A Parallel Decomposition Solver for SVM: Distributed Dual Ascend using Fenchel Duality

Tamir Hazan Amit Man Amnon Shashua
School of Eng. & Computer Science
The Hebrew University of Jerusalem
Israel

Abstract

We introduce a distributed algorithm for solving large scale Support Vector Machines (SVM) problems. The algorithm divides the training set into a number of processing nodes each running independently an SVM sub-problem associated with its subset of training data. The algorithm is a parallel (Jacobi) block-update scheme derived from the convex conjugate (Fenchel Duality) form of the original SVM problem. Each update step consists of a modified SVM solver running in parallel over the sub-problems followed by a simple global update. We derive bounds on the number of updates showing that the number of iterations (independent SVM applications on sub-problems) required to obtain a solution of accuracy ϵ is $O(\log(1/\epsilon))$. We demonstrate the efficiency and applicability of our algorithms by running on large scale experiments on standardized datasets while comparing the results to the state-of-the-art SVM solvers.

1. Introduction

Statistical learning has become a cornerstone theme in visual recognition paradigms and the Support Vector Machine (SVM) [2, 15] in particular has become a valuable tool in this regard. As typical visual recognition tasks require very large training sets in order to cope with the large variability in the appearance of challenging object classes, there is a need to derive an efficient scalable SVM solver which can be distributed over a cluster of computers. However, SVMs require to solve a constrained quadratic optimization problem which needs resources that are at least quadratic in the number of training examples, thus rendering the tool unwieldy, even for the highest end workstations, for problem sizes in the hundreds of thousands to millions of examples.

We describe and analyze in this paper a simple distributed algorithm, called Parallel Decomposition Solver (PDS), that decomposes the training set of examples into separate subsets each handled independently and in parallel by a computing node. Each computing node solves an SVM problem on a fixed subset of training examples. The separate SVMs are computed iteratively on the same fixed subsets of training examples with a global "correction" vector passed on to all the nodes after each iteration. The machinery behind the algorithm is derived from the principle of Fenchel Duality with a block update dual ascend and is guaranteed to converge to the optimal solution. We show that the number of iterations required in order to achieve an ϵ-accurate solution is $O(\log(1/\epsilon))$.

The benefits of our algorithm are: (i) the local SVMs are generic and can make use of the state-of-the-art SVM solvers, (ii) the scheme applies to both the linear and non-linear kernel cases, (iii) the convergence rate is logarithmic — which is very close to the best convergence rates for sequential Interior Point methods which are doubly logarithmic $O(\log\log(1/\epsilon))$, but which are known to be impractical for large scale SVMs, (iv) unlike previous attempts to parallelize SVMs the algorithm does not make assumptions on the density of the support vectors, i.e., the efficiency of the algorithm holds also for the "difficult" cases where the number of support vectors is very high, and (v) the algorithm behaves well even under small parallelism, i.e., when the number of computing nodes is small.

The latter advantage is especially important as we envision the application of a large scale SVM to run on a small cluster of workstations, rather than on a massively parallel network of computers which are less accessible in conventional environments.

1.1. Previous Work

Due to the central position of SVM in the classification literature, much attention on efficient implementation of SVM was spent, and a variety of methods were introduced and analyzed — including parallel implementations. The different serial approaches can be roughly divided into (i) Interior Point methods, (ii) decomposition methods, and

(iii) gradient-based primal updates. With regard to parallel implementations, the literature is somewhat sketchier but can be divided into two main approaches: (i) cascade SVM, and (ii) parallel implementation of matrix-vector products and ways to distribute matrix blocks to a number of processors. We will briefly highlight below the main points in each category.

Interior Point (IP) methods: IP methods (see for instance [3] and the references therein) replace the constraints with a barrier function. The result is a sequence of unconstrained problems which can be optimized very efciently using Newton or Quasi-Newton methods with a doubly logarithm convergence rate $O(\log\log(1/\epsilon))$. However, the general purpose IP methods have a run time which is cubic, and memory requirements which are quadratic, in the number of examples. Attempts to exploit the special structure of the SVM constraints in IP methods, in order to gain further efficiency, have surfaced but are either focused solely on linear kernels [5] or require specialized assumption like low-rank kernel matrices [6].

Decomposition methods: the most popular approach whereby the solution is sought after in the dual domain and employ an active set of constraints thus working on a subset of dual variables at a time [13, 8, 12]. The various techniques differ in the strategy employed for choosing the dual variables to update and in the size (number of dual variables) to update in each iteration.

Gradient based primal updates: these are methods optimized for linear SVM. The two fastest algorithms are the Pegasus [14] and SVM-perf [10]. Pegasus samples a subset of examples and from the subset selects the margin error points. Those selected points define a sub-gradient for updating the separating hyper-plane. The number of iterations is governed by $O(1/\epsilon)$. SVM-perf is based on a similar principle whereby SVM is applied only to margin errors in a successive manner. Although both algorithms can in principle handle non-linear kernels, the convergence rate results do no longer apply and each iteration becomes considerably more costly thereby compromising their efficiency.

Parallel SVMs: compared to serial SVM solvers, the literature on parallel SVM solvers is relatively small. The most popular line of work falls under the category of "Cascade SVM" (cf. [7, 17]). The general idea is to create a hierarchy (an inverted binary tree) of computing nodes where the top layer perform SVM on their associated subset of examples and the layers below are fed with the support vectors of the layers above — with the bottom layer consisting of a single node. This then proceeds iteratively until convergence (from top to bottom). There are two issues with this approach: first is the tacit assumption that the number of support vectors is relatively small, i.e., the learning problem is "easy", and the second issue with the framework is that the convergence rate is not well defined, i.e., it is not clear how many passes through the tree are necessary per desired accuracy error. Other parallel approaches focus on parallelizing the matrix vector operations of SVM [16] or creating a weighted mixture of SVM solutions [4] — the latter solves a different problem rather than parallelizing the SVM procedure on the original dataset.

