The Thirty-Second AAAI Conference
on Artificial Intelligence (AAAI-18)

Asynchronous Doubly Stochastic Sparse Kernel Learning

Bin Gu,[1] **Xin Miao,**[2] **Zhouyuan Huo,**[1] **Heng Huang**[1*]
[1]Department of Electrical & Computer Engineering, University of Pittsburgh, USA
[2]Dept. of Computer Science and Engineering, University of Texas at Arlington, USA
big10@pitt.edu, xin.miao@mavs.uta.edu, zhouyuan.huo@pitt.edu, heng.huang@pitt.edu

Abstract

Kernel methods have achieved tremendous success in the past two decades. In the current big data era, data collection has grown tremendously. However, existing kernel methods are not scalable enough both at the *training and predicting* steps. To address this challenge, in this paper, we first introduce a general sparse kernel learning formulation based on the random feature approximation, where the loss functions are possibly non-convex. Then we propose a new asynchronous parallel doubly stochastic algorithm for large scale sparse kernel learning (AsyDSSKL). To the best our knowledge, AsyDSSKL is the first algorithm with the techniques of asynchronous parallel computation and doubly stochastic optimization. We also provide a comprehensive convergence guarantee to AsyDSSKL. Importantly, the experimental results on various large-scale real-world datasets show that, our AsyDSSKL method has the significant superiority on the computational efficiency at the *training and predicting* steps over the existing kernel methods.

Introduction

Kernel methods have achieved tremendous success in the past two decades for non-linear learning problems. There are a large number of successful and popular kernel methods (Vapnik 1998; Zhu and Hastie 2005; Zhu et al. 2004; Baudat and Anouar 2000; Vovk 2013; Li, Yang, and Xing 2005) for various learning problems. We take binary classification and regression for example. Support vector classification (SVC) (Vapnik 1998), kernel logistic regression (Zhu and Hastie 2005) and 1-norm SVC (Zhu et al. 2004) are the popular kernel methods for binary classification. Support vector regression (Vapnik 1998), kernel ridge regression (Vovk 2013) and kernel Lasso (Li, Yang, and Xing 2005) are the popular kernel methods for regression. These kernel methods have been successfully applied to solve various real-world applications (such as computational biology (Schölkopf, Tsuda, and Vert 2004) and remote sensing data analysis (Camps-Valls and Bruzzone 2009)).

However, traditional kernel methods need to store and compute the kernel matrix with the size of $O(l^2)$ where l is the training sample size. When l is large, the kernel matrix can be neither stored in local memory nor computed. Even worse, the kernel methods normally have the computational complexity $O(l^3)$ for the training. To address the scalability issue of kernel methods in the training step, several decomposition algorithms (Takahashi and Nishi 2006) have been proposed for training the kernel methods. However, even for the state-of-the-art implementations (*e.g.* LIBSVM software package), the observed computational complexity is $O(l^\kappa)$ where $1 < \kappa < 2.3$ (Chang and Lin 2011a). More related works of training kernel methods are discussed in the next section. To sum up, in the current big data era, the existing kernel methods are not scalable enough at the training step.

Besides the scalability issue at the training step, traditional kernel methods are also not scalable at the predicting step. Specifically, the computational complexity for predicting a testing sample is $O(l)$, because normally the number of support vectors grows linearly with the sample size. Thus, the computational complexity of predicting a testing set with similar size of training set is $O(l^2)$. To address the scalability issue of kernel methods in the predicting step, a compact (or sparse) model is preferred. The direct method for obtaining a compact model is adding sparse constraint (normally 1-norm) on the coefficients of the model. For example, (Zhu et al. 2004) imposed the 1-norm on the SVC formulation. (Yen et al. 2014) imposed 1-norm on the random features. Li, Yang, and Xing (2005) reformulated Lasso into a form isomorphic to SVC, which generates a sparse solution in the nonlinear feature space. However, these methods are not scalable at the training step.

To address the scalability issues of kernel methods at the *training and predicting* steps, in this paper, we first introduce a general sparse kernel learning formulation with the random feature approximation, where the loss functions are possibly non-convex. Then, we propose a new asynchronous parallel doubly stochastic algorithm for sparse kernel learning (AsyDSSKL). We believe this is very important for kernel methods for four reasons. **1) Generality:** AsyDSSKL works for a general class of sparse kernel methods based on the random feature approximation, where the loss function is possibly non-convex. **2) Efficient computation:** The pivotal step of AsyDSSKL is to compute the doubly stochastic gradient on a mini-batch of samples and a selected coordinate, which is quite efficient. **3) Comprehensive convergence guarantees:** AsyDSSKL achieves a sublinear rate when the smooth loss functions are convex or non-convex.

*To whom all correspondence should be addressed.

AsyDSSKL achieves a linear (geometric) convergence rate when the objective function is with the optimal strong convexity. **4) Strong empirical performance:** The experimental results on various large-scale real-world datasets show that, AsyDSSKL has the significant superiority on the scalability and efficiency at the *training and predicting* steps over the existing kernel methods.

Novelties. The main novelties of this paper are summarized as follows.

1. To the best of our knowledge, our AsyDSSKL is the first algorithm with the techniques of asynchronous parallel computation and doubly stochastic optimization. Although our objective function (4) can be solved by (Liu and Wright 2015), their method is stochastic only on coordinates which is not scalable to large sample size. However, our AsyDSSKL is stochastic both on samples and coordinates.
2. We provide the convergence rates of AsyDSSKL in the different settings, which is nontrivial although our proofs follow (Liu and Wright 2015). Specially, we provide the convergence rate of AsyDSSKL in the non-convex setting, which is new and not included in (Liu and Wright 2015).

