

# Fast Kronecker product kernel methods via generalized vec trick

Antti Airola, Tapio Pahikkala

---

## Abstract

Kronecker product kernel provides the standard approach in the kernel methods literature for learning from graph data, where edges are labeled and both start and end vertices have their own feature representations. The methods allow generalization to such new edges, whose start and end vertices do not appear in the training data, 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 training data is a complete bipartite graph. In this work we generalize these results to non-complete training graphs. This allows us to derive a general framework for training Kronecker product 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.

## Index Terms

kernel methods, Kronecker product kernel, bipartite graph learning, ridge regression, support vector machine, transfer learning, zero-shot learning

## 1 INTRODUCTION

This work concerns the problem of learning supervised machine learning models from labeled bipartite graphs. Given a training set  $(\mathbf{d}_i, \mathbf{t}_j, y_h)_{h=1}^n$  of edges  $(\mathbf{d}_i, \mathbf{t}_j)$ , where  $\mathbf{d}_i$  is the start vertex, and  $\mathbf{t}_j$  the end vertex and  $y_h$  the label, the goal is to learn to predict labels for new edges with unknown labels. We assume that both the start and end vertices have their own feature representations. Further, we assume that the same vertices tend to appear as start or end vertices in multiple edges (for example,  $(\mathbf{d}_t, \mathbf{t}_v)$ ,  $(\mathbf{d}_u, \mathbf{t}_w)$ ,  $(\mathbf{d}_t, \mathbf{t}_w)$  and  $(\mathbf{d}_u, \mathbf{t}_v)$  might all belong to the same training set). This latter property is what differentiates this learning setting from the standard supervised learning setting, as the data violates the very basic i.i.d. assumption. This overlapping nature of the data has major implications both on the correct way to estimate the generalization performance of learned predictors, and on how to design efficient algorithms for learning from such data.

As discussed by [1], [2], [3], we may divide this type of learning problem into a number of distinct settings, depending whether the training and test graphs are vertex disjoint or not. In this work, we assume that the training and test graphs are vertex disjoint, that is, neither the start vertex  $\mathbf{d}$  nor the end vertex  $\mathbf{t}$  of a test edge is part of any edge in the training set. This is in contrast to, for example, the less challenging setting usually considered in recommender systems literature, in which the training and test graphs are only edge disjoint. There the task is to impute missing labels to customers (start vertex)  $\times$  products (end vertex) matrix, where each row and column already contains at least some known labels (see Section 1.1 and [4] for more discussion). Following works such as [5], [6], we use the term zero-shot learning to describe our setting, as it requires generalization to new graphs that are vertex disjoint with the training graph. This setting is illustrated in Figure 1, where zero-shot learning corresponds to generalizing from a partially observed start vertices  $\times$  end vertices label matrix to predicting unknown labels at the bottom right for edges connecting new start and end vertices that have no edges connected to them in the training set.

We consider the setting, where both the feature representations of the start and end vertices are defined implicitly via kernel functions [7]. Further, following works such as [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [6], we assume that the feature representation of an edge is defined as the product of the start and end vertex kernels, resulting in the so-called Kronecker product kernel. Kronecker product kernel methods can be trained by plugging this edge kernel to any existing kernel machine solver. However, for problems where the number of edges is large this is not feasible in practice, as typically the runtime and in many cases also the memory use of kernel solvers grows at least quadratically with respect to number of edges.

In this work, we propose the first general Kronecker product kernel based learning algorithm for large-scale problems. This is realized by generalizing the computational short-cut implied by Roth’s column lemma [18] (often known as the vec-trick). As an example, we demonstrate in the experiments how the approach outperforms existing state-of-the-art SVM solver by several orders of magnitude when using the Gaussian kernel, the most well-known special case of the Kronecker

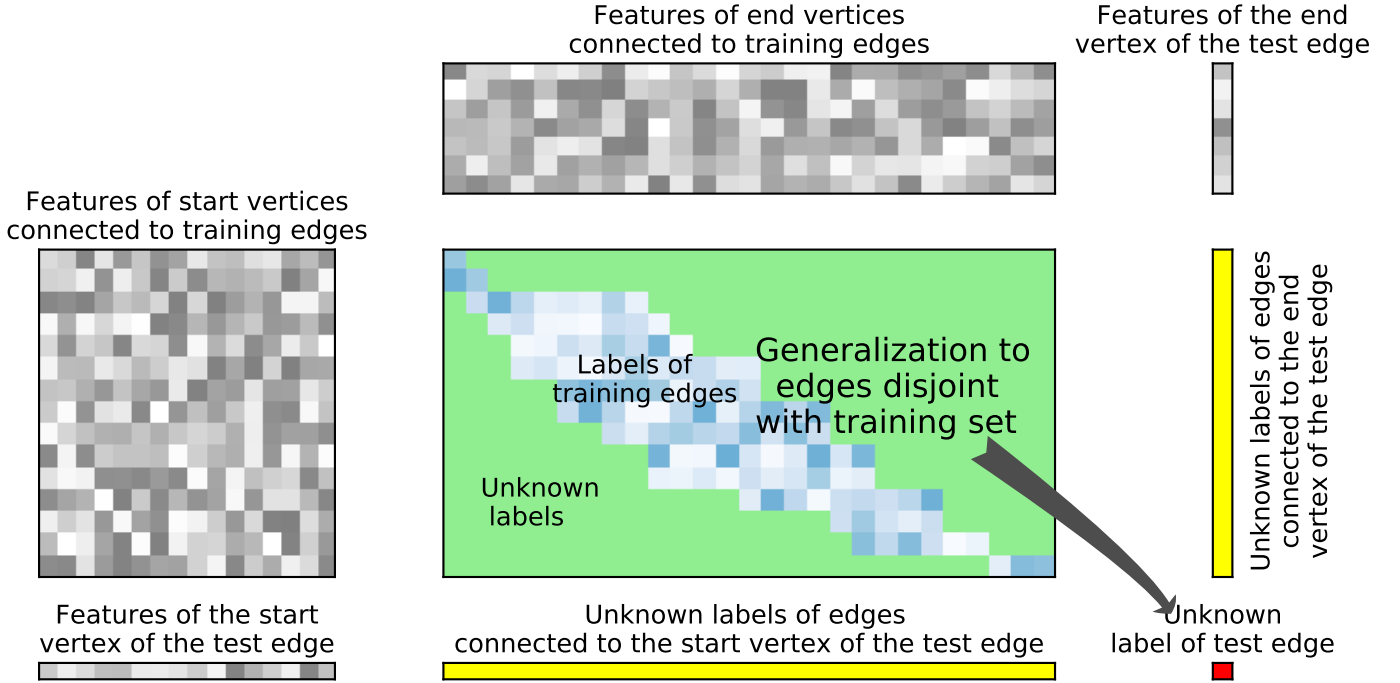


Fig. 1. Zero-shot learning. A predictor trained on edges and associated correct labels generalizes to such a test edge, for which both the start and end vertices are not part of the training set.

product kernel. Further, we show that the proposed methods perform well in comparison to other types of graph prediction methods. This work extends our previous work [19], that specifically considered the case of efficient ridge regression with Kronecker product kernel.

### 1.1 Related work

The bipartite graph learning problem considered in this work appears under various names in the machine learning literature, including link prediction [10], relational learning [15], pair-input prediction [1], and dyadic prediction [4]. The setting provides a unifying framework for prominent machine learning application areas in biology and chemistry dealing with pairwise interaction predictions, such as drug-target [3], and protein-RNA interaction prediction. The setting is prominent also in collaborative filtering, where the central task is to predict ratings or rankings for (customer, product) -pairs (see e.g. [8], [20], [4]). Also, in information retrieval the reranking of search engine results has been popularly cast as a bipartite graph ranking problem over (query, document) -pairs [21]. Many social network prediction tasks such as matching people to advertisements fall also naturally within the considered framework. Applications where the entities whose relations are predicted are of same type (i.e. protein-protein interactions, friendship in social network) can also be modeled as bipartite graph learning problems. In this case two vertices are used to represent each entity, one included in the set of start vertices and one in the set of end vertices.

In this work we consider the supervised graph learning problem, where the training edges are labeled. A limited range of graph learning problems may also be solved using unsupervised clustering methods [22], [23], for example by predicting that a relation holds between nodes assigned to the same cluster.

Kernel methods [7] are among the most widely used approaches in machine learning, both due to their strong foundations in statistical learning theory, as well as their practical applicability for a wide range of learning problems. The methods represent the data using positive semidefinite kernel functions, that allow incorporating prior knowledge about the problem domain, provide implicit non-linear mappings, and may be learned directly from data (see e.g. [24]). Recently, multi-task learning has been a topic of substantial research in the area of kernel methods (see e.g. [25], [26], [27]).