2. PDS: the Distributed Dual Ascend Method for SVM

Given a set $\{(\mathbf{x}_i, y_i)\}_{i=1}^m$ of training examples where $\mathbf{x}_i \in R^n$ and $y_i \in \{-1, 1\}$, the linear SVM problem seeks the minimizer of:

$$\min_{\mathbf{w}} \frac{C}{2}\|\mathbf{w}\|_2^2 + \frac{1}{m}\sum_{i=1}^{m} loss(\mathbf{w}; (\mathbf{x}_i, y_i)), \qquad (1)$$

where C tradeoffs the weight given to the margin between the hyperplane $\mathbf{w}$ to the closest examples (the support vectors) to it and the weight given to margin errors (example points which do not lie respectively farther from the support vectors), and

$$loss(\mathbf{w}; (\mathbf{x}_i, y_i)) = \max(0, 1 - y_i \mathbf{w}^\top \mathbf{x}_i).$$

The original SVM problem also includes a bias term b and accommodates high dimensional mappings of the measurement vectors $\mathbf{x}_i$ onto "feature" spaces via the use of Mercer kernels in the dual form (a.k.a nonlinear SVM) — both of which will be discussed later on in Sec. 2.1 and 2.2.

Let $S_1, ..., S_k$ be subsets (not necessarily disjoint) of the training set indices: $S_j \subset \{1, ..., m\}$ such that $S_j \neq \emptyset$ and $\cup_j S_j = \{1, ..., m\}$ where k is a fixed number representing the number of available processors. Let $P_j(\mathbf{d})$ be the solution to:

$$P_j(\mathbf{d}) = \underset{\mathbf{w}}{\operatorname{argmin}} \frac{C}{2k}\|\mathbf{w}\|_2^2 + \mathbf{w}^\top \mathbf{d} + \frac{1}{m}\sum_{i \in S_j} loss(\mathbf{w}; (\mathbf{x}_i, y_i)) \qquad (2)$$

which for $\mathbf{d} = 0$ is the SVM separating hyperplane $\mathbf{w}_j$ of the training data represented by S_j. The role of the vector $\mathbf{d}$ is to tie together the local results $\mathbf{w}_1, ..., \mathbf{w}_k$ as we shall see later. The scheme below iterates over the local solutions $P_j(\cdot)$ and returns the global SVM optimal hyper-plane:

Algorithm 1 (PDS) *Set* $\boldsymbol{\lambda}_j^{(0)} = 0$ *and* $\boldsymbol{\mu}_j^{(0)} = 0$.

```
1. For $t = 1, 2, ..., T$
    (a) Parallel update $j \in \{1, ..., k\}$:
        $$\boldsymbol{\lambda}_j^{(t)} \leftarrow -\boldsymbol{\mu}_j^{(t-1)} - \frac{C}{k} P_j(\boldsymbol{\mu}_j^{(t-1)}) \qquad (3)$$
    (b) message passing $j = 1, ..., k$:
        $$\boldsymbol{\mu}_j^{(t)} \leftarrow -\boldsymbol{\lambda}_j^{(t)} + \frac{1}{k}\sum_{l=1}^{k} \boldsymbol{\lambda}_l^{(t)} \qquad (4)$$
```

Output: $\mathbf{w}^* = -\frac{1}{C}\sum_{j=1}^{k} \boldsymbol{\lambda}_j^{(T)}$.

The derivation of the algorithm is based on the principle of Fenchel Duality and is described briefly in Appendix A. Note that $\boldsymbol{\lambda}_j$ holds a separating hyperplane calculated by the training examples of S_j and a weighted sum of all previous separating hyperplanes. When $t = 1$, $\boldsymbol{\lambda}_j^{(1)}$ equal to the SVM solution over S_j, i.e., the classifier $\mathbf{w}_j$ of the training set S_j. Then $\boldsymbol{\mu}_j^{(1)}$ contain a weighted sum of all those classifiers. The result of the weighted sum is passed onto the next iteration as the vector $\mathbf{d}$ in the operator $P_j(\mathbf{d})$. This completes the description of the algorithm with linear kernels (i.e., $\mathbf{w}$ is represented explicitly) and with zero bias (i.e., $b = 0$). We will address below the details for handling kernels and the extension for non-zero bias.

2.1. Using Mercer Kernels

One of the most appealing benefits of SVM, and most likely the key for its success, is the ability to construct non-linear classifiers using kernels which satisfy Mercer's conditions. The key for making this possible is that $\mathbf{w}$ is represented as a linear combination of the input training data and that the dual form involves only inner-products between the training data points. The common approach for solving SVM when kernels are employed is to switch to the dual problem and find the optimal set of dual variables.

Kernels are incorporated into our algorithm in the same way. The dual form of the operator $P_j(\mathbf{d})$ is:

$$P_j^*(\mathbf{d}) = \underset{0\leq\boldsymbol{\alpha}\leq(1/m)\mathbf{1}}{\operatorname{argmax}} \sum_{i\in S_j} \alpha_i - \frac{k}{2C}\boldsymbol{\alpha}^\top Q_j^\top Q_j\boldsymbol{\alpha} + \frac{k}{C}\mathbf{d}^\top Q_j\boldsymbol{\alpha} \tag{5}$$

where Q_j is the matrix whose columns consists of $y_i\mathbf{x}_i$, $i \in S_j$ and $\boldsymbol{\alpha}_j^* = P_j^*(\mathbf{d})$ is the dual variables vector of the subproblem represented by S_j. The connection between $P_j^*(\mathbf{d})$ and the primal vector $P_j(\mathbf{d})$ is described by

$$P_j(\mathbf{d}) = \frac{k}{C}(Q_j P_j^*(\mathbf{d}) - \mathbf{d}). \tag{6}$$

The entries of the matrix $Q_j^\top Q_j$ are of the form $y_r y_s k(\mathbf{x}_r, \mathbf{x}_s)$ where $k(\cdot)$ is the Mercer kernel. As mentioned above, the vector $\mathbf{d}$ is a linear superposition of the training set of points, i.e., $\mathbf{d} = \sum_r \beta_r y_r \mathbf{x}_r$ where the coefficients β_r represent the integration of results from all the other sub-problems over all previous iterations. Each entry of the vector $\mathbf{d}^\top Q_j\boldsymbol{\alpha}$ is therefore of the form $\alpha_r\beta_s y_r y_s k(\mathbf{x}_r, \mathbf{x}_s)$. Taken together, the dual form of each sub-problem is fully represented in terms of kernels. The dual form is solved using state-of-the-art SVM dual solvers (we used SVM-light [8] following a simple modification for incorporating the extra term $(k/C)\mathbf{d}^\top Q_j\boldsymbol{\alpha}$). Substituting eqn. 6 into the update rules of eqn. 3 and eqn. 4 we obtain:

$$\begin{aligned} \boldsymbol{\alpha}_j^{(t)} &\leftarrow P_j^*(\boldsymbol{\mu}_j^{(t-1)}) \\ \boldsymbol{\mu}_j^{(t)} &\leftarrow Q_j\boldsymbol{\alpha}_j^{(t)} - \frac{1}{k}\sum_{l=1}^{k} Q_l\boldsymbol{\alpha}_l^{(t)} \end{aligned}$$

