Fast Gradient Computation for Learning with Tensor Product Kernels and Sparse Training Labels

Tapio Pahikkala

Department of Information Technology, University of Turku
Lemminkäisenkatu 14 A, FIN-20520 Turku, Finland
tapio.pahikkala@utu.fi

Abstract. Supervised learning with pair-input data has recently become one of the most intensively studied topics in pattern recognition literature, and its applications are numerous, including, for example, collaborative filtering, information retrieval, and drug-target interaction prediction. Regularized least-squares (RLS) is a kernel-based learning algorithm that, together with tensor product kernels, is a successful tool for solving pair-input learning problems, especially the ones in which the aim is to generalize to new types of inputs not encountered in during the training phase. The training of tensor kernel RLS models for pair-input problems has been traditionally accelerated with the so-called vec-trick. We show that it can be further accelerated by taking advantage of the sparsity of the training labels. This speed improvement is demonstrated in a running time experiment and the applicability of the algorithm in a practical problem of predicting drug-target interactions.

1 Introduction

In supervised learning, such as regression, classification and ranking, one is given a training data comprised of a sequence $S = \{\mathbf{x}_h\}_{h=1}^{n}$ of inputs and a vector $\mathbf{y} \in \mathbb{R}^n$ consisting of their real-valued labels. Here, we consider learning tasks in which the inputs are paired, a property that is characterized by the following two circumstances. Firstly, the 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. Namely, $\mathbf{x} = (\mathbf{d}, \mathbf{t})$, where $\mathbf{d} \in \mathcal{D}$, $\mathbf{t} \in \mathcal{T}$, and $\mathcal{D}$ and $\mathcal{T}$ are sets consisting of all possible tasks and data points, respectively. Secondly, the data with known labels tends to be available in sets, in which both parts of a single input are likely to also appear as parts of other inputs, that is, the input sequence for training contains several inputs associated to the same task part and several inputs associated to the same data point part.

Typical examples of learning problems in which this type of split makes sense can be found, for example, in the fields of recommender systems, where the inputs consist of customers and products (Basilico and Hofmann, 2004), information retrieval, where they consist of queries and data to be retrieved (Liu, 2011),

biochemical interaction prediction, where the inputs can be split, for instance, to drugs and targets (see e.g. Ding et al. (2013) for a recent review), prediction of game outcomes (Pahikkala et al., 2010b), and several types of multi-task learning problems involving task-specific features (see e.g. Bonilla et al. (2007); Hayashi et al. (2012)) can be considered under this framework. In these problems, both parts of the input may appear several times in the training set, for example, the same customer may have rated several products and the same product may have been rated by several customers.

Let $D \subset \mathcal{D}$ and $T \subset \mathcal{T}$ denote, respectively, the in-sample data points and in-sample tasks, that is, the sets of data points and tasks encountered in the training set. Given a new input $\mathbf{x} = (\mathbf{d}, \mathbf{t})$, whose label is to be predicted with the model learned from the training set, the above type of learning problems can be coarsely divided into four different settings of varying difficulty, shown in the following table:

$\mathbf{d} \in D$ and $\mathbf{t} \in T$	$\mathbf{d} \in D$ and $\mathbf{t} \notin T$
$\mathbf{d} \notin D$ and $\mathbf{t} \in T$	$\mathbf{d} \notin D$ and $\mathbf{t} \notin T$

Of these, learning problems corresponding to the upper left setting are often encountered in missing value estimation and link prediction problems, where a partially filled matrix needs to be completed without the need for considering new rows and columns, as in collaborative filtering. The upper right and lower left settings can be interpreted as typical multi-task or multi-label learning problems, where the tasks are fixed in advance and the aim is to learn to solve several tasks together, so that the performance in the individual learning tasks is improved compared to the approach in which the individual tasks would be learned in isolation. The lower right setting is usually the most challenging one. Neither the data point nor the task parts are in this case known during training. This paper focuses mainly on this setting, but the proposed algorithms can be straightforwardly applied for any of the above settings.

