Communication-Efficient Distributed Block Minimization for Nonlinear Kernel Machines*

Cho-Jui Hsieh
University of California, Davis
Davis, CA 95616
chohsieh@ucdavis.edu

Si Si†
Google Research
Mountain View, CA 94043
sisidaisy@google.com

Inderjit S. Dhillon
University of Texas at Austin
Austin, TX 78712
inderjit@cs.utexas.edu

ABSTRACT

Nonlinear kernel machines often yield superior predictive performance on various tasks; however, they suffer from severe computational challenges. In this paper, we show how to overcome the important challenge of speeding up kernel machines using multiple computers. In particular, we develop a parallel block minimization framework, and demonstrate its good scalability in solving nonlinear kernel SVM and logistic regression. Our framework proceeds by dividing the problem into smaller subproblems by forming a block-diagonal approximation of the Hessian matrix. The subproblems are then solved approximately in parallel. After that, a communication efficient line search procedure is developed to ensure sufficient reduction of the objective function value by exploiting the problem structure of kernel machines. We prove global linear convergence rate of the proposed method with a wide class of subproblem solvers, and our analysis covers strongly convex and some non-strongly convex functions. We apply our algorithm to solve large-scale kernel SVM problems on distributed systems, and show a significant improvement over existing parallel solvers. As an example, on the covtype dataset with half-a-million samples, our algorithm can obtain an approximate solution with 96% accuracy in 20 seconds using 32 machines, while all the other parallel kernel SVM solvers require more than 2000 seconds to achieve a solution with 95% accuracy. Moreover, our algorithm is the first distributed kernel SVM solver that can scale to massive data sets. On the kddb dataset (20 million samples and 30 million features), our parallel solver can compute the kernel SVM solution within half an hour using 32 machines with 640 cores in total, while existing solvers can not scale to this dataset.

KEYWORDS

Distributed Algorithm, Kernel Methods, Classification.

*The code is available in https://github.com/cjhsieh/Distributed-KSVM.
†Si Si is now working at Google Research; this work was done before she joined Google.

KDD '17, August 13-17, 2017, Halifax, NS, Canada

ACM ISBN 978-1-4503-4887-4/17/08…$15.00
https://doi.org/10.1145/3097983.3098080

1 INTRODUCTION

Kernel methods are a class of algorithms that map samples from input space to a high-dimensional feature space. The representative kernel machines include kernel SVM, kernel logistic regression and support vector regression. When a nonlinear kernel is used, all the above algorithms are hard to scale to large-scale problems due to the high computation cost and memory requirement of computing and storing the kernel matrix. Efficient sequential algorithms have been proposed for kernel machines, where the most widely-used one is the SMO algorithm [10, 18] implemented in LIBSVM and SVMLight. However, single-machine kernel SVM algorithms still cannot scale to larger datasets (e.g. LIBSVM takes 3 days on the MNIST dataset with 8 million samples), so there is a tremendous need for developing distributed algorithms for kernel machines.

Although many distributed algorithms have been proposed for solving linear SVM/logistic regression [9, 14, 34], they cannot be applied to nonlinear kernel machines since they synchronize information using primal variables. To the best of our knowledge, there are very few existing algorithms for distributed nonlinear kernel SVM training because of two challenges: How to deal with the huge kernel matrix and how to solve the large quadratic optimization problem in a distributed and communication-efficient manner. Most existing algorithms try to approximate kernel matrix in parallel to reduce the problem to a linear machine [3, 19], and then solve the resulting problem by a distributed optimization solver. However, due to the kernel approximation step, they cannot obtain the exact kernel SVM solution and this often leads to suboptimal prediction accuracy.

In this paper, we propose a distributed kernel SVM solver that is able to solve the quadratic optimization problem exactly by a Parallel Block Minimization (PBM) framework. At each iteration, PBM divides the whole problem into smaller subproblems by first forming a block-diagonal approximation of the kernel matrix. Each subproblem is much smaller than the original problem and they can be solved in parallel using existing serial solvers such as SMO (or greedy coordinate descent) [18] implemented in LIBSVM. After that, we develop a communication-efficient line search procedure to ensure a reduction of the objective function value.

Our contribution can be summarized below:

1. The paper is the first to apply the parallel block minimization (PBM) framework for training kernel machines, where each machine updates a block of dual variables. We develop an efficient line search procedure to synchronize the updates at the end of each iteration without any additional cost by exploiting the problem structure of kernel machines.

2. Our algorithm works for any partition of dual variables, and we show that the convergence speed can be significantly improved using a better partition (obtained by kmeans) .
3. We show that our proposed algorithm significantly outperforms existing kernel SVM and logistic regression solvers on a distributed system. In particular, our algorithm is the first distributed nonlinear SVM solver that scales to kddb dataset with 20 million samples and 30 million features. On this dataset, our parallel solver only takes half an hour to compute the kernel SVM solution with 32 machines containing 640 cores in total.
4. We prove a global linear convergence rate for PBM under mild conditions: we allow a wide range of inexact subproblem solvers, and our analysis covers many widely-used functions where some of them may not be strongly convex (e.g., SVM dual problem with a positive semi-definite kernel).

The rest of the paper is organized as follows. We discuss the related work in Section 2. In Section 3, we introduce the problem setting, and the proposed PBM framework is presented in Section 4. We prove a global linear convergence rate in Section 5. Experimental results are shown in Section 6, while Section 7 concludes the paper.

2 RELATED WORK

Algorithms for solving kernel machines. Several optimization algorithms have been proposed to solve kernel machines [10, 18]. Among them, decomposition methods [10, 11] have become widely-used in software packages such as LIBSVM [2] and SVMLight.

Parallelizing kernel SVM has also been studied in the literature. Recently an asynchronous greedy coordinate descent algorithm has been proposed for solving kernel SVM [30]. Although it achieves good performance on a multicore shared memory system, the algorithm cannot be used in distributed systems due to the need of asynchronous global parameter access. Several distributed SVM algorithms have been proposed: Cascade-SVM [4] proposed to randomly divide the problem into subproblems, but they still need to solve a global kernel SVM problem in the final stage; PSVM [3] uses an incomplete Cholesky factorization to approximate the original kernel SVM problem, and solves the resulting problem by an interior point method; P-pack SVM [35] applies a distributed SGD algorithm for training kernel SVM. Unfortunately, as shown in the experiments, none of the existing distributed kernel SVM algorithms can scale to extremely large datasets.

Two clustering based approaches for kernel SVM have been recently developed: Divide-and-conquer SVM (DC-SVM) [5] and Communication Avoiding SVM (CA-SVM) [29]. In DC-SVM, the global kernel SVM problem is hierarchically divided into smaller subproblems by kmeans or kernel kmeans, and the subproblems can be solved independently. The solutions from subproblems can be used directly for prediction (called DC-SVM early prediction) or can be used to initialize the global SVM solver. Although the subproblems can be solved in parallel, DC-SVM still needs a parallel solver for the global SVM problem in the final stage. Our idea of using clustering is similar to DC-SVM [5], but here we apply it to develop a distributed SVM solver, while DC-SVM is just a single-machine serial algorithm. The experimental comparisons between our proposed method and DC-SVM are shown in the experiments.

Kernel approximation approaches. Another line of approach for kernel machines aim to approximate the kernel matrix and then solve the reduced size problem. Nyström approximation [6, 22, 23, 27] and random Fourier features [19] can be used to form low-rank approximation of kernel matrices, and then the problem can be reduced to a linear SVM or logistic regression problem. To parallelize these algorithms, a distributed linear SVM or logistic regression solver is needed. The comparisons are in Figure 2. Using random Fourier features in kernel machine is also applied in [8] and [25]. Comparing with our proposed method, (1) they only consider solving linear systems, and it is nontrivial for them to solve kernel SVM and logistic regression problems; (2) they cannot obtain the exact solution of kernel machines, since they only use the approximate kernel.

Distributed linear SVM and logistic regression solvers. Several distributed algorithms have been proposed for solving the primal linear SVM problems [14, 15, 34], but they require synchronizing the primal variables at each iteration. Unfortunately, the primal variables cannot be explicitly formed in nonlinear kernel machines, so these methods cannot be directly applied to nonlinear kernel SVM or logistic regression. Distributed dual coordinate descent [9, 14, 28] and asynchronous dual coordinate descent [32] is also investigated for linear SVM.