Note that $\boldsymbol{\mu}_j$ is a superposition of the training set of points and then is fed back into the dual of the sub-problems. After T iterations, the separating hyperplane is obtained by:

$$\mathbf{w} = -\frac{1}{C}\sum_{j=1}^{k} Q_j\boldsymbol{\alpha}_j^{(T)}.$$

2.2. Incorporating a Bias Term

It is often desirable to augment the weight vector $\mathbf{w}$ with a scalar bias term, typically denoted as b. The classifier becomes $\mathbf{w}^\top\mathbf{x} + b$ and the loss is defined accordingly:

$$loss((\mathbf{w}, b); (\mathbf{x}_i, y_i)) = \max(0, 1 - y_i(\mathbf{w}^\top\mathbf{x}_i + b)).$$

The introduction of b into the derivation process described in Appendix A introduces a fatal problem by tying together the $(n+1)$'th coordinate of the $\boldsymbol{\lambda}_j$ through a new constraint $\sum_{j=1}^{k}\lambda_{j,n+1} = 0$ (we omit the underlying reasons as it will require a deeper detour into conjugate duality which we feel is not instrumental to this paper). The global constraint on the conjugate dual variables does not allow for a block update approach — which is the key for making our algorithm work. Therefore, we need to find a plausible work-around which would be as close as possible to the original problem.

We add the term $(C/2)\epsilon b^2$ to the SVM primal criterion function where $\epsilon > 0$ is arbitrarily small:

$$\min_{\mathbf{w},b} \frac{C}{2}\|\mathbf{w}\|_2^2 + \frac{C\epsilon}{2}b^2 + \frac{1}{m}\sum_{i=1}^{m} loss((\mathbf{w}, b); (\mathbf{x}_i, y_i)) \tag{7}$$

We accordingly redefine the sub-problem operator $P_j(\mathbf{d})$:

$$\begin{aligned} P_j(\mathbf{d}) &= \underset{\mathbf{w},b}{\operatorname{argmin}} \frac{C}{2k}\|\mathbf{w}\|_2^2 + \frac{C\epsilon}{2k}b^2 \\ &+ (\mathbf{w}, b)^\top\mathbf{d} + \frac{1}{m}\sum_{i\in S_j} loss((\mathbf{w}, b); (\mathbf{x}_i, y_i)) \end{aligned}$$

The sub-problem dual operator $P_j^*(\mathbf{d})$ becomes:

$$\begin{aligned} P_j^*(\mathbf{d}) &= \underset{0\leq\boldsymbol{\alpha}\leq 1/m}{\operatorname{argmax}} \sum_{i\in S_j}\alpha_i - \frac{k}{2C}\boldsymbol{\alpha}^\top Q_j^\top Q_j\boldsymbol{\alpha} \\ &+ \frac{k}{C}\mathbf{d}^\top Q_j\boldsymbol{\alpha} - \frac{k}{2C\epsilon}\left(\sum_{i\in S_j}\alpha_i y_i - d_{n+1}\right)^2 \end{aligned}$$

Note that for sufficiently small value of ϵ the last term will drive the solution such that $\sum_{i\in S_j}\alpha_i y_i \approx d_{n+1}$.

The operator $P_j(\mathbf{d})$ outputs $\mathbf{w}$ through the equation $(k/C)(Q_j P_j^*(\mathbf{d}) - \mathbf{d})$ and the scalar b (given $\mathbf{w}$ and a support vector $\mathbf{x}$ one can obtain b from the equation $|\mathbf{w}^\top \mathbf{x} - b| = 1$). We denote the process of recovering b by an operator $P_j^b(\mathbf{d})$. The conjugate dual vectors $\mathbf{u}_j$ contain $n+1$ coordnates and are broken into two parts $\boldsymbol{\mu}_j = (\bar{\boldsymbol{\mu}}_j; \mu_j)$ where $\bar{\boldsymbol{\mu}}_j \in R^n$ and μ_j is a scalar. The update rules are as follows (we omit the derivation but note that it follows the derivation found in Appendix A):

$$\begin{aligned}
\boldsymbol{\alpha}_j^{(t)} &\leftarrow P_j^*(\boldsymbol{\mu}_j^{(t-1)}) \\
b_j^{(t)} &\leftarrow P_j^b(\boldsymbol{\mu}_j^{(t-1)}) \\
\bar{\boldsymbol{\mu}}_j^{(t)} &\leftarrow Q_j \boldsymbol{\alpha}_j^{(t)} - \frac{1}{k}\sum_{l=1}^{k} Q_l \boldsymbol{\alpha}_l^{(t)} \\
\mu_j^{(t)} &\leftarrow \mu_j^{(t-1)} - \frac{1}{k}\sum_{l=1}^{k} \mu_l^{(t-1)} + \frac{C\epsilon}{k} b_j^{(t)} - \frac{C\epsilon}{k^2}\sum_{l=1}^{k} b_l^{(t)}
\end{aligned}$$

3. Convergence Rate Analysis

The convergence rate analysis is based on a reduction of the conjugate dual of the optimization (described in Appendix A) to a general form described by:

$$\max_{\boldsymbol{\beta},\ 0 \le \boldsymbol{\alpha} \le \frac{1}{m}\mathbf{1}} -\frac{k}{2C}\|A\boldsymbol{\alpha} + B\boldsymbol{\beta}\|^2 + \mathbf{1}^\top \boldsymbol{\alpha} \qquad (8)$$

With this reduction we could rely on previous work by [11] showing that a block coordinate ascent achieves a linear convergence rate, i.e. it takes $O(\log(1/\epsilon))$ to get ϵ-close to the optimal solution. The details are found in Appendix B.

4. Experiments

The experiments in this section are focused solely on Kernel-SVM as representing the ultimate computational (run-time) challenge compared to linear-SVM where very efficient algorithms are emerging who can handle data-sets of an order of 10^6 data points [14, 10]. For the local SVM solver we used the SVM-light algorithm [8].