The use of Kronecker product kernel for graph learning problems was originally proposed at the same time in several independent works for collaborative filtering [8] protein-protein interaction prediction [9] and entity resolution [28]. Since then the approach has become especially popular in predicting biological interactions (see e.g. [29], [30], [3]), as well as a standard building block in more theoretical work concerning the development of multi-task learning methods (see e.g. [25], [31], [27]). Concerning specifically the zero-shot learning setting, Kronecker product kernel methods have been shown to outperform a variety of baseline methods in areas such as recommender systems [11], drug-target prediction [3] and image categorization [6].

Theoretical properties of the Kronecker product kernel were recently analyzed by [15]. They showed that if the start and end vertex kernels have the so-called universality property (e.g. Gaussian kernel) [32], then the resulting Kronecker

edge kernel is also universal, resulting in universal consistency when training kernel-based learning algorithms, such as support vector machines and ridge regression using the kernel.

Recently, it has been shown that the property that the same vertices tend to appear many times in the training data can be exploited in order to develop much faster Kronecker product kernel method training algorithms. Several efficient machine learning algorithms have been proposed for the special case, where the training sets consists of a complete bipartite graph, meaning that each possible start-end vertex pair appears exactly once, and a ridge regression loss is minimized. Specifically [13], [14], [16], [17], [6] derive closed form solutions based on Kronecker algebraic optimization (see also [33] for the basic mathematical results underlying these studies). Further, iterative methods based on Kronecker product kernel matrix - vector multiplications, have been proposed (see e.g. [10], [14], [16]).

However, the assumption about the training set graph being complete can be considered a major limitation on the applicability of these methods. As far as we are aware, similar efficient training algorithms as ours for regularized kernel methods have not been previously proposed for the case, where the training graph is sparse in the sense that only a subset of the possible edges appear in the training set.

Much of the work done in multi-task learning can be considered to fall under the bipartite graph prediction, and even zero-shot learning framework [25], [34], [14], [31], [17], [6], [35]. For example, in multi-label data one might have as training data images and labels describing the properties of the images (e.g. food, person, outside). In a traditional multi-label learning setting only the images have features describing them, and we assume that the set of possible labels is fixed in advance, and that examples of each label must appear in the training set. However, if the labels have also features describing them, we may relax these assumptions by casting the task as a graph prediction problem over (data point, label) -edges (e.g. predict 1 if label  $t_i$  should be assigned to data point  $d_j$  and -1 if not, or rank a set of candidate labels from best matching to worst). This allows predictions even for new data points and labels both not part of the training set.

There exist two related yet distinct graph learning problems, that have been studied in-depth in the machine learning literature. Let  $D$  and  $T$  denote, respectively, the sets of start and end vertices connected to the training edges, and  $(d, t)$  a new edge, for which the correct label needs to be predicted. First, if  $d \in D$  and  $t \in T$ , the problem becomes that of matrix completion (predicting the unknown labels within the center rectangle in Figure 1). Recently this setting has been considered especially in the recommender systems literature, where matrix factorization methods have become the de-facto standard approach for solving these problems (see e.g. [20]). The latent feature representations learned by the factorization methods, however, only generalize to edges that have both vertices connected to at least one edge in the training set. The second setting corresponds to the situation where  $d \in D$  and  $t \notin T$ , or alternatively  $d \notin D$  and  $t \in T$  (predicting the block on the right or below in Figure 1). Here predictions are needed only for new rows or columns of the label 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 [36].

While simultaneous use of both start and end vertex features can and has been integrated to learning also in these two related settings, they are not essential for generalization, and can even be harmful (so called negative transfer, see e.g. [17], [3]). The learning algorithms considered in this work are applicable in these related settings, provided that both start and end vertex features are available. However, simpler methods that do not need both of these information sources 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.

## 2 GENERALIZED VEC-TRICK

In this section, we introduce the novel generalized Vec-trick algorithm (Algorithm 1) for computing such Kronecker products, where a submatrix of a Kronecker product matrix ( $M \otimes N$ ) is multiplied to a vector. The algorithm forms the basic building block for developing computationally efficient training methods for Kronecker product kernel methods.

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

There exist 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** (Roth’s column lemma; “Vec trick” [18]). *Let  $M \in \mathbb{R}^{a \times b}$ ,  $Q \in \mathbb{R}^{b \times c}$ , and  $N \in \mathbb{R}^{c \times d}$  be matrices. Then,*

$$(N^T \otimes M)\text{vec}(Q) = \text{vec}(MQN). \quad (1)$$

It is obvious that the right hand side 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 correspond to a submatrix of a Kronecker product matrix. First, we introduce some further notation. To index certain of rows or columns of a matrix, the following construction may be used:

---

**Algorithm 1** Generalized Vec trick:  $\mathbf{u} \leftarrow \mathbf{R}(\mathbf{M} \otimes \mathbf{N})\mathbf{C}^T \mathbf{v}$ 


---

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

```

1: if  $ae + df < ce + bf$  then
2:    $\mathbf{T} \leftarrow \mathbf{0} \in \mathbb{R}^{d \times a}$ 
3:   for  $h = 1, \dots, e$  do
4:      $i, j \leftarrow r_h, t_h$ 
5:     for  $k = 1, \dots, a$  do
6:        $T_{j,k} \leftarrow T_{j,k} + v_h M_{k,i}$ 
7:    $\mathbf{u} \leftarrow \mathbf{0} \in \mathbb{R}^f$ 
8:   for  $h = 1, \dots, f$  do
9:      $i, j \leftarrow p_h, q_h$ 
10:    for  $k = 1, \dots, d$  do
11:       $u_h \leftarrow u_h + N_{j,k} T_{k,i}$ 
12: else
13:    $\mathbf{S} \leftarrow \mathbf{0} \in \mathbb{R}^{c \times b}$ 
14:   for  $h = 1, \dots, e$  do
15:      $i, j \leftarrow r_h, t_h$ 
16:     for  $k = 1, \dots, c$  do
17:        $S_{k,i} \leftarrow S_{k,i} + v_h N_{k,j}$ 
18:    $\mathbf{u} \leftarrow \mathbf{0} \in \mathbb{R}^f$ 
19:   for  $h = 1, \dots, f$  do
20:      $i, j \leftarrow p_h, q_h$ 
21:     for  $k = 1, \dots, b$  do
22:        $u_h \leftarrow u_h + S_{j,k} M_{i,k}$ 
23: return  $\mathbf{u}$ 

```

---

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

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

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

The above type of ordinary indexing matrices can of course be used with 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 index matrix). Let  $\mathbf{M} \in \mathbb{R}^{a \times b}$ ,  $\mathbf{N} \in \mathbb{R}^{c \times d}$  and let  $\mathbf{S}$  be an index matrix for the Kronecker product  $\mathbf{M} \otimes \mathbf{N}$ . Then,  $\mathbf{S}$  can be expressed as

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

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

$$p = \lfloor (i-1)/c \rfloor + 1 \quad q = (i-1) \bmod c + 1, \quad (2)$$

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. The column indexing matrices are defined analogously.

*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).  $\square$

**Theorem 1.** 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 index matrix and a column index matrix of  $\mathbf{M} \otimes \mathbf{N}$ , such that  $\mathbf{R}$  is determined by the sequences  $\mathbf{p} = (p_1, \dots, p_f)^T \in [a]^f$  and  $\mathbf{q} = (q_1, \dots, q_f)^T \in [c]^f$ ,

TABLE 1  
Notation concerning training set dimensions.

$n$	number of labeled edges in training set
$m$	number of unique start vertices connected to training edges
$q$	number of unique end vertices connected to training edges
$d$	number of start vertex features
$r$	number of end vertex features

and  $\mathbf{C}$  by  $\mathbf{r} = (r_1, \dots, r_e)^T \in [b]^e$  and  $\mathbf{t} = (t_1, \dots, t_e)^T \in [d]^e$ . Further, we assume that the sequences  $\mathbf{p}$ ,  $\mathbf{q}$ ,  $\mathbf{r}$  and  $\mathbf{t}$  when considered as mappings, are all surjective. Further, let  $\mathbf{v} \in \mathbb{R}^e$ . The product

$$\mathbf{R}(\mathbf{M} \otimes \mathbf{N})\mathbf{C}^T\mathbf{v} \quad (3)$$

can be computed in  $O(\min(ae + df, ce + bf))$  time using the generalized Vec trick (Algorithm 1).

*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  $\mathbf{V} \in \mathbb{R}^{d \times b}$  be a matrix such that  $\text{vec}(\mathbf{V}) = \mathbf{C}^T\mathbf{v}$ . Then, according to Lemma 1, (3) can be rewritten as

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

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  $\mathbf{T} = \mathbf{V}\mathbf{M}^T$  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. We note that the index matrices  $\mathbf{R}$  and  $\mathbf{C}$  are not explicitly given as input to the algorithm, because the elements of these sparse matrices are encoded in the vectors  $\mathbf{p}$ ,  $\mathbf{q}$ ,  $\mathbf{r}$  and  $\mathbf{t}$ .  $\square$

**Remark 1** (Vec trick as special case). If  $\mathbf{R} = \mathbf{C} = \mathbf{I}$ , the generalized vec trick reduces to the standard vec trick, and the complexity of Algorithm 1 is  $O(\min(abd + acd, bcd + abc))$ , the same as for the standard vec trick in Lemma 1.