In this work we consider the setting where both the tasks and data points of interest have feature representations, possibly defined implicitly via a kernel function (Shawe-Taylor and Cristianini, 2004). Previously, learning methods based on the tensor product (of Kronecker product) kernel have been successfully applied in such settings in order to solve problems such as product recommendation (Basilico and Hofmann, 2004; Park and Chu, 2009), prediction of protein-protein interactions (Ben-Hur and Noble, 2005; Kashima et al., 2009).

The pair-input modes based on tensor product kernels can be trained very efficiently with singular value decomposition based approaches (see e.g. Martin and Van Loan (2006); Raymond and Kashima (2010); Pahikkala et al. (2010a, 2013)), if the training set is complete in the sense that it contains every possible datum-task pair with data point in D and task in T exactly once. However, if the training set is not complete, no computationally efficient closed form solutions are known, and one must resort to iterative optimization approaches, such as those based on the conjugate gradient (CG) method.

There has been several articles about accelerating the gradient computation used in these methods, all of which are based on the so-called "vec-trick", which avoids the expensive computation of the tensor product (see e.g. Kashima et al. (2009)). In this paper, we show that the gradient computation can be further accelerated by taking advantage of the sparsity of the training data, that is, only a small subset of the datum-task pairs in $D \times T$ having a known label in training time. This can not be achieved with the standard algorithms and data structures used to implement sparse matrices and computations with them, but we propose new algorithms specially tailored for solving the problem in question.

2 Training Algorithms for Pair-Input Problems

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 vector $\mathbf{y} \in \mathbb{R}^n$ of the real-valued labels of the inputs. 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 $q = |T|$. We further define

$$\gamma : [n] \to [m] \times [q],$$

where the square bracket notation denotes the set $[n] = \{1, \ldots, n\}$, to be the function that maps the indices of the labeled inputs pairs to the index pairs corresponding to the data point and task the data consist of, that is, $\gamma(h) = (i, j)$ if $\mathbf{x}_h = (\mathbf{d}_i, \mathbf{t}_j)$. Note that γ does not necessarily have an inverse, since in some learning settings the training set may contain several data points with the same datum-task pair.

Next, unless stated otherwise, we assume that 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. 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. Then, the joint tensor feature representation for the training data (used in several studies in the machine learning literature as discussed in Section 1) can be expressed as $\mathbf{X} = \mathbf{B}(\mathbf{T} \otimes \mathbf{D})$, where $\otimes$ denotes the tensor (or Kronecker) product of matrices and $\mathbf{B} \in \{0, 1\}^{n \times mq}$ is a bookkeeping matrix, whose rows are indexed by the n training points and columns by the mq different tensor feature vector combinations, that is, the entries of $\mathbf{B}$ are

$$\mathbf{B}_{h,k} = \begin{cases} 1 & \text{if } k = (j-1)d + i, \text{ where } (i, j) = \gamma(h) \\ 0 & \text{otherwise} \end{cases}.$$

Each row of $\mathbf{B}$ contains a single nonzero entry indicating to which training input the datum corresponds. This matrix covers both the situation in which some of

the possible paired inputs are not in the training data and the one in which there are several occurrences of the same pair. We note that there are several alternative approaches for constructing a joint feature representation for pair-input data but the tensor-based representation is the most expressive one and it enables the simultaneous generalization to both out-of-sample data points and tasks (for further analysis about the expressivity and universality of the tensor-based representation, we refer to our previous work in Waegeman et al. (2012)).

The objective function of the ridge regression problem (Hoerl and Kennard, 1970) can be expressed as

$$J(\mathbf{w}) = (\mathbf{X}\mathbf{w} - \mathbf{y})^{\mathrm{T}}(\mathbf{X}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}^{\mathrm{T}}\mathbf{w}, \tag{1}$$

where $\lambda > 0$ is a regularization parameter controlling the trade-off between the regression error made on the training set and the complexity of the model represented by the real-valued vector $\mathbf{w}$. The minimizer of J can be found by solving the following system of linear equations:

$$\left(\mathbf{X}^{\mathrm{T}}\mathbf{X} + \lambda \mathbf{I}\right)\mathbf{w} = \mathbf{X}^{\mathrm{T}}\mathbf{y}$$

with respect to $\mathbf{w}$. By substituting the tensor feature representations for $\mathbf{X}$, the system becomes

$$\left((\mathbf{T} \otimes \mathbf{D})^{\mathrm{T}}\mathbf{B}^{\mathrm{T}}\mathbf{B}(\mathbf{T} \otimes \mathbf{D}) + \lambda \mathbf{I}\right)\mathbf{w} = (\mathbf{T} \otimes \mathbf{D})^{\mathrm{T}}\mathbf{B}^{\mathrm{T}}\mathbf{y}. \tag{2}$$