A key feature of the PDS architecture is that each processing node uses independent memory and CPU resources with limited communication bandwidth. This is ideal for running SVM on a cluster of workstations with no special infrastructure designed for optimizing parallelism like memory sharing or multi-core CPUs. For example, [16] produce impressive parallel speedup of roughly k but at the price of using a specialized high-performance cluster architecture where the nodes are interconnected by a high-performance switch and where each node consists of a multi-core of 8 processors sharing 16GB of RAM.

The cluster we used for our experiments consists of separate nodes, interconnected via TCP/IP protocol, each with a 2.8GHZ Intel Xeon CPU with Cache size of 512KB and 2GB of RAM. We report results when using $k = 2, 4, 10$ nodes of the cluster.

The expected parallel speedup of PDS using k processing nodes is roughly $k/2$. Each node needs to cope only with part of the design matrix with m^2/k entries (in the worst case when the number of support vectors is very high) — therefore the parallel speed-up just on computing the kernels is k. The communication bandwidth needs to support an order of m (worst case) — the j'th node transmits its $\boldsymbol{\alpha}_j$ vector consisting of m/k entries (worst case), and receives all other $\boldsymbol{\alpha}_l, l \neq j$, vectors consisting of $m(1-1/k)$ entries. Assuming that the parallel speed-up on each local problem is $k^2 - k^{2.5}$ depending on the complexity of the problem, then with the number of iterations of $O(\log(1/\epsilon)) \approx k$ (since k is a small fixed number of order 10 in our experiments) we obtain a total speed-up of $k/2$.

We begin with synthetic data experiments designed to test the parallel speedup under two extreme conditions: one being challenging dataset with a high percentage of support vectors and small margin and the other being a simple problem with the opposite criteria. The stopping conditions for the synthetic problems were the energy level reached by the single processor ($k = 1$) SVM-light energy-based stopping rule.

The "challenging" dataset consisted of $m = 20,000$ 2D points arranged in a "spiral" configuration with coordinates $x = t\cos(t), y = t\sin(t)$ and $t = 80\pi \cdot \mathrm{rand}([10000\ 1])$. With an RBF kernel $k(x,y) = e^{-\|x-y\|^2/2\sigma^2}$ with $\sigma = 1$, and $C = 1/m$, the percentage of support vectors stands on 80%.

Table 1 highlights some key results on PDS running on the Spiral dataset for $k = 2, 4, 10$ processing nodes. For $k = 10$ processing nodes, for example, PDS underwent 1400 iterations where most of the work per local processor was done during the first iteration (2040 msec) with 10-20 msec per each subsequent iteration totaling 14630 msec spent on the local SVMs. The integration time spent after the first iteration is 19560 msec where most of the work goes into computing the kernels between points assigned to the node and other remaining points. Integration time on subsequent iterations take 10-20 msec due to retrieval from RAM of pre-computed kernel computations and extensive re-use of previous samples made by SVM-light. The total time spent on integration is 36590 msec making the overall parallel speedup stand on 4.7 which is slightly less than the expected $k/2$ speedup.

The PDS profile of behavior is dramatically different for an "easy" problem. Table 2 shows the same performance entries but on a 2D training set generated by two Gaussians where the distance between the means is ten times their variance. The dataset contains $m = 20,000$ points; the kernel used is an RBF with $\sigma = 1$; $C = 1/m$ and the percentage of support vectors stands on 4%. Considering the case of $k = 10$, we see that the number of iterations is small (14 compared to 1400 for the Spiral) and the overall parallel speedup stands on 3.54. The speedup is smaller than for the difficult scenario of the Spiral dataset mainly because the SVM-light on a single processor samples much less than m points in order to converge making the integration cycles of PDS somewhat non-efficient.

We next move to two real datasets (i) class 1(out of 8) in the Forest Cover Type data set[1] with $m = 300,000$ (sampled out of 581012), RBF kernel with $\sigma = 1.7$ and $C = 1/10m$, and (ii) class 8 in the MNIST database of handwritten digits[2] contains 784-dimensional nonbinary sparse vectors; the data set size is $m = 60000$ and we used the RBF kernel with $\sigma = 1800$ and $C = 1/10m$.

The PDS uses one of two stopping criteria one being energy-based and the other is generalization based on leave-one-out procedure. The energy-based stopping rule has each node monitor the value of its respective dual energy and raise a flag when the energy flattens out over a number of iterations. The leave-one-out stopping rule estimates the generalization error of (by each node separately) by the leave-one-out procedure [9] at predetermined intervals and stopping when the accuracy flattens out. PDS stops when the earlier of the two stopping conditions have been met.

Tables 3, 4 show the PDS results (using the same format as the previous tables) of the two datasets. The MNIST stopped when generalization error flattened on 0.44% and Cover-Type stopped when the dual energy flattened. Cover-Type had significantly a higher percentage of support vectors (45%) than MNIST (5.7%) and thus achieved a higher parallel speedup for $k = 10$ nodes (6.36 compared to 3.45).

5. Summary

We have introduced a simple and efficient parallel decomposition algorithm for SVM where each computing node is responsible for a fixed and pre-determined subset of the training data. The results of the sub-problem solutions are sum-weighted and sent back to the computing nodes in an iterative fashion. We derived the algorithm from the principles of convex conjugate duality as a parallel block-update coordinate ascent. The convergence rate of the algorithm has been analyzed and shown to scale gracefully with the required accuracy ϵ by $O(\log(1/\epsilon))$. The algorithm is mostly appropriate for computing clusters of independent nodes with independent memory and limited communication bandwidth. Experimental results on Kernel-SVM on synthetic and real data show an approximate parallel speedup of $k/2$ when using k processing nodes. Further study is required for evaluating the effects of RAM size on the performance of PDS. Generally, the smaller the RAM size compared to the size of the dataset less caching can be done on pre-computed kernel computations which invariably will have a detrimental effect on the performance of PDS.

