n_P$ is the number of parameters in the family of kernels, $\mathcal{K}$. In addition, we show that OPT_A is a convex QP and may be similarly solved with a complexity of $O(m^3)$ (or even $O(m^{2.3})$ for LibSVM implementation), where m is the number of data points. As a result, the resulting computational complexity of the proposed algorithm is dramatically reduced compared with the complexity of the SDP-based algorithms proposed in Colbert and Peet (2020) (with complexity (Brian and Young, 2007) $O(m^4)$ with respect to m and $O(n_P^6)$ with respect to n_P. Summarizing, the proposed algorithm does not require the use of SDP, QCQP or SOCP and, when applied to several standard test cases, has observed complexity which scales as $O(m^{2.3})$ or less.

In addition to the proposed algorithm, this paper also extends the convex kernel learning framework proposed in Colbert and Peet (2020) to the problem of kernel learning for regression. The kernel learning problem in regression has been studied for kernel matrix learning as in Lanckriet et al. (2004) and for MKL in e.g. Rakotomamonjy et al. (2008); Jain et al. (2012). However, the regression problem has not previously been considered using the generalized framework for kernel learning presented in Colbert and Peet (2020). In this paper, we provide such extension and demonstrate significant increases in performance, as measured by both computation time and Mean Square Error (MSE) when compared to other optimization-based kernel learning algorithms as well as when compared to more heuristic approaches such as Random Forest and deep learning.

2. Notation

We use $\mathbb{N}, \mathbb{R}$ to denote the natural and real numbers, respectively. We use $\mathbf{1}^n \in \mathbb{R}^n$ to denoted the vector of ones. For $x \in \mathbb{R}^n$ we use $\|x\|_p$ for the L_p-norm and use $x \geq 0$ to indicate the positive orthant – i.e. $x_i \geq 0$ for all i. For $x, y \in \mathbb{R}^n$, $x \odot y$ denotes elementwise multiplication. The space of $n \times n$-square real symmetric matrices is denoted $\mathbb{S}^n$ with $\mathbb{S}^n_+ \subset \mathbb{S}^n$ being the cone of positive semi-definite matrices. We use $\mathbb{S}^n_{++} \subset \mathbb{S}^n_+$ to denote the space of positive definite matrices. For a matrix $P \in S^n$, we use $P \succeq 0$ ($P \succ 0$) if P is a positive semi-definite matrix (positive definite matrix). For $A, B \in \mathbb{R}^{n \times n}$, $\langle A, B \rangle$ denotes the Frobenius matrix inner product. For a compact set $X \subset \mathbb{R}^n$, we denote $\mathcal{C}(X)$ to be the space of scalar continuous functions defined on X and equipped with the uniform norm $\|f\|_{\mathcal{C}} := \sup_{x \in X} |f(x)|$. For a differentiable function with a single argument, we use $\nabla f(x)$ to denote the gradient of function f at point x. For functions with two explicit arguments

where we do not need to specify a point of differentiation, the partial gradient of function $f(x,y)$ is denoted $\nabla_x f(x,y)$. In cases where we need to specify both the argument and the point of differentiation, we use $\nabla_x f(x,y)\big|_{x=x_0}$. Furthermore, if $x \in \mathbb{R}^{n\times n}$ and $f(x,y) \in \mathbb{R}$, we use $\nabla_x f(x,y) \in \mathbb{R}^{n\times n}$ to denote the matrix where $(\nabla_x f(x,y))_{ij} = \frac{\partial f(x,y)}{\partial x_{ij}}$. For a given positive definite kernel $k \in \mathcal{C}(X \times X)$, $\mathcal{H}_k$ denotes the associated Reproducing Kernel Hilbert Space (RKHS), where the subscript k is dropped if clear from context.

3. Kernel Sets and Kernel learning

Consider a generalized representation of the kernel learning problem, which encompasses both classification and regression where (using the representor theorem as in Schölkopf et al., 2001) the learned function is of the form $f_{\alpha,k}(z) = \sum_{i=1}^m \alpha_i k(x_i, z)$.

$$\min_{k\in\mathcal{K}} \min_{\substack{\alpha\in\mathbb{R}^m \\ b\in\mathbb{R}}} \|f_{\alpha,k}\|^2 + C\sum\nolimits_{i=1}^m l(f_{\alpha,k}, b)_{y_i,x_i}. \tag{2}$$

Here $\|f_{\alpha,k}\| = \sum_{i=1}^m \sum_{j=1}^m \alpha_i\alpha_j k(x_i,x_j)$ is the norm in the Reproducing Kernel Hilbert Space (RKHS) and $l(f_{\alpha,k},b)_{y_i,x_i}$ is the loss function defined for SVM binary classification and SVM regression as $l_c(f_{\alpha,k},b)_{y_i,x_i}$ and $l_r(f_{\alpha,k},b)_{y_i,x_i}$, respectively, where

$$\begin{aligned} l_c(f_{\alpha,k},b)_{y_i,x_i} &= \max\{0, 1 - y_i(f_{\alpha,k}(x_i) - b)\}, \\ l_r(f_{\alpha,k},b)_{y_i,x_i} &= \max\{0, |y_i - (f_{\alpha,k}(x_i) - b)| - \epsilon\}. \end{aligned}$$

The properties of the classifier/predictor, $f_{\alpha,k}$, resulting from Optimization Problem 2 will depend on the properties of the set $\mathcal{K}$, which is presumed to be a subset of the convex cone of all positive kernels. To understand how the choice of $\mathcal{K}$ influences the tractability of the optimization problem and the resulting fit, we consider three properties of the set, $\mathcal{K}$. These properties can be precisely defined as follows.

3.1 Tractability

We say a set of kernel functions, $\mathcal{K}$, is tractable if it can be represented using a countable basis.

Definition 1 *The set of kernels $\mathcal{K}$ is **tractable** if there exist a countable set $\{G_i(x,y)\}_{i=1}^\infty$ such that, for any $k \in \mathcal{K}$, there exists $n_k \in \mathbb{N}$ where $k(x,y) = \sum_{i=1}^{n_k} v_i G_i(x,y)$ for some $v \in \mathbb{R}^{n_k}$.*

Note the $G_i(x,y)$ need not be positive kernel functions. The tractable property is required for the associated kernel learning problem to be solvable in polynomial time.

3.2 Universality

Universal kernel functions always have positive definite (full rank) kernel matrices, implying that for arbitrary data $\{y_i, x_i\}_{i=1}^m$, there exists a function $f(z) = \sum_{i=1}^m \alpha_i k(x_i, z)$, such that $f(x_j) = y_j$ for all $j = 1, .., m$. Conversely, if a kernel is not universal, then there exists a data set $\{x_i, y_i\}_{i=1}^m$ such that for any $\alpha \in \mathbb{R}^m$, there exists some $j \in \{1, \cdots, m\}$ such

that $f(y_j) \neq \sum_{i=1}^{m} \alpha_i k(x_i, x_j)$. The universality property ensures that the classifier of an SVM designed using a universal kernel will become increasingly precise as the training data increases, whereas classifiers from a non-universal kernel have limited ability to fit data sets. (See Micchelli et al., 2006).

Definition 2 (Schölkopf et al. (2001)) *A kernel $k : X \times X \to \mathbb{R}$ is said to be **universal** on the compact metric space X if it is continuous and there exists an inner-product space $\mathcal{W}$ and feature map, $\Phi : X \to \mathcal{W}$ such that $k(x, y) = \langle \Phi(x), \Phi(y) \rangle_{\mathcal{W}}$ and where the unique Reproducing Kernel Hilbert Space (RKHS), $\mathcal{H} := \{f \ : \ f(x) = \langle v, \Phi(x) \rangle, \ v \in \mathcal{W}\}$ with associated norm $\|f\|_{\mathcal{H}} := \inf_v \{\|v\|_{\mathcal{W}} \ : \ f(x) = \langle v, \Phi(x) \rangle\}$ is dense in $\mathcal{C}(X) := \{f : X \to \mathbb{R} \ : f$ is continuous$\}$ where $\|f\|_{\mathcal{C}} := \sup_{x \in X} |f(x)|$.*

Recall that the universality property implies the strictly positive definite and characteristic properties on compact domains. The following definition extends the universal property to a set of kernels.

Definition 3 *A set of kernel functions $\mathcal{K}$ has the universal property if every kernel function $k \in \mathcal{K}$ is universal.*

3.3 Density

The third property of a kernel set, $\mathcal{K}$, is density which ensures that a kernel can be chosen from $\mathcal{K}$ with an associated feature map which optimizes fitting of the data in the associated feature space. This optimality of fit in the feature space may be interpreted differently for SVM and SVR. Specifically, considering SVM for classification, the kernel learning problem determines the kernel $k \in \mathcal{K}$ for which we may obtain the maximum separation in the kernel-associated feature space. According to Boehmke and Greenwell (2019), increasing this separation distance makes the resulting classifier more robust (generalizable). The density property, then, ensures that the resulting kernel learning algorithm will be maximally robust (generalizable) in the sense of separation distance. In the case of SVR, meanwhile, the kernel learning problem finds the kernel $k \in \mathcal{K}$ which permits the "flattest" (see Smola and Schölkopf, 2004) function in feature space. In this case, the density property ensures that the resulting kernel learning algorithm will be maximally robust (generalizable) in the sense of flatness.

Note that the density properties is distinct from the universality property. For instance consider a set containing a single Gaussian kernel function - which is clearly not ideal for kernel learning. The set containing a single Gaussian is tractable (it has only one element) and every member of the set is universal. However, it is not dense.

These arguments motivate the following definition of the pointwise density property.

Definition 4 *The set of kernels $\mathcal{K}$ is said to be **pointwise dense** if for any positive kernel, k^*, any set of data $\{x_i\}_{i=1}^m$, and any $\epsilon > 0$, there exists $k \in \mathcal{K}$ such that*

$$\|k(x_i, x_j) - k^*(x_i, x_j)\| \leq \epsilon.$$

4. A General Framework for Representation of Tractable Kernel Sets

Having defined three desirable properties of a set of kernels, we now consider a framework designed to facilitate the creation of sets of kernels which meet these criteria. This framework ensures tractability by providing a linear map from positive matrices to positive kernels. This map is defined by a set of bases, N. These bases themselves parameterize kernels the image of whose associated integral operators define the feature space. As we will show in Section 5, kernel learning over a set of kernels parameterized in this way can be performed efficiently using a combination of QP and the Singular Value Decomposition. Moreover, as we will show in Section 6, suitable choices of N will ensure that the set of kernels has the density and universality properties.

Lemma 5 *Let N be any bounded measurable function $N : Y \times X \to \mathbb{R}^{n_P}$ on compact X and Y. If we define*

$$\mathcal{K} := \left\{ k \;\middle|\; k(x,y) = \int_Y N(z,x)^T P N(z,y) dz, \quad P \succeq 0 \right\}, \tag{3}$$

then any $k \in \mathcal{K}$ is a positive kernel function and $\mathcal{K}$ is tractable.

Proof The proof is straightforward. Given a kernel, k, denote the associated integral operator by I_k so that

$$(I_k \phi)(s) = \int_X k(s,t)\phi(t) dt.$$

If $k \in \mathcal{K}$, it has the form of Eqn. (3) for some $P \succeq 0$. Now define $k_{\frac{1}{2}}(x,y) := P^{\frac{1}{2}} N(x,y)$. Then $I_k = I^*_{k_{\frac{1}{2}}} I_{k_{\frac{1}{2}}} \succeq 0$ where adjoint is defined with respect to the L_2 inner product. This establishes positivity of the kernel. Note that if $I_{k_{\frac{1}{2}}} \in \mathcal{A}$ for some *-algebra $\mathcal{A}$, then $I_k \in \mathcal{A}$.

For tractability, we note that for a given N, the map $P \mapsto k$ is linear. Specifically,

$$k(x,y) = \sum\nolimits_{i,j=1}^{n_P} P_{i,j} G_{i,j}(x,y),$$

where

$$G_{i,j}(x,y) = \int_Y N_i(z,x) N_j(z,y) dz, \tag{4}$$

and thus by Definition 1, $\mathcal{K}$ is tractable. ■

Note: Using the notation for integral operators in the proof of Lemma 5, we also note that for any $k \in \mathcal{K}$,

$$I_k = \sum\nolimits_{i,j=1}^{n_P} P_{i,j} I_{G_{i,j}} = \sum\nolimits_{i,j=1}^{n_P} P_{i,j} I^*_{N_i} I_{N_j}. \tag{5}$$

For convenience, we refer to a set of kernels defined as in Eqn. (3) as a Generalized Kernel Set, a kernel from such set as a Generalized Kernel, and the associated kernel learning problem in (2) as Generalized Kernel Learning (GKL) (3). This is to distinguish such kernels, sets and problems from Tessellated Kernel Learning, which arises from a particular choice of N in the parameterization of $\mathcal{K}$. This distinction is significant, as the algorithms in Sections 5 apply to the Generalized Kernel Learning problem, while the results in Section 6 only apply to the particular case of Tessellated Kernel Learning.

5. An Efficient Algorithm for Generalized Kernel Learning in Classification and Regression Problems

In this section, we assume a family of kernel functions, $\mathcal{K}$, has been parameterized as in (3), and formulate the kernel learning optimization problem for both classification and regression — representing this as a minimax saddle point problem. This formulation enables a decomposition into convex primal and dual sub-problems, $OPT\_A(P)$ and $OPT\_P(\alpha)$ with no duality gap. We then consider the Frank-Wolfe algorithm and show using Danskin's Theorem that the gradient step can be efficiently computed using the primal and dual sub-problems. Finally, we propose efficient algorithms for computing $OPT\_A(P)$ and $OPT\_P(\alpha)$: in the former case using an efficient Sequential Minimal Optimization (SMO) algorithm for convex QP and in the latter case, using an analytic solution based on the Singular Value Decomposition.

5.1 Primal-Dual Decomposition

For convenience, we define the feasible sets for the sub-problems as

$$\begin{aligned}\mathcal{X} &:= \{P \in \mathbb{S}^{n_P} \ : \ \text{trace}(P) = n_P, \ P \succeq 0\},\\ \mathcal{Y}_c &:= \{\alpha \in \mathbb{R}^m \ : \ \sum\nolimits_{i=1}^{m} \alpha_i y_i = 0, \ 0 \leq \alpha_i \leq C\},\\ \mathcal{Y}_r &:= \{\alpha \in \mathbb{R}^m \ : \ \sum\nolimits_{i=1}^{m} \alpha_i = 0, \ \alpha_i \in [-C, C]\},\end{aligned}$$

where m and C are as defined in Optimization Problem 2. In this section, we typically use the generic form $\mathcal{Y}_\star$ to refer to either $\mathcal{Y}_c$ or $\mathcal{Y}_r$ depending on whether the algorithm is being applied to the classification or regression problem. To specify the objective function we define $\lambda(\alpha, P)$ as

$$\lambda(\alpha, P) := -\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} \alpha_i \alpha_j \int_Y N(z, x_i)^T P N(z, y_j) dz, \tag{6}$$

where the bases, N, and domain, Y, are those used to specify the kernel set, $\mathcal{K}$, in Eqn. (3). Additionally, we define $\kappa_c(\alpha) := \sum_{i=1}^{m} \alpha_i$ and

$$\kappa_r(\alpha) := -\epsilon \sum\nolimits_{i=1}^{m} |\alpha_i| + \sum\nolimits_{i=1}^{m} y_i \alpha_i$$

where, again, we use $\kappa_\star = \kappa_c$ for classification and $\kappa_\star = \kappa_r$ for regression.

