A Distributed Machine Learning Framework

Tansu Alpcan and Christian Bauckhage

Abstract— **A distributed online learning framework for support vector machines (SVMs) is presented and analyzed. First, the generic binary classification problem is decomposed into multiple relaxed subproblems. Then, each of them is solved iteratively through parallel update algorithms with minimal communication overhead. This computation can be performed by individual processing units, such as separate computers or processor cores, in parallel and possibly having access to only a subset of the data.**

Convergence properties of continuous- and discrete-time variants of the proposed parallel update schemes are studied. A sufficient condition is derived under which synchronous and asynchronous gradient algorithms converge to the approximate solution. Subsequently, a class of stochastic update algorithms, which may arise due to distortions in the information flow between units, is shown to be globally stable under similar sufficient conditions. Active set methods are utilized to decrease communication and computational overhead. A numerical example comparing centralized and distributed learning schemes indicates favorable properties of the proposed framework such as configurability and fast convergence.

I. Introduction

In the form of multicore processors and cloud computing platforms, powerful parallel and distributed computing systems have recently become widely accessible. This development makes various **distributed and parallel computing** schemes applicable to a variety of problems that have been traditionally addressed by centralized and sequential approaches. For example, distributed schemes are particularly suitable to address machine learning (ML) problems in networked systems such as wireless sensor networks, where data collection and storage processes are inherently distributed. Both efficiency and robustness of such systems are improved by taking advantage of individual processors distributed over the system. Furthermore, overall communication overhead is decreased by not sending all the data to a single fusion center. Similar advantages also hold for multiprocessor systems as well as multicore processors.

In [1], we have introduced a novel decentralized **machine learning** scheme for support vector machines (SVMs) that has unique advantages over its centralized counterparts. Applicability of distributed and parallel approaches to SVMs has been established in [2] through experimental studies. Centralized iterative and gradient descent methods for ML have been discussed in [3]. Another centralized iterative approach has been proposed by Joachims [4] to increase efficiency of SVM training.

Unlike sequential or centralized approaches in the literature [2], [4]–[7], the focus here is exclusively on **parallel update schemes** for machine learning allowing individual processing units to do simultaneous computations. Most of the existing contributions investigate training SVMs either sequentially or in parallel first and then fusing them into a centralized classifier. In [6], a decentralized learning scheme is described where a large training set is partitioned into smaller parts. However, as they do not consider a client server architecture, the communication overhead between the individual units is significant. On the other hand, our approach is especially useful when each unit has access to a different subset of the overall dataset, which may be time-varying, and has its own computing resources. For example, in the context of wireless sensor networks, several distributed learning schemes have been proposed. However, all these schemes are either based on centralized data fusion or Gauss-Seidel methods, where sensors process information sequentially one at a time [2], [8]. In addition to applications in the context of sensor networks, the framework proposed can be applied to networked or multiprocessor systems as well as multicore processors, where each processing core is independent and has its own cache memory as in the case of the *Cell processor* or recent *graphical processing units* (GPUs), such as *Nvidia CUDA* systems.

As part of the proposed distributed ML framework, we first divide the quadratic **SVM binary classification** problem to multiple separate subproblems by relaxing it using a penalty function. We then analyze distributed continuous- and discrete-time gradient algorithms that solve the relaxed problem iteratively. A sufficient condition is derived under which the synchronous parallel update converges to the approximate solution geometrically. Next, we investigate an asynchronous algorithm where only a random subset of processing units update at a given time instance and show its convergence under the same condition. Subsequently, we study stochastic update algorithms which may arise due to imperfect information flow between units or distortions in parameters. Sufficient conditions are derived under which a broad class of stochastic algorithms converge to the solution. Hence, this paper provides a comprehensive analysis of the framework introduced in [1].