[1] Available at http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

[2] Available in http://yann.lecun.com/exdb/mnist

References

[1] D. P. Bertsekas, A. Nedić, and A. E. Ozdaglar. *Convex Analysis and Optimization*. Athena Scientific, 2003. 6

[2] B. Boser, I. Guyon, and V. Vapnik. A training algorithm for optimal margin classifier. In *Proc. 5th Workshop on Computational Learning Theory*, pages 144–152, 1992. 1

[3] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004. 2

[4] R. Collobert, S. Bengio, and Y. Bengio. A Parallel Mixture of SVMs for Very Large Scale Problems, 2002. 2

[5] M. C. Ferris and T. S. Munson. Interior-point methods for massive support vector machines. *SIAM J. on Optimization*, 13(3):783–804, 2002. 2

[6] S. Fine and K. Scheinberg. Efficient SVM training using low-rank kernel representations. *Journal of Machine Learning Research*, 2(243264):20, 2001. 2

[7] H. Graf, E. Cosatto, L. Bottou, I. Dourdanovic, and V. Vapnik. Parallel Support Vector Machines: The Cascade SVM. *Advances in Neural Information Processing Systems*, 17:521–528, 2005. 2

[8] T. Joachims. Making large-Scale SVM Learning Practical. Advances in Kernel Methods-Support Vector Learning. *B. Scoelkopf, C. Burges, A. Smola*, 1999. 2, 3, 4

[9] T. Joachims. Estimating the Generalization Performance of a SVM Efficiently. *international conference on Machine Learning* (ICML), pages 431-438, San Francisco 2000. 5

[10] T. Joachims. Training linear SVMs in linear time. *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 217–226, 2006. 2, 4

[11] Z. Luo and P. Tseng. On the convergence of a matrix splitting algorithm for the symmetric linear complementarity problem. *SIAM J. on Control and Opt.*, 29(5):1037–1060, 1991. 4, 8

[12] E. Osuna, R. Freund, F. Girosi, et al. Training support vector machines: an application to face detection. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 24, 1997. 2

[13] J. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. *Advances in Kernel Methods-Support Vector Learning*, pages 185–208, 1999. 2

[14] S. Shalev-Shwartz, Y. Singer, and N. Srebro. Pegasos: Primal Estimated sub-GrAdient SOlver for SVM. *Proceedings of the 24th international conference on Machine learning*, pages 807–814, 2007. 2, 4

[15] V. Vapnik. *The nature of statistical learning*. Springer, 1995. 1

[16] L. Zanni, T. Serafini and G. Zanghirati. Parallel Software for Training Large Scale Support Vector Machines on Multiprocessor Systems. *Journal of Machine Learning Research* 7:14671492, 2006. 2, 4

Processors	Iterations	SVM-light $t=1$	SVM-light $t=2$	Total SVM	Integration $t=1$	Integration $t=2$	Total Integration	Parallel Speed-up
$k=2$	900	60000	1200	107440	50000	20	71630	1.34
$k=4$	1120	14410	30	40760	45390	20	64220	2.29
$k=10$	1400	2040	10	14630	19560	10	36590	4.7
(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)	(i)

Table 1. Performance analysis of PDS on the "Spiral" set of $20,000$ points (80% of the points are support vectors). (a) number of processors, (b) number of iterations T; (c),(d) time in msec spent on the local SVM-light during the first two iterations, (e) total time spent spent on local SVM-light over T iterations; (f),(g) time spent on the integration phase (communication and Kernel evaluations) during the first two iterations, (h) total time spent spent on integration over T iterations, and (i) the parallel speed achieved (should be compared to $k/2$).

Processors	Iterations	SVM-light $t=1$	SVM-light $t=2$	Total SVM	Integration $t=1$	Integration $t=2$	Total Integration	Parallel Speed-up
$k=2$	6	380	5	403	303	2	314	1.49
$k=4$	10	135	2	150	317	2	335	2.2
$k=10$	14	32	1	43	233	2	259	3.54

Table 2. Performance analysis of PDS on the "Gaussians" set of $20,000$ points (4% of the points are support vectors). The column description follow the same format as Table 1.

[17] J. Zhang, Z. Li, and J. Yang. A Parallel SVM Training Algorithm on Large-Scale Classification Problems. *Machine Learning and Cybernetics, 2005. Proceedings of 2005 International Conference on*, 3, 2005. 2

A. Derivation of the Distributed Block Update Dual Ascend Algorithm

We derive a parallel block update algorithm for the SVM problem eqn. 1, which is equivalent to:

$$\min_{\mathbf{w}_1=\ldots=\mathbf{w}_k} \frac{C}{2k}\sum_{j=1}^{k}\|\mathbf{w}_j\|_2^2 + \frac{1}{m}\sum_{j=1}^{k}\sum_{i\in S_j} loss(\mathbf{w}_j;(\mathbf{x}_i,y_i)) \tag{9}$$

where $\mathbf{w}_1,...,\mathbf{w}_k$ live in R^n as well. Let $\mathcal{H}$ be the set defined by:

$$\mathcal{H} = \left\{\bar{\mathbf{w}} = (\mathbf{w}_1,...,\mathbf{w}_k) \in R^{nk} \ : \ \mathbf{w}_1 = ... = \mathbf{w}_k\right\}.$$

Let $\delta_{\mathcal{H}}(\bar{\mathbf{w}})$ be the indicator function $\delta_{\mathcal{H}}(\bar{\mathbf{w}}) = 0$ if $\bar{\mathbf{w}} \in \mathcal{H}$ and $\delta_{\mathcal{H}}(\bar{\mathbf{w}}) = \infty$ if $\bar{\mathbf{w}} \notin \mathcal{H}$, and let $h_j(\mathbf{w}_j) = \sum_{i\in S_j} loss(\mathbf{w}_j;(\mathbf{x}_i,y_i))$. Eqn. 9 is equivalent to:

$$\min_{\bar{\mathbf{w}}=(\mathbf{w}_1,...,\mathbf{w}_k)} \left(\frac{1}{m}\sum_{j=1}^{k} h_j(\mathbf{w}_j) + \delta_{\mathcal{H}}(\bar{\mathbf{w}})\right) - \left(-\frac{C}{2k}\sum_{j=1}^{k}\|\mathbf{w}_j\|_2^2\right) \tag{10}$$

which is of the form $\min_{\bar{\mathbf{w}}} f_1(Q\bar{\mathbf{w}}) - f_2(\bar{\mathbf{w}})$ where $Q = [I;I]$, i.e., $Q\bar{\mathbf{w}} = (\bar{\mathbf{w}},\bar{\mathbf{w}})$. From Fenchel Duality (cf. [1], pp. 421-446), we have:

$$\min_{\bar{\mathbf{w}}} f_1(Q\bar{\mathbf{w}}) - f_2(\bar{\mathbf{w}}) = \max_{\bar{\boldsymbol{\lambda}},\bar{\boldsymbol{\mu}}} g_2(\bar{\boldsymbol{\lambda}}+\bar{\boldsymbol{\mu}}) - g_1(\bar{\boldsymbol{\lambda}},\bar{\boldsymbol{\mu}})$$