Using the formulation in, e.g. Lanckriet et al. (2004), it can be shown that if the family $\mathcal{K}$ is parameterized as in Eqn. (3), then the Generalized Kernel Learning optimization problem in Eqn. (2) can be recast as the following minimax saddle point optimization problem.

$$OPT_P := \min_{P \in \mathcal{X}} \max_{\alpha \in \mathcal{Y}_\star} \quad \lambda(e_\star \odot \alpha, P) + \kappa_\star(\alpha), \tag{7}$$

where $\odot$ indicates elementwise multiplication. For classification, $\mathcal{Y}_\star = \mathcal{Y}_c$, $\kappa_\star = \kappa_c$, and $e_\star = e_c := y$ (vector of labels). For regression, $\mathcal{Y}_\star = \mathcal{Y}_c$, $\kappa_\star = \kappa_r$, and $e_\star = e_r := \mathbf{1}_m$ (vector of ones).

Minimax Duality. To find the dual, OPT_D of the kernel learning optimization problem (OPT_P), we formulate two sub-problems:

$$OPT\_A(P) := \max_{\alpha \in \mathcal{Y}_\star} \lambda(e_\star \odot \alpha, P) + \kappa_\star(\alpha) \tag{8}$$

and

$$OPT\_P(\alpha) := \min_{P \in \mathcal{X}} \quad \lambda(e_\star \odot \alpha, P) + \kappa_\star(\alpha) := \min_{P \in \mathcal{X}} \langle D(\alpha), P \rangle , \tag{9}$$

where

$$D_{i,j}(\alpha) = \sum\nolimits_{k,l=1}^{m} (\alpha_k y_k) G_{i,j}(x_k, x_l)(\alpha_l y_l) \tag{10}$$

and the $G_{i,j}(x, y)$ are as defined in (4). Now, we have that

$$OPT_P = \min_{P \in \mathcal{X}} \; OPT\_A(P)$$

and its minmax dual is

$$OPT_D = \max_{\alpha \in \mathcal{Y}_\star} OPT\_P(\alpha) = \max_{\alpha \in \mathcal{Y}_\star} \min_{P \in \mathcal{X}} \quad \lambda(e_\star \odot \alpha, P) + \kappa_\star(\alpha).$$

The following lemma states that there is no duality gap between OPT_P and OPT_D - a property we will use in our termination criterion.

Lemma 6 *$OPT_P = OPT_D$. Furthermore, $\{\alpha^*, P^*\}$ solve OPT_P if and only if $OPT\_P(\alpha^*) = OPT\_A(P^*)$.*

Proof

For any minmax optimization problem with objective function ϕ, we have

$$d^* = \max_{\alpha \in \mathcal{Y}} \min_{P \in \mathcal{X}} \phi(P, \alpha) \leq \min_{P \in \mathcal{X}} \max_{\alpha \in \mathcal{Y}} \phi(P, \alpha) = p^*$$

and strong duality holds ($p^* - d^* = 0$) if $\mathcal{X}$ and $\mathcal{Y}$ are both convex and one is compact, $\phi(\cdot, \alpha)$ is convex for every $\alpha \in \mathcal{Y}$ and $\phi(P, \cdot)$ is concave for every $P \in \mathcal{X}$, and the function ϕ is continuous (See Fan, 1953). In our case, these conditions hold for both classification ($\phi(P, \alpha) = \lambda(\alpha \odot y, P) + \kappa_c(\alpha)$) and regression ($\phi(P, \alpha) = \lambda(\alpha, P) + \kappa_r(\alpha)$). Hence $OPT_P = OPT_D$. Furthermore, if $\{\alpha^*, P^*\}$ solve OPT_P then

$$OPT\_P(\alpha^*) = \max_{\alpha \in \mathcal{Y}} OPT\_P(\alpha) = \min_{P \in \mathcal{X}} OPT\_A(P) = OPT\_A(P^*).$$

Conversely, suppose $\alpha \in \mathcal{Y}$, $P \in \mathcal{X}$, then

$$\begin{aligned} OPT\_P(\alpha) \leq \max_{\alpha \in \mathcal{Y}} \; OPT\_P(\alpha) &= OPT\_P(\alpha^*) \\ &= OPT\_A(P^*) = \min_{P \in \mathcal{X}} OPT\_A(P) \leq OPT\_A(P). \end{aligned}$$

Hence if $OPT\_A(P) = OPT\_P(\alpha)$, then $OPT\_A(P) = OPT\_A(P^*) = OPT\_P(\alpha^*) = OPT\_P(\alpha)$ and hence P and α solve OPT_A and OPT_P, respectively. ■

Finally, we show that $OPT\_A(P)$ is convex with respect to P - a property we will use in Thm. 17.

`Initialize` P_0 as any point in $\mathcal{X}$.;
`Step 1:`
$S_k = \arg\min_{S\in\mathcal{X}} \langle \nabla f(P_k), S\rangle$
`Step 2:`
$\gamma_k = \arg\min_{\gamma\in[0,1]} f(P_k + \gamma(S_k - P_k))$
`Step 3:`
$P_{k+1} = P_k + \gamma_k (S_k - P_k), k = k+1,$
Return to step 1 unless stopping criteria is met.

Algorithm 1: The Frank-Wolfe Algorithm for Matrices.

`Initialize` $P_0 = I$, $k = 0$, $\alpha_0 = OPT\_A(P_0)$;
`Step 1a:` $\alpha_k = \arg OPT\_A(P_k)$
`Step 1b:` $S_k = \arg OPT\_P(\alpha_k)$
`Step 2:`
$\gamma_k = \arg\min_{\gamma\in[0,1]} OPT\_A(P_k + \gamma(S_k - P_k))$
`Step 3:`
$P_{k+1} = P_k + \gamma_k(S_k - P_k)$, $k = k+1$
Return to step 1 unless
$OPT\_P(\alpha_k) - OPT\_A(P_k) < \epsilon$.

Algorithm 2: Proposed FW Algorithm for GKL.

Lemma 7 *Let $OPT\_A(P)$ be as defined in (8). Then, the function $OPT\_A(P)$ is convex with respect to P.*

Proof

It is a well known property of convex functions (e.g. Bertsekas (2016)) that if $\mathcal{Y}_*$ is a compact set and $\phi(\alpha, P)$ is convex with respect to $P \in \mathcal{X}$ for every $\alpha \in \mathcal{Y}_*$, then if $g(P) = \max_{\alpha\in\mathcal{Y}_*} \phi(\alpha, P)$ exists for every $P \in \mathcal{X}$, then $g(P)$ is convex with respect to $P \in \mathcal{X}$. These conditions are readily verified using the definition of $OPT\_A(P)$ for both classification and regression, where $\phi(\alpha, P) = \lambda(e_\star \odot \alpha, P) + \kappa_\star(\alpha)$ and $g(P) = OPT\_A(P)$. ■

5.2 Primal-Dual Frank-Wolfe Algorithm

For an optimization problem of the form

$$\min_{S\in\mathcal{X}} f(S),$$

where $\mathcal{X}$ is a convex subset of matrices and $\langle\cdot,\cdot\rangle$ is the Frobenius matrix inner product, the Frank-Wolfe (FW) algorithm (See, e.g. Frank and Wolfe, 1956) may be defined as in Algorithm 1.

In our case, we have $f(Q) = OPT\_A(Q)$ so that

$$OPT_P = \min_{P\in\mathcal{X}} OPT\_A(P).$$

Implementation of the FW algorithm requires us to compute $\nabla OPT\_A(P_k)$ at each iteration. To address this issue, we propose a way to efficiently compute the sub-problems OPT_A and OPT_P, as shown in Subsections 5.3 and 5.4. Furthermore, in Lemma 9, we will show that these sub-problems can be used to efficiently compute the gradient $\nabla OPT\_A(P_k)$ - allowing for an efficient implementation of the FW algorithm. Lemma 9 uses a variation of Danskin's theorem (generalized in Bertsekas, 2016).

Proposition 8 (Danskin's Theorem) *Let $\mathcal{Y} \subset \mathbb{R}^m$ be a compact set, and let $\phi : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ be continuous such that $\phi(\cdot, \alpha) : \mathcal{X} \to \mathbb{R}$ is convex for each $\alpha \in \mathcal{Y}$. Then for $P \in \mathcal{X}$, if*

$$\mathcal{Y}_0(P) = \left\{ \bar{\alpha} \in \mathcal{Y} \;\middle|\; \phi(P, \bar{\alpha}) = \max_{\alpha \in \mathcal{Y}} \phi(P, \alpha) \right\}$$

consists of only one unique point, $\bar{\alpha}$, and $\phi(\cdot, \bar{\alpha})$ is differentiable at P then the function $f(Q) = \max_{\alpha \in \mathcal{Y}} \phi(Q, \alpha)$ is differentiable at P and

$$\nabla f(P) = \nabla_Q \phi(Q, \bar{\alpha})\big|_{Q=P}.$$

Prop. 8 can now be used to prove the following.

Lemma 9 *If OPT_A and OPT_P are as defined in Eqns.* (8) *and* (9)*, then for any $P_k \succ 0$, we have*

$$\arg\min_{S \in \mathcal{X}} \langle \nabla OPT\_A(P_k), S \rangle = \arg OPT\_P(\arg OPT\_A(P_k)).$$

Proof For simplicity, we utilize the definition of $D(\alpha)$ which will be given in Eqn. (10) so that $\lambda(e_\star \odot \alpha, P) := \langle D(\alpha), P \rangle$. Now, since $\lambda(\alpha, P)$ is strictly concave in α, for any $P_k \succ 0$, $OPT\_A(P_k)$ has a unique solution and hence we have by Danskin's Theorem that

$$\begin{aligned}
&\arg\min_{S \in \mathcal{X}} \left\langle \nabla OPT_A(P_k), S \right\rangle \\
&= \arg\min_{S \in \mathcal{X}} \left\langle \nabla_Q \left[\max_{\alpha \in \mathcal{Y}_\star} \left(\langle D(\alpha), Q \rangle + \kappa_\star(\alpha) \right) \right]_{Q=P_k}, S \right\rangle \\
&= \arg\min_{S \in \mathcal{X}} \left\langle \nabla_Q \left[\langle D(\bar{\alpha}), Q \rangle + \kappa_\star(\bar{\alpha}) \right]_{Q=P_k}, S \right\rangle
\end{aligned}$$

where $\bar{\alpha} = \arg OPT\_A(P_k)$. Hence,

$$\begin{aligned}
&\arg\min_{S \in \mathcal{X}} \left\langle \nabla_Q \left[\langle D(\bar{\alpha}), Q \rangle + \kappa_\star(\bar{\alpha}) \right]_{Q=P_k}, S \right\rangle = \arg\min_{S \in \mathcal{X}} \left\langle \nabla_Q \left[\langle D(\bar{\alpha}), Q \rangle \right]_{Q=P_k}, S \right\rangle \\
&= \arg\min_{S \in \mathcal{X}} \left\langle D(\bar{\alpha}), S \right\rangle = \arg OPT_P(\bar{\alpha}) = \arg OPT_P(\arg OPT_A(P_k)).
\end{aligned}$$

■

We now propose an efficient implementation of the FW GKL algorithm, as defined in Algorithm 2, based on efficient algorithms for computing OPT_A and OPT_P as will be defined in Subsections 5.3 and 5.4.

In the following theorem, we use convergence properties of the FW GKL algorithm to show that Algorithm 2 has worst-case $O\left(\frac{1}{k}\right)$ convergence. Note that when higher accuracy is required, we may utilize recently proposed primal-dual algorithms for $O\left(\frac{1}{k^2}\right)$ convergence, as will be discussed in Section 5.5.

Theorem 10 *Algorithm 2 returns iterates P_k and α_k such that, $|\lambda(\alpha_k, P_k) + \kappa_\star(\alpha_k) - OPT_P| < O(\frac{1}{k})$.*

Proof If we define $f = OPT\_A$, then Lemma 9 shows that f is differentiable and, if the P_k satisfy Algorithm 2, that the P_k also satisfy Algorithm 1. In addition, Lemma 7 shows that $f(Q) = OPT\_A(Q)$ is convex in Q. It has been shown in, e.g. Jaggi (2013), that if

$\mathcal{X}$ is convex and compact and $f(Q)$ is convex and differentiable on $Q \in \mathcal{X}$, then the FW Algorithm produces iterates P_k, such that, $f(P_k) - f(P^*) < O(\frac{1}{k})$ where

$$f(P^*) = \min_{P \in \mathcal{X}} f(P) = \min_{P \in \mathcal{X}} OPT\_A(P) = OPT_P.$$

Finally, we note that

$$\begin{aligned}\lambda(\alpha_k, P_k) + \kappa_\star(\alpha_k) =& \lambda(\arg OPT\_A(P_k), P_k) + \kappa_\star(\arg OPT\_A(P_k)) \\ &= \max_{\alpha \in \mathcal{Y}_\star} \lambda(\alpha, P_k) + \kappa_\star(\alpha) = OPT\_A(P_k) = f(P_k),\end{aligned}$$

which completes the proof. ■

In the following subsections, we provide efficient algorithms for computing the sub-problems OPT_A and OPT_P.

5.3 Step 1, Part A: Solving $OPT\_A(P)$

For a given $P \succeq 0$, $OPT\_A(P)$ is a convex Quadratic Program (QP). General purpose QP solvers have a worst-case complexity which scales as $O(m^3)$ (See Ye and Tse (1989)) where, when applied to OPT_A, m becomes the number of samples. This computational complexity may be improved, however, by noting that OPT_A is compatible with the representation defined in Chang and Lin (2011) for QPs derived from support vector machine problems. In this case, the algorithm in LibSVM reduces the computational burden somewhat. This improved performance is illustrated in Figure 2 where we observe the achieved complexity scales as $O(m^{2.3})$. Note that for the 2-step algorithm proposed in this manuscript, solving the QP in $OPT\_A(P)$ is significantly slower that solving the Singular Value Decomposition required for $OPT\_P(\alpha)$, which is defined in the following subsection.

5.4 Step 1, Part B: Solving $OPT\_P(\alpha)$

For a given α, $OPT\_P(\alpha)$ is an SDP. Fortunately, however, this SDP is structured so as to admit an analytic solution using the Singular Value Decomposition (SVD). To solve $OPT\_P(\alpha)$ we minimize $\lambda(e_\star \odot \alpha, P)$ from Eq. (6) which is linear in P and can be formulated as

$$OPT\_P(\alpha) := \min_{\substack{P \in \mathbb{S}^{n_P} \\ \text{trace}(P) = n_P \\ P \succeq 0}} \lambda(e_\star \odot \alpha, P) := \min_{\substack{P \in \mathbb{S}^{n_P} \\ \text{trace}(P) = n_P \\ P \succeq 0}} \langle D(\alpha), P \rangle,$$

where

$$D_{i,j}(\alpha) = \sum_{k,l=1}^{m} (\alpha_k y_k) G_{i,j}(x_k, x_l)(\alpha_l y_l)$$

and the $G_{i,j}(x, y)$ are as defined in (4).