In practice, the communication overhead and the number of support vectors resulting from the relaxation of the original problem can be unacceptably high regardless of the update algorithm used. We resort to well-known **active set methods** to address this problem. They have been widely

This work has been supported by Deutsche Telekom AG Laboratories

T. Alpcan is with the Deutsche Telekom Laboratories, 10587 Berlin, Germany tansu.alpcan@telekom.de

C. Bauckhage is with the University of Bonn and the Fraunhofer Institute for Intelligent Analysis and Information Systems, Schloss Birlinghoven 53754 Sankt Augustin, Germany christian.bauckhage@iais.fraunhofer.de

used in solving general quadratic problems as well as in centralized SVM formulations [9]–[11]. The resulting algorithm is greedy in nature and has been observed to converge to a solution in low number of rounds. The distributed learning approach studied is applied to a well-known binary classification example data set. The numerical analysis conducted indicates that the framework has favorable properties such as suitability for online learning and choice of an upper-bound on the resulting support vectors.

The next **section** describes the classical SVM-based classification problem and decomposition of the centralized problem to subproblems for the purpose of decentralized support vector learning. Continuous and discrete-time parallel update algorithms and their convergence properties are analyzed in Section III. Section IV investigates asynchronous and stochastic variants of the update algorithms in Section III. In section V, active set methods are discussed and a numerical example is provided. The paper ends with a short summary and remarks in Section VI.

II. SVM Classification and Decomposition

An overview of the classical SVM-based classification problem is provided for completeness. Subsequently, a relaxation of the centralized formulation and its decomposition into parallel subproblems are presented.

A. Centralized Classification Problem

Assume a given set of labeled data $S := \{(x_1, y_1), \ldots, (x_N, y_N)\}$, where $x_d \in \mathbb{R}^L$ and $y_d \in \{\pm 1\}$, $d = 1, \ldots, N$. A *classification problem* is considered with the objective of deriving a generalized rule from this *training data* that associates an input x with a label y as accurately as possible. It is important to note that no assumption is made on the nature of the training or test data. A well-known method for addressing this binary classification problem involves representation of input vectors in a high(er)-dimensional feature space through a (nonlinear) transformation. Define the dot product of two feature vectors, say x_d and x_e, as a *kernel*, $k(x_d, x_e)$. In many cases this dot product can be computed without actually calculating the individual transformations, which is known as the *kernel trick*.

The optimal margin nonlinear binary **SVM classification** problem [3] described above is formalized using the quadratic problem

$$\max_{\alpha_d} \quad \sum_{d=1}^{N} \alpha_d - \frac{1}{2} \sum_{d=1}^{N} \sum_{e=1}^{N} \alpha_d \alpha_e q_{d,e}$$
$$\text{such that} \quad \alpha_d \geq 0 \quad d = 1, \ldots, N \tag{1}$$
$$\text{and} \quad \sum_{d=1}^{N} \alpha_d y_d = 0. \tag{2}$$

where the α_d are the Lagrange multipliers of the corresponding *support vectors* (SVs) and $q_{d,e}$ are the entries of the positive definite matrix

$$Q := \left[\left(y_d y_e \, k(x_d, x_e) \right)_{d,e} \right]_{N \times N}. \tag{3}$$

The positive definiteness of Q simply follows from the assumption that the kernel matrix $K_{d,e} := [k(x_d, x_e)]_{N \times N}$ is positive definite [3]. The decision function classifying an input x is then

$$f(x) = \operatorname{sgn} \left(\sum_{l=1}^{N_D} \alpha_l y_l k(x, x_l) + b \right),$$

where b is the bias term and $N_D < N$ is the number of support vectors. Here, the well-known *representer theorem* [3] enables a finite solution to the infinite-dimensional optimization theorem in the span of N particular kernels, $k(x, x_d)$, centered on the training points x_d, $\forall d$.

B. Decomposition into Subproblems

In order to decompose the centralized classification problem into **subproblems**, define a set of $\mathcal{M} := \{1, 2, \ldots, M\}$ separate *processing units* with access to different (possibly overlapping) subsets, S_i, $i \in \mathcal{M}$, of the labeled training data such that $S = \bigcup_{i=1}^{M} S_i$. The initial partition of the data S can be due to the nature of the specific problem at hand or as a result of a partitioning scheme at the preprocessing stage. Given the partition, define the vectors $\{\alpha^{(1)}, \alpha^{(2)}, \ldots, \alpha^{(M)}\}$ with the ith one having a size of N_i (elements). In order to devise a distributed algorithm that solves the optimization problem (1), it is relaxed by substituting the constraint $\sum_{d=1}^{N} \alpha_d y_d = 0$ by a quadratic penalty function, $0.5 M \beta (\sum_{d=1}^{N} \alpha_d y_d)^2$, where $\beta > 0$. Next, an upper-bound α_{max} on α_d is imposed such that $\alpha_d < \alpha_{max}$ $\forall d$. This upper-bound can be chosen to derive a soft margin hyperplane (i.e. maximizing the margin). Alternatively, it can be chosen very large to minimize the training error ignoring the margin. Thus, the following constrained optimization problem

