Fast Kronecker product kernel methods via sampled vec trick

Antti Airola, Tapio Pahikkala

Abstract

Kronecker product kernel provides the standard approach in the kernel methods literature for learning from pair-input data, where both data points and prediction tasks have their own feature representations. The methods allow simultaneous generalization to both new tasks and data unobserved in the training set, a setting known as zero-shot or zero-data learning. Such a setting occurs in numerous applications, including drug-target interaction prediction, collaborative filtering and information retrieval. Efficient training algorithms based on the so-called vec trick, that makes use of the special structure of the Kronecker product, are known for the case where the output matrix for the training set is fully observed, i.e. the correct output for each data point-task combination is available. In this work we generalize these results, proposing an efficient algorithm for sampled Kronecker product multiplication, where only a subset of the full Kronecker product is computed. This allows us to derive a general framework for training Kronecker kernel methods, as specific examples we implement Kronecker ridge regression and support vector machine algorithms. Experimental results demonstrate that the proposed approach leads to accurate models, while allowing order of magnitude improvements in training and prediction time.

1 Introduction

This work concerns the problem of how to efficiently learn supervised machine learning models from pair-input data. By pair-input, we mean that the inputs can be split to two parts (each having its own feature representation), and real-valued outputs that are known for training inputs, but need to be predicted for future inputs. Further, the pair-input data tend to appear in such sets, where the same parts will in different combinations form several different inputs, meaning that the data violates the i.i.d. assumption commonly made in the field of machine learning. One of the major applications of this setting is the zero-shot or zero-data setting in transfer learning, where both data points and tasks have their own feature representations, and one needs to predict an output value for a new data point given the new task (see e.g. [1], [2], [3], [4]). For example, in a multi-class learning setting one can make predictions for classes not present in the training set, if the classes have feature representations, whereas in a multi-task ranking setting one may need to rank data points with respect to novel tasks. Even though often not recognized as transfer learning problems, such settings occur in a wide variety of machine learning applications. Typical examples include biochemical interaction prediction, where the inputs can be split, for instance, to drugs and targets (see e.g. [5] for a recent review), recommender systems, where the inputs consist of customers and products [6] and information retrieval, where they consist of queries and documents [7]. In these problems, both parts of the inputs tend to appear several times in the training set, for example, the same drug may have been tested for interaction with several targets and vice versa.

The overall learning problem is described in Figure 1. Let us assume, that our inputs can be naturally split into two parts, in this paper referred to as the data point and task parts, of which both have their own feature representations. Let D denote the in-sample data points (corresponding to the rows of the in-sample output matrix in Figure 1). Further, let T denote the in-sample tasks (corresponding to the columns of the in-sample output matrix in Figure 1). By in-sample, we mean that the data point or task is encountered in the training set at least once. Now let $\mathbf{x} = (\mathbf{d}, \mathbf{t})$ denote a new input, for which the correct output (typically class or real number) is unknown and needs to be predicted. In this work we focus on the most challenging setting, where it is assumed that $\mathbf{d} \notin D$ and $\mathbf{t} \notin T$, that is, neither the data or the task part has been previously observed in the training set. This corresponds to zero-shot learning, where it is necessary to generalize simultaneously both to such new tasks and data that are unknown during training time. In Figure 1, this is represented as predicting the unknown target output at the bottom right.

There exist two related yet distinct learning problems, that have been studied in-depth in the machine learning literature. First, if $\mathbf{d} \in D$ and $\mathbf{t} \in T$, the problem becomes that of matrix completion (predicting the the unknown outputs within the center rectangle in Figure 1). Recently this setting has been considered especially in the recommender systems literature, where matrix factorization methods that generate latent feature representations have become the de-facto standard approach for solving these problems (see e.g. [8]). The second setting corresponds to the situation where $\mathbf{d} \in D$ and $\mathbf{t} \notin T$, or alternatively $\mathbf{d} \notin D$ and $\mathbf{t} \in T$ (predicting the block on the right or

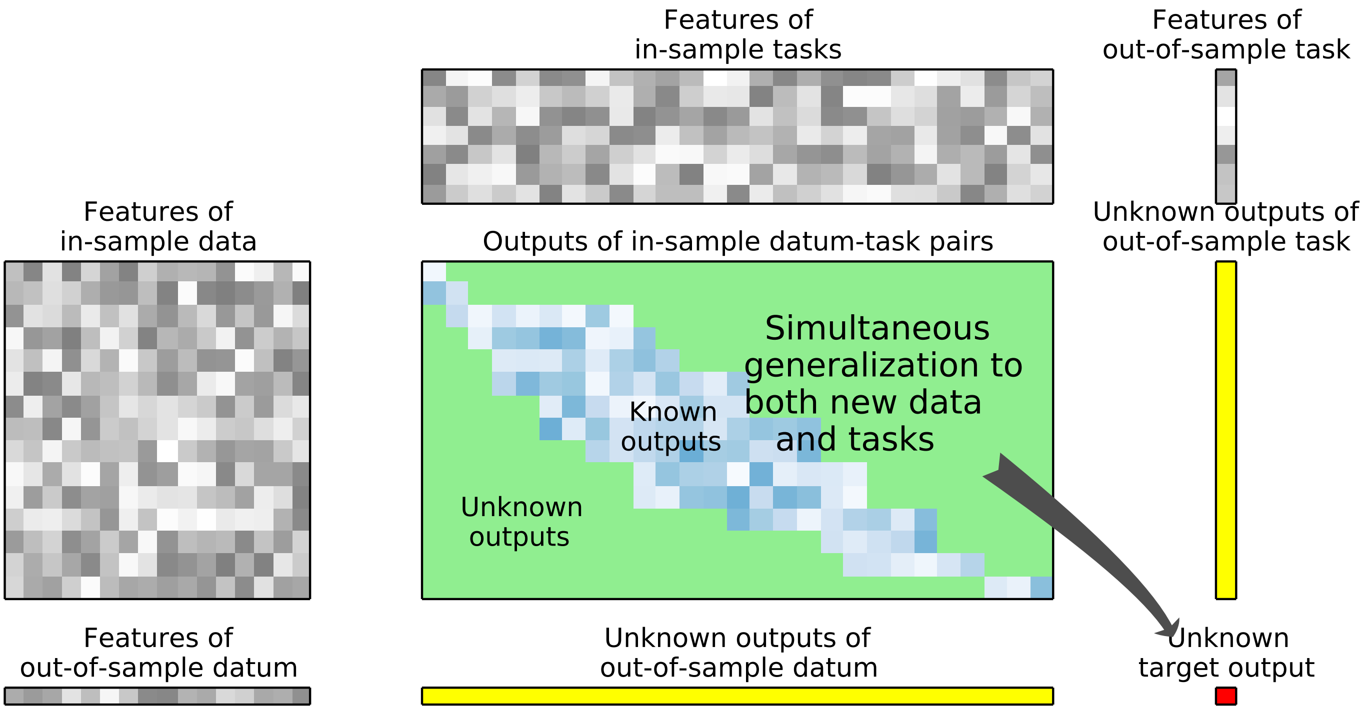

Fig. 1. Zero-shot learning. A predictor trained on (datum, task)-pairs and associated correct outputs generalizes out-of-sample to such a new pair, for which both the datum and task are not part of the training set.

below in Figure 1). Here predictions are needed only for new rows or columns of the output matrix, but not both simultaneously. The problem may be solved by training a standard supervised learning method for each row or column separately, or by using multi-target learning methods that also aim to utilize correlations between similar rows or columns [9]. While simultaneous use of both datum and task features can and has been integrated to learning also in these two related settings in order to increase prediction accuracy, they are not essential for generalization, and can even be harmful (so called negative transfer, see e.g. [10], [11] for experimental results). The learning algorithms considered in this work are also applicable in these two related settings, provided that both task and data specific features are available. However, simpler methods that do not need both of these information sources may often provide a competitive alternative. Thus we do not consider further these two settings in this work, as these have been already quite thoroughly explored in previous literature, and we do not expect the proposed methods to offer major advantages there beyond existing state-of-the-art.

In this work we assume that both the tasks and data points have feature representations defined implicitly via a kernel function [12]. Further, following works such as [6], [13], [14], [15], [16], [17], [18], [19], [20], [10], [4], we assume that their joint feature representation is defined by the Kronecker product kernel, which is the product of the data and task kernels. A significant advantage of this approach is that if the data point and task kernels are universal (e.g. the Gaussian kernel), the Kronecker kernel is also universal, meaning that it allows representing arbitrarily complex non-linear functions [19].

Several efficient machine learning algorithms have been proposed for the special case, where the training set contains the outputs for each possible datum-task pair in the training set exactly once (i.e. all the values in the center output matrix in Figure 1 are known). For minimizing ridge regression types of losses [17], [18], [20], [10], [4] derive closed form solutions based on Kronecker algebraic optimization (see also [21] for the basic mathematical results underlying these studies). Further, iterative methods based on Kronecker kernel matrix - vector multiplications, have been proposed (see e.g. [14], [18], [20]).

However, the requirement that the outputs are known for each datum-task pair in $D \times T$ can be considered a major limitation on the applicability of these methods. As far as we are aware, similar efficient training algorithms for regularized kernel methods have not been previously proposed for the case, when the training data is sparse in the sense that only a subset of the datum-task pairs in $D \times T$ has a known output in the training set. Kernel methods can still be trained also in this setting by explicitly constructing a kernel matrix that contains one entry for each pair-input. However, this approach soon becomes computationally non-feasible as the number of pair-inputs grows. In this work, we show how to substantially increase the computational efficiency of kernel methods in this setting, assuming that same data points and tasks appear several times in the training set.

Algorithm 1 Compute $\mathbf{u} \leftarrow \mathbf{R}(\mathbf{M} \otimes \mathbf{N})\mathbf{C}^{\mathrm{T}}\mathbf{v}$

Require: $\mathbf{M} \in \mathbb{R}^{a \times b}, \mathbf{N} \in \mathbb{R}^{c \times d}, \mathbf{v} \in \mathbb{R}^{e}, \mathbf{p} \in [a]^{f}, \mathbf{q} \in [c]^{f}, \mathbf{r} \in [b]^{e}, \mathbf{t} \in [d]^{e}$

if $ae + df < ce + bf$ **then**
  $\mathbf{T} \leftarrow \mathbf{0} \in \mathbb{R}^{d \times a}$
  for $h = 1, \ldots, e$ **do**
    $i, j \leftarrow r_h, t_h$
    for $k = 1, \ldots, a$ **do**
      $T_{j,k} \leftarrow T_{j,k} + v_h M_{k,i}$
  $\mathbf{u} \leftarrow \mathbf{0} \in \mathbb{R}^{f}$
  for $h = 1, \ldots, f$ **do**
    $i, j \leftarrow p_h, q_h$
    for $k = 1, \ldots, d$ **do**
      $u_h \leftarrow u_h + N_{j,k} T_{k,i}$
else
  $\mathbf{S} \leftarrow \mathbf{0} \in \mathbb{R}^{c \times b}$
  for $h = 1, \ldots, e$ **do**
    $i, j \leftarrow r_h, t_h$
    for $k = 1, \ldots, c$ **do**
      $S_{k,i} \leftarrow S_{k,i} + v_h N_{k,j}$
  $\mathbf{u} \leftarrow \mathbf{0} \in \mathbb{R}^{f}$
  for $h = 1, \ldots, f$ **do**
    $i, j \leftarrow p_h, q_h$
    for $k = 1, \ldots, b$ **do**
      $u_h \leftarrow u_h + S_{j,k} M_{i,k}$
return u

We generalize the computational short-cut implied by Roth's column lemma [22] (often known as vec-trick) to the sampled Kronecker product, where only a subset of the rows and columns of the full Kronecker product matrix is needed. Based on this result we derive efficient training algorithms for Kronecker kernel based regularized kernel methods. As case studies, we show how to apply this framework for training ridge regression and support vector machine methods; extending the results to other types of loss functions or optimization methods can be done in a similar manner. In addition to the computational complexity analysis we also experimentally demonstrate, that the proposed methods are the current state-of-the art in learning with Kronecker kernels. This work extends our previous work [23], that specifically considered the case of efficient ridge regression with Kronecker product kernel.

