BCDNPKL : Scalable Non-Parametric Kernel Learning Using Block Coordinate Descent

En-Liang Hu[†] HUMATH@YNNU.EDU.CN
Bo Wang[‡] BOWANG@NUAA.EDU.CN
Songcan Chen[‡] S.CHEN@NUAA.EDU.CN
[†]Department of Mathematics, Yunnan Normal University, Kunming, Yunnan, 650092, China
[‡]Department of Computer Science & Engineering, Nanjing University of Aeronautics & Astronautics, China

Abstract

Most existing approaches for non-parametric kernel learning (NPKL) suffer from expensive computation, which would limit their applications to large-scale problems. To address the scalability problem of NPKL, we propose a novel algorithm called BCDNPKL, which is very efficient and scalable. Superior to most existing approaches, BCDNPKL keeps away from semidefinite programming (SDP) and eigen-decomposition, which benefits from two findings: 1) The original SDP framework of NPKL can be reduced into a far smaller-sized counterpart which is corresponding to the sub-kernel (referred to as boundary kernel) learning; 2) The sub-kernel learning can be efficiently solved by using the proposed block coordinate descent (BCD) technique. We provide a formal proof of global convergence for our BCDNPKL algorithm. The extensive experiments verify the scalability and effectiveness of BCDNPKL, compared with the state-of-the-art algorithms.

1. Introduction

Kernel methods have attracted more and more attentions of researchers in computer science and engineering due to their superiority in classification, clustering, dimensionality reduction and so on. However, manually choosing an appropriate kernel requires specific domain knowledge, which limits the application of kernel methods in some situations. Even for a given kernel, how to tune the kernel parameters is also difficult. Therefore, it has become a more and more important research issue for how to automatically identify the appropriate kernel which is consistent with the data characteristics. Recently, a large amount of kernel learning algorithms (Chapelle et al., 2002; Smola & Kondor, 2003; Lanckriet et al., 2004; Zhu et al., 2004; Kulis et al., 2006; Hoi et al., 2007; Zhuang et al., 2009; Hu et al., 2010) have been proposed for learning the kernel from side-information (i.e., incomplete prior knowledge). There are two kinds of representative side information: class labels, and pairwise constraints in which a must-link constraint indicates two objects should belong to the same class while a cannot-link for different classes. In this work, we focus on learning the kernel matrix from pairwise constraints.

Despite of many successes, most existing kernel matrix learning algorithms need expensive computation, which limits their wide applications in large-scale problems. For example, a family of studies (Lanckriet et al., 2004; Zhu et al., 2004; Kulis et al., 2006; Hoi et al., 2007; Zhuang et al., 2009; Hu et al., 2010), referred to as non-parametric kernel learning (NPKL), are devoted to learning the entire kernel matrix, which generally leads to a SDP optimization problem. However, the time complexity of standard SDP solvers based on the interior-point method could be as high as $\mathcal{O}(n^{6.5})$ (Zhuang et al., 2009). To address the scalability issue of NPKL, an algorithm called NPK was proposed by Hoi et al. (Hoi et al., 2007), which reduces the cost of SDP optimization by using dualization and a heuristic scheme. More recently, Zhuang et al. (Zhuang et al., 2009) introduced an algorithm called SimpleNPKL to further improve the scalability of NPKL, which first converts the SDP framework into a min-max problem and then solves this problem by using an alternative iteration strategy. However, despite escaping from SDP optimization, SimpleNPKL still performs one eigen-decomposition in its iteration,

Appearing in *Proceedings of the 28th International Conference on Machine Learning*, Bellevue, WA, USA, 2011.

which greatly limits its scalability.

In this paper, we focus on addressing the scalability of NPKL problem in (Hoi et al., 2007; Zhuang et al., 2009). First, we find that the original SDP optimization of NPKL can be reduced into a far smaller-sized counterpart, implying that we can obtain a same solution only with a significantly lower computation. This finding is inspired by the boundary problem (Jost, 2005), stating that a harmonic mapping can be uniquely determined by the boundary value condition. In like manner, if regarding the data points with or without pairwise constraints as the boundary or interior nodes of an underlying graph G respectively, an implicit harmonic kernel mapping ϕ_G can be uniquely determined by its subpart only restricted to the boundary nodes. Following such inspiration, we formally prove that a desired kernel matrix Z can be uniquely determined by its subpart Z^{BD} (referred to as boundary kernel) over the constrained points.

For further speedup, we explore a BCD technique instead of SDP optimization to solve the reduced problem (i.e., boundary kernel). As is well known, BCD is a classical optimization technique that has witnessed a resurgence of interest in machine learning (Hsieh et al., 2008), reasons for which include its simplicity, efficiency and stability if each minimization of subproblem can be performed very efficiently. Specifically, we first reformulate the SDP problem w.r.t. Z^{BD} into a nonlinear programming (NLP) by performing a low-rank factorization, then solve this NLP reformulation by using a BCD technique in Gauss-Seidel style (Grippo & Sciandrone, 2000). An interesting observation is that if we use the hinge loss, square hinge loss or square loss function, the corresponding subproblem will be a strictly convex quadratic programming (QP), which has a closed-form solution for square loss. To summarize, our main contributions are as follows:

1) We formally prove that the original SDP framework of NPKL can be reduced into a far smaller-sized counterpart which is only restricted to the constrained points. This implies that we can obtain a same solution with a significantly lower computation.

2) For further speedup, we exploit a BCD technique rather than SDP optimization for boundary kernel learning.

3) We extend NPKL to the square loss and thus get a low-computational closed-form solution for the subproblem of boundary kernel learning, which contributes to more than 80 times speedup over SimpleNPKL algorithm.

The rest of this paper is organized as follows. Section 2 reviews the basics of NPKL briefly. Section 3 first reduces the original SDP framework of NPKL into the boundary kernel learning and then introduces a BCD technique for solving it. Further, the efficient implementation and global convergence of the proposed algorithm are also discussed. Section 4 shows our experimental results and Section 5 concludes this work.

