Submitted to *Management Science*
manuscript MS-0001-1922.65

A Consensus Algorithm for Linear Support Vector Machines

(Authors' names blinded for peer review)

In the era of big data, an important weapon in a machine learning researcher's arsenal is a *scalable* Support Vector Machine (SVM) algorithm. Traditional algorithms for learning SVMs scale super linearly with training set size which becomes infeasible quickly for large data sets. In recent years, scalable algorithms have been designed which study the primal or dual formulations of the problem. This often suggests a way to decompose the problem and facilitate development of distributed algorithms. In this paper, we present a distributed algorithm for learning linear SVMs in the primal form for binary classification called Gossip-bAseD sub-GradiEnT (GADGET) SVM. The algorithm is designed such that it can be executed locally on sites of a distributed system; each site processes its local homogeneously partitioned data and learns a primal SVM model; it then gossips with random neighbors about the classifier learnt and uses this information to update the model. To learn the model, the SVM optimization problem is solved using several techniques including a gradient estimation procedure, stochastic gradient descent (SGD) method and several variants including mini-batches of varying sizes. Our theoretical results indicate that the rate at which the GADGET SVM algorithm converges to the global optima at each site is dominated by an $O(\frac{1}{\sqrt{\lambda}})$ term, where λ measures the degree of convexity of the function at the site. Empirical results suggest that this *anytime* algorithm – where the quality of results improve gradually as computation time increases – has performance comparable to its centralized, pseudo-distributed and other state-of-the-art gossip based SVM solvers. It is at least 1.5 times (often several orders of magnitude) faster than other gossip-based SVM solvers known in literature and has a message complexity of $O(d)$ per iteration, where d represents the number of features of the data set. Finally, a large scale case study is presented wherein the consensus based SVM algorithm is used to predict failures of advanced mechanical components in a chocolate manufacturing process using more than a million data points.

Key words: distributed support vector machine, primal SVM, consensus based learning, gossip, anytime algorithm

1. Introduction

Predictive models play an important role in business intelligence tasks. Such models can be learnt from historical data and may be used to predict customer behavior or fraudulent

activity. One such predictive model used extensively by businesses is the Support Vector Machine (SVM) (Cecchini et al. (2010), Simester et al. (2019), Li et al. (2017)). Quoting Provost and Fawcett (2013), "If you're even on the periphery of the world of data science these days, you eventually will run into the Support Vector Machine or "SVM". This is a notion that can strike fear into the hearts even of people quite knowledgeable in data science. Not only is the name itself opaque, but the method often is imbued with the sort of magic that derives from perceived effectiveness without understanding".

However, in recent years, businesses are forced to deal with large and complex collections of digital data (Zadorojniy et al. (2017), Fadda et al. (2018)). This has necessitated the development of *scalable* predictive models (Rajaraman and Ullman (2011), Bottou et al. (2007), Bekkerman et al. (2011)) in general, and not surprisingly, therefore, *scalable* SVM algorithms. Scalable SVMs rely fundamentally on well established techniques of parallelization and distributed computing (Kargupta and Chan (2000), Zaki and Ho (2000), Tanenbaum and Steen (2006), Lynch (1996), Bertsekas and Tsitsiklis (1997)). Parallel algorithms for learning predictive models are often tightly coupled including shared memory systems (SMP), distributed memory machines (DMM) or clusters of SMP workstations (CLUMPS) with fast interconnection between them. Distributed systems, on the other hand, are loosely-coupled – for example, mobile ad-hoc networks or sensor networks (Bliman and Ferrari-Trecate (2008), Blondel et al. (2005), Boyd et al. (2005)). They can function without a central server for co-ordination and are often subject to abrupt changes in topology due to sites joining or leaving, and are susceptible to link failures. An important characteristic of these distributed systems is that they are collectively capable of storing large amounts of data of different modalities (such as text, audio, video). Often, distributed predictive modeling tasks are designed by executing them on data distributed in a network – for example, distributed sensor networks for co-ordination and control of Unmanned Aerial Vehicles (UAVs), fleet, self-driving cars (Tortonesi et al. (2012), Kargupta et al. (2010)), automated products and parts in transportation, life science and energy markets.

In this paper, we present a scalable consensus based linear SVM algorithm for binary classification called **G**ossip-b**A**se**D** sub-**G**radi**E**n**T** solver (GADGET SVM). Such an algorithm can be used by businesses deploying large clusters to store data or those which collect streaming data from Internet Of Things (IoT) devices. In the large cluster setting, the underlying topology of the sites is usually fixed apriori while mobile and adhoc sensor

networks have varying topologies. In this work, we assume that there is a network of computational units arranged in a fixed topology, each containing samples from the training data such that all sites have the same set of features or attributes. Sites are capable of building support vector machine models from *local* data i.e. data available to or sensed by the site. They can update local models by exchanging information with their neighbors. This is important, since the local model by itself is not sufficient to provide a global picture. The overall goal, is to learn at each site, a close approximation of the *global* function. Communication between sites is permitted by use of a *gossip*-based protocol i.e. each site contacts a neighbor at random and exchanges information. The process continues until there are no significant changes in performance of the local model. This gossip-based communication protocol used to design a predictive model gives it the much needed scalability including resilience to sites joining or leaving the network. Finally, it must be noted that the GADGET SVM algorithm is an *anytime* algorithm (Zilberstein (1996), Mouaddib and Zilberstein (1995), Zilberstein (1993)) without any predefined termination criteria. Anytime algorithms are those whose quality of results improve gradually as computation time increases. They extend the traditional notion of a computational procedure by allowing it to return many possible approximate answers to any given input. These notions of approximate processing (Lesser et al. (1988)) and the use of principles of bounded rationality (Simon (1982)) have proved useful in many applications. What is special about anytime algorithms is the use of well-defined quality measures to monitor the progress in problem solving and allocate computational resources effectively.