2 Computation with Sampled Kronecker products

First, let us introduce some notation. By $[n]$, where $n \in \mathbb{N}$, we denote the set $\{1, ..., n\}$. By $\mathbf{A} \in \mathbb{R}^{a \times b}$ we denote a $a \times b$ matrix, and by $A_{i,j}$ the i, j:th element of this matrix. By $\mathrm{vec}(\mathbf{A})$ we denote the vectorization of $\mathbf{A}$, which is the $ab \times 1$ column vector obtained by stacking all the columns of $\mathbf{A}$ in order starting from the first column. $\mathbf{A} \otimes \mathbf{C}$ denotes the Kronecker product of $\mathbf{A}$ and $\mathbf{C}$. By $\mathbf{a} \in \mathbb{R}^{c}$ we denote a column vector of size $c \times 1$, and by a_i the i:th element of this vector.

There exists several studies in the machine learning literature in which the systems of linear equations involving Kronecker products have been accelerated with the so-called "vec-trick". This is characterized by the following result known as the Roth's column lemma in the Kronecker product algebra:

Lemma 1 ([22]). *Let $\mathbf{P} \in \mathbb{R}^{a \times b}$, $\mathbf{Q} \in \mathbb{R}^{b \times c}$, and $\mathbf{R} \in \mathbb{R}^{c \times d}$ be matrices. Then,*

$$(\mathbf{R}^{\mathrm{T}} \otimes \mathbf{P})\mathrm{vec}(\mathbf{Q}) = \mathrm{vec}(\mathbf{PQR}). \tag{1}$$

It is obvious that the right hand size of (1) is considerably faster to compute than the left hand side, because it avoids the direct computation of the large Kronecker product.

We next shift our consideration to matrices that can not be directly expressed as Kronecker products but whose rows and columns are sampled from such a product matrix. First, we introduce some further notation. To sample a sequence of rows or columns from a matrix with replacement, the following construction is widely used (see e.g. [24]):

Definition 1 (Sampling matrix)**.** *Let* $\mathbf{M} \in \mathbb{R}^{a \times b}$ *and let* $\mathbf{s} = (s_1, \ldots, s_f)^{\mathrm{T}} \in [a]^f$ *be a sequence of* f *row indices of* $\mathbf{M}$ *sampled with replacement. We say that*

$$\mathbf{S} = \begin{pmatrix} \mathbf{e}_{s_1}^{\mathrm{T}} \\ \vdots \\ \mathbf{e}_{s_f}^{\mathrm{T}} \end{pmatrix} ,$$

where $\mathbf{e}_i$ *is the* i*th standard basis vector of* $\mathbb{R}^a$*, is a row sampling matrix for* $\mathbf{M}$ *determined by* $\mathbf{s}$*. The column sampling matrices are defined analogously.*

The above type of ordinary sampling matrices can of course be used to sample the Kronecker product matrices as well. However, the forthcoming computational complexity considerations require a more detailed construction in which we refer to the indices of the factor matrices rather than to those of the product matrix. This is characterized by the following lemma which follows directly from the properties of the Kronecker product:

Lemma 2 (Kronecker product sampling matrix)**.** *Let* $\mathbf{M} \in \mathbb{R}^{a \times b}$*,* $\mathbf{N} \in \mathbb{R}^{c \times d}$ *and let* $\mathbf{S}$ *be a sampling matrix for the Kronecker product* $\mathbf{M} \otimes \mathbf{N}$*. Then,* $\mathbf{S}$ *can be expressed as*

$$\mathbf{S} = \begin{pmatrix} \mathbf{e}_{(p_1-1)c+q_1}^{\mathrm{T}} \\ \vdots \\ \mathbf{e}_{(p_f-1)c+q_f}^{\mathrm{T}} \end{pmatrix} ,$$

where $\mathbf{p} = (p_1, \ldots, p_f)^{\mathrm{T}} \in [a]^f$ *and* $\mathbf{q} = (q_1, \ldots, q_f)^{\mathrm{T}} \in [c]^f$ *are sequences of row indices of* $\mathbf{M}$ *and* $\mathbf{N}$*, respectively, which are sampled with replacement. The entries of* $\mathbf{p}$ *and* $\mathbf{q}$ *are given as*

$$p = \lfloor (i-1)/c \rfloor + 1 \tag{2}$$

$$q = (i-1) \mod c + 1 , \tag{3}$$

where $i \in [ac]$ denotes a row index of $\mathbf{M} \otimes \mathbf{N}$, and $p \in [a]$ and $q \in [c]$ map to the corresponding rows in $\mathbf{M}$ and $\mathbf{N}$, respectively.

Proof: The claim is an immediate consequence of the definition of the Kronecker product, where the relationship between the row indices of $\mathbf{M}$ and $\mathbf{N}$ with those of their Kronecker product is exactly the one given in (2) and (3). □

Definition 2 (Sampled Kronecker product)**.** *Let* $\mathbf{M} \in \mathbb{R}^{a \times b}, \mathbf{N} \in \mathbb{R}^{c \times d}$*, and let* $\mathbf{R} \in \{0,1\}^{f \times ac}$ *and* $\mathbf{C} \in \{0,1\}^{e \times bd}$ *to be, respectively, a row sampling matrix and a column sampling matrix of* $\mathbf{M} \otimes \mathbf{N}$*, such that* $\mathbf{R}$ *is determined by the sequences* $\mathbf{p} = (p_1, \ldots, p_f)^{\mathrm{T}} \in [a]^f$ *and* $\mathbf{q} = (q_1, \ldots, q_f)^{\mathrm{T}} \in [c]^f$*, and* $\mathbf{C}$ *by* $\mathbf{r} = (r_1, \ldots, r_e)^{\mathrm{T}} \in [b]^e$ *and* $\mathbf{t} = (t_1, \ldots, t_e)^{\mathrm{T}} \in [d]^e$*. Further, we assume that the sequences* $\mathbf{p}$*,* $\mathbf{q}$*,* $\mathbf{r}$ *and* $\mathbf{t}$ *when considered as mappings, are all surjective. We call*

$$\mathbf{R}(\mathbf{M} \otimes \mathbf{N})\mathbf{C}^{\mathrm{T}}$$

the sampled Kronecker product matrix. The surjectivity requirement ensures that all entries of $\mathbf{M}$ *and* $\mathbf{N}$ *are factors of the sampled Kronecker product matrix entries. While, this in principle restricts the concept of sampled Kronecker product matrix, it can still be done without losing generality, because the matrices* $\mathbf{M}$ *and* $\mathbf{N}$ *can be replaced with their appropriate submatrices omitting the unnecessary rows and columns.*

Proposition 1. *Let* $\mathbf{R}$*,* $\mathbf{M}$*,* $\mathbf{N}$ *and* $\mathbf{C}$ *be as in Definition 2, and* $\mathbf{v} \in \mathbb{R}^e$*. The product*

$$\mathbf{R}(\mathbf{M} \otimes \mathbf{N})\mathbf{C}^{\mathrm{T}}\mathbf{v} \tag{4}$$

can be computed in $O(\min(ae + df, ce + bf))$ *time.*

Proof: Since the sequences $\mathbf{p}$, $\mathbf{q}$, $\mathbf{r}$ and $\mathbf{t}$, determining $\mathbf{R}$ and $\mathbf{C}$ are all surjective on their co-domains, we imply that $\max(a, c) \leq f$ and $\max(b, d) \leq e$.

Let $\mathbf{V} \in R^{d \times b}$ be a matrix such that $\mathrm{vec}(\mathbf{V}) = \mathbf{C}^{\mathrm{T}}\mathbf{v}$. Then, according to Lemma 1, (4) can be rewritten as

$$\mathbf{R}\mathrm{vec}(\mathbf{N}\mathbf{V}\mathbf{M}^{\mathrm{T}}) ,$$

which does not involve a Kronecker product. Calculating the right hand side can be started by first computing either $\mathbf{S} = \mathbf{N}\mathbf{V}$ or $\mathbf{T} = \mathbf{V}\mathbf{M}^{\mathrm{T}}$ requiring $O(ce)$ and $O(ae)$ time, respectively, due to $\mathbf{V}$ having only at most e nonzero entries. Then, each of the f elements of the product can be computed either by calculating the inner product between a row of $\mathbf{S}$ and a row of $\mathbf{M}$ or between a row of $\mathbf{N}$ and a column of $\mathbf{T}$, depending whether $\mathbf{S}$ or $\mathbf{T}$ was computed. The former inner products require b and the latter d flops, and hence the overall complexity is $O(\min(ae + df, ce + bf))$. The pseudocode for the algorithm following the ideas of the proof is presented in Algorithm 1. □

Remark 1 (Sampling with replacement)**.** *If either the rows or columns of the sampled Kronecker product contain duplicates, the above complexity can be improved even further. Namely, the variables e and f then correspond to the numbers of distinct rows and columns. This can be achieved by replacing the original sampled Kronecker product with a one that is done by sampling without replacement so that the vector $\mathbf{v}$ contains only a single entry per row of the Kronecker product matrix such that consists of the sum of all entries of the original $\mathbf{v}$ associated with the same row. After running Algorithm 1, entries of $\mathbf{u}$ can be duplicated as many times as indicated by the original setting.*

Remark 2 (Roth's column lemma as a special case)**.** *If $\mathbf{R} = \mathbf{C} = \mathbf{I}$, the sampled Kronecker product reduces to the standard Kronecker product with no sampling, and the complexity of Algorithm 1 is $O(\min(abd + acd, bcd + abc))$, which is the same as the complexity of computing the product directly based on Roth's column lemma (1).*

3 General framework for sampled Kronecker product kernel methods

In this section, we connect the above abstract considerations with practical learning problems with pair-input data. We start by defining some notation which will be used through the forthcoming considerations. As observed from Figure 1, the relevant data dimensions are the sizes of the two data matrices and the number of inputs with known outputs. The notations are summarized in the following table:

n	# of inputs (task, datum) and outputs in training set.
m	# of data points encountered in training set.
q	# of tasks encountered in training set.
d	# of data features.
r	# of task features.