2. Review for Non-Parametric Kernel Learning

For completeness, we briefly review the previous work (Hoi et al., 2007; Zhuang et al., 2009). Denoting the entire data point collection by $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n\}$, $S = (S_{i,j}) \in \mathbb{R}^{n\times n}$ is a symmetric matrix in which each $S_{i,j}$ represents the similarity between $\mathbf{x}_i$ and $\mathbf{x}_j$. Thus, S implies a graph G, in which each data point $\mathbf{x}_i \in \mathcal{X}$ corresponds to a graph node of G and each $S_{i,j}$ acts as an edge weight between the nodes in terms of $\mathbf{x}_i$ and $\mathbf{x}_j$. Thus, a normalized graph Laplacian Δ of G can be constructed as follows:

$$\Delta = (1+\delta)I - D^{-\frac{1}{2}} S D^{-\frac{1}{2}}$$

where I is the identity matrix of proper size and $D = \mathrm{diag}(d_1, d_2, \cdots, d_n)$ is a diagonal (degree) matrix with $d_i = \sum_j S_{i,j}$. A parameter $\delta > 0$ is introduced to prevent Δ from being singular, such that $\Delta \succ 0$. Let $\mathcal{T}$ be the indices of the given constraints, an indicator matrix T can be constructed for representing $\mathcal{T}$ as follows

$$T_{i,j} = \begin{cases} +1 & \text{if } (\mathbf{x}_i, \mathbf{x}_j) \text{ is a must-link pair in } \mathcal{T} \\ -1 & \text{if } (\mathbf{x}_i, \mathbf{x}_j) \text{ is a cannot-link pair in } \mathcal{T} \end{cases}$$

The goal of NPKL is to identify a kernel matrix Z that is consistent with both Δ and T. Following (Hoi et al., 2007), we formulate it into the following SDP problem:

$$\min_{Z\succeq 0} : tr(\Delta Z) + C \sum_{(i,j)\in\mathcal{T}} \ell(T_{i,j} Z_{i,j}) \qquad (1)$$

where the first term plays a similar role as the manifold regularization, in which Δ is used to impose smoothness. The second term measures the inconsistency between Z and the given constraints, and $\ell(\cdot)$ is specified as hinge loss function in (Hoi et al., 2007) while it can be linear or square hinge loss in (Zhuang et al., 2009).

3. Non-Parametric Kernel Learning Using BCD

In the sequel, let L and U be the indices of the data points involved with and without pairwise constraints respectively, i.e., the data points in $\mathcal{X}_L = \{\mathbf{x}_i | (\mathbf{x}_i, \cdot) \in$

$\mathcal{T}\}$ are constrained points while unconstrained points for the ones in $\mathcal{X}_U = \mathcal{X} \setminus \mathcal{X}_L$, we denote $|\mathcal{X}_L| = l$. Thus, based on L and U, Z and Δ can be represented as follows

$$Z = \begin{pmatrix} Z_{L,L} & Z_{L,U} \\ Z_{U,L} & Z_{U,U} \end{pmatrix} \quad \Delta = \begin{pmatrix} \Delta_{L,L} & \Delta_{L,U} \\ \Delta_{U,L} & \Delta_{U,U} \end{pmatrix} \tag{2}$$

3.1. Derivation to Boundary Kernel Learning

Following (Hoi et al., 2007), learning a kernel Z corresponds to seek an embedding mapping ϕ_G from the nodes of underlying graph G to a RKHS, i.e., $\phi_G : \mathbf{x}_i \longmapsto \phi_G(\mathbf{x}_i)$, that is, learning Z is equivalent to seek ϕ_G. Let $\Phi = \phi_G(\mathcal{X}) = (\phi_G(\mathbf{x}_1), \phi_G(\mathbf{x}_2), \cdots, \phi_G(\mathbf{x}_n))$ and $\Phi_L = \Phi|_L$ be as restricting Φ to $\mathcal{X}_L$, then

$$Z = \Phi^\top \Phi \quad and \quad Z_{L,L} = \Phi_L^\top \Phi_L \tag{3}$$

If viewing the nodes associated with $\mathcal{X}_L$ and $\mathcal{X}_U$ as the boundary and interior of G respectively, ϕ_G is a mapping restricted to G. Inspired by the boundary value problem (Jost, 2005), we can give the following Proposition 1 stating that ϕ_G can be uniquely determined by its subpart only restricted to the boundary.

Proposition 1: The optimal solution of Eq.1 is $Z = QZ^{BD}Q^\top$, in which $Z^{BD} = Z_{L,L}$ is an optimal solution of the following SDP problem:

$$\min_{Z^{BD} \succeq 0} : tr(\tilde{\Delta} Z^{BD}) + C \sum_{(i,j)\in\mathcal{T}} \ell(T_{i,j} Z^{BD}_{i,j}) \tag{4}$$

where we denote I_L the identity matrix of proper size, $Q = \begin{pmatrix} I_L \\ -\Delta_{U,U}^{-1}\Delta_{L,U}^\top \end{pmatrix}$ and $\tilde{\Delta} = \Delta_{L,L} - \Delta_{L,U}\Delta_{U,U}^{-1}\Delta_{L,U}^\top$.

Proof: Let $\Omega(Z) = tr(\Delta Z) + C \sum_{(i,j)\in\mathcal{T}} \ell(T_{i,j} Z_{i,j})$, by substituting Eq.s 2∼3, we have

$$\begin{aligned} \Omega(Z) \quad &= tr(\Phi_L \Delta_{L,L} \Phi_L^\top + 2\Phi_U \Delta_{L,U}^\top \Phi_L^\top \\ &\quad + \Phi_U \Delta_{U,U} \Phi_U^\top) + C \sum_{(i,j)\in\mathcal{T}} \ell(T_{i,j}(\Phi_L^\top \Phi_L)_{i,j}) \end{aligned}$$

By setting $\frac{1}{2}\frac{\partial \Omega}{\partial \Phi_U} = \Delta_{L,U}^\top \Phi_L^\top + \Delta_{U,U}\Phi_U^\top = 0$, we have

$$\Phi_U^\top = -\Delta_{U,U}^{-1}\Delta_{L,U}^\top \Phi_L^\top$$

Thus, $\Omega(Z) = tr(\tilde{\Delta} Z^{BD}) + C \sum_{(i,j)\in\mathcal{T}} \ell(T_{i,j} Z^{BD}_{i,j})$ and

$$Z = Q\Phi_L^\top \Phi_L Q^\top = QZ^{BD}Q^\top, \ \Phi^\top = Q\Phi_L^\top. \ \blacksquare$$

Remark: Proposition 1 reveals a nice property that a full-kernel matrix Z can be uniquely determined by its subpart counterpart $Z_{L,L}$ only restricted to the pairwise constraints. This property holds for any loss function, and will significantly simplify the computation of Eq.1 when $l \ll n$.