One can also introduce the corresponding optimization problem known as the dual problem of (1), whose solution is obtained via solving the following system

$$\left(\mathbf{X}\mathbf{X}^{\mathrm{T}} + \lambda \mathbf{I}\right)\mathbf{a} = \mathbf{y} \tag{3}$$

with respect to the dual variables $\mathbf{a}$. According to the KKT conditions, the primal and dual solutions are connected as $\mathbf{w} = \mathbf{X}^{\mathrm{T}}\mathbf{a}$. In addition to having computational advantages under certain circumstances more elaborated below, the dual problem also makes it possible to use nonlinear kernel functions in place of the ordinary inner products between feature vectors (Shawe-Taylor and Cristianini, 2004). Note also that, when using kernels, the input space (e.g. here consisting the Cartesian product of the data point space and task space) does not have to be a finite dimensional vector space but, depending of the kernel function used, any kind of set of inputs will do. By introducing the kernel matrices $\mathbf{K} = \mathbf{D}\mathbf{D}^{\mathrm{T}}$ and $\mathbf{G} = \mathbf{T}\mathbf{T}^{\mathrm{T}}$ for the data points and tasks, respectively, (3) can be rewritten as

$$\left(\mathbf{B}(\mathbf{G} \otimes \mathbf{K})\mathbf{B}^{\mathrm{T}} + \lambda \mathbf{I}\right)\mathbf{a} = \mathbf{y}. \tag{4}$$

If one solves the primal system (2) with, for example, the conjugate gradient (CG) algorithm (see e.g. Nocedal and Wright (2000)), it is easy to conclude that the computationally most expensive operations are the following two types of matrix-vector products:

$$\mathbf{u} \leftarrow \mathbf{B}(\mathbf{T} \otimes \mathbf{D})\mathbf{v} \tag{5}$$

$$\mathbf{v} \leftarrow (\mathbf{T} \otimes \mathbf{D})^{\mathrm{T}}\mathbf{B}^{\mathrm{T}}\mathbf{u} \tag{6}$$

where $\mathbf{v} \in \mathbb{R}^{dr}$ and $\mathbf{u} \in \mathbb{R}^n$. Similarly, the computationally most expensive operation involved in a CG step for solving the dual system (4) is the following matrix-vector product:

$$\mathbf{u} \leftarrow \mathbf{B}(\mathbf{G} \otimes \mathbf{K})\mathbf{B}^{\mathrm{T}}\mathbf{u} \tag{7}$$

where $\mathbf{u} \in \mathbb{R}^{mq}$.

The machine learning literature consists of several studies in which these products have been accelerated with the so-called "vec-trick", which is characterized by the following well-known results of the tensor product algebra:

Lemma 1. *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,*

$$(\mathbf{R}^{\mathrm{T}} \otimes \mathbf{P})\mathrm{vec}(\mathbf{Q}) = \mathrm{vec}(\mathbf{PQR}), \tag{8}$$

where vec *is the vectorization operator that stacks the columns of a matrix to a vector.*

It is obvious that the right hand size of (8) is considerably faster to compute than the left hand side, because it avoids the direct computation of the large tensor product.