Distributed Block Coordinate Descent. The main idea of our algorithm is similar to a class of Distributed Block Coordinate Descent (DBCD) parallel computing methods. As discussed recently in literature [17, 20, 21], different DBCD parallel algorithms have different ways to solve subproblems in each machine, and synchronize the results. However, no existing work has applied DBCD to kernel machines, partly due to the challenge of line search in distributed systems. Recent distributed dual coordinate descent linear SVM solvers [14, 34] also belongs to DBCD class.

Our Innovations. In contrast to the above algorithms, ours is the first one that adopts a block-partition based coordinate descent algorithm to kernel machines; previous work either focused on linear SVM [9, 14, 28] or primal ERM problems [17]. These approaches proposed to maintain and synchronize the vector $\boldsymbol{y} = X^T\boldsymbol{d}$ or $\boldsymbol{w} = X\boldsymbol{d}$, where $\boldsymbol{d}$ is the direction and X is the data matrix. Unfortunately, in kernel methods this strategy does not work since each sample may have infinite dimension after the nonlinear mapping. We overcome the challenges by synchronizing the $Q\boldsymbol{d}$ vector ($Q \in \mathbb{R}^{n\times n}$) and developing an efficient line search procedure. We further show that a better partition (obtained by kmeans) can speedup the convergence of the algorithm.

On the theoretical front: previous papers [9, 28] can only show linear convergence when the objective function is $f(x) = g(x) + h(x)$ and g is strongly convex. [14] considered some non-strongly convex functions, but they assume each subproblem is solved exactly. In this paper, we prove a global linear convergence even when g is not strongly convex and each subproblem is solved approximately. Our proof covers general DBCD algorithms for some non-strongly convex functions, for which previous analysis can only show sub-linear convergence.

3 PROBLEM SETUP

We focus on the following composite optimization problem:

$$\underset{\boldsymbol{\alpha}\in\mathbb{R}^n}{\operatorname{argmin}}\Big\{\boldsymbol{\alpha}^T Q\boldsymbol{\alpha} + \sum_i g_i(\alpha_i)\Big\} := f(\boldsymbol{\alpha}) \text{ s.t. } \boldsymbol{a}\le\boldsymbol{\alpha}\le\boldsymbol{b}, \tag{1}$$

where $Q \in \mathbb{R}^{n\times n}$ is positive semi-definite; each g_i is a univariate convex function; $\boldsymbol{a} = [a_1, \cdots, a_n]$ and $\boldsymbol{b} = [b_1, \cdots, b_n]$ are vectors. Note that we can easily handle the box constraint $\boldsymbol{a} \le \boldsymbol{\alpha} \le \boldsymbol{b}$ by setting $g_i(\alpha_i) = \infty$ if $\alpha_i \notin [a_i, b_i]$, so we will omit the constraint in most parts of the paper.

An important application of (1) in machine learning is that it is the dual problem of ℓ_2-regularized empirical risk minimization. Given a set of instances $\{\boldsymbol{x}_i, y_i\}_{i=1}^n$, we consider the following ℓ_2-regularized empirical risk minimization problem:

$$\arg\min_{\boldsymbol{w}} \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^n \ell_i(\boldsymbol{w}^T\Phi(\boldsymbol{x}_i)), \tag{2}$$

where ℓ_i is the loss function depending on the label y_i, and $\Phi(\cdot)$ is the nonlinear feature mapping. For example, $\ell_i(u) = \max(0, 1-y_i u)$ for SVM with hinge loss. The dual problem of (2) can be written as

$$\arg\min_{\boldsymbol{\alpha}} \frac{1}{2}\boldsymbol{\alpha}^T Q\boldsymbol{\alpha} + \sum_{i=1}^n \ell_i(-\alpha_i), \tag{3}$$

where $Q \in \mathbb{R}^{n\times n}$ in this case is the kernel matrix with $Q_{ij} = y_i y_j \Phi(\boldsymbol{x}_i)^T\Phi(\boldsymbol{x}_j)$. Our proposed approach works in the general setting, but we will discuss in more detail its applications to kernel SVM, where Q is the kernel matrix and $\boldsymbol{\alpha}$ is the vector of dual variables. Note that, similar to [5], we ignore the bias term in (2). Indeed, in our experimental results we did not observe improvement in test accuracy by adding the bias term.

4 PROPOSED ALGORITHM

We describe our proposed framework PBM for solving (1) on a distributed system with k worker machines. We partition the variables $\boldsymbol{\alpha}$ into k disjoint index sets $\{S_r\}_{r=1}^k$ such that

$$S_1 \cup S_2 \cup \cdots \cup S_k = \{1, \ldots, n\} \quad \text{and} \quad S_p \cap S_q = \phi \ \forall p \ne q,$$

and we use $\pi(i)$ to denote the cluster indicator that i belongs to. We associate each worker r with a subset of variables $\boldsymbol{\alpha}_{S_r} := \{\alpha_i \mid i \in S_r\}$. Note that our framework allows any partition, and we will discuss how to obtain a better partition in Section 4.3.

At each iteration, we form the quadratic approximation of problem (1) around the current solution:

$$\begin{aligned} f(\boldsymbol{\alpha}+\Delta\boldsymbol{\alpha}) \approx \bar{f}_{\boldsymbol{\alpha}}(\Delta\boldsymbol{\alpha}) = &\frac{1}{2}\boldsymbol{\alpha}^T Q\boldsymbol{\alpha} + \boldsymbol{\alpha}^T Q\Delta\boldsymbol{\alpha} + \frac{1}{2}\Delta\boldsymbol{\alpha}^T\bar{Q}\Delta\boldsymbol{\alpha} \\ &+ \sum_i g_i(\alpha_i + \Delta\alpha_i), \end{aligned} \tag{4}$$

where the second order term of the quadratic part ($\frac{1}{2}\Delta\boldsymbol{\alpha}^T Q\Delta\boldsymbol{\alpha}$) is replaced by $\frac{1}{2}\Delta\boldsymbol{\alpha}^T\bar{Q}\Delta\boldsymbol{\alpha}$, and $\bar{Q}$ is the block-diagonal approximation of Q such that

$$\bar{Q}_{ij} = \begin{cases} Q_{ij} & \text{if } \pi(i) = \pi(j) \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

By solving (4), we obtain the descent direction $\boldsymbol{d}$:

$$\boldsymbol{d} := \arg\min_{\Delta\boldsymbol{\alpha}} \bar{f}_{\boldsymbol{\alpha}}(\Delta\boldsymbol{\alpha}). \tag{6}$$

Since $\bar{Q}$ is block-diagonal, problem (6) can be decomposed into k independent subproblems based on the partition $\{S_r\}_{r=1}^k$:

$$\boldsymbol{d}_{S_r} = \underset{\Delta\boldsymbol{\alpha}_{S_r}}{\operatorname{argmin}}\Big\{\frac{1}{2}\Delta\boldsymbol{\alpha}_{S_r}^T Q_{S_r,S_r}\Delta\boldsymbol{\alpha}_{S_r} + \sum_{i\in S_r}\bar{g}_i(\Delta\alpha_i)\Big\} := f_{\boldsymbol{\alpha}}^{(r)}(\Delta\boldsymbol{\alpha}_{S_r}), \tag{7}$$

where $\bar{g}_i(\Delta\alpha_i) = g_i(\alpha_i + \Delta\alpha_i) + (Q\boldsymbol{\alpha})_i\Delta\alpha_i$. Subproblem (7) has the same form as the original problem (1), so can be solved by any existing kernel SVM/kernel regression solver on each machine. After solving the subproblems (7) independently for each r, the descent direction $\boldsymbol{d}$ is the concatenation of $\boldsymbol{d}_{S_1}, \ldots \boldsymbol{d}_{S_r}$. Since $f(\boldsymbol{\alpha}+\boldsymbol{d})$ might even increase the objective function value $f(\boldsymbol{\alpha})$, we find the step size β to ensure the following sufficient decrease condition of the objective function value:

$$f(\boldsymbol{\alpha}+\beta\boldsymbol{d}) - f(\boldsymbol{\alpha}) \le \beta\sigma\Delta, \tag{8}$$

where $\Delta = \nabla f(\boldsymbol{\alpha})^T\boldsymbol{d}$, and $\sigma \in (0, 1)$ is a constant. We then update $\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha} + \beta\boldsymbol{d}$. Now we discuss details of each step of our algorithm.