The training data consists of a sequence $S = (\mathbf{x}_h)_{h=1}^n \in \mathcal{X}^n$ of inputs, $\mathcal{X}$ being the set of all possible inputs, and a sequence of associated outputs $\mathbf{y}$. The most common settings are regression where $\mathbf{y} \in \mathbb{R}^n$ and binary classification where $\mathbf{y} \in \{-1, 1\}^n$, though the output space may also have more complicated structure, for example in structured prediction problems. As described above, we assume each input can be represented as a pair consisting of a data point and task $\mathbf{x} = (\mathbf{d}, \mathbf{t}) \in \mathcal{D} \times \mathcal{T}$, where $\mathcal{D}$ and $\mathcal{T}$ are the sets of all possible data points and tasks, respectively, to which we refer as the data point space and the task space. Moreover, let $D \subset \mathcal{D}$ and $T \subset \mathcal{T}$ denote the in-sample data points and in-sample tasks, that is, the sets of data points and tasks encountered in the training sequence, respectively, and let $m = |D|$ and $q = |T|$. Finally, let $\mathbf{r} = (r_1, \ldots, r_n)^{\mathrm{T}} \in [q]^n$ and $\mathbf{s} = (s_1, \ldots, s_n)^{\mathrm{T}} \in [m]^n$ represent the index sequences for the training data points and tasks, and $\mathbf{R} \in \{0, 1\}^{n \times mq}$ be the Kronecker product sampling matrix determined by these index sequences.

Let $k : \mathcal{D} \times \mathcal{D} \to [0, \infty)$ and $g : \mathcal{T} \times \mathcal{T} \to [0, \infty)$ denote positive semi-definite kernel functions [12] defined for the data points and tasks, respectively. Further, the Kronecker product kernel $k^{\otimes}(\mathbf{d}, \mathbf{t}, \mathbf{d}', \mathbf{t}') = k(\mathbf{d}, \mathbf{d}')g(\mathbf{t}, \mathbf{t}')$ is defined as the product of these two base kernels. Let $K_{i,j} = k(\mathbf{d}_i, \mathbf{d}_j)$ and $G_{i,j} = g(\mathbf{t}_i, \mathbf{t}_j)$ denote the data- and task kernel matrices for the training data, respectively, then $\mathbf{R}(\mathbf{G} \otimes \mathbf{K})\mathbf{R}^{\mathrm{T}}$ is the Kronecker kernel matrix containing the kernel evaluations corresponding to the training inputs in S.

We consider the regularized risk minimization problem

$$f^* = \operatorname*{argmin}_{f \in \mathcal{H}} J(f) \tag{5}$$

with

$$J(f) = \mathcal{L}((f(\mathbf{x}_1), ..., f(\mathbf{x}_n)), \mathbf{y}) + \frac{\lambda}{2}\|f\|_{\mathcal{H}}^2$$

with $\mathcal{L}$ a convex nonnegative loss function and $\lambda > 0$ a regularization parameter. Choosing as $\mathcal{H}$ the reproducing kernel Hilbert space defined by $k^{\otimes}$, the generalized representer theorem [25] guarantees that the optimal solutions are of the form

$$f^*(\mathbf{d}, \mathbf{t}) = \sum_{i=1}^{n} a_i k(\mathbf{d}_{r_i}, \mathbf{d})g(\mathbf{t}_{s_i}, \mathbf{t}),$$

where we call $\mathbf{a} \in \mathbb{R}^n$ the vector of dual coefficients.

Now let us consider the special case, where the data point space and the task space are real vector spaces, that is, $\mathcal{D} = \mathbb{R}^d$ and $\mathcal{T} = \mathbb{R}^r$, and hence both the data points and tasks have a finite dimensional feature representation. Further, if the data- and task kernels are linear, the Kronecker kernel can be written open as the inner product $k^{\otimes}(\mathbf{d}, \mathbf{t}, \mathbf{d}', \mathbf{t}') = \langle \mathbf{d} \otimes \mathbf{t}, \mathbf{d}' \otimes \mathbf{t}' \rangle$, that is the (datum, task) -pairs have an explicit feature representation with respect to this kernel as the Kronecker product of the two feature vectors. Let $\mathbf{D} \in \mathbb{R}^{m \times d}$ and $\mathbf{T} \in \mathbb{R}^{q \times r}$, respectively, contain the feature representations of the in-sample data points and tasks. Now the joint Kronecker feature representation

TABLE 1
Loss functions and their (sub)gradients and (generalized) Hessians. For binary classification losses marked * we assume that $y_i \in \{-1, 1\}$, while for the rest $y_i \in \mathbb{R}$.

Method	$\mathcal{L}$	$\mathbf{g}_i$	$\mathbf{H}_{i,i}$	$\mathbf{H}_{i,j}, i \neq j$
Ridge [26]	$\frac{1}{2}\sum_{i=1}^{n}(p_i - y_i)^2$	$p_i - y_i$	1	0
L1-SVM* [27]	$\sum_{i=1}^{n}\max(0, 1 - p_i \cdot y_i)$	$-y_i$ if $p_i \cdot y_i < 1$ 0 otherwise.	0	0
L2-SVM* [28]	$\frac{1}{2}\sum_{i=1}^{n}\max(0, 1 - p_i \cdot y_i)^2$	$p_i - y_i$ if $p_i \cdot y_i < 1$ 0 otherwise.	1 if $p_i \cdot y_i < 1$ 0 otherwise.	0
Logistic* [29]	$\sum_{i=1}^{n}\log(1 + e^{-y_i p_i})$	$-y_i(1 + e^{y_i p_i})^{-1}$	$e^{y_i p_i}(1 + e^{y_i p_i})^{-2}$	0
RankRLS [30], [31]	$\frac{1}{4}\sum_{i=1}^{n}\sum_{j=1}^{n}(y_i - p_i - y_j + p_j)^2$	$\sum_{j=1}^{n}(y_j - p_j) + n(p_i - y_i)$	$n - 1$	-1

for the training data can be expressed as $\mathbf{X} = \mathbf{R}(\mathbf{T} \otimes \mathbf{D})$. In this case we can equivalently define the regularized risk minimizer as

$$f^*(\mathbf{d}, \mathbf{t}) = \langle \mathbf{d} \otimes \mathbf{t}, \sum_{i=1}^{n} a_i \mathbf{d}_{r_i} \otimes \mathbf{t}_{s_i} \rangle = \langle \mathbf{d} \otimes \mathbf{t}, \mathbf{w} \rangle,$$

where we call $\mathbf{w} \in \mathbb{R}^{dr}$ the vector of primal coefficients.

3.1 Efficient prediction

Let $T = \{\mathbf{z}_h\}_{h=1}^{s} \in \mathcal{X}^s$ represent a set of new pair-inputs for which predictions are needed, with u and v denoting the number of data points and tasks in this set, respectively. Further, we overload our notation defining $f(T) = [f(\mathbf{z}_1), ..., f(\mathbf{z}_s)]^{\mathrm{T}}$.

Let $\hat{\mathbf{R}} \in \{0, 1\}^{s \times uv}$ denote the sampling matrix defined by these new pair-inputs set, and $\hat{\mathbf{K}} \in \mathbb{R}^{u \times m}$ and $\hat{\mathbf{G}} \in \mathbb{R}^{v \times q}$ contain the kernel evaluations between the training and new data points and tasks, respectively. For the dual model, the predictions can be computed as

$$f(T) = \hat{\mathbf{R}}(\hat{\mathbf{G}} \otimes \hat{\mathbf{K}})\mathbf{R}^{\mathrm{T}}\mathbf{a}$$

resulting in a computational complexity of $O(\min(vn + ms, un + qs))$. Further, we note that for sparse models where a large portion of the a_i coefficients have value 0 (most notably, support vector machine predictors), we can further substantially speed up prediction by removing these entries from a and $\mathbf{R}$ and correspondingly replace the term n in the complexity with the number of non-zero coefficients. In this case, the prediction complexity will be

$$O(\min(v\|\mathbf{a}\|_0 + ms, u\|\mathbf{a}\|_0 + qs)), \tag{6}$$

where $\|\mathbf{a}\|_0$ is the zero-norm measuring the number of non-zero elements in $\mathbf{a}$. This is in contrast to the explicit computation by forming the full test kernel matrix, which would result in

$$O(s\|\mathbf{a}\|_0) \tag{7}$$

complexity.

In the primal case the predictions can be computed as

$$f(T) = \hat{\mathbf{R}}(\hat{\mathbf{T}} \otimes \hat{\mathbf{D}})\mathbf{w}$$

resulting in a computational complexity of $O(\min(vdr + ds, udr + rs))$. For high-dimensional data ($dr >> n$) one will save substantial computation by using the dual model instead of the primal.

3.2 Efficient learning

The regularized risk minimization problem provides a convex minimization problem, whose optimum can be located with (sub)gradient information. Next, we show how to efficiently compute the gradient of the objective function, and for twice differentiable loss functions Hessian-vector products, when using the Kronecker kernel. For non-smooth losses or losses with non-smooth derivatives, we may instead consider subgradients and the generalized Hessian matrix (see e.g. [28], [32], [33]). In this section, we will make use of the denominator-layout notation.

First, let us consider making predictions on training data in the dual case. Based on previous considerations, we can compute

$$f(S) = \mathbf{p} = \mathbf{R}(\mathbf{G} \otimes \mathbf{K})\mathbf{R}^{\mathrm{T}}\mathbf{a}. \tag{8}$$

The regularizer can be computed as $\frac{\lambda}{2}\|f\|_{\mathcal{H}}^2 = \frac{\lambda}{2}\mathbf{a}^{\mathrm{T}}\mathbf{R}(\mathbf{G} \otimes \mathbf{K})\mathbf{R}^{\mathrm{T}}\mathbf{a}$.

Algorithm 2 Dual Truncated Newton optimization for regularized risk minimization

Require: $\mathbf{R} \in \{0,1\}^{n \times mq}$, $\mathbf{K} \in \mathbb{R}^{m \times m}$, $\mathbf{G} \in \mathbb{R}^{q \times q}$, $\mathbf{y} \in \mathbb{R}^n$, $\lambda > 0$

1: $\mathbf{a} \leftarrow \mathbf{0}$
2: **repeat**
3: $\quad\mathbf{p} \leftarrow \mathbf{R}(\mathbf{G} \otimes \mathbf{K})\mathbf{R}^{\mathrm{T}}\mathbf{a}$
4: $\quad$COMPUTE $\mathbf{H}$, $\mathbf{g}$
5: $\quad$SOLVE $(\mathbf{H}\mathbf{R}(\mathbf{G} \otimes \mathbf{K})\mathbf{R}^{\mathrm{T}} + \lambda\mathbf{I})\mathbf{x} = \mathbf{g} + \lambda\mathbf{a}$
6: $\quad\mathbf{a} \leftarrow \mathbf{a} - \delta\mathbf{x}$ (δ constant or found by line search)
7: **until** convergence

Algorithm 3 Primal Truncated Newton optimization for regularized risk minimization

Require: $\mathbf{R} \in \{0,1\}^{n \times mq}$, $\mathbf{D} \in \mathbb{R}^{m \times d}$, $\mathbf{T} \in \mathbb{R}^{q \times r}$, $\mathbf{y} \in \mathbb{R}^n$, $\lambda > 0$

1: $\mathbf{w} \leftarrow \mathbf{0}$
2: **repeat**
3: $\quad\mathbf{p} \leftarrow \mathbf{R}(\mathbf{T} \otimes \mathbf{D})\mathbf{w}$
4: $\quad$COMPUTE $\mathbf{H}$, $\mathbf{g}$
5: $\quad$SOLVE $((\mathbf{T}^{\mathrm{T}} \otimes \mathbf{D}^{\mathrm{T}})\mathbf{R}^{\mathrm{T}}\mathbf{H}\mathbf{R}(\mathbf{T} \otimes \mathbf{D}) + \lambda\mathbf{I})\mathbf{x} =$
$\quad(\mathbf{T}^{\mathrm{T}} \otimes \mathbf{D}^{\mathrm{T}})\mathbf{R}^{\mathrm{T}}\mathbf{g} + \lambda\mathbf{w}$.
6: $\quad\mathbf{w} \leftarrow \mathbf{w} - \delta\mathbf{x}$ (δ constant or found by line search)
7: **until** convergence

We use the following notation to denote the gradient and the Hessian (or a subgradient and/or generalized Hessian) of the loss function, with respect to $\mathbf{p}$:

$$\mathbf{g} = \frac{\partial \mathcal{L}}{\partial \mathbf{p}} \text{ and } \mathbf{H} = \frac{\partial^2 \mathcal{L}}{\partial \mathbf{p}^2}$$

The exact form of $\mathbf{g}$ and $\mathbf{H}$ depends on the loss function, Table 1 contains some common choices in machine learning. While maintaining the full $n \times n$ Hessian would typically not be computationally feasible, for univariate losses the matrix is diagonal. Further, for many multivariate losses efficient algorithms for computing Hessian-vector products are known (see e.g. [31], [34], [35] for examples of efficiently decomposable ranking losses).