Algorithm 1 Compute $\mathbf{u} \leftarrow \mathbf{B}(\mathbf{T} \otimes \mathbf{D})\mathbf{v}$

$\mathbf{u} \leftarrow \mathbf{0} \in \mathbb{R}^n$
if $mdr + rn < qdr + dn$ **then**
    $\mathbf{M} \leftarrow \mathbf{DV}$ ▷ $O(mdr)$ time operation
    for $h = 1, \ldots, n$ **do**
        $i, j \leftarrow \gamma(h)$
        $\mathbf{u}_h \leftarrow \mathbf{M}_i\mathbf{T}_{:,j}$ ▷ $O(r)$ time operation
else
    $\mathbf{N} \leftarrow \mathbf{VT}^{\mathrm{T}}$ ▷ $O(qdr)$ time operation
    for $h = 1, \ldots, n$ **do**
        $i, j \leftarrow \gamma(h)$
        $\mathbf{u}_h \leftarrow \mathbf{D}_i\mathbf{N}_{:,j}$ ▷ $O(d)$ time operation
return u

Let us consider both (5) and (6) in detail. Let $\mathbf{V} \in \mathbb{R}^{d \times r}$ be the matrix for which $\mathbf{v} = \mathrm{vec}(\mathbf{V})$ and $\mathbf{U} \in \mathbb{R}^{m \times q}$ be the matrix for which $\mathbf{B}^{\mathrm{T}}\mathbf{u} = \mathrm{vec}(\mathbf{U})$. Then, applying (8) leads to

$$\mathbf{u} \leftarrow \mathbf{B}\mathrm{vec}(\mathbf{DVT}^{\mathrm{T}}) \tag{9}$$

$$\mathbf{v} \leftarrow \mathrm{vec}(\mathbf{D}^{\mathrm{T}}\mathbf{UT}), \text{ with } \mathbf{B}^{\mathrm{T}}\mathbf{u} = \mathrm{vec}(\mathbf{U}). \tag{10}$$

Similarly, using the vec-trick on (7) transforms it to

$$\mathbf{u} \leftarrow \mathbf{B}\mathrm{vec}(\mathbf{KUG}), \text{ with } \mathbf{B}^{\mathrm{T}}\mathbf{u} = \mathrm{vec}(\mathbf{U}). \tag{11}$$

Multiplying a vector with the matrix $\mathbf{B}$ does not increase the complexity, because it contains at most mq nonzero entries, and hence it can be performed with the standard data structures and algorithms for sparse matrix-vector products. Thus, if we restrict our consideration on only the products between the other matrices, the complexity of the vec-trick method without taking advantage of the sparsity of the label information is characterized by the following lemma:

Lemma 2. *With the vec-trick, the computational complexity of a single gradient step for solving the primal form (e.g. the computation of the right hand sides of both (9) and (10)) is*

$$O(\min(mdr + mrq, drq + mdq)),$$

and the corresponding complexity for solving the dual form (e.g. the computation of the right hand side of (11)) is

$$O(m^2q + mq^2).$$

Proof. The complexity results directly from performing the matrix multiplications in the optimal order. □

Solving the primal problem is more cost-effective than solving the dual when the number of features is smaller than the number data points. For pair input data and tensor features, this is the case especially when both $d << m$ and $r << q$ hold simultaneously. In the opposite case, or if nonlinear kernel functions are used, it pays to solve the dual form instead.[1]

Next, we consider how the sparsity of the label information can be taken advantage of to further accelerate the gradient computations for both the primal and dual cases. With sparsity, we refer to the property that only a small portion of the datum-task pairs with the data point and task parts encountered in the training set has a known label. Formally, this means that $n << mq$.

Proposition 1. *The right hand sides of (9) and (10) can be computed in*

$$O(\min(mdr + rn, drq + dn))$$

time. The complexity for computing the right hand side of (11) is

$$O(mn + qn).$$

Proof. Calculating the right hand side of (9) can be started by first computing either $\mathbf{DV}$ or $\mathbf{VT}^{\mathrm{T}}$ requiring $O(mdr)$ and $O(qdr)$ time, respectively. Assume that we start with the former, and compute the matrix $\mathbf{M} \leftarrow \mathbf{DV}$. Then, each entry of $\mathbf{u}$ can be computed by taking the inner product between a row of $\mathbf{M}$

[1] The convergence properties of gradient descent methods (e.g. the number of steps required for achieving good prediction performance) may differ considerably between the primal and dual forms (Chapelle, 2007), but we leave this consideration out from this article, since it would divert the discussion too far from the scope of the paper.