Summary of main contributions This work has three main contributions. First, we develop a consensus algorithm for learning a SVM in the primal with non-differentiable hinge loss and a squared regularization term. The choice of this formulation is motivated by the state-of-the-art Pegasos solver (Singer and Srebro (2007)). The optimization problem that arises in this setting is solved in a *network* using stochastic sub-gradient descent algorithm and its mini-batch variant. This is accomplished by resorting to the use of a consensus based averaging protocol – the Push Sum protocol (Kempe et al. (2003a)). Second, we empirically study the performance of the algorithm using standard stochastic gradient descent (SGD) method in the network setting. The presence of noise in the gradients arising both from local sub-gradient estimates and gossip averages in the network does not

necessarily hinder the performance of the algorithm which is comparable to state-of-the-art gossip based distributed SVMs. This is achieved by communicating only the weight vectors in the network which provides significant benefits in terms of time and message complexity unlike the state-of-the-art gossip based solvers which typically resort to sending data samples to neighbors. Third, analytical results reveal that the consensus algorithm at each site converges to the global optima at a rate $O(\frac{1}{\sqrt{\lambda}})$, where λ refers to the degree of convexity of the function at the site. Finally, we present a case study to predict failures of advanced mechanical components in a chocolate manufacturing process using more than a million data points.

Organization of the paper The rest of this paper is organized as follows: Section 2 motivates the need for consensus based learning; Section 3 provides the background and related work; Section 4 presents the GADGET SVM algorithm; Section 5 presents empirical results on real world data; Section 6 presents a case-study on a Bosch manufacturing process and Section 7 discusses directions of future work and concludes the paper. The analytical results for the GADGET SVM algorithm are presented in the Appendix.

2. Why consensus based learning?

We posit that a consensus based SVM algorithm takes a step towards addressing concerns identified by a recent McKinsey Global Institute Report (Manyika et al. (2015)):

> "Most IoT data are not used currently ... The data that are used today are mostly for anomaly detection and control, not optimization and prediction which provide the greatest value."

For example, in the manufacturing industry – with increasing automation in production processes, combining sensor signals and product inspections is common (Gong et al. (1997), Wardell et al. (1992)). Gong et al. (1997) present a two phase procedure for combining readings from online sensors and control charts to improve statistical process control decisions. In the first phase, the production process is monitored continually with sensors and warnings are issued when there are process shifts; in the second phase, samples are drawn from the process and inspected. The sensors and IoT devices here are primarily used for anomaly detection and control.

In the consumer, enterprise and societal IoT world, including predictive maintenance, intelligent healthcare, smart cities and housing, etc the dominant paradigm has been IoT

just senses its environment and transmits the sensor readings to the cloud where all the decision making happens. For instance, in recent work (Zhang et al. (2019)) that studies the value of pop-up stores (temporal sales spaces designed to offer a direct and experiential consumer–brand interaction), on both retailers and retailing platforms, consumers' pop-up store visits were tracked with Alibaba's WiFi tracking modem. When the Wi-Fi modem probes a phone, it usually returns a message with its unique Media Access Control (MAC) address. The strength of the message returned by the phone signals its distance to the modem. The MAC address is also recorded when a phone is used to log on to Alibaba's mobile app which all consumers in the experiment must use. This allowed Alibaba to match phones detected within the pop-up store with consumers, enabling linkage between consumers' offline presence at the pop-up store with their online behavior. In this setting, the dependence is on the cloud for matching data from the IoT device while the device itself is not participating in predictive analytics.

In the context of healthcare, Staats et al. (2017) resort to electronic monitoring of individuals using a Radio Frequency IDentification (RFID)-based system deployed in 71 hospital units to test hand hygiene compliance. RFID badges are distributed to hospital caregivers who wear personalized badges along with their hospital identification. The RFID readers are also installed throughout the hospital unit. These badges then communicate wirelessly with a network of sensors connected throughout the monitored area. In addition, communication units are installed on hand hygiene dispensers within a focal hospital unit. As a result, both the date and time when a caregiver enters the area monitored by a given sensor as well as whether or not the caregiver uses a dispenser are recorded. The information is then transmitted to central servers for further processing. This is yet another example where the IoT data collected is first centralized before algorithms for predictive analytics can be run on them.

While the above examples provide options to deal with streaming data collected at sites, the dependence on the cloud or large scale centralized storage makes the IoT device "dumb". What happens to decision making when connection to the cloud fails? The consensus based predictive algorithms proposed here fundamentally address this issue by allowing the sites or computational units to communicate with one another and therefore are robust to network failures which prohibit interaction with the cloud.

Finally, it must be noted that these consensus based predictive modeling algorithms (Datta et al. (2006)) build on the seminal work of Tsitsiklis (Tsitsiklis (1984), Bertsekas and Tsitsiklis (1997)). They are completely decentralized (without a central server for coordination), asynchronous, resilient to changes in underlying topology, and scalable in size of the network. They are capable of learning *cooperatively* and when in agreement, can reach a *consensus*. At that time, the sites have a good approximation (measured by ϵ tolerance, where ϵ is usually user-defined parameter) of the global solution.

Having motivated the need for consensus based learning for businesses dealing with large complex data, we provide a brief review of the popular support vector machine algorithm which can be learnt in a consensus based framework.

3. Background

3.1. Support Vector Machines - A brief review

Formally, given a training set S comprising of feature vectors $x_i \in \mathbb{R}^d, i = 1, 2, \cdots, N$ and binary labels $y_i \in \{-1, +1\}$, the goal is to find a linear classifier of the form $f(x) = w^T x, w \in \mathbb{R}^d$. More generally, in the primal SVM formulation the goal is to find the minimizer of the problem

$$\min_{\mathbf{w}} \frac{\lambda}{2} \parallel \mathbf{w} \parallel^2 + \frac{1}{N} \sum_{j=1}^{N} l(\mathbf{w}; (\mathbf{x_j}, y_j)), \tag{1}$$

where l is the loss function defined as $l(\mathbf{w}; (\mathbf{x}, y)) = \max\{0, 1 - y\langle \mathbf{w}, \mathbf{x} \rangle\}$ for hinge loss and λ is the SVM regularization parameter.

From a risk minimization perspective, the model shown in Equation 1 can also be written as

$$\begin{aligned} &\min_{w,\xi} \frac{1}{2} \parallel \mathbf{w} \parallel^2 + C \sum_{i=1}^{N} \xi_i, \\ &\text{subject to } \xi_i \geq 0 \\ &y_i(w^T x_i) \geq 1 - \xi_i, \ i = 1, 2, \cdots, N \end{aligned} \tag{2}$$

where $\xi_i = |y_i - f(x_i)|$ measures the training error.

3.2. Related Work

The problem of scaling SVM algorithms have been studied extensively (Osuna et al. (1997b,a), Joachims (1998), Menon (2009)) with a majority of the algorithms developing faster variants of the primal, dual or primal-dual formulations. In this section, we

present related work for solving the SVM optimization in primal form. Following this, we present *scalable* SVM algorithms including parallel and distributed variants which are closely related to the current work.