3.2. Boundary Kernel Learning Using BCD

Proposition 1 reduces learning Z into learning its subpart $Z^{BD} = Z_{L,L}$. However if learning Z^{BD} by using SDP optimization, the time complexity could be as high as $\mathcal{O}(|\mathcal{X}_L|^{6.5})$(Zhuang et al., 2009). Therefore, we resort to explore a BCD technique for solving the SDP problem of Eq.4. That is, we first reformulate Eq.4 into a NLP problem by low-rank factorization, and then solve this NLP by BCD iteration in Gauss-Seidel style (Grippo & Sciandrone, 2000).

3.2.1. LOW-RANK FACTORIZATION

Due to its symmetry and positive semidefiniteness, Z^{BD} can be factorized as

$$Z^{BD} = V^\top V, \quad V = (\mathbf{v}_1, \cdots, \mathbf{v}_l)$$

where $\mathbf{v}_i \in \mathbb{R}^r$ can be viewed as the new data representation of $\mathbf{x}_i \in \mathcal{X}_L$. Substituting $Z^{BD} = V^\top V$ into Eq.4, we get a NLP formulation as

$$\min_V : tr(V\tilde{\Delta}V^\top) + C \sum_{(i,j)\in\mathcal{T}} \ell(T_{i,j}\mathbf{v}_i^\top \mathbf{v}_j) \tag{5}$$

Now, the rising problem is what relationship exists between the solutions of Eq.4 and Eq.5. A possible answer is given by the following Proposition 2.

Proposition 2 (Burer & Monteiro, 2003): For sufficiently large values of r, a global solution of Eq.5 gives a global solution of Eq.4.

Proposition 2 states that a value of r exists such that there is a one-to-one correspondence between the global solutions of Eq.5 and Eq.4. Further, for the estimation of r, Theorem 2.2 in (Burer & Monteiro, 2003) states that for a SDP problem with only linear constraints, the rank of its optimal solution satisfies $r(r+1) \leq m$, where m is the number of linear constrains. Thus, we have Proposition 3.

Proposition 3: If optimal solution $Z^{BD^\star}$ exists for Eq.4, then the rank r of $Z^{BD^\star}$ satisfies the inequality $r(r+1)/2 \leq m$ where $m = |\mathcal{T}|$.

Proposition 2 implies that optimizing Eq.5 can get a global solution of Eq.4 for all $r \geq \max\{s \in \mathbb{N} | s(s+1) \leq m\}$. However, the problem in Eq.5 is non-convex so that searching its global optimal solutions is difficult. So, we aim at finding a local optimal solution of Eq.5 by using the BCD technique (Grippo & Sciandrone, 2000) in the next subsection.

3.2.2. BCD FORMULATION FOR DIFFERENT LOSS FUNCTIONS

Let Ω be the objective of Eq.5 again and taking each $\mathbf{v}_i (i = 1, \cdots, l)$ as one coordinate block, the subprob-

lem for updating $\mathbf{v}_i$ becomes

$$\mathbf{v}_i^{(t+1)} = \underset{\mathbf{y}_i}{\operatorname{argmin}}\ \Omega(\mathbf{v}_1^{(t+1)}, \cdots, \mathbf{v}_{i-1}^{(t+1)}, \mathbf{y}_i, \mathbf{v}_{i+1}^{(t)}, \cdots, \mathbf{v}_l^{(t)})$$

which updates V column by column. Starting from an initial matrix $V^{(0)}$, this update can generate a sequence of matrices $\{V^{(t)}\}_{t=0}^{\infty}$ and we refer to the process from $V^{(t)}$ to $V^{(t+1)}$ as an outer iteration. Also, we have l inner iterations so that $\mathbf{v}_1, \cdots, \mathbf{v}_l$ are updated within each outer iteration. The subproblem for updating $\mathbf{v}_i (i = 1, \cdots, l)$ can be detailed as

$$\min_{\mathbf{v}_i} : \frac{1}{2}\mathbf{v}_i^\top(\tilde{\Delta}_{i,i}\mathbf{v}_i + 2(\sum_{k\neq i}\tilde{\Delta}_{i,k}\mathbf{v}_k)) + \frac{C}{2}\sum_{j\in\mathcal{T}(i)}\ell(T_{i,j}\mathbf{v}_i^\top\mathbf{v}_j) \quad (6)$$

where $\mathcal{T}(i) = \{j|(i,j) \in \mathcal{T}\}$ and all $\mathbf{v}_k (k\neq i)$ should be fixed at the current inner iteration. Next, we will show how to get a compact convex QP when the hinge loss (HL), square hinge loss (SHL) or square loss (SL) is plugged into Eq.6 respectively.

(I) **Hinge Loss**: $\ell(f) = 2\max(1-f, 0)$

The problem in Eq.6 can be reformulated as

$$\begin{array}{l}\min_{\mathbf{v}_i,\epsilon_j} : \frac{1}{2}\tilde{\Delta}_{i,i}\mathbf{v}_i^\top\mathbf{v}_i + \mathbf{v}_i^\top(\sum_{k\neq i}\tilde{\Delta}_{i,k}\mathbf{v}_k) + C\sum_{j\in\mathcal{T}(i)}\epsilon_j \\ s.t.\ : \forall j \in \mathcal{T}(i),\ T_{i,j}\mathbf{v}_i^\top\mathbf{v}_j \geq 1-\epsilon_j,\ \epsilon_j \geq 0.\end{array} \quad (7)$$

By introducing the dual vectors $\boldsymbol{\alpha} = (\alpha_1, \cdots, \alpha_{|\mathcal{T}(i)|})^\top$ and $\boldsymbol{\beta} \in \mathbb{R}^{|\mathcal{T}(i)|}$, we have a Lagrangian function:

$$\begin{aligned}\mathcal{L}(\mathbf{v}_i, \epsilon_j; \alpha_{\pi(j)}, \beta_{\pi(j)}) &= \tfrac{1}{2}\tilde{\Delta}_{i,i}\mathbf{v}_i^\top\mathbf{v}_i + \mathbf{v}_i^\top(\sum_{k\neq i}\tilde{\Delta}_{i,k}\mathbf{v}_k) \\ + \sum_{j\in\mathcal{T}(i)}(C-\beta_{\pi(j)})\epsilon_j &- \sum_{j\in\mathcal{T}(i)}\alpha_{\pi(j)}(T_{i,j}\mathbf{v}_i^\top\mathbf{v}_j - 1 + \epsilon_j)\end{aligned}$$

where $\pi : j \in \mathcal{T}(i) \longmapsto \pi(j) \in \{1, \cdots, |\mathcal{T}(i)|\}$ is a permutation. By vanishing the first order derivative of $\mathcal{L}$, we have

$$\begin{cases}\frac{\partial\mathcal{L}}{\partial\epsilon_j} = C - \alpha_{\pi(j)} - \beta_{\pi(j)} = 0 \\ \frac{\partial\mathcal{L}}{\partial\mathbf{v}_i} = \tilde{\Delta}_{i,i}\mathbf{v}_i + \sum_{k\neq i}\tilde{\Delta}_{i,k}\mathbf{v}_k - \sum_{j\in\mathcal{T}(i)}\alpha_{\pi(j)}T_{i,j}\mathbf{v}_j = 0\end{cases}$$

