P-packSVM: Parallel Primal grAdient desCent Kernel SVM

Zeyuan Allen Zhu[12*], Weizhu Chen[2], Gang Wang[2], Chenguang Zhu[23], Zheng Chen[2]

[1] Fundamental Science Class, Department of Physics, Tsinghua University
zhuzeyuan@hotmail.com

[2] Microsoft Research Asia
{v-zezhu, wzchen, gawa, v-chezhu, zhengc}@microsoft.com

[3] Department of Computer Science and Technology, Tsinghua University
zcg.cs60@gmail.com

***Abstract*—It is an extreme challenge to produce a nonlinear SVM classifier on very large scale data. In this paper we describe a novel P-packSVM algorithm that can solve the Support Vector Machine (SVM) optimization problem with an arbitrary kernel. This algorithm embraces the best known stochastic gradient descent method to optimize the primal objective, and has $1/\epsilon$ dependency in complexity to obtain a solution of optimization error ϵ. The algorithm can be highly parallelized with a special packing strategy, and experiences sub-linear speed-up with hundreds of processors. We demonstrate that P-packSVM achieves accuracy sufficiently close to that of SVM-light, and overwhelms the state-of-the-art parallel SVM trainer PSVM in both accuracy and efficiency. As an illustration, our algorithm trains CCAT dataset with 800k samples in 13 minutes and 95% accuracy, while PSVM needs 5 hours but only has 92% accuracy. We at last demonstrate the capability of P-packSVM on 8 million training samples.**

Keywords–parallel; kernel; support vector machine; stochastic gradient descent; packing strategy

I. INTRODUCTION

Since its first introduction by V. Vapnik in 1963 [26], support vector machine (SVM) has been a widely used supervised learning method for classification, regression [6] and ranking [13] problem. Strictly speaking, a set of training data, consisting of m samples is given:

$$\Psi = \{(\boldsymbol{x}_i, y_i) | \boldsymbol{x}_i \in \mathbb{R}^n, y_i \in \{-1,1\}\}_{i=1}^m \quad (1)$$

where $\boldsymbol{x_i}$ is the feature vector for the i[th] sample and y_i is either $+1$ or -1, indicating the binary class this sample belongs to. The soft margin SVM problem [26] is aiming to find a *maximum margin separating hyper-plane* for the two classes, indicating by its normal vector, or *predictor* $\boldsymbol{w}$, by minimizing the following quadratic convex objective, which is also known as the *primal SVM objective*:

$$f(\boldsymbol{w}) = \frac{\sigma}{2}\|\boldsymbol{w}\|_2^2 + \frac{1}{m}\sum_{i=1}^{m} \max\{0,1 - y_i\langle \boldsymbol{w}, \phi(\boldsymbol{x}_i)\rangle\} \quad (2)$$

where the first term is a 2-norm regularizer $1/2\,\|\boldsymbol{w}\|_2^2$ with the regularizer weight σ, while the second term is the empirical loss function:

$$\ell(\boldsymbol{w}) = \frac{1}{m}\sum_{i=1}^{m} \max\{0,1 - y_i\langle \boldsymbol{w}, \phi(\boldsymbol{x}_i)\rangle\} \quad (3)$$

Originally proposed by Aizerman et al [1], $\phi(.)$ is the mapping that projects the point from feature space to the *Reproducing Kernel Hilbert Space* (RKHS), satisfying $\langle \phi(\boldsymbol{x_i})^T, \phi(\boldsymbol{x_j})\rangle = \mathcal{K}(\boldsymbol{x_i}, \boldsymbol{x_j})$ for some *Mercer* kernel $\mathcal{K}(.,.)$. The integration of the kernel enables SVM to produce a non-linear predictor, w. This is often called *non-linear SVM* or *kernel SVM*. For example, a polynomial kernel allows one to model feature conjunctions, while a Gaussian kernel enables us to pick out hyper spheres in features [20]. Recent works, such as PEGASOS, effectively solved the linear SVM problems [24] [14] [30]; however, to accelerate the kernel SVM is a very desirable and difficult research problem.

We analyze in this paper a simple stochastic gradient descent (SGD) based algorithm that directly optimizes the primal objective (2), called packSVM, to solve SVM for an arbitrary kernel. The algorithm embraces a bunch of iterations. At each iteration, it first randomly picks up a single sample from the training sample pool to approximate $\ell(\boldsymbol{w})$, and then calculates the gradient and updates the predictor $\boldsymbol{w}$ accordingly. It is worth noting that our proposed packSVM algorithm embraces the best known learning rate [24] and requires $O(m/\sigma\epsilon\delta)$ in time, where δ is the confidence parameter and ϵ is the optimization error. This means, with probability at least $1-\delta$ we can obtain a predictor $\boldsymbol{w}$ that is guaranteed to satisfy $f(\boldsymbol{w}) \le f(\boldsymbol{w}^*) + \epsilon$, if w^* is the optimal solution.

An important contribution of this paper is that we parallelize the above algorithm with the help of a distributed hash table and our innovative packing strategy. We call our proposed parallel algorithm P-packSVM. Notice that it is naturally difficult to parallelize SGD algorithms in hundreds of processors due to their huge communication cost. The packing strategy non-trivially reduces the communication cost and allows a sub-linear speed-up with 512 processors. The time complexity of P-packSVM is thus reduced to $O(m/\sigma t\delta p)$ if using p processors. At the same time, P-packSVM uses only $O(m/p)$ space for each processor.

We conduct extensive experiments and show that P-packSVM overwhelms the state-of-the-art PSVM [5] in both

[*] This work was done when the first author was visiting Microsoft Research Asia.

accuracy and efficiency, and runs hundreds of times faster than SVM-light. Meanwhile, its accuracy is comparable to SVM-light. For example, P-packSVM trains a CCAT dataset of 800k samples in 761 seconds with a speed-up of 295 times on 512 processors; it trains a CovType dataset of 500k samples in 236 seconds with a speed-up of 416 times on 512 processors. The 8 million *MNIST8m* test set has also been employed and we state that our proposed algorithm is capable of performing well in million scale data set.

This reminder of this paper is organized as follows. We first state the related works of SVM in Section II. Next in Section III, we propose our P-packSVM algorithm by introducing its sequential implementation and then move onto the parallel one with introducing the distributed hash and the innovative packing strategy. We also emphasize the differences of our P-packSVM with other contemporary works in this section. Experimental results are then provided in Section IV. Finally we leave several enhancements to our algorithm in Section V and conclude our work in Section VI.