3.2.1. Primal Formulations. Optimizations of the primal formulation of linear SVMs have been studied extensively (Mangasarian (2002), Keerthi and DeCoste (2005), Keerthi et al. (2006)). Mangasarian (2002) presents finitely terminating Newton methods with Armijo method while Keerthi and DeCoste (2005) extend this work by performing exact line searches to determine the step size for L_2[1] loss functions. They also suggest methods to solve the primal SVM formulations using L_1 loss by approximating the loss using modified Huber and logistic regression (Zhang et al. (2003)). Chapelle (2007) complements the literature by extending the above techniques to the non-linear case.

Other large scale primal SVM formulations have been solved by using SGD, (Menon (2009)) methods. Bottou proposed the SVM-SGD (Bottou and Bousquet (2011), SVM-SGD) algorithm which builds a model by solving Equation 1 on benchmark datasets. Zhang (2004) studied SGD algorithms on regularized forms of linear prediction methods such as least squares for regression, logistic regression and SVMs for classification. One of the popular SGD algorithms, Pegasos (Shalev-Shwartz et al. (2007)), operates by choosing a random subset of k training examples, evaluating the sub-gradient of the objective function on these examples and updating the weight vector accordingly. The weight vector is then projected on a ball of radius $\frac{1}{\sqrt{\lambda}}$. The parameter k does not affect the run time or its convergence to the optimal solution. Duchi and Singer (2009) present the FOrward Backward Splitting algorithm (FOBOS) which alternates between two phases - in the first, an unconstrained gradient is estimated. This is followed by solving an instantaneous optimization problem that trades off minimization of the regularization term while keeping close proximity to the result of the first phase. Finally, Chang et al. (2008) propose a coordinate descent method for solving primal L_2-SVM which does not work for the L_1 SVM due to its non-differentiability. To the best of our knowledge, none of the primal SVM formulations discussed above have been used in the context of distributed consensus based learning.

[1] SVMs with linear sum of slack variables, which are commonly used, are called L_1-SVMs, and SVMs with the square sum of slack variables are called L_2-SVMs.

3.2.2. Distributed Optimization. Distributed optimization algorithms which can be used to solve the primal SVM formulation and its variants are relevant to this discussion. Nedić and Ozdaglar (2009) propose a generic distributed computation framework for optimizing a sum of convex objective functions for multiple agents. In this framework, every agent maintains estimates of the solution of the global objective which are communicated to other agents directly or indirectly using asynchronous communication (Jadbabaie et al. (2003), Tsitsiklis et al. (1986)). Each agent updates it's estimates based on local information concerning the estimates received from it's immediate neighbors and it's own cost function using a sub-gradient method. This work, however, was not used to solve a regularized SVM formulation and did not deal with *stochastic* sub-gradient descent methods. Duchi et al. (2012) develop and analyze algorithms for dual sub-gradient averaging which is based on the availability of a proximal function. Their analysis enables separation of the convergence of the optimization algorithm and the effects of communication dependent on the network structure. Sundhar et al. (2010) present an algorithm for a distributed multi-agent system where the goal is to minimize a sum of convex objective functions of the agents subject to a common convex constraint set. While the formulation of the problems studied in these papers are interesting and relevant, they have not been studied in the context of SVMs. Block minimization, a classical technique used in optimization literature has been used to train linear SVMs by Yu et al. (2012). Under this framework, a solver splits data into blocks and stores them as separate files. Then, every time the solver trains a data block it is loaded from disk. This setting is different from the one studied in this paper where data is stored in a distributed environment.

3.2.3. Distributed and Parallel SVMs. Algorithms for scaling SVMs using distributed and parallel learning have been studied by Stolpe et al. (2016). Two different kinds of algorithms are popular (a) methods designed to run in high performance compute clusters, assuming high bandwidth connections and an unlimited amount of available energy (b) pervasive computing systems (e.g. wireless sensor networks) consisting of battery-powered devices, which usually require algorithms whose primary focus is the preservation of energy.

Syed et al. (1999) proposed a Distributed SVM (DSVM) algorithm which finds support vectors locally at each site and then sends them to a central server for processing. However, it produces globally sub-optimal solutions and the communication cost depends on the total size of the dataset. The algorithm is improved in Caragea et al. (2005) by allowing

the centralized server to send the support vectors back to the distributed sites and then repeating the process until a global optimum is achieved. Despite reaching optimality, this approach is slow. Cascade SVMs (Graf et al. (2005)) work by eliminating non-support vectors early from the optimization and generating a filtering process that can be parallelized effectively. A number of small, independent partitions are explored at a time to generate partial results, which are eventually combined in a hierarchical fashion. Lu et al. (2008) propose an algorithm similar to Cascade SVM suited for Kurtowski graphs.

Hazan et al. (2008) present a parallel algorithm for solving large scale SVMs by dividing the training set amongst a number of sites each running an SVM sub-problem associated with that training set. The algorithm uses 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. The algorithm has a linear convergence rate and takes $O(\log(\frac{1}{\epsilon}))$ iterations to get ϵ-close to the optimal solution. Sai et al. (2016) describe a Budgeted Parallel Pack Gradient Descent algorithm (BPPGD) that can improve the primal SVM optimization problem with Gaussian Radial Basis Function (RBF) kernels for large-scale data. This method has been shown to run efficiently on Apache Spark with high degree of parallelization.

A distributed SVM resilient to Byzantine failures has been proposed by Yang and Bajwa (2016). The premise of this work is that in the distributed setting an SVM can have arbitrarily bad performance if there are Byzantine nodes in the network. If this happens, it is possible for a node to ignore some values received from its neighbors, assuming that these values were erroneous and tends to push the node away incorrectly from its local solution. Consequently, the solution is to identify such error prone values and treat them as outliers that can be filtered out.

Finally, Lee et al. (2012) present a framework for training SVMs over distributed sensor networks by making use of multiple local kernels and explicit approximations to feature mappings induced by them.

3.2.4. Gossip based Distributed SVM In this section we present several algorithms that use gossip based protocols for the design of distributed SVMs. The algorithms presented here primarily differ in what is sent (training data, extrema of the convex hull, weight vectors, etc.) to the neighbors during gossip.