Notations. To make the paper easier to follow, we give the following notations.

1. e_j is the zero vector in $\mathbb{R}^n$ except that the j-th coordinate equal to 1.
2. $\mathcal{P}_{j,g_j}$ is the componentwise proximal operator as $\mathcal{P}_{j,g_j}(w') = \arg\min_w \frac{1}{2}\|w - w'\|^2 + h\left((w)_j\right)$.
3. $\|\cdot\|$ denotes the Euclidean norm, and $\|\cdot\|_\infty$ denotes the infinity norm.

Related Works

In this section, we briefly review the techniques of parallel computation and stochastic optimization for training kernel methods.

Parallel computation. The parallel computation as the basic big data technique can be roughly divided into synchronous and asynchronous models according to whether the reading or writing lock is used. The most of parallel algorithms for kernel methods are based on the synchronous model due to its simplicity. For example, Chang and Lin (2011b) proposed a modified LIBSVM implementation by using the synchronous parallel technique to compute a column of kernel matrix. Zhao and Magoules (2011) and You et al. (2015) proposed synchronous parallel SVM algorithms for the parallel environment with shared memory. The asynchronous computation is much more efficient than the synchronous computation, because it keeps all computational resources busy all the time. However, the convergence analysis for the asynchronous computation is much more difficult due to the *inconsistent reading and writing*. To the best of our knowledge, the only work for the asynchronous parallel kernel methods is the asynchronous parallel greedy coordinate descent algorithm for SVC (You et al. 2016). However, their work does not exploit another important big data technology, *i.e.*, stochastic optimization, and is not scalable at the predicting step.

Stochastic optimization. Stochastic optimization is another big data technique which could be stochastic on samples and coordinates. Specifically, Shalev-Shwartz, Singer, and Srebro (2007) and Kivinen, Smola, and Williamson (2004) proposed stochastic algorithms for kernel SVMs which are stochastic on samples. Shalev-Shwartz and Zhang (2013) proposed a stochastic dual coordinate ascend algorithm for kernel methods which is stochastic on coordinates. However, they are not scalable enough because the computational complexities are $O(T^2)$ where T is the iteration number. To address this issue, (Dai et al. 2014) proposed a doubly stochastic kernel method, which is stochastic both on samples and random features. Random feature mapping (Rahimi and Recht 2007) (a method of approximating kernel function) is an effective method to scale up kernel methods. However, as analyzed in (Rahimi and Recht 2009), to achieve a good generalization ability, the number of random features needs to be $O(l)$. Thus, the predicting of (Rahimi and Recht 2009) is not scalable due to a huge number of random features in the model. In addition, (Rahimi and Recht 2009) does not exploit the parallel computation techniques as mentioned above.

Asynchronous Doubly Stochastic Sparse Kernel Learning

In this section, we first give a brief review of random features, and then introduce a general sparse kernel learning formulation based on the random feature approximation, where the loss functions are possibly non-convex. Finally, we propose our AsyDSSKL algorithm.

Random Features

Given a training set $S = \{(X_i, y_i)\}_{i=1}^l$ with $X_i \in \mathbb{R}^n$, and $y_i \in \{+1, -1\}$ for binary classification or $y_i \in \mathbb{R}$ for regression. For kernel methods, the optimal model can be represented as $f(X_i) = \sum_{k=1}^l \alpha_i K(X_k, X_i)$ according to the representer theorem (Schölkopf, Herbrich, and Smola 2001). Thus, one benefit of kernel methods is that, we can learn a nonlinear model directly using kernel functions, instead of computing the inner production in the explicit kernel space H. However, the number of non-zero α_i often increases linearly with data size (Steinwart and Christmann 2008). Even worse, as mentioned previously, kernel methods need to store and compute the kernel matrix with the size $O(l^2)$. To address the scalability issue, random feature mapping (Rahimi and Recht 2007) was proposed to explicitly approximate the kernel function $K(\cdot, \cdot)$. Specifically, by Mercer's theorem (Mercer 1909), for any positive semi-definite kernel $K(\cdot, \cdot)$, there exists a kernel space H, a probability distribution $p(h)$ and a random feature mapping $\phi_h(x)$, such that

$$K(X_i, X_j) = \int_{h \in H} p(h)\phi_h(X_i)\phi_h(X_j)dh \tag{1}$$

Based on the random features, the kernel function (1) is approximated by $K(X_i, X_j) \approx \sum_{h \in A} \phi_h(X_i)\phi_h(X_j)$, where A is a set of random features drawn from the kernel space H according to the distribution $p(h)$. Thus, the model function

Table 1: Commonly used smooth loss functions for binary classification (BC) and regression (R).

Type of function	Name of loss	Type of task	The loss function
Convex	Square loss	BC+R	$L(f(X_i), y_i) = (f(X_i) - y_i)^2$
	Logistic loss	BC	$L(f(X_i), y_i) = \log(1 + \exp(-y_i f(X_i)))$
	Smooth hinge loss	BC	$L(f(X_i), y_i) = \begin{cases} \frac{1}{2} - z_i & \text{if } z_i \leq 0 \\ \frac{1}{2}(1 - z_i)^2 & \text{if } 0 < z_i < 1 \\ 0 & \text{if } z_i \geq 1 \end{cases}$, where $z_i = y_i f(X_i)$.
Non-convex	Correntropy induced loss	R	$L(f(X_i), y_i) = \frac{\sigma^2}{2}\left(1 - e^{-\frac{(y_i - X_i^T x)^2}{\sigma^2}}\right)$
	Sigmoid loss	BC	$L(f(X_i), y_i) = \frac{1}{1+\exp(-y_i f(X_i))}$

can be approximated by

$$f(X_i) \approx \sum_{h \in A} w(h)\phi_h(X_i) \tag{2}$$

However, as analyzed in (Rahimi and Recht 2009; Yen et al. 2014), to achieve a good generalization ability, the number of random features needs to be $O(l)$. Because the model size of (2) grows linearly with l, it is highly desirable to have a sparse learning algorithm which would be beneficial to the efficiency at the predicting step.