II. Related Works

Historically, the SVM problem has been well studied with the help of the *dual objective*. The method of *Lagrangian multipliers* introduces a transformation from the primal objective (2) into its dual form:

$$\min \frac{1}{2}\boldsymbol{\alpha}^T \boldsymbol{Q}\boldsymbol{\alpha} - \boldsymbol{\alpha}^T \mathbf{1}, \qquad \text{s.t.}\, \mathbf{0} \le \boldsymbol{\alpha} \le \boldsymbol{C}, \boldsymbol{y}^T \boldsymbol{\alpha} = 0 \qquad (4)$$

where $[Q]_{ij} = y_i y_j \mathcal{K}(x_i, x_j)$, and $\alpha \in \mathbb{R}^m$ is the vector of the *Lagrangian dual variable*, also known as the *support vector* in SVM. The predictor $\boldsymbol{w}$ is a superposition of $\phi(x_i)$, namely $\boldsymbol{w} = \sum_{i=1}^{n} \alpha_i y_i \phi(\boldsymbol{x}_i)$. We divide the state-of-the-art SVM trainers mainly into the following three categories.

Interior Point Method (IPM): minimizing the dual objective is a convex Quadratic Programming (QP) problem and can be solved via the primal-dual Interior-Point Method [21]. The idea of IPM is to incorporate Newton or Quasi-Newton methods with the number of iterations proportional to $\log(1/\epsilon)$ [27], where ϵ is the desired accuracy. However, the memory requirements of IPM are as high as $O(m^2)$ and the computational cost is $O(m^3)$ for each iteration.

Recently, E. Y. Chang et al [5] proposed an algorithm called PSVM. It enables a parallel implementation of IPM and Incomplete Cholesky Factorization [9] (ICF) $Q \approx HH^T$, where H has the dimension $m \times m'$. [5] empirically showed that $m' = m^{0.5}$ gives a good approximation, and thus induced an algorithm with the time complexity of $O(m^2/p)$ for each iteration and the space requirement of $O(m^{1.5}/p)$, where p is the number of processors. To the best of our knowledge, E. Y. Chang et al firstly studied the parallel kernel SVM on 500 processors in experiments, and they reported a parallel speed-up of up to 169 times with 800k training samples.

Sequential Minimal Optimization (SMO): to make SVM more practical, SMO algorithms are developed by decomposing the large QP problem into an inactive part and an active part – a so called "working set". Many open source tools, like Osuna's decomposition [7], libSVM [4] and SVM-light [12], are capable of training as large as several hundred thousand training samples on a single machine.

Attempts to parallel SMO algorithms have also been made. For example, Zanghirati and Zanni [28] proposed a parallel implementation of SVM-light, especially effective for Gaussian kernels; Cao et al [3] also parallelized a slightly modified SMO algorithm. For these two papers, the authors conducted experiments on up to 32 processors with 60k training samples, claiming a speed-up of approximately 20 times.

Stochastic Gradient Descent (SGD): until recently, a growing amount of attention had been paid towards stochastic gradient descent algorithms, in which the gradient is approximated by evaluating on a single training sample. This algorithm has been applied to the primal objective of linear-SVM algorithms. T. Zhang [29] proved that a constant learning rate (no parameter sweep required) in SGD will numerically achieve good accuracy, enabling a running time of $O(1/\epsilon^2)$ for a linear kernel. The algorithm Norma [16] suggests a learning rate proportional to $1/\sqrt{t}$, where t is the number of iteration. Shai Shalev-Shwartz *et al* [24] aggressively adopted a learning rate of $1/\sigma t$. It turns out this learning rate is up-to-now the most efficient [24] for linear SVM, and even endowed with an inverse time dependency for fixed accuracy [25]. Notice that these works focus on linear predictors only. Some of them addressed their potential to be extended to kernel SVM, but with an extra time complexity of $O(m)$.

Hush et al [10] proved that the convergence rate in the primal objective is slow when an algorithm tries to optimize the dual one instead. This applies to the algorithms in the first two categories. Our proposed method falls into the third category, and thus is born with advantages. We incorporate a distributed hash table to enable the parallelism and the packing strategy to facilitate the parallelism. We will show that though in general only the algorithm in the first category can be effectively parallelized, our proposed packing strategy reverses the adversity.

III. The Algorithm

In this section we first adopt the stochastic gradient descent method to the kernel SVM problem, and provide the result of its convergence analysis. Next, we propose its parallel implementation and a special packing strategy. We finally compare our proposed method with other contemporary works.

A. Sequential packSVM

In this sub-section we describe a sequential stochastic gradient descent (SGD) algorithm on the primal SVM objective. With the incorporation of kernels, we call it S-packSVM. We adopt the framework discussed in [24], which has the best known learning rate and an additional projection phrase.

Considering the empirical loss (3), it averages the hinge loss among all training examples. In the spirit of the SGD algorithm, this empirical loss can be approximated by the

1. INPUT: σ, T, training sample space Ψ
2. INITIALIZE: $\boldsymbol{w} = 0$
3. FOR $t = 1,2, \dots, T$
4. Randomly pick up $(\boldsymbol{x}, y) \in \Psi$
5. Predict $y' \leftarrow \langle \boldsymbol{w}, \phi(\boldsymbol{x}) \rangle$
6. $\boldsymbol{w} \leftarrow (1 - 1/t)\boldsymbol{w}$
7. IF $yy' < 1$ THEN $\boldsymbol{w} \leftarrow \boldsymbol{w} + \frac{y}{\sigma t}\phi(\boldsymbol{x})$
8. $\boldsymbol{w} \leftarrow \min\left\{1, \frac{1/\sqrt{\sigma}}{\|w\|_2}\right\}\boldsymbol{w}$
9. RETURN $\boldsymbol{w}$

Figure 1. S-packSVM algorithm

1. INPUT: σ, T, training sample space Ψ
2. INITIALIZE: $\mathcal{H} = \emptyset, s = 1, norm = 0$
3. FOR $t = 1,2, \dots, T$
4. Randomly pick up $(\boldsymbol{x}, y) \in \Psi$
5. $y' \leftarrow s\langle \boldsymbol{v}, \phi(\boldsymbol{x}) \rangle$ by iterating all entries in $\mathcal{H}$
6. $s \leftarrow (1 - 1/t)s$
7. IF $yy' < 1$ THEN
8. $norm \leftarrow norm + \frac{2y}{\sigma t} \cdot y' + \left(\frac{y}{\sigma t}\right)^2 \mathcal{K}(\boldsymbol{x}, \boldsymbol{x})$
9. IF key $\boldsymbol{x}$ is found in $\mathcal{H}$, THEN add its value by $\frac{y}{\sigma t s}$ in $\mathcal{H}$; ELSE add $\mathcal{H}$ a new entry $\left(\boldsymbol{x}, \frac{y}{\sigma t s}\right)$
10. IF $norm > 1/\sigma$ THEN $s \leftarrow s \cdot \frac{1}{\sqrt{\sigma \cdot norm}}$; $norm \leftarrow 1/\sigma$
11. RETURN $s\boldsymbol{v}$ by iterating all entries in $\mathcal{H}$