Flouri et al. (2009) present a gossip-based SVM algorithm which uses a selection function to rank training vectors in order of importance in the learning process. Their Adaptive Selective Gossip (ASG) algorithm begins by training an SVM on the local data, determining an optimal hyperplane and ranking based on local data. The ranking function selects data points from both classes and defines a *Selected Set* which is exchanged with neighbors. Unlike the algorithm proposed here, this algorithm resorts to sending actual data in the network, significantly increasing the communication cost incurred during its execution and possibly compromising privacy concerns. Prior work of Flouri et al. (2006) also presents two other algorithms for selectively picking data points for gossiping – (a) Minimum Selective Gossip (MSG-SVM) algorithm where-in only the support vectors obtained locally are communicated to the neighbors and (b) Sufficient Selective Gossip (SSG-SVM) where the convex hull of the two classes is constructed and the extremum points are communicated to the neighbors. According to the authors, MSG-SVM is not guaranteed to converge and produces sub-optimal results, while SSG-SVM requires communication of a large amount of data and incurs additional power cost in sensor networks.

Wang et al. (2010) propose a gossip-based SVM algorithm whose objective is to gossip the *labels* learnt after local learning at each node. The local nodes learn a non-linear SVM and obtain a set of Lagrange multipliers. Once the training converges at a node, the local prediction value for a given instance is estimated and communicated to its neighbors. The global prediction value is updated based on the values predicted by neighbors. It is interesting to note, that the gossip protocol is never used *during* the training phase of the algorithm, which is strikingly different from the other algorithms discussed in this section.

The algorithm to learn linear models presented by Ormandi et al. (2013) consists of a gossip based peer sampling procedure which helps to generate a local random sample at the peer. Each peer periodically checks on incoming models and potentially combines it with the previous incoming model. This newly created model is stored in a cache of a fixed size and when full, the model stored for the longest time is replaced. The cache provides a pool of recent models that can be combined using voting based prediction.

Kim et al. (2015) (similar to Flouri et al. (2006)) show that a data set can be represented using geometric convex hulls, and the classification problem can be converted to a nearest point problem which leads to an efficient solution to SVM classification. For the separable case, this amounts to finding the closest points between the convex hulls generated by the

positive and negative classes. For the non-separable case, the above procedure does not make sense, because there are the infinite number of the points in the over-lapped area and all these points are the closest points to the convex hulls with zero distance. In this case, the notion of a reduced convex hull is examined wherein the coefficients of the convex hull are upper bounded by a non-negative number. In the distributed setting, they propose to gossip the set of extreme points of the convex hull of local data set with neighboring nodes. Specifically, each local node generates the extreme point sets which are then communicated to one-hop neighbors. On receiving the extreme points, the neighbors add these to their existing extreme point sets and the process continues. It has been theoretically shown that this process is equivalent to finding the convex hull of the union of two (or more) convex sets. While the above algorithm operates both in linear and non-linear settings, the sending of extremum points amounts to significant increase in communication cost in the network (Flouri et al. (2006)), hindering scalability of the solution.

The algorithm closest in spirit to ours is the consensus based SVM algorithm proposed by Forero et al. (2010). The fundamental differences are (1) the Alternating Direction Method of Multipliers DSVM (MoM-DSVM) solves the *dual* of the linear SVM formulation (given by Equation 2) whereas, GADGET SVM solves the *primal* formulation similar to the Pegasos algorithm (Shalev-Shwartz et al. (2007)) with modifications due to the distributed nature of the problem. (2) GADGET uses SGD for solving the optimization problem while MoM-DSVM relies on the Alternating Direction Method of Multipliers (Bertsekas and Tsitsiklis (1997), Boyd et al. (2011)). (3) The underlying protocol used for communication in GADGET is a randomized gossip algorithm whereby each site exchanges information with *only one* randomly chosen immediate neighbor within a one-hop distance from itself. In contrast, MoM-DSVM broadcasts its current augmented vector $v_j = [w_j^T; b_j]^T$ thereby having a higher communication cost than the algorithm described here.

Finally, it must be noted that Hensel and Dutta (2009) present a consensus based SVM algorithm with two calls to the Push-Sum protocol at each site. Our algorithm is different from the one presented in Hensel and Dutta (2009) for the following reasons: (1) The two calls to Push-Sum in Hensel and Dutta (2009) makes synchronization amongst sites a difficult problem. GADGET SVM uses only a single Push-Sum call. (2) Theoretical results on convergence of GADGET SVM have been presented in this paper (refer to the Appendix). (3) The current manuscript presents a detailed study of the effect of stochastic optimization

techniques such as SGD and its variants including Mini-Batch SGD on convergence properties of the algorithm (4) The empirical results presented in the paper compares GADGET SVM to state-of-the-art variants including LIBSVM, LIBLINEAR, SVM^{Perf} and a pseudo distributed solver using Alternating Direction Method of Multipliers (ADMM). (5) It also compares GADGET SVM with state-of-the-art gossip based SVM solvers. (6)The utility of developing a large scale SVM algorithm, given the existence of other scalable classification algorithms such as adaptions of (penalized) regression, boosting and random forests has been studied empirically. (7) A real world case study on an imbalanced classification problem involving failures of advanced mechanical components in a chocolate manufacturing process with more than a million data points has been presented.

3.3. Communication protocols - Gossip

Gossip based protocols are popular in distributed systems because of their fault tolerant information dissemination (Boyd et al. (2006), Shah (2009), Dimakis et al. (2010, 2006), Narayanan (2007)). Dating back to early work in the database community (Demers et al. (1987)), they provide a simple and effective information spreading strategy, in which every site selects one of its neighbors uniformly at random for message exchange during the process of spread of information. They are more efficient than widely adopted information exchange protocols such as broadcasting and flooding. Gossip can be used for computation of sums, averages, quantiles, random samples and other aggregate functions and probabilistic guarantees of convergence are ascertained for such computations. The problem of *aggregation* was first proposed by Bawa et al. (2003) wherein it is assumed that there is a network of m sites, each containing a value $q_i \in R$. The goal is to compute the aggregate functions in a decentralized fault tolerant fashion. Kempe et al. (2003b) extend this work further by demonstrating that these protocols converge exponentially fast to the correct answer when using uniform gossip. They present the Push-Sum algorithm (also presented in Algorithm 1 for completeness) an n-dimensional extension of which is used in this work for communication amongst sites in the distributed setting for the GADGET SVM algorithm (presented in Section 4). It operates as follows – at all times t, each site i maintains a sum $s_{t,i}$, initialized to $s_{0,i} = q_i$, and a weight $r_{t,i}$, initialized to $r_{0,i} = 1$. In addition, a site also maintains $\alpha_{t,i,j}$ (The $\alpha_{t,i,j}$ should not be confused with the α values arising from the dual formulation of the SVM.), a non-negative value for site j which indicates how much information can be shared between sites i and j in iteration t of the Push-Sum

Push-Sum (PS)

Input: A graph $G(V,E)$ with m vertices (or nodes/sites) i.e. $|V|=m$ and $|E|$ edges; Each site stores a single value $q_i \in R, 1 \leq i \leq m$; t : Number of iterations of the algorithm; $\alpha_{t,i,j}$.