The following theorem gives an analytic solution for OPT_P using the SVD.

Theorem 11 *For a given α, denote $D_\alpha := D(\alpha) \in \mathbb{S}^{n_P}$ where $D(\alpha)$ is as defined in Eqn. (10) and let $D_\alpha = V \Sigma V^T$ be its SVD. Let v be the right singular vector corresponding to the minimum singular value of D_α. Then $P^* = n_P v v^T$ solves $OPT\_P(\alpha)$.*

Proof Recall $OPT\_P(\alpha)$ has the form

$$\min_{P \in \mathbb{S}^{n_P}} \langle D_\alpha, P \rangle \quad \text{s.t. } P \succeq 0,\ \text{trace}(P) = n_P.$$

Denote the minimum singular value of D_α as $\sigma_{\min}(D_\alpha)$. Then for any feasible $P \in \mathcal{X}$, by Fang et al. (1994) we have

$$\langle D_\alpha, P \rangle \geq \sigma_{\min}(D_\alpha)\text{trace}(P) = \sigma_{\min}(D_\alpha) n_P.$$

Now consider $P = n_P v v^T \in \mathbb{S}^{n_P}$. P is feasible since $P \succeq 0$, and $\text{trace}(P) = n_P$. Furthermore,

$$\begin{aligned} \langle D_\alpha, P \rangle &= n_P \,\text{trace}(V \Sigma V^T v v^T) = n_P \,\text{trace}(v^T V \Sigma V^T v) \\ &= n_P \,\sigma_{\min}(D_\alpha) \end{aligned}$$

as desired. ■

Note that the size, n_P, of D_α in $OPT\_P(\alpha)$ scales with the number of features, but not the number of samples (m). As a result, we observe that the OPT_P step of Algorithm 2 is significantly faster than the OPT_A step for large data sets.

5.5 An Accelerated Algorithm for $O\left(\frac{1}{k^2}\right)$ Convergence

The Frank-Wolfe (FW) GKL algorithm proposed in Section 5 has provable sublinear convergence $O\left(\frac{1}{k}\right)$. While we observe in practice that achieved convergence rates of FW initially exceed this provable bound, when the number of iterations is large (e.g. when duality gap $< 10^{-5}$ is desired), the FW algorithm tends to return to sublinear convergence (See Fig. 1). While $O\left(\frac{1}{k}\right)$ convergence is adequate for most problems, occasionally we may require highly accurate solutions. In such cases, we may look for saddle-point algorithms with $O\left(\frac{1}{k^2}\right)$ convergence, so as to reduce the overall computation time. One such Accelerated Primal Dual (APD) algorithm was recently proposed in Hamedani and Aybat (2021) and the QP/SVD approach proposed in Subsections 5.3 and 5.4 can also be used to implement this algorithm. Specifically, we find that when the QP/SVD approach is applied to APD algorithm, the result is reduced convergence rates for the first few iterations, but improved convergence rates at subsequent iterations. Furthermore, while the per-iteration computational complexity increases with the use of APD, the scaling with respect to feature and sample size remains essentially the same – See Fig. 1. However, because most numerical tests in this paper were performed to a primal-dual gap of 10^{-5}, the APD implementation was not used to produce the results in Section 8 and hence details are not included. Please see Appendix B for additional details.

6. Tessellated Kernels: Tractable, Dense and Universal

In this section, we examine the family of kernels defined as in (12) for a particular choice of N. Specifically, let $Y = X = \mathbb{R}^n$ and $Z_d : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^{\binom{d+2n+1}{d}}$ be the vector of monomials of degree d or less and define the indicator function for the positive orthant, $\mathbf{I} : \mathbb{R}^n \to \mathbb{R}$ as follows.

$$\mathbf{I}(z) = \begin{cases} 1 & z \geq 0 \\ 0 & \text{otherwise,} \end{cases}$$

where recall $z \geq 0$ if $z_i \geq 0$ for all i. We now specify the N which defines $\mathcal{K}$ in (3) as $N_T^d : Y \times X \to \mathbb{R}^{2\binom{d+2n+1}{d}}$ for $d \in \mathbb{N}$ as

$$N_T^d(z,x) = \begin{bmatrix} Z_d(z,x)\mathbf{I}(z-x) \\ Z_d(z,x)\left(1 - \mathbf{I}(z-x)\right) \end{bmatrix}. \tag{11}$$

This assignment $N \to N_T^d$ defines an associated families of kernel functions, denoted $\mathcal{K}_T^d$ where

$$\mathcal{K}_T^d := \left\{ k : k(x,y) = \int_Y N_T^d(z,x)^T P N_T^d(z,y) dz, \;\; P \succeq 0, \right\}. \tag{12}$$

The union of such families is denoted $\mathcal{K}_T := \{k \; : \; k \in \mathcal{K}_T^d, \; d \in \mathbb{N}\}$.

Note that since $Z_d(x,y)$ consists of monomials, it is separable and hence has the form $Z_d(x,y) = Z_{d,a}(x)Z_{d,b}(y)$. This implies that $I_{P^{\frac{1}{2}}N_T^d}$ (as defined in Eqn. (5)) has the form given in Eqn. (1). It can be shown that this class of operators forms a *-algebra and hence any kernel in $\mathcal{K}_T$ is semiseparable (extending this term to cover n-dimensions) — implying that for any $k \in \mathcal{K}_T^d$, I_k has the form in Eqn. (1).

In Colbert and Peet (2020), this class of kernels was termed "Tessellated" in the sense that each datapoint defines a vertex which bisects each dimension of the domain of the resulting classifier/predictor - resulting in a tessellated partition of the feature space.

6.1 $\mathcal{K}_T$ is Tractable

The class of Tessellated kernels is prima facie in the form of Eqn. (3) in Lemma 5 and hence is tractable. However, we will expand on this result by specifying the basis for the set of Tessellated kernels, which will then be used in combination with the results of Section 5 to construct an efficient algorithm for kernel learning using Tessellated kernels.

Corollary 12 *Suppose that $a < b \in \mathbb{R}^n$, and $d \in \mathbb{N}$. We define the finite set $D_d := \{(\delta,\lambda) \in \mathbb{N}^n \times \mathbb{N}^n : \|(\delta,\lambda)\|_1 \leq d\}$. Let $\{[\delta_i,\gamma_i]\}_{i=1}^{\frac{1}{2}n_p} \subseteq D_d$ be some ordering of D_d where $n_p = 2\binom{d+2n+1}{d}$. Define $Z_d(x,z)_j = x^{\delta_j}z^{\gamma_j}$ where $x^{\delta_j}z^{\gamma_j} := \prod_{i=1}^n x_i^{\delta_{j,i}} z_i^{\gamma_{j,i}}$. Now let k be as defined in Eqn. (3) for some $P \succeq 0$ and where N is as defined in Eqn. (11). Then we have*

$$k(x,y) = \sum\nolimits_{i,j=1}^{n_P} P_{i,j} G_{i,j}(x,y),$$

where

$$G_{i,j}(x,y) := \begin{cases} g_{i,j}(x,y) & \text{if } i \leq \frac{n_P}{2}, j \leq \frac{n_P}{2} \\ t_{i,j}(x,y) & \text{if } i \leq \frac{n_P}{2}, j > \frac{n_P}{2} \\ t_{i,j}(y,x) & \text{if } i > \frac{n_P}{2}, j \leq \frac{n_P}{2} \\ h_{i,j}(x,y) & \text{if } i > \frac{n_P}{2}, j > \frac{n_P}{2} \end{cases}$$

and where $g_{i,j}, t_{i,j}, h_{i,j} : \mathbb{R}^{2n} \to \mathbb{R}$ are defined as

$$\begin{aligned} g_{i,j}(x,y) &:= x^{\delta_i} y^{\delta_j} T(p^*(x,y), b, \gamma_i + \gamma_j + \mathbf{1}), \\ t_{i,j}(x,y) &:= x^{\delta_i} y^{\delta_j} T(x, b, \gamma_i + \gamma_j + \mathbf{1}) - g_{i,j}(x,y), \\ h_{i,j}(x,y) &:= x^{\delta_i} y^{\delta_j} T(a, b, \gamma_i + \gamma_j + \mathbf{1}) - g_{i,j}(x,y) - t_{i,j}(x,y) - t_{i,j}(y,x), \end{aligned} \tag{13}$$

where $\mathbf{1} \in \mathbb{N}^n$ *is the vector of ones,* $p^* : \mathbb{R}^{2n} \to \mathbb{R}^n$ *is defined elementwise as* $p^*(x,y)_i = \max\{x_i, y_i\}$*, and* $T : \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{N}^n \to \mathbb{R}$ *is defined as*

$$T(x,y,\zeta) = \prod_{j=1}^{n} \left(\frac{y_j^{\zeta_j}}{\zeta_j} - \frac{x_j^{\zeta_j}}{\zeta_j} \right).$$

The proof of Corollary 12 can be found in Colbert and Peet (2020).

6.2 $\mathcal{K}_T$ is Dense

As per the following Lemma from Colbert and Peet (2020), the set of Tessellated kernels satisfies the pointwise density property.

Theorem 13 *For any positive semidefinite kernel matrix* K^* *and any finite set* $\{x_i\}_{i=1}^m$*, there exists a* $d \in \mathbb{N}$ *and* $k \in \mathcal{K}_T^d$ *such that* $K^*_{i,j} = k(x_i, x_j)$ *for all* i, j*.*

6.3 $\mathcal{K}_T$ is Universal

To show that $\mathcal{K}_T$ is universal, we first show that the auxiliary kernel $k(x,y) = \int_Y \mathbf{I}(z - x)\mathbf{I}(z - y)dz$ is universal.

Lemma 14 *For any* $a, b, \delta \in \mathbb{R}^n$ *with* $a < b$ *and* $\delta > 0$*, let* $Y = [a - \delta, b + \delta]$ *and* $X = [a, b]$*. Then the kernel*

$$k(x,y) = \int_Y \mathbf{I}(z-x)\mathbf{I}(z-y)dz = \int_{\substack{x \le z \\ y \le z \\ z \le b+\delta}} 1dz = \prod_{i=1}^{n} (b_i + \delta_i - \max\{x_i, y_i\}) = \prod_{i=1}^{n} k_i(x_i, y_i)$$

is universal where $k_i(x,y) := b_i + \delta_i - \max\{x_i, y_i\}$*.*

For brevity, consider the following proof summary. A complete proof is provided in Appendix A.

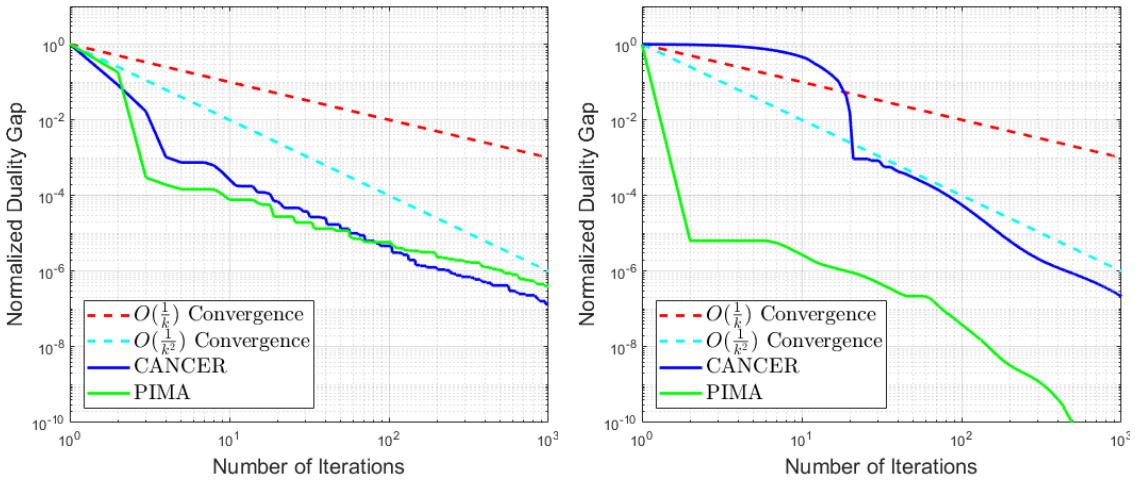
Sketch of proof: To show the universality of a kernel, k, one must prove that k is continuous and the corresponding RKHS is dense. Now let us consider each k_i in the product $k(x,y) = \prod_{i=1}^n k_i(x_i, y_i)$. As shown in Colbert and Peet (2020), each kernel, k_i, is continuous, and every triangle function is in the corresponding RKHS, $\mathcal{H}_{k_i}$. Also, since $k(x, b_i) = \delta_i$, the constant function is also in $\mathcal{H}_{k_i}$. Consequently, we conclude that the RKHS associated with k_i contains a Schauder basis for $\mathcal{C}([a_i, b_i])$ – implying that the kernel k_i is universal. Since k is the product of n universal kernels, we may now show that k is also universal.

The following theorem extends Lemma 14 and shows that if $k \in \mathcal{K}_T$ is defined by a positive definite parameter, $P \succ 0$, then k is universal.

Theorem 15 *Suppose* $k : X \times X \to \mathbb{R}$ *is as defined in Eqn.* (3) *for some* $P \succ 0$*,* $d \in \mathbb{N}$ *and* N *as defined in Eqn.* (11)*, then* k *is a universal kernel for* $Y = [a - \delta, b + \delta]$*,* $X = [a, b]$ *and* $\delta > 0$

Proof Since $P \succ 0$, then there exists $\epsilon > 0$ such that

$$\hat{P} = P - \epsilon \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 \end{bmatrix} \geq 0.$$



(a) The duality gap for 1000 iterations of FW Algorithm 2, applied to two different classification data sets.

(b) The duality gap for 1000 iterations of APD Algorithm, applied to two different classification data sets.

Figure 1: Convergence rates of the Franke-Wolfe algorithm 2 and the alternative APD algorithm described in Subsection 5.5. In (a) we plot the gap between $OPT\_A(P_k)$ and $OPT\_P(\alpha_k)$ of the Franke-Wolfe Algorithm 2 vs. iteration number; in (b) we again plot the gap between $OPT\_A(P_k)$ and $OPT\_P(\alpha_k)$ vs. iteration number for the APD Algorithm and in (c) we plot the boosted algorithm. Both demonstrate sublinear convergence, but with enhanced performance for the hybrid algorithm.

Next

$$k(x,y) = \int_Y N_T^d(z,x)^T P N_T^d(z,y)dz = \int_Y N_T^d(z,x)^T \hat{P} N_T^d(z,y)dz + \epsilon \int_Y \mathbf{I}(z-x)\mathbf{I}(z-y)dz$$
$$= \hat{k}(x,y) + k_1(x,y)$$

where $k_1(x,y) = \delta \int_Y \mathbf{I}(z-x)\mathbf{I}(z-y)dz$.