Figure 2. S-packSVM pseudo-code

hinge loss on a single training sample. Based on this idea, we propose our S-packSVM with T iterations. At iteration $t \in \{1, \dots T\}$, it picks up a random example $\left(x_i(t), y_i(t)\right) \in \Psi$, and approximates the empirical loss (3) and the objective (2) as the following:

$$\begin{aligned} \ell(w) \approx \ell_t(w) &:= \max\{0, 1 - y_{i(t)} \cdot \langle \boldsymbol{w}, \phi(\boldsymbol{x}_{i(t)}) \rangle\} \\ f(w) \approx f_t(w) &:= \frac{\sigma}{2}\|w\|_2^2 + \ell_t(w) \end{aligned} \tag{5}$$

Directly adopting the learning rate suggested in [24], we modify the predictor as below in iteration t:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \frac{1}{\sigma t}\nabla f_t(\boldsymbol{w}) \tag{6}$$

We notice that the operator ∇ does not require the differentiability of function f_t, but the existence of its sub-gradient [24] [23]. We write down the sub-gradient explicitly:

$$\nabla f_t(\boldsymbol{w}) = \sigma \boldsymbol{w} - \begin{cases} 0, & y_{i(t)} \cdot \langle \boldsymbol{w}, \phi(\boldsymbol{x}_{i(t)}) \rangle \geq 1 \\ y_{i(t)}\phi(\boldsymbol{x}_{i(t)}), & y_{i(t)} \cdot \langle \boldsymbol{w}, \phi(\boldsymbol{x}_{i(t)}) \rangle < 1 \end{cases} \tag{7}$$

When kernels are introduced, we usually write $\boldsymbol{w}$ as a superposition of samples $\boldsymbol{w} = \sum_{i=1}^{m} \alpha_i y_i \phi(\boldsymbol{x}_i)$, and the subtraction in (6) simply consists of an overall shrinking and the addition of at most one term.

$$\boldsymbol{w} \leftarrow \left(1 - \frac{1}{t}\right)\boldsymbol{w} + \begin{cases} 0, & y_{i(t)} \cdot \langle \boldsymbol{w}, \phi(\boldsymbol{x}_{i(t)}) \rangle \geq 1 \\ \frac{y_{i(t)}}{\sigma t} \cdot \phi(\boldsymbol{x}_{i(t)}), & y_{i(t)} \cdot \langle \boldsymbol{w}, \phi(\boldsymbol{x}_{i(t)}) \rangle < 1 \end{cases} \tag{8}$$

After each update to $\boldsymbol{w}$, a projection is applied to help $\boldsymbol{w}$ to get closer to the optimum [24]:

$$\boldsymbol{w} \leftarrow \min\left\{1, \frac{1/\sqrt{\sigma}}{\|\boldsymbol{w}\|_2}\right\}\boldsymbol{w} \tag{9}$$

In the implementation of S-packSVM, we express $\boldsymbol{w} = s\boldsymbol{v}$ where $s \in \mathbb{R}$ is a scalar that allows Line 6 and 8 of Figure 1 to run in a constant time. This is because when performing scaling we can simply change the value of s instead of modifying the coefficients of all the terms in $\boldsymbol{w}$, and the adding $\boldsymbol{w} \leftarrow \boldsymbol{w} + \frac{y}{\sigma t}\phi(\boldsymbol{x}_i)$ implies $\boldsymbol{v} \leftarrow \boldsymbol{v} + \frac{y}{\sigma t s}\phi(\boldsymbol{x}_i)$. Besides, a variable $norm$ is employed to store the up-to-date value of $\|w\|_2^2$, and a hash table $\mathcal{H}$ is used to store the key-value pairs (x_i, β_i) in the representation of $\boldsymbol{v} = \sum_i \beta_i \phi(x_i)$. Figure 2 gives the pseudo code of the algorithm presented in Figure 1.

Considering that our objective (2) is a *strongly convex* function [23] with respect to $\boldsymbol{w}$, we follow the convergence analysis in [24], which is a special case of S-packSVM when linear kernel is adopted. Due to limited space, we only provide the sketch of the proof, while the details simply follow the idea of [24].

The first observation is that with the strong convexity, we can substitute the main result of [15], and arrive at the following inequality for some constant C:

$$\frac{1}{T}\sum_{t=1}^{T} f_t(\boldsymbol{w_t}) - \frac{1}{T}\min_{\boldsymbol{w} \in \mathrm{S}} \sum_{t=1}^{T} f_t(\boldsymbol{w}) \leq \frac{C \cdot \ln \mathrm{T}}{T\sigma} \tag{10}$$

The second term above can be related to the optimal objective $f(\boldsymbol{w}^*)$ using Markov inequality, while the first term is related to the empirical objective. The final result is that S-packSVM requires $T = \tilde{O}(1/\sigma\delta\epsilon)$ iterations to obtain a predictor $\boldsymbol{w}$, satisfying $f(\boldsymbol{w}) \leq f(\boldsymbol{w}^*) + \epsilon$ with probability at least $1 - \delta$, assuming $\boldsymbol{w}^*$ to be the optimal predictor. This suggests a total running time of $\tilde{O}(m/\sigma\delta\epsilon)$ for S-packSVM, as all commands except Line 5 take a constant running time, while Line 5 needs a complete enumeration through all the entries of $\mathcal{H}$ in at most $O(m)$ time.

B. Parallel packSVM

Although the complexity of S-packSVM depends linearly on $1/\epsilon$, which is much better than the general SGD algorithm [29], its sequential behavior does not show a significant superiority in efficiency. This is because from many experimental observations, we find the optimal σ on the same order of $1/m$ (see Appendix), and thus the overall time complexity is in square dependence on the number of samples m. In this sub-section, we provide the parallel packSVM, called P-packSVM, and show that it has some unique advantages in kernel SVM training. Before going into

PROCESSOR i

1. INPUT: σ, T, training sample space Ψ
2. INITIALIZE: $\mathcal{H}_i = \emptyset, s = 1, norm = 0$
3. FOR $t = 1,2,\dots,T$
4. All processors pick up the same random $(\boldsymbol{x}, y) \in \Psi$
5. $y_i' \leftarrow s\langle \boldsymbol{v_i}, \phi(\boldsymbol{x})\rangle$ by iterating all entries in $\mathcal{H}_i$
6. Sum up $y' \leftarrow y_i'$ via inter-processor communication
7. $s \leftarrow (1 - 1/t)s$
8. IF $yy' < 1$ THEN
9. $norm \leftarrow norm + \frac{2y}{\sigma t} \cdot y' + \left(\frac{y}{\sigma t}\right)^2 \mathcal{K}(\boldsymbol{x}, \boldsymbol{x})$
10. IF key $\boldsymbol{x}$ is found in $\mathcal{H}_i$ THEN add its value by $\frac{y}{\sigma t}/s$ in $\mathcal{H}_i$;
11. IF no processor reports the existence
    THEN Find a least occupied processor j
    and add $\mathcal{H}_j$ a new entry $\left(\boldsymbol{x}, \frac{y}{\sigma t}/s\right)$
12. IF $norm > 1/\sigma$ THEN $s \leftarrow s \cdot \frac{1}{\sqrt{\sigma \cdot norm}}$; $norm \leftarrow 1/\sigma$
13. RETURN $s\boldsymbol{v}$ by iterating all entries in $\mathcal{H}_1, \dots H_p$