where g_1, g_2 are convex conjugates of f_1, f_2, respectively, and $\bar{\boldsymbol{\lambda}} = (\boldsymbol{\lambda}_1,...,\boldsymbol{\lambda}_k)$ and $\bar{\boldsymbol{\mu}} = (\boldsymbol{\mu}_1,...,\boldsymbol{\mu}_k)$. Specifically,

$$\max_{\bar{\boldsymbol{\lambda}},\bar{\boldsymbol{\mu}}} g_2(\bar{\boldsymbol{\lambda}}+\bar{\boldsymbol{\mu}}) - g_1(\bar{\boldsymbol{\lambda}},\bar{\boldsymbol{\mu}}) = \tag{11}$$

$$\max_{\bar{\boldsymbol{\lambda}},\bar{\boldsymbol{\mu}}} -\frac{k}{2C}\sum_{j=1}^{k}\|\boldsymbol{\lambda}_j+\boldsymbol{\mu}_j\|_2^2 - \sum_{j=1}^{k} h_j^*(\boldsymbol{\lambda}_j) - \sigma_{\mathcal{H}}(\bar{\boldsymbol{\mu}})$$

Where $\sigma_{\mathcal{H}}(\bar{\boldsymbol{\mu}})$ is the support function of $\mathcal{H}$ (convex conjugate of $\delta_{\mathcal{H}}(\bar{\mathbf{w}})$). We employ a Jacobi-style block-update approach, with a time counter (t), for maximizing the dual vectors $\boldsymbol{\lambda}_j, \boldsymbol{\mu}_j$, $j = 1,...,k$, where at each step we maximize $g_2(\bar{\boldsymbol{\lambda}}+\bar{\boldsymbol{\mu}}) - g_1(\bar{\boldsymbol{\lambda}},\bar{\boldsymbol{\mu}})$ with respect to $\boldsymbol{\lambda}_j$ while keeping the other dual vectors fixed. The value of the dual vector $\boldsymbol{\lambda}_j^{(t)}$ becomes:

$$\boldsymbol{\lambda}_j^{(t)} = \underset{\boldsymbol{\lambda}}{\operatorname{argmax}} \ -\frac{k}{2C}\|\boldsymbol{\lambda}+\boldsymbol{\mu}_j^{(t-1)}\|_2^2 - h_j^*(\boldsymbol{\lambda}) \tag{12}$$

which is of the form (employing FD again) $\max_{\boldsymbol{\lambda}} g_2(\boldsymbol{\lambda}) - g_1(\boldsymbol{\lambda})$ which is equal to $\min_{\mathbf{w}} f_1(\mathbf{w}) - f_2(\mathbf{w})$ where

$$\begin{aligned} f_2(\mathbf{w}) &= -\frac{C}{2k}\|\mathbf{w}\|_2^2 - \mathbf{w}^\top\boldsymbol{\mu}_j^{(t-1)} \\ f_1(\mathbf{w}) &= h_j(\mathbf{w}) \end{aligned}$$

from which we obtain the solution for $\mathbf{w}$ for the j'th subproblem (eqn. 2):

$$\begin{aligned} \mathbf{w}_j^{(t)} \equiv P_j(\boldsymbol{\mu}_j^{(t-1)}) &= \underset{\mathbf{w}}{\operatorname{argmin}} \frac{C}{2k}\|\mathbf{w}\|_2^2 + \mathbf{w}^\top\boldsymbol{\mu}_j^{(t-1)} \\ &+ \ \frac{1}{m}\sum_{i\in S_j} loss(\mathbf{w};(\mathbf{x}_i,y_i)) \end{aligned}$$

Processors	Iterations	SVM-light	SVM-light	Total SVM	Integration	Integration	Total Integration	Parallel
		$t = 1$	$t = 2$		$t = 1$	$t = 2$		Speed-up
$k = 2$	23	192360	8460	209122	162666	4980	176342	1.29
$k = 4$	70	62216	2376	75933	160014	3792	186420	1.9
$k = 10$	106	13169	522	18534	99730	224	126918	3.45

Table 3. Performance analysis of PDS on the MNIST set of $60,000$ points (5.7% of the points are support vectors). The column description follow the same format as Table 1.

Processors	Iterations	SVM-light	SVM-light	Total SVM	Integration	Integration	Total Integration	Parallel
		$t = 1$	$t = 2$		$t = 1$	$t = 2$		Speed-up
$k = 2$	1000	62321803	813354	71202282	11125508	125476	13065440	1.58
$k = 4$	2000	32693808	535729	42807721	5429803	126784	8333806	2.6
$k = 10$	3000	9622014	307999	16742959	1605412	92042	4196778	6.36

Table 4. Performance analysis of PDS on the CoverType set of $300,000$ points (45% of the points are support vectors). The column description follow the same format as Table 1.

The connection between $\boldsymbol{\lambda}_j^{(t)}$ and $\mathbf{w}_j^{(t)}$ follows from Lagrangian optimality:

$$\mathbf{w}_j^{(t)} = \underset{\mathbf{w}}{\text{argmin}}\, \mathbf{w}^\top \boldsymbol{\lambda}_j^{(t)} + \frac{C}{2k}\|\mathbf{w}\|_2^2 + \mathbf{w}^\top \boldsymbol{\mu}_j^{(t-1)}$$

from which we obtain the update equation for $\boldsymbol{\lambda}_j$ (eqn. 3):

$$\boldsymbol{\lambda}_j^{(t)} = -\boldsymbol{\mu}_j^{(t-1)} - \frac{C}{k} P_j(\boldsymbol{\mu}_i^{(t-1)})$$

Next we consider the block update for $\bar{\boldsymbol{\mu}}$. The value of the dual vector $\bar{\boldsymbol{\mu}}^{(t)}$ is equal to:

$$\bar{\boldsymbol{\mu}}^{(t)} = \underset{\bar{\boldsymbol{\mu}}}{\text{argmax}} \; -\frac{k}{2C}\sum_{j=1}^{k} \|\boldsymbol{\lambda}_j^{(t)} + \boldsymbol{\mu}_j\|_2^2 - \sigma_{\mathcal{H}}(\bar{\boldsymbol{\mu}}) \quad (13)$$

which is of the the form $\max g_2(\bar{\boldsymbol{\mu}}) - g_1(\bar{\boldsymbol{\mu}})$ which from Fenchel Deuality is equal to $\min \delta_{\mathcal{H}}(\bar{\mathbf{w}}) - f_2(\bar{\mathbf{w}})$ where

$$f_2(\bar{\mathbf{w}}) = -\frac{C}{2k}\sum_{j=1}^{k} \|\mathbf{w}_j\|_2^2 - \sum_{j=1}^{k} \mathbf{w}_j^\top \boldsymbol{\lambda}_j^{(t)}.$$