It was shown in Lemma 14, that $k_1(x,y)$ is universal. Since k is a sum of two positive kernels and one of them is universal, then according to Wang et al. (2013) and Borgwardt et al. (2006) we have that $k : X \times X \to \mathbb{R}$ is universal for $Y = [a-\delta, b+\delta]$ and $X = [a,b]$ ■

This theorem implies that even $\mathcal{K}_T^0$ has the universal property.

7. Numerical Convergence and Scalability

The computational complexity of the algorithms proposed in this paper will depend both on the computational complexity required to perform each iteration as well as the number of iterations required to achieve a desired level of accuracy. In this section, we use numerical tests to determine the observed convergence rate of Algorithm 2 and the observed com-

putational complexity of each iteration when applied to several commonly used machine learning data sets.

7.1 Convergence Properties

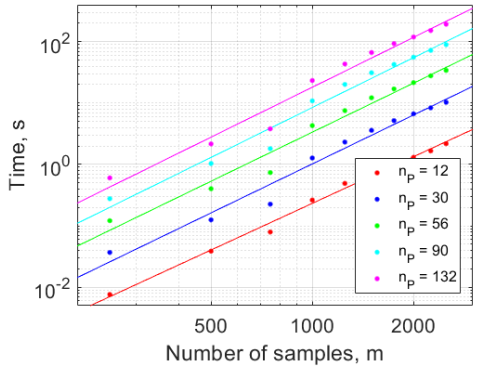
In this subsection, we briefly consider the estimated number of iterations of the FW algorithm 2 required to achieve a given level of accuracy as measured by the gap between the primal and dual solutions. Primal-Dual algorithms such as the proposed FW method typically achieve high rates of convergence. While the number of iterations required to achieve a given level of accuracy does not typically change with the size or type of the problem, if the number of iterations required to achieve convergence is excessive, this will have a significant impact on the performance of the algorithm. In Section 5, we established that the proposed algorithm has worst-case $O\left(\frac{1}{k}\right)$ convergence and proposed an alternative ADP approach with provable $O\left(\frac{1}{k^2}\right)$ performance. However, provable bounds on convergence rates are often conservative and in this subsection we examine the observed convergence rates as applied to several test cases in both the classification and regression frameworks.

First, to study the convergence properties of the FW Algorithm 2, in Figure 1(a), we plot the gap between $OPT\_A(P_k)$ and $OPT\_P(\alpha_k)$ as a function of iteration number for the CANCER and PIMA data sets. The use of the $OPT\_A(P_k)$-$OPT\_P(\alpha_k)$ gap for an error metric is a slight improvement over typical implementations of the FW error metric – which uses a predicted *bound* on the primal-dual gap. However, in practice, we find that the observed convergence rate does not change significantly depending on which metric is used. For reference, Fig. 1 also includes a plot of theoretical worst-case $O\left(\frac{1}{k}\right)$ and $O\left(\frac{1}{k^2}\right)$ convergence. As is common in primal-dual algorithms, we observe that the achieved convergence rates significantly exceed the provable sublinear bound, with this difference being especially noticeable for the first few iterations. These results indicate that for a moderate level of accuracy, the performance of the FW algorithm is adequate – especially combined with the low per-iteration complexity described in the following subsection. However, benefits of the FW algorithm are more limited at high levels of accuracy. Thus, in Fig. 1 (b), we find convergence rates for the suggested APD algorithm mentioned in Subsection 5.5. Unlike the FW algorithm, convergence rates for the first few iterations of APD are not uniformly high. This observation, combined with a slightly higher per-iteration complexity of APD is the reason for our focus on the FW implementation. However, as noted in Appendix B.4, these algorithms can be combined by switching to the APD algorithm after a fixed number of iterations – an approach which offers superior convergence when desired accuracy is high.

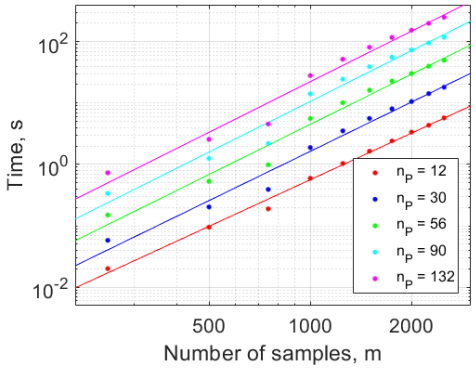
7.2 Computational Complexity

In this subsection, we consider the computational complexity of a single iteration of the proposed FW algorithm 2. Specifically, we examine how the expected complexity of an iteration of each subproblem scales with number of samples and hyperparameters. We then examine how this performance compares with observed complexity in several test cases for both the classification and regression frameworks. Finally, these numerical results are compared with the alternative APD algorithm mentioned in Subsection 5.5.

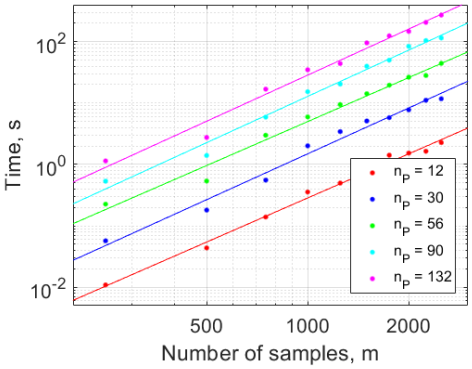
The computational complexity of the proposed FW algorithm depends on the size of the data set m (number of samples) and the hyper-parameter $n_P = 2\binom{d+2n+1}{d}$, where d is



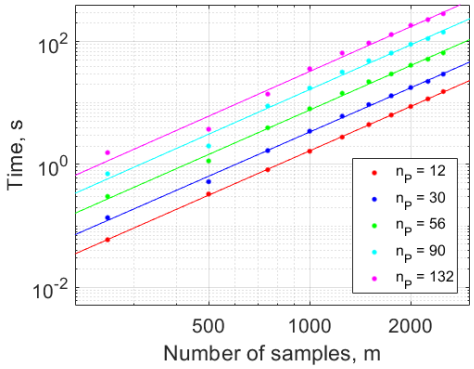
(a) Iteration complexity for the FW algorithm applied to classification



(b) Iteration complexity for the APD algorithm applied to classification



(c) Iteration complexity for the FW algorithm applied to regression



(d) Iteration complexity for the APD algorithm applied to regression

Figure 2: Per-iteration complexity of the proposed FW algorithm 2 and the alternative APD algorithm described in Subsection 5.5. In (a) and (c) we find log-log plots of iteration complexity of the Franke Wolfe (FW) TKL classification and regression algorithms, respectively, as a function of m for several values of n_P. Here m is number of samples and n_P^2 is the number of parameters in $\mathcal{K}$, so that $P \in \mathbb{S}^{n_P}$. In (b) and (d) we find log-log plots of iteration complexity of the Accelerated Primal Dual (APD) for classification and regression, respectively as a function of m for several values of n_P. In both cases, best linear fit is included for reference.

a degree of monomials and n is the number of features. As discussed in Section 5, each iteration of proposed FW algorithm consists of three steps. The first step (`Step 1a`) of the FW algorithm is the optimization problem $OPT\_A(P)$ – an SVC or SVR learning problem. For this step, we use a LibSVM implementation for which expected complexity scales as $O(m^{2.3})$. In this step, then kernel defined by matrix P is fixed and hence this step does not depend on n_P. We note, however, that observed complexity for this step is a function of the rank of P – as shown in Appendix C. Specifically, we find that observed complexity of LibSVM is significantly reduced when the matrix P is near optimal in both classification and regression problems.

Method	Liver	Cancer	Heart	Pima
SDP	95.75 ± 2.68	636.17 ± 25.43	221.67 ± 29.63	1211.66 ± 27.01
Algorithm 2	0.12 ± 0.03	0.41 ± 0.23	4.71 ± 1.15	0.80 ± 0.36

Table 1: The mean computation time (in seconds), along with standard deviation, for 30 trials comparing the SDP algorithm in Colbert and Peet (2020) and Algorithm 2. All tests are run on an Intel i7-5960X CPU at 3.00 GHz with 128 Gb of RAM.

The second step (`Step 1b`) entails computing the minimum singular vectors of the D matrix (Eq,. (10)) of dimension $n_P \times n_P$ – for which we use the standard Singular Value Decomposition (SVD). Before performing an SVD, we must calculate D – which requires $O(m^2)$ operations for each of the n_P^2 elements of the matrix D. We note, however, that the cost of these calculations is relatively minor compared with the overall complexity of the SVD and SVC/SVR from `Step 1a`. For solving the SVD, we use a LAPACK implementation which has worst-case complexity which scales as $O(n_P^3)$ – and which does not depend on m (the number of samples). The last step, `Step 2` is a primitive line search algorithm, where for each iteration, we evaluate a fixed number of n_γ candidate step sizes (γ). Each candidate step size requires solving a QP problem ($O(m^{2.3})$), leading to worst-case complexity scaling as $O(m^{2.3})$. We conclude that the expected complexity of the proposed algorithm scales as $O(m^{2.3} + m^2 n_P^2 + n_P^3)$.

To compare expected iteration complexity with observed complexity, we next test the proposed FW algorithm on several test cases and compare these results with observed complexity of the alternative APD algorithm mentioned in Subsection 5.5. In Figures 2(a-d), we find the computation time of a single iteration of the FW TKL and APD algorithms for both classification and regression on an Intel i7-5960X CPU with 128 Gb of RAM as a function of m for several values of n_P, where m is the number of samples used to learn the TK kernel function and the size of P is $n_P \times n_P$ (so that n_P is a function of the number of features and the degree of the monomial basis Z_d). The data set used for these plots is California Housing (CA) in Pace and Barry (1997b), containing 9 features and $m = 20,640$ samples. In the case of classification, labels with value greater than or equal to the median of the output were relabeled as 1, and those less than the median were relabeled as -1. Figures 2(a-d) demonstrate that the complexity of Algorithm 2 (Algorithm 4) scales as approximately $O(m^{2.6} n_P^{1.8})$ ($O(m^{2.7} n_P^{1.6})$) for classification and $O(m^{2.4} n_P^{1.9})$ ($O(m^{2.4} n_P^{1.2})$) for regression. While this complexity scaling is consistent with theoretical bounds, and while the difference in iteration complexity between the FW and APD algorithms for these data sets is minimal, we find significant differences in scaling between data sets. Furthermore, we find that for similar scaling factors, the FW iteration is approximately 3 times faster than the APD iteration. This multiplicative factor increases to 100 when compared to the SDP algorithm in Colbert and Peet (2020). This factor is illustrated for classification using four data sets in Table 1.

8. Accuracy of the New TK Kernel Learning Algorithm

In this section, we compare the accuracy of the classification and regression solutions obtained from the FW TKL algorithm with N as defined in Eq. (11) to the accuracy of

Name	Type	Source	References
Liver	Classification	UCI	McDermott and Forsyth (2016)
Cancer	Classification	UCI	Wolberg et al. (1990)
Heart	Classification	UCI	No Associated Publication
Pima	Classification	UCI	No Associated Publication
Hill Valley	Classification	UCI	No Associated Publication
Shill Bid	Classification	UCI	Alzahrani and Sadaoui (2018, 2020)
Abalone	Classification	UCI	Waugh (1995)
Transfusion	Classification	UCI	Yeh et al. (2009)
German	Classification	LIBSVM	No Associated Publication
Four Class	Classification	LIBSVM	Ho and Kleinberg (1996)
Gas Turbine	Regression	UCI	Kaya et al. (2019)
Airfoil	Regression	UCI	Brooks et al. (1989)
CCPP	Regression	UCI	Tüfekci (2014); Kaya et al. (2012)
CA	Regression	LIBSVM	Pace and Barry (1997b)
Space	Regression	LIBSVM	Pace and Barry (1997a)
Boston Housing	Regression	LIBSVM	Harrison and Rubinfeld (1978)

Table 2: References for the data sets used in Section 8. All data sets are available on the UCI Machine Learning Repository or from the LIBSVM database.

SimpleMKL, Neural Networks, Random Forest, and XGBoost algorithms. In the case of classification we also include three algorithms from the MKLpy python toolbox (AMKL, PWMK, and CKA).

References to the original sources for the data sets used in Section 8 of the paper are included in Table 2. Six classification and six regression data sets were chosen arbitrarily from Dua and Graff (2017) and Chang and Lin (2011) to contain a variety of number of features and number of samples. In both classification and regression, the accuracy metric uses 5 random divisions of the data into test sets (m_t samples $\cong 20\%$ of data) and training sets (m samples $\cong 80\%$ of data). For regression, the training data is used to learn the kernel and predictor. The predictor is then used to predict the test set outputs.

Regression analysis Using six different regression data sets, the MSE accuracy of the proposed algorithm (TKL) with N as defined in Eq. (11) was below average on five of the data sets, an improvement over all other algorithms but XGBoost which also scored above average on five of the data sets. To evaluate expected improvement in accuracy, we next compute the average MSE improvement for TKL averaged over all algorithms and data sets to be 23.6% – i.e.

$$\frac{1}{6}\sum_{j=1}^{6}\frac{MSE_{TKL,Dataset(j)}}{\frac{1}{5}\sum_{i=1}^{5}MSE_{Algorithm(i),Dataset(j)}}\cdot 100 = 100\% - 23.6\%$$

This improvement in average performance was better than all other tested algorithms including XGBoost.

Data set	Method	Error	Time (s)	Data set	Method	Error	Time (s)
Gas	TKL	0.23 ± 0.01	13580 ± 2060	CCPP	TKL	10.57 ± 0.82	626.7 ± 456.0
Turbine	SMKL	N/A	N/A	$n = 4$	SMKL	13.93 ± 0.78	13732 ± 1490
$n = 11$	NNet	0.27 ± 0.03	1172 ± 100	$m = 8000$	NNet	15.20 ± 1.00	305.71 ± 9.25
$m = 30000$	RF	0.38 ± 0.02	16.44 ± 0.57	$m_t = 1568$	RF	10.75 ± 0.70	1.65 ± 0.19
$m_t = 6733$	XGBoost	0.33 ± 0.005	49.46 ± 1.93		XGBoost	8.98 ± 0.81	5.47 ± 2.73
Airfoil	TKL	1.41 ± 0.44	49.87 ± 4.29	CA	TKL	.012 ± .0003	1502 ± 2154
$n = 5$	SMKL	4.33 ± 0.79	617.8 ±161.6	$n = 8$	SMKL	N/A	N/A
$m = 1300$	NNet	6.06 ± 3.84	211.9 ± 41.0	$m = 16500$	NNet	.0113 ± .0004	914.3 ± 95.9
$m_t = 203$	RF	2.36 ± 0.42	0.91 ± 0.20	$m_t = 4140$	RF	.0096 ± .0003	5.28 ± 3.13
	XGBoost	1.51 ± 0.40	2.59 ± 0.06		XGBoost	.0092 ± .0002	5.28 ± 3.13
Space	TKL	.013 ± .001	121.8 ± 49.2	Boston	TKL	10.36 ± 5.80	63.05 ± 2.90
$n = 12$	SMKL	.019 ± .005	3384 ± 589	Housing	SMKL	15.46 ± 11.49	10.39 ± 0.89
$m = 6550$	NNet	.014 ± .004	209.7 ± 37.4	$n = 13$	NNet	50.90 ± 44.19	79.2 ± 42.8
$m_t = 1642$	RF	.017 ± .003	1.06 ± 0.27	$m = 404$	RF	10.27 ± 5.70	0.68 ± 0.40
	XGBoost	.015 ± .002	0.32 ± 0.02	$m_t = 102$	XGBoost	9.40 ± 4.17	0.14 ± 0.06

Table 3: Regression performance of Tessellated Kernel learning for 6 regression data sets with comparison of 5 different ML algorithms. Each measurement was repeated 5 times. The resulting test Mean Squared Error (MSE) and training time are included in the table. All tests are run on a desktop with Intel i7-5960X CPU at 3.00 GHz and with 128 Gb of RAM. N/A indicates that the algorithm was stopped after 24 hours without a solution. For each data set. the number of samples, m, the size of the test part m_t and the number of features, n, are presented.