### 3 GENERAL FRAMEWORK FOR KRONECKER PRODUCT KERNEL METHODS

In this section, we connect the above abstract considerations with practical learning problems with bipartite graph 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 edges with known labels. The notations are summarized in Table 1.

The training set consists of a labeled sequence of edges  $(\mathbf{d}_{r_h}, \mathbf{t}_{s_h}, y_h)_{h=1}^n \in (\mathcal{D} \times \mathcal{T} \times \mathcal{Y})^n$  of a labeled bipartite graph, each edge consisting of a start vertex  $\mathbf{d}_{r_h} \in \mathcal{D}$ , end vertex  $\mathbf{t}_{s_h} \in \mathcal{T}$  and an associated label  $y_h$ , where  $\mathcal{D}$  and  $\mathcal{T}$  are the spaces of start and end vertices, respectively, and  $\mathcal{Y}$  is the label space ( $\mathcal{Y} = \mathbb{R}$  for regression and  $\mathcal{Y} = \{-1, 1\}$  for binary classification). Moreover, let  $\{\mathbf{d}_i\}_{i=1}^m \subset \mathcal{D}$  and  $\{\mathbf{t}_i\}_{i=1}^q \subset \mathcal{T}$  denote the sets of such start vertices and end vertices, respectively, that are encountered in the training set as part of at least one edge. Further, let  $\mathbf{r} = (r_1, \dots, r_n)^T \in [q]^n$  and  $\mathbf{s} = (s_1, \dots, s_n)^T \in [m]^n$  be index sequences that map the training edges to their corresponding start and end vertices, respectively. Looking at Figure 1, one may think of  $r_i$  and  $s_i$  as row and column indices for the known entries in the start vertices times end vertices matrix, that correspond to the training data.

Let us assume a randomly drawn new sequence of edges  $(\mathbf{d}_i, \mathbf{t}_j)_{h=1}^t \in (\mathcal{D} \times \mathcal{T})^t$ , where  $1 < i < u$ ,  $1 < j < v$ , and the labels are unknown. Our aim is to learn from the training set a prediction function  $f : \mathcal{D} \times \mathcal{T} \rightarrow \mathcal{Y}$ , such that can correctly predict the labels of such edges. We further assume that the graphs corresponding to the train and test data are completely disconnected, that is, the sets of their start vertices are mutually disjoint and so are the sets of their end vertices. This is the main difference of our setting to the types of problems solved typically for example within the recommender systems literature, rather our setting corresponds to the so-called zero-shot, or cold-start problem where test vertices do not appear in the training set.

Let  $k : \mathcal{D} \times \mathcal{D} \rightarrow [0, \infty)$  and  $g : \mathcal{T} \times \mathcal{T} \rightarrow [0, \infty)$  denote positive semi-definite kernel functions [7] defined for the start and end vertices respectively. Further, the Kronecker product kernel  $k^\otimes(\mathbf{d}, \mathbf{t}, \mathbf{d}', \mathbf{t}') = k(\mathbf{d}, \mathbf{d}')g(\mathbf{t}, \mathbf{t}')$  for the edges 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 start and end vertex kernel matrices for the training data, respectively. Further, let  $\mathbf{R} \in \{0, 1\}^{n \times mq}$  be the Kronecker product index matrix determined by the start and end vertex index sequences  $\mathbf{r}$  and  $\mathbf{s}$ , as defined in Lemma 2. Then  $\mathbf{R}(\mathbf{G} \otimes \mathbf{K})\mathbf{R}^T$  is the Kronecker product kernel matrix containing the kernel evaluations corresponding to the training edges. Note that the methods to be introduced will never explicitly form this prohibitively large Kronecker product matrix, rather kernel matrix multiplications are implemented with Algorithm 1, that requires as input  $\mathbf{G}$  and  $\mathbf{K}$ , as well as the index sequences  $\mathbf{r}$  and  $\mathbf{s}$  that implicitly define  $\mathbf{R}$ .

We consider the regularized risk minimization problem

$$f^* = \operatorname{argmin}_{f \in \mathcal{H}} J(f) \quad (4)$$

with

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

where  $\mathbf{p} \in \mathbb{R}^n$  and  $\mathbf{y} \in \mathbb{R}^n$  are the predicted and correct outputs for the training set,  $\mathcal{L}$  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 [37] 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 start and end vertex spaces are real vector spaces, that is,  $\mathcal{D} = \mathbb{R}^d$  and  $\mathcal{T} = \mathbb{R}^r$ , and hence both the start and end vertices have a finite dimensional feature representation. Further, if the start and end vertex kernels are linear, the Kronecker product 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 edges 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 training set start and end vertices. Now the joint Kronecker feature representation 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

Next, we consider how predictions can be efficiently computed for graph data both with dual and primal predictors. This operation is necessary both during training when training set predictions are needed to evaluate the quality of the current predictor, as well as when applying the final trained predictor on new edges. Let us assume a new sequence of edges  $(\mathbf{d}_i, \mathbf{t}_j)_{h=1}^t \in (\mathcal{D} \times \mathcal{T})^t$ , where  $1 < i < u$ ,  $1 < j < v$ , and  $\mathbf{d}_i$  and  $\mathbf{t}_j$  may or may not overlap with the training set start and end vertices. Let  $\hat{\mathbf{K}} \in \mathbb{R}^{u \times m}$  and  $\hat{\mathbf{G}} \in \mathbb{R}^{v \times q}$  contain the kernel evaluations between the training start and end vertices, and the new start and end vertices, respectively. Further, the Kronecker product index matrix  $\hat{\mathbf{R}} \in \{0, 1\}^{t \times uv}$  encodes those new edges, for which the predictions are needed for.

For the dual model, the predictions can be computed as

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

resulting in a computational complexity of  $O(\min(vn + mt, un + qt))$ . 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  $\mathbf{a}$  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

$$O(\min(v\|\mathbf{a}\|_0 + mt, u\|\mathbf{a}\|_0 + qt)), \quad (5)$$

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

$$O(t\|\mathbf{a}\|_0) \quad (6)$$

complexity.

In the primal case the predictions can be computed as

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

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

TABLE 2

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 [38]	$\frac{1}{2} \sum_{i=1}^n (p_i - y_i)^2$	$p_i - y_i$	1	0
L1-SVM* [39]	$\sum_{i=1}^n \max(0, 1 - p_i \cdot y_i)$	$-y_i$ if $p_i \cdot y_i < 1$ 0 otherwise.	0	0
L2-SVM* [40]	$\frac{1}{2} \sum_{i=1}^n \max(0, 1 - p_i \cdot y_i)^2$	$p_i - y_i$ if $p_i \cdot y_i < 1$ 0 otherwise.	1 if $p_i \cdot y_i < 1$ 0 otherwise.	0
Logistic* [41]	$\sum_{i=1}^n \log(1 + e^{-y_i p_i})$	$-y_i(1 + e^{y_i p_i})^{-1}$	$e^{y_i p_i}(1 + e^{y_i p_i})^{-2}$	0
RankRLS [42]	$\frac{1}{4} \sum_{i=1}^n \sum_{j=1}^n (y_i - p_i - y_j + p_j)^2$	$\sum_{j=1}^n (y_j - p_j) + n(p_i - y_i)$	$n - 1$	-1

### 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 product kernel. For non-smooth losses or losses with non-smooth derivatives, we may instead consider subgradients and the generalized Hessian matrix (see e.g. [40], [43], [44]). 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

$$\mathbf{p} = \mathbf{R}(\mathbf{G} \otimes \mathbf{K})\mathbf{R}^T \mathbf{a}. \quad (7)$$

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

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 2 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. [42] for examples of efficiently decomposable ranking losses).