The solution for $\bar{\mathbf{w}} = (\mathbf{w}_1, ..., \mathbf{w}_k)$ is:

$$\bar{\mathbf{w}}^{(t)} = \underset{\mathbf{w}_1 = \ldots = \mathbf{w}_k}{\text{argmin}}\; \frac{C}{2k}\sum_{j=1}^{k} \|\mathbf{w}_j\|_2^2 + \sum_{j=1}^{k} \mathbf{w}_j^\top \boldsymbol{\lambda}_j^{(t)},$$

from which we obtain:

$$\mathbf{w}^{(t)} = \underset{\mathbf{w}}{\text{argmin}}\, \frac{C}{k}\|\mathbf{w}\|_2^2 + \mathbf{w}^\top (\sum_{j=1}^{k} \boldsymbol{\lambda}_j^{(t)}),$$

from which it follows that:

$$C\mathbf{w}^{(t)} + \sum_{j=1}^{k} \boldsymbol{\lambda}_j^{(t)} = 0. \quad (14)$$

The connection between $\bar{\boldsymbol{\mu}}^{(t)} = (\boldsymbol{\mu}_1^{(t)}, ..., \boldsymbol{\mu}_k^{(t)})$ and $\bar{\mathbf{w}}^{(t)} = (\mathbf{w}^{(t)}, ..., \mathbf{w}^{(t)})$ follows from Lagrange optimality:

$$\bar{\mathbf{w}}^{(t)} = \underset{\bar{\mathbf{w}}}{\text{argmin}}\, \bar{\mathbf{w}}^\top \bar{\boldsymbol{\mu}}^{(t)} + \frac{C}{2k}\sum_{j=1}^{k} \|\mathbf{w}_j\|_2^2 + \sum_{j=1}^{k} \mathbf{w}_j^\top \boldsymbol{\lambda}_j^{(t)},$$

from which it follows:

$$\boldsymbol{\mu}_j^{(t)} = -\boldsymbol{\lambda}_j^{(t)} - \frac{C}{k}\mathbf{w}^{(t)}.$$

Substituting $\mathbf{w}^{(t)}$ from eqn. 14 we obtain the update formula (eqn. 4):

$$\boldsymbol{\mu}_j^{(t)} \leftarrow -\boldsymbol{\lambda}_j^{(t)} + \frac{1}{k}\sum_{l=1}^{k} \boldsymbol{\lambda}_l^{(t)}.$$

B. Derivation of the conjugate functions

We analyze the convergence properties of our dual optimization scheme, which is described in Eqn. 11. Throughout this section we denote the objective function in Eqn. 11 by $g(\bar{\boldsymbol{\lambda}}, \bar{\boldsymbol{\mu}})$ and

$$g(\bar{\boldsymbol{\lambda}}^*, \bar{\boldsymbol{\mu}}^*) = \max_{\bar{\boldsymbol{\lambda}}, \bar{\boldsymbol{\mu}}} g(\bar{\boldsymbol{\lambda}}, \bar{\boldsymbol{\mu}}).$$

An optimization method finds a ϵ-accurate solution $(\bar{\boldsymbol{\lambda}}', \bar{\boldsymbol{\mu}}')$ if $g(\bar{\boldsymbol{\lambda}}', \bar{\boldsymbol{\mu}}') \geq g(\bar{\boldsymbol{\lambda}}^*, \bar{\boldsymbol{\mu}}^*) - \epsilon$. We prove that the number of iterations required by PDS in order to obtain an ϵ-accurate solution is $O(\log(1/\epsilon))$. Specifically,

$$|g(\bar{\boldsymbol{\lambda}}^t, \bar{\boldsymbol{\mu}}^t) - g(\bar{\boldsymbol{\lambda}}^*, \bar{\boldsymbol{\mu}}^*)| \leq \epsilon, \quad \text{for} \quad t = O(\log(1/\epsilon))$$

In this case we say that the algorithm converges linearly to the solution since for every t larger than some constant N the ratio of $|g(\bar{\boldsymbol{\lambda}}^{t+1}, \bar{\boldsymbol{\mu}}^{t+1}) - g(\bar{\boldsymbol{\lambda}}^*, \bar{\boldsymbol{\mu}}^*)|$ and $|g(\bar{\boldsymbol{\lambda}}^t, \bar{\boldsymbol{\mu}}^t) - g(\bar{\boldsymbol{\lambda}}^*, \bar{\boldsymbol{\mu}}^*)|$ is at most linear (in this case constant).

The analysis of the algorithm's convergence rate is based on reducing the dual problem in Eqn. 11 to eqn. 8 and relating our update schemes in Eqn. 3 and Eqn. 4 to block coordinate ascent of the $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ variables. With this reduction we could rely on previous work by [11] showing that a block coordinate ascent achieves a linear convergence rate, i.e. it takes $O(\log(1/\epsilon))$ to get ϵ-close to the optimal solution.

Our main effort is to derive the relations:

$$\bar{\boldsymbol{\mu}} = -B\boldsymbol{\beta}, \quad \bar{\boldsymbol{\lambda}} = -A\boldsymbol{\alpha} \quad \text{for } 0 \le \boldsymbol{\alpha} \le \frac{1}{m}.$$

We also show that whenever these equalities hold the conjugate dual function in Eqn. 11 takes the values $\sum_{j=1}^{k} h_j^*(\boldsymbol{\lambda}_j) = \mathbf{1}^\top \boldsymbol{\alpha}$ and $\sigma_{\mathcal{H}}(\boldsymbol{\mu}) = 0$. The proof of the above assertions fully describes the equivalence of the optimization schemes in Eqn. 11 and Eqn. 8. The key to this proof is the realization of the conjugate functions $h_j^*(\cdot)$ and $\sigma_{\mathcal{H}}(\cdot)$. These functions enforce the Lagrange multipliers to posses a specific form: $\bar{\boldsymbol{\mu}} = -B\boldsymbol{\beta}$ and $\boldsymbol{\lambda}_j = -Q_j\boldsymbol{\alpha}_j$ where Q_j is a matrix whose columns are the vectors $y_i\mathbf{x}_i$ in the measurements subset $i \in S_j$.