Algorithm 2 Compute $\mathbf{v} \leftarrow (\mathbf{T} \otimes \mathbf{D})^{\mathrm{T}}\mathbf{B}^{\mathrm{T}}\mathbf{u}$

if $mdr + rn < qdr + dn$ **then**
  $\mathbf{M} \leftarrow \mathbf{0} \in \mathbb{R}^{m \times r}$
  for $h = 1, \ldots, n$ **do**
    $i, j \leftarrow \gamma(h)$
    $\mathbf{M}_i \leftarrow \mathbf{M}_i + \mathbf{u}_h \mathbf{T}_j$ ▷ $O(r)$ time operation
  $\mathbf{v} \leftarrow \mathrm{vec}(\mathbf{D}^{\mathrm{T}}\mathbf{M})$ ▷ $O(mdr)$ time operation
else
  $\mathbf{N} \leftarrow \mathbf{0} \in \mathbb{R}^{d \times q}$
  for $h = 1, \ldots, n$ **do**
    $i, j \leftarrow \gamma(h)$
    $\mathbf{N}_{:,j} \leftarrow \mathbf{N}_{:,j} + (\mathbf{D}^{\mathrm{T}})_{:,i}\mathbf{u}_h$ ▷ $O(r)$ time operation
  $\mathbf{v} \leftarrow \mathrm{vec}(\mathbf{NT})$ ▷ $O(qdr)$ time operation
return v

and a column of $\mathbf{T}^{\mathrm{T}}$, which are of length r. Since $\mathbf{u}$ has n entries, the overall complexity becomes $O(mdr + rn)$. If the computation is started with the latter way, each entry of $\mathbf{u}$ then requires the inner product between vectors of length d, resulting to an overall complexity $O(qdr + dn)$. This idea is summarized in Algorithm 1.

The matrix $\mathbf{U}$ in (10) contains at most n nonzero entries, and hence computing either the matrix product $\mathbf{D}^{\mathrm{T}}\mathbf{U}$ or $\mathbf{UT}$ require $O(dn)$ and $O(rn)$ time, respectively. The subsequent multiplications of either with $\mathbf{T}$ from right or with $\mathbf{D}^{\mathrm{T}}$ from left, increase the overall complexities to $O(drq + dn)$ or $O(mdr + rn)$ time, respectively. This is illustrated in Algorithm 2

Algorithm 3 Compute $\mathbf{u} \leftarrow \mathbf{B}(\mathbf{G} \otimes \mathbf{K})\mathbf{B}^{\mathrm{T}}\mathbf{u}$

$\mathbf{M} \leftarrow \mathbf{0} \in \mathbb{R}^{m \times q}$
for $h = 1, \ldots, n$ **do**
  $i, j \leftarrow \gamma(h)$
  $\mathbf{M}_i \leftarrow \mathbf{M}_i + \mathbf{u}_h(\mathbf{G})_j$ ▷ $O(q)$ time operation
$\mathbf{u} \leftarrow \mathbf{0} \in \mathbb{R}^{n}$
for $h = 1, \ldots, n$ **do**
  $i, j \leftarrow \gamma(h)$
  $\mathbf{u}_h \leftarrow \mathbf{K}_i\mathbf{M}_{:,j}$ ▷ $O(m)$ time operation
return u

The matrix $\mathbf{U}$ in (11) has at most n nonzero entries, and hence multiplying it with $\mathbf{K}$ from left or with $\mathbf{G}$ from right take $O(mn)$ and $O(qn)$ time, respectively. The subsequent filling of the entries of $\mathbf{u}$ require n inner products between vectors of size q or m depending whether $\mathbf{U}$ was multiplied with $\mathbf{K}$ or $\mathbf{G}$, resulting in an overall time complexity of $O(mn + qn)$. This is illustrated in Algorithm 3. □

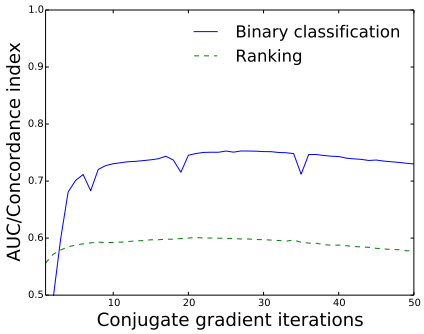

Fig. 1. Prediction performance as a function of CG iterations for both the ranking and classification tasks.