Each site maintains a sum $s_{t,i}=q_i$ and weight $r_{t,i}=1$.

1. Let $\{(\hat{s}_{p,i}, \hat{r}_{p,i})\}$ be all the pairs sent to site i in round $t-1$ (p is the index of pairs sent to site i in round $t-1$).
2. Let $r_{t,i}=\sum_p \hat{r}_{p,i}$ i.e. perform a sum of all weights received by site i in round $t-1$.
3. Let $s_{t,i}=\sum_{p,i} \hat{s}_{p,i}$ i.e. perform a sum of $\hat{s}_{p,i}$ received by site i in round $t-1$.
4. Choose shares $\alpha_{t,i,j}$ for each j that site i wishes to communicate with.
5. Send $(\alpha_{t,i,j} \times s_{t,i}, \alpha_{t,i,j} \times r_{t,i})$ to the site j and update the same for i

Output: $\frac{s_{t,i}}{r_{t,i}}$, this is the current estimate of the average at site i at time t.

Algorithm 1: Push-Sum (PS), as proposed by Kempe et al. (2003b). Presented here for the sake of completeness.

algorithm (Note that $\sum_j \alpha_{t,i,j}=1$). At time 0, it sends the pair $(s_{0,i}, r_{0,i})$ to itself and in each subsequent time step t, each site i follows the protocol given as Algorithm 1 and updates the weight and sum. The algorithm, as presented, helps to estimate the average in the network. A simple extension to protocol Push-Sum, called Push-Vector, wherein each site holds a vector $v_{t,i}$ (instead of the real-valued sum $s_{t,i}$) has also been presented in Kempe et al. (2003b). The primitive of estimating the average in the network forms the crux of GADGET SVM, wherein each site has a weight vector learnt on local data; then it participates in exchange of information with neighbors by averaging weights. The details of the GADGET SVM algorithm is presented in the following section.

4. The GADGET SVM Algorithm

The Gossip bAseD sub-GradiEnT solver for linear SVMs aims to solve Equation 1 in a decentralized setting. Let M denote an $N \times d$ matrix with real-valued entries. This matrix represents a dataset of N tuples of the form $x_i \in \mathbb{R}^d, 1 \leq i \leq N$. Assume this dataset has been *horizontally* distributed over m sites $S_1, S_2, \cdots, S_m$ such that site S_i has a data set $M_i \subset M, M_i : n_i \times d$ and each $x_j \in M_i$ is in $\mathbb{R}^d$. Thus, $M = M_1 \cup M_2 \cup \cdots \cup M_m$ denotes the concatenation of the local datasets. The goal is to learn a linear SVM on the global data

set M, by learning local models at the sites and allowing exchange of information among them using a gossip based protocol.

Pegasos

Input: Number of instances in the dataset: M, Learning Rate: λ, Number of Iterations: T

Initialize: Set $w_1 = 0$

1. For $t = 1, 2, \cdots, T$
2. Choose $i_t \in \{1, \cdots, |M|\}$ uniformly at random
3. Set $\eta_t = \frac{1}{\lambda t}$ { ▷ Set the learning rate.}
4. If $y_i \langle w_t, x_i \rangle < 1$, then { ▷ Test if instance has non-zero loss..}
5. Set $w_{t+1} \leftarrow (1 - \eta_t \lambda) w_t + \eta_t y_i x_i$
6. Else (if $y_i \langle w_t, x_i \rangle \geq 1$) { ▷ If no loss incurred, no changes required.}
7. Set $w_{t+1} \leftarrow (1 - \eta_t \lambda) w_t$
8. [Optional: $w_{t+1} \leftarrow min\{1, \frac{\frac{1}{\sqrt{\lambda}}}{||w_{t+1}||}\} w_{t+1}$] { ▷ Projection Step}

Output: w_{T+1}

Algorithm 2: Primal Estimated sub-GrAdient SOlver for SVM (Pegasos), as proposed by Shalev-Shwartz et al. (2007) – presented here for ease of readability.

In this work, the local models are constructed using the Pegasos algorithm (Shalev-Shwartz et al. (2007)) (presented in Algorithm 2). The implicit assumption is that updating a local model with insight from neighbors is likely to be cheaper than transferring data from all the sites to a central server and also prevents creation of a single point of failure in the distributed setting. We note that algorithms with this flavor have been studied in multi-agent systems (Nedic and Ozdaglar (2009), Nedić et al. (2010)) and optimization literature (Ram et al. (2010)), for general convex optimization problems using gradient descent and projection style optimization algorithms. Our algorithm extends this literature, by explicitly studying the linear SVM in the horizontally partitioned setting with theoretical and empirical analysis.

Model of Distributed Computation. The distributed algorithm evolves over discrete time with respect to a "global" clock. Each site may have access to a local clock or

Table 1 Summary of Notation

$\hat{\mathbf{w}}_i^{(t)}$	site i's weight vector at iteration t
$\hat{\mathbf{w}}_i^{(t+\frac{1}{2})}$	site i's approximate network average update at time t
$\tilde{\mathbf{w}}_i^{(t+\frac{1}{2})}$	site i's update of the local weight vector in the direction of descent
$\mathbf{w}^{(t)}$	Network average weight vector
$\hat{L}_i^{(t)}$	Loss at site i using weight vector $\hat{\mathbf{w}}_i^{(t)}$
$L^{(t)}$	Loss estimated using weight vector $\mathbf{w}^{(t)}$
λ	Learning parameter
$\alpha^{(t)}$	Learning parameter
B	Doubly stochastic transition probability matrix
n_i	Number of training examples at site i
N	Total number of training examples in the network
m	Total number of sites
d	The dimension of weight vector

no clock at all. When the global clock "ticks", each node randomly selects a neighbor and gossips with it. In this formulation, denoting the probability that node i chooses a neighbor j by P_{ij} conditions for convergence can be expressed directly as properties of these probabilities. Furthermore, each site has its own memory and can perform local computation (such as estimating the local weight vector). It stores f_i, which is the estimated local function. Besides its own computation, sites may receive messages from their neighbors which will help in evaluation of the next estimate for the local function.