Proposition 1

$$h_j^*(\boldsymbol{\lambda}_j) = \begin{cases} -\sum_{i\in S_j} \alpha_{j,i} & \textit{if } \boldsymbol{\lambda}_j = -Q_j\boldsymbol{\alpha}_j, \ 0 \le \boldsymbol{\alpha}_j \le \frac{1}{m} \\ \infty & \textit{otherwise} \end{cases}$$

There exists a matrix B of dimensions $nk \times (nk-k)$ such that

$$\sigma_{\mathcal{H}}(\boldsymbol{\mu}) = \begin{cases} 0 & \textit{if } \boldsymbol{\mu} = -B\boldsymbol{\beta} \\ \infty & \textit{otherwise} \end{cases}$$

Proof: first we describe the hinge-loss function $loss(\mathbf{w};(\mathbf{x}_i,y_i)) = \max(0, 1 - y_i\mathbf{w}^\top\mathbf{x}_i)$ as an optimization function, a view which simplifies the derivations below and can be verified by inspecting the cases $y_i\mathbf{w}^\top\mathbf{x}_i \ge 1$ and $y_i\mathbf{w}^\top\mathbf{x}_i < 1$:

$$\max(0, 1 - y_i\mathbf{w}^\top\mathbf{x}_i) = \min_{\epsilon_i \in C_i} \epsilon_i,$$
$$\text{where } C_i = \{\epsilon_i \ge 0 \mid y_i\mathbf{w}^\top\mathbf{x}_i \ge 1 - \epsilon_i\}.$$

The above relation yields a simple derivation of the conjugate dual of the hinge-loss function:

$$h_j^*(\boldsymbol{\lambda}) = \begin{cases} -\sum_{i\in S_j} \alpha_i & \text{if } \boldsymbol{\lambda} = -\sum_{i\in S_j} \alpha_i y_i \mathbf{x}_i, \ 0 \le \alpha_i \le 1/m \\ \infty & \text{otherwise} \end{cases}$$

To verify its correctness we use the fact $\max_{\mathbf{w}} f(\mathbf{w}) = -\min_{\mathbf{w}}(-f(\mathbf{w}))$ to deduce from the optimization formulation of the hinge-loss the equality

$$h_j^*(\boldsymbol{\lambda}) = -\min_{\mathbf{w},\epsilon_i \in C_i} (-\boldsymbol{\lambda}^\top\mathbf{w} + \frac{1}{m}\sum_{i\in S_j} \epsilon_i).$$

To realize the explicit expression of $h_j^*(\boldsymbol{\lambda})$ we define the Largrangian, for $\delta_i \ge 0$,

$$L(\mathbf{w},\epsilon_i,\alpha_i) = -\boldsymbol{\lambda}^\top\mathbf{w} + \sum_{i\in S_j}\left(\frac{1}{m}\epsilon_i - \alpha_i(y_i\mathbf{w}^\top\mathbf{x}_i - 1 + \epsilon_i) - \delta_i\epsilon_i\right)$$

and its stationary points

$$\frac{\partial L}{\partial \mathbf{w}} = -\boldsymbol{\lambda} - \sum_{i\in S_j} \alpha_i y_i \mathbf{x}_i = 0, \qquad \frac{\partial L}{\partial \epsilon_i} = \frac{1}{m} - \alpha_i - \delta_i = 0$$

Substituting these results/constraints back into the Lagrangian $L()$ we obtain the function $h_j^*(\boldsymbol{\lambda})$ in Prop 1.
To complete the proof of Prop. 1 we describe explicitly the support function $\sigma_{\mathcal{H}}()$ which is the conjugate dual of the indicator function $\delta_{\mathcal{H}}(\bar{\mathbf{w}})$. The set $\mathcal{H}$ contains the vectors $\bar{\mathbf{w}}$ with nk entries which have k copies of a vector with n entries, therefore it is a linear space since for every $\bar{\mathbf{w}}, \bar{\mathbf{u}} \in \mathcal{H}$ holds $c_1\bar{\mathbf{w}} + c_2\bar{\mathbf{u}} \in \mathcal{H}$ for every real constants c_1, c_2. The support function of the linear subspace $\mathcal{H}$ is the indicator function of its orthogonal subspace, i.e. $\sigma_{\mathcal{H}}(\bar{\boldsymbol{\mu}}) = \delta_{\mathcal{H}^\perp}(\bar{\boldsymbol{\mu}})$. Assume B is the matrix whose columns are the basis of $\mathcal{H}^\perp$ then $\delta_{\mathcal{H}^\perp}(\bar{\boldsymbol{\mu}}) = 0$ if $\bar{\boldsymbol{\mu}} = -B\boldsymbol{\beta}$ for some vector $\boldsymbol{\beta}$ and $\delta_{\mathcal{H}^\perp}(\bar{\boldsymbol{\mu}}) = \infty$ otherwise. □

Finally, we bundle the vectors $\boldsymbol{\lambda}_j = -Q_j\boldsymbol{\alpha}_j$ to obtain $\bar{\boldsymbol{\lambda}} = -A\boldsymbol{\alpha}$ where we relate the matrices Q_j to A in the following manner:

$$A = \begin{bmatrix} Q_1 & 0 & 0 & 0 & 0 \\ 0 & \ddots & 0 & 0 & 0 \\ 0 & 0 & Q_j & 0 & 0 \\ 0 & 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & 0 & Q_k \end{bmatrix}$$

The results in [11] state that a block coordinate ascent for the optimization scheme in Eqn. 8 with respect to $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ converges linearly to its optimal solution. In order to prove that PDS sketched in Algo. 1 converges linearly to its solution we need to show that the optimization of $\bar{\boldsymbol{\lambda}}$ is equivalent to the optimization of the $\boldsymbol{\alpha}$, and the optimization of the $\bar{\boldsymbol{\mu}}$ is equivalent to the optimization of $\boldsymbol{\beta}$. These two assertions are proved in Appendix. A in Eqn. 12 for $\bar{\boldsymbol{\lambda}}$ and Eqn. 13 for $\bar{\boldsymbol{\mu}}$.