Predictably, the computational time of TKL is significantly higher than non-convex non-kernel-based approaches such as RF or XGBoost. However, the computation time of TKL is lower than other kernel-learning methods such as SMKL — note that for large data sets ($m > 5000$) TKL is at least 20 times faster than SMKL. Surprisingly, the computation time of TKL is comparable to over-parameterized non-convex stochastic descent methods such as NNet.

Classification analysis Using six classification data sets and comparing 7 algorithms, the TSA of the proposed TKL algorithm was above average on all of the data sets, an improvement over all other algorithms. Next, we compute the average improvement in accuracy of TKL over average TSA for all algorithms to be 6.77% – i.e.

$$\frac{1}{6}\sum_{j=1}^{6}\frac{TSA_{TKL,Dataset(j)}}{\frac{1}{8}\sum_{i=1}^{8}TSA_{Algorithm(i),Dataset(j)}}\cdot 100 = 100\% + 6.77\%$$

This was close to the top score of 6.84% achieved by the AMKL algorithm (The PWMK algorithm failed to converge on one data set, and the TSA from this test was not included in the calculation).

Again, the computational time of TKL is significantly higher than RF or XGBoost, but comparable to other kernel learning methods and NNet. Unlike TKL, the computational time of other MKL methods is highly variable and often does not seem to scale predictably with the number of samples and features – e.g. PWMK for FourClass and Shill Bid data sets and AMKL for Transfusion and German, where the computational time is much higher for smaller data sets.

Details of the implementation of the algorithms used in this study are as follows.

Data set	Method	Accuracy (%)	Time (s)	Data set	Method	Accuracy (%)	Time (s)
Abalone	TKL	84.61 ± 1.60	17.63 ± 3.77	Hill Valley	TKL	86.70 ± 5.49	86.7 ± 48.2
$n = 8$	SMKL	83.13 ± 1.06	350.4 ± 175.1	$n = 100$	SMKL	51.23 ± 3.55	2.81 ± 2.83
$m = 4000$	NNet	84.70 ± 1.82	4.68 ± 0.64	$m = 1000$	NNet	70.00 ± 4.79	3.79 ± 1.75
$m_t = 677$	RF	84.11 ± 1.33	0.98 ± 0.21	$m_t = 212$	RF	56.04 ± 3.27	0.75 ± 0.33
	XGBoost	82.69 ± 1.06	0.20 ± 0.06		XGBoost	55.66 ± 2.37	0.58 ± 0.34
	AMKL	84.64 ± 1.01	0.95 ± 0.07		AMKL	94.71 ± 1.72	5.50 ± 3.84
	PWMK	84.64 ± 1.01	3.13 ± 0.12		PWMK	94.34 ± 1.69	13.10 ± 5.19
	CKA	65.05 ± 0.76	21.43 ± 0.32		CKA	47.92 ± 0.57	0.50 ± 0.08
Transfusion	TKL	77.84 ± 3.89	0.25 ± 0.08	Shill Bid	TKL	99.76 ± 0.08	23.66 ± 2.63
$n = 4$	SMKL	76.62 ± 4.79	2.44 ± 3.08	$n = 9$	SMKL	97.71 ± 0.32	81.0 ± 13.1
$m = 600$	NNet	78.78 ± 3.26	1.01 ± 0.47	$m = 5000$	NNet	98.64 ± 0.86	3.56 ± .60
$m_t = 148$	RF	75.00 ± 3.58	0.54 ± 0.24	$m_t = 1321$	RF	99.35 ± 0.14	0.78 ± 0.36
	XGBoost	73.92 ± 3.95	0.13 ± 0.11		XGBoost	99.61 ± 0.06	0.13 ± 0.04
	AMKL	74.46 ± 1.50	766.9 ± 315.4		AMKL	99.72 ± 0.10	1.24 ± 0.04
	PWMK	N/A	N/A		PWMK	99.72 ± 0.10	3.03 ± 0.04
	CKA	76.35 ± 4.27	0.18 ± 0.03		CKA	99.65 ± 0.21	55.8 ± 0.9
German	TKL	75.80 ± 1.89	58.7 ± 36.1	FourClass	TKL	99.77 ± 0.32	0.13 ± 0.01
$n = 24$	SMKL	74.30 ± 3.55	17.78 ± 4.79	$n = 2$	SMKL	94.53 ± 12.2	0.85 ± 0.48
$m = 800$	NNet	72.70 ± 3.98	0.61 ± 0.05	$m = 690$	NNet	99.99 ± 0.01	0.53 ± 0.03
$m_t = 200$	RF	74.90 ± 1.35	0.64 ± 0.28	$m_t = 172$	RF	99.30 ± 0.44	0.68 ± 0.53
	XGBoost	72.40 ± 2.89	0.10 ± 0.03		XGBoost	98.95 ± 0.44	0.04 ± 0.00
	AMKL	70.80 ± 1.47	2.88 ± 0.38		AMKL	99.99 ± 0.01	1.39 ± 0.03
	PWMK	70.70 ± 1.50	907.0 ± 77.6		PWMK	99.99 ± 0.01	990 ± 60.2
	CKA	68.50 ± 1.58	0.25 ± 0.03		CKA	66.16 ± 3.44	0.16 ± 0.03

Table 4: Classification performance of Tessellated Kernel learning for 6 different classification data sets with comparison of 7 ML methods. Each measurement was repeated 5 times. The resulting Test Set Accuracy and training time are presented in the table. All tests are run on a desktop with Intel i7-5960X CPU at 3.00 GHz and with 128 Gb of RAM. N/A indicates that the algorithm was stopped after 24 hours without a solution. For each data set. the number of samples, m, the size of the test part m_t and the number of features, n, are presented.

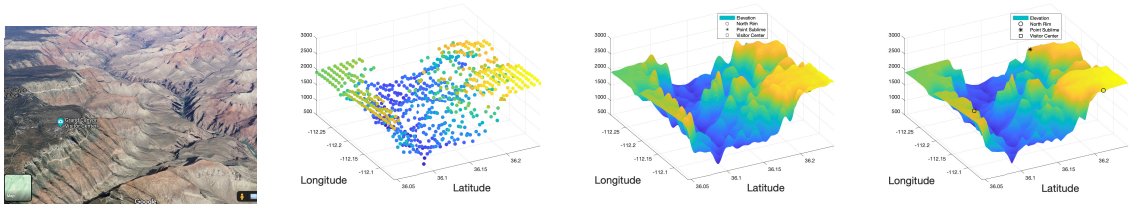
[**TKL**] Algorithm 2 with N as defined in Eqn. (11), where Z_d is a vector of monomials of degree $d = 1$ or less. The regression problem is posed using $\epsilon = .1$. The data is scaled so that $x_i \in [0, 1]^n$ and $[a, b] = [0-\delta, 1+\delta]^n$, where $\delta \geq 0$ and C in the kernel learning problem are chosen by 2-fold cross-validation. Implementation and documentation of this method is described in Appendix D.1 and is publicly available via Github (Colbert et al., 2021);

[**SMKL**] SimpleMKL proposed in Rakotomamonjy et al. (2008) with a standard selection of Gaussian and polynomial kernels with bandwidths arbitrarily chosen between .5 and 10 and polynomial degrees one through three - yielding approximately $13(n + 1)$ kernels. The regression and classification problems are posed using $\epsilon = .1$ and C is chosen by 2-fold cross-validation;

[**NNet**] A neural network with 3 hidden layers of size 50 using MATLAB's `patternnet` for classification and `feedforwardnet` for regression where learning is halted after the error in a validation set decreased sequentially 50 times;

[**RF**] The Random Forest algorithm as in Breiman (2004) as implemented on the scikit-learn python toolbox (see Pedregosa et al., 2011)) for classification and regression. Between 50 and 650 trees (in 50 tree intervals) are selected using 2-fold cross-validation;

[**XGBoost**] The XGBoost algorithm as implemented in Chen and Guestrin (2016) for classification and regresion. Between 50 and 650 trees (in 50 tree intervals) are selected using 2-fold cross-validation;



(a) An image from Google Maps of a section of the Grand Canyon corresponding to (36.04, -112.05) latitude and (36.25, -112.3) longitude.

(b) Elevation data ($m = 750$) from Becker et al. (2009) for a section of the Grand Canyon between (36.04, -112.05) latitude and (36.25, -112.3) longitude.

(c) Predictor using a hand-tuned Gaussian kernel trained on the elevation data in (b). The Gaussian predictor poorly represents the sharp edge at the north and south rim.

(d) Predictor from Algorithm 2 trained on the elevation data in (b). The TK predictor accurately represents the north and south rims of the canyon.

Figure 3: Subfigure (a) shows an 3D representation of the section of the Grand Canyon to be fitted. In (b) we plot elevation data of this section of the Grand Canyon. In (c) we plot the predictor for a hand-tuned Gaussian kernel. In (d) we plot the predictor from Algorithm 2 for $d = 2$.

[**AMKL**] The AverageMKL implementation from the MKLpy python package proposed in Lauriola and Aiolli (2020) – averages a standard selection of Gaussian and polynomial kernels;
[**PWMK**] The PWMK implementation from the MKLpy python package proposed in Lauriola and Aiolli (2020), which uses a heuristic based on individual kernel performance as in Tanabe et al. (2008) to learn the weights of a standard selection of Gaussian and polynomial kernels;
[**CKA**] The CKA implementation from the MKLpy python package (Lauriola and Aiolli, 2020), uses the centered kernel alignment optimization in closed form as in Cortes et al. (2010) to learn the weights of a standard selection of Gaussian and polynomial kernels.

To further illustrate the importance of density property and the TKL framework for practical regression problems, Algorithm 2 with $N = N_2^T$ was applied to elevation data from Becker et al. (2009) to learn a SVM predictor representing the surface of the Grand Canyon in Arizona. This data set is particularly challenging due to the variety of geographical features. The results can be seen in Figure 3(d) where we see that the regression surface visually resembles a photograph of this terrain, avoiding the artifacts present in the SVM from an optimized Gaussian kernel seen in Figure 3(c).

9. Conclusion

We have proposed a generalized kernel learning framework – using positive matrices to parameterize positive kernels. While such problems can be solved using semidefinite programming, the use of SDP results in large computational overhead. To reduce this computational complexity, we have proposed a saddle-point formulation of the generalized kernel learning problem. This formulation leads to a primal-dual decomposition, which can then be solved efficiently using algorithms of the Frank-Wolfe or accelerated primal dual type – with corresponding theoretical guarantees of convergence. In both cases, we have shown that

the primal and dual sub-problems can be solved using a singular value decomposition and quadratic programming, respectively. Numerical experiments confirm that the FW-based algorithm is approximately 100 times faster than the previous SDP algorithm from Colbert and Peet (2020). Finally, 12 large and randomly selected data sets were used to test accuracy of the proposed algorithms compared to 8 existing state-of-the-art alternatives – yielding uniform increases in accuracy with similar or reduced computational complexity.

Acknowledgments

We would like to acknowledge support for this project from the National Science Foundation grant CCF 2323532 and National Institutes of Health grant NIH R01 GM144966.

Appendix A. Proof of Universality of Tessellated Kernels

In this appendix, we find a detailed proof of Lemma 14.

Lemma 14 *For any $a, b, \delta \in \mathbb{R}^n$ with $a < b$ and $\delta > 0$, let $Y = [a-\delta, b+\delta]$ and $X = [a, b]$. Then the kernel*

$$k(x,y) = \int_Y \mathbf{I}(z-x)\mathbf{I}(z-y)dz = \int_{\substack{x\leq z\\ y\leq z\\ z\leq b+\delta}} 1dz = \prod_{i=1}^n (b_i + \delta_i - \max\{x_i, y_i\}) = \prod_{i=1}^n k_i(x_i, y_i)$$

is universal where $k_i(x,y) := b_i + \delta_i - \max\{x_i, y_i\}$.

Proof As per Micchelli et al. (2006), $k : X \times X \to \mathbb{R}$ is universal if it is continuous and, if for any $g \in \mathcal{C}(X)$ and any $\epsilon > 0$ there exist $f \in \mathcal{H}$ such that $\|f - g\|_{\mathcal{C}} \leq \epsilon$ where

$$\mathcal{H} := \{f \ : \ f(x) = \sum_{i=1}^m \alpha_i k(x, y_i) \ : \ y_i \in X, \ \alpha_i \in \mathbb{R}, \ m \in \mathbb{N}\}.$$

First, we show that each kernel, $k_i(x,y)$, with corresponding RKHS $\mathcal{H}_i$, is universal for $i = 1, \cdots, n$. As shown in Colbert and Peet (2020), any triangle function, Λ, of height 1, of width 2β, and centered at $y_2 \in [a_i + \beta, b_i - \beta]$, is in $\mathcal{H}_i$, since

$$\Lambda(x) = \sum_{j=1}^3 \alpha_j k_i(x, y_j) = \begin{cases} 0, & \text{if } x < y_1 \\ \frac{1}{\beta}(x - y_1), & \text{if } y_1 \leq x < y_2 \\ 1 - \frac{1}{\beta}(x - y_2), & \text{if } y_2 \leq x < y_3 \\ 0, & \text{if } y_3 \leq x \end{cases}$$

where $y_1 = y_2 - \beta$, $y_3 = y_2 + \beta$ and $\alpha_1 = \alpha_3 = -\frac{1}{\beta}$, $\alpha_2 = 2\frac{1}{\beta}$. Furthermore, using $\alpha_1 = \frac{1}{\delta}$ and $y_1 = b_i$, we have $\alpha_1 k_i(x, y_1) = \frac{1}{\delta} k_i(x, b_i) = 1$ and so the constant function $\Lambda = 1$ is also in $\mathcal{H}_i$. Thus we conclude that $\mathcal{H}_i$ contains the Schauder basis for $\mathcal{C}([a_i, b_i])$ (See Hunter and Nachtergaele, 2001) and hence the kernel is universal for $n = 1$.