3 Experiments

In the experiments, we demonstrate the use of the algorithm on a practical problem of predicting drug-target (DT) interactions, and compare the computational speed of the proposed training algorithm based on the one that employs the vec-trick only. The data we use for DT interaction prediction experiments consists of 1421 drug compounds, 156 protein targets, and 93356 interaction binding affinity values for DT pairs originally measured by Metz et al. (2011). That is, a bit less than half of the possible DT pairs are labeled with a known binding value. The binding values vary between 4.0 and 10.3, the larger the values the tighter binding. The features of the drugs consists of their 2D Tanimoto coefficient similarities with the other drugs, that is, the feature matrix $\mathbf{D}$ is a symmetric 1421×1421-matrix. The feature representation for the protein targets is their normalized Waterman-Smith sequence similarity with the other targets, resulting to a symmetric 156×156-dimensional feature matrix $\mathbf{T}$. We refer to Pahikkala et al. (2014) for more in depth description of the data and the similarities.[2] The implementation of the algorithm will be put online as a part of the RLScore open source machine learning library.[3]

As practical example problems, we consider the task of learning to rank the DT pairs with respect to their binding value and a binary classification problem in which a drug and target are said to interact if the binding value is larger than 7.6. In both experiments, we perform nine train-test splits of the whole data over which the performance is averaged. The splits reflect the most challenging of the four settings considered in the introduction section, that is, the one in which the model must simultaneously generalize for new drugs and targets. The performance of both learning problems is measured using the concordance index (Gönen and Heller, 2005) (C-index), also known as the pairwise ranking accuracy $\frac{1}{|\{(i,j)|y_i>y_j\}|}\sum_{y_i>y_j} H(\hat{y}_i - \hat{y}_j)$ where y_i denote the true and $\hat{y}_i$ the predicted

[2] The data is avaliable at `http://staff.cs.utu.fi/~aatapa/data/DrugTarget/`
[3] Available at `https://github.com/aatapa/RLScore`

	Drug-target	Simulation
New method	57	0.17
Vec-trick method	67	11.43

Table 1. The time (in seconds) spent for gradient computations by the proposed accelerated method and the traditional vec-trick based approach.

labels, and H is the Heaviside step function. Note that this measure reduces to the area under ROC curve (AUC) in the binary classification problem. The prediction performances for the tasks are illustrated in Figure 1. The Tikhonov regularization parameter value is set to 0, and hence the only regularization mechanism is the number of CG iterations. We observe that in both tasks one requires only a few CG iterations until the performance converges, to a slightly better than random level (concordance index 0.6) for the ranking tasks but notable better classification performance (AUC 0.75). These results are in line with those published in our previous study with the data (Pahikkala et al., 2014).

We compare the running speeds of the new and the vec-trick based approach on both the DT interaction prediction problem and with a simulated experiment with randomly generated data. Table 1 presents the running time of both algorithms on 50 CG iterations for the DT problem. Since almost half of the possible DT pairs is known, the training labels are not really sparse and the difference between the running times is small. We next generated artificial data and task similarity matrices $\mathbf{D}, \mathbf{T} \in \mathbb{R}^{10000 \times 100}$ and generated a vector $\mathbf{y} \in \mathbb{R}^{10000}$ labels for inputs with random datum-task indices. For this experiment, the running time of the proposed algorithm for a single gradient iteration is almost two orders of magnitude smaller than that of the vec-trick method, demonstrating the potential of the new approach for large-scale and sparse data sets.