The gradient of the empirical loss can be decomposed via chain rule as

$$\frac{\partial \mathcal{L}}{\partial \mathbf{a}} = \frac{\partial \mathbf{p}}{\partial \mathbf{a}}\frac{\partial \mathcal{L}}{\partial \mathbf{p}} = \mathbf{R}(\mathbf{G} \otimes \mathbf{K})\mathbf{R}^{\mathrm{T}}\mathbf{g}$$

The gradient of the regularizer can be computed as $\lambda\mathbf{R}(\mathbf{G} \otimes \mathbf{K})\mathbf{R}^{\mathrm{T}}\mathbf{a}$. Thus, the gradient of the objective function becomes

$$\frac{\partial J}{\partial \mathbf{a}} = \mathbf{R}(\mathbf{G} \otimes \mathbf{K})\mathbf{R}^{\mathrm{T}}(\mathbf{g} + \lambda\mathbf{a}) \tag{9}$$

The Hessian of $\mathcal{L}$ with respect to $\mathbf{a}$ can be defined as

$$\frac{\partial^2 \mathcal{L}}{\partial \mathbf{a}^2} = \frac{\partial \mathbf{p}}{\partial \mathbf{a}}\frac{\partial}{\partial \mathbf{p}}(\frac{\partial \mathbf{p}}{\partial \mathbf{a}}\frac{\partial \mathcal{L}}{\partial \mathbf{p}}) = \mathbf{R}(\mathbf{G} \otimes \mathbf{K})\mathbf{R}^{\mathrm{T}}\mathbf{H}\mathbf{R}(\mathbf{G} \otimes \mathbf{K})\mathbf{R}^{\mathrm{T}}$$

Hessian for the regularizer is defined as $\lambda\mathbf{R}(\mathbf{G} \otimes \mathbf{K})\mathbf{R}^{\mathrm{T}}$. Thus the Hessian for the objective function becomes

$$\frac{\partial^2 J}{\partial \mathbf{a}^2} = \mathbf{R}(\mathbf{G} \otimes \mathbf{K})\mathbf{R}^{\mathrm{T}}(\mathbf{H}\mathbf{R}(\mathbf{G} \otimes \mathbf{K})\mathbf{R}^{\mathrm{T}} + \lambda\mathbf{I})$$

For commonly used univariate losses, assuming $\mathbf{p}$ has been computed for the current solution, $\mathbf{g}$, and $\mathbf{H}$ can be computed in $O(n)$ time. The overall cost of the loss, gradient and Hessian-vector product computations will then be dominated by the efficiency of the Kronecker-kernel vector product algorithm, resulting in $O(qn + mn)$ time.

Conversely. in the primal case we can compute the predictions as $f(S) = \mathbf{R}(\mathbf{T} \otimes \mathbf{D})\mathbf{w}$, loss gradient as $(\mathbf{T}^{\mathrm{T}} \otimes \mathbf{D}^{\mathrm{T}})\mathbf{R}^{\mathrm{T}}\mathbf{g}$ and Hessian for the loss as $(\mathbf{T}^{\mathrm{T}} \otimes \mathbf{D}^{\mathrm{T}})\mathbf{R}^{\mathrm{T}}\mathbf{H}\mathbf{R}(\mathbf{T} \otimes \mathbf{D})$, (both w.r.t. $\mathbf{w}$). For the regularizer the corresponding values are $\frac{\lambda}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w}$, $\lambda\mathbf{w}$ and $\lambda\mathbf{I}$. The overall complexity of the loss, gradient and Hessian-vector product computations is $O(\min(qdr + dn, mdr + rn))$.

3.3 Optimization framework

As a simple approach that can be used for training regularized risk minimization methods with access to gradient and and (generalized) Hessian-vector product operations, we consider a Truncated Newton optimization scheme (Algorithms 2 and 3; these implement approaches similar to [28], [36]). This is a second order optimization method that on each iteration computes the Newton step direction $\partial^2 J(f)\mathbf{x} = \partial J(f)$ approximately up to a pre-defined number of steps for the linear system solver. The approach is well suited for Kronecker kernel method optimization, since while computing the Hessians explicitly is often not computationally feasible, computing Hessian-vector products can be done efficiently using the sampled Kronecker product algorithm.

Alternative optimization schemes can certainly be used, such as the Limited-memory BFGS algorithm [37], trust-region Newton optimization [32], and methods tailored for non-smooth optimization such as the bundle method [33]. Such work is orthogonal to our work, as the proposed algorithm can be used to speed up computations for any optimization method that is based on (sub)gradient, and possibly (generalized) Hessian-vector product computations. Optimization methods that process either the samples or the model coefficients individually or in small batches (e.g. stochastic gradient descent [38], coordinate descent [39], SVM decomposition methods [40]) may however not be a good match for the proposed algorithm, as the largest speed-up is gained when doing the computations for all of the training data in one single batch.

Each iteration of the Truncated Newton algorithm starts by computing the vector of training set predictions $\mathbf{p}$ for the current solution, after which the gradient $\mathbf{g}$ and Hessian $\mathbf{H}$ can be computed (see Table 1 for examples on how to compute these for several widely used loss functions). After this, the algorithm solves approximately the linear system

$$\frac{\partial^2 J}{\partial \mathbf{a}^2}\mathbf{x} = \frac{\partial J}{\partial \mathbf{a}},$$

to find the next direction of descent (for the primal case we solve analogously $\frac{\partial^2 J}{\partial \mathbf{w}^2}\mathbf{x} = \frac{\partial J}{\partial \mathbf{w}}$). Here, we may use methods developed for solving linear systems, such as the Quasi-Minimal Residual iteration method [41] used in our experiments. For the dual case, we may simplify this system of equations by removing the common term $\mathbf{R}(\mathbf{G} \otimes \mathbf{K})\mathbf{R}^{\mathrm{T}}$ from both sides, resulting in the system

$$(\mathbf{HR}(\mathbf{G} \otimes \mathbf{K})\mathbf{R}^{\mathrm{T}} + \lambda \mathbf{I})\mathbf{x} = \mathbf{g} + \lambda \mathbf{a}\ . \tag{10}$$

While the optimization approaches can require a large number of iterations in order to converge, in practice often good predictive accuracy can be obtained using early stopping both in solving the system of linear equations on line 5 of Algorithm 2 and line 5 of Algorithm 3 and the outer truncated Newton optimization loops, as there is no need to continue optimization once the error of the prediction function stops decreasing on a separate validation set (see e.g. [42], [43], [44]).

3.4 Complexity comparison

The operations that dominate the computational cost for Algorithms 2 and 3 are matrix-vector products of the form $\mathbf{R}(\mathbf{G} \otimes \mathbf{K})\mathbf{R}^{\mathrm{T}}\mathbf{v}$ and $(\mathbf{T}^{\mathrm{T}} \otimes \mathbf{D}^{\mathrm{T}})\mathbf{R}^{\mathrm{T}}\mathbf{v}$. A natural question to ask is, how much does the proposed fast sampled Kronecker product algorithm (Algorithm 1) speed these operations up, compared to explicitly forming the sampled Kronecker product matrices (here denoted the 'Baseline' method).

We consider three different settings:

- Independent: each data point and task appears only as part of a single input in the training set. That is $n = m = q$.
- Dependent: data points and tasks can appear as parts of multiple training inputs, and $\max(m, q) < n < mq$
- Complete: all possible combinations of the training data points and tasks appear as training inputs. That is $n = mq$.

'Dependent' is the default setting assumed in this paper, while 'Independent' and 'Complete' are extreme cases of the setting.

In Table 2 we provide the comparison for the dual case. In the 'Independent' -case the complexity of the proposed algorithm reduces to that of the baseline method, since there are no data or tasks shared between the pair-inputs, that the algorithm could make use of. However, beyond that the complexity of the baseline approach grows quadratically with respect to the number of pair-inputs, while the complexity of the proposed method grows linearly with respect to the numbers of data, tasks and pair-inputs. In Table 3 we provide the comparison for the primal case. Again, the complexity is the same for the 'Independent' case, but the proposed method is much more efficient for the other settings assuming $m << n$ and $q << n$.

Finally, we note that for both the primal and the dual settings, for the 'Complete' case where $\mathbf{R} = \mathbf{I}$, the multiplication could as efficiently be computed using the 'VecTrick' - method implied by Roth's column lemma (1).

TABLE 2
Dual case: complexity comparison of the proposed method and the baseline approach that constructs the Kronecker kernel matrix explicitly.

	Baseline	Proposed
Independent	$O(n^2)$	$O(n^2)$
Dependent	$O(n^2)$	$O(qn + mn)$
Complete	$O(m^2q^2)$	$O(m^2q + mq^2)$

TABLE 3
Primal case: complexity comparison of the proposed method and the baseline approach that constructs the Kronecker data matrix explicitly.

	Baseline	Proposed
Independent	$O(ndr)$	$O(ndr)$
Dependent	$O(ndr)$	$O(\min(qdr + dn, mdr + rn))$
Complete	$O(mqdr)$	$O(\min(mdr + mqr, drq + dmq))$

To conclude, we observe that the proposed algorithm considerably outperforms previously known methods, if $\max(m, q) << n < mq$.

4 Case studies: ridge regression and support vector machines

Next, as an example on how our framework may be applied we consider two commonly used losses, deriving fast ridge regression and support vector machine training algorithms for the Kronecker kernel.

4.1 Kronecker ridge regression

As our first example learning algorithm that uses the above defined concepts, let us consider the well-known ridge regression method (aka regularized least-squares, aka least-squares SVM) [26], [45]. For this specific case, the resulting algorithm is simpler than the general optimization framework of Algorithms 2 and 3, as the linear system appearing in the algorithms needs to be solved only once.