Sparse Kernel Learning with Random Features

In the infinite-dimensional kernel space, the kernel methods are normally written as (3) based on (1).

$$\min_{w(h), h \in H} Q(w) = \tag{3}$$
$$\frac{1}{l}\sum_{i=1}^{l} L\left(\int_{h \in H} p(h)w(h)\phi_h(X_i)dh, y_i\right) + \lambda\|w\|^2$$

To produce a compact model, Yen et al. (2014) proposed a sparse kernel learning formulation (4) based on the random feature approximation. They proved that $\min_{w(h), h \in A} P(w)$ is an $O(\frac{1}{|A|})$-approximation of $Q(w^*)$ in Corollary 1 of (Yen et al. 2014).

$$\min_{w(h), h \in A} P(w) = \overbrace{\frac{1}{l}\sum_{i=1}^{l} \underbrace{L\left(\sum_{h \in A} w(h)\phi_h(X_i), y_i\right)}_{f_i(w)}}^{F(w)} + \underbrace{\lambda \sum_{h \in A} |w(h)|}_{g(w)} \tag{4}$$

where $g(w)$ is a sparse constraint on w by 1-norm, and $L(\cdot, \cdot)$ is a smooth and convex loss function. In this paper, we relax $L(\cdot, \cdot)$ as a smooth, possibly non-convex loss function. Thus, the formulation (4) is a more general formulation which includes a large number of learning problems (e.g. regression, binary classification, multiple classification and so on). Table 1 presents several common used smooth loss functions for binary classification and regression. Note that the correntropy induced loss (Feng et al. 2015) and the sigmoid loss (Lin 2004) are non-convex.

Yen et al. (2014) provided an iterative framework to solve the sparse kernel learning formulation (4). In order to incorporate the parallel computation, we modify the framework as Algorithm 1, where the iteration number is limited. Empirically $M = 5$ can achieve a good result. The key of Algorithm 1 is to design a scalable algorithm for the sparse kernel learning problem (4) such that scaling well in sample size and dimensionality simultaneously.

Algorithm 1 Asynchronous sparse random feature learning framework

1: Initialize $w^0 = \mathbf{0}$, working set $A^0 = \emptyset$, and $t = 0$.
2: **for** $t = 0, 1, 2, \cdots, M - 1$ **do**
3: *All threads parallelly* sample $h_1, h_2, \cdots, h_R$ i.i.d. from a distribution $p(h)$, and add $h_1, h_2, \cdots, h_R$ into the set A.
4: Asynchronously solve (4) by AsyDSSKL.
5: Parallelly update $A^{t+1} = A^t - \{h : w^t(h) = 0\}$.
6: **end for**

AsyDSSKL Algorithm

In this section, we propose our AsyDSSKL algorithm to solve the sparse kernel learning problem (4). Our AsyDSSKL exploits the asynchronous parallel computation and doubly stochastic optimization, which is significantly different from (Liu and Wright 2015) as discussed before. Specifically,

1) For asynchronous parallel computation, our AsyDSSKL works in the parallel environment with shared memory, such as multi-core processors and GPU-accelerators. All cores in CPU or GPU can read and write the vector x in the shared memory simultaneously without any lock.

2) For doubly stochastic optimization, we use the techniques of stochastic variance reduced gradient (SVRG) (Zhang 2004) and stochastic coordinate descent (SCD) (Tseng and Yun 2009; You et al. 2016). SVRG is an accelerated version of stochastic gradient descent (SGD) algorithm (Shamir and Zhang 2013) which is stochastic on samples, and scales well in sample size. SCD is stochastic on features which scales well in dimensionality. Thus, our AsyDSSKL

can scale well in sample size and dimensionality simultaneously.

Specifically, following the SVRG (Zhang 2004) and SCD (Tseng and Yun 2009; You et al. 2016), our AsyDSSKL has two-layer loops. The outer layer is to parallelly compute the full gradient $\nabla F(w^s) = \frac{1}{l}\sum_{i=1}^{l}\nabla f_i(w^s)$, where the superscript s denotes the s-th outer loop. The inner layer is parallelly repeating the following three steps:

1. **Read:** Read the vector w from the shared memory to the local memory without reading lock. We use $\widehat{w}_t^{s+1}$ to denote the value in the local memory, where the subscript t denotes the t-th inner loop.

2. **Compute:** Randomly choose a mini-batch $\mathcal{B}$ from $\{1, ..., l\}$ and a coordinate j from $\{1, ..., n\}$, and locally compute $\widehat{v}_j^{s+1} = \frac{1}{|\mathcal{B}|}\sum_{i\in\mathcal{B}}\nabla_j f_i(\widehat{w}_t^{s+1}) - \frac{1}{|\mathcal{B}|}\sum_{i\in\mathcal{B}}\nabla_j f_i(\widetilde{w}^s) + \nabla_j F(\widetilde{w}^s)$.

3. **Update:** Update the coordinate j of the vector x in the shared memory as $w_{t+1}^{s+1} \leftarrow \mathcal{P}_{j,\frac{\gamma}{L_{\max}}g_j}\left((w_t^{s+1})_j - \frac{\gamma}{L_{\max}}\widehat{v}_j^{s+1}\right)$ without writing lock, where γ is the steplength, $L_{\max}$ is defined in the next section.