4.1 Solving the Subproblems

Note that subproblem (7) has the same form as the original problem (1), so we can apply any existing algorithm to solve it for each worker independently. In our implementation, we apply the following greedy coordinate descent method (a similar algorithm was used in LIBSVM). Assume the current subproblem solution is $\Delta\boldsymbol{\alpha}_{S_r}$, we choose variable with the largest projected gradient:

$$i^* := \underset{i\in S_r}{\operatorname{argmax}}\Big|\Pi\Big(\alpha_i + \Delta\alpha_i - (Q_{S_r,S_r}\Delta\boldsymbol{\alpha}_{S_r})_i - \bar{g}_i'(\Delta\alpha_i)\Big) - \alpha_i - \Delta\alpha_i\Big| \tag{9}$$

where Π is the projection to the interval $[a_i, b_i]$. The selection only requires $O(|S_r|)$ time if $Q_{S_r,S_r}\Delta\boldsymbol{\alpha}_{S_r}$ is maintained in local memory. Variable $\Delta\alpha_i$ is then updated by solving the following one-variable subproblem:

$$\begin{aligned} \delta^* = \underset{\delta}{\operatorname{argmin}}\ &\frac{1}{2}Q_{i^*,i^*}\delta^2 + (Q_{S_r,S_r}\Delta\boldsymbol{\alpha}_{S_r})_{i^*}\delta + \bar{g}_{i^*}(\Delta\alpha_{i^*} + \delta) \\ &\text{with} \quad a_{i^*} \le \alpha_{i^*} + \Delta\alpha_{i^*} + \delta \le b_{i^*}. \end{aligned} \tag{10}$$

For kernel SVM, the one-variable subproblem (10) has a closed form solution, while for logistic regression the subproblem can be solved by Newton's method (see [31] for the detailed discussions). The bottleneck of both (9) and (10) is to compute $Q_{S_r,S_r}\Delta\boldsymbol{\alpha}_{S_r}$, which can be maintained after each update using $O(|S_r|)$ time.

Note that in our framework each subproblem does not need to be solved exactly. In Section 5 we will give theoretical analysis to the in-exact PBM method, and show that the linear convergence is guaranteed when each subproblem runs more than 1 coordinate update.

Communication Cost. There is no communication needed for solving the subproblems between workers; however, after solving the subproblems and obtaining the update direction $\boldsymbol{d}$, each worker needs to obtain the updated $(Q\boldsymbol{d})_{S_r}$ vector for next iteration, where this term in (7) is the only "global information" needed for each machine. Since each worker only has the local update $\boldsymbol{d}_{S_r}$, we compute $Q_{:,S_r}(\boldsymbol{d}_{S_r})$ in each worker, and use a REDUCE_SCATTER collective communication to obtain updated $(Q\boldsymbol{d})_{S_r}$ for each worker. The

communication cost for the collective REDUCE_SCATTER operation for an n-dimensional vector requires

$$\log(k)T_{\text{initial}} + nT_{\text{byte}} \tag{11}$$

communication time, where T_{initial} is the message startup time and T_{byte} is the transmission time per byte (see Section 6.3 of [1]). When n is large, the second term usually dominates, so we need $O(n)$ communication time and this *does not grow with the number of processors.*

4.2 Communication-efficient Line Search

After obtaining $(Q\boldsymbol{d})_{S_r}$ for each worker, we propose the following efficient line search approaches.

Armijo-rule based step size selection. For general $g_i(\cdot)$, a commonly used line search approach is to try step sizes $\beta \in \{1, \frac{1}{2}, \dots\}$ until β satisfies the sufficient decrease condition (8). The only cost is to evaluate the objective function value. For each choice of β, $f(\boldsymbol{\alpha} + \beta\boldsymbol{d})$ can be computed as

$$f(\boldsymbol{\alpha} + \beta\boldsymbol{d}) = f(\boldsymbol{\alpha}) + \sum_r \{\beta \boldsymbol{d}_{S_r}^T (Q\boldsymbol{\alpha})_{S_r} + \frac{1}{2}\beta^2 \boldsymbol{d}_{S_r}^T (Q\boldsymbol{d})_{S_r} + \sum_{i \in S_r} g_i(\alpha_i + \beta d_i) - g_i(\alpha_i)\},$$

so if each worker r has the vector $(Q\boldsymbol{d})_{S_r}$, we can compute $f(\boldsymbol{\alpha}+\beta\boldsymbol{d})$ using $O(n/k)$ time and $O(1)$ communication cost. In order to prove convergence, we add another condition that $f(\boldsymbol{\alpha} + \beta\boldsymbol{d}) \le f(\boldsymbol{\alpha} + \boldsymbol{d}/k)$, although in practice we do not find any difference without adding this condition.

Optimal step size selection. If each g_i is a linear function with bounded constraint (such as for the kernel SVM case), the optimal step size can be computed without communication. The optimal step size is defined by

$$\beta^* := \arg\min_{\beta} f(\boldsymbol{\alpha} + \beta\boldsymbol{d}) \text{ s.t. } \boldsymbol{a} \le \boldsymbol{\alpha} + \beta\boldsymbol{d} \le \boldsymbol{b}. \tag{12}$$

If $\sum_i g_i(\alpha_i) = \boldsymbol{p}\boldsymbol{\alpha}$, then $f(\boldsymbol{\alpha} + \beta\boldsymbol{d})$ with respect to β is a univariate quadratic function, and thus β_t^* can be obtained by the following closed form solution:

$$\beta = \min(\bar{\eta}, \max(\underline{\eta}, -(\boldsymbol{\alpha}^T Q\boldsymbol{d} + \boldsymbol{p}^T\boldsymbol{d})/(\boldsymbol{d}^T Q\boldsymbol{d}))), \tag{13}$$

where $\bar{\eta} := \min_{i=1}^n (b_i - \alpha_i)$ and $\underline{\eta} := \max_{i=1}^n (a_i - \alpha_i)$. This can also be computed in $O(n/k)$ time and $O(1)$ communication time. Our proposed algorithm is summarized in Algorithm 1.

4.3 Variable Partitioning

Our PBM algorithm converges for any choice of partition $\{S_r\}_{r=1}^k$. However, it is important to select a good partition in order to achieve faster convergence. Since our algorithm is solving an "approximate function" at each iteration, the difference between the approximate function $\bar{f}_{\boldsymbol{\alpha}}$ and the true function f controls the convergence rate of the algorithm. If $\bar{Q} = Q$ in subproblem (4) (no approximation error), then $\bar{f}_{\boldsymbol{\alpha}}(\Delta\boldsymbol{\alpha}) = f(\boldsymbol{\alpha} + \Delta\boldsymbol{\alpha})$ for any $\Delta\boldsymbol{\alpha}$, so our algorithm converges in one iteration. However, there is no parallelism in this case. On the other hand, if $\|Q - \bar{Q}\|$ is very large, the algorithm will converge slowly. Therefore, the convergence rate can be speed up

Algorithm 1: PBM: Parallel Block Minimization for solving (1)

Input : The objective function (1), initial $\boldsymbol{\alpha}_0$.
Output: The solution $\boldsymbol{\alpha}^*$.

1 Obtain a disjoint index partition $\{S_r\}_{r=1}^k$.
2 Compute $Q\boldsymbol{\alpha}_0$ in parallel ($Q\boldsymbol{\alpha}_0 = \boldsymbol{0}$ if $\boldsymbol{\alpha}_0 = 0$).
3 **for** $t = 0, 1, \dots$ *(until convergence)* **do**
4 Obtain $\boldsymbol{d}_{S_r}$ by solving subproblems (7) **in parallel**.
5 Compute $Q_{:,S_r}\boldsymbol{d}_{S_r}$ **in parallel**.
6 Use REDUCE_SCATTER to obtain $(Q\boldsymbol{d})_{S_r}$ in each worker.
7 Obtain the step size β using line search (see Section 4.2 for details)
8 $\boldsymbol{\alpha}_{S_r} \leftarrow \boldsymbol{\alpha}_{S_r} + \beta\boldsymbol{d}_{S_r}$ and $(Q\boldsymbol{\alpha})_{S_r} \leftarrow (Q\boldsymbol{\alpha})_{S_r} + \beta(Q\boldsymbol{d})_{S_r}$ **in parallel**.

by finding a partition $\{S_r\}_{r=1}^k$ to minimize error

$$\|\bar{Q} - Q\|_F^2 = \sum_{i,j} Q_{ij}^2 - \sum_{r=1}^k \sum_{i,j \in S_r} Q_{ij}^2,$$

and the minimizer can be obtained by maximizing the second term. However, we also want to have a balanced partition in order to achieve better parallelization speedup.