In this case, $\mathcal{L} = \frac{1}{2}(\mathbf{p} - \mathbf{y})^{\mathrm{T}}(\mathbf{p} - \mathbf{y})$, $\mathbf{H} = \mathbf{I}$ and $\mathbf{g} = \mathbf{p} - \mathbf{y}$ (see Table 1). For the dual case, based on (9) it can be observed that the full gradient for the ridge regression is

$$\mathbf{R}(\mathbf{G} \otimes \mathbf{K})\mathbf{R}^{\mathrm{T}}(\mathbf{p} - \mathbf{y} + \lambda\mathbf{a})$$

Writing $\mathbf{p}$ open (8) and setting the gradient to zero, we recover a linear system

$$\mathbf{R}(\mathbf{G} \otimes \mathbf{K})\mathbf{R}^{\mathrm{T}}(\mathbf{R}(\mathbf{G} \otimes \mathbf{K})\mathbf{R}^{\mathrm{T}} + \lambda\mathbf{I})\mathbf{a} = \mathbf{R}(\mathbf{G} \otimes \mathbf{K})\mathbf{R}^{\mathrm{T}}\mathbf{y}$$

A solution to this system can be recovered by canceling out $\mathbf{R}(\mathbf{G} \otimes \mathbf{K})\mathbf{R}^{\mathrm{T}}$ from both sides, resulting in the linear system

$$(\mathbf{R}(\mathbf{G} \otimes \mathbf{K})\mathbf{R}^{\mathrm{T}} + \lambda\mathbf{I})\mathbf{a} = \mathbf{y}$$

This system can be solved directly via standard iterative solvers for systems of linear equations. Combined with the sampled Kronecker product algorithm, this results in $O(mn + qn)$ cost for each iteration of the method.

For the primal case, the gradient can be expressed as

$$(\mathbf{T}^{\mathrm{T}} \otimes \mathbf{D}^{\mathrm{T}})\mathbf{R}^{\mathrm{T}}(\mathbf{p} - \mathbf{y}) + \lambda\mathbf{w}$$

Again, writing $\mathbf{p}$ open and setting the gradient to zero, we recover a linear system

$$(\mathbf{T}^{\mathrm{T}} \otimes \mathbf{D}^{\mathrm{T}})\mathbf{R}^{\mathrm{T}}\mathbf{R}(\mathbf{T} \otimes \mathbf{D}) + \lambda\mathbf{I})\mathbf{w} = (\mathbf{T}^{\mathrm{T}} \otimes \mathbf{D}^{\mathrm{T}})\mathbf{R}^{\mathrm{T}}\mathbf{y}$$

Solving this system with a linear system solver together with the sampled Kroneker product algorithm, each iteration will require $O(\min(mdr + nr, drq + dn))$ cost.

The naive approach of training ridge regression with the full sampled Kronecker kernel or data matrix using standard solvers of systems of linear equations would result in $O(n^2)$ and $O(ndr)$ cost per iteration for the dual and primal cases, respectively.

4.2 Kronecker support vector machine

Support vector machine is one of the most popular classification methods in machine learning. Two of the most widely used variants of the method are the so-called L1-SVM and L2-SVM (see Table 1). Following works such as [28], [36], [32] we consider the L2-SVM variant, since unlike L1-SVM it has an objective function that is differentiable and yields a non-zero generalized Hessian matrix, making it compatible with the presented Truncated Newton optimization framework.

The loss can be defined as $\mathcal{L} = \frac{1}{2}\sum_{i=1}^{n} \max(0, 1 - p_i \cdot y_i)^2$, where $y \in \{-1, 1\}$. Let $\mathcal{S} = \{i | y \cdot p(\mathbf{x}_i) < 1, i \in [n]\}$ denote the set of training data that have non-zero loss for given prediction function. Further, let $\mathbf{S}_{\mathcal{S}}$ denote the sampling matrix corresponding to this index set. Now we can re-write the loss in a least-squares form as $\mathcal{L} = \frac{1}{2}(\mathbf{S}_{\mathcal{S}}\mathbf{p} - \mathbf{S}_{\mathcal{S}}\mathbf{y})^{\mathrm{T}}(\mathbf{S}_{\mathcal{S}}\mathbf{p} - \mathbf{S}_{\mathcal{S}}\mathbf{y})$, its gradient as $\mathbf{g} = \mathbf{S}_{\mathcal{S}}^{\mathrm{T}}(\mathbf{S}_{\mathcal{S}}\mathbf{p} - \mathbf{S}_{\mathcal{S}}\mathbf{y})$, and the Hessian as $\mathbf{H} = \mathbf{S}_{\mathcal{S}}^{\mathrm{T}}\mathbf{S}_{\mathcal{S}}$, which is a diagonal matrix with entry 1 for all members of $\mathcal{S}$, and 0 otherwise.

By inserting $\mathbf{g}$ to (9) and writing $\mathbf{p}$ open (8) we recover the gradient of the L2-SVM objective function, with respect to $\mathbf{a}$, as

$$\mathbf{R}(\mathbf{G} \otimes \mathbf{K})\mathbf{R}^{\mathrm{T}}(\mathbf{S}_{\mathcal{S}}^{\mathrm{T}}(\mathbf{S}_{\mathcal{S}}\mathbf{R}(\mathbf{G} \otimes \mathbf{K})\mathbf{R}^{\mathrm{T}}\mathbf{a} - \mathbf{S}_{\mathcal{S}}\mathbf{y}) + \lambda\mathbf{a}) \tag{11}$$

And inserting $\mathbf{g}$ and $\mathbf{H}$ to (10), we see that $\frac{\partial J}{\partial \mathbf{a} \partial \mathbf{a}}\mathbf{x} = \frac{\partial J}{\partial \mathbf{a}}$ can be solved from:

$$(\mathbf{S}_{\mathcal{S}}^{\mathrm{T}}\mathbf{S}_{\mathcal{S}}\mathbf{R}(\mathbf{G} \otimes \mathbf{K})\mathbf{R}^{\mathrm{T}} + \lambda\mathbf{I})\mathbf{x} = (\mathbf{S}_{\mathcal{S}}^{\mathrm{T}}(\mathbf{S}_{\mathcal{S}}\mathbf{R}(\mathbf{G} \otimes \mathbf{K})\mathbf{R}^{\mathrm{T}}\mathbf{a} - \mathbf{S}_{\mathcal{S}}\mathbf{y}) + \lambda\mathbf{a}) \tag{12}$$

The gradient and Hessian-vector products can be computed again at $O(qn + mn)$ time. However, it should be noted that the support vector machine algorithm encourages sparsity, meaning that as the optimization progresses $\mathbf{a}$ may come to contain many zero coefficients, and the set $\mathcal{S}$ may shrink (at the optimal solution these two sets coincide, denoting the so-called support vectors). Thus given a new solution $\mathbf{a}$, we may compute $\mathbf{p} = \mathbf{R}(\mathbf{G} \otimes \mathbf{K})\mathbf{R}^{\mathrm{T}}\mathbf{a}$ and thus also the right-hand size of (12) in $O(\min(q\|\mathbf{a}\|_0 + mn, m\|\mathbf{a}\|_0 + qn)$ time, by removing the zero-coefficients from $\mathbf{a}$ and the corresponding columns from $\mathbf{R}^T$. Further, when solving (12) with Truncated Newton optimization, we can on each inner iteration of the method compute matrix-vector multiplication $\mathbf{S}_{\mathcal{S}}\mathbf{R}(\mathbf{G} \otimes \mathbf{K})\mathbf{R}^{\mathrm{T}}\mathbf{v}$ in $O(\min(q\|\mathbf{v}\|_0 + m|\mathcal{S}|, m\|\mathbf{v}\|_0 + q|\mathcal{S}|)$ time.

In the primal case the gradient can be written as

$$(\mathbf{T}^{\mathrm{T}} \otimes \mathbf{D}^{\mathrm{T}})\mathbf{R}^{\mathrm{T}}\mathbf{S}_{\mathcal{S}}^{\mathrm{T}}(\mathbf{S}_{\mathcal{S}}\mathbf{p} - \mathbf{S}_{\mathcal{S}}\mathbf{y}) + \lambda\mathbf{w}$$

And, a generalized Hessian matrix can be defined as

$$\mathbf{H} = (\mathbf{T}^{\mathrm{T}} \otimes \mathbf{D}^{\mathrm{T}})\mathbf{R}^{\mathrm{T}}\mathbf{S}_{\mathcal{S}}^{\mathrm{T}}\mathbf{S}_{\mathcal{S}}\mathbf{R}(\mathbf{T} \otimes \mathbf{D}) + \lambda\mathbf{I}$$

The most efficient existing SVM solvers can be expected at best to scale quadratically with respect to sample size when solving the dual problem, and linearly with respect to sample size and number of features when solving the primal case [46], [47]. Thus using the full Kronecker kernel or data matrices, such solvers would have $O(n^2)$ and $O(ndr)$ scaling for the dual and primal cases, respectively.

5 Experiments

Computational costs for iterative Kronecker kernel method training depend on two factors. The first one is the cost of a single iteration, where operations such as gradient or Hessian-vector product computations dominate the costs. These can be speeded up with the sampled Kronecker product algorithm introduced in this work. The second is the number of iterations needed to reach a good solution. While training Kronecker kernel methods requires solving optimization problems with a huge number of variables, in practice it is enough to obtain an approximate solution that generalizes well to new data. For prediction times, the dominating costs are the data matrix or kernel matrix multiplications with the primal or (possibly sparse) dual coefficient vectors, both operations that can be speeded up with the sampled Kronecker product algorithm.

In the experiments, we aim to demonstrate the following three things:

1) Utility of early stopping: often only a small number of iterations is needed to learn an accurate predictor.
2) Fast training: the combination of the proposed short-cuts and early stopping allows orders of magnitude faster training than with regular kernel method solvers.
3) Fast prediction: proposed short-cuts allow orders of magnitude faster prediction for new data, than with regular kernel predictors.

TABLE 4
Data sets

Data set	drugs	targets	pair-inputs	pos. interactions
Ki	1421	156	93356	3200
GPCR	223	95	21185	635
IC	210	204	42840	1476
E	445	664	295480	2926

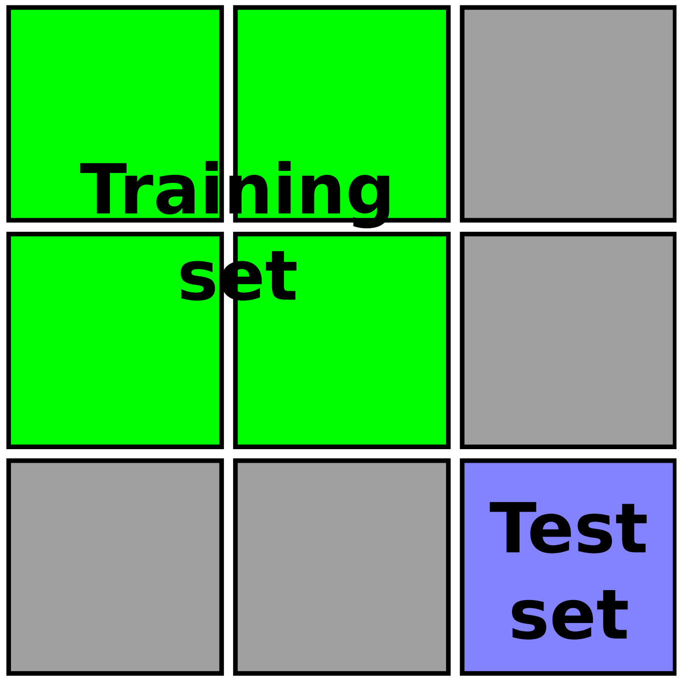

Fig. 2. Ninefold cross-validation. The output matrix is divided into nine non-overlapping blocks. On each round, the test set is formed from all the pair-inputs corresponding to one of the blocks. Then training set is formed from all the blocks, that share no common rows or columns with the test set, resulting in four blocks being assigned to the training set. Four blocks are left unused, as either their rows or columns overlap with the test set.

We implement the Kronecker SVM and ridge regression methods in Python, the methods are made freely available under open source license as part of the RLScore machine learning library[1]. For comparison, we consider the LibSVM software [48], which implements a highly efficient support vector machine training algorithm [49].