The detailed description of AsyDSSKL is presented in Algorithm 2. Note that in line 9 of Algorithm 2, $\widehat{v}_{j(t)}^{s+1}$ computed locally is an unbiased estimation of $\nabla_{j(t)}F(\widehat{w}_t^{s+1})$, because the expectation of $\widehat{v}_t^{s+1}$ on $\mathcal{B}$ is equal to $\nabla F(\widehat{w}_t^{s+1})$, i.e., $\mathbb{E}\widehat{v}_t^{s+1} = \nabla F(\widehat{w}_t^{s+1})$.

Algorithm 2 Asynchronous doubly stochastic sparse kernel learning algorithm (AsyDSSKL)

Input: The steplength γ, the number of outer loop iterations S, and the number of inner loop iterations m.
Output: x^S.

1: Initialize $w^0 \in \mathbb{R}^d$.
2: **for** $s = 0, 1, 2, \cdots, S-1$ **do**
3: $\quad\widetilde{w}^s \leftarrow w^s$
4: $\quad$*All threads parallelly* compute the full gradient $\nabla F(\widetilde{w}^s) = \frac{1}{l}\sum_i^l \nabla f_i(\widetilde{w}^s)$
5: $\quad$*For each thread*, do:
6: $\quad$**for** $t = 0, 1, 2, \cdots, m-1$ **do**
7: $\quad\quad$Randomly sample a mini-batch $\mathcal{B}$ from $\{1, ..., l\}$ with equal probability.
8: $\quad\quad$Randomly choose a coordinate $j(t)$ from $\{1, ..., n\}$ with equal probability.
9: $\quad\quad$Compute $\widehat{v}_{j(t)}^{s+1} = \frac{1}{|\mathcal{B}|}\sum_{i\in\mathcal{B}}\nabla_{j(t)} f_i(\widehat{w}_t^{s+1}) - \frac{1}{|\mathcal{B}|}\sum_{i\in\mathcal{B}}\nabla_{j(t)} f_i(\widetilde{w}^s) + \nabla_{j(t)}F(\widetilde{w}^s)$
10: $\quad\quad w_{t+1}^{s+1} \leftarrow \mathcal{P}_{j(t),\frac{\gamma}{L_{\max}}g_{j(t)}}\left(w_t^{s+1} - \frac{\gamma}{L_{\max}}e_{j(t)}\widehat{v}_{j(t)}^{s+1}\right)$.
11: $\quad$**end for**
12: $\quad w^{s+1} \leftarrow w_m^{s+1}$
13: **end for**

Convergence Rate Analysis of AsyDSSKL

Because AsyDSSKL runs in asynchronously without any lock, the *inconsistent reading and writing* could arise to the vector w in the shared memory, which makes the convergence analysis of AsyDSSKL more challenging. In this section, we assume that the delay for the vector w in the shared memory is bounded (i.e., Assumption 4), and provide the comprehensive convergence guarantees: AsyDSSKL achieves a sublinear convergence rate when the smooth loss functions are convex or non-convex (Theorems 1 and 2), and a linear convergence rate when the objective function has optimal strong convexity (Theorem 1).

In the following, we first give the several widely used assumptions (i.e., optimal strong convexity, Lipschitz smoothness, and bound of delay), then provide the main conclusions to the convergence rates of our AsyDSSKL.

Preliminaries

Optimal Strong Convexity: Let P^* denote the optimal value of (4), and let S denote the solution set of P such that $P(w) = P^*, \forall w \in S$. Firstly, we assume that S is nonempty (i.e., Assumption 1).

Assumption 1. *The solution set S of (4) is nonempty.*

Based on S, we define $\mathcal{P}_S(w) = \arg\min_{y\in S}\|y - w\|^2$ as the Euclidean-norm projection of a vector w onto S. Then, we assume that the convex function $P(w)$ is with the optimal strong convexity (i.e., Assumption 2).

Assumption 2 (Optimal strong convexity)**.**

$$P(w) - P(\mathcal{P}_S(w)) \geq \frac{\ell}{2}\|w - \mathcal{P}_S(w)\|^2 \tag{5}$$

As mentioned in (Liu and Wright 2015), the condition of optimal strong convexity is significantly weaker than the normal strong convexity condition. And several examples of optimally strongly convex functions that are not strongly convex are provided in (Liu and Wright 2015).

Lipschitz Smoothness: We define the normal Lipschitz constant (L_{nor}), restricted Lipschitz constant (L_{res}) and coordinate Lipschitz constant ($L_{\max}$) as follows.

Definition 1 (Lipschitz constants)**.** *L_{nor}, L_{res} and $L_{\max}$ are the normal Lipschitz constant, restricted Lipschitz constant and coordinate Lipschitz constant, respectively, for ∇f_i ($\forall i \in \{1, \cdots, l\}$) in (4). We have*

$$\|\nabla f_i(w_1) - \nabla f_i(w_2)\| \leq L_{nor}\|w_1 - w_2\|, \ \forall w_1, \forall w_2 \tag{6}$$

$$\|\nabla f_i(w) - \nabla f_i(w + e_j t)\| \leq L_{res}|t|, \ \forall w, \forall t \tag{7}$$

$$\|\nabla f_i(w) - \nabla f_i(w + e_j t)\|_\infty \leq L_{\max}|t|, \ \forall w, \forall t \tag{8}$$

Based on L_{nor} L_{res} and L_{max} as defined above, we assume that the function f_i ($\forall i \in \{1, \cdots, l\}$ is Lipschitz smooth with L_{nor} L_{res} and L_{max} (i.e., Assumption 3). In addition, we define $\Lambda_{res} = \frac{L_{res}}{L_{max}}$, $\Lambda_{nor} = \frac{L_{nor}}{L_{max}}$.