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

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

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

$$\frac{\partial J}{\partial \mathbf{a}} = \mathbf{R}(\mathbf{G} \otimes \mathbf{K})\mathbf{R}^T (\mathbf{g} + \lambda \mathbf{a}) \quad (8)$$

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

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

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

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

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

---

**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:    $\mathbf{p} \leftarrow \mathbf{R}(\mathbf{G} \otimes \mathbf{K})\mathbf{R}^T \mathbf{a}$
- 4:   COMPUTE  $\mathbf{H}$ ,  $\mathbf{g}$
- 5:   SOLVE  $(\mathbf{H}\mathbf{R}(\mathbf{G} \otimes \mathbf{K})\mathbf{R}^T + \lambda \mathbf{I})\mathbf{x} = \mathbf{g} + \lambda \mathbf{a}$
- 6:    $\mathbf{a} \leftarrow \mathbf{a} - \delta \mathbf{x}$  ( $\delta$  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:    $\mathbf{p} \leftarrow \mathbf{R}(\mathbf{T} \otimes \mathbf{D})\mathbf{w}$
- 4:   COMPUTE  $\mathbf{H}$ ,  $\mathbf{g}$
- 5:   SOLVE  $((\mathbf{T}^T \otimes \mathbf{D}^T)\mathbf{R}^T \mathbf{H}\mathbf{R}(\mathbf{T} \otimes \mathbf{D}) + \lambda \mathbf{I})\mathbf{x} = (\mathbf{T}^T \otimes \mathbf{D}^T)\mathbf{R}^T \mathbf{g} + \lambda \mathbf{w}$ .
- 6:    $\mathbf{w} \leftarrow \mathbf{w} - \delta \mathbf{x}$  ( $\delta$  constant or found by line search)
- 7: **until** convergence

---

### 3.3 Optimization framework

As a simple approach that can be used for training regularized risk minimization methods with access to gradient and (generalized) Hessian-vector product operations, we consider a truncated Newton optimization scheme (Algorithms 2 and 3; these implement approaches similar to [40], [45]). 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 product kernel method optimization, since while computing the Hessians explicitly is often not computationally feasible, computing Hessian-vector products can be done efficiently using the generalized vec trick algorithm.

Alternative optimization schemes can certainly be used, such as the Limited-memory BFGS algorithm [46] or trust-region Newton optimization [43]. Further, for non-smooth losses such as the hinge loss, methods tailored for non-smooth optimization such as the bundle method [44] should be used rather than the truncated Newton method. Such works are 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 edges or the model coefficients individually or in small batches (e.g. stochastic gradient descent [47], coordinate descent [48], SVM decomposition methods [49]) 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 2 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 [50] used in our experiments. For the dual case, we may simplify this system of equations by removing the common term  $\mathbf{R}(\mathbf{G} \otimes \mathbf{K})\mathbf{R}^T$  from both sides, resulting in the system

$$(\mathbf{H}\mathbf{R}(\mathbf{G} \otimes \mathbf{K})\mathbf{R}^T + \lambda \mathbf{I})\mathbf{x} = \mathbf{g} + \lambda \mathbf{a}. \quad (9)$$

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. [51], [52], [53]).

### 3.4 Complexity comparison

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



TABLE 3

Dual case: complexity comparison of the proposed method and the baseline approach that constructs the Kronecker product 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 4

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	$O(mqdr)$	$O(\min(mdr + mqr, drq + dmq))$

We consider three different settings:

- Independent: all edges in the training set are mutually disjoint, indicating that  $n = m = q$ .
- Dependent: training edges may share start or end vertices or both with each other, that is  $\max(m, q) < n < mq$ .
- Complete: the training set is a complete bipartite graph, 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 3 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 vertices shared between the edges, that the algorithm could make use of. However, beyond that the complexity of the baseline approach grows quadratically with respect to the number of edges, while the complexity of the proposed method grows linearly with respect to the numbers of start vertices, end vertices, and edges. In Table 4 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  $m \ll n$  and  $q \ll n$ . 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). To conclude, we observe that the proposed algorithm considerably outperforms previously known methods, if  $\max(m, q) \ll n < mq$ .

## 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 product 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) [38], [54]. 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,  $\mathcal{L} = \frac{1}{2}(\mathbf{p} - \mathbf{y})^T(\mathbf{p} - \mathbf{y})$ ,  $\mathbf{H} = \mathbf{I}$  and  $\mathbf{g} = \mathbf{p} - \mathbf{y}$  (see Table 2). For the dual case, based on (8) it can be observed that the full gradient for the ridge regression is

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

Writing  $\mathbf{p}$  open (7) and setting the gradient to zero, we recover a linear system

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

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

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

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

For the primal case, the gradient can be expressed as

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

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

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

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

The naive approach of training ridge regression with the explicitly formed Kronecker product kernel or data matrix using standard solvers of systems of linear equations would result in  $O(n^2)$  and  $O(n dr)$  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 2). Following works such as [40], [45], [43] 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 subset of training set that have non-zero loss for given prediction function. Further, let  $\mathbf{S}_{\mathcal{S}}$  denote the index matrix corresponding to this index set. Now we can re-write the loss in a least-squares form as  $\mathcal{L} = \frac{1}{2} (\mathbf{S}_{\mathcal{S}} \mathbf{p} - \mathbf{S}_{\mathcal{S}} \mathbf{y})^T (\mathbf{S}_{\mathcal{S}} \mathbf{p} - \mathbf{S}_{\mathcal{S}} \mathbf{y})$ , its gradient as  $\mathbf{g} = \mathbf{S}_{\mathcal{S}}^T (\mathbf{S}_{\mathcal{S}} \mathbf{p} - \mathbf{S}_{\mathcal{S}} \mathbf{y})$ , and the Hessian as  $\mathbf{H} = \mathbf{S}_{\mathcal{S}}^T \mathbf{S}_{\mathcal{S}}$ , which is a diagonal matrix with entry 1 for all members of  $\mathcal{S}$ , and 0 otherwise.

By inserting  $\mathbf{g}$  to (8) and writing  $\mathbf{p}$  open (7) we recover the gradient of the L2-SVM objective function, with respect to  $\mathbf{a}$ , as

$$\mathbf{R}(\mathbf{G} \otimes \mathbf{K}) \mathbf{R}^T (\mathbf{S}_{\mathcal{S}}^T (\mathbf{S}_{\mathcal{S}} \mathbf{R}(\mathbf{G} \otimes \mathbf{K}) \mathbf{R}^T \mathbf{a} - \mathbf{S}_{\mathcal{S}} \mathbf{y}) + \lambda \mathbf{a}) \quad (10)$$

Inserting  $\mathbf{g}$  and  $\mathbf{H}$  to (9), we see that  $\frac{\partial J}{\partial \mathbf{a}} \frac{\partial J}{\partial \mathbf{a}} \mathbf{x} = \frac{\partial J}{\partial \mathbf{a}}$  can be solved from:

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

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  $\mathbf{p} = \mathbf{R}(\mathbf{G} \otimes \mathbf{K}) \mathbf{R}^T \mathbf{a}$  and thus also the right-hand size of (4.2) 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 (4.2) with Truncated Newton optimization, we can in each inner iteration of the method compute matrix-vector multiplication  $\mathbf{S}_{\mathcal{S}} \mathbf{R}(\mathbf{G} \otimes \mathbf{K}) \mathbf{R}^T \mathbf{v}$  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

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

A generalized Hessian matrix can be defined as

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

The most efficient existing SVM solvers can be expected at best to scale quadratically with respect to training set size when solving the dual problem, and linearly with respect to training set size and number of features when solving the primal case [55], [56]. Thus using the full Kronecker product kernel or data matrices, such solvers would have  $O(n^2)$  and  $O(n dr)$  scaling for the dual and primal cases, respectively.

## 5 EXPERIMENTS

Computational costs for iterative Kronecker product kernel method training depend on two factors: the cost of a single iteration, and the number of iterations needed to reach a good solution. The cost of a single iteration is dominated by gradient computations and Hessian-vector products. These can be efficiently performed with Algorithm 1. Further, as discussed at the end of Section 3.3, the number of iterations needed can be limited via early stopping once a predictor that works well on independent validation data has been reached. 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 Algorithm 1.

In the experiments, we demonstrate the following:

- 1) Parameter selection: we show how the hyperparameters of the methods can be tuned using validation data.
- 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.
- 4) Accurate predictions: the predictive accuracy of Kronecker models is competitive compared to alternative types of scalable graph learning methods.

We implement the Kronecker methods in Python, they are made freely available under open source license as part of the RLScore machine learning library<sup>1</sup> [57]. For comparison, we consider the LibSVM software [58], which implements a highly efficient support vector machine training algorithm [49]. Further, we compare our approach to stochastic gradient descent and K-nearest neighbor based graph prediction methods.

## 5.1 Data and general setup

TABLE 5  
Data sets

Data set	edges	pos.	neg.	start vertices	end vertices
Ki	93356	3200	90156	1421	156
GPCR	5296	165	5131	223	95
IC	10710	369	10341	210	204
E	73870	732	73138	445	664
Checker	250000	125000	12500	1000	1000
Checker+	10240000	5120000	5120000	6400	6400

As a typical bipartite graph learning problem, 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. We consider four drug-target interaction data sets, the GPCR, IC, E data sets [59], and the Ki data [60]. The Ki-data is a naturally sparse data set where the class information is available only for a subset of the edges. For the other four drug-target data sets we use for the experiments a subset of all the possible drug-target interactions. We use exactly the same preprocessing of the data as [3]. The characteristics of the data sets are described in Table 5. The edges in the data are  $(\mathbf{d}_i, \mathbf{t}_j, y_h)$  triplets, consisting of a pair of vectors encoding the features of the drug and the target (see [3] for details of the features), and a label having value 1 if the drug and target interact, and  $-1$  if they do not. Mapping the problem to the terminology used in this paper, we may consider the drugs as start vertices, the targets as end vertices, and the drug-target pairs with known interaction or non-interaction as edges.