References

Basilico, J., Hofmann, T.: Unifying collaborative and content-based filtering. In: Brodley, C.E. (ed.) Proceedings of the twenty-first international conference on Machine learning (ICML'04). ACM International Conference Proceeding Series, vol. 69. ACM (2004)

Ben-Hur, A., Noble, W.: Kernel methods for predicting protein-protein interactions. Bioinformatics 21 Suppl 1, 38–46 (2005)

Bonilla, E.V., Agakov, F.V., Williams, C.K.I.: Kernel multi-task learning using task-specific features. In: Meila, M., Shen, X. (eds.) 11th International Conference on Artificial Intelligence and Statistics. JMLR Proceedings, vol. 2, pp. 43–50. JMLR.org (2007)

Chapelle, O.: Training a support vector machine in the primal. Neural Computation 19(5), 1155–1178 (May 2007)

Ding, H., Takigawa, I., Mamitsuka, H., Zhu, S.: Similarity-based machine learning methods for predicting drugtarget interactions: a brief review. Briefings in Bioinformatics (2013)

Gönen, M., Heller, G.: Concordance probability and discriminatory power in proportional hazards regression. Biometrika 92(4), 965–970 (2005)
Hayashi, K., Takenouchi, T., Tomioka, R., Kashima, H.: Self-measuring similarity for multi-task gaussian process. In: Guyon, I., Dror, G., Lemaire, V., Taylor, G.W., Silver, D.L. (eds.) ICML Unsupervised and Transfer Learning Workshop. JMLR Proceedings, vol. 27, pp. 145–154. JMLR.org (2012)
Hoerl, A.E., Kennard, R.W.: Ridge regression: Biased estimation for nonorthogonal problems. Technometrics 12, 55–67 (1970)
Kashima, H., Kato, T., Yamanishi, Y., Sugiyama, M., Tsuda, K.: Link propagation: A fast semi-supervised learning algorithm for link prediction. In: Proceedings of the SIAM International Conference on Data Mining (SDM 2009). pp. 1099–1110. SIAM (2009)
Liu, T.Y.: Learning to Rank for Information Retrieval. Springer (2011)
Martin, C.D., Van Loan, C.F.: Shifted Kronecker product systems. SIAM Journal on Matrix Analysis and Applications 29(1), 184–198 (2006)
Metz, J.T., Johnson, E.F., Soni, N.B., Merta, P.J., Kifle, L., Hajduk, P.J.: Navigating the kinome. Nature Chemical Biology 7(4), 200–202 (Apr 2011)
Nocedal, J., Wright, S.J.: Numerical Optimization. Springer, 1 edn. (2000)
Pahikkala, T., Airola, A., Pietilä, S., Shakyawar, S., Szwajda, A., Tang, J., Aittokallio, T.: Toward more realistic drug-target interaction predictions. Briefings in Bioinformatics (2014), In press. DOI: 10.1093/bib/bbu010
Pahikkala, T., Airola, A., Stock, M., Baets, B.D., Waegeman, W.: Efficient regularized least-squares algorithms for conditional ranking on relational data. Machine Learning 93(2-3), 321–356 (2013)
Pahikkala, T., Waegeman, W., Airola, A., Salakoski, T., De Baets, B.: Conditional ranking on relational data. In: Balcázar, J.L., Bonchi, F., Gionis, A., Sebag, M. (eds.) Machine Learning and Knowledge Discovery in Databases (ECML PKDD 2010). Lecture Notes in Computer Science, vol. 6322, pp. 499–514. Springer (2010a)
Pahikkala, T., Waegeman, W., Tsivtsivadze, E., Salakoski, T., De Baets, B.: Learning intransitive reciprocal relations with kernel methods. European Journal of Operational Research 206(3), 676–685 (2010b)
Park, S.T., Chu, W.: Pairwise preference regression for cold-start recommendation. In: Proceedings of the Third ACM Conference on Recommender Systems. pp. 21–28. ACM, New York, NY, USA (2009)
Raymond, R., Kashima, H.: Fast and scalable algorithms for semi-supervised link prediction on static and dynamic graphs. In: Balcázar, J.L., Bonchi, F., Gionis, A., Sebag, M. (eds.) Machine learning and knowledge discovery in databases (ECML PKDD 2010), Lecture Notes in Computer Science, vol. 6323, pp. 131–147. Springer (2010)
Shawe-Taylor, J., Cristianini, N.: Kernel Methods for Pattern Analysis. Cambridge University Press, Cambridge (2004)
Waegeman, W., Pahikkala, T., Airola, A., Salakoski, T., Stock, M., De Baets, B.: A kernel-based framework for learning graded relations from data. IEEE Transactions on Fuzzy Systems 20(6), 1090–1101 (December 2012)