Assumption 3 (Lipschitz smoothness)**.** *The function f_i ($\forall i \in \{1, \cdots, l\}$ is Lipschitz smooth with the normal Lipschitz constant L_{nor}, restricted Lipschitz constant L_{res} and coordinate Lipschitz constant L_{max}.*

Bound of Delay: Because AsyDSSKL does not use the reading lock, the vector $\widehat{w}_t^{s+1}$ read to the local memory may be inconsistent to the vector w_t^{s+1} in the shared memory, which means that some components of $\widehat{w}_t^{s+1}$ are same with the ones in x_t^{s+1}, but others are different to the ones in x_t^{s+1}. However, we can define a set $K(t)$ of inner iterations, such that,

$$w_t^{s+1} = \widehat{w}_t^{s+1} + \sum_{t' \in K(t)} (w_{t'+1}^{s+1} - w_{t'}^{s+1}) \tag{9}$$

where $t' \leq t-1$. It is reasonable to assume that there exists an upper bound τ such that $\tau \geq t - \min\{t' | t' \in K(t)\}$ (i.e., Assumption 4).

Assumption 4 (Bound of delay)**.** *There exists an upper bound τ such that $\tau \geq t - \min\{t' | t' \in K(t)\}$ for all inner iterations t in AsyDSSKL.*

Convergence Rate Analysis

We provide the convergence rates of AsyDSSKL in the convex and non-convex settings (Theorems 1 and 2). The detailed proofs of Theorem 1 and 2 are presented in Appendix.

Convex Setting We first give the convergence rates of AsyDSSKL at the convex setting in Theorem 1.

Theorem 1. *Let ρ be a constant that satisfies $\rho > 1$, and define the quantity $\theta_1 = \frac{\rho^{\frac{1}{2}} - \rho^{\frac{\tau+1}{2}}}{1-\rho^{\frac{1}{2}}}$, $\theta_2 = \frac{\rho^{\frac{1}{2}} - \rho^{\frac{m}{2}}}{1-\rho^{\frac{1}{2}}}$ and $\theta' = \frac{\rho^{\tau+1}-\rho}{\rho-1}$. Suppose the nonnegative steplength parameter $\gamma > 0$ satisfies*

$$1 - \Lambda_{nor}\gamma - \frac{\gamma\tau\theta'}{n} - \frac{2(\Lambda_{res}\theta_1 + \Lambda_{nor}\theta_2)\gamma}{n^{1/2}} \geq 0 \tag{10}$$

If the optimal strong convexity holds for $P(w)$ with $\ell > 0$ (i.e., Assumption 2), we have

$$\mathbb{E}P(w^s) - P^* \leq \frac{L_{\max}}{2\gamma}\left(\frac{1}{1+\frac{m\gamma\ell}{n(\ell\gamma+L_{\max})}}\right)^s \cdot \left(\|w^0 - \mathcal{P}_S(w^0)\|^2 + \frac{2\gamma}{L_{\max}}\left(\mathbb{E}P(w^0) - P^*\right)\right)$$

If $f_i(w)$ is a general smooth convex function with Assumption 3, we have

$$\mathbb{E}P(w^s) - P^* \leq \frac{nL_{\max}\|w^0 - \mathcal{P}_S(w^0)\|^2 + 2\gamma n\left(P(w^0) - P^*\right)}{2\gamma n + 2m\gamma s} \tag{11}$$

Remark 1. *Theorem 1 shows that, if the objective function $P(w)$ is with the optimal strong convexity, AsyDSSKL achieves a linear convergence rate (see (11)). If the loss function $f_i(w)$ is general smooth convex, AsyDSSKL achieves a sublinear convergence rate (see (11)).*

Remark 2. *The thread number is not explicitly considered in Theorems 1. As discussed in (Liu and Wright 2015), the parameter τ is closely related to the number of threads that can be involved in the computation, without degrading the convergence performance of the algorithm. In other words, if the number of threads is small enough such that (10) holds, the convergence expressions (11), (11) do not depend on the number of threads, implying that linear speedup can be expected.*

Non-Convex Setting If the function $P(w)$ is non-convex, the global optimum point cannot be guaranteed. Thus, the closeness to the optimal solution (i.e., $P(w) - P^*$ and $\|x - \mathcal{P}_S(w)\|$) cannot be used for the convergence analysis. To analyze the convergence rate of AsyDSSKL in the non-convex setting, we define the expectation of a subgradient $\xi \in \partial P(w_t^s)$ as $\mathbb{E}\widetilde{\nabla}P(w_t^s)$. Specifically, $\mathbb{E}\widetilde{\nabla}P(w_t^s)$ can be written as following.

$$\mathbb{E}\widetilde{\nabla}P(w_t^s) \stackrel{\text{def}}{=} \frac{L_{\max}}{\gamma}\left(w_t^s - \overline{w}_{t+1}^s\right) \tag{12}$$

where $\overline{w}_{t+1}^s \stackrel{\text{def}}{=} \mathcal{P}_{\frac{\gamma}{L_{\max}}g}\left(w_t^s - \frac{\gamma}{L_{\max}}\widehat{v}_t^s\right)$. Based on (12), it is easy to verify that $(\overline{w}_{t+1}^s)_{j(t)} = (w_{t+1}^{s+1})_{j(t)}$. Thus, we have $\mathbb{E}_{j(t)}(w_{t+1}^s - w_t^s) = \frac{1}{n}\left(\overline{w}_{t+1}^s - w_t^s\right)$, which means that $\overline{w}_{t+1}^s - w_t^s$ captures the expectation of $w_{t+1}^s - w_t^s$. It is easy to verify that $\mathbb{E}\widetilde{\nabla}P(w_t^s)$ is equal to $\mathbf{0}$ when AsyDSSKL approaches to a stationary point.