This means $0 \leq \alpha_{\pi(j)} \leq C$ and

$$\mathbf{v}_i = \tilde{\Delta}_{i,i}^{-1}(\sum_{j\in\mathcal{T}(i)}\alpha_{\pi(j)}T_{i,j}\mathbf{v}_j - \sum_{k\neq i}\tilde{\Delta}_{i,k}\mathbf{v}_k) \quad (8)$$

By substituting Eq.8 into $\mathcal{L}$, we have

$$\begin{aligned}\mathcal{L}(\boldsymbol{\alpha}) = &\sum_{s\in\mathcal{T}(i)}\alpha_\pi(s)(T_{i,s}\mathbf{v}_s^\top\sum_{k\neq i}(\tilde{\Delta}_{i,k}\mathbf{v}_k) + \tilde{\Delta}_{i,i}) \\ &-\tfrac{1}{2}\sum_{s,t\in\mathcal{T}(i)}T_{i,s}T_{i,t}\mathbf{v}_s^\top\mathbf{v}_t\alpha_{\pi(s)}\alpha_{\pi(t)}\end{aligned}$$

Let $q = (q_1, \cdots, q_{|\mathcal{T}(i)|})^\top$ and $P = (p_{\pi(s),\pi(t)})$ in terms of $q_{\pi(s)} = T_{i,s}\mathbf{v}_s^\top(\sum_{k\neq i}\tilde{\Delta}_{i,k}\mathbf{v}_k) + \tilde{\Delta}_{i,i}$ and $p_{\pi(s),\pi(t)} = T_{i,s}T_{i,t}\mathbf{v}_s^\top\mathbf{v}_t$, the dual of Eq.7 can be induced as

$$\min_{0\leq\boldsymbol{\alpha}\leq C} : \frac{1}{2}\boldsymbol{\alpha}^\top P\boldsymbol{\alpha} - \boldsymbol{\alpha}^\top q \quad (9)$$

where $P \succeq 0$ and $0 \leq \boldsymbol{\alpha} \leq C$ means $0 \leq \alpha_{\pi(j)} \leq C$ for every $j \in \mathcal{T}(i)$.

(II) **Square Hinge Loss**: $\ell(f) = (\max(1-f, 0))^2$

The problem in Eq.6 can be reformulated as

$$\begin{array}{l}\min_{\mathbf{v}_i,\epsilon_j} : \frac{1}{2}\tilde{\Delta}_{i,i}\mathbf{v}_i^\top\mathbf{v}_i + \mathbf{v}_i^\top(\sum_{k\neq i}\tilde{\Delta}_{i,k}\mathbf{v}_k) + \frac{C}{2}\sum_{j\in\mathcal{T}(i)}{\epsilon_j}^2 \\ s.t.\ : \forall j \in \mathcal{T}(i),\ T_{i,j}\mathbf{v}_i^\top\mathbf{v}_j \geq 1-\epsilon_j.\end{array}$$

Let $H = P + \frac{1}{C}\tilde{\Delta}_{i,i}I \in \mathbb{R}^{|\mathcal{T}(i)|\times|\mathcal{T}(i)|}$, the dual becomes

$$\min_{\boldsymbol{\alpha}\geq 0} : \frac{1}{2}\boldsymbol{\alpha}^\top H\boldsymbol{\alpha} - \boldsymbol{\alpha}^\top q \quad (10)$$

where both P and q have the same forms as in Eq.9, and the primal solution is also similar to Eq.8.

(III) **Square Loss**: $\ell(f) = (1-f)^2$

The problem in Eq.6 can be reformulated as

$$\min_{\mathbf{v}_i} : \frac{1}{2}\mathbf{v}_i^\top M\mathbf{v}_i + \mathbf{v}_i^\top(\sum_{k\neq i}\tilde{\Delta}_{i,k}\mathbf{v}_k - C\sum_{j\in\mathcal{T}(i)}T_{i,j}\mathbf{v}_j)$$

where $M = \tilde{\Delta}_{i,i}I + C\sum_{j\in\mathcal{T}(i)}\mathbf{v}_j\mathbf{v}_j^\top$. Clearly, we can get its closed-form solution as

$$\mathbf{v}_i = -M^{-1}(\sum_{k\neq i}\tilde{\Delta}_{i,k}\mathbf{v}_k - C\sum_{j\in\mathcal{T}(i)}T_{i,j}\mathbf{v}_j) \quad (11)$$

3.3. Algorithm and Implementation Issues

Based on the above analyses, the complete BCDNPKL algorithm can be summarized in *Algorithm 1*.

Algorithm 1 BCDNPKL for NPKL

Input: $T, Q, \tilde{\Delta}, \delta, r, C, \varepsilon$ and $IterMax$
Output: Target kernel matrix $Z^\star$
Initialization: set $V^{(0)} \in \mathbb{R}^{r\times l}$ and $t = 0$.
repeat
  for $i = 1$ **to** l **do**
    (i) Update $\mathbf{v}_i^{(t)}$ into $\mathbf{v}_i^{(t+1)}$ by Eq.8 or Eq.11;
    (ii) $V^{(t)} = (\mathbf{v}_1^{(t+1)}, \cdots, \mathbf{v}_i^{(t+1)}, \mathbf{v}_{i+1}^{(t)}, \cdots, \mathbf{v}_l^{(t)})$.
  end for
  Set $V^{(t+1)} = (\mathbf{v}_1^{(t+1)}, \mathbf{v}_2^{(t+1)}, \cdots, \mathbf{v}_l^{(t+1)}), t = t+1$;
until $\{||V^{(t+1)} - V^{(t)}||_{\mathcal{F}} < \varepsilon$ or $t > IterMax\}$
Obtain $Z^\star = QZ^{BD}Q^\top$ w.r.t. $Z^{BD} = {V^{(t)}}^\top V^{(t)}$.