$$\max_{\alpha_d \in [0, \alpha_{max}]} F(\alpha) = \sum_{d=1}^{N} \alpha_d - \frac{1}{2} \sum_{d=1}^{N} \sum_{e=1}^{N} \alpha_d \alpha_e q_{d,e} - \frac{M\beta}{2} \left(\sum_{l=1}^{N} \alpha_l y_l \right)^2 \tag{4}$$

approximates (1). Note that the objective function $F(\alpha)$ is strictly concave in all its arguments and the constraint set $X := [0, \alpha_{max}]^N$ is convex, compact, and nonempty.

The convex optimization problem (4) is next **partitioned** into M subproblems through Lagrangian decomposition [12]. Hence, the i^{th} units optimization problem is

$$\underset{\alpha_d^{(i)} \in [0, \alpha_{max}]}{\text{maximize}} \; F_i(\alpha) = \sum_{d \in S_i} \alpha_d - \frac{1}{2} \sum_{d \in S_i} \sum_{e=1}^{N} \alpha_d \alpha_e q_{d,\,e} - \frac{\beta}{2} \left(\sum_{l=1}^{N} \alpha_l y_l \right)^2 \tag{5}$$

Again, the individual objective functions $F_i(\alpha)$ are strictly concave in their all arguments and the respective constraint sets are convex, compact, and nonempty for all i.

The individual optimization problems of the units are interdependent. Hence, they cannot be solved individually

without deployment of an information exchange scheme such as the one depicted in Figure 1 between the processing units.

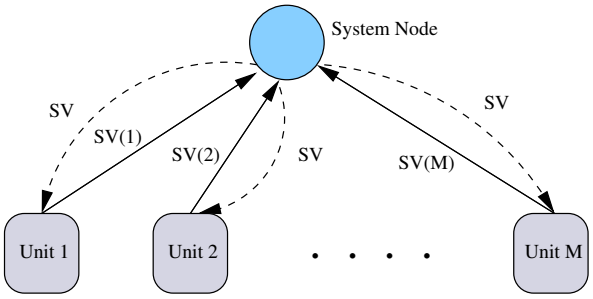


Fig. 1. The information flow within the distributed ML framework.

III. Parallel Update Algorithms

A. Continuous-Time Gradient Algorithm

A distributed continuous-time algorithm, similar to the one in [12], solves the problem (4). Clearly, solving all unit problems at the same time is equivalent to finding the solution of the relaxed problem (4). One possible way of achieving this objective is to utilize a gradient algorithm that converges to the unique maximum, α^*, of (4) which closely approximates the one of the original optimization problem. Then, every unit implements the following **gradient algorithm** for each of its training samples

$$\frac{d\alpha_d}{dt} = \kappa_d \left[1 - \frac{\alpha_d q_{dd} - \sum_e \alpha_e q_{d,e}}{2} - \beta y_d \sum_{l=1}^{N} \alpha_l y_l \right]^P \quad (6)$$

for all $d \in S_i$ and all i where $\kappa_d > 0$ is a unit-specific step-size constant. Here, the shorthand $[\cdot]^P$ refers to the following projection of the $\dot{\alpha}_d := d\alpha_d/dt$:

$$\dot{\alpha}_d = \begin{cases} \dot{\alpha}_d & \text{if } 0 < \alpha_d < \alpha_{max} \\ \dot{\alpha}_d & \text{if } \alpha_d = 0,\ \dot{\alpha}_d > 0 \text{ or } \alpha_d = \alpha_{max},\ \dot{\alpha}_d < 0 \\ 0 & \text{otherwise} \end{cases}$$