Based on $\mathbb{E}\widetilde{\nabla}P(w_t^s)$, we give the convergence rate at the non-convex setting in Theorem 2.

Theorem 2. *Let ρ be a constant that satisfies $\rho > 1$, and define the quantities $\theta_1 = \frac{\rho^{\frac{1}{2}} - \rho^{\frac{\tau+1}{2}}}{1-\rho^{\frac{1}{2}}}$ and $\theta_2 = \frac{\rho^{\frac{1}{2}} - \rho^{\frac{m}{2}}}{1-\rho^{\frac{1}{2}}}$. Suppose the nonnegative steplength parameter $\gamma > 0$ satisfies $\gamma \leq \min\left\{\frac{n^{1/2}(1-\rho^{-1})-4}{4(\Lambda_{res}(1+\theta_1)+\Lambda_{nor}(1+\theta_2))}, \frac{n^{1/2}}{\frac{1}{2}n^{1/2}+2\Lambda_{nor}\theta_2+\Lambda_{res}\theta_1}\right\}$. Let T denote the number of total iterations of AsyDSSKL. If $f_i(w)$ is a smooth non-convex function with Assumption 3, we have*

$$\frac{1}{T}\sum_{s=0}^{S-1}\sum_{t=0}^{m-1}\mathbb{E}\left\|\widetilde{\nabla}P(w_t^s)\right\|^2 \leq \frac{P(w^0) - P^*}{T\left(\frac{\gamma}{n}\left(\frac{1}{\gamma} - \frac{1}{2} - \frac{2L_{nor}\theta_2 + L_{res}\theta_1}{n^{1/2}L_{\max}}\right)\right)} \tag{13}$$

Remark 3. *Theorem 2 shows that, if the loss function $f_i(w)$ is non-convex, AsyDSSKL converges to a stationary point with a sublinear convergence rate.*

Experimental Results

Experimental Setup

Design of Experiments: To demonstrate the superiority of our AsyDSSKL on the efficiency at the training step, we compare the accuracies (or errors) on the testing set vs. training time of the different kernel methods. To demonstrate the superiority of our AsyDSSKL on the efficiency of predicting, we compare the testing time vs. the size of testing samples of the different kernel methods. The state-of-the-art kernel methods compared in the experiments are LIBSVM(P), AsyGCD

and DoubleSGD which are summarized in Table 2. Note that, LIBSVM(P) is an modified implementation of LIBSVM with parallelly computing a column of kernel matrix (Chang and Lin 2011b). To show a near-linear speedup obtained by asynchronous parallel computation, we also test the speedup of our AsyDSSKL on different datasets.

Table 2: The state-of-the-art kernel methods compared in our experiments. (BC=binary classification, R=regression, Asy=asynchronous, DS=double stochastic, SC=sparse constraint)

Algorithm	Reference	Problem	Parallel	Asy	DS	SC
LIBSVM(P)	(Chang and Lin 2011b)	BC+R	Yes	No	No	No
AsyGCD	(You et al. 2016)	BC	Yes	Yes	No	No
DoubleSGD	(Dai et al. 2014)	BC+R	No	No	Yes	No
AsyDSSKL	Our	BC+R	Yes	Yes	Yes	Yes

Implementation Details: Our experiments are performed on a 32-core two-socket Intel Xeon E5-2699 machine where each socket has 16 cores. We implement our AsyDSSKL in C++, where the shared memory parallel computation is handled via OpenMP (Chandra 2001). We implement LIBSVM(P) by modifying LIBSVM with OpenMP according to the instruction[1] provided by Chang and Lin (2011b). We use the implementation[2] provided by (You et al. 2016) for AsyGCD. We use the implementation[3] provided by (Dai et al. 2014) for DoubleSGD. Note that the implementation of DoubleSGD only works for binary classification because only the logistic loss was implemented in their code.

In the experiments, we consider the binary classification and regression problems. Specifically, our AsyDSSKL uses the logistic loss (see Table 1) for binary classification, and the correntropy induced loss (see Table 1) for regression. We use the accuracy as the measure criterion of binary classification, and use the mean squared error (MSE) as the measure criterion of regression. In each experiment, the accuracy and MSE values are the average over 5 trials. In the experiments, the value of steplength γ is selected from $\{10^2; 10; 1; 10^-1; 10^-2; 10^-3; 10^-4; 10^-5\}$. The $\#$ of inner loop iterations m is set as the size of training set, and the $\#$ of outer loop iterations S is set as 10.

Datasets: Table 3 summarizes the six large-scale real-world datasets used in our experiments. They are the Covtype_B, RCV1, SUSY, Covtype_M, MNIST and Aloi datasets which are from https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/. Covtype_B and Covtype_M are from a same source. Covtype_M, MNIST and Aloi are originally for multi-class classification. In the experiments, we treat the multi-class classification as regression problem.

[1]The instruction to implement LIBSVM(P) is available at http://www.csie.ntu.edu.tw/~cjlin/libsvm/faq.html\#f432.

[2]The AsyGCD code is available at https://github.com/cjhsieh/asyn_kernel_svm.

[3]The DoubleSGD code is available at https://github.com/zixu1986/Doubly_Stochastic_Gradients.

Table 3: The large-scale dasetsets used in the experiments, where the multi-class classification datasets are treated as regression problem.