Figure 3. P-packSVM pseudo-code, without packing

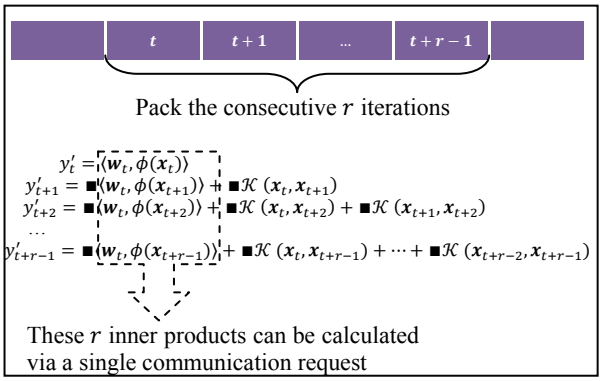

Figure 4. Packing strategy.
We use ■ to hide the complex coefficients.

detail, we consider the two characteristics related to the parallelism that packSVM embodies:

- **Merit**: A single iteration can be highly parallelized. The sole time-consuming process – the calculation of $\langle \boldsymbol{v}, \phi(\boldsymbol{x})\rangle$ can be highly parallelized via a distributed storage of the entries (x_i, β_i) in $\mathcal{H}$.
- **Defect**: Too many iterations exist. It initiates at least one communication request among all processors in each iteration. The mass communication will slow down the parallel program when the number of processors increases (This is due to the synchronization overhead).

Considering the above two characteristics, we propose a distributed hash table to develop the merit, and a packing strategy to overcome the defect.

Distributed Hash Table. We enable a distributed hash table to speed up the bottleneck process in Line 5 of Figure 2. Entries in $\mathcal{H}$ are averagely divided to all the processors. Suppose the i^{th} processor saves a subset

$$\mathcal{H}_i = \left\{\left(\boldsymbol{x}_{i,j}, \beta_{i,j}\right)\right\}_{j=1}^{|\mathcal{H}_i|} \subset \mathcal{H}$$

to represent $\boldsymbol{v}_i = \sum_j \beta_{i,j}\phi(\boldsymbol{x}_{i,j})$. Specifically, we explain two important operations:

- Enumeration (Line 5 of Figure 2): the calculation of inner product $\langle \boldsymbol{v}, \phi(x)\rangle$ can be distributed to all the processors, by each calculating $\langle \boldsymbol{v}_i, \phi(x)\rangle = \sum_j \beta_{i,j}\mathcal{K}(\boldsymbol{x}_{i,j}, \boldsymbol{x})$ and a sum-up via inter-processor communications, like AllReduce in MPI [22].
- Look-up & Modification (Line 9 of Figure 2): all the processors check whether the given key $\boldsymbol{x}$ exists in the local hash table $\mathcal{H}_i$. If any of the processors finds the key, it simply updates the value and informs other processors of the existence of the key; otherwise the new entry is inserted to the least-occupied processor.

The above parallelization of packSVM, shown in Figure 3 can be experimentally shown to overwhelm many contemporary kernel SVM tools and run well on up to hundreds of thousands of training samples. We go one step further by introducing the following packing strategy.

Packing Strategy. Given an integer r,we aim to pack r iterations into a single one, and thus reduce the number of communications by a factor of $O(r)$. Notice that the total bits in communication will not be reduced in our proposed strategy. Nevertheless, the reduction of communication frequency speeds up the algorithm significantly, as to be shown in Section IV.C.

We use notations $\boldsymbol{w}_t, \boldsymbol{x}_t, y_t$ to denote the predictor $\boldsymbol{w}$ and the random sample $(\boldsymbol{x}_t, y_t)$ in the t^{th} iteration. Considering equation (8) and (9), the calculation from $\boldsymbol{w}_{t-1}$ to $\boldsymbol{w}_t$ actually needs no more than two scaling processes and one additional term. For the sake of simplicity, we combine them and write the recursive formula implicitly, where a_t, b_t are calculated from $\boldsymbol{w}_{t-1}, \boldsymbol{x}_{t-1}, y_{t-1}$:

$$\boldsymbol{w}_t = a_t\boldsymbol{w}_{t-1} + b_t\phi(\boldsymbol{x}_t) \tag{11}$$

In the iteration t, we need to calculate $y_t' = \langle \boldsymbol{w}_t, \phi(\boldsymbol{x}_t)\rangle$, but $\boldsymbol{w}_t$ is dependent on the previous iteration, since a_t and b_t can only be calculated in iteration $t-1$. At first glance, this suggests it is
unrealistic to calculate an iteration before the previous one ends. Next, we will show how to calculate $y_t', \dots y_{t+r-1}'$ simultaneously.

As illustrated in Figure 4, we expand the formula of $y_i' = \langle \boldsymbol{w}_i, \phi(\boldsymbol{x}_i)\rangle$ to terms of $w_t, \phi(x_t), \dots \phi(x_i)$, for $i = t \dots t + r - 1$, and hide those complex coefficients. One can see that although coefficients ■ are unknown at the iteration t, we can pre-calculate the time-consuming part $\langle \boldsymbol{w_t}, \phi(\boldsymbol{x_i})\rangle$ for $i = t, \dots, t + r - 1$ all together at iteration t. Besides, the pair-wise values $\mathcal{K}(\boldsymbol{x_i}, \boldsymbol{x_j})$ for $t \le i < j \le t + r - 1$ can also be pre-processed in a distributed manner. This all needs two communication requests like AllReduce in MPI. We summarize our packing algorithm for r consecutive iterations $t, \dots, t + r - 1$ as follows:

- Pre-calculate $y'_i = \langle \boldsymbol{w_t}, \phi(\boldsymbol{x_i}) \rangle$ for $i = t \dots t + r - 1$
- Pre-calculate $\mathcal{K}(\boldsymbol{x_i}, \boldsymbol{x_j})$ for $t \le i < j \le t + r - 1$
- Iterate i through t to $t + r - 1$ and process the i^{th} iteration as before. Whenever iteration i is finished, a_i, b_i can be calculated and $y'_{i+1}, \dots y'_{t+r-1}$ are updated offline (without communication):
$$y'_{i+j} \leftarrow a_i y'_{i+j} + b_i \mathcal{K}(\boldsymbol{x_{i+j}}, \boldsymbol{x_i})$$
- Update the distributed hash table $\mathcal{H}$ after all r iterations finish, by communicating to confirm the existing entries, and then add new entries to the least occupied processors.

We provide the pseudo-code of P-packSVM with the packing strategy in Appendix. We remark on the coefficient r that it is not the larger the better. As one may see from the pre-calculation of $\mathcal{K}(\boldsymbol{x_i}, \boldsymbol{x_j})$, it needs $O(r^2 d/p)$ in time, assuming d to be the feature dimension. If this time exceeds the communication cost saved by the packing strategy, the acceleration will be undermined. We will practically show that $r = 100$ is a good parameter in Section IV.C.