Communication Protocols. Sites S_i are connected to one another via an underlying communication framework represented by a graph $G(V, E)$, such that each site $S_i \in \{S_1, S_2, \cdots, S_m\}$ is a vertex and an edge $e_{ij} \in E$ connects sites S_i and S_j. Communication delays on the edges in the graph are assumed to be zero. It must be noted that the communication framework is usually expected to be application dependent. In cases where no intuitive framework exists, it may be possible to simply rely on the physical connectivity of the machines, for example, if the sites S_i are part of a large cluster.

GADGET (λ, T, B)

Input: $M_i : n_i \times d$ matrix with real valued inputs at each site S_i; a graph $G(V, E)$ which encapsulates the underlying communication framework ;

Parameters: λ; $\mathcal{T}$; B

Initialization: $\hat{\mathbf{w}}_i^{(1)} = 0$;

for *t = 1 to* $\mathcal{T}$ **do**

(a) Choose an instance uniformly at random from the local dataset M_i.

(b) Set $M_i^+ = \left\{(\mathbf{x}, y) \in M_i : y\langle \hat{\mathbf{w}}_i^{(t)}, \mathbf{x}\rangle < 1\right\}$ { ▷ Identify the instances with non-zero loss.}

(c) Set $\hat{L_i}^{(t)} = y\mathbf{x}$

(d) Set $\alpha^{(t)} = \frac{1}{\lambda t}$ { ▷ Set the learning rate.}

(e) Set $\tilde{\mathbf{w}}_i^{t+\frac{1}{2}} = (1 - \lambda\alpha^{(t)})\hat{\mathbf{w}}_i^{(t)} + \alpha^{(t)}\hat{L_i}^{(t)}$ { ▷ Learn the local loss}

(f) [Optional] Set $\tilde{\mathbf{w}}_i^{(t+\frac{1}{2})} = \min\left\{1, \frac{\frac{1}{\sqrt{\lambda}}}{||\tilde{\mathbf{w}}_i^{(t+\frac{1}{2})}||}\right\}\tilde{\mathbf{w}}_i^{(t+\frac{1}{2})}$

(g) Set $\hat{\mathbf{w}}_i^{(t+\frac{1}{2})} \leftarrow \text{PS}(B, \tilde{\mathbf{w}}^{t+\frac{1}{2}})$ { ▷ Execute Push Sum with local weight vector }

(h) [Optional] Set $\hat{\mathbf{w}}_i^{(t+1)} = \min\left\{1, \frac{\frac{1}{\sqrt{\lambda}}}{||\hat{\mathbf{w}}_i^{(t+\frac{1}{2})}||}\right\}\hat{\mathbf{w}}_i^{(t+\frac{1}{2})}$;

end

Output: $\hat{\mathbf{w}}_i^{(t+1)}$, the weight vector at site S_i.

Algorithm 3: GADGET SVM Algorithm

Algorithm Description. The distributed SVM algorithm (described in Algorithm 3) takes as input the following parameters: λ – the learning rate, $\mathcal{T}$ - the number of iterations to perform, and B – a doubly stochastic transition probability matrix. It proceeds as follows: each site S_i builds a linear SVM model on its local data M_i by learning a weight vector $\hat{\mathbf{w}}_i^{(t)}$ at iteration t of the algorithm. At the beginning, $\hat{\mathbf{w}}_i^{(t)}$ is set to the zero vector. On iteration, t of the algorithm, a random training example (x_i, y_i) is chosen by picking an index $i \in \{1, 2, \cdots |M_i|\}$ uniformly at random. The approximate local loss $\hat{L}_i^{(t)}$ corresponding to the current weight vector $\hat{\mathbf{w}}_i^{(t)}$ is estimated. This is done by replacing the objective in Equation 1 with an approximation based on the training example (x_i, y_i) as follows:

$$\min_{\hat{\mathbf{w}_i^{(t)}}} \frac{\lambda}{2} \parallel \hat{\mathbf{w}}_i^{(t)} \parallel^2 + l(\hat{\mathbf{w}}_i^{(t)}; (\mathbf{x_i}, y_i)), \tag{3}$$

The sub-gradient of the above objective is given by:

$$\nabla_t = \lambda\hat{\mathbf{w}}_i^{(t)} + \mathbb{I}[y_i\langle\hat{\mathbf{w}}_i^{(t)} x_i\rangle < 1] y_i x_i \tag{4}$$

where $\mathbb{I}[y_i\langle\hat{\mathbf{w}}_i^{(t)} x_i\rangle < 1]$ is an indicator function that takes a value 1 if the argument is true and 0 otherwise. The replacement of the gradient estimation over all examples (Equation 1)

with the above mentioned approximation (Equation 3) based on a *single* training example is referred to in literature as SGD. At iteration $t+1$, the local weight vector is updated as follows: $\hat{\mathbf{w}}_i^{(t+1)} \leftarrow \hat{\mathbf{w}}_i^{(t)} - \eta_t \nabla_t$, using the step-size $\eta_t = \frac{1}{\lambda t}$. The update can be written as

$$\hat{\mathbf{w}}_i^{(t)} \leftarrow (1 - \frac{1}{t})\hat{\mathbf{w}}_i^{(t)} + \eta_t \mathbb{I}[y_i \langle \hat{\mathbf{w}}_i^{(t)} x_i \rangle < 1] y_i x_i \tag{5}$$

This is also called the intermediate weight vector, $\tilde{\mathbf{w}}_i^{(t+\frac{1}{2})}$.
Site S_i then gossips the learnt $\tilde{\mathbf{w}}_i^{(t+\frac{1}{2})}$ with a randomly chosen neighbor using protocol Push-Sum (PS). Protocol Push-Sum takes as input the doubly stochastic $m \times m$ matrix B that stores the transition probability between sites, in addition to the approximate weight vector $\tilde{\mathbf{w}}_i^{(t+\frac{1}{2})}$. On termination of the network wide Push-Sum protocol, the local weight vector at site i, $\hat{\mathbf{w}}_i^{(t+\frac{1}{2})}$ is updated by projecting $\hat{\mathbf{w}}_i^{(t+\frac{1}{2})}$ onto the ball of radius $1/\sqrt{\lambda}$ in order to bound the maximum sub-gradient, in the same spirit as in the Pegasos algorithm (Singer and Srebro (2007)). The projection steps are optional and the algorithms proceeds correctly even without this step being explicitly implemented. The following theorem provides a bound on the rate of convergence at each site.