For implementing *Algorithm 1*, the time complexity mainly lies in solving the subproblem of Eq.9, 10 or

11, which is always a strictly convex QP (its size is $|\mathcal{T}(i)|$ to Eq.9 or 10). When $|\mathcal{T}(i)| = 1$, we can obtain a closed-form solution $\boldsymbol{\alpha} = \min(\max(P^{-1}q, 0), C)$ to Eq.9 and $\boldsymbol{\alpha} = \max(H^{-1}q, 0)$ to Eq.10. It is worth noting that the size of the QP of Eq.9 or Eq.10 is often very small since the given constraints are distributed very sparse in general. When SL is chosen, the subproblem has the closed-form solution in Eq.11, in which computing the matrix inverse can become very fast by using a generalization of Sherman-Morrison-Woodbury formula[1]:

$$\left(W + \sum_{k=1}^{s} \mathbf{a}_k \mathbf{b}_k^\top\right)^{-1} = W^{-1} - W^{-1} A \mathcal{M}^{-1} B^\top W^{-1}$$

where $A = (\mathbf{a}_1, \cdots, \mathbf{a}_s)$, $B = (\mathbf{b}_1^\top, \cdots, \mathbf{b}_s^\top)$ and $\mathcal{M}$ is a square matrix of size s. All above analyses indicate that the BCD subproblem can be efficiently solved, which will contribute to the high efficiency of BCDNPKL.

3.4. Global Convergence

The convergence of BCD optimization has been intensively studied, interested readers can refer to (Tseng & Yun, 2009; Grippo & Sciandrone, 2000). It is well known that without certain safeguards, BCD implementation cannot be guaranteed to converge. Fortunately, from the componentwise convexity of $\Omega(V)$, i.e., $\Omega(V)$ is strictly convex w.r.t. each $\mathbf{v}_i$, we can provide a formal proof for the convergence of *Algorithm 1* in Theorem 1.

Theorem 1: If $\{V^{(t)}\}$ is the sequence of iterates generated from the updates in the *Algorithm* 1, then $V^{(t)}$ globally converges to a stationary point.

Proof: Clearly, the update of V in Algorithm 1 produces a sequence of nondecreasing objective function values $\Omega(V^{(0)}) \geq \Omega(V^{(1)}) \geq, \cdots, \geq \Omega(V^{(t)}) \geq 0$. Because of $\Delta \succ 0$ (see Section 2), we known $\Delta_{U,U} \succ 0$, which means $\tilde{\Delta} \succ 0$ because as a Schur complement, $\tilde{\Delta} \succ 0$ if and only if $\Delta \succ 0$ and $\Delta_{U,U} \succ 0$. Hence, the minimal eigenvalue of $\tilde{\Delta}$, $\lambda_{\min}(\tilde{\Delta}) > 0$. Further because of $\ell(\cdot) \geq 0$, we have $\lambda_{\min}(\tilde{\Delta})||V^{(t)}||_{\mathcal{F}}^2 \leq tr({V^{(t)}}^\top \tilde{\Delta} V^{(t)}) \leq \Omega(V^{(t)}) \leq \Omega(V^{(0)})$. Hence,

$$||V^{(t)}||_{\mathcal{F}}^2 \leq \Omega(V^{(0)})/\lambda_{\min}(\tilde{\Delta})$$

This means that the level set of $\{V|\Omega(V) \leq \Omega(V^{(0)})\}$ is compact so that $\{V^{(t)}\}$ can converge to a limit point $V^\star$. From Eq.s 9∼11, we know that for any loss function, the subproblem w.r.t. $\mathbf{v}_i (i = 1, \cdots, l)$ is strictly convex. Hence, according to Proposition 5 in (Grippo & Sciandrone, 2000), the limit point $V^\star$ is also a stationary point. ∎

[1]Available at: http://arxiv.org/abs/0807.3860.

Remark: Theorem 1 states that BCDNPKL produces at least a local-optimal solution. Also, we can conclude that BCDNPKL has at least linear convergence rate since BCD method has the same convergence rate as gradient descent method (Tseng & Yun, 2009).

4. Experiments

Like (Hoi et al., 2007), we examine both effectiveness and efficiency of the proposed approach by clustering. That is, a kernel matrix is first learned from the pairwise constraints using the proposed algorithms, and then the kernelized K-means algorithm is employed to cluster examples. The clustering accuracy of the kernelized K-means will be used to evaluate the quality of the learned kernel matrix, while CPU time (the time for clustering is excluded) for efficiency. Each clustering experiment is repeated by 20 trials with multiple restarts and all baselines are given the same random set of initial cluster centers in each trial. For effectiveness evaluation, we adopt two measures. One is the pairwise accuracy (Hoi et al., 2007)

$$\text{Pair-Accuracy} = \sum_{i>j} \frac{\mathbf{1}\{\mathbf{1}\{c_i=c_j\}=\mathbf{1}\{\hat{c}_i=\hat{c}_j\}\}}{0.5n(n-1)}$$

This metric measures the percentage of example pairs that are correctly clustered together. The other is the normalized mutual information (NMI) also used in (Kulis et al., 2006). NMI measures the amount of statistical information shared by the random variables representing the cluster and ground-truth class distributions.

4.1. Baselines

We compare BCDNPKL+HL/SHL/SL with the following state-of-the-art NPKL approaches:

• **NPK**: This approach is specific to hinge loss and focuses on deriving the dual problem of Eq.1. Following (Zhuang et al., 2009), we solve the dual problem by using a standard SDP solver SeDuMi[2].

• **SimpleNPKL+SHL**: This approach is specific to square hinge loss by adding a constraint into the primal problem in Eq.1 so that Z has a closed-form solution under min-max (i.e., primal-dual) framework. This min-max problem is solved by alternatively iterating between primal and dual variables. The linear loss (LL) also suggested by the authors, however, we do not compare with SimpleNPKL+LL in that it is uncompetitive to SimpleNPKL+SHL in effectiveness.

• **ITML**(Davis et al., 2007): This approach focuses on learning a Mahalanobis distance matrix A from

[2]Available at http://sedumi.ie.lehigh.edu.

pairwise constraints by using Bregman optimization. Clearly, for a learned A, a kernel matrix $Z = X^\top \mathrm{A} X$ can be generated as our comparison, where $X = (\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n)$.

• **SDPLR**(Burer & Monteiro, 2003): This approach reformulates SDP framework into a NLP problem by the idea of low-rank factorization, and then solves the NLP reformulation by using the augmented Lagrangian method.