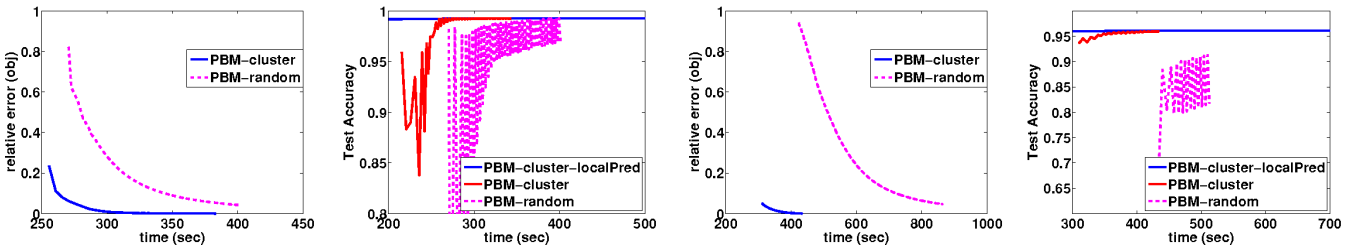
The same problem has been encountered in [5, 24] for forming a good kernel approximation. They have shown that kernel kmeans can achieve a good partition for general kernel matrix, and if the kernels are shift-invariant (e.g., Gaussian, Laplacian, …), we can use kmeans clustering on the data points to find a balanced partition with small kernel approximation error. Since we do not need the "optimal" partition, in practice we only run kmeans or kernel kmeans on a subset of 20000 training samples, so the overhead is very small compared with the whole training procedure.

We observe PBM with kmeans partition converges much faster compared to random partition. By random partition, we will assign each variable into a random partition. Due to randomness, each partition will roughly have the same number of variables. In Figure 1, we test the PBM algorithm on the kernel SVM problem with Gaussian kernel, and show that the convergence is much faster when the partition is obtained by kmeans clustering.

Local Prediction. We further propose a local prediction strategy for PBM when data is partitioned by kmeans clustering. Let $\boldsymbol{\alpha}_t$ be the solution before the t-th iteration of Algorithm 1, and $\boldsymbol{d}_t$ be the solution of the quadratic subproblem (4). The traditional way is to use $\boldsymbol{\alpha}_{t+1} = \boldsymbol{\alpha}_t + \beta_t\boldsymbol{d}_t$ for predicting new data.

However, we find the following procedure gives better prediction accuracy compared to using $\boldsymbol{\alpha}_{t+1}$: we first identify the cluster indicator of the test point $\boldsymbol{x}$ by choosing the nearest kmeans center. If $\boldsymbol{x}$ belongs to the r-th cluster, we then compute the prediction by the local model $\boldsymbol{\alpha}_t + (\boldsymbol{d}_t)_{<S_r>}$, where $(\boldsymbol{d}_t)_{<S_r>}$ is an n dimensional vector that sets all the elements outside S_r to be 0. Experimental results in Figure 1 show that this local prediction strategy is generally better than predicting by $\boldsymbol{\alpha}_{t+1}$. The main reason is that during the optimization procedure each local machine fits the local data by $\boldsymbol{\alpha}_t + (\boldsymbol{d}_t)_{<S_r>}$, so the prediction accuracy is better than the global model $(\boldsymbol{\alpha}_t + \boldsymbol{d}_t)$ which may not fit each local model that well.

Summary: Computational and Memory Cost: In Section 4.1, we showed that the greedy coordinate descent solver only requires



(a) webspam **obj** (b) webspam **accuracy** (c) covtype **obj** (d) covtype **accuracy**

Figure 1: Comparison of different variances of PBM. We observe PBM with kmeans partitioning converges faster than random partition, and the accuracy can further be improved by using our local prediction heuristic.

$O(|S_r|)$ time complexity per inner iteration. Before communication, we need to compute $Q_{:,S_r}\boldsymbol{d}_{S_r}$ in each machine, which requires $O(tn)$ time complexity, where t is number of inner iterations. The line search requires only $O(n/k)$ time. Therefore, the overall time complexity is $O(tn)$ for totally $O(tk)$ coordinate updates from all workers, so the average time per update is $O(n/k)$, which is k times faster than the original greedy coordinate descent (or SMO) algorithm for kernel SVM, where k is number of machines.

The time for running kmeans is $O(\bar{n}\bar{d})$ using k computers, where $\bar{n}$ is number subsamples (we set it to 20,000). This is a one time cost and is very small comparing to the cost of solving kernel machines. For example, on Covtype dataset the kmeans step only took 13 seconds, while the overall training time is 772 seconds. Note that **we include the clustering time in all the experimental results**.

Memory Cost: In kernel SVM, the main memory bottleneck is the memory needed for storing the kernel matrix. Without any trick (such as shrinking) to reduce the kernel storage, the space complexity is $O(n^2)$ for kernel SVM (otherwise we have to recompute kernel values when needed). Using our approach, (1) the subproblem solver only requires $O(n^2/k^2)$ space for the sub-matrix of kernel Q_{S_r,S_r} (2) Before synchronization, computing $Q_{:,S_r}\boldsymbol{d}_{S_r}$ requires $O(n^2/k)$ kernel entries. Therefore, the memory requirement will be reduced from n^2 to n^2/k using our algorithm. However, for large datasets the memory is still not enough. Therefore, similar to the LIBSVM software, we maintain a kernel cache to store columns of Q, and maintain the cache using the Least Recent Used (LRU) policy. In short, if memory is not enough, we use the kernel caching technique (implemented in LIBSVM) that computes the kernel values on-the-fly when it is not in the kernel cache and maintains recently used values in memory.

If each machine contains all the training samples, then $Q_{:,S_r}\boldsymbol{d}_{S_r}$ can be computed in parallel without any communication, and we only need one REDUCE_SCATTER mpi operation to gather the results. However, if each machine only contains a subset of samples, they have to broadcast $\{\boldsymbol{x}_i \mid i \in S_r, d_i \neq 0\}$ to other machines so that each machine q can compute $(Q\boldsymbol{d})_{S_q}$. In this case, we do not need to store the whole training data in each machine, and the communication time will be proportional to number of support vectors, which is usually much smaller than n.

5 CONVERGENCE ANALYSIS

In this section we show that PBM has a global linear convergence rate under certain mild conditions. Note that our result is stronger than some recent theoretical analysis of distributed coordinate descent. Compared to [9], we show linear convergence even when the objective function (1) is *not* strictly positive definite (e.g., for SVM with hinge loss), while [9] only has sub-linear convergence rate for those cases. In comparing to [14], they assume that the subproblems in each worker are solved exactly, while we allow an approximate subproblem solver (e.g., coordinate descent with ≥ 1 steps).

First we assume the objective function satisfies the following global error bound property. This is a weaker notion of strong convexity, in the sense that all the strongly convex functions satisfy this assumption (in those cases, κ is the condition number), but many other machine learning problems also satisfy this assumption even if they are not strongly convex [7, 26].

Definition 5.1 (Global error bound). Problem (1) admits a "global error bound" if there is a constant κ such that

$$\|\boldsymbol{\alpha} - P_S(\boldsymbol{\alpha})\| \leq \kappa \|T(\boldsymbol{\alpha}) - \boldsymbol{\alpha}\|, \tag{14}$$

where $P_S(\cdot)$ is the Euclidean projection to the set S of optimal solutions, and $T : \mathbb{R}^n \to \mathbb{R}^n$ is the operator defined by

$$T_i(\boldsymbol{\alpha}) = \arg\min_u f(\boldsymbol{\alpha} + (u - \alpha_i)\boldsymbol{e}_i), \quad \forall i = 1, \ldots, n. \tag{15}$$

where $\boldsymbol{e}_i$ is the standard i-th unit vector and u is the variable to minimize. We say that the algorithm satisfies a global error bound from the beginning if (14) holds for the level set $\{\boldsymbol{\alpha} \mid f(\boldsymbol{\alpha}) \leq f(\mathbf{0})\}$.