Theorem 1 *Assume that the local function learnt at each site is λ-strongly convex. Then it can be shown that*

$$f(\bar{\mathbf{w}}_i) - f(\mathbf{w}^\star) \leq \frac{2c}{\sqrt{\lambda}} + \frac{c^2 log(T)}{2T\lambda} + \frac{2}{\sqrt{\lambda}}\left(\frac{\gamma R}{\sqrt{\lambda}} + \gamma R\right)$$

where c is the maximum sub-gradient, T the number of iterations, γ the radius of the ball in which the global loss resides and R is the Lipschitz constant.

□

The proof is presented in the Appendix.

The Push-Sum protocol (Kempe et al. (2003b)) deterministically simulates a random walk across G and estimates network sums. If an arbitrary stochastic matrix $B = (b_{i,j})$ is created for the network ensuring that if there is no edge from i to j in G, then $b_{i,j} = 0$, and otherwise B is ergodic and reversible, then Push-Sum converges to a ϵ-relative error solution in $O(\tau_{mix} \log \frac{1}{\epsilon})$, where τ_{mix} is the mixing speed of the Markov Chain defined by B (Dutta and Srinivasan (2018)). Informally, τ_{mix} is the the time until B is "close" to its steady state distribution. An obvious choice for B is to use the random walk on the underlying topology, i.e. $b_{i,j} = \frac{1}{\deg i}$. In general, the sites are not expected to know τ_{mix},

and a simple technique with multiplicative overhead for the sites to calculate a stopping time for Push-Sum (assuming an upper bound on network diameter) has been proposed in work done by Kempe et al. (2003b).

Variants of GADGET SVM. The approximation described in Equation 3 can be implemented in several ways. SGD as described above is a popular technique – however, the processing of one training example in each iteration makes it susceptible to noise. A work around is to use *mini-batch* training (Li et al. (2014), Takáč et al. (2013)) which aggregates multiple examples at each iteration. Assume that $|M_i|$ is divisible by the number of mini-batches m. Then we partition the examples into m mini-batches, each of size $b = \frac{|M_i|}{m}$. Given a random mini-batch of size b, the stochastic update procedure can be implemented on each batch using Equation 1.

Using Mercer Kernels. SVMs can be used with *kernels* instead of directly accessing the feature vectors $\mathbf{x}$. This is possible due to the Representer Theorem (Kimeldorf and Wahba (1971)), which indicates that the optimal solution of Equation 1 can be expressed as a linear combination of training instances. However, instead of considering the features themselves, it is possible to consider implicit mappings of them $\phi(x)$ and use linear functions of these mappings to solve the minimization problem described in Equation 1. Thus, x_j in Equation 1 is simply replaced with $\phi(x_j)$ and l is the loss function defined as $l(\mathbf{w}; (\phi(\mathbf{x}), y)) = \max\{0, 1 - y\langle \mathbf{w}, \phi(\mathbf{x})\rangle\}$ for hinge loss and λ is the SVM regularization parameter. The mapping $\phi(.)$ is never expressed explicitly, but only using a kernel operator $K(x, x') = \langle \phi(x), \phi(x')\rangle$, producing inner products after the mapping $\phi(.)$. The above problem can be solved in the primal while still using kernels (Chapelle (2007), Freund and Schapire (1998)). The kernelized version of the Pegasos Algorithm (Shalev-Shwartz et al. (2007)) can be solved using only kernel evaluations without direct access to the feature vectors. It must be noted that developing a kernelized version of the GADGET algorithm is more involved for the following reasons: (a) Kernel evaluations are required between an example at the site under consideration and all other available examples in the graph G. One way to achieve this is by sending the example to all other sites, computing the kernel evaluation and sending the evaluations back to the site that initiated the process. This process greatly increases the communication cost in the network. (b) One possibility could be to compute kernel evaluations only amongst examples available at site. This would

introduce additional approximations to the kernel evaluation process. We leave the design of kernelized version of the GADGET algorithm for future work.

On Using Sophisticated Gossip Protocols. In recent years, gossip protocols such as Push-Sum and its variants have been used for solving decentralized convex and non-convex (Blot et al. (2019), Daily et al. (2018), Scardapane et al. (2016)) problems. Would it be useful to use a more sophisticated gossip protocol to design a decentralized SVM algorithm?

One line of research has explored the use of communication compression such as gradient compression (Lin et al. (2017), Alistarh et al. (2017)) and sparsification (Wangni et al. (2018), Konecny and Richatrik (2018)) for reducing the amount of data transferred in the network. Tang et al. (2018) propose two strategies for gradient compression – the first called difference compression, involves compressing the difference of local models between two successive iterations and the other an improvement based on extrapolation. However, their analysis only covers unbiased compression operators with very (unreasonably) high accuracy constraints. Low accuracy and biased compression operators are studied in Koloskova et al. (2019b), Alistarh et al. (2017), Wen et al. (2017), Zhang et al. (2017). We advocate that careful choice be made of such algorithms since reduction in communication comes at the cost of accuracy. Clearly, techniques that introduce noise and do not preserve averages over successive iterations will distract from the objective of learning a distributed SVM.

Another line of research studies convergence of gossip protocols with delays (Carli et al. (2007), Agarwal and Duchi (2011)). Carli et al. (2007) provide a bound on the time required to reach consensus, which is a function of system parameters (such as the delay bound) and bound on intercommunication intervals. Agarwal and Duchi (2011) show that for smooth stochastic problems, the delays are asymptotically negligible. The function studied in this work is non-smooth and to the best of our knowledge, we are not aware of any work that studies the effects of delays from gossiping on non-smooth objectives.