4.2. Experiment Setup

We divide all test data sets into three groups: The **first group** includes nine small data sets: *chessboard, double-spiral, glass, heart, iris, protein, sonar, soybean* and *wine*, all of which are also used in the previous studies (Hoi et al., 2007; Zhuang et al., 2009). The **second group** (depicted in Table 1) includes six data sets and all of them are also used in (Zhu et al., 2004). The **third group** (depicted in Table 2) includes the seven *Adult*[1] data sets, of which the first five are also used in (Zhuang et al., 2009).

For complete evaluation, we consider both sparse and dense graphs for constructing Δ. For sparse graph test, following (Zhuang et al., 2009), we set $k = 5$ as the number of nearest neighbors for the first group data sets and $k = 50$ for the third group. Following (Zhu et al., 2004), we set $k = 100$ for *isolet*, and $k = 10$ for the other data sets in second group. For dense graph test, we construct the weighted graphs for all third group data sets, where the similarity matrix $S = (S_{i,j})$ is given by $S_{i,j} = exp\{-||\mathbf{x}_i - \mathbf{x}_j||^2/(2\sigma^2)\}$ for $i \neq j$ and 0 otherwise, and the factor σ is fixed as half of averaged distance between each sample and its top-10 nearest neighbors. The factor C involved in BCDNPKL/SimpleNPKL/NPK and γ in ITML, are fixed as 1 for all the second and third group data sets, but they are tuned in the range of $\{0.1, 0.2, 0.4, 0.6, 0.8, 1, 2, 4, 6, 8, 10\}$ for all the first group ones. We set the parameter $\delta = \frac{C}{2}$ in Algorithm 1 and the parameter B in SimpleNPKL following (Zhuang et al., 2009). The rank parameter r is estimated from Proposition 3 to the involved algorithms BCDNPKL and SimpleNPKL. For iteration, we generate the initial point $V^{(0)}$ by using the MATLAB function $rand(r, l)$.

To examine the performance extensively, on the second group data sets, we set both *little* and *much* side-information (but only *much* case for the first and third group data sets). Specifically, similar to (Hoi et al., 2007), we randomly generate $4n \times 10\%$ pairs of constraints for *little* side-information setting and $4n \times 30\%$ for *much* case, where one half of constraints belongs to must-link and the other to cannot-link.

[1] Available at http://www.csie.ntu.edu.tw/ ∼cjlin/libsvmtools/datasets/binary.html.

Table 1. The second group data sets.

Data set	pc -mac	baseball -hockey	one -two	odd -even	ten -digits	isolet
#Classes	2	2	2	2	10	26
#Instances	1,943	1,993	2,200	4,000	4,000	7,797

Table 2. The third group data sets including seven *Adult* data sets (for each one, #Classes=2, #Features=123).

Data set	a1a	a2a	a3a	a4a	a5a	a6a	a7a
#Instances	1,605	2,265	3,185	4,781	6,414	11,220	16,100

All the codes are implemented in MATLAB 7.1(R14), and all the experiments on the second and third group data sets are carried out on a server running Ubuntu with AMD CPU (4 cores, 2.3GHz and 8G RAM), while the experiments on the first group data sets on a PC running Windows XP with AMD-Turion(tm)-64-X2 (1.6 GHz, 960MB RAM).

4.3. Results and Analyses

The results on the first group data sets are reported in Table 3, from which our main conclusions are: 1) BCDNPKL+SL is always faster than SimpleNPKL+SHL; 2) Both NPK and SDPLR are significantly slower than the other algorithms in general, and ITML seems uncompetitive to SimpleNPKL+SHL in terms of accuracy. The 2) is also the reason why we give up comparison with NPK, ITML and SDPLR further on the second and third group date sets.

Regarding the results on the second group data sets, the NMI accuracy and CPU time of BCDNPKL and SimpleNPKL are reported in Table 4, from which we can conclude: 1) BCDNPKL+SL gets all the best efficiencies and both BCDNPKL with HL and SHL are also faster than SimpleNPKL+SHL on *isolet* data; 2) The speedups of BCDNPKL over SimpleNPKL are more sensitive on the *little* constraint setting than those on the *much* case.

For further comparisons in scalability, the results on the third group data sets are listed in Table 5. Our main conclusions are: 1) BCDNPKL+SL is always fastest and all BCDNPKL with HL, SHL or SL are always faster than SimpleNPKL+SHL on the data sets from *a4a* to *a7a*; 2) Due to avoiding finding k-nearest neighbors, the time of constructing the dense graphs is considerably lower than that on the sparse ones; 3) Implementing BCDNPKL on a large-scale dense graph is almost as efficient as that on the correspond-

ing sparse graph, on the contrary, implementing SimpleNPKL on a large-scale dense graph is significantly slower than that on its sparse graph because of eigen-decomposition. So SimpleNPKL is prohibitive from running on the data sets *a6a* and *a7a* when the dense graphs are chosen.

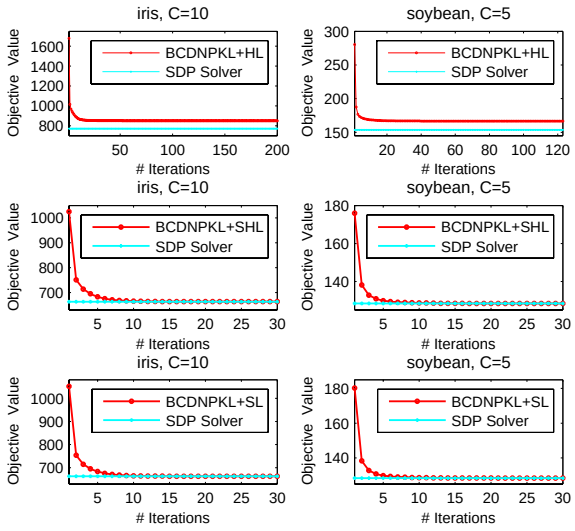

Figure 1. The comparison of convergent objective values between BCDNPKL and the baselines based on SDP solver on data sets: *iris* ($C = 10$) and *soybean* ($C = 5$).

Further, we compare the convergent objective values between BCDNPKL+HL/SHL/SL and the baselines based on standard SDP solvers (SeDuMi/YALMIP). Figure 1 shows the comparison on two data sets: *iris* ($C = 10$) and *soybean* ($C = 5$), from which our main conclusions are: 1) BCDNPKL often converges within 10 iterations; 2) the difference of the objective values between BCDNPKL+SHL/SL and the SDP solver approaches zero, our conjecture is that BCDNPKL+SHL/SL maybe contribute to a global optimal solution. However, the difference of the objective values between BCDNPKL+HL and the SDP solver is clearly larger than zero, which should be that BCDNPKL+HL actually generate a different solution from the SDP solver.