Further, we generated two data sets using a variant of the Checkerboard simulation, a standard non-linear problem for benchmarking large scale SVM solvers (see e.g. [61]). In our simulation both start and end vertices have a single feature describing them, drawn from continuous uniform distribution in range  $(0, 100)$ . The output assigned to an edge  $(d, t)$  is  $+1$  whenever both  $\lfloor d \rfloor$  and  $\lfloor t \rfloor$  are either odd or even, and  $-1$  when one of them is odd and the other even. When plotting the output against  $d$  and  $t$ , this results in a highly non-linear checkerboard-type of pattern. Finally, random noise is introduced to the data by flipping with 0.2 probability the class of each edge. The numbers of start and end vertices are same, and labels are assigned for 25% of all the possible edges (i.e.  $m = q$  and  $n = 0.25m^2$ ).

In the zero-shot learning setting the aim is to generalize to such  $(\mathbf{d}_i, \mathbf{t}_j)$ -edges that are vertex disjoint with the training graph. Therefore, cross-validation with graph data is more complicated than for standard i.i.d. data, since this aim must be reflected in the train-test split (see e.g. [1], [3]). The split is illustrated in Figure 2. To ensure that the training and test graphs are vertex disjoint, the edges are divided into training and test edges as follows. First, both the start vertex-indices  $[1, \dots, m]$  and end vertex-indices  $[1, \dots, q]$  are randomly divided into a training and test part. Then, an edge  $(\mathbf{d}_i, \mathbf{t}_j, y_h)$  is assigned to training set if  $i$  belongs to the training start vertex indices and  $j$  belongs to the training end vertex indices. It is assigned to the test set if  $i$  belongs to the test start vertex indices and  $j$  belongs to test end vertex indices. Finally, the rest of the edges are discarded, that is, they belong neither to the training nor test part (the greyed out blocks in Figure 2). To tune the hyper-parameters without the risk of overfitting, one can split the data into training, validation and test parts in an analogous way to the above described train-test split. Combining this with cross-validation is in the literature known as nested cross-validation (for a detailed description of this approach for graph learning, see [3]).

Ridge regression is trained with the minimum residual iteration algorithm [62] implemented in the `scipy.sparse.linalg.minres` package, while the inner optimization loop of the SVM training algorithm uses the `scipy.sparse.linalg.qmr` implementing of quasi-minimal residual iteration [50] (SciPy version 0.14.1), with  $\delta = 1$ . Regularization parameters from the grid  $[2^{-20}, \dots, 2^{20}]$  were tested. We restrict our plots to values  $[2^{-10}, 2^{-5}, 2^0, 2^5, 2^{10}]$ , as these allow representing all the main trends in the experiments. The classification performance on test data was measured with area under ROC curve (AUC). The experiments were run on a desktop computer with Intel Core i7-3770 CPU (3.40GHz) running Ubuntu Linux 15.04.

Early stopping experiments were run with the linear kernel, in order to allow comparing dual and primal optimization. The LibSVM comparison was done using the Gaussian kernel. LibSVM does not directly support the Kronecker product kernel, this issue was resolved as follows. If both start vertex and end vertex kernels are Gaussian with width  $\gamma$ , then  $k(\mathbf{d}, \mathbf{d}')k(\mathbf{t}, \mathbf{t}') = e^{-\gamma\|\mathbf{t}-\mathbf{t}'\|^2}e^{-\gamma\|\mathbf{d}-\mathbf{d}'\|^2} = e^{-\gamma\|[\mathbf{t}, \mathbf{d}]-[\mathbf{t}', \mathbf{d}']\|^2}$ , that is, the Kronecker product kernel is equal to using the Gaussian kernel with concatenated features of the start and end vertex.

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

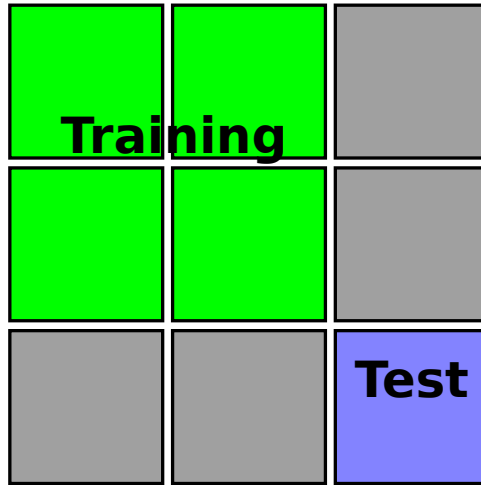


Fig. 2. Ninefold cross-validation. The matrix rows represent start vertices and columns end vertices, the edges with known labels corresponds to a subset of the elements of this matrix. The row- and column indices are both divided into three sets. The Cartesian product sets of the row and column sets index nine blocks of possible edges. On each round, the test fold is formed from all the edges that belong to one of the blocks, and further have also known label. The training folds are formed from the four blocks, that share no common rows or columns with the test fold. Four blocks are left unused, as their edges connect vertices belonging to training and test folds.

## 5.2 Choosing hyperparameters

In this section, we set up guidelines for selecting the values of the hyperparameters based on experimental verification. The Kronecker algorithms have the following hyperparameters: the regularization parameter, the number of iterations for ridge regression, and both the number of inner and outer iterations for the SVM. We consider only the dual optimization, the observed behavior was very similar also for the primal case.

We run the optimization up to 100 iterations measuring regularized risk  $J(f)$ , and AUC on the test set (in selected experiments with 500 iterations we found little improvements). Ridge regression results are presented in Figure 3, while SVM results with inner optimization loop terminated after 10 and 100 iterations are presented in Figures 4 and 5.

In all the experiments 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 product kernel methods, often only a handful of iterations need to be performed in order to obtain maximal predictive accuracy.

Several observations can be made based on the experiments. The regularized risk decreases quite quickly even if the SOLVE operation in Algorithms 2 and 3 is terminated after a small number of iterations. Moreover, early termination of SOLVE provides us more fine-grained control of the degree of fitting (contrast Figures 4 and 5, where for 100 inner iterations test performance in some cases starts immediately decreasing).

To conclude, the results suggest that on large data sets a good strategy is to start by setting the number of iterations to a small constant (e.g. 10 inner and outer iterations), and increase these parameters only if the predictive accuracy keeps increasing beyond the first iterations on independent validation set. Furthermore, we note that one could sidestep the selection of regularization parameter  $\lambda$  by setting it to a small constant and regularizing only with early stopping, however tuning also  $\lambda$  on separate validation data can sometimes yield even better performance.

## 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 start and end vertices. 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 it vary more based on the choice of regularization parameter. We perform 9-fold cross-validation on the  $K_i$  data, for various training set sizes. In Figure 6 (left) we present the running times for training the methods for different numbers of training edges, 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 edges the