5. Experimental Results

5.1. Aims

Our objective is to investigate empirically the utility of the consensus based SVM algorithm (and its variants) we have described. We intend to examine if there is empirical support for the conjecture that the performance of the *Distributed* model is better than that of the *Centralized* – our centralized model is based on the execution of the Pegasos algorithm (Shalev-Shwartz et al. (2007)) on the entire dataset at a single site. We are assuming that

the performance of a model-construction method is given by the pair (A, T) where A is an unbiased estimate of the predictive accuracy of the classifier, and T is an unbiased estimate of the time taken to construct a model. In all cases, the time taken to construct a model does not include the time to read the dataset into local memory. Comparison of pairs (A_1, T_1) and (A_2, T_2) will simply be lexicographic comparisons. Furthermore, we compare the time taken by the distributed algorithm to "converge" on all sites against a centralized execution on a single site by estimating *Speed-up*. Speed-up is defined as:

$$\text{Speed-up} = \frac{\text{Time taken by the centralized algorithm to converge}}{\text{Time taken by the distributed algorithm to converge on all sites}} \quad (6)$$

As explained later in this section, GADGET SVM is an anytime algorithm and therefore the definition of *convergence* is relative to the time when the algorithm was interrupted.

In addition to analyzing the effectiveness of the distributed algorithm against a centralized one, we study its performance (along with its variants) by comparing it to state-of-the-art scalable SVM algorithms – SVM^{Perf}, LIBLINEAR, LIBSVM and an Alternating Direction Method of Multipliers (ADMM) algorithm for SVMs. None of the state-of-the-art algorithms are inherently distributed. To enable meaningful comparisons with GADGET SVM, we execute them on horizontally partitioned data and average the performance on each partition. We refer to this as a *pseudo distributed* implementation, since no exchange / sharing of information amongst partitions is possible in this setting unlike GADGET SVM which learns from other partitions. We also study the performance of GADGET SVM relative to other gossip-based SVMs (such as ASG, Dist-Eval and CHOCO-SGD).

Finally, the utility of using a scalable SVM algorithm (GADGET SVM) for classification is studied in comparison to other techniques well known in literature such as thresholded (penalized) regression models, decision tree based learning by construction of random forests, boosting based methods and multilayer perceptrons. A comparison of this kind is aimed at providing end users with knowledge of the performance of the distributed algorithm against other scalable algorithms, not necessarily SVM-based.

In the following sections, we first describe the data set, algorithms and methodology of empirical evaluation before presenting extensive results along the lines indicated above.

5.2. Materials

Data The real world datasets used for experiments reported in this paper are organized into two groups – small and large. The first three (described below) are small datasets while the remaining are large.

- *Gisette:* This dataset was obtained from the LibSVM binary data collection `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html`. There are 6000 labeled examples of which 5000 are used for training. The training data is oversampled to compensate for under-representation of one class label. It has 5000 features.

- *Sido:* This dataset was obtained from the first causality challenge `http://www.causality.inf.ethz.ch/repository.php`, the goal of which was to make predictions under manipulations. SIDO (SImple Drug Operation mechanisms) contains descriptors of molecules, which have been tested against the AIDS HIV virus. The target values indicate the molecular activity (+1 active, -1 inactive). The task is to uncover causes of molecular activity among the molecule descriptors. This would help chemists in the design of new compounds, retaining activity, but having perhaps other desirable properties (less toxic, easier to administer). This dataset is semi-artificial: it contains both real variables and artificial variables (probes) – there are 4932 molecular descriptors. Of the 12678 labeled examples, 11000 are used for training the model and the rest are used for testing.

- *Quantum:* This dataset has been obtained from the 2004 KDD Cup website `http://osmot.cs.cornell.edu/kddcup/`. The task is to learn a classification rule that differentiates between two types of particles generated in high energy collider experiments. It is a binary classification problem with 78 attributes and has several attributes with missing values. There are 50000 labeled examples of which 40000 are used for training.

- *Protein:* This dataset was obtained from the 2004 KDD Cup website `http://osmot.cs.cornell.edu/kddcup/`. The goal is to predict which proteins are homologous to a native sequence. There are 145,751 labeled examples each with 74 features. 100000 examples are used for training and the rest for testing.

- *Adult:* This dataset has been extracted from the census bureau database found at `http://www.census.gov/ftp/pub/DES/www/welcome.html`. The task is to predict whether a person makes above $50000 a year using 123 attributes that include race, sex, occupation and others. This dataset comprises of 32562 labeled examples of which 25000 were used for training.

- *Covertype:* The task is to predict forest cover type (spruce or fir) in the dataset `http://archive.ics.uci.edu/ml/datasets/Covertype` of Blackard, Jock and Dean, which has 54 features and a sparsity of 22.22%. There are $581,012$ labeled examples of which $500,000$ examples were used for training.

- *Real-Sim:* This dataset was obtained from UseNet articles from four discussion groups for simulated and real auto-racing and aviation. It is available from `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html`. The task is to differentiate between real and simulated data. Of the 72309 labeled examples, 60000 are used for training the distributed model. The number of attributes is 20958.

5.3. Algorithms and Machines

The consensus algorithm (GADGET SVM) has been implemented on Peersim (`http://peersim.sourceforge.net/`), a peer-to-peer network(P2P) simulator. This software[2] allows simulation of the P2P network by initializing sites and the communication protocols to be used by them. GADGET implements a cycle driven protocol that has periodic activity in approximately regular time intervals. Sites are able to communicate with others using the Push-Sum protocol (described in Section 3.3). The experiments are performed on a single site, on a DELL E7-4830 architecture, equipped with 32 x 2.13GHz Intel Xeon CPU E7-4830 Processor Cores; instruction and data cache sizes of 24576 KB; a main memory size of 512 GB and 128 GB RAM. The computational resources were provided by (REMOVED FOR BLIND REVIEW).

5.4. Method

The following method was executed on each of the datasets mentioned above (Section 5.2):

1. A network of k sites is setup using the Peersim simulator.

2. The train and test sets are split into k different files, containing approximately equal number of instances and distributed amongst the sites.

3. The GADGET algorithm (Section 4) is executed on each site independently until the local weight vectors converge i.e. they do not change more than an user-defined parameter ϵ. The local models are then used to determine the primal objective and the test error on the corresponding test sets. The results are averaged over all the sites in the network. This entire process is executed several times, and an average value of primal objective and test error is obtained.

4. The optimization problem (posed in Equation 1) is solved using three different techniques: (a) Full Gradient Descent (GD) (b