5. Conclusions

For performing NPKL from pairwise constraints, our BCDNPKL algorithm is superior especially in scalability. In the future, we will parallelize the implementation of BCDNPKL for further speedup.

Acknowledgments

This research work was partially supported by NSFC (60973097, 61035003).

References

Burer, S. and Monteiro, R.D.C. A nonlinear programming algorithm for solving semidefinite programs via low-rank factorization. *Mathematical Programming (series B)*, 95:329–357, 2003.

Chapelle, O., Weston, J., and Schölkopf, B. Cluster kernels for semi-supervised learning. In *NIPS 15*, pp. 585–592, 2002.

Davis, J.V., Kulis, B., Jain, P., Sra, S., and Dhillon, I.S. Information-theoretic metric learning. In *ICML*, pp. 209–216, 2007.

Grippo, L. and Sciandrone, M. On the convergence of the block nonlinear gauss-seidel method under convex constraints. *Operations Research Letters*, 26(3): 127–136, 2000.

Hoi, S.C.H., Jin, R., and Lyu, M.R. Learning nonparametric kernel matrices from pairwise constraints. In *ICML*, pp. 361–368, 2007.

Hsieh, C.-J., Chang, K.-W., Lin, C.-J., and Sundararajan, S. A dual coordinate descent method for large-scale linear svm. In *ICML*, pp. 408–415, 2008.

Hu, E.-L., Chen, S.-C., Zhang, D.-Q., and Yin, X.-S. Semi-supervised kernel matrix learning by kernel propagation. *IEEE TNN*, 21(11):1831–1841, 2010.

Jost, J. *Riemannian geometry and geometric analysis*. Berlin, New York, 4th edition, 2005.

Kulis, B., Sustik, M., and Dhillon, I.S. Learning low-rank kernel matrices. In *ICML*, pp. 505–512, 2006.

Lanckriet, G.R.G., Cristianini, N., Bartlett, P.L., Ghaoui, L.E., and Jordan, M.I. Learning the kernel matrix with semidefinite programming. *JMLR*, 5(1):27–72, 2004.

Smola, A. and Kondor, R. Kernels and regularization on graphs. In *COLT*, pp. 144–158, 2003.

Tseng, P. and Yun, S. Block-coordinate gradient descent method for linearly constrained nonsmooth separable optimization. *Journal of Optimization Theory and Applications*, 140(3):513–535, 2009.

Zhu, X., Kandola, J.S., Ghahramani, Z., and Lafferty, J.D. Nonparametric transforms of graph kernels for semi-supervised learning. In *NIPS 17*, pp. 1641–1648, 2004.

Zhuang, J., Tsang, I.W., and Hoi, S.C.H. Simplenpkl: simple non-parametric kernel learning. In *ICML*, pp. 1273–1280, 2009.

Table 3. Pair-Accuracy plus standard deviation ($\times 10^2$) and CPU time(s) (in parenthesis) of BCDNPKL, compared with ITML, NPK, SDPLR and SimpleNPKL. (The best results are in bold font and "#Cons." is the shorthand of "the number of pairwise constraints".)

Data Set	#Cons.	ITML	NPK	SDPLR	SimpleNPKL	BCDNPKL		
					SHL	HL	SHL	SL
iris	180	97.6±0.59 (**0.2**)	98.1±1.78 (68.0)	98.1±1.78 (43.4)	98.8±0.77 (1.0)	98.6±0.82 (5.7)	**98.9±0.75** (4.0)	**98.9±0.75** (0.7)
sonar	249	66.3±4.78 (2.2)	73.9±1.66 (91.0)	74.1±1.68 (21.9)	93.2±2.30 (1.3)	93.6±2.28 (5.3)	**94.4±1.58** (3.8)	**94.4±1.58** (**0.7**)
chboard	120	49.9±0.75 (**0.1**)	75.0±2.08 (9.4)	74.8±1.08 (5.3)	91.6±4.24 (0.6)	92.7±3.73 (2.6)	**94.1±3.27** (1.8)	**94.1±3.27** (0.3)
glass	256	69.0±1.65 (**0.2**)	74.8±2.44 (58.3)	74.8±1.41 (9.6)	74.2±3.04 (1.1)	78.7±1.91 (5.2)	79.9±1.99 (4.1)	**80.2±2.00** (0.8)
heart	324	58.9±2.05 (**0.3**)	63.6±3.97 (119.4)	63.6±3.97 (19.0)	83.4±4.21 (1.9)	82.0±2.22 (6.3)	**85.1±2.74** (5.3)	**85.1±2.59** (1.2)
protein	139	85.5±1.70 (**0.2**)	85.0±2.10 (12.8)	85.1±2.15 (5.9)	84.2±2.87 (0.7)	**86.4±2.77** (2.9)	86.2±2.43 (2.1)	**86.4±2.53** (0.4)
soybean	56	**100.0±0.00** (**0.1**)	96.3±6.23 (2.1)	95.8±7.15 (3.8)	96.5±4.79 (0.3)	96.7±4.04 (1.5)	96.0±5.83 (0.9)	96.0±5.83 (**0.1**)
wine	214	71.9±0.02 (**0.2**)	67.6±2.40 (47.5)	67.6±2.44 (11.1)	78.8±4.18 (1.0)	81.0±3.10 (4.4)	**81.5±3.32** (3.3)	**81.5±3.40** (0.6)
spiral	120	50.3±0.73 (**0.1**)	**99.8±0.62** (8.0)	**99.8±0.62** (3.0)	99.5±0.89 (0.6)	**99.8±0.62** (3.5)	**99.8±0.62** (1.9)	**99.8±0.62** (0.3)

Table 4. NMI-Accuracy plus standard deviation ($\times 10^2$) and CPU time (s) of BCDNPKL, compared with SimpleNPKL+SHL. For each data set, we test both the *little* and *much* constraint settings. (The best results are in bold font and the last "Speedup" columns show the speedups over SimpleNPKL+SHL.)