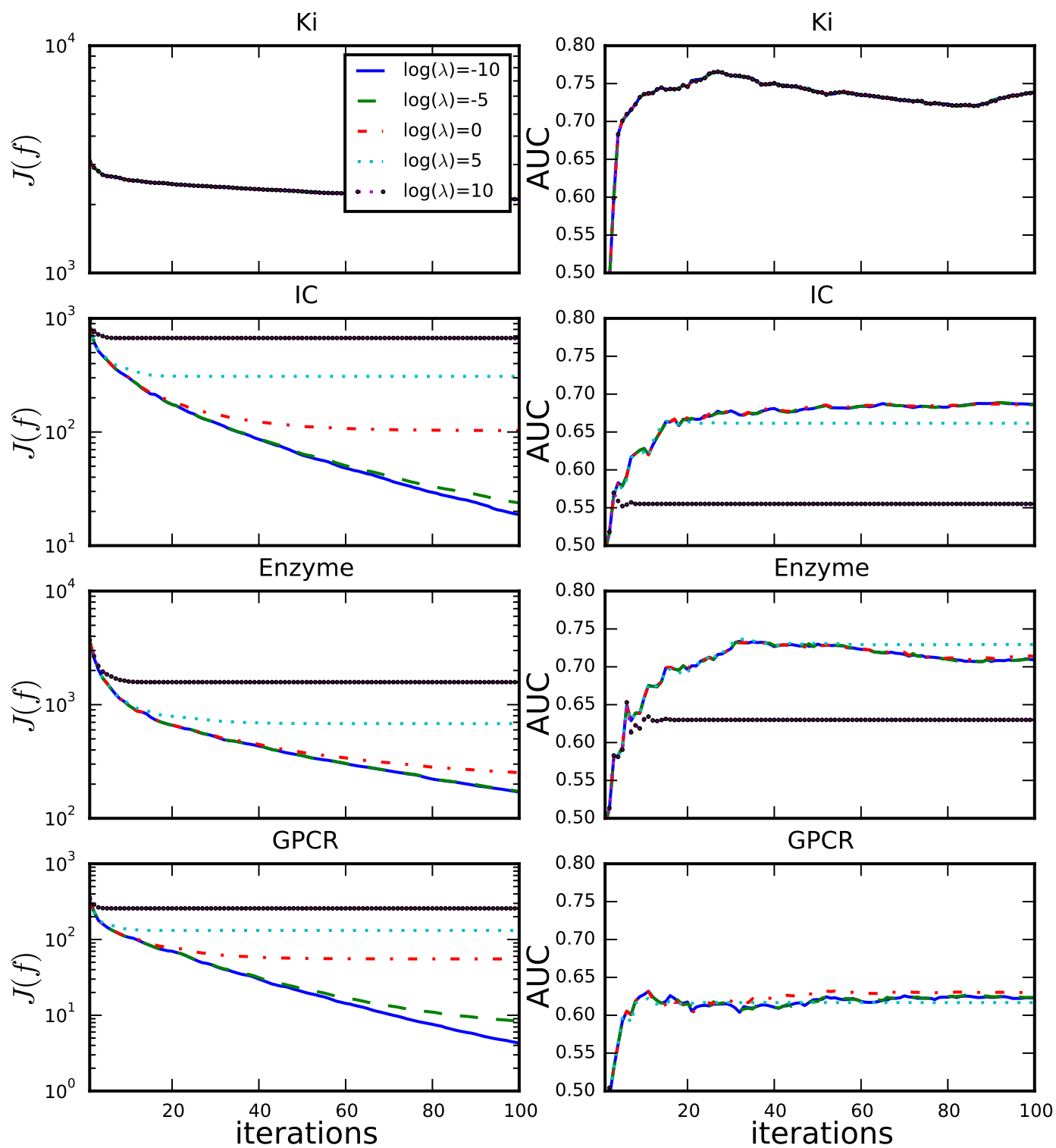


Fig. 3. Ridge regression regularized risk (left) and test set AUC (right) as a function of optimization iterations.

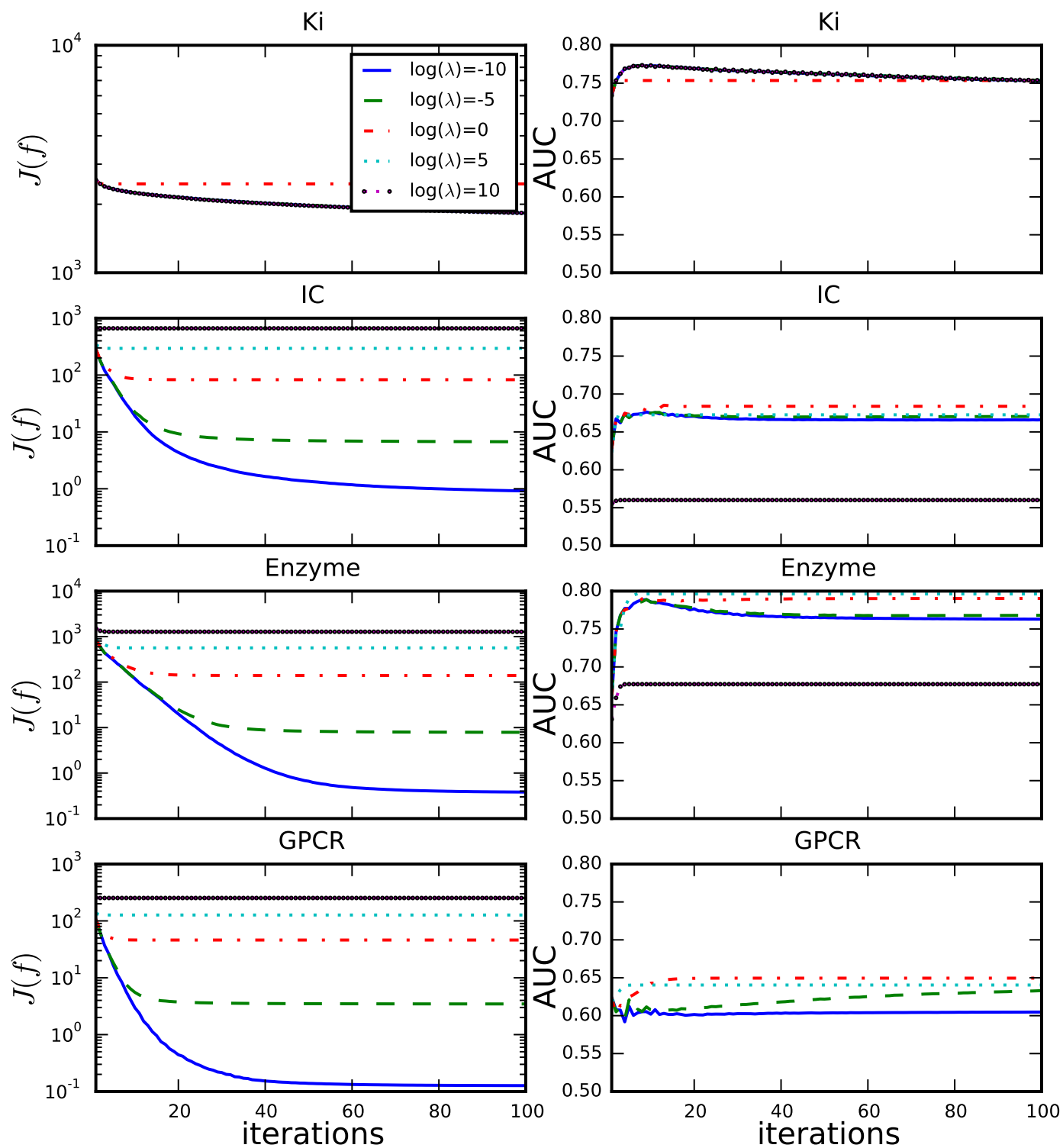


Fig. 4. SVM with 10 inner iterations. Regularized risk (left) and test set AUC (right) as a function of outer optimization iterations.