C. Comparisons

After introducing our algorithm, P-packSVM (based on SGD method), we are ready to compare it with other sequential or parallel trainers mentioned in Section II, for the large scale kernel SVM training.

Accuracy. The prediction accuracy is associated with two factors: how well we optimize the objective and how well the objective is related to the accuracy. Since we are only considering the SVM trainers, we ignore the latter and only pay attention here the former – the optimization error.

First of all, IPM and SMO algorithms both focus on the dual objective, but Hush et al [10] proved that this dual approach converges slowly in the desired primal objective. On the contrary, our P-packSVM directly optimizes on the primal. If the algorithm terminates early, the optimization on the primal produce better solution than on the dual. Secondly, we consider the optimization effectiveness – how fast each algorithm converges to its own objective. IPM and SMO algorithms do well in this aspect, since the predictor always goes closer to the optimal in every step. SGD algorithms do not have such property and the accuracy fluctuate as the number of iterations increases. However, the strong convexity [23] [24] ensures that P-packSVM achieves good accuracy, as we analyzed in Section III.A.

Speed on a single machine. SGD algorithms are the fastest for linear SVM [25], and SMO algorithms are generally believed the fastest for non-linear kernels [12], while IPM algorithms fall far behind. Few of the papers substantially address the incorporation of kernels in SGD, because before the introduction of the best known learning rate in [24], SGD algorithms like [16] take a much longer time than SMO. We will show in Section IV.A that our SGD algorithm, P-packSVM, can achieve similar efficiency as SMO on a single machine.

Parallel speed-up. Regarding the parallel capability, we need to consider the following two factors:

- The communication cost.
  With the increasing number of processors, communications start to become the bottleneck, so the *algorithm* that invokes fewer communications shows its superiority. Under such a magnitude, IPM algorithms take the lead, for their number of iterations is logarithmic to $1/\epsilon$ [27], while SMO and SGD both experience a large number of iterations. In this paper we turn the tide and reduce the number of communications requests by a factor of r. This makes our P-packSVM highly parallelized.
- The parallel efficiency - Amdahl's law. [2]
  The law states that a small portion of the program that cannot be parallelized will limit the overall speed-up. In the view-point of Amdahl's law, IPM algorithms are the most difficult to be parallelized, due to its complex matrix operations. In PSVM [5] there exists a small scale Cholesky *factorization* in each iteration that cannot be parallelized, which becomes the bottleneck as to be shown in Section IV.C. SMO algorithms are relatively easier but need modification, like [3]. Our proposed P-packSVM has highly parallelized each iteration, except for only a constant number of commands, and thus attains the potential to reach high scalability.

Compare with PSVM. We pay special attention to the comparisons with our well-matched adversary PSVM. Firstly, in order to achieve an endurable speed, PSVM forces an approximation to the kernel matrix. This approximation, by Incomplete Cholesky Factorization, lacks theoretical error bounds. We empirically show in the next section that this decomposition is not accurate enough in many datasets. On the contrary, though in stochastic manner, the theoretical convergence analysis on P-packSVM guarantees good accuracy. Secondly, as previously stated, PSVM optimizes the dual objective while our P-packSVM directly optimizes on the primal. Thirdly, the parallel speed-up of PSVM cannot achieve the height of P-packSVM, due to Amdahl's law mentioned above. Fourthly, the memory requirement for PSVM is as high as $O(m^{1.5}/p)$, while P-packSVM uses only $O(m/p)$ for each processor, making memory no longer a bottleneck for the algorithm.

IV. EXPERIMENTS

In this section we perform experiments on training sets varying in size from 1,000 to 8,000,000 samples. We use 144 equally configured machines in our data center, where each machine is equipped with two 2.5GHz Intel Xeon CPUs with a total of eight cores and a memory of 16GB. We use the Message-Passing Interface (MPI) as our parallel platform [22]. We first introduce the binary classification datasets in the experiments:

- *CCAT* dataset, retrieved from *RCV1* collection [18]. The samples are scaled by the author and have a sparsity of 0.16%.
- *CovType* dataset, prepared by J. T-Y Kwok [17]. No normalization has been performed on this dataset, and it has 54 features in total.

TABLE I. COMPARISONS ON THE TRAINING TIME. #PROCESSORS IS ONLY APPLIES TO PSVM AND P-PACKSVM.

Data set	#samples(train/test)	#features	#processors	SVM-light	PSVM	P-pack 1	P-pack 1.5	P-pack 2
Splice[1]	1,000 / 2,175	60	8	0.3s	0.6s	2s	3s	4s
Adult[3]	32,561 / 16,281	123	128	1103s	12s	5s	8s	12s
Web[3]	49,749 / 14,951	300	128	2483s	17s	8s	14s	19s
CovType[3]	522,910 / 58,102	54	256	280101s	748s	321s	574s	864s
CCAT[3]	781,265 / 23,149	47,236	256	219744s	18173s	918s	1741s	2739s
RCV1-All[3]	781,265 / 23,149	47,236	256	3819441s	74888s[4]	32363s	55323s	79686s
MNIST8m[2]	8,000,000 / 10,000	784	512	-	-	12880s	41866s	145248s

[1] We used $T = 10m, 15m, 20m$ for P-pack 1 / 1.5 / 2 resp., and $m' = 0.1m$ for PSVM.
[2] We used $T = m/8, m/4, m/2$ for P-pack 1 / 1.5 / 2 resp. In this set, both SVM-light and PSVM fail to run within ten days.
[3] For the rest of the datasets, we used $T = m, 1.5m, 2m$ for P-pack 1 / 1.5 / 2 resp.
[4] We forced to use $m' = m^{0.4}$ instead of $m^{0.5}$ to reduce PSVM's running time.

- *Splice* / *Web* / *Adult* prepared by the libSVM project team [8]. They are three relatively small datasets.
- *RCV1-All*, the entire 103 categories in RCV1 topics collection. This is a multi-label problem and we consider it as 103 binary classifications and add the correct / incorrect predictions together to verify the accuracy.
- Class 2 in the *MNIST8m* dataset, prepared by the libSVM project team [8]. This set contains 8.1 million samples and was generated [19] by performing careful elastic deformation of the original *MNIST* training set. We use the scaled version, with values in [0,1].
  We use the first 8 million samples as training data, and prepare two sets of testing data: the last 100,000 samples in *MNIST8m*, and the 10,000 samples in the original *MNIST* testing set.

For convenience, throughout the experiments we stick to the Gaussian rbf kernel

$$\mathcal{K}(x_1, x_2) = \exp\left(-rbf \cdot \|x_1 - x_2\|_2^2\right)$$

though our proposed algorithm can deal with arbitrary kernels like the polynomial kernel, Laplacian kernel, etc.