Data Set	"little"/"much" #Cons.	NMI accuracy (%)				CPU Time(s)				Speedup		
		SimpleNPKL	BCDNPKL			SimpleNPKL	BCDNPKL			BCDNPKL		
		SHL	HL	SHL	SL	SHL	HL	SHL	SL	HL	SHL	SL
baseball-hockey	796	91.9 ± 1.29	**92.6 ± 1.24**	**92.6 ± 1.29**	**92.6 ± 1.29**	12.2	14.2	11.0	**4.0**	0.9	1.1	**3.1**
	2388	98.0 ± 0.72	**98.4 ± 0.62**	98.3 ± 0.75	98.3 ± 0.75	27.4	59.5	35.9	**12.0**	0.5	0.8	**2.3**
pc-mac	776	79.7 ± 1.22	**81.2 ± 1.34**	80.6 ± 1.07	80.6 ± 0.39	12.2	13.4	10.7	**3.9**	0.9	1.1	**3.1**
	2328	94.8 ± 2.63	95.5 ± 5.35	**95.7 ± 3.38**	**95.7 ± 1.13**	26.3	53.5	33.8	**11.3**	0.5	0.8	**2.3**
one-two	876	98.3 ± 0.46	**98.5 ± 0.55**	98.4 ± 0.50	98.4 ± 0.50	14.2	16.7	12.5	**4.9**	0.9	1.1	**2.9**
	2640	99.5 ± 0.58	**99.6 ± 0.46**	**99.6 ± 0.47**	**99.6 ± 0.47**	33.5	71.2	42.0	**14.6**	0.5	0.8	**2.3**
odd-even	1596	87.2 ± 1.11	**88.7 ± 1.00**	88.6 ± 1.01	88.6 ± 1.00	53.9	36.1	28.0	**14.7**	1.5	1.9	**3.7**
	4800	97.0 ± 0.73	97.5 ± 0.61	**97.6 ± 0.65**	**97.6 ± 0.65**	125.1	163.3	119.5	**69.7**	0.8	1.0	**1.8**
ten digits (10 classes)	1596	46.8 ± 1.81	**64.3 ± 1.76**	58.7 ± 1.74	58.6 ± 1.75	53.3	32.5	27.6	**14.3**	1.6	1.9	**3.7**
	4800	48.8 ± 2.33	**83.8 ± 1.67**	78.3 ± 1.03	78.2 ± 0.95	123.3	134.9	117.3	**68.7**	0.9	1.1	**1.8**
isolet (26 classes)	3116	51.7 ± 2.10	**61.5 ± 1.16**	60.3 ± 1.27	60.2 ± 1.12	299.7	117.3	108.8	**83.5**	2.6	2.8	**3.6**
	9356	57.9 ± 1.00	**74.1 ± 0.60**	72.6 ± 0.70	72.7 ± 0.82	610.4	403.6	374.8	**288.8**	1.5	1.6	**2.1**

Table 5. Pair-Accuracy plus standard deviation ($\times 10^2$) and CPU time (s) of BCDNPKL, compared with SimpleNPKL+SHL. For each data set, we test both the sparse (*S*) and dense (*D*) graph constructions. The CPU times of constructing graphs are shown in parenthesis of the third column. ("—" means that SimpleNPKL+SHL runs too slow to produce a result.)

Data Set	#Cons.	Time (s) of constructing Graph	Pair-Accuracy (%)				CPU Time (s)				Speedup		
			SimpleNPKL	BCDNPKL			SimpleNPKL	BCDNPKL			BCDNPKL		
			SHL	HL	SHL	SL	SHL	HL	SHL	SL	HL	SHL	SL
a1a	1926	*S* (10. 7)	89.3 ± 0.85	96.3 ± 0.64	**97.1 ± 0.56**	**97.1 ± 0.56**	23.3	40.7	27.2	**8.8**	0.6	0.9	**2.6**
		D (1.3)	94.7 ± 0.84	96.4 ± 0.79	**97.3 ± 0.55**	**97.3 ± 0.55**	274.6	38.3	25.0	**8.0**	7.2	11.0	**34.3**
a2a	2718	*S* (17. 7)	94.6 ± 0.48	96.0 ± 0.45	**96.9 ± 0.42**	**96.9 ± 0.42**	45.1	66.4	48.3	**17.9**	0.7	0.9	**2.5**
		D (2.19)	93.2 ± 0.68	96.0 ± 0.50	**97.0 ± 0.43**	**97.0 ± 0.43**	747.7	62.9	44.1	**15.8**	11.9	17.0	**47.3**
a3a	3822	*S* (31. 2)	65.2 ± 0.83	95.0 ± 0.65	**96.6 ± 0.33**	**96.6 ± 0.33**	94.4	103.5	85.8	**45.0**	0.9	1.1	**2.1**
		D (2.2)	87.2 ± 0.85	94.1 ± 0.69	**96.2 ± 0.48**	**96.2 ± 0.49**	2,047.2	97.7	80.1	**40.2**	21.0	25.6	**50.9**
a4a	5738	*S* (82. 2)	80.6 ± 0.66	94.6 ± 0.50	**96.3 ± 0.41**	**96.3 ± 0.41**	214.3	194.9	169.4	**112.2**	1.1	1.3	**1.9**
		D (5.0)	87.5 ± 0.71	93.7 ± 0.44	**96.0 ± 0.35**	**96.0 ± 0.35**	6,834.1	185.8	159.4	**103.2**	36.8	42.9	**66.2**
a5a	7698	*S* (160.3)	70.3 ± 0.67	94.7 ± 0.50	**96.3 ± 0.39**	**96.3 ± 0.40**	393.3	312.4	278.2	**207.7**	1.3	1.4	**1.9**
		D (8.9)	87.2 ± 0.72	94.0 ± 0.36	**96.3 ± 0.20**	**96.3 ± 0.20**	16,320.6	301.9	265.4	**193.9**	54.1	61.5	**84.2**
a6a	13464	*S* (595.1)	78.3 ± 0.83	94.9 ± 0.39	**96.5 ± 0.32**	**96.5 ± 0.32**	1,213.4	844.7	787.1	**685.9**	1.4	1.5	**1.8**
		D (26.9)	—	94.1 ± 0.35	**96.3 ± 0.21**	**96.3 ± 0.21**	>52,000	831.3	762.5	**650.0**	—	—	—
a7a	19320	*S* (1447)	95.8 ± 2.3	94.9 ± 0.27	**96.5 ± 0.21**	**96.5 ± 0.21**	2,640.0	1,714.5	1,636.8	**1,501.7**	1.5	1.6	**1.8**
		D (140.8)	—	93.7 ± 0.27	**96.0 ± 0.21**	**96.0 ± 0.21**	>118,500	1,721.1	1,630.4	**1,481.9**	—	—	—