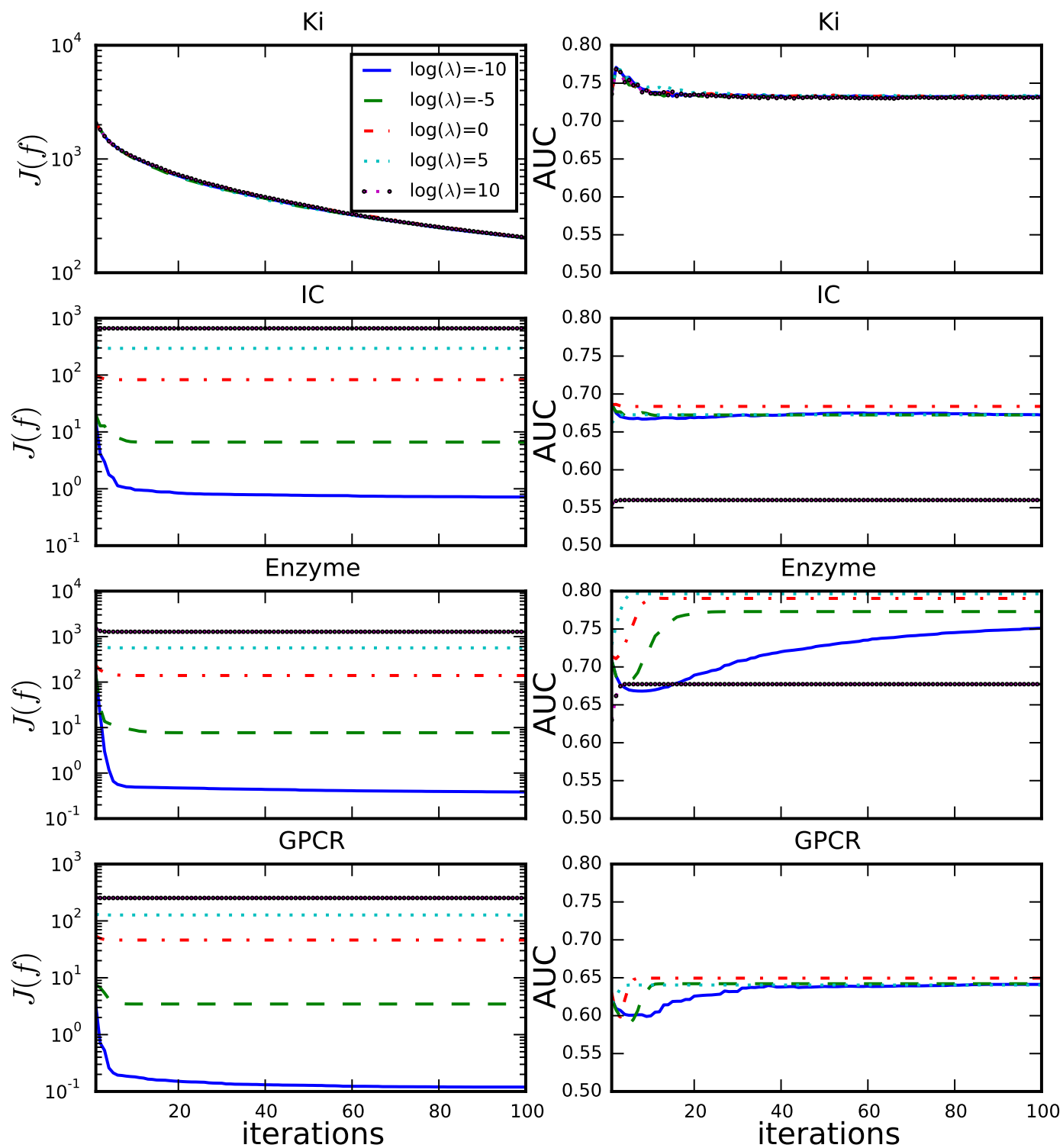


Fig. 5. SVM with 100 inner iterations. Regularized risk (left) and test set AUC (right) as a function of outer optimization iterations.

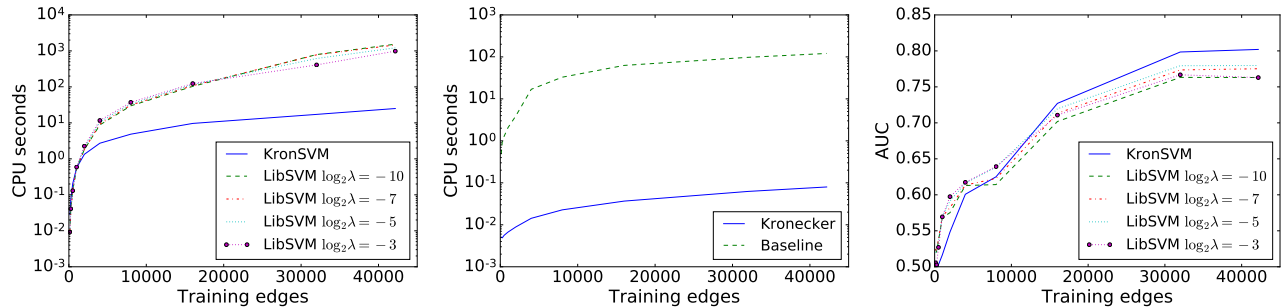


Fig. 6. Drug-target experiment. Runtime comparison between KronSVM and LIBSVM (left). Prediction times for regular LibSVM decision function, and one that uses sparse Kronecker product shortcuts (middle). Cross-validated AUCs (right).

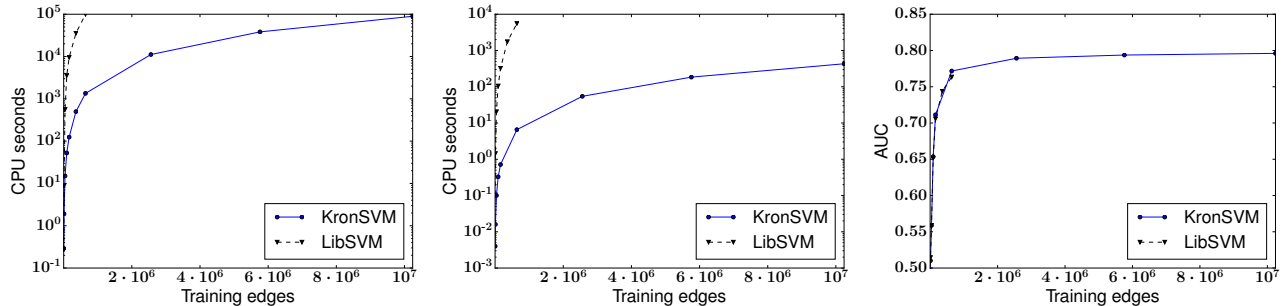


Fig. 7. Checkerboard simulation. Runtime comparison between KronSVM and LIBSVM training (left) and prediction times (middle), and the corresponding test set AUCs (right).

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 generalized Kronecker product algorithm (Algorithm 1) proposed in our work, one will need to construct a significant part of the kernel matrix for the edges. For LibSVM, the scaling is roughly quadratic in the number of edges, 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 graph data than if one uses current state-of-the-art solvers, that are however not able to make use of the shared structure.

#### 5.4 Prediction time

Next, 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 data set sizes, with the training and test 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 (5) and (6)). 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.

#### 5.5 Large scale experiments with simulated data

In order to demonstrate the scaling of the proposed Kronecker product kernel methods to larger problem sizes than those encountered in the drug-target data sets, we implemented scalability experiments on various sized subsets of the checkerboard data. The training set is formed as follows. First, we generate the same number of start vertices  $m$  and end vertices  $q$ , then generate The independent test set is generated in the same way. We use the Gaussian kernel, based on preliminary tests we set  $\lambda = 2^{-7}$  and  $\gamma = 1$ , as parameters around this range allow learning the simulated function. As



TABLE 6  
AUCs for learning methods

	Ki	GPCR	IC	E	Checker	Checker+
KronSVM	<b>0.77</b>	0.62	0.68	<b>0.79</b>	<b>0.73</b>	<b>0.80</b>
KronRidge	0.75	0.62	<b>0.69</b>	0.72	0.71	0.79
SGD hinge	0.76	0.60	0.63	0.72	0.50	0.50
SGD logistic	0.76	<b>0.67</b>	0.64	0.72	0.50	0.50
KNN	0.71	0.63	0.68	0.70	0.68	0.79

TABLE 7  
CPU runtime in seconds for learning methods

	Ki	GPCR	IC	E	Checker	Checker+
KronSVM	298	7	14	191	371	89500
KronRidge	68	4	8	60	188	45150
SGD hinge	57	14	19	41	17	130
SGD logistic	63	19	25	57	17	128
KNN	5554	43	267	26457	46	1756

before, KronSVM uses 10 inner and 10 outer iterations. We train both KronSVM and LibSVM for varying data set sizes, up until they reach the point where training takes more than 24 hours to complete. We also measure how long computing predictions takes for a test set of the same size as the training set. Finally, we also measure test set AUC in order to show that the learners can really learn to solve the simulated problem (note that due to random flipping of classes, even the optimal predictor would have only 0.8 AUC). The results are presented in Figure 7.

Again, KronSVM outperforms LibSVM by several orders of magnitude. KronSVM can be trained in 24 hours on approximately 10 million edges (correspondingly, with 6400 start and end vertices). The LibSVM experiments were discontinued after training on 64000 edges (correspondingly 1600 start and end vertices) took more than 27 hours. For the same training set size, KronSVM can be trained in 23 minutes. Regarding prediction times, for LibSVM model trained with 64000 edges it took 93 minutes to compute predictions for a test set of same size, whereas with the generalized Kronecker product shortcuts the same computations can be done in 7 seconds. KronSVM can with a model trained on 10 million edges, make predictions for a test set of also 10-million edges in 7 minutes. When trained on 10 million edges, the KronSVM implementation used roughly 1.5 Gigabytes of memory.

## 5.6 Comparison of graph learning methods

Finally, we compare the proposed approach to alternative scalable graph learning approaches, such that can generalize to making predictions for edges that are vertex disjoint with the training graph. We consider the following baseline methods, that use as feature representation the concatenation  $[d, t]$  of the start and end vertex features:

- Linear model, stochastic gradient descent (SGD) [47]: We fit a linear model  $f(d, t) = \langle w, [d, t] \rangle$  to the data using stochastic gradient descent over the edges. The approach is extremely scalable, and it is not necessary to load all data into memory at once. Previously, [4] has used sgd with logistic regression for cold start learning with recommender systems. We consider both the logistic and the hinge loss.
- K-nearest neighbors (KNN): KNN methods have enjoyed substantial popularity in graph prediction applications such as biological interaction prediction [63] and recommender systems [64]. The method can model highly non-linear functions and can scale well especially to low-dimensional problems by using efficient data structures for speeding up the neighborhood-search.

For the baseline methods, we use the implementations from the scikit-learn package [65]. The sgd regularization parameter and KNN number of neighbors parameters are selected with internal 3-fold cross-validation. The number of stochastic gradient descent updates is set to  $10^6$  (or at minimum one full pass through data). For KronSVM and KronRidge we set  $\lambda = 0.0001$ , as the methods were not very sensitive to amount of regularization when optimization was terminated early. KronSVM uses 10 inner and 10 outer iterations on each data set, while KronRidge uses 100 iterations. As before, for Checker data sets the Kronecker methods use Gaussian kernel with  $\gamma = 1$  for both vertex kernels, for other data sets we use linear vertex kernels. For the drug-target interaction data sets we use 3x3 fold cross-validation as before, for the checkerboard data sets we generate separate test set with 6250000 edges.