A. Performance Test

In the first experiment we compare the running time and the accuracy of our proposed P-packSVM against two state-of-the-art SVM trainers: SVM-light [12] and PSVM [5]. For SVM-light we use its default convergence parameters. For PSVM we set a gap threshold and the residual (primal & dual) threshold to 0.1, and an upper limit of 1000 iterations. Unless otherwise state, we adopt the suggested $m' = m^{0.5}$ approximation (see Section II), which was claimed to balance the accuracy and the efficiency in [5].
For all of the test sets, we choose the best selected $\sigma(C = 1/m\sigma$ in SVM-light) and rbf for the Gaussian kernel (see Appendix for a detailed configuration). These parameters are equally set to the three trainers. In our P-packSVM, we set $T = m, 1.5m, 2m$ as three different iteration limits, and notate them as *P-pack 1*, *P-pack 1.5* and *P-pack 2*. The program runs three times and the mean accuracy, mean number of support vectors and mean training time are calculated. We give special regard to the fact that for the small training set *splice*, neither PSVM nor P-packSVM can achieve reasonable accuracy under the above configurations, and thus we choose $m' = 0.1m$ for PSVM and $T = 10m, 15m, 20m$ for P-packSVM.

Considering the training time in TABLE I. Our poposed method is undoubtedly the fastest for large-scale learning. Notice that the column "#processors" applies to both PSVM and P-packSVM, while SVM-light is a sequential SVM trainer. We conclude that P-packSVM is hundreds of times faster than SVM-light and several times faster than PSVM for large datasets like *CovType* and *CCAT*. Notice that, by performing simple multiplication, one can see even in a single machine, our proposed P-packSVM may achieve a similar speed as SVM-light for large scale data.

The number of support vectors in our model is the smallest among the three (Figure 5), partially because the number of iterations is limited and some samples are not selected in the entire execution of P-packSVM. The accuracy report in Figure 6 demonstrates that our proposed method can get accuracy very close to SVM-light's, and overwhelm the state-of-the-art trainer PSVM on datasets except *CovType*. We remark here that the approximation – incomplete Cholesky decomposition – makes PSVM not accurate enough for datasets with large-rank kernel matrices, like *CCAT*.

RCV1-All. We test on a sequential of 103 labels in *RCV1*, and add the number of correct / incorrect instances together. We pay special attention to this test because most of the labels are extremely biased (number of negative samples dominate). Results in Figure 7 show that our proposed P-packSVM can handle this situation successfully. In sharp contrast, PSVM receives no more than 50% in the F_1 measure [11] (we use $m' = m^{0.4}$ to make PSVM stop in several days).

MNIST8m. We emphasize that our proposed method can run against the very large scale dataset *MNIST8m* with 8 million training samples. For the lack of computing resources, we did not spend extra time choosing the best fit σ and rbf (see Appendix). To the best of our knowledge, no generic kernel SVM trainer has claimed its success on training this dataset. [19] used the invariance property of *MNIST8m* and achieved an accuracy of 99.33% for all 10 classes in 8 days (predicting

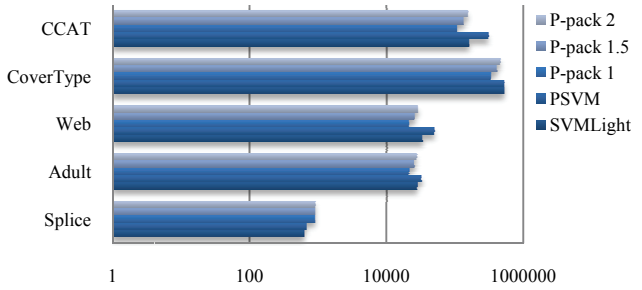

Figure 5. Comparisons on # support vectors.

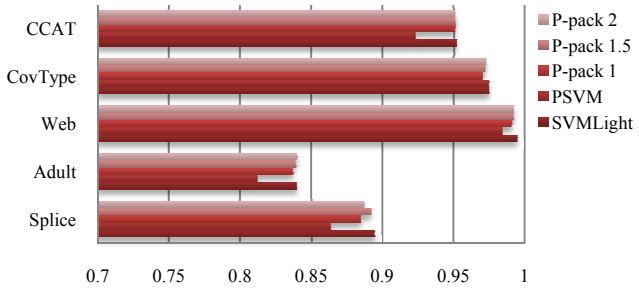

Figure 6. Comparisons on the accuracy.

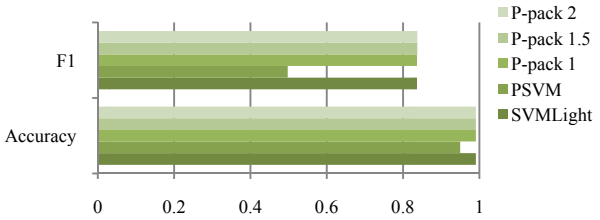

Figure 7. Comparisons on *RCV1-All*.

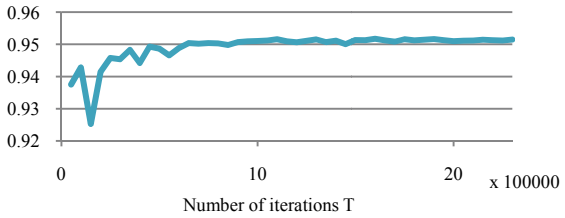

Figure 8. *CCAT* accuracy with *T*

on the original *MNIST* test set). In our experiment, we set $T = m/8, m/4, m/2$ and obtained an accuracy of 99.49%, 99.54% and 99.57% on the class 2 of the original *MNIST* testing set, and an accuracy of 100%, 100% and 100% for the last 100,000 samples in *minst8m* as the testing set. The running time is shown in TABLE I. Although without a good competitor, our results show the competence of P-packSVM on million scale training.

B. Convergence Test

In the second experiment, we analyze the accuracy curve with respect to T - the number of iterations spent. The experiment is conducted on the *CCAT* dataset and the prediction accuracy is calculated every 50,000 iterations. As shown in Figure 8, the accuracy exceeds 94.5% only after around 200,000 iterations (1 minute for 256 processors). This chart also helps the user to actively decide the number of iterations on request. We notice that although equipped with a stochastic method, the stability of our proposed P-packSVM with respect to T is still sufficient.

C. Scalability Test

In the third experiment we run PSVM and P-packSVM on *CovType* and *CCAT*, and measure the elapsed training time with a different number of processors $p = 8$, 16, 32, 64, 128, 256, 512. We define the parallel speed-up measurement as the following:

$$\text{speed\_up} = \frac{\text{time for 8 processors}}{\text{time for p processors}} \times 8 \quad \text{(PSVM)}$$

$$\text{speed\_up} = \frac{\text{time for 8 processors,r=100}}{\text{time for p processors}} \times 8 \text{ (P-packSVM)}$$

We use the 8-processor results as the baseline (the numerator in the above equation), since both PSVM and P-packSVM experience close-to-linear speed-up below 8 processors. For P-packSVM we use the 8-processor $r = 100$ running time to be the baseline for $r = 1{,}10$, because one eighth of $r = 100$ is much alike to the single-processor task for $r = 1{,}10$ in speed.