Next, since $k = \prod_{i=1}^n k_i$, we have that

$$\mathcal{H} = \left\{ \prod_{i=1}^n f_i \ : \ f_i \in \mathcal{H}_i \right\}.$$

Thus, since the k_i are universal, for any for any $\gamma \in \mathbb{N}^n$ and $x_i^{\gamma_i}$, there exist $g_i \in \mathcal{H}_i$ such that $|x_i^{\gamma_i} - g_i(x_i)| \leq \epsilon$.

Note that $g_i(x_i)$ are bounded, indeed define $C_i := (\max\{|a_i|, |b_i|\})^{\gamma_i} + 1$ and $C := \prod_{i=1}^n C_i$, then

$$\sup_{x_i \in [a_i, b_i]} |g_i(x_i)| \leq \sup_{x_i \in [a_i, b_i]} |x_i^{\gamma_i}| + \epsilon \leq ((\max\{|a_i|, |b_i|\})^{\gamma_i} + \epsilon \leq (\max\{|a_i|, |b_i|\})^{\gamma_i} + 1 = C_i.$$

According to Hardy et al. (1952), for all $a_i, b_i \in \mathbb{R}$ such that $|a_i| \leq 1$ and $|b_i| \leq 1$, we have that $|\prod_{i=1}^n a_i - \prod_{i=1}^n b_i| \leq \sum_{i=1}^n |a_i - b_i|$. Let $g(x) = \prod_{i=1}^n g_i(x_i)$. Then, since

$\left|\frac{g_i(x_i)}{C_i}\right| \leq 1$, $\left|\frac{x_i^{\gamma_i}}{C_i}\right| \leq 1$ and $C_i > 1$, we have that

$$\begin{aligned}\sup_{x\in X}|x^\gamma - g(x)| &= \sup_{x\in X}\left|\prod_{i=1}^n x_i^{\gamma_i} - \prod_{i=1}^n g_i(x_i)\right| \\ &= C\sup_{x\in X}\left|\prod_{i=1}^n \frac{x_i^{\gamma_i}}{C_i} - \prod_{i=1}^n \frac{g_i(x_i)}{C_i}\right| \\ &\leq C\sum_{i=1}^n \left|\frac{x_i^{\gamma_i}}{C_i} - \frac{g_i(x_i)}{C_i}\right| \\ &\leq nC\max_{i\in\{1,\cdots,n\}}|x_i^{\gamma_i} - g_i(x_i)| \leq nC\epsilon.\end{aligned}$$

We conclude that for any $\gamma \in \mathbb{N}^n$, $\lambda_\gamma \in \mathbb{R}$ and $\varepsilon > 0$, there exists $\{\alpha_{\gamma,j}\}_{j=1}^{m_\gamma}$ and $\{y_{\gamma,j}\}_{j=1}^{m_\gamma}$ such that

$$\sup_{x\in X}\left\|\sum_{j=1}^{m_\gamma}\alpha_{\gamma,j}k(x,y_{\gamma,j}) - \lambda_\gamma x^\gamma\right\| \leq \varepsilon.$$

Now, the Weierstrass approximation theorem (Willard, 2012) states that polynomials as a linear combination of monomials are dense in $\mathcal{C}(X)$. Thus for any $f \in \mathcal{C}(X)$ and $\varepsilon > 0$ there exists a maximal degree $d \in \mathbb{N}$ and a polynomial function $p(x) = \sum_{\|\gamma\|_1\leq d}\lambda_\gamma x^\gamma$ such that

$$\sup_{x\in X}\Big\|f(x) - p(x)\Big\| \leq \varepsilon.$$

Let $\{\alpha_{\gamma,j}\}_{j=1}^{m_\gamma}$ and $\{y_{\gamma,j}\}_{j=1}^{m_\gamma}$ be as defined above. Then, using the triangle inequality, we have

$$\begin{aligned}\sup_{x\in X}\left\|f(x) - \sum_{\|\gamma\|_1\leq d}\sum_{j=1}^{m_\gamma}\alpha_{\gamma,j}k(x,y_{\gamma,j})\right\| &= \sup_{x\in X}\left\|f(x) - p(x) + p(x) - \sum_{\|\gamma\|_1\leq d}\sum_{j=1}^{m_\gamma}\alpha_{\gamma,j}k(x,y_{\gamma,j})\right\| \\ &\leq \sup_{x\in X}\Big\|f(x) - p(x)\Big\| + \sup_{x\in X}\left\|p(x) - \sum_{\|\gamma\|_1\leq d}\sum_{j=1}^{m_\gamma}\alpha_{\gamma,j}k(x,y_{\gamma,j})\right\| \\ &\leq \sup_{x\in X}\Big\|f(x) - p(x)\Big\| + \sum_{\|\gamma\|_1\leq d}\sup_{x\in X}\left\|\lambda_\gamma x^\gamma - \sum_{j=1}^{m_\gamma}\alpha_{\gamma,j}k(x,y_{\gamma,j})\right\| \\ &\leq \left(\binom{n+d}{d} + 1\right)\varepsilon.\end{aligned}$$

Thus, since for any $f \in \mathcal{C}(X)$ and any $\epsilon > 0$, there exists $g = \sum_{\|\gamma\|_1\leq d}\sum_{j=1}^{m_\gamma}\alpha_{\gamma,j}k(x,y_{\gamma,j})$ such that $\|f - g\|_C \leq \epsilon$. Therefore, since any $k \in \mathcal{K}_T$ is continuous (See Colbert and Peet, 2020), we have that k is universal. ■

Appendix B. An Accelerated Algorithm for Quadratic Convergence

In this appendix, we consider the Accelerated Primal-Dual (APD) algorithm discussed in the main text which can be used to achieve quadratic convergence for Generalized Kernel Learning. First, we define the APD algorithm and prove quadratic convergence. Furthermore, we show that the APD algorithm can be decomposed into primal and dual sub-problems which may be solved using QP and the SVD in a manner similar to the FW algorithm. Finally, we note that the APD algorithm underperforms the proposed FW algorithm for the first several iterations and hence we propose a hybrid algorithm which uses FW until an error tolerance is satisfied and then switched to APD for subsequent iterations.

B.1 An Algorithm with $O\left(\frac{1}{k^2}\right)$ Convergence

As discussed in Section 3, the Generalized Kernel Learning problem can be represented as a minmax or saddle point optimization problem (2) which is linear in P and convex in α

$$\min_{\alpha\in\mathcal{Y}}\max_{P\in\mathcal{X}} f(\alpha)+\Phi(\alpha)-h(P), \tag{14}$$

where for Generalized Kernel learning we have that $f(\alpha)=\kappa_\star(\alpha)$, $\Phi(\alpha)=\lambda(e_\star\odot\alpha)$, and $h(P)=0$ as defined in section 5.

Numerically, we observe that for several first iterations, the Frank-Wolfe GKL algorithm achieves super-sublinear convergence - which is often sufficient to achieve an error tolerance of 10^{-5}. However, if lower error tolerances are desired, the sublinear convergence rate of the FW algorithm at higher iterations can be accelerated by a algorithm with $O\left(\frac{1}{k^2}\right)$ convergence.

Fortunately, Hamedani and Aybat (2021) has shown that provable $O\left(\frac{1}{k^2}\right)$ convergence can be achieved using a variation of an algorithm originally proposed in Chambolle and Pock (2016). This algorithm, the accelerated primal-dual (APD) algorithm, requires computation of the same sub-problems, OPT_A and OPT_P, but it achieves the worst case $O\left(\frac{1}{k^2}\right)$ convergence.

Specifically, this algorithm can be used to solve problems of the form

$$\min_{\alpha\in\mathcal{Y}_\star}\max_{P\in\mathcal{X}} -\kappa_\star(\alpha)-O(\alpha\odot e_\star,P)=f(\alpha)+\Phi(P,\alpha)-h(P), \tag{15}$$

where $f(\alpha)+\Phi(\alpha)$ is strongly convex and $h(P)$ is concave. Since the Generalized Kernel Learning Problem (GKL (7)) has the same form as problem (15), application of this approach is relatively straightforward. Specifically, consider Algorithm 3 from Hamedani and Aybat (2021) which requires the following definition.

Definition 15 (Bregman distance (Hamedani and Aybat, 2021)) *Given f and h, let $\phi_Y:\mathcal{Y}\to\mathbb{R}$ and $\phi_X:\mathcal{X}\to\mathbb{R}$ be differentiable functions on open sets $\mathcal{Y}\subset\mathbf{dom}\,f$ and $\mathcal{X}\subset\mathbf{dom}\,h$. Suppose ϕ_X and ϕ_Y have closed domains and are 1-strongly convex with respect to $\|\cdot\|_{\mathcal{X}}$ and $\|\cdot\|_{\mathcal{Y}}$, respectively. We define the Bregman distances $D_{\mathcal{X}}:\mathcal{X}\times\mathcal{X}\to\mathbb{R}$ and $D_{\mathcal{Y}}:\mathcal{Y}\times\mathcal{Y}\to\mathbb{R}$ corresponding to ϕ_X and ϕ_Y, respectively, as $D_{\mathcal{X}}(x,\hat{x})=\phi_{\mathcal{X}}(x)-\phi_{\mathcal{X}}(\hat{x})-\langle\nabla\phi_{\mathcal{X}}(\hat{x}),x-\hat{x}\rangle$ and $D_{\mathcal{Y}}(y,\hat{y})=\phi_{\mathcal{Y}}(y)-\phi_{\mathcal{Y}}(\hat{y})-\langle\nabla\phi_{\mathcal{Y}}(\hat{y}),y-\hat{y}\rangle$.*

`Initialize` μ, τ_0, σ_0, $k_{\max}$ and $\alpha_0, P_0 \in \mathcal{Y} \times \mathcal{X}$
$k = 0$, $(\tau_{-1}, \sigma_{-1}) = (\tau_0, \sigma_0)$, $(P_{-1}, \alpha_{-1}) = (P_0, \alpha_0)$ and $\gamma_0 = \frac{\sigma_0}{\tau_0}$;
while $k < k_{\max}$ **do**
1: $\sigma_k = \gamma_k \tau_k, \quad \theta_k = \sigma_{k-1}/\sigma_k$
2: $x_k = \nabla_P \Phi(\alpha_k, P_k)$
3: $S = (1 + \theta_k)x_k - \theta_k x_{k-1}$
4: $P_{k+1} = \arg\min_{P \in \mathcal{X}} \frac{1}{\sigma_k} D_X(P, P_k) - \langle S, P \rangle$
5: $\alpha_{k+1} = \arg\min_{\alpha \in \mathcal{Y}} f(\alpha) + \Phi(P_{k+1}, \alpha) + \frac{D_Y(\alpha, \alpha_k)}{\tau_k}$
6: $\gamma_{k+1} = \gamma_k(1 + \mu\tau_k)$, $\tau_{k+1} = \tau_k \sqrt{\frac{\gamma_k}{\gamma_{k+1}}}$, $k = k + 1$
end while

Algorithm 3: APD algorithm

Theorem 16 proves $O\left(\frac{1}{k^2}\right)$ convergence of Algorithm 3 when $f(\alpha) + \Phi(P, \alpha)$ is strongly convex in α and $h(P)$ is concave.

Theorem 16 (Hamedani and Aybat (2021)) *Let $D_{\mathcal{X}}$ and $D_{\mathcal{Y}}$ be Bregman distance functions. Suppose that for any $P \in \mathcal{X}$, $f(\alpha)$ and $\Phi(P, \alpha)$ are convex in α and for any $\alpha \in \mathcal{Y}$, $\Phi(P, \alpha)$ and $-h(P)$ are concave in P. In addition, suppose f is strongly convex with modulus $\mu > 0$. Furthermore, suppose that $L_{\alpha\alpha}, L_{P\alpha}$ satisfy*

$$\|\nabla_P \Phi(P, \alpha) - \nabla_P \Phi(\hat{P}, \hat{\alpha})\|_{\mathcal{X}^*} \leq L_{P\alpha} \|\alpha - \hat{\alpha}\|_{\mathcal{Y}}$$
$$\|\nabla_\alpha \Phi(P, \alpha) - \nabla_\alpha \Phi(P, \hat{\alpha})\|_{\mathcal{Y}^*} \leq L_{\alpha\alpha} \|\alpha - \hat{\alpha}\|_{\mathcal{Y}}$$

for all $\alpha, \hat{\alpha} \in \mathcal{Y}$ and $P, \hat{P} \in \mathcal{X}$ and that the starting parameters τ_0 and σ_0 satisfy:

$$\left(\frac{1 - \delta}{\tau_0} - L_{\alpha\alpha}\right) \frac{1}{\sigma_0} \geq \frac{L_{P\alpha}^2}{c_a}$$

for some $\delta, c_a \in \mathbb{R}_+$, such that $c_a + \delta \leq 1$. If

$$\{\alpha^*, P^*\} = \arg\min_{\alpha \in \mathcal{Y}_\star} \max_{P \in \mathcal{X}} f(\alpha) + \Phi(P, \alpha) - h(P),$$

exists, then for any sequence produced by Algorithm 3, $\{\alpha_k, P_k\}$, we have that
1. $\lim_{k \to \infty} \{P_k, \alpha_k\} \to \{P^*, \alpha^*\}$
2. If $\delta > 0$, and we define $L(\alpha, P) := f(\alpha) + \Phi(P, \alpha) - h(P)$ then

$$0 \leq L(P^*, \alpha_k) - L(P_k, \alpha^*) \leq O\left(\frac{1}{k^2}\right).$$

Proof See Hamedani and Aybat (2021) ■

Remark: As stated in Hamedani and Aybat (2021), the conditions of Theorem 16 are satisfied using $\tau_0 = \frac{1}{3L_{\alpha\alpha}}$ and $\sigma_0 = \frac{L_{\alpha\alpha}}{2L_{P\alpha}^2}$.

B.2 Proposed Booster Algorithm

In this subsection, we propose the following algorithm applicable to our optimization problem. This algorithm is a specification of APD for the Generalized Kernel learning.