Task	Dataset	Features	Samples
Binary classification	Covtype_B	54	581,012
	RCV1	47,236	677,399
	SUSY	18	5,000,000
Regression	Covtype_M	54	581,012
	MNIST	784	1,000,000
	Aloi	128	108,000

Results and Discussions

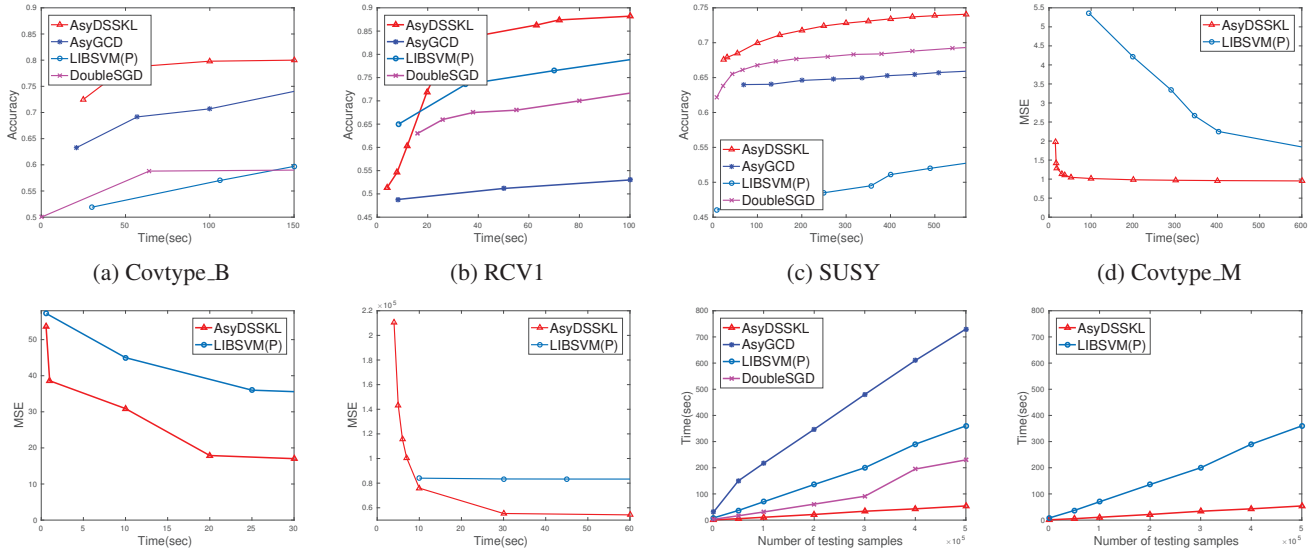
In the experiments of comparing the training time of different algorithms, AsyGCD, LIBSVM(P) and our AsyDSSKL is running on 16 cores. Figures 1a-1c provide the results of accuracy vs. training time on the Covtype_B, RCV1, and SUSY datasets, respectively. The results show that AsyDSSKL achieves the best accuracy value in the most time for a fixed training time. Especially, AsyDSSKL can converge to a good accuracy with a tremendous little time, mostly less than 60 seconds for large datasets. Figures 1d-1f provide the results of MSE vs. training time on the Covtype_M, MNIST, and Aloi datasets, respectively. The results show that AsyDSSKL always achieves the smallest MSE value for a fixed training time, and AsyDSSKL can converge to a good value with a tremendous little time, mostly less than 60 seconds for large datasets. Figures 1a-1f confirm that our AsyDSSKL has the significant superiority on the scalability and efficiency at the training step over the state-of-the-art kernel methods.

Figures 1g-1h present the testing time vs. the number of testing samples on the datasets of Covtype_B and Covtype_M. The results show that the testing time of our AsyDSSKL is smallest, compared with the other methods without sparse constraint. Figures 1g-1h confirm that our AsyDSSKL has the significant superiority on the scalability and efficiency at the predicting step over the existing kernel methods.

We perform AsyDSSKL on 1, 2, 4, 8 and 16 cores to observe the speedup. Figure 2 presents the speedup results of AsyDSSKL on the Covtype_B and Covtype_M datasets. The results show that AsyDSSKL can have a near-linear speedup on a parallel system with shared memory. This is because we do not use any lock in the implementation of AsyDSSKL.

Conclusion

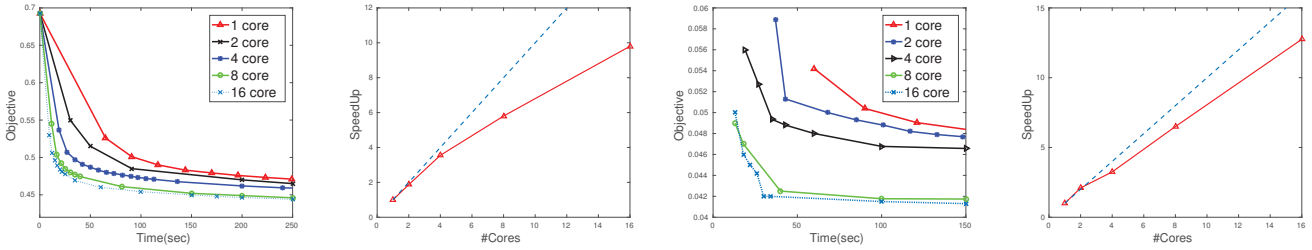
In this paper, we introduced a general sparse kernel learning formulation with a new asynchronous parallel doubly stochastic algorithm for sparse kernel learning (AsyDSSKL). We provided a comprehensive convergence guarantee to AsyDSSKL: 1) AsyDSSKL achieves a sublinear rate when the smooth loss functions are convex or non-convex. 2) AsyDSSKL achieves a linear (geometric) convergence rate when the objective function is with the optimal strong convexity. Experimental results show that our AsyDSSKL method has the significant superiority on the computational efficiency at the *training and predicting* steps over the existing kernel methods.