We summarize our scalability test results below: the running time in seconds is shown in TABLE II. , and the speed-up values are illustrated in Figure 9 and Figure 10. It can be seen that without our packing strategy, the training time increases when the number of processors exceeds 256. What is worse, the speed-up does not exceed 100 times for both *CCAT* and *CovType*. With the help of packing, we obtain a speed-up of 295 times and 416 times for *CCAT* and *CovType* respectively, with 512 processors. On the contrary, PSVM gains a parallel speed-up of no more than 180 times for both *CCAT* and *CovType*. Thus, PSVM falls far behind our proposed P-packSVM in efficiency for all values of p.

V. Enhancement

In this section we dialectically analyze the limitation of our P-packSVM, and propose some enhancements.

Bias term. P-packSVM does not incorporate a bias term in the objective (2). The naive integration of a variable b will result in a theoretical challenge of the convergence rate $T = \tilde{O}(1/\sigma\delta\epsilon)$, because the linear dependency on b in the objective breaks the strong convexity [24]. The best solution to this is to enable a slightly different regularizer $(\|w\|_2^2 + b^2)/2$ to ensure the strong convexity, and at the same time stay close to the original objective. In the experiment, we tested this new regularizer and results show a constant factor (about 2) in running time increase.

Convergence Criterion. Our algorithm lacks a convergence criterion. It is true that without a dual view of the problem it lacks an explicit measurement of the convergence, like the duality gap. If we insist on calculating the primal objective, an extra factor of m will appear in the time complexity. We propose a substitute that the average hinge loss over for example 1000 random samples can be calculated (this needs

TABLE II. SCALABILITY TEST FOR PSVM AND P-PACKSVM. (MEAN TIME OF THREE RUNS, $T = n$)

Data set	Algorithm	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$	$p = 256$	$p = 512$
CCAT	PSVM	278780s	138246s	71933s	46989s	31235s	18313s	12917s
CCAT	P-packSVM, $r = 1$	30599s	15976s	8528s	4793s	2928s	2570s	3282s
CCAT	P-packSVM, $r = 10$	29308s	14734s	7386s	3631s	1930s	1122s	1265s
CCAT	P-packSVM, $r = 100$	28061s	13838s	6953s	3307s	1552s	917s	761s
CovType	PSVM	14294s	7099s	5626s	2866s	1342s	934s	1587s
CovType	P-packSVM, $r = 1$	13224s	6374s	3144s	1529s	1144s	1128s	1346s
CovType	P-packSVM, $r = 10$	12895s	6014s	2728s	959s	544s	390s	389s
CovType	P-packSVM, $r = 100$	12267s	5710s	2611s	924s	514s	316s	236s

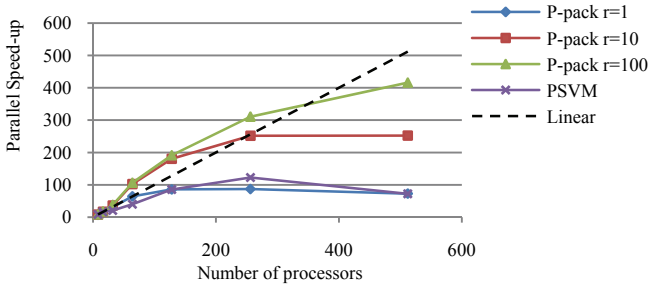

Figure 9. CovType speed-up.

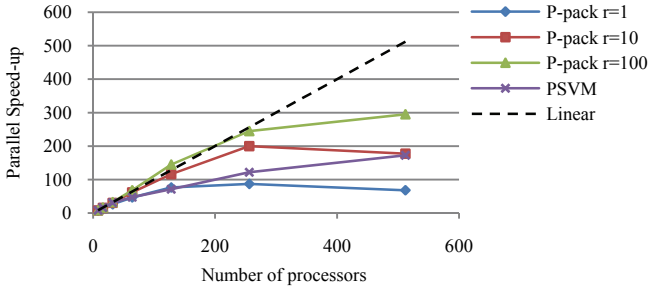

Figure 10. CCAT speed-up.

no extra effort since our algorithm in Figure 1 already calculates the hinge loss for random examples), and if this value is numerically stable enough, the program can automatically stop. Experimental results show that the iteration limit T is around the sample size m.

Extension to other loss. We emphasize that our proposed algorithm can be easily generalized to convex loss functions, other than the hinge loss. For example, L_2 Kernel Logistic Regression can be similarly solved where we only need to slightly change ∇f_t in Equation (7). We have shown that the convergence rate still holds in a counterpart of this paper [30]. We will perform this research in our further work.

VI. CONCLUSION

This paper analyzes a stochastic gradient descent method that optimizes the primal SVM objectives for arbitrary kernels. Parallel implementation is provided by introducing a distributed hash table and the innovative packing strategy. The proposed algorithm, P-packSVM, averagely distributes the support vector to all processors, and the r consecutive gradient descent steps can be packed with constant times of communications requests. We emphasize that this packing strategy compensates for the defect of SGD – large communication cost, and is effective in increasing the parallel speed-up.

We conduct extensive experiments on benchmark datasets that vary in size, in sparsity and in the number of features. Extensive experimental results show that our proposed algorithm can run much faster than the state-of-the-art parallel SVM trainer PSVM [5], and hundreds of times faster than the sequential trainer SVM-light. For example, P-packSVM trains *CovType* with 500k samples in 4 minutes and *CCAT* with 800k samples in 13 minutes. We emphasize that P-packSVM attains accuracy that is sufficiently close to SVM-light, and prevails over that of *PSVM*.

ACKNOWLEDGMENT

Zeyuan Allen Zhu wants to thank Shai Shalev-Shwartz of Hebrew University for his valuable discussions, Teng Gao from Tsinghua University for his construction and maintenance on our parallel platform, and Zhijie Ren from Peking University for her comparable experiments on PSVM. Zeyuan Allen Zhu is partially supported by the National Innovation Research Project for Undergraduates (NIRPU).

The authors also acknowledge Matt Callcut and all three anonymous reviewers for their fruitful comments.

REFERENCES

[1] Mark A. Aizerman, Emmanuel M. Braverman, and Lev I. Rozonoér, "Theoretical foundations of the potential function method in pattern recognition learning," *Automation and Remote Control 25*, 1964.

[2] Gene Amdahl, "Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities," in *AFIPS*, 1967, pp. 483-485.

[3] Li Juan Cao et al., "Parallel sequential minimal optimization for the training of support vector machines," *IEEE Transactions on Neural Networks*, vol. 17, no. 4, pp. 1039-1049, July 2006.

[4] Chih-Chung Chang and Chih-Jen Lin. (2001) LIBSVM: a library for support vector machines. http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[5] Edward Y. Chang, Kaihua Zhu, Hao Wang, and Hongjie Bai, "PSVM: Parallelizing Support Vector Machines on Distributed Computers," in *NIPS*, 2007, Software available at http://code.google.com/p/psvm.