`Initialize` $(\alpha_0, P_0) \in \mathcal{Y}_\star \times \mathcal{X}$, μ
$\sigma_{-1} = \sigma_0 = \frac{L_{\alpha\alpha}}{2L_{\alpha P}^2}$, $\tau_{-1} = \tau_0 = \frac{1}{3L_{\alpha\alpha}}$
$k = 0$, $(P_{-1}, \alpha_{-1}) = (P_0, \alpha_0)$ and $\gamma_0 = \frac{\sigma_0}{\tau_0}$;
while $L(P_{k+1}, \alpha_k) - L(P_k, \alpha_{k+1}) > \epsilon$ **do**
1: $\sigma_k = \gamma_k \tau_k, \quad \theta_k = \sigma_{k-1}/\sigma_k$
2: $x_k = \frac{1}{2} D(e_\star \odot \alpha_k)$
3: $S = (1 + \theta_k) x_k - \theta_k x_{k-1}$
4: $P_{k+1} = \arg APD\_P(P_k, S, \sigma_k)$
5: $\alpha_{k+1} = \arg APD\_A(\alpha_k, P_{k+1}, \tau_k)$
6: $\gamma_{k+1} = \gamma_k(1 + \mu\tau_k)$, $\tau_{k+1} = \tau_k \sqrt{\frac{\gamma_k}{\gamma_{k+1}}}$, $k = k + 1$
end while

Algorithm 4: APD algorithm

where $L(P, \alpha) := -\lambda(e_\star \odot \alpha, P) - \kappa_\star(\alpha)$ and

$$D_{i,j}(\alpha) := \sum_{k,l=1}^{m} \alpha_k e_{*k} G_{i,j}(x_k, x_l) \alpha_l e_{*l}$$

is defined in Eqn. (12) and the two subroutines APD_P and APD_A are defined as

$$APD\_A(P, \alpha_k, \tau) := \max_{\alpha \in \mathcal{Y}_\star} \lambda(e_\star \odot \alpha, P) + \kappa_\star(\alpha) - \frac{1}{\tau} D_{\mathcal{Y}}(\alpha, \alpha_k),$$
$$APD\_P(P_k, S, \sigma) := \min_{P \in \mathcal{X}} \frac{1}{\sigma} D_{\mathcal{X}}(P, P_k) - \langle S, P \rangle$$

where $D_{\mathcal{Y}_\star} := \frac{1}{2}\|\cdot\|_2^2$, and $D_{\mathcal{X}} := \frac{1}{2}\|\cdot\|_F^2$.

Furthermore,

$$\tau_0 = \frac{1}{3L_{\alpha\alpha}}, \text{ and } \sigma_0 = \frac{L_{\alpha\alpha}}{2L_{\alpha P}^2} \tag{16}$$

where

$$L_{\alpha\alpha} := \frac{n_P}{2} \sum_{ij} |D_{ij}(e_\star)|^2, \quad L_{\alpha P} := \frac{C}{n_P} L_{\alpha\alpha} \tag{17}$$

where recall that n_P is determined by the size of P ($P \in \mathbb{S}^{n_P}$) and $C > 0$ can be chosen arbitrarily. Finally, we choose $\mu > 0$ sufficiently small such that $-\lambda(\alpha \odot e_\star, P) - \mu\alpha^T\alpha$ is convex for all $P \in \mathcal{X}$.

B.3 $O\left(\frac{1}{k^2}\right)$ Convergence Proof for Algorithm 4

Formally, we state the theorem.

Theorem 17 *Algorithm 4 returns iterates P_k and α_k such that, $L(\alpha_k, P_{k+1}) - L(\alpha_{k+1}, P_k) < O(\frac{1}{k^2})$.*

Proof In this proof, we first show that Algorithm 4 returns iterates P_k and α_k which satisfy Algorithm 3. Next, we show that if τ_0, σ_0 are chosen as per Eqn. (16), then the conditions of Theorem 16 are satisfied. First, let us define

$$\begin{aligned} f(\alpha) &:= -\kappa_\star(\alpha) + \mu\alpha^T\alpha \\ \Phi(P,\alpha) &:= -\lambda(\alpha \odot e_\star, P) - \mu\alpha^T\alpha \\ h(P) &:= 0 \end{aligned} \tag{18}$$

for some sufficiently small $\mu > 0$. Now, suppose that $\{P_k, \alpha_k, \gamma_k, S_k, x_k, \sigma_k, \tau_k\}$ satisfy Algorithm 4. Clearly, these iterations also satisfy Steps 1, 3 and 6 of Algorithm 3. Furthermore, these iterations satisfy the equation defined in Step 2 since

$$\begin{aligned} \nabla_P \Phi(\alpha, P)_{ij} &= \frac{\partial}{\partial P_{ij}} \left[-\lambda(\alpha \odot e_\star, P) - \mu\alpha^T\alpha \right] \\ &= \frac{\partial}{\partial P_{ij}} \left[\frac{1}{2} \sum_{i,j=1}^{n_P} P_{ij} \sum_{k,l=1}^{m} (\alpha_k e_{*k}) G_{i,j}(x_k, x_l)(\alpha_l e_{*l}) \right] \\ &= \frac{1}{2} \sum_{k,l=1}^{m} (\alpha_k e_{*k}) G_{i,j}(x_k, x_l)(\alpha_l e_{*l}) = \frac{1}{2} D_{ij}(e_\star \odot \alpha_k). \end{aligned}$$

Next, the proposed iterations satisfy the equation defined in Step 4 of Algorithm 3 since by the definition of APD_P

$$P_{k+1} = \arg APD\_P(P_k, S, \sigma_k) = \arg\min_{P \in \mathcal{X}} \frac{1}{\sigma_k} D_{\mathcal{X}}(P, P_k) - \langle S, P \rangle .$$

Finally, the equality in Step 5 of Algorithm 3 is satisfied since

$$\begin{aligned} \alpha_{k+1} &= \arg APD\_A(P_{k+1}, \alpha_k, \tau_k) \\ &= \arg\max_{\alpha \in \mathcal{Y}_\star} \lambda(e_\star \odot \alpha, P_{k+1}) + \kappa_\star(\alpha) - \frac{1}{\tau_k} D_{\mathcal{Y}}(\alpha, \alpha_k) \\ &= \arg\min_{\alpha \in \mathcal{Y}_\star} -\lambda(e_\star \odot \alpha, P_{k+1}) - \kappa_\star(\alpha) + \frac{1}{\tau_k} D_{\mathcal{Y}}(\alpha, \alpha_k) \\ &= \arg\min_{\alpha \in \mathcal{Y}_\star} f(\alpha) + \Phi(P_{k+1}, \alpha) + \frac{1}{\tau_k} D_Y(\alpha, \alpha_k). \end{aligned}$$

Therefore, we have that $\{P_k, \alpha_k, \gamma_k, S_k, x_k, \sigma_k, \tau_k\}$ satisfy Algorithm 3.

We next must show that Φ is concave in P, convex in α and f is strongly convex. Φ is linear in P and thus concave in P. As defined, Φ is convex in α and clearly, f is strongly convex for any $\mu > 0$. Since $L_{\alpha\alpha}, L_{\alpha P}$ are as defined in Equation (17), then

$$\begin{aligned} \|\nabla_P \Phi(P, \alpha) - \nabla_P \Phi(\hat{P}, \hat{\alpha})\|_{\mathcal{X}^*} &\leq L_{P\alpha} \|\alpha - \hat{\alpha}\|_{\mathcal{Y}} \\ \|\nabla_\alpha \Phi(P, \alpha) - \nabla_\alpha \Phi(P, \hat{\alpha})\|_{\mathcal{Y}^*} &\leq L_{\alpha\alpha} \|\alpha - \hat{\alpha}\|_{\mathcal{Y}} \end{aligned}$$

as desired. Finally, we have that if τ_0 and σ_0 are as defined in Equation (16), $\delta = \frac{1}{4}$, and $c_a = \frac{1}{2}$, then $c_a + \delta \leq 1$ and

$$\left(\frac{1-\delta}{\tau_0} - L_{\alpha\alpha} \right) \frac{1}{\sigma_0} \geq \frac{L_{P\alpha}^2}{c_a}$$

as desired.

Therefore, we have by Theorem 16 that

$$0 \leq L(P^*, \alpha_k) - L(P_k, \alpha^*) \leq O\left(\frac{1}{k^2}\right),$$

and hence

$$L(\alpha_k, P_{k+1}) - L(\alpha_{k+1}, P_k) < L(P^*, \alpha_k) - L(P_k, \alpha^*) \leq O\left(\frac{1}{k^2}\right).$$

■

We next define efficient algorithms to solve the subroutines APD_P and APD_D.

B.4 Solving *APD_P*

The fourth step of the APD algorithm requires solving APD_P. For arbitrary matrix $S \in \mathbb{S}^{n_P}$ this optimization problem is formulated as follows.

$$P_{k+1} = \arg\min_{P \in \mathcal{Y}} \ \frac{1}{\sigma_k} D_X(P, P_k) - \langle S, P \rangle \tag{19}$$

The following algorithm solves the optimization task 19

`Input` $P_k, S \in \mathbb{S}^{n_P}$, $\sigma_k, \varepsilon > 0$;
`Set` $r = +\infty$ and $A = P_k + \sigma_k S$,
`Singular Value Decomposition:` $A = \sum_i \lambda_i p_i p_i^T$
`Initialize` $y_l = \min_i \lambda_i$ and $y_u = \max_i \lambda_i$;
while $r > \varepsilon$ **do**
1: $y = \frac{1}{2}(y_l + y_u)$
2: $r = \sum_i |\lambda_i - y|_+ - n_P$
3: update $y_u = y$ if $r \geq 0$, or $y_l = y$ otherwise
end while
`Return` $P = \sum_i |\lambda_i - y|_+ p_i p_i^T = \arg APD\_P(P_k, S, \sigma)$.

Algorithm 5: APD_P Subroutine

Lemma 18 *Let the optimization problem be as defined in* (19)*, then the algorithm 5 returns the solution of APD_P.*

Proof Firstly, we should reformulate Optimization Problem (19) as,

$$\arg \min_{\substack{P \in \mathbb{S}^{n_P} \\ \mathrm{trace}(P) = n_P \\ P \succ 0}} \frac{1}{\sigma} D_X(P, P_k) - \langle S, P \rangle = \arg \min_{\substack{P \in \mathbb{S}^{n_P} \\ \mathrm{trace}(P) = n_P \\ P \succ 0}} \frac{1}{2\sigma} \|P - P_k\|_F^2 - \langle S, P \rangle$$

$$= \arg \min_{\substack{P \in \mathbb{S}^{n_P} \\ \mathrm{trace}(P) = n_P \\ P \succ 0}} \frac{1}{2\sigma} \langle P - P_k, P - P_k \rangle - 2\sigma \langle S, P \rangle = \arg \min_{\substack{P \in \mathbb{S}^{n_P} \\ \mathrm{trace}(P) = n_P \\ P \succ 0}} \frac{1}{2\sigma} \|P - (P_k + \sigma S)\|_F^2,$$

$$= \arg \min_{\substack{P \in \mathbb{S}^{n_P} \\ \mathrm{trace}(P) = n_P \\ P \succ 0}} \|P - A\|_F^2, \tag{20}$$

where $A = P_k + \sigma S$.

Having reduced APD_P to a convex distance minimization problems of the form of Eqn. (20). According to Harada (2018), we use a subroutine defined by Algorithm 5 to solve problem (20) and therefore to find P_{k+1} in Step 4 of Algorithm 4. ■

B.5 Solving *APD_A*

APD_A is a QP of the form

$$\min_{\alpha \in \mathcal{Y}_\star} \quad \frac{1}{2}\alpha^T Q_* \alpha + c_*^T \alpha,$$

where, $c_*^T \alpha = -\tau \kappa_\star(\alpha) - \alpha_p^T \alpha$, and $\alpha^T Q_* \alpha = -\tau \lambda(e_\star \odot \alpha, P) + \alpha^T \alpha$.

QP's of this form can be solved using a slight variation of the algorithm proposed in Subsection 5.3.

B.6 Combined Solution for General Kernel Learning

We can see that the Frank-Wolfe GKL Algorithm converges quite quickly up to a certain value, but after that the convergence slows down and becomes linear. Moreover, APD algorithm return a smaller objective function after 3000-4000 iterations. But, the pure APD has non-monotonic convergence at the early stage. All this together prompted us to create a combined Frank-Wolfe GKL and APD algorithm. The tolerance for the Frank-Wolfe GKL was chosen according to the numerical results.

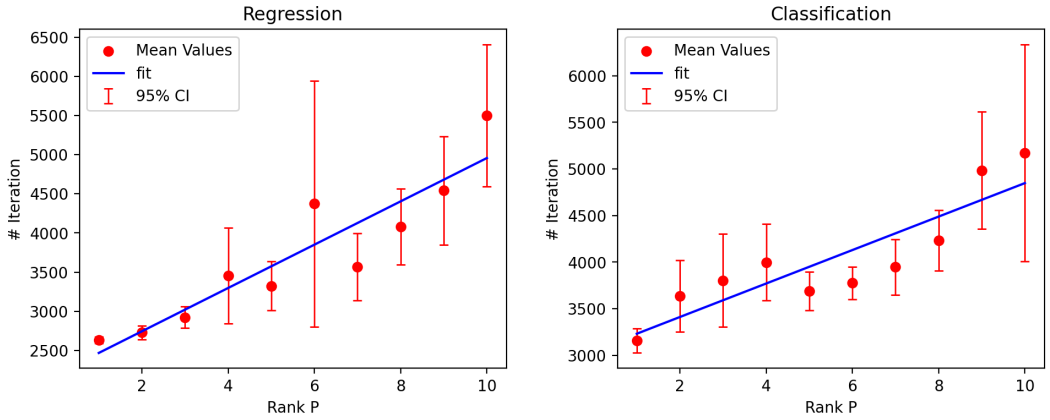
`INPUT` ϵ - tolerance;
1: (α_1, P_1) = Frank Wolfe GKL with tolerance ϵ
2: (α_2, P_2) = APD with initial guess α_1, P_1 and desired tolerance
3: $\alpha = OPT\_A(P_2)$, $P = P_2$
`OUTPUT:` α, P

Algorithm 6: Final version of GKL

We assume, that the proposed algorithm 6 will show both fast initial convergence and $O\left(\frac{1}{k^2}\right)$ convergence in the worst case, which is necessary for application to arbitrary data sets.

Appendix C. Computational Complexity of SVM Problem with Optimal Kernel

In this section, we consider the computational complexity of SVM subproblem as a function of matrix P. Although, as discussed in Subsection 7.2, the computation complexity of SVM subproblem does not depend on the size of matrix P, it implicitly depends on the kernel function. Note, that the proposed solution demonstrates that the optimal matrix P is always a rank 1 matrix – See Section 5. However, the proposed set of kernels includes many different kernels – e.g. for matrix P with different ranks. The computational complexity of SVM subproblem is uniquely determined by kernel function and therefore, in our case, depends on matrix P. To investigate this issue, we generate random positive semidefinite matrices



(a) The number of iteration for random matrix P for Support Vector Regression.

(b) The number of iteration for random matrix P for Support Vector Classification.

Figure 4: The number of iterations of SVM subproblem required to achieve the desired tolerance $\varepsilon = 0.1$ as a function of the rank P. The SVM subproblem has been solved using LibSVM implementation. The red dots and error bars represent average number of iterations of the SVM algorithm and 95% confidence interval using 20 trials for a) regression problem for California Housing (CA) data set in Pace and Barry (1997b) and b) for classification problem for Shill Bid data set in Alzahrani and Sadaoui (2018, 2020). We also included the blue line, that indicates the best linear fit of the average number of iterations.

with different ranks and consider the number of iterations of SVM learning problem. We compute matrix P randomly

$$P = \frac{n_P}{r} \sum_{i=1}^{r} \frac{1}{\|v_i\|_2} v_i v_i^T$$

where v is a normal distributed vector and r is a desired rank of matrix P.

In Figure 4 we plot the number of iterations required to achieve the fixed tolerance $\varepsilon = 0.1$ for different ranks of matrix P. The data set used for these plots is California Housing (CA) in Pace and Barry (1997b). The results shows that the rank 1 optimal solution of the proposed algorithm is significantly faster in comparison with other random positive semidefinite matrices P with different ranks.