(a) Covtype_B (b) RCV1 (c) SUSY (d) Covtype_M

(e) MNIST (f) Aloi (g) Covtype_B (h) Covtype_M

Figure 1: Comparisons between AsyDSSKL and other state-of-the-art kernel methods. (a-c) Accuracy vs. training time. (d-f) MSE vs. training time. (g-h) Testing time vs. testing samples.



(a) Objective vs. time (b) Speedup (c) Objective vs. time (d) Speedup

Figure 2: Speedup results of AsyDSSKL. (a-b) Covtype_B dataset. (c-d) Covtype_M dataset.

Acknowledgement

This work was partially supported by the following grants: NSF-IIS 1302675, NSF-IIS 1344152, NSF-DBI 1356628, NSF-IIS 1619308, NSF-IIS 1633753, NIH R01 AG049371.

References

Baudat, G., and Anouar, F. 2000. Generalized discriminant analysis using a kernel approach. *Neural computation* 12(10):2385–2404.

Camps-Valls, G., and Bruzzone, L. 2009. *Kernel methods for remote sensing data analysis*. John Wiley & Sons.

Chandra, R. 2001. *Parallel programming in OpenMP*. Morgan kaufmann.

Chang, C.-C., and Lin, C.-J. 2011a. Libsvm: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)* 2(3):27.

Chang, C.-C., and Lin, C.-J. 2011b. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology* 2:27:1–27:27. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

Dai, B.; Xie, B.; He, N.; Liang, Y.; Raj, A.; Balcan, M.-F. F.; and Song, L. 2014. Scalable kernel methods via doubly stochastic gradients. In *Advances in Neural Information Processing Systems*, 3041–3049.

Feng, Y.; Huang, X.; Shi, L.; Yang, Y.; and Suykens, J. A. 2015. Learning with the maximum correntropy criterion induced losses for regression. *Journal of Machine Learning Research* 16:993–1034.

Kivinen, J.; Smola, A. J.; and Williamson, R. C. 2004. Online learning with kernels. *IEEE transactions on signal processing* 52(8):2165–2176.

Li, F.; Yang, Y.; and Xing, E. 2005. From lasso regression to feature vector machine. In *NIPS*, 779–786.

Lin, Y. 2004. A note on margin-based loss functions in classification. *Statistics & probability letters* 68(1):73–82.

Liu, J., and Wright, S. J. 2015. Asynchronous stochastic

coordinate descent: Parallelism and convergence properties. *SIAM Journal on Optimization* 25(1):351–376.

Mercer, J. 1909. Functions of positive and negative type, and their connection with the theory of integral equations. *Philosophical transactions of the royal society of London. Series A, containing papers of a mathematical or physical character* 209:415–446.

Rahimi, A., and Recht, B. 2007. Random features for large-scale kernel machines. In *Advances in neural information processing systems*, 1177–1184.

Rahimi, A., and Recht, B. 2009. Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning. In *Advances in neural information processing systems*, 1313–1320.

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

Schölkopf, B.; Tsuda, K.; and Vert, J.-P. 2004. *Kernel methods in computational biology*. MIT press.

Shalev-Shwartz, S., and Zhang, T. 2013. Stochastic dual coordinate ascent methods for regularized loss minimization. *Journal of Machine Learning Research* 14(Feb):567–599.

Shalev-Shwartz, S.; Singer, Y.; and Srebro, N. 2007. Pegasos: Primal estimated sub-gradient solver for svm. In *Proceedings of the 24th international conference on Machine learning*, 807–814. ACM.

Shamir, O., and Zhang, T. 2013. Stochastic gradient descent for non-smooth optimization: Convergence results and optimal averaging schemes. In *International Conference on Machine Learning*, 71–79.

Steinwart, I., and Christmann, A. 2008. *Support vector machines*. Springer Science & Business Media.

Takahashi, N., and Nishi, T. 2006. Global convergence of decomposition learning methods for support vector machines. *IEEE Transactions on Neural Networks* 17(6):1362–1369.

Tseng, P., and Yun, S. 2009. A coordinate gradient descent method for nonsmooth separable minimization. *Mathematical Programming* 117(1-2):387–423.

Vapnik, V. N. 1998. *Statistical learning theory*, volume 1. Wiley New York.

Vovk, V. 2013. Kernel ridge regression. In *Empirical inference*. Springer. 105–116.

Yen, I. E.-H.; Lin, T.-W.; Lin, S.-D.; Ravikumar, P. K.; and Dhillon, I. S. 2014. Sparse random feature algorithm as coordinate descent in hilbert space. In *Advances in Neural Information Processing Systems*, 2456–2464.

You, Y.; Fu, H.; Song, S. L.; Randles, A.; Kerbyson, D.; Marquez, A.; Yang, G.; and Hoisie, A. 2015. Scaling support vector machines on modern hpc platforms. *Journal of Parallel and Distributed Computing* 76:16–31.

You, Y.; Lian, X.; Liu, J.; Yu, H.-F.; Dhillon, I. S.; Demmel, J.; and Hsieh, C.-J. 2016. Asynchronous parallel greedy coordinate descent. In *Advances in Neural Information Processing Systems*, 4682–4690.

Zhang, T. 2004. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *Proceedings of the twenty-first international conference on Machine learning*, 116. ACM.

Zhao, H.-x., and Magoules, F. 2011. Parallel support vector machines on multi-core and multiprocessor systems. In *11th International Conference on Artificial Intelligence and Applications (AIA 2011)*. IASTED.

Zhu, J., and Hastie, T. 2005. Kernel logistic regression and the import vector machine. *Journal of Computational and Graphical Statistics* 14(1):185–205.

Zhu, J.; Rosset, S.; Tibshirani, R.; and Hastie, T. J. 2004. 1-norm support vector machines. In *Advances in neural information processing systems*, 49–56.