Next, we discuss some problems that admit a global error bound:

Corollary 5.2. *Problem* (1) *satisfies a global error bound from the beginning if one of the following conditions is true.*

- *Q is positive definite (kernel is positive definite).*
- *For all $i = 1, \ldots, n$, $g_i(\cdot)$ is strongly convex for all the iterates (e.g., dual ℓ_2-regularized logistic regression with positive semi-definite kernel).*
- *Q is positive semi-definite and $g_i(\cdot)$ is linear for all i with a box constraint (e.g., dual hinge loss SVM with positive semi-definite kernel).*

Corollary 5.2 implies that many widely used machine learning problems, including dual formulation of SVM and ℓ_2-regularized logistic regression, admit a global error bound from the beginning even when the kernel has a zero eigenvalue. These have been proved in Theorem 1 of [7] and in [26].

We do not require the inner solver to obtain the exact solution of (7). Instead, we define the following condition for the inexact inner solver.

Definition 5.3. An inexact solver for solving the subproblem (7) achieves a "local linear improvement" if the inexact solution $\boldsymbol{d}_{S_r}$ satisfies

$$f_{\boldsymbol{\alpha}}^{(r)}(\boldsymbol{d}_{S_r}) - f_{\boldsymbol{\alpha}}^{(r)}(\hat{\boldsymbol{d}}_{S_r}) \leq \eta\Big(f_{\boldsymbol{\alpha}}^{(r)}(\mathbf{0}) - f_{\boldsymbol{\alpha}}^{(r)}(\hat{\boldsymbol{d}}_{S_r})\Big), \tag{16}$$

for all iterates, where $f_{\boldsymbol{\alpha}}^{(r)}$ is the subproblem defined in (7), $\hat{\boldsymbol{d}}_{S_r} := \arg\min_{\Delta} f_{\boldsymbol{\alpha}}^{(r)}(\Delta)$ is the optimal solution of the subproblem, and $\eta \in (0,1)$ is a constant.

In the following we list some widely-used subproblem solvers that satisfy Definition 5.3.

Corollary 5.4. *The following subproblem solvers satisfy Definition 5.3 if the objective function admits a global error bound from the beginning: (1) Greedy Coordinate Descent with at least one step. (2) Cyclic coordinate descent with at least one epoch.*

The condition of (16) will be satisfied if an algorithm has a global linear convergence rate. The global linear convergence rate of cyclic coordinate descent has been proved in Section 3 of [26], and the linear convergence rate for greedy coordinate descent can be shown easily using the same analysis (see http://arxiv.org/abs/1608.02010).

We now show our proposed method PBM in Algorithm 1 has a global linear convergence rate in the following theorem.

Theorem 5.5. *Assume (1) the objective function admits a global error bound from the beginning (Definition 5.1), (2) the inner solver achieves a local linear improvement (Definition 5.3), (3) $R_{min} = \min_i Q_{ii} \neq 0$, and (4) the objective function is L-Lipschitz continuous for the level set. Then the following global linear convergence rate holds:*

$$f(\boldsymbol{\alpha}_{t+1}) - f(\boldsymbol{\alpha}^*) \leq \Big(1 - (1-\eta)\frac{R_{min}}{kBL\kappa^2}\Big)\Big(f(\boldsymbol{\alpha}_t) - f(\boldsymbol{\alpha}^*)\Big),$$

where $\boldsymbol{\alpha}^*$ is an optimal solution, and $B = \max_{r=1}^{k} |S_r|$ is the maximum block size.

Proof. We first define some notations that we will use in the proof. Let $\boldsymbol{\alpha}_t$ be the current solution of iteration t, $\boldsymbol{d}_t$ is the approximate solution of (7) satisfying Definition 5.3. For convenience we will omit the subscript t here (so $\boldsymbol{d} := \boldsymbol{d}_t$). We use $\boldsymbol{d}_{S_r}$ to denote the size $|S_r|$ subvector, and $\boldsymbol{d}_{<S_r>}$ to denote the n dimensional vector with

$$\boldsymbol{d}_{<S_r>} = \begin{cases} d_i & \text{if } i \in S_r \\ 0 & \text{otherwise} \end{cases}$$

By the definition of our line search procedure described in Section 4.2, we have

$$f(\boldsymbol{\alpha}_t + \beta\boldsymbol{d}) \leq f(\boldsymbol{\alpha}_t + \frac{1}{k}\sum_{r=1}^{k} \boldsymbol{d}_{<S_r>})$$

$$= f(\frac{1}{k}\sum_{r=1}^{k}(\boldsymbol{\alpha}_t + \boldsymbol{d}_{<S_r>})) \leq \frac{1}{k}\sum_{r=1}^{k} f(\boldsymbol{\alpha}_t + \boldsymbol{d}_{<S_r>}),$$

where the last inequality is from the convexity of $f(\cdot)$. We define $\hat{\boldsymbol{d}}$ to be the optimal solution of (4) (so each $\hat{\boldsymbol{d}}_{S_r}$ is the optimal solution of the r-th subproblem (7)). Then we have

$$f(\boldsymbol{\alpha}_t) - f(\boldsymbol{\alpha}_t + \boldsymbol{d}_t) \geq f(\boldsymbol{\alpha}_t) - \frac{1}{k}\sum_{r=1}^{k} f(\boldsymbol{\alpha}_t + \boldsymbol{d}_{<S_r>})$$

$$= \frac{1}{k}\sum_{r=1}^{k}\Big(f(\boldsymbol{\alpha}_t) - f(\boldsymbol{\alpha}_t + \hat{\boldsymbol{d}}_{<S_r>}) + f(\boldsymbol{\alpha}_t + \hat{\boldsymbol{d}}_{<S_r>}) - f(\boldsymbol{\alpha}_t + \boldsymbol{d}_{<S_r>})\Big)$$

$$\geq \frac{1}{k}\sum_{r=1}^{k}(1-\eta)\Big(f(\boldsymbol{\alpha}_t) - f(\boldsymbol{\alpha}_t + \hat{\boldsymbol{d}}_{<S_r>})\Big), \tag{17}$$

where the last inequality is from the local linear improvement of the in-exact subproblem solver (Definition 5.3). We then define a vector $\bar{\boldsymbol{d}}$ where each element is the optimal solution of the one variable subproblem:

$$\bar{d}_i = T_i(\boldsymbol{\alpha}_t) - (\boldsymbol{\alpha}_t)_i \;\; \forall i = 1, \ldots, n,$$

where $T_i(\boldsymbol{\alpha}_t)$ was defined in (15). First, by the definition of $\bar{Q}$ in (5), there is no approximation if the update vector is within one block, so we have $\bar{f}_{\boldsymbol{\alpha}}(\boldsymbol{d}_{<S_r>}) = f(\boldsymbol{\alpha} + \boldsymbol{d}_{<S_r>})$ for any block r. Since $\hat{\boldsymbol{d}}_{S_r}$ is the optimal solution of each subproblem, we have

$$f(\boldsymbol{\alpha}_t + \hat{\boldsymbol{d}}_{<S_r>}) \leq f(\boldsymbol{\alpha}_t + \bar{\boldsymbol{d}}_{<S_r>}/|S_r|), \;\forall r = 1, \ldots, k.$$