Each unit i has access to only its own (possibly overlapping) **training data** S_i whereas the algorithm (6) requires more information to be shared between units. One possible solution to this communication problem is to define a *system node* facilitating the information flow between individual units by collecting and sending back all the necessary information from and to each unit, respectively (Figure 1). Every unit sends to the central node its own set of support vectors $D^{(i)} = \{(x_d, y_d, \alpha_d) | x_d, y_d \in S_i, \forall d \text{ s.t. } \alpha_d^{(i)} > 0\}$. The system node aggregates this information $D := \bigcup_{i=1}^{M} D^{(i)}$ to obtain

$$\begin{aligned} x_D &= \{x_d \in S | \alpha_d > 0\} \\ y_D &= \{y_d \in S | \alpha_d > 0\} \\ \alpha_D &= \{\alpha_d > 0\}. \end{aligned}$$

Subsequently, the system node broadcasts the triple (x_D, y_D, α_D) back to all units $\{1, \ldots, M\}$ after which each unit i locally computes the terms

$$\begin{aligned} t^{(i)} &= Q^{(i)}_{N_i \times N_D}\, \alpha_D \\ u &= \alpha_D^T\, y_D, \end{aligned} \quad (7)$$

where $[\cdot]^T$ denotes transpose operation, N_D is the number of elements of x_D and $Q^{(i)}_{N_i \times N}$ is the portion of the matrix Q relevant to the unit. Thus, the **unit update algorithm** (6) is redefined as

$$\dot{\alpha}_d = \kappa \left[1 - \frac{1}{2}\alpha_d q_{dd} - \frac{1}{2} t_d^{(i)} - \beta y_d u \right]^P, \ \forall d \in S_i,\ \forall i. \quad (8)$$

Theorem 1. *The distributed scheme (6) globally asymptotically converges to the unique maximum of the centralized problem (4).*

Proof. The proof, which is based on Lyapunov methods, is omitted due to space restrictions. □

B. Discrete-Time Gradient Projection Algorithm

This section presents a discrete-time counterpart of the parallel update scheme (6). In this case, every processing unit implements the following **discrete-time gradient projection** algorithm for each of its training samples

$$\begin{aligned} \alpha_d(n+1) = \ & [\alpha_d(n) + \kappa_d G_d(\alpha)]^+ \ \forall d, \text{ where} \\ G_d(\alpha(n)) := \ & 1 - \frac{1}{2}\left(\alpha_d(n) q_{dd} - \textstyle\sum_{e=1}^{N} \alpha_e(n) q_{d,e} \right) \\ & - \beta y_d \textstyle\sum_{l=1}^{N} \alpha_l(n) y_l \,. \end{aligned} \quad (9)$$

Here n denotes the update instances and the notation $[\cdot]^+$ represents the orthogonal projection of a vector onto the convex set X defined by $[\alpha]^+ := \arg\min_{z \in X} \|z - \alpha\|_2$, where $\|\cdot\|_2$ is the Euclidean norm. In this special case, the projection of α_d onto X can be computed in a straightforward way by mapping α_d onto $[0, \alpha_{max}]$ for each d. This facilitates an easy implementation of the parallel algorithm.

The function $F(\alpha)$ in (4) is a polynomial and hence continuously differentiable in its arguments. Furthermore, there exists a scalar constant C such that

$$\|\nabla F(\gamma) - \nabla F(\delta)\|_2 \le C \|\gamma - \delta\|_2, \ \forall \gamma, \delta \in X,$$

where ∇F is the gradient operator. Define $z := \gamma - \delta$. Then,

$$\|\nabla F(\gamma) - \nabla F(\delta)\|_2^2 = z^T A^T A\, z,$$

where

$$A := \frac{1}{2}\mathrm{diag}(Q) + \frac{1}{2} Q + \beta y\, y^T. \quad (10)$$

The matrix $\mathrm{diag}(Q)$ contains the diagonal elements of Q with all its off-diagonal elements set to zero. The scalar constant C is then given by

$$C = \sqrt{\max \lambda(A^T A)},$$

where $\max \lambda(\cdot)$ is the maximum eigenvalue. Therefore, the gradient of the objective function $F(\alpha)$, ∇F, is Lipschitz continuous. Moreover, it is bounded from above on X.