5.1 Data and general setup

As a typical zero-shot learning problem from the field of bioinformatics, we consider the problem of predicting drug-target interactions. From a data base of known drugs, targets and their binary interactions, the goal is to learn a model that can for new previously unseen drugs and targets predict whether they interact. Such a model would be useful for example for suggesting promising candidate pair-inputs for further laboratory experiments. We experiment on four drug-target interaction data sets, the GPCR, IC, and E and datasets [50], and the kinase inhibition constant (Ki) data [51]. Each data set contains a number of drugs and targets, as well as binary outputs for drug-target pairs denoting whether they are known to interact or not. The Ki-data is a naturally sparse data set where the class information is available only for a subset of the pair-inputs. For the other four data sets the full drug-target matrix is available, we simulate a setting with sparse data by randomly sampling 25% of the outputs in each of these. We use exactly the same preprocessing of the data as [11], where more details about the data sets and features can be found. The characteristics of the data sets are described in Table 4.

For pair-input data cross-validation is more complicated than for standard i.i.d. data, since same data points and tasks can appear as parts of different inputs. For example the standard leave-one-out cross-validation approach does not measure correctly generalization in the zero-shot learning case, since the holdout input may share the data point or task parts, or both, with inputs in the training folds. In cross-validation one needs to ensure that the

1. https://github.com/aatapa/RLScore

inputs selected for the holdout set contain neither common data points nor tasks with the training set, by leaving all such inputs out of the training set. This approach is illustrated in Figure 2, for a more detailed description of the cross-validation approach, see [52], [11].

The ridge regression implementation is trained using the Minimum Residual iteration algorithm [53] implemented in the scipy.sparse.linalg.minres package, while the inner optimization loop of the SVM training algorithm (i.e. the one solving the system of linear equations) uses the scipy.sparse.linalg.qmr package implementing the Quasi-Minimal Residual iteration method [41] (SciPy version 0.14.1), with $\delta = 1$. Experiments were performed for regularization parameter values on the grid $[2^{-20}, ..., 2^{20}]$. In order to present the results in a human interpretable form, we restrict our plots to values $[2^{-10}, 2^{-5}, 2^{0}, 2^{5}, 2^{10}]$, as these allow representing all the main trends observed in the experiments. Both ridge regression and SVM were run for 100 iterations, and the classification performance on test data was measured with area under ROC curve (AUC). In order to allow comparing the primal and dual optimization, the experiments were run using the linear kernel, with the exception of the LibSVM comparison considered later. The experiments were run on a desktop computer with Intel Core i7-3770 CPU (3.40GHz) running Ubuntu Linux 15.04.

LibSVM does not directly support the Kronecker product kernel, however this issue can be resolved as follows. Let us consider using both as task and datum kernels the Gaussian kernel, with width γ same for both. In this case $k(\mathbf{d}, \mathbf{d}')k(\mathbf{t}, \mathbf{t}') = e^{-\gamma\|\mathbf{t}-\mathbf{t}'\|^2} e^{-\gamma\|\mathbf{d}-\mathbf{d}'\|^2} = e^{-\gamma\|[\mathbf{t},\mathbf{d}]^{\mathrm{T}}-[\mathbf{t}',\mathbf{d}']^{\mathrm{T}}\|^2}$, that is, the Kronecker kernel is equal to using the Gaussian kernel with concatenated features of the datum and the task.

5.2 Early stopping experiments

Training Kronecker kernel methods requires solving optimization problems with a huge number of variables. In the dual and primal Truncated Newton algorithms (Algorithms 2 and 3), one needs to repeatedly solve a linear system, with sizes corresponding either to the number of training inputs, or to the product of data and task dimensions. However, as discussed in Section 3.3, the classical approach of early stopping can be used to make the optimization more efficient; optimization can be terminated once the accuracy of the model stops increasing on separate validation data. We test how well this approach works both for the SVM and ridge regression methods. We consider only the dual optimization case, since we observed that the behavior was very similar also for the primal case.

We run the optimization up to 100 iterations measuring two quantities: the regularized risk $J(f)$, and AUC on the test set (we also did selected experiments running the methods further up to 500 iterations, and found little improvements in the results). For ridge regression the results are presented in Figure 3. For SVM we run two variants, one where the inner optimization loop is terminated at 10 iterations (Figure 4), and one where it is terminated at 100 iterations (Figure 5).

A trend that holds over all the methods and data sets considered is that the optimal test set AUC is obtained within tens of iterations. Beyond this point reduction in regularized risk no longer translates into better predictions. Further, for SVMs increasing the number of inner iterations from 10 to 100 allows achieving much faster decrease in regularized risk. However, this comes at the cost of having to perform ten times more computation each iteration, and does not lead into faster increase in test set AUC. Thus it can be observed, that rather than having to solve exactly the large optimization problems corresponding to training Kronecker kernel methods, often only a handful of iterations need to be performed in order to obtain maximal predictive accuracy.

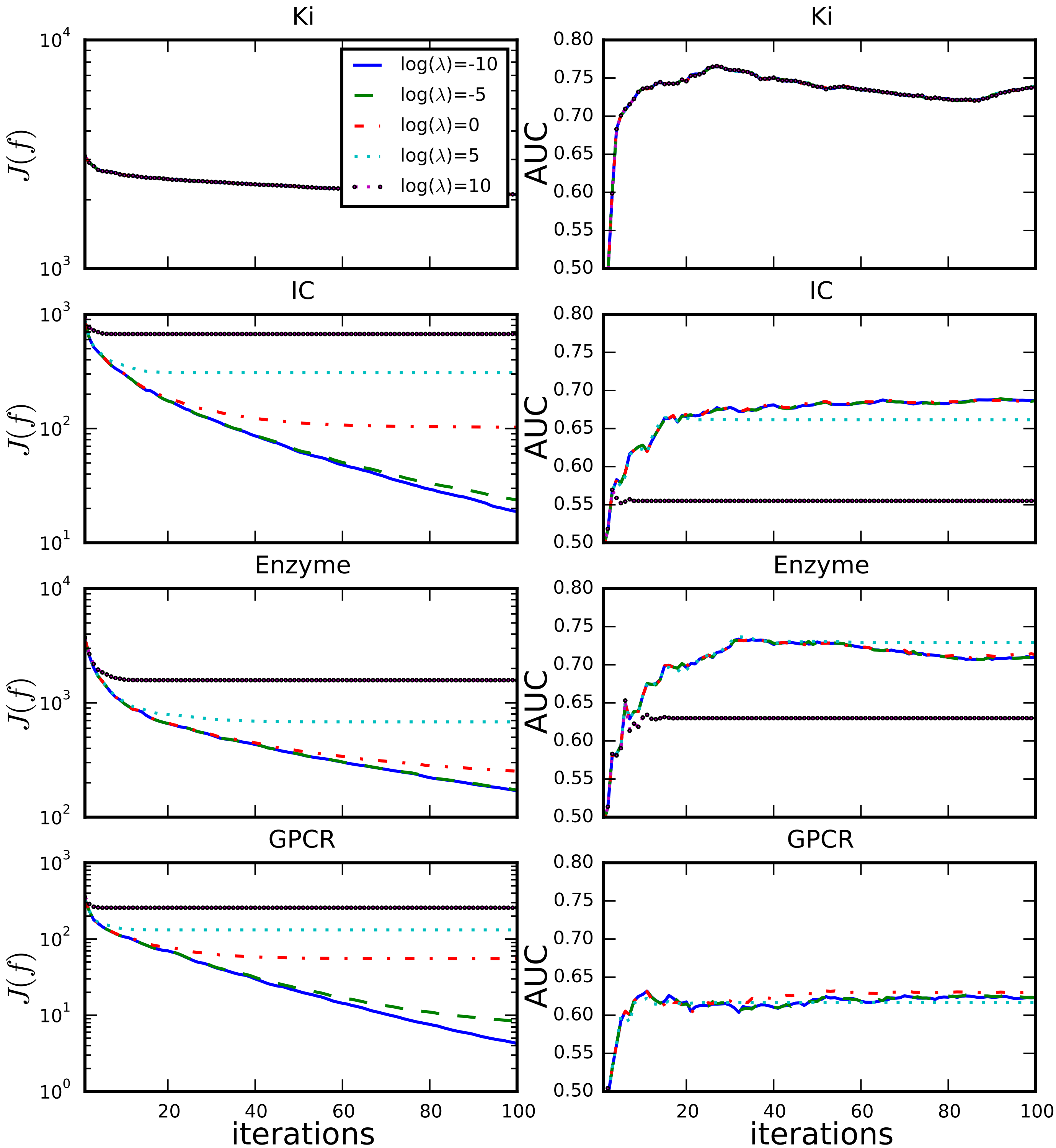

Fig. 3. Ridge regression regularized risk (left) and test set AUC (right) as a function of optimization iterations.

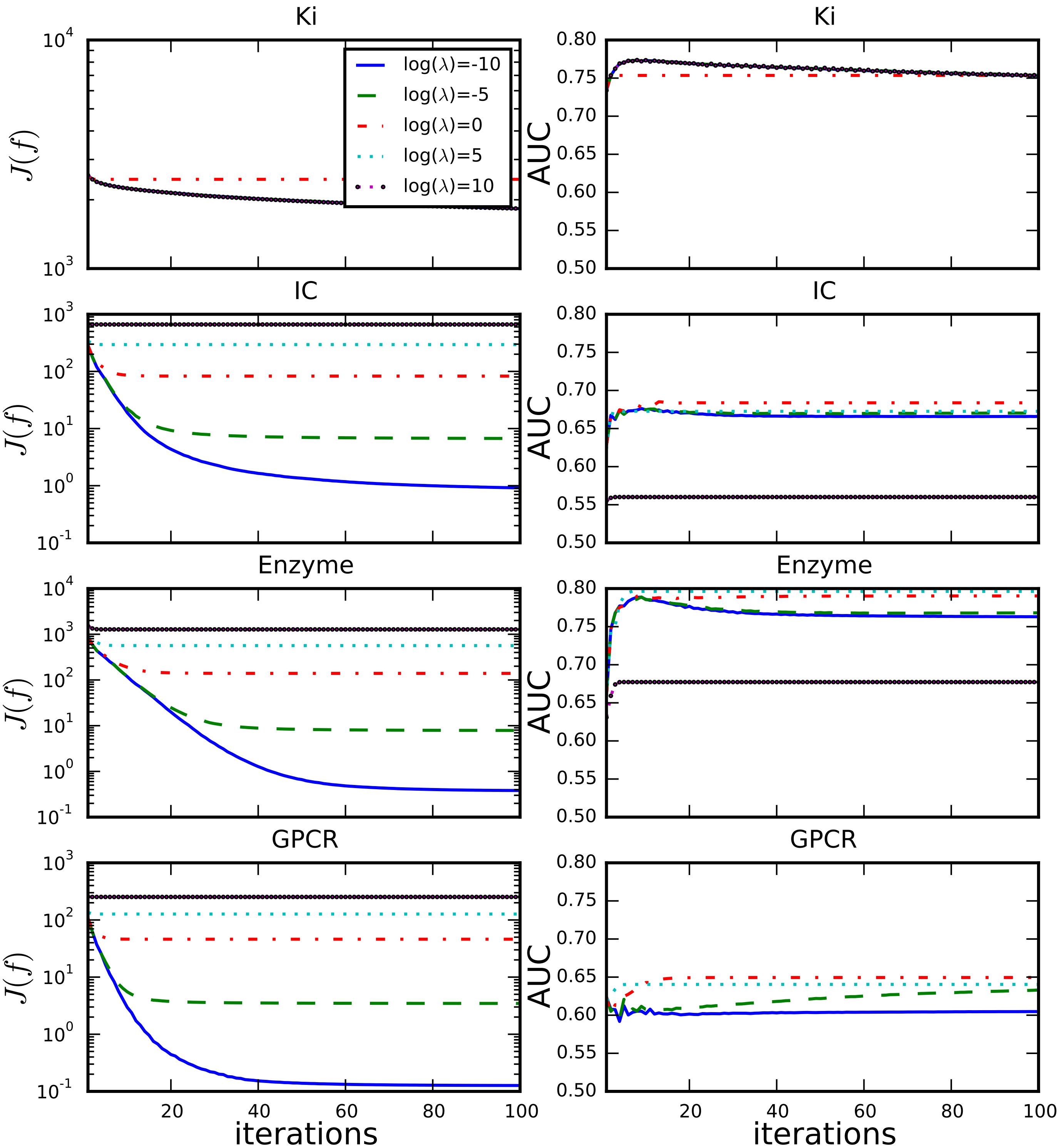

Fig. 4. SVM with 10 inner iterations. Regularized risk (left) and test set AUC (right) as a function of outer optimization iterations.