Combining with (17) we get

$$f(\boldsymbol{\alpha}_t) - f(\boldsymbol{\alpha}_t + \boldsymbol{d}_t) \geq \frac{1-\eta}{k}\sum_{r=1}^{k}\Big(f(\boldsymbol{\alpha}_t) - f(\boldsymbol{\alpha}_t + \bar{\boldsymbol{d}}_{<S_r>}/|S_r|)\Big)$$

$$\geq \frac{1-\eta}{k}\sum_{r=1}^{k}\Big(f(\boldsymbol{\alpha}_t) - \frac{1}{|S_r|}\sum_{i\in S_r} f(\boldsymbol{\alpha}_t + \bar{\boldsymbol{d}}_{<i>})\Big) \quad \text{(by convexity)}$$

$$\geq \frac{1-\eta}{kB}\sum_{i=1}^{n}\Big(f(\boldsymbol{\alpha}_t) - f(\boldsymbol{\alpha}_t + \bar{\boldsymbol{d}}_{<i>})\Big),$$

where the last inequality is by the definition of B. Now consider the one variable problem $f(\boldsymbol{\alpha}_t + u\boldsymbol{e}_i)$. Since R_{min} is the lower bound of $f''(\boldsymbol{\alpha}_t + u\boldsymbol{e}_i)$ and $\bar{d}_i$ is the optimal for this single variate function, we have $f(\boldsymbol{\alpha}_t) \geq f(\boldsymbol{\alpha}_t + \bar{\boldsymbol{d}}_{<i>}) + \frac{1}{2}R_{min}\bar{d}_i^2$. As a result,

$$\sum_{i=1}^{n}\Big(f(\boldsymbol{\alpha}_t) - f(\boldsymbol{\alpha}_t + \bar{\boldsymbol{d}}_{<i>})\Big) \geq \sum_{i=1}^{n} R_{min}\bar{d}_i^2 = R_{min}\|\bar{\boldsymbol{d}}\|^2.$$

Therefore,

$$f(\boldsymbol{\alpha}_t) - f(\boldsymbol{\alpha}_t + \boldsymbol{d}_t) \geq \frac{R_{min}(1-\eta)}{kB}\|T(\boldsymbol{\alpha}_t) - \boldsymbol{\alpha}_t\|^2$$

$$\geq \frac{R_{min}(1-\eta)}{kB\kappa^2}\|P_S(\boldsymbol{\alpha}_t) - \boldsymbol{\alpha}_t\|^2 \text{ (by (14))}$$

$$\geq \frac{LR_{min}(1-\eta)}{kB\kappa^2}\|f(\boldsymbol{\alpha}_t) - f(\boldsymbol{\alpha}^*)\|.$$

Table 1: Dataset statistics

Dataset	# training samples	# testing samples	# features	C	γ
cifar	50,000	10,000	3072	2^3	2^{-22}
webspam	280,000	70,000	254	2^3	2^5
covtype	464,810	116,202	54	2^5	2^5
mnist8m	8,000,000	100,000	784	2^0	2^{-21}
kddb	19,264,097	748,401	29,890,095	2^1	2^{-1}

Therefore, we have

$$f(\boldsymbol{\alpha}_{t+1}) - f(\boldsymbol{\alpha}^*) = f(\boldsymbol{\alpha}_t) - \Big(f(\boldsymbol{\alpha}_t) - f(\boldsymbol{\alpha}_{t+1})\Big) - f(\boldsymbol{\alpha}^*)$$
$$\leq \left(1 - \frac{R_{min}(1-\eta)}{kBL\kappa^2}\right)\Big(f(\boldsymbol{\alpha}_t) - f(\boldsymbol{\alpha}^*)\Big).$$

□

Note that we do not make any assumption on the partition used in PBM, so our analysis works for a wide class of optimization solvers, including distributed linear SVM solver in [14] where they only consider that case when the subproblems are solved exactly.

To sum up, Theorem 5.5 shows that the proposed PBM framework converges linearly as long as the subproblem solver satisfies Definition 5.3, which means that the subproblem solver linearly reduces the approximate function $\bar{f}_{\boldsymbol{\alpha}}$. Since coordinate descent has linear convergence rate (Corollary 5.4), the PBM algorithm converges linearly if we run ≥ 1 greedy or random coordinate descent update in the inner iteration for solving the subproblem.

6 EXPERIMENTAL RESULTS

We conduct experiments on five public large-scale datasets listed in Table 1. We follow the procedure in [5, 33] to transform cifar and mnist8m into binary classification problems, and Gaussian kernel $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = e^{-\gamma\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}$ is used in all the comparisons. Our code can be downloaded from https://github.com/cjhsieh/Distributed-KSVM.

We follow the parameter settings in [5], where C and γ are selected by 5-fold cross validation on a grid of parameters. The experiments are conducted on Texas Advanced Computing Center Maverick cluster, where each machine has a 20-core CPU with 256GB memory. In this paper, we mainly focus on kernel SVM and kernel logistic regression.

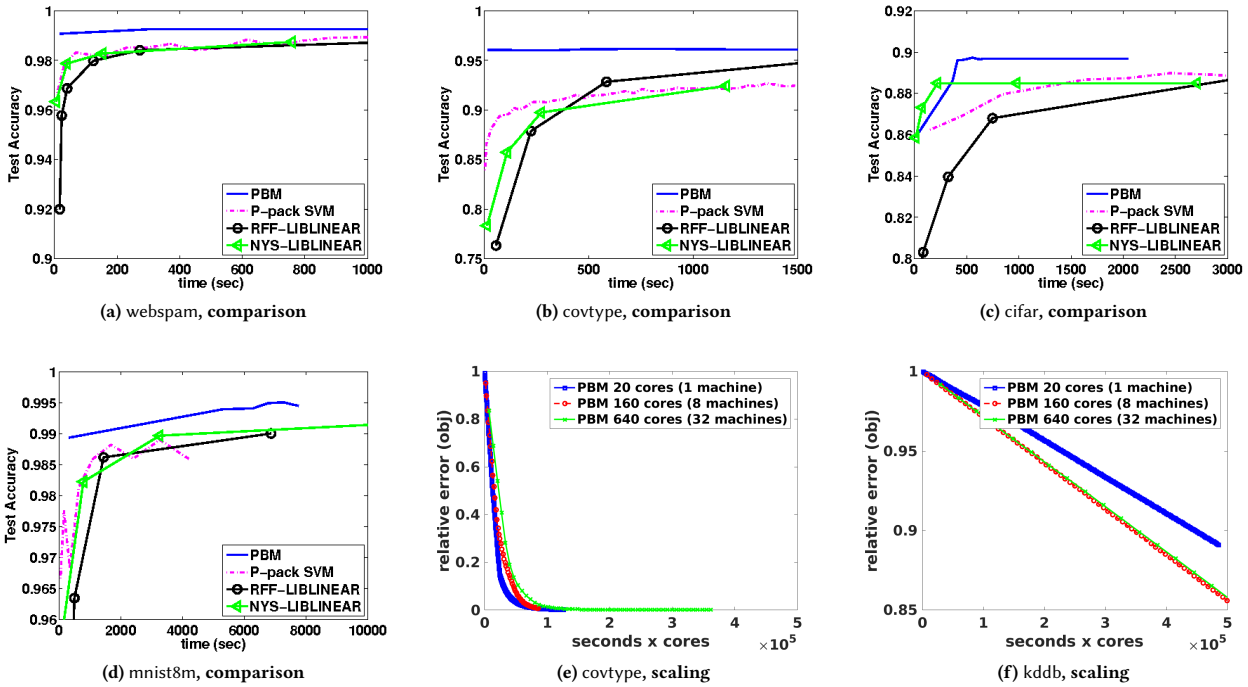
We compare our PBM method with the following distributed kernel SVM solvers, where all of them are implemented in C++ and the inter-machine communication is using MPI:

1. P-pack SVM [35]: a parallel Stochastic Gradient Descent (SGD) algorithm for kernel SVM training. We set the pack size $r = 100$ according to the original paper.
2. Random Fourier features with distributed LIBLINEAR (RFF-LIBLINEAR): RFF-LIBLINEAR generates random Fourier features and solves the resulting problem by LIBLINEAR. In a distributed system, we can compute random features [19] for each sample in parallel (this is a one-time preprocessing step), and then solve the resulting linear SVM problem by distributed dual coordinate descent [14] implemented in MPI LIBLINEAR. Note that although Fastfood [13] can generate random features in a faster way, the bottleneck for RFF-LIBLINEAR is solving the resulting linear SVM problem after generating random features, so the performance is similar.
3. Nyström approximation with distributed LIBLINEAR (NYS-LIBLINEAR): We implemented the ensemble Nyström approximation [12] with kmeans sampling in a distributed system and solved the resulting linear SVM problem by MPI LIBLINEAR. The approach is similar to [16].
4. PSVM [3]: a parallel kernel SVM solver by in-complete Cholesky factorization and a parallel interior point method. We test the performance of PSVM with the rank suggested by the original paper ($n^{0.5}$ or $n^{0.6}$ where n is number of samples).