Theorem 2. *The gradient projection algorithm (9) converges to the unique maximum, α^*, of the objective function F in (4), if the step-size constant κ_d satisfies*

$$0 < \kappa_d < \frac{2}{\sqrt{\max \lambda(A^T A)}}, \ \forall d.$$

Proof. The proof follows directly from the upper-boundedness and strict concavity of the function F and Lipschitz continuity of its gradient ∇F. Propositions 3.3 and 3.4 in [13, pp. 213] contain the details. □

Theorem 3. *Assume that the conditions in Theorem 2 hold. Then, the gradient projection algorithm (9) converges to the unique maximum α^* of (4) geometrically.*

Proof. This result follows directly from Proposition 3.5 [13, p. 215], if there exists a constant $c > 0$ such that

$$(\nabla F(\gamma) - \nabla F(\delta))^T (\gamma - \delta) \geq c \left\| \gamma - \delta \right\|_2^2, \ \forall \gamma, \delta \in X.$$

Let us again define $z := \gamma - \delta$. Then,

$$\begin{aligned} (\nabla F(\gamma) - \nabla F(\delta))^T (\gamma - \delta) &= z^T A^T z \\ &\geq z^T \max \lambda(A^T) z, \end{aligned}$$

$\forall \gamma, \delta \in X$, where the matrix A is defined in (10). We note that the matrix A is the sum of two positive definite matrices $\mathrm{diag}(Q)$, Q and a positive semidefinite one, $y\, y^T$. It is hence positive definite and all of its eigenvalues are positive. Hence, there exists a positive constant c which satisfies the sufficient condition for the theorem to hold. □

IV. Asynchronous and Stochastic Algorithms

A. Asynchronous Update Algorithm

A natural generalization of the parallel update algorithms presented is the asynchronous update scheme where only a random subset of processing units update their α values at a given time instance. Notice that the synchronous update scheme can be thought of as a limiting case of this more general version where all units instead of only a random subset update. Define the set of units that update at a given instance n as $\mathcal{M}_u^{(n)}$ and the rest as $\mathcal{M}_{no}^{(n)}$, such that $\mathcal{M}_u^{(n)} \cup \mathcal{M}_{no}^{(n)} = \mathcal{M} \ \forall n$. Then, the update algorithm for the ith unit is:

$$\alpha_d(n+1) = \begin{cases} \left[\alpha_d(n) + \kappa_d G_d(\alpha)\right]^+, \forall d \in S_i, \text{ if } i \in \mathcal{M}_u^{(n)} \\ \alpha_d(n), \forall d \in S_i, \text{ if } i \in \mathcal{M}_{no}^{(n)}, \end{cases} \tag{11}$$

where $G_d(\alpha)$ is defined in (9).

Asynchronous update schemes are in fact more relevant in practical implementations since it is usually difficult for the units to synchronize their exact update instances. The two well known synchronous convergence and box conditions together sufficient for asynchronous convergence of a nonlinear iterative mapping $\mathbf{x}(n+1) = T(\mathbf{x})$ [13, p. 431]. Here it can be shown that both conditions hold (proof omitted due to space constraints), and hence the next convergence result for the asynchronous counterpart of the parallel update algorithm (9) immediately follows from the asynchronous convergence theorem [13, p. 431]:

Theorem 4. *Assume that the conditions in Theorem 2 hold. If a random subset of the units update their α values at each iteration according to (9) while others keep theirs fixed, then the resulting (totally) asynchronous update algorithm defined in (11) converges to the unique maximum α^* of (4).*

B. Stochastic Update Algorithm

All of the update schemes described in the previous sections require an information exchange system to function properly. Until now, the information flow within the system is assumed to be perfect, i.e. the units have access to all the parameters needed by the update algorithms. However, this may not be the case in practice due to a variety of reasons such as **communication errors**, i.e. if the units are not collocated or approximations imposed in order to reduce the communication load between the units. Consider the update algorithm