In Table 6 we present the AUCs for the compared methods. Overall, KronSVM performs the best, yielding the best performance on Ki, E, Checker and Checker+ data sets. KronRidge performs slightly worse than KronSVM on most data sets, possibly due to the fact that the squared loss is not as well suited for classification as the squared hinge loss. While the linear SGD methods provide a surprisingly competitive baseline, they do not quite reach the performance of the best methods, and it is impossible for them to outperform random guessing for the Checker data sets, due to the non-linearity of the task. The KNN method performs reliably over all the data sets, but does not yield the best performance on any of them.

Regarding the runtimes presented in Table 7, KronSVM and KronRidge strike a good balance between accurate predictions and the ability to scale to all the data sets. The linear SGD methods provide overall the best scalability, but at the cost of not being able to model nonlinearities in the data. The scalability of the KNN depends on the dimensionality of the data; on Checker and Checker+ the method excels because there are only 2 features, whereas on Ki, IC, E, and GPCR, the method is not competitive.

## 6 CONCLUSION

In this work, we have proposed a generalized 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. Further, we show that the approach compares favorably to other types of graph learning methods. The implementations for the generalized Kronecker product algorithm, as well as for Kronecker product kernel learners are made freely available under open source license.

## REFERENCES

- [1] 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.
- [2] M. Schrynmackers, R. Küffner, and P. Geurts, “On protocols and measures for the validation of supervised methods for the inference of biological networks,” *Frontiers in Genetics*, vol. 4, no. 262, pp. 1–16, 2013.
- [3] T. Pahikkala, A. Airola, S. Pietil, S. Shakyawar, A. Szwejda, J. Tang, and T. Aittokallio, “Toward more realistic drug-target interaction predictions,” *Briefings in Bioinformatics*, vol. 16, no. 2, pp. 325–337, 2015.
- [4] A. Menon and C. Elkan, “A log-linear model with latent features for dyadic prediction,” in *The 10th IEEE International Conference on Data Mining (ICDM)*, 2010, pp. 364–373.
- [5] M. Palatucci, D. Pomerleau, G. E. Hinton, and T. M. Mitchell, “Zero-shot learning with semantic output codes,” in *Advances in Neural Information Processing Systems 22*, Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta, Eds. Curran Associates, Inc., 2009, pp. 1410–1418.
- [6] 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.
- [7] 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.
- [8] 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.
- [9] A. Ben-Hur and W. Noble, “Kernel methods for predicting protein-protein interactions,” *Bioinformatics*, vol. 21 Suppl 1, pp. 38–46, 2005.
- [10] 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.
- [11] 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.
- [12] 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.
- [13] 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.
- [14] 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.
- [15] 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.
- [16] 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.
- [17] 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.
- [18] W. E. Roth, “On direct product matrices,” *Bulletin of the American Mathematical Society*, vol. 40, pp. 461–468, 1934.
- [19] 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.
- [20] Y. Koren, R. Bell, and C. Volinsky, “Matrix factorization techniques for recommender systems,” *Computer*, vol. 42, no. 8, pp. 30–37, Aug 2009.
- [21] T.-Y. Liu, *Learning to Rank for Information Retrieval*. Springer, 2011.
- [22] X. Peng, H. Tang, L. Zhang, Z. Yi, and S. Xiao, “A unified framework for representation-based subspace clustering of out-of-sample and large-scale data,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 27, no. 12, pp. 2499–2512, 2016.
- [23] X. Peng, Z. Yu, Z. Yi, and H. Tang, “Constructing the l2-graph for robust subspace learning and subspace clustering,” *IEEE Transactions on Cybernetics*, vol. 47, no. 4, pp. 1053–1066, 2017.
- [24] Jayadeva, S. Shah, and S. Chandra, “Kernel optimization using a generalized eigenvalue approach,” in *Proceedings of the third international conference on Pattern Recognition and Machine Intelligence*, ser. Lecture Notes in Computer Science, S. Chaudhury, S. Mitra, C. A. Murthy, P. S. Sastry, and S. K. Pal, Eds., vol. 5909. Berlin, Heidelberg: Springer, 2009, pp. 32–37.
- [25] 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.
- [26] Jayadeva, R. Khemchandani, and S. Chandra, “Regularized least squares support vector regression for the simultaneous learning of a function and its derivatives,” *Information Sciences*, vol. 178, no. 17, pp. 3402 – 3414, 2008.

- [27] M. A. Alvarez, L. Rosasco, and N. D. Lawrence, "Kernels for vector-valued functions: A review," *Foundations and Trends in Machine Learning*, vol. 4, no. 3, pp. 195–266, 2012.
- [28] S. Oyama and C. D. Manning, "Using feature conjunctions across examples for learning pairwise classifiers," in *Machine Learning: ECML 2004: 15th European Conference on Machine Learning, Pisa, Italy, September 20–24, 2004. Proceedings*, J.-F. Boulicaut, F. Esposito, F. Giannotti, and D. Pedreschi, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 322–333.
- [29] T. van Laarhoven, S. B. Nabuurs, and E. Marchiori, "Gaussian interaction profile kernels for predicting drugtarget interaction," *Bioinformatics*, vol. 27, no. 21, pp. 3036–3043, 2011.
- [30] M. Gönen, "Predicting drugtarget interactions from chemical and genomic kernels using bayesian matrix factorization," *Bioinformatics*, vol. 28, no. 18, pp. 2304–2310, 2012.
- [31] 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.
- [32] I. Steinwart, "Consistency of support vector machines and other regularized kernel classifiers," *IEEE Transactions on Information Theory*, vol. 51, no. 1, pp. 128–142, 2005.
- [33] 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.
- [34] 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.
- [35] D. Schäfer and E. Hüllermeier, "Dyad ranking using a bilinear plackett-luce model," in *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2015, Porto, Portugal, September 7–11, 2015, Proceedings, Part II*, A. Appice, P. P. Rodrigues, V. Santos Costa, J. Gama, A. Jorge, and C. Soares, Eds. Springer International Publishing, 2015, pp. 227–242.
- [36] 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.
- [37] B. Schölkopf, R. Herbrich, and A. J. Smola, "A generalized representer theorem," in *Proceedings of the 14th Annual Conference on Computational Learning Theory and 5th European Conference on Computational Learning Theory*, D. Helmbold and B. Williamson, Eds. London, UK, UK: Springer-Verlag, 2001, pp. 416–426.
- [38] A. E. Hoerl and R. W. Kennard, "Ridge regression: Biased estimation for nonorthogonal problems," *Technometrics*, vol. 12, pp. 55–67, 1970.
- [39] V. N. Vapnik, *The nature of statistical learning theory*. New York, NY, USA: Springer-Verlag New York, Inc., 1995.
- [40] 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.
- [41] 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.
- [42] 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.
- [43] 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.
- [44] C. H. Teo, S. Vishwanathan, 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.
- [45] O. Chapelle, "Training a support vector machine in the primal," *Neural Computation*, vol. 19, no. 5, pp. 1155–1178, 2007.
- [46] 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.
- [47] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of the 19th International Conference on Computational Statistics*, Y. Lechevallier and G. Saporta, Eds. Paris, France: Springer, August 2010, pp. 177–187.
- [48] 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.
- [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] 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.
- [51] Y. Yao, L. Rosasco, and A. Caponnetto, "On early stopping in gradient descent learning," *Constructive Approximation*, vol. 26, no. 2, pp. 289–315, 2007.
- [52] 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.
- [53] 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.
- [54] 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.
- [55] 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.
- [56] 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.
- [57] T. Pahikkala and A. Airola, "Rlscore: Regularized least-squares learners," *Journal of Machine Learning Research*, vol. 17, pp. 1–5, 2016.
- [58] 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.
- [59] 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.
- [60] 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.
- [61] O. Mangasarian and D. R. Musicant, "Lagrangian support vector machines," *Journal of Machine Learning Research*, vol. 1, pp. 161–177, 2001.
- [62] 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.
- [63] T. van Laarhoven and E. Marchiori, "Predicting drug-target interactions for new drug compounds using a weighted nearest neighbor profile," *PLOS ONE*, vol. 8, no. 6, pp. 1–6, 06 2013.
- [64] C. Desrosiers and G. Karypis, "A comprehensive survey of neighborhood-based recommendation methods," in *Recommender Systems Handbook*, F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, Eds. Boston, MA: Springer US, 2011, pp. 107–144.

- [65] 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*, vol. 12, pp. 2825–2830, 2011.