We also compare with the state-of-the-art single-thread nonlinear kernel SVM solver DC-SVM [5] and multi-core kernel SVM solver Asyn-GCD [30] in our experiments. Unfortunately, they cannot run on distributed systems (see our discussions in Section 2, so we just use them to serve as baselines for single-thread and single-machine multi-thread kernel SVM solvers respectively. For these two solvers we directly run the code released by the authors.

Comparison with other distributed kernel SVM solvers. We use 32 machines (each with 1 thread) and the best C, γ for all the solvers. Our parameter settings are exactly the same with [5] chosen by cross-validation in $[2^{-30}, 2^{10}]$. For cifar and mnist8m the samples are not normalized, so the averaged norm mean($\|\boldsymbol{x}_i\|$) is large (it is 876 on cifar). Since Gaussian kernel is $e^{-\gamma\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}$, a good γ will be very small. We can normalize the data as well, and then a good γ will become larger. We mainly compare the *prediction accuracy* in the paper because most of the parallel kernel SVM solvers are "approximate solvers"—they solve an approximated problem, so it is not fair to evaluate them using the original objective function. In Figure 2 (a)-(d) we compare our proposed algorithm with other kernel SVM solvers. Note that in these figures, we vary the number random features for RFF-LIBLINEAR and the number of landmark points for NYS-LIBLINEAR. The results in Figure 2 (a)-(d) indicate that our proposed algorithm is much faster than other approaches. We further test the algorithms with varied number of workers and parameters in Table 2. Note that PSVM usually gets much lower test accuracy since it approximates the kernel matrix by incomplete Cholesky factorization, so we only show its results in Table 2.

MPI+OpenMP Hybrid Implementation. In the previous experiment, each machine only runs with one MPI worker. To exploit the power of each machine, we further implement an MPI+OpenMP hybrid version of our algorithm. In this setting, each machine can use multiple cores with OpenMP, and the between-machine communication is still done by MPI. Since cores in the same machine can access the shared memory space, the parallelism is often simpler because there is almost no communication cost.



(a) webspam, **comparison** (b) covtype, **comparison** (c) cifar, **comparison**

(d) mnist8m, **comparison** (e) covtype, **scaling** (f) kddb, **scaling**

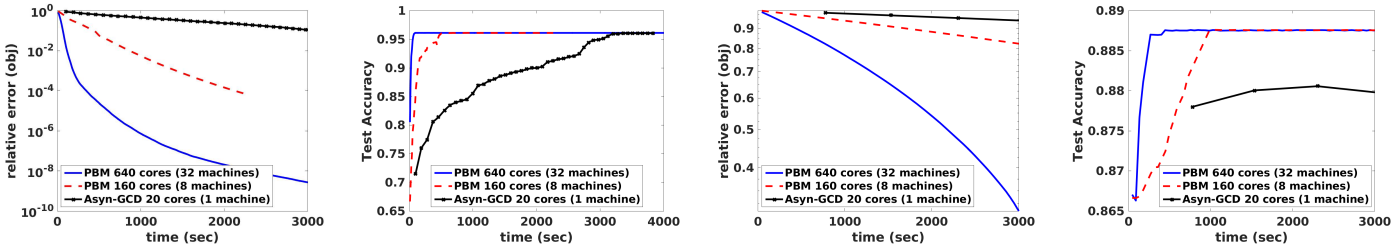
Figure 2: (a)-(d): Comparison with other distributed kernel SVM solvers using 32 workers. Markers for RFF-LIBLINEAR and NYS-LIBLINEAR are obtained by varying the number random features and landmark points respectively. (e)-(f): The objective function of PBM as a function of computation time (time in seconds × the number of workers), when the number of workers is varied. Results show that PBM has very good scalability on large datasets.

Table 2: Comparison on real datasets for kernel SVM. Here we use 32 machines (each machine with 1 thread) for all the distributed solvers (PBM, P-packSGD, PSVM), and 1 machine for the serial solver (DC-SVM). For kddb, we use 32 machines and 20 cores in each machine to test MPI+OpenMP Hybrid Implementation of PBM. The first column of PBM shows that PBM achieves good test accuracy after 1 iteration, and the second column of PBM shows PBM can achieve an accurate solution (with $\frac{f(\boldsymbol{\alpha})-f(\boldsymbol{\alpha}^*)}{|f(\boldsymbol{\alpha}^*)|} < 10^{-3}$) quickly and obtain even better accuracy. Note that "-" indicates the training time is more than 10 hours.

	PBM (first step)		PBM (10^{-3} error)		P-packSGD		PSVM $p = n^{0.5}$		PSVM $p = n^{0.6}$		DC-SVM (10^{-3} error)	
	time(s)	acc(%)	time(s)	acc(%)	time(s)	acc(%)	time(s)	acc(%)	time(s)	acc(%)	time(s)	acc(%)
webspam	**16**	99.07	360	**99.26**	1478	98.99	773	75.79	2304	88.68	8742	99.26
covtype	**14**	96.05	772	**96.13**	1349	92.67	286	76.00	7071	81.53	10457	96.13
cifar	**15**	85.91	540	**89.72**	1233	88.31	41	79.89	1474	69.73	13006	89.72
mnist8m	**321**	98.94	8112	**99.45**	2414	98.60	-	-	-	-	-	-
kddb	**1832**	88.59	32740	**88.76**	-	-	-	-	-	-	-	-

In our implementation, within each machine, we use OpenMP to compute the elements of kernel matrix in parallel. There are two places requiring kernel value computations: (1) When using the greedy coordinate descent algorithm to solve the subproblem (Step 4 in Algorithm 1), we need to access the i-th column of kernel matrix Q when updating the variable α_i, and (2) Step 5 in Algorithm 1 for computing $Q_{:,S_r}\boldsymbol{d}_{S_r}$. Note that we use the kernel cache strategy developed in LIBSVM for maintaining several columns of kernel matrix in memory, and remove the Least Recent Used (LRU) column when running out of memory space. Therefore, the OpenMP parallelism is used when the required kernel matrix column is not found in kernel cache (cache miss) and needed to be computed from scratch. Surprisingly, this simple hybrid implementation achieves superb speedup, especially in very large datasets when the memory can only cache a small portion of kernel matrix. For example, on covtype dataset, PBM with 32 machines and using 20 cores per machine is 8.9 times faster than its pure MPI implementation that only uses 1 core per machine; on kddb dataset, with the same setting, hybrid PBM is 15.2 times faster than its MPI version with 1 core per machine.

Scalability of PBM. For the second experiment we varied the number of machines from 1 to 32, and plot the scaling behavior of PBM. In Figure 2 (e)-(f), we set y-axis to be the relative error defined



(a) covtype, **obj** **(b)** covtype, **accuracy** **(c)** kddb, **obj** **(d)** kddb, **accuracy**

Figure 3: Comparison with Asyn-GCD, the state-of-the-art multi-core single machine kernel SVM solver. Results show that our algortihm is uch faster than Asyn-GCD by using multiple computers.

by $(f(\boldsymbol{\alpha}_t) - f(\boldsymbol{\alpha}^*))/f(\boldsymbol{\alpha}^*)$ where $\boldsymbol{\alpha}^*$ is the optimal solution, and x-axis to be the total CPU time expended which is given by the number of seconds elapsed multiplied by the number of workers. We plot the convergence curves by changing number of machines (and using all the 20 cores in each machine). The perfect linear speedup is achieved if the curves overlap. This is indeed the case for both covtype and kddb.

Interestingly, in the kddb dataset, we observe a super-linear speedup when using 8 and 32 machines compared with the single-machine version. This is mainly due to the increasing amount of memory available for caching kernel values—for example, a single machine can only afford 100GB memory for storing the computed kernel values, while using 32 machines there will be a total amount of 3200GB memory for storing those values. Therefore, the "kernel cache miss rate" (portion of kernel values that needs to be recomputed) is much better as number of machine increases. As a result, the speedup of is almost linear and sometimes even super linear.

Comparison with single-machine solvers. We also compare PBM with state-of-the-art single machine solver for kernel SVM: Asyn-GCD [30] which uses multiple cores in a single machine, and DC-SVM [5] which can only run with a single thread. The results are in Table 2 and Figure 3. DC-SVM first computes the solutions in each partition, and then use the concatenation of local (dual) solutions to initialize a global kernel SVM solver (e.g., LIBSVM). However, the top level of DC-SVM is the bottleneck (taking 2/3 of the run time), so the speed is still slow in Table 2. Asyn-GCD is an asynchronous parallel coordinate descent algorithm for kernel SVM, and it performs much better than other single-machine SVM solvers including LIBSVM (see [30]). In Figure 3, we show that our distributed algorithm is much faster using multiple machines which might not be the case for some other distributed solvers. This confirms that our algorithm is much faster than the best single-machine kernel SVM solver.