$$\left[\alpha(n+1) = \alpha(n) + \kappa s(n)\right]^+. \tag{12}$$

Define $s(n) := G(\alpha, n) + \beta(n)$, where $G(\alpha, n)$ is defined component-wise in (9). Here, $\beta(n)$ is the random *distortion* at time instance n. It is possible to establish convergence of the algorithm (12) which can be interpreted as a parallel update scheme under **stochastic distortions**, if these random effects satisfy some conditions. Towards this end, let us characterize the relationship between the distortion term $\beta(n)$ and the real gradient $G(\alpha, n)$ through the variables $\rho(n) > 0$ and $-\pi \leq \theta(n) \leq \pi$:

$$\left|\beta(n)\right| = \rho(n) \left|G(\alpha, n)\right|, \quad \cos(\theta(n)) = \frac{\beta^T(n) G(\alpha, n)}{\left|\beta(n)\right| \left|G(\alpha, n)\right|}. \tag{13}$$

Theorem 5. *Let the stochastic distortion $\beta(n)$ defined in (13) through parameters $\rho(n) > 0$ and $-\pi \leq \theta(n) \leq \pi$ satisfy:*

$$\rho^2(n) + 2\rho(n)\cos(\theta(n)) + 1 > 0, \ \forall n,$$

and

$$\frac{1 + \rho(n)\cos(\theta(n))}{(1 + \rho(n))^2} \geq \bar{K} > 0, \ \forall n,$$

where $\bar{K}$ is a positive real number. Then, the update algorithm (12) converges to the unique maximum, α^*, of the objective function F in (4) geometrically, if the elements of the step-size vector, κ_d, satisfy

$$0 < \kappa_d < \frac{2\bar{K}}{\sqrt{\max \lambda(A^T A)}}, \ \forall d.$$

Proof. The proof makes use of the upper-boundedness and strict concavity of the function F and Lipschitz continuity of its gradient $G(\alpha) = \nabla F$ as in the proof of Theorem 2. Modify the function F, without any loss of generality, such that it is bounded above by zero. It is straightforward to show that the conditions in the theorem on the distortion, $\beta(n)$, imposed through parameters $\rho(n)$ and $\theta(n)$ ensure that

$$\left\| s(n) \right\|_2 \geq k \left\| G(\alpha, n) \right\| \ \forall n,$$

where $k > 0$ is a positive constant and

$$s(n)^T G(\alpha, n) \geq \bar{K} \left\| s(n) \right\|_2^2 \ \forall n.$$

These conditions, together with the one on the positive step-size constant vector k, ensure that the update algorithm (12) converges to the unique maximum of the objective function F, which follows immediately from Propositions 2.1 and 2.3 in [13, pp. 204-206]. Furthermore, since the objective function is strictly concave, the rate of convergence is geometrical (see Propositions 2.4 in [13] for details). □

The conditions in Theorem 5 are investigated through a couple of **illustrative examples** (see Figure 2 for a visualization):

1) Example 1: Let $-\pi/2 < \theta(n) < \pi/2 \ \forall n$. The theorem holds for any value of ρ as long as the condition on the step-size κ is satisfied, where $\bar{K} \leq (1 + \rho(n)\cos(\theta(n)))/(1 + \rho(n))^2 \ \forall n$. This means in practice that if the angle between the gradient $G(\alpha)$ and distortion β vectors remains less than $90°$, then it is possible to choose a sufficiently small step-size to compensate for the errors and ensure convergence.

2) Example 2: Let $|\theta(n)| > \pi/2 \ \forall n$. If $\rho(n) < 1$, then the step-size can be chosen sufficiently small and the conditions in the theorem can be satisfied. In this case, the distortion is in the almost opposite direction of the correct gradient, and hence its norm with respect to the gradient needs to be bounded.

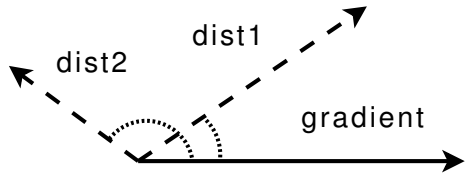


Fig. 2. An example with two different distortion vectors.

V. Active Set-based Algorithm and Numerical Example

Although the update algorithms presented in Section III provably converge to the unique solution of (4), and hence approximately solve the original binary classification problem (1), they often result in a large number of support vectors. This is undesirable not only for efficiency reasons but also due to the communication overhead it brings to the system. To address this problem, *active set* methods are proposed, which have been widely used in solving general quadratic problems. Furthermore, they have also been applied to SVMs in centralized classification formulations [9]–[11].