Appendix D. Implementation and Documentation of Algorithms

In this appendix material, we have provided a MATLAB implementation of the proposed algorithms which can be used to reproduce the numerical results given in Section 7. The primary executable is `PMKL.m`. The demo files `exampleClassification.m` and `exampleRegression.m` illustrate typical usage of this executable for classification and regression problems respectively. This software is available from Github (Colbert et al., 2021).

D.1 Documentation for Included Software

Also included in the main material are the 5 train and test partitions used for each of the 12 data sets used in the numerical results section of the paper. The code `numericalTest.m` allows the user to select the data set and run the FW PMKL algorithm on the five partitions to calculate the average and standard deviation of the MSE (for regression) or TSA (for classification).

The PMKL subroutine

PMKL[1] - Positive Matrix Kernel Learning,

```
>> f = PMKL(x,y,Type,C,params);
```

yields an optimal solution to the minimax program

$$\min_{P\in\mathcal{X}} \max_{\alpha\in\mathcal{Y}_\star} \lambda(e_\star \odot \alpha, P) + \kappa_\star(\alpha),$$

where $\mathcal{X} := \{P \in \mathbb{S}^{n_P} \ : \ \text{trace}(P) = n_P, \ P \succeq 0\}$,

$$\mathcal{Y}_c := \{\alpha \in \mathbb{R}^m \ : \ \alpha^T y = 0, \ \alpha_i \in [0, C]\}, \quad \mathcal{Y}_r := \{\alpha \in \mathbb{R}^m \ : \ \alpha^T e = 0, \ \alpha_i \in [-C, C]\},$$

and where,

- `x` $\in \mathbb{R}^{n\times m}$ is a matrix of n rows corresponding to the number of features and m columns corresponding to the number of samples where $x(:, i)$ is the i'th sample,
- `y` $\in \mathbb{R}^m$ (`y` $\in [-1, 1]^m$) is a row of outputs (labels) for each of the samples in `x` where `y(:,i)` is the i'th output corresponding to the i'th sample `x(:,i)`,
- `Type` is the string `'Classification'` for classification or `'Regression'` for regression,
- `C` is the penalty for miss-classifying a point for classification, or for predicting an output with less than ϵ accuracy for regression,
- `params` is a structure containing additional optional parameters such as the kernel function (default is the TK function), kernel parameters (degree of monomial basis for TK or GPK kernels), the domain of integration, the ϵ-loss term for regression, the maximum number of iterations and the tolerance,
- The output `f` is an internal data structure containing the solutions P^* and α^*, as well as the other user selected parameters. This data structure defines the resulting regressor/classifier. This regressor/classifier can be evaluated using the `EvaluatePMKL` command as described below.

Default parameters of the params structure are

1. `PMKL_Boosted` is the combined Frank-Wolfe and Accelerated Primal Dual method that is used when high accuracy is required. The algorithm usage is identical to the PMKL algorithm and can be used with these same instructions.

```
>> params.kernel  = 'TK'
>> params.delta  = .5
>> params.epsilon = .1
>> params.maxit = 100
>> params.tol = .01
```

where `kernel` specifies the kernel function to use, `delta` determines the bounds of integration $[a, b] = [0 - \delta, 1 + \delta]^n$, `epsilon` is the epsilon-loss of the support vector regression problem, `maxit` is the maximum number of iterations, and `tol` is the stopping tolerance. The `PMKL.m` function can be run with only some of the inputs manually specified, as discussed next.

Default Implementation
To run the PMKL algorithm with all default values for the samples `x` and outputs `y` and to automatically select the type of problem (classification or regression) the MATLAB command is,

```
>> f = PMKL(x,y)
```

where if `y` only contains two unique values, the algorithm defaults to classification. Otherwise, the algorithm defaults to regression.

Manual Selection of Classification or Regression
To run the PMKL algorithm with all default values, for the samples `x` and outputs `y` but manually select the type of problem, the MATLAB command is,

```
>> f = PMKL(x,y,Type)
```

where `Type = 'Classification'` for classification or `'Regression'` for regression.

Manually Specifying the Penalty C
To run the PMKL algorithm with all default values except for Type and the penalty term `C`, for the samples `x` and outputs `y` the MATLAB command is,

```
>> f = PMKL(x,y,Type,C)
```

where the user must select a $\texttt{C} > 0$. It is recommended that the value of `C` be selected via k-fold cross-validation with data split into training and validation sets.

Manually Specifying Additional Parameters
For help generating the params structure we have included the paramsTK.m function which allows the user to generate a params structure for TK kernels as follows.

```
>> params = paramsTK(degree,delta,epsilon,maxit,tol)
```

An empty matrix can be used for any input where the default value is desired, and using the paramsTK framework is recommended when modifying the default values.

The Evaluate Subroutine Once the optimal kernel function has been learned, you can evaluate the predicted output of a set of samples using the following function.

```
>> yPred = evaluatePMKL(f,xTest)
```

The output of `evaluatePMKL` are the predicted outputs of the optimal support vector machine, trained on the data x and y with the designated kernel function optimized by the PMKL subroutine.

- The input `f` is an internal data structure output from the PMKL function.
- `xTest` $\in \mathbb{R}^{n\times m}$ is a matrix of n rows corresponding to the number of features and m columns corresponding to the number of samples where `xTest(:,i)` is the i'th sample,
- The output `yPred` $\in \mathbb{R}^{1\times m}$ is a vector with m columns corresponding to the number of samples in `xTest`.

References

A Alzahrani and S Sadaoui. Scraping and preprocessing commercial auction data for fraud classification. arXiv preprint arXiv:1806.00656, 2018.

A. Alzahrani and S. Sadaoui. Clustering and labeling auction fraud data. In Data Management, Analytics and Innovation, pages 269–283. Springer, 2020.

J. Becker, D. Sandwell, W. Smith, J. Braud, B. Binder, J. Depner, D. Fabre, J. Factor, S. Ingalls, S. Kim, et al. Global bathymetry and elevation data at 30 arc seconds resolution: SRTM30_PLUS. Marine Geodesy, 32(4):355–371, 2009.

D. Bertsekas. Nonlinear Programming. Athena Scientific Optimization and Computation Series. Athena Scientific, 2016.

B. Boehmke and B.M. Greenwell. Hands-On Machine Learning with R. Chapman & Hall/CRC The R Series. CRC Press, 2019.

K. Borgwardt, A. Gretton, M. Rasch, H. Kriegel, B. Schölkopf, and A. Smola. Integrating structured biological data by kernel maximum mean discrepancy. Bioinformatics, 2006.

L. Breiman. Random forests. Machine Learning, 45:5–32, 2004.

B. Brian and J. G. Young. Implementation of a primal–dual method for SDP on a shared memory parallel architecture. Computational Optimization and Applications, 37(3):355–369, 2007.

T. Brooks, D. Pope, and M. Marcolini. Airfoil Self-noise and Prediction. NASA reference publication. National Aeronautics and Space Administration, Office of Management, Scientific and Technical Information Division, 1989.

A. Chambolle and T. Pock. On the ergodic convergence rates of a first-order primal–dual algorithm. Mathematical Programming, 159(1):253–287, 2016.

C-C. Chang and C-J. Lin. LIBSVM: a library for support vector machines. ACM Transactions on Intelligent Systems and Technology, 2:27:1–27:27, 2011.

T. Chen and C. Guestrin. XGBoost: a scalable tree boosting system. In ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016.

B. Colbert and M. Peet. A convex parametrization of a new class of universal kernel functions. Journal of Machine Learning Research, 21(45):1–29, 2020.

B. Colbert, A. Talitckii, and M. Peet. Tessellated kernel learning. https://github.com/CyberneticSCL/TKL-version-0.9, 2021.

C. Cortes, M. Mohri, and A. Rostamizadeh. Two-stage learning kernel algorithms. In International Conference on Machine Learning, 2010.

D. Dua and C. Graff. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

K. Fan. Minimax theorems. Proceedings of the National Academy of Sciences of the United States of America, 39(1):42, 1953.

Y. Fang, K. Loparo, and X. Feng. Inequalities for the trace of matrix product. IEEE Transactions on Automatic Control, 39(12):2489–2490, 1994.

M. Frank and P. Wolfe. An algorithm for quadratic programming. Naval Research Logistics Quarterly, 3(1-2):95–110, 1956.

K. Fukumizu, A. Gretton, X. Sun, and B. Schölkopf. Kernel measures of conditional dependence. Advances in Neural Information Processing Systems, 20, 2007.

I. Gohberg, S. Goldberg, and M. Kaashoek. Basic Classes of Linear Operators. Birkhäuser Basel, 2012.

M. Gönen and E. Alpaydın. Multiple kernel learning algorithms. Journal of Machine Learning Research, 2011.

E. Hamedani and N. Aybat. A primal-dual algorithm with line search for general convex-concave saddle point problems. SIAM Journal on Optimization, 31(2):1299–1329, 2021.

K. Harada. Positive semidefinite matrix approximation with a trace constraint. 2018. URL http://www.optimization-online.org/DB_FILE/2018/08/6765.pdf.

G. Hardy, J. Littlewood, and G. Pólya. Inequalities. Cambridge Mathematical Library. Cambridge University Press, 1952.

D. Harrison and D. Rubinfeld. Hedonic housing prices and the demand for clean air. Journal of Environmental Economics and Management, 1978.

T. Ho and E. Kleinberg. Building projectable classifiers of arbitrary complexity. In International Conference on Pattern Recognition, volume 2, 1996.

J. Hunter and B. Nachtergaele. Applied Analysis. World Scientific, 2001.

M. Hussain, S. Wajid, A. Elzaart, and M. Berbar. A comparison of SVM kernel functions for breast cancer detection. In International Conference Computer Graphics, Imaging and Visualization, pages 145–150. IEEE, 2011.

M. Jaggi. Revisiting Frank-Wolfe: Projection-free sparse convex optimization. In International Conference on Machine Learning, 2013.

A. Jain, S. Vishwanathan, and M. Varma. SPF-GMKL: generalized multiple kernel learning with a million kernels. In ACM International Conference on Knowledge Discovery and Data Mining, 2012.

H. Kaya, P. Tüfekci, and F. Gürgen. Local and global learning methods for predicting power of a combined gas & steam turbine. In International Conference on Emerging Trends in Computer and Electronics Engineering, pages 13–18, 2012.

H. Kaya, P. Tüfekci, and E. Uzun. Predicting CO and NOx emissions from gas turbines: novel data and a benchmark PEMS. Turkish Journal of Electrical Engineering & Computer Sciences, 27(6):4783–4796, 2019.

G. Lanckriet, N. Cristianini, P. Bartlett, L. El Ghaoui, and M. Jordan. Learning the kernel matrix with semidefinite programming. Journal of Machine Learning Research, 2004.

I. Lauriola and F. Aiolli. MKLpy: a python-based framework for multiple kernel learning. arXiv preprint arXiv:2007.09982, 2020.

J. McDermott and R. Forsyth. Diagnosing a disorder in a classification benchmark. Pattern Recognition Letters, 73:41–43, 2016.

C. Micchelli, Y. Xu, and H. Zhang. Universal kernels. Journal of Machine Learning Research, 2006.

K. Ni, S. Kumar, and T. Nguyen. Learning the kernel matrix for superresolution. In Proceedings of the IEEE Workshop on Multimedia Signal Processing, pages 441–446, 2006.

K. Pace and R. Barry. Quick computation of spatial autoregressive estimators. Geographical Analysis, 1997a.

K. Pace and R. Barry. Sparse spatial autoregressions. Statistics & Probability Letters, 1997b.

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

S. Qiu and T. Lane. Multiple kernel learning for support vector regression. Computer Science Department, The University of New Mexico, Albuquerque, NM, USA, Tech. Rep, 2005.

A. Rakotomamonjy, F. R. Bach, S. Canu, and Y. Grandvalet. SimpleMKL. Journal of Machine Learning Research, 2008.

B. Schölkopf, R. Herbrich, and A. Smola. A generalized representer theorem. In International Conference on Computational Learning Theory, pages 416–426, 2001.

C-J Simon-Gabriel and B. Schölkopf. Kernel distribution embeddings: universal kernels, characteristic kernels and kernel metrics on distributions. Journal of Machine Learning Research, 19(1):1708–1736, 2018.

A. Smola and B. Schölkopf. A tutorial on support vector regression. Statistics and Computing, 14(3):199–222, 2004.

S. Sonnenburg, G. Rätsch, S. Henschel, C. Widmer, J. Behr, A. Zien, F. De Bona, A. Binder, C. Gehl, and V. Franc. The SHOGUN machine learning toolbox. Journal of Machine Learning Research, 11(60):1799–1802, 2010.

B. Sriperumbudur, K. Fukumizu, and G. Lanckriet. Universality, characteristic kernels and RKHS embedding of measures. Journal of Machine Learning Research, 12(7), 2011.

I. Steinwart. On the influence of the kernel on the consistency of support vector machines. Journal of Machine Learning Research, 2(Nov):67–93, 2001.

I. Steinwart and A. Christmann. Support Vector Machines. Information Science and Statistics. Springer New York, 2008.

H. Tanabe, B.T. Ho, C.H. Nguyen, and S. Kawasaki. Simple but effective methods for combining kernels in computational biology. In International Conference on Research, Innovation and Vision for the Future in Computing and Communication Technologies, 2008.

P. Tüfekci. Prediction of full load electrical power output of a base load operated combined cycle power plant using machine learning methods. International Journal of Electrical Power & Energy Systems, 60:126–140, 2014.

H. Wang, Q. Xiao, and D. Zhou. An approximation theory approach to learning with ℓ_1 regularization. Journal of Approximation Theory, 2013.

S. Waugh. Extending and benchmarking Cascade-Correlation: extensions to the Cascade-Correlation architecture and benchmarking of feed-forward supervised artificial neural networks. PhD thesis, University of Tasmania, 1995.

S. Willard. General Topology. Dover Books on Mathematics. Dover Publications, 2012.

W. Wolberg, O. Mangasarian, T. Coleman, and Y. Li. Pattern recognition via linear programming: Theory and application to medical diagnosis. Large-Scale Numerical Optimization, SIAM Publications, Citeseer, pages 22–30, 1990.

W. Wolberg, O. Mangasarian, N. Street, and W. Street. Breast Cancer Wisconsin (Diagnostic). UCI Machine Learning Repository, 1995.

Z. Xu, R. Jin, H. Yang, I. King, and M.R. Lyu. Simple and efficient multiple kernel learning by group Lasso. In International Conference on Machine Learning, pages 1175–1182, 2010.

H. Yang, Z. Xu, J. Ye, I. King, and M.R. Lyu. Efficient sparse generalized multiple kernel learning. IEEE Transactions on Neural Networks, 22(3):433–446, 2011.

Y. Ye and E. Tse. An extension of Karmarkar's projective algorithm for convex quadratic programming. Mathematical Programming, 44(1-3):157–179, 1989.

I. Yeh, K. Yang, and T. Ting. Knowledge discovery on RFM model using Bernoulli sequence. Expert Systems with Applications, 36, 2009.