[6] Nello Cristianini and John Shawe-Taylor, *An introduction to support vector machines*.: Cambridge University Press, 2000.

[7] Osuna Edgar, Robert Freund, and Federico Girosi, "An improved

algorithm for support vector machines," in *IEEE Signal Processing Society Workshop*, 1997.

[8] Rong-En Fan. LIBSVM Data: Classification, Regression, and Multi-label. [Online]. http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/

[9] Shai Fine and Katya Scheinberg, "Efficient SVM Training Using Low-Rank Kernel Representations," *JMLR*, pp. 243-264, 2001.

[10] Don Hush, Patrick Kelly, Clint Scovel, and Ingo Steinwart, "QP Algorithms with Guaranteed Accuracy and Run Time for Support Vector Machines," *JMLR*, vol. 7, pp. 733-769, 2006.

[11] Thorsten Joachims, *Learning to Classify Text Using Support Vector Machines*.: Kluwer Academic Publisher, 2002.

[12] Thorsten Joachims, "Making large scale SVM learning practical," in *Advances in Kernel Methods - Support Vector Learning*.: MIT Press, 1998.

[13] Thorsten Joachims, "Optimizing search engines using clickthrough data," in *SIGKDD*, 2002.

[14] Thorsten Joachims, "Training Linear SVMs in Linear Time," in *KDD*, 2006.

[15] Sham Kakade and Shai Shalev-Shwartz, "Mind the Duality Gap: Logarithmic regret algorithms for online optimization," in *NIPS*, 2009.

[16] Jyrki Kivinen, Alexander J. Smola, and Robert C. Williamson, "Online Learning with Kernels," in *IEEE Transactions on Signal Processing*, 2004.

[17] James Tin-Yau Kwok. CovType classification data. [Online]. http://www.cs.ust.hk/~jamesk/data/forest.zip

[18] David D. Lewis, Yiming Yang, Tony G. Rose, and Fan Li, "RCV1: A New Benchmark Collection for Text Categorization Research," *Journal of Machine Learning Research*, vol. 5, pp. 361-397, 2004.

[19] Gaëlle Loosli, Stéphane Canu, and Léon Bottou, "Training Invariant Support Vector Machines using Selective Sampling," in *Large Scale Kernel Machines*.: MIT Press, 2007, pp. 301-320.

[20] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze, *Introduction to Information Retrieval*.: Cambridge University Press, 2008.

[21] Sanjay Mehrotra, "On the Implementation of a Primal-Dual Interior Point Method," *SIAM Journal on Optimization*, vol. 2, no. 4, pp. 575-601, November 1992.

[22] MPI Documents. [Online]. http://www.mpi-forum.org/docs/

[23] Shai Shalev-Shwartz, "Online Learning: Theory, Algorithms, and applications," The Hebrew University, PhD Thesis 2007.

[24] Shai Shalev-Shwartz, Yoram Singer, and Nathan Srebro, "Pegasos: Primal Estimated sub-GrAdient SOlver for SVM," in *ICML*, 2007.

[25] Shai Shalev-Shwartz and Nathan Srebro, "SVM Optimization: Inverse Dependence on Training Set Size," in *ICML*, 2008.

[26] Vladimir Vapnik, *The Nature of Statistical Learning Theory*.: Springer-Verlag, 1995.

[27] Stephen J. Wright, *Primal-dual interior-point methods*.: SIAM, 1997.

[28] Gaetano Zanghirati and Luca Zanni, "A parallel solver for large quadratic programs in training support vector machines," *Parallel Computing*, vol. 29, no. 4, April 2003.

[29] Tong Zhang, "Solving Large Scale Linear Prediction Problems Using Stochastic Gradient Descent Algorithms," in *ICML*, 2004.

[30] Zeyuan Allen Zhu et al., "Inverse Time Dependency in Regularized Learning," in *ICDM*, 2009.

APPENDIX

PROCESSOR i

1. INPUT: σ, T, r, training sample space Ψ
2. INITIALIZE: $\mathcal{H}_i = \emptyset$, $s = 1, norm = 0$
3. FOR $t = 1,2, \dots, T/r$
4. Randomly pick up r samples $(\boldsymbol{x}_1, y_1) \dots (\boldsymbol{x}_r, y_r) \in \Psi$. Ensure all processors receive the same samples.
5. FOR $k = 1, \dots r$ DO
6. $y_{i,k}{}' \leftarrow s\langle \boldsymbol{v_k}, \phi(\boldsymbol{x_i}) \rangle$ by iterating all entries in $\mathcal{H}_i$
7. Communicate with other processors to get $y'_k = \sum_i y_{i,k}{}'$
8. Calculate $pair_{i,j} = \mathcal{K}(x_i, x_j)$ in distribution
9. $LocalSet \leftarrow \emptyset$
10. FOR $k = 1, \dots r$ DO
11. $s \leftarrow (1 - 1/t)s$
12. FOR $l = k + 1 \dots r$ DO $y'_l \leftarrow (1 - 1/t)y'_l$
13. IF $y_k y'_k < 1$ THEN
14. $norm \leftarrow norm + \frac{2y_k}{\sigma t} \cdot y'_k + \left(\frac{y_k}{\sigma t}\right)^2 pair_{k,k}$
15. $LocalSet \leftarrow LocalSet \cup \left\{(\boldsymbol{x_k}, \frac{y_k}{\sigma t}/s)\right\}$
16. For $l = k + 1 \dots r$ DO $y'_l \leftarrow y'_l + \frac{y_k}{\sigma t} \cdot pair_{k,l}$
17. IF $norm > 1/\sigma$ THEN
18. $s \leftarrow s \cdot \frac{1}{\sqrt{\sigma \cdot norm}}$; $norm \leftarrow 1/\sigma$
19. FOR $l = k + 1 \dots r$ DO $y'_l \leftarrow (1 - 1/t)y'_l$
20. Update $\mathcal{H}_i$ according to $LocalSet$, for those elements reported not existed in $\mathcal{H}_1 \dots \mathcal{H}_p$, add them to the least occupied processors.
21. RETURN $s\boldsymbol{v_i}$ by iterating all entries in $\mathcal{H}_1, \dots \mathcal{H}_p$

Figure 11. P-packSVM pseudo-code, with packing strategy

TABLE III. THE PARAMETERS USED IN EXPERIMENTS.

Data set	σ	$C = 1/m\sigma$	rbf	$rank\_ratio = m'/m$ (PSVM)	r (P-packSVM)
splice	0.001	1	0.01	0.1	100
adult	0.0001	0.307116	1	$0.0055418 (m' = m^{0.5})$	100
web	0.00001	2.010091	1	$0.00448341 (m' = m^{0.5})$	100
CovType	0.000005	0.382475	0.002	$0.00138289 (m' = m^{0.5})$	100
CCAT	0.00001	0.127998	1	$0.00113136 (m' = m^{0.5})$	100
RCV1-All	0.00001	0.127998	1	$0.000291287 (m' = m^{0.4})$	100
MNIST8m	0.000001	0.123457	1	-	100