Active set algorithms solve the problem iteratively by changing the set of active constraints, i.e. the inequality constraints that are fulfilled with equality, starting with an initial active set $\mathcal{A}_0$. Since the initial set is in most cases not the correct one, it is modified iteratively by adding and removing constraints according to some criteria and testing if the solution remains feasible [9]. Active set methods are considered to be more robust and better suited for warm starts, i.e. when the there is some prior knowledge on the initial set [14]. For the relaxed problem (4), there are only non-negativity constraints on α. In this case, the vectors with $\alpha > 0$ constitute the set of support vectors. Hence, the set of support vectors is the complement of the active set and both sets are mutually exclusive. Furthermore, the union of both sets gives the (universal) set of all feature vectors.

The following **greedy algorithm** has been proposed [1] for updating the active set. At each iteration, the data point with the highest positive gradient (derivative of the objective function F in (4)) is removed from the active set as a support vector candidate. Next, the problem described in Section III is solved through parallel updates under the resulting active set constraints. Finally, the support vector with the lowest α is added to the active set which is a rough approximation to finding the constraint that is violated by the solution.

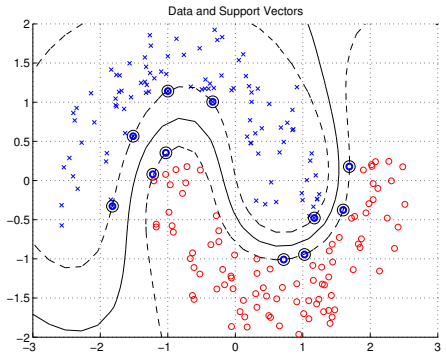
The last step in the algorithm both speeds up the convergence and ensures a **user-defined upper-bound** on the number of support vectors thus turning it into a tunable parameter of the scheme. The total number of active set updates (iterations) in the algorithm should be at most as many as the desired number of support vectors in general unless specific assumptions are made on the initial active set. Due to its iterative nature, this algorithm is also very suitable for online learning where training data is dynamic. Clearly, there is no need to rerun the whole algorithm in order to incorporate new data into the decision function each step.

A. Communication Overhead

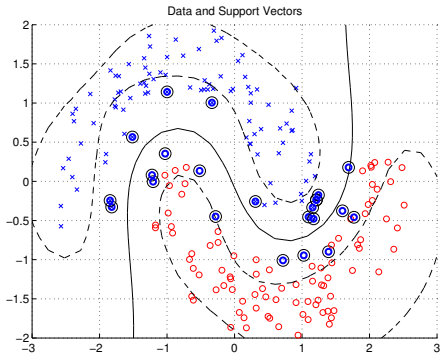
The communication overhead of the active set-based approach is significantly lower than the plain algorithm presented in Section III. During parallel updates at each active set stage (inner loop), individual nodes exchange only the scalar α parameter values. They send a support vector to the system node only when there is a change in the active set. Naturally, the amount of information sent by each unit would have been excessive if there were no active set restriction and all the corresponding data points were transmitted. Therefore, limiting the number of support vectors through the active set decreases the communication load significantly. The information flow within the distributed system depicted in Figure 1 results, thus, in an efficient communication scheme.

B. Numerical Example

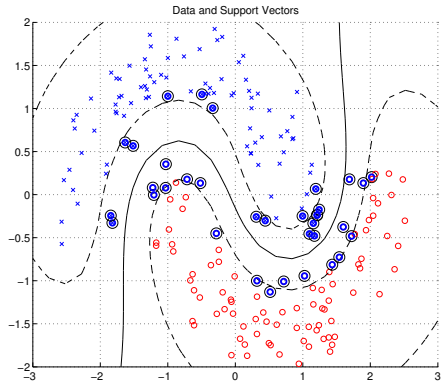
The framework presented is studied numerically on the well-known **half-moon classification problem** and compared with the centralized solution. First, the problem is solved using a standard RBF kernel and QP tools provided by Matlab. Next, the results of the distributed algorithm are obtained with an imposed upper-bound on the number of support vectors of 23 and 40, respectively. The starting point of the algorithm is random and perfect information flow is assumed in the system. The resulting support vectors and their α values are depicted in Figure 3. It is observed the distributed algorithm yields solutions that closely approximate the centralized one.