Kernel logistic regression. We implement the PBM algorithm to solve the kernel logistic regression problem. Note that PSVM cannot be directly applied to kernel logistic regression. We use greedy coordinate descent proposed in [11] to solve each subproblem (7). The results are presented in Table 3, showing that our algorithm is faster than P-packSGD.

Table 3: Comparison on real datasets for kernel logistic regression. Here we use 32 machines (each machine with 1 thread).

	PBM (first step)		PBM (10^{-3} error)		P-packSGD	
	time(s)	acc(%)	time(s)	acc(%)	time(s)	acc(%)
webspam	**1679**	92.01	2131	**99.07**	4417	98.96
cifar	**471**	83.37	758	**88.14**	2115	87.07

7 CONCLUSION

We have proposed a parallel block minimization (PBM) framework for solving kernel machines on distributed systems. We show that PBM significantly outperforms other approaches on large-scale datasets, and prove a global linear convergence of PBM under mild conditions. By using 32 machines with totally 640 cores, our algorithm can solve the kddb dataset with 20 millions samples and 30 millions features in half hour.

Acknowledgments This research was supported by NSF grants CCF-1320746, IIS- 1546452 and CCF-1564000. Cho-Jui Hsieh also acknowledges support from XSEDE.

REFERENCES

[1] E. Chan, M. Heimlich, A. Purkayastha, and R. van de Geijn. 2007. Collective Communication: Theory, Practice, and Experience. *Concurrency and Computation: Practice and Experience* 19 (2007), 1749–1783.
[2] Chih-Chung Chang and Chih-Jen Lin. 2000. *LIBSVM: Introduction and Benchmarks.* Technical Report. Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan.
[3] E. Chang, K. Zhu, H. Wang, H. Bai, J. Li, Z. Qiu, and H. Cui. 2008. Parallelizing Support Vector Machines on Distributed Computers. In *NIPS*.
[4] H. P. Graf, E. Cosatto, L. Bottou, I. Dundanovic, and V. Vapnik. 2005. Parallel Support Vector Machines: The Cascade SVM. In *NIPS*.
[5] C. J. Hsieh, S. Si, and I. S. Dhillon. 2014. A Divide-and-Conquer Solver for Kernel Support Vector Machines. In *ICML*.
[6] Cho-Jui Hsieh, Si Si, and Inderjit S Dhillon. 2014. Fast prediction for large-scale kernel machines. In *Advances in Neural Information Processing Systems*. 3689–3697.
[7] C.-J. Hsieh, H. F. Yu, and I. S. Dhillon. 2015. PASSCoDe: Parallel ASynchronous Stochastic dual Coordinate Descent. In *ICML*.
[8] P.-S. Huang, H. Avron, T. Sainath, V. Sindhwani, and B. Ramabhadran. 2014. Kernel Methods Match Deep Neural Networks on TIMIT. In *ICASSP*. 205–209.
[9] M. Jaggi, V. Smith, M. Takáč, J. Terhorst, T. Hofmann, and M. Jordan. 2014. Communication-efficient Distributed Dual Coordinate ascent. In *NIPS*.
[10] Thorsten Joachims. 1998. Making Large-scale SVM Learning Practical. In *Advances in Kernel Methods – Support Vector Learning*.
[11] S. Sathiya Keerthi, Kaibo Duan, Shirish Shevade, and Aun Neow Poo. 2005. A Fast Dual Algorithm for Kernel Logistic Regression. *Machine Learning* 61 (2005), 151–165.
[12] S. Kumar, M. Mohri, and A. Talwalkar. 2009. Ensemble Nyström Method. In *NIPS*.

[13] Q. Le, T. Sarlós, and A. Smola. 2013. Fastfood – Approximating Kernel Expansions in Loglinear Time. In *ICML*.

[14] C.-P. Lee and D. Roth. 2015. Distributed Box-Constrained Quadratic Optimization for Dual Linear SVM. In *ICML*.

[15] Chenxin Ma, Virginia Smith, Martin Jaggi, Michael I. Jordan, Peter Richtárik, and Martin Takáč. 2015. Adding vs. Averaging in Distributed Primal-Dual Optimization. In *ICML*.

[16] Dhruv Mahajan, S. Sathiya Keerthi, and Sundararajan Sellamanickam. 2014. A Distributed Algorithm for Training Nonlinear Kernel Machines. *arXiv:1405.4543* (2014).

[17] Dhruv Mahajan, S. Sathiya Keerthi, and Sundararajan Sellamanickam. 2014. A distributed block coordinate descent method for training L1 regularized linear classifiers. *arXiv:1405.4544* (2014).

[18] John C. Platt. 1998. Fast Training of Support Vector Machines using Sequential Minimal Optimization. In *Advances in Kernel Methods - Support Vector Learning*.

[19] Ali Rahimi and Benjamin Recht. 2008. Random Features for Large-scale Kernel Machines. In *NIPS*.

[20] P. Richtárik and M. Takáč. 2012. Parallel Coordinate Descent Methods for Big Data Optimization. *Mathematical Programming* (2012).

[21] C. Scherrer, A. Tewari, M. Halappanavar, and D. Haglin. 2012. Feature Clustering for Accelerating Parallel Coordinate Descent. In *NIPS*.

[22] Si Si, Kai-Yang Chiang, Cho-Jui Hsieh, Nikhil Rao, and Inderjit S. Dhillon. 2016. Goal-Directed Inductive Matrix Completion. In *KDD*. 1165–1174.

[23] Si Si, Cho-Jui Hsieh, and Inderjit S. Dhillon. 2016. Computationally Efficient Nyström Approximation using Fast Transforms. In *ICML*. 2655–2663.

[24] S. Si, C. J. Hsieh, and I. S. Dhillon. 2017. Memory Efficient Kernel Approximation. (2017).

[25] Stephen Tu, Rebecca Roelofs, Shivaram Venkataraman, and Benjamin Recht. 2016. Large Scale Kernel Learning using Block Coordinate Descent. *CoRR* abs/1602.05310 (2016).

[26] P.-W. Wang and C.-J. Lin. 2014. Iteration Complexity of Feasible Descent Methods for Convex Optimization. *Journal of Machine Learning Research* 15 (2014), 1523–1548.

[27] C. K. I. Williams and M. Seeger. 2001. Using the Nyström Method to Speed Up Kernel Machines. In *NIPS*.

[28] T. Yang. 2013. Trading Computation for Communication: Distributed Stochastic Dual Coordinate Ascent. In *NIPS*.

[29] Y. You, J. Demmel, K. Czechowski, L. Song, and R. Vuduc. 2015. CA-SVM: Communication-Avoiding Support Vector Machines on Clusters. In *IPDPS*.

[30] Yang You, Xiangru Lian, Ji Liu, Hsiang-Fu Yu, Inderjit Dhillon, James Demmel, and Cho-Jui Hsieh. 2016. Asynchronous Parallel Greedy Coordinate Descent. In *NIPS*.

[31] H.-F. Yu, F.-L. Huang, and C.-J. Lin. 2011. Dual Coordinate Descent Methods for Logistic Regression and Maximum Entropy Models. *Machine Learning* 85, 1-2 (2011), 41–75.

[32] Huan Zhang and Cho-Jui Hsieh. 2016. Fixing the convergence problems in parallel asynchronous dual coordinate descent. In *ICDM*.

[33] K. Zhang, L. Lan, Z. Wang, and F. Moerchen. 2012. Scaling up Kernel SVM on Limited Resources: A Low-rank Linearization Approach. In *AISTATS*.

[34] Y. Zhang and L. Xiao. 2015. DiSCO: Communication-Efficient Distributed Optimization of Self-Concordant Loss. In *ICML*.

[35] Zeyuan A. Zhu, Weizhu Chen, Gang Wang, Chenguang Zhu, and Zheng Chen. 2009. P-packSVM: Parallel Primal Gradient Descent Kernel SVM. In *ICDM*.