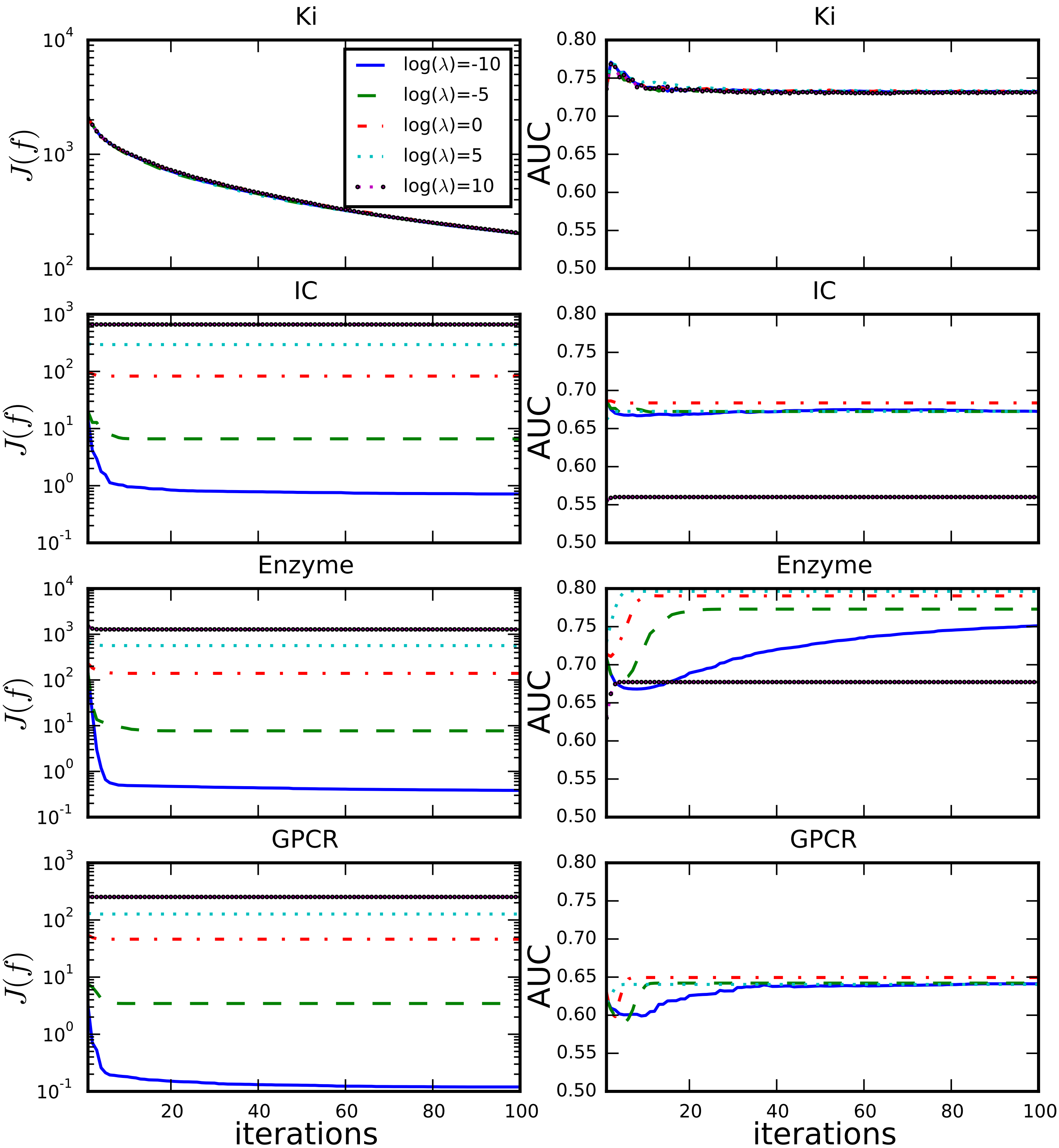

Fig. 5. SVM with 100 inner iterations. Regularized risk (left) and test set AUC (right) as a function of outer optimization iterations.

5.3 Training time

In order to demonstrate the improvements in training speed that can be realized using the sparse Kronecker product algorithm, we compare our Kronecker SVM algorithm to the LibSVM solver on the Ki-data set. Based on preliminary tests, we set $\gamma = 10^{-5}$, as this value produces informative (not too close to identity matrix, or to matrix full of ones) kernel matrices for both the datums and the tasks. For the Kronecker SVM, we perform 10 inner and 10 outer iterations with $\lambda = 2^{-5}$. For LibSVM, we present the values for the grid $[2^{-7}, 2^{-5}, 2^{-3}, 2^{-1}]$, as results for

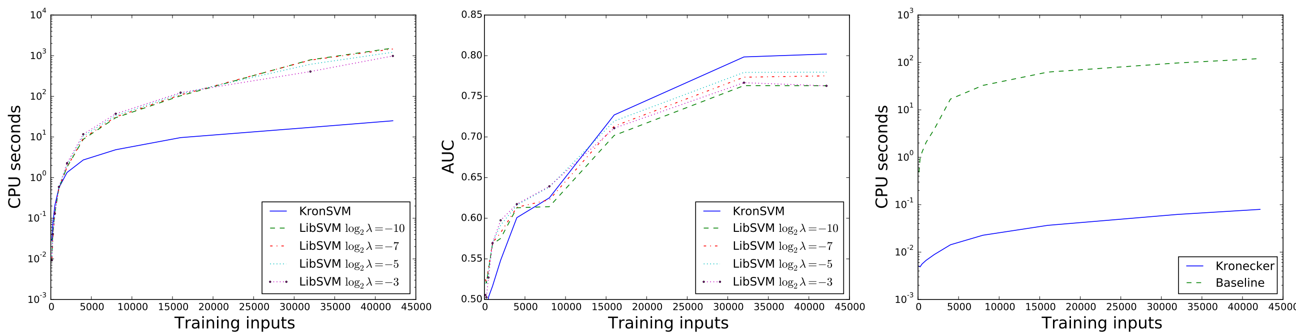

Fig. 6. Runtime comparison between KronSVM and LIBSVM (left) and the corresponding cross-validated AUCs (middle). Prediction times for regular LibSVM decision function, and one that uses the sparse Kronecker product shortcuts (right).

it vary more based on the choice of regularization parameter. We perform 9-fold cross-validation on the K_i data, subsampling the number of inputs in the training data. In Figure 6 (left) we present the running times for training the methods for different numbers of training inputs, as well as the corresponding cross-validated AUCs. The KronSVM results, while similar as before, are not directly comparable to those in the earlier experiments due to different kernel function being used.

As can be expected based on the computational complexity considerations, the KronSVM algorithm has superior scalability compared to the regular SVM implemented in the LibSVM package. In the experiment on 42000 pair-inputs the difference is already 25 seconds versus 15 minutes. The LibSVM runtimes could certainly be improved for example by using earlier termination of the optimization. Still, this would not solve the basic issue that without using computational shortcuts such as the sampled Kronecker product algorithm (Algorithm 1) proposed in our work, one will need to construct a significant part of the kernel matrix for the inputs. For LibSVM, the scaling is roughly quadratic in the number of inputs, while for KronSVM it is linear. Considering the cross-validated AUCs (Figure 6 (middle)), it can be observed that the AUC of the KronSVM with early stopping is very much competitive with that of LibSVM. To conclude, the proposed Kronecker product algorithm allows substantially faster training for pairwise data than if one uses current state-of-the-art solvers, that are however not able to make use of the pairwise structure.

5.4 Prediction time

Finally, we show how the sparse Kronecker product algorithm can be used to accelerate predictions made for new data. As in previous experiment, we train LibSVM on the Ki-data for varying sample sizes, with the training and test set split done as in the 9-fold cross-validation experiment. We use $\lambda = -5$, and the same kernel and kernel parameters as before. We use the predictor learned by the SVM to make predictions for the 10000 drug-target pairs in the test set.

We compare the running times for two approaches for doing the predictions. 'Baseline' refers to the standard decision function implemented in LibSVM. 'Kronecker', refers to implementation that computes the predictions using the sparse Kronecker product algorithm. The 'Kronecker' method is implemented by combining the LibSVM code with additional code that after training LibSVM, reads in the dual coefficients of the resulting predictor and generates a new predictor that uses the shortcuts proposed in this paper. Both predictors are equivalent in the sense that they produce (within numerical accuracy) exactly the same predictions.

The results are plotted in Figure 6 (right). For both methods the prediction time increases linearly with the training set size (see equations (6) and (7)). However, the Kronecker method is more than 1000 times faster than Baseline, demonstrating that significant speedups can be realized by using the sparse Kronecker product algorithm for computing predictions.

6 Conclusion

In this work, we have proposed a sampled Kronecker product algorithm. A simple optimization framework is described in order to show how the proposed algorithm can be used to develop efficient training algorithms for pairwise kernel methods. Both computational complexity analysis and experiments show that the resulting algorithms can provide order of magnitude improvements in computational efficiency both for training and making predictions, compared to existing kernel method solvers. Our work opens future avenues of research both for developing even better optimized training algorithms for individual kernel methods, as well as for improving the efficiency of any other such methods that can be expressed in terms of sampled Kronecker products. The implementations for the sampled Kronecker product algorithm, as well as for Kronecker ridge regression and Kronecker SVM are made freely available under open source license.

REFERENCES

[1] E. V. Bonilla, F. V. Agakov, and C. K. I. Williams, "Kernel multi-task learning using task-specific features," in *11th International Conference on Artificial Intelligence and Statistics*, ser. JMLR Proceedings, M. Meila and X. Shen, Eds., vol. 2. JMLR.org, 2007, pp. 43–50.

[2] H. Larochelle, D. Erhan, and Y. Bengio, "Zero-data learning of new tasks," in *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 2*, A. Cohn, Ed. AAAI Press, 2008, pp. 646–651.

[3] K. Hayashi, T. Takenouchi, R. Tomioka, and H. Kashima, "Self-measuring similarity for multi-task Gaussian process," in *ICML Unsupervised and Transfer Learning Workshop*, ser. JMLR Proceedings, I. Guyon, G. Dror, V. Lemaire, G. W. Taylor, and D. L. Silver, Eds., vol. 27. JMLR.org, 2012, pp. 145–154.