(a)



(b)



(c)

Fig. 3. The results of the numerical example in Section V-B: (a) centralized solution (b) distributed solution with 23 support vectors (c) distributed solution with 32 support vectors

VI. Conclusion

A distributed machine learning framework based on support vector machines is studied. It allows classification problems to be solved iteratively through parallel update algorithms with minimal communication overhead. Decomposing the quadratic SVM classification problem into multiple relaxed subproblems allows them to be solved by individual processing units operating in parallel and having access to only a subset of the data. Toward this end, distributed continuous and discrete-time gradient algorithms are investigated that solve the relaxed problem iteratively. A sufficient condition is derived under which the synchronous parallel update converges to the approximate solution geometrically. In addition, the same condition is shown to be sufficient for the asynchronous version of the algorithm. Subsequently, a class of stochastic update algorithms are studied which may arise due to imperfect information flow between units or distortions in parameters. Sufficient conditions are derived for their convergence. Finally, active set methods have been utilized to decrease communication and computational overhead of the framework.

The proposed decentralized learning framework is also studied numerically. Initial results demonstrate that the SVMs trained according to our procedure yield robust results and the distributed ML framework has favorable characteristics such as efficiency, configurability, robustness, suitability for online learning, and low communication overhead.

Future research directions include experimental investigation of the effects of distortions in information flow and communication errors. Another open research question is the study of clustering methods and hierarchical structuring of the input data.

References

[1] T. Alpcan and C. Bauckhage, "A discrete-time parallel update algorithm for distributed learning," in *Proc. of 19th Intl. Conf. on Pattern Recognition (ICPR)*, Tampa, FL, USA, December 2008.

[2] R. U. Pedersen, *Using Support Vector Machines for Distributed Machine Learning*, Ph.D. thesis, University of Copenhagen, August 2004.

[3] B. Scholkopf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*, MIT Press, Cambridge, MA, USA, 2002.

[4] T. Joachims, *Making Large-Scale Support Vector Machine Learning Practical*, chapter 11, pp. 169–184, Advances in Kernel Methods: Support Vector Learning. MIT Press, Cambridge, MA, USA, 1998.

[5] H.P. Graf, E. Cosatto, L. Bottou, I. Dourdanovic, and V. Vapnik, "Parallel Support Vector Machines: The Cascade SVM," in *Proc. NIPS*, 2005, pp. 521–528.

[6] Y. Lu and V. Roychowdhury, "Parallel Randomized Support Vector Machine," in *Proc. PAKDD*, 2006, pp. 205–214.

[7] G. Zanghirati and L. Zanni, "A parallel solver for large quadratic programs in training support vector machines," *Parallel Comput.*, vol. 29, no. 4, pp. 535–551, 2003.

[8] J. B. Predd, S. B. Kulkarni, and H. V. Poor, "Distributed learning in wireless sensor networks," *IEEE Signal Processing Magazine*, vol. 23, no. 4, pp. 56–69, July 2006.

[9] M. Vogt and V. Kecman, *Active-Set Methods for Support Vector Machines*, pp. 133–158, Support Vector Machines: Theory and Applications. Springer-Verlag, Berlin, Heidelberg, August 2005.

[10] D. R. Musicant and A. Feinberg, "Active set support vector regression," *IEEE Trans. on Neural Networks*, vol. 15, no. 2, pp. 268– 275, March 2004.

[11] K. Scheinberg, "An efficient implementation of an active set method for svms," *J. of Machine Learning Research*, vol. 7, pp. 2237–2257, December 2006.

[12] T. Alpcan, X. Fan, T. Başar, M. Arcak, and J. T. Wen, "Power control for multicell CDMA wireless networks: A team optimization approach," *Wireless Networks*, p. (online first), 2007.

[13] D. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*, Prentice Hall, Upper Saddle River, NJ, 1989.

[14] S. Leyffer, "The return of the active set method," *Oberwolfach Reports*, vol. 2, no. 1, 2005.