[4] B. Romera-Paredes and P. H. S. Torr, "An embarrassingly simple approach to zero-shot learning," in *Proceedings of the 32nd International Conference on Machine Learning*, ser. JMLR Proceedings, F. R. Bach and D. M. Blei, Eds., vol. 37. JMLR.org, 2015, pp. 2152–2161.

[5] H. Ding, I. Takigawa, H. Mamitsuka, and S. Zhu, "Similarity-based machine learning methods for predicting drug-target interactions: a brief review," *Briefings in Bioinformatics*, vol. 15, no. 5, pp. 734–747, 2013.

[6] J. Basilico and T. Hofmann, "Unifying collaborative and content-based filtering," in *Proceedings of the twenty-first international conference on Machine learning*, C. E. Brodley, Ed. ACM, 2004, pp. 65–72.

[7] T.-Y. Liu, *Learning to Rank for Information Retrieval*. Springer, 2011.

[8] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, Aug 2009.

[9] T. Evgeniou, C. A. Micchelli, and M. Pontil, "Learning multiple tasks with kernel methods," *Journal of Machine Learning Research*, vol. 6, pp. 615–637, 2005.

[10] T. Pahikkala, M. Stock, A. Airola, T. Aittokallio, B. De Baets, and W. Waegeman, "A two-step learning approach for solving full and almost full cold start problems in dyadic prediction," in *Machine Learning and Knowledge Discovery in Databases*, ser. Lecture Notes in Computer Science, T. Calders, F. Esposito, E. Hüllermeier, and R. Meo, Eds., vol. 8725. Springer, 2014, pp. 517–532.

[11] T. Pahikkala, A. Airola, S. Pietil, S. Shakyawar, A. Szwajda, J. Tang, and T. Aittokallio, "Toward more realistic drug-target interaction predictions," *Briefings in Bioinformatics*, vol. 16, no. 2, pp. 325–337, 2015.

[12] K. Muller, S. Mika, G. Ratsch, K. Tsuda, and B. Scholkopf, "An introduction to kernel-based learning algorithms," *IEEE Transactions on Neural Networks*, vol. 12, no. 2, pp. 181–201, 2001.

[13] A. Ben-Hur and W. Noble, "Kernel methods for predicting protein-protein interactions," *Bioinformatics*, vol. 21 Suppl 1, pp. 38–46, 2005.

[14] H. Kashima, T. Kato, Y. Yamanishi, M. Sugiyama, and K. Tsuda, "Link propagation: A fast semi-supervised learning algorithm for link prediction," in *Proceedings of the SIAM International Conference on Data Mining*. SIAM, 2009, pp. 1099–1110.

[15] S.-T. Park and W. Chu, "Pairwise preference regression for cold-start recommendation," in *Proceedings of the Third ACM Conference on Recommender Systems*. New York, NY, USA: ACM, 2009, pp. 21–28.

[16] M. Hue and J. Vert, "On learning with kernels for unordered pairs," in *Proceedings of the 27th International Conference on Machine Learning*, J. Fürnkranz and T. Joachims, Eds. Omnipress, 2010, pp. 463–470.

[17] R. Raymond and H. Kashima, "Fast and scalable algorithms for semi-supervised link prediction on static and dynamic graphs," in *Machine learning and knowledge discovery in databases*, ser. Lecture Notes in Computer Science, J. L. Balcázar, F. Bonchi, A. Gionis, and M. Sebag, Eds. Springer, 2010, vol. 6323, pp. 131–147.

[18] T. Pahikkala, W. Waegeman, A. Airola, T. Salakoski, and B. De Baets, "Conditional ranking on relational data," in *Machine Learning and Knowledge Discovery in Databases*, ser. Lecture Notes in Computer Science, J. L. Balcázar, F. Bonchi, A. Gionis, and M. Sebag, Eds., vol. 6322. Springer, 2010, pp. 499–514.

[19] W. Waegeman, T. Pahikkala, A. Airola, T. Salakoski, M. Stock, and B. De Baets, "A kernel-based framework for learning graded relations from data," *IEEE Transactions on Fuzzy Systems*, vol. 20, no. 6, pp. 1090–1101, 2012.

[20] T. Pahikkala, A. Airola, M. Stock, B. D. Baets, and W. Waegeman, "Efficient regularized least-squares algorithms for conditional ranking on relational data," *Machine Learning*, vol. 93, no. 2-3, pp. 321–356, 2013.

[21] C. D. Martin and C. F. Van Loan, "Shifted Kronecker product systems," *SIAM Journal on Matrix Analysis and Applications*, vol. 29, no. 1, pp. 184–198, 2006.

[22] W. E. Roth, "On direct product matrices," *Bulletin of the American Mathematical Society*, vol. 40, pp. 461–468, 1934.

[23] T. Pahikkala, "Fast gradient computation for learning with tensor product kernels and sparse training labels," in *Structural, Syntactic, and Statistical Pattern Recognition*, ser. Lecture Notes in Computer Science, P. Fränti, G. Brown, M. Loog, F. Escolano, and M. Pelillo, Eds., vol. 8621. Springer, 2014, pp. 123–132.

[24] P. Drineas, M. W. Mahoney, and S. Muthukrishnan, "Sampling algorithms for l2 regression and applications," in *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2006, pp. 1127–1136.

[25] B. Schölkopf, R. Herbrich, and A. J. Smola, "A generalized representer theorem," in *Proceedings of the 14th Annual Conference on Computational Learning Theory and and 5th European Conference on Computational Learning Theory*, D. Helmbold and B. Williamson, Eds. London, UK, UK: Springer-Verlag, 2001, pp. 416–426.

[26] A. E. Hoerl and R. W. Kennard, "Ridge regression: Biased estimation for nonorthogonal problems," *Technometrics*, vol. 12, pp. 55–67, 1970.

[27] V. N. Vapnik, *The nature of statistical learning theory*. New York, NY, USA: Springer-Verlag New York, Inc., 1995.

[28] S. S. Keerthi and D. DeCoste, "A modified finite Newton method for fast solution of large scale linear SVMs," *Journal of Machine Learning Research*, vol. 6, pp. 341–361, Dec. 2005.

[29] S. H. Walker and D. B. Duncan, "Estimation of the probability of an event as a function of several independent variables," *Biometrika*, vol. 54, no. 1-2, pp. 167–179, 1967.

[30] C. Cortes, M. Mohri, and A. Rastogi, "Magnitude-preserving ranking algorithms," in *Proceedings of the 24th International Conference on Machine Learning*. New York, NY, USA: ACM, 2007, pp. 169–176.

[31] T. Pahikkala, E. Tsivtsivadze, A. Airola, J. Järvinen, and J. Boberg, "An efficient algorithm for learning to rank from preference graphs," *Machine Learning*, vol. 75, no. 1, pp. 129–165, 2009.

[32] C.-J. Lin, R. C. Weng, and S. S. Keerthi, "Trust region Newton method for logistic regression," *Journal of Machine Learning Research*, vol. 9, pp. 627–650, Jun. 2008.

[33] C. H. Teo, S. Vishwanthan, A. J. Smola, and Q. V. Le, "Bundle methods for regularized risk minimization," *Journal of Machine Learning Research*, vol. 11, pp. 311–365, Mar. 2010.

[34] A. Airola, T. Pahikkala, and T. Salakoski, "Training linear ranking SVMs in linearithmic time using red-black trees," *Pattern Recognition Letters*, vol. 32, no. 9, pp. 1328–1336, Jul. 2011.

[35] C.-P. Lee and C.-J. Lin, "Large-scale linear RankSVM," *Neural Computation*, vol. 26, no. 4, pp. 781–817, Apr. 2014.

[36] O. Chapelle, "Training a support vector machine in the primal," *Neural Computation*, vol. 19, no. 5, pp. 1155–1178, 2007.

[37] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu, "A limited memory algorithm for bound constrained optimization," *SIAM Journal on Scientific and Statistical Computing*, vol. 16, no. 5, pp. 1190–1208, Sep. 1995.

[38] S. Shalev-Shwartz, Y. Singer, and N. Srebro, "Pegasos: Primal Estimated sub-GrAdient SOlver for SVM," in *Proceedings of the 24th International Conference on Machine Learning*. New York, NY, USA: ACM, 2007, pp. 807–814.

[39] K.-W. Chang, C.-J. Hsieh, and C.-J. Lin, "Coordinate descent method for large-scale l2-loss linear support vector machines," *Journal of Machine Learning Research*, vol. 9, pp. 1369–1398, Jun. 2008.

[40] C.-J. Lin, "A study on SMO-type decomposition methods for support vector machines," *IEEE Transactions on Neural Networks*, vol. 17, no. 4, pp. 893–908, July 2006.

[41] R. Freund and N. Nachtigal, "QMR: a quasi-minimal residual method for non-Hermitian linear systems," *Numerische Mathematik*, vol. 60, no. 1, pp. 315–339, 1991.

[42] Y. Yao, L. Rosasco, and A. Caponnetto, "On early stopping in gradient descent learning," *Constructive Approximation*, vol. 26, no. 2, pp. 289–315, 2007.

[43] L. L. Gerfo, L. Rosasco, F. Odone, E. D. Vito, and A. Verri, "Spectral algorithms for supervised learning," *Neural Computation*, vol. 20, no. 7, pp. 1873–1897, 2008.

[44] A. Airola, T. Pahikkala, and T. Salakoski, "Large scale training methods for linear RankRLS," in *Proceedings of the ECML/PKDD-Workshop on Preference Learning*, E. Hüllermeier and J. Fürnkranz, Eds., 2010.

[45] T. Poggio and S. Smale, "The mathematics of learning: Dealing with data," *Notices of the American Mathematical Society (AMS)*, vol. 50, no. 5, pp. 537–544, 2003.

[46] L. Bottou and C.-J. Lin, "Support vector machine solvers," in *Large-Scale Kernel Machines*, ser. Neural Information Processing, L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, Eds. Cambridge, MA, USA: MIT Press, 2007, pp. 1–28.

[47] T. Joachims, "Training linear SVMs in linear time," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, T. Eliassi-Rad, L. H. Ungar, M. Craven, and D. Gunopulos, Eds. New York, NY, USA: ACM Press, 2006, pp. 217–226.

[48] C.-C. Chang and C.-J. Lin, "Libsvm: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, no. 3, pp. 27:1–27:27, 2011.

[49] R.-E. Fan, P.-H. Chen, and C.-J. Lin, "Working set selection using second order information for training support vector machines," *Journal of Machine Learning Research*, vol. 6, 2005.

[50] Y. Yoshihiro, A. Michihiro, G. Alex, H. Wataru, and K. Minoru, "Prediction of drug-target interaction networks from the integration of chemical and genomic spaces," *Bioinformatics*, vol. 24, no. 13, p. i232, 2008.

[51] J. T. Metz, E. F. Johnson, N. B. Soni, P. J. Merta, L. Kifle, and P. J. Hajduk, "Navigating the kinome," *Nature Chemical Biology*, vol. 7, no. 4, pp. 200–202, 2011.

[52] Y. Park and E. M. Marcotte, "Flaws in evaluation schemes for pair-input computational predictions," *Nature Methods*, vol. 9, no. 12, pp. 1134–1136, 2012.

[53] C. C. Paige and M. A. Saunders, "Solution of sparse indefinite systems of linear equations," *SIAM Journal on Numerical Analysis*, vol. 12, no. 4, pp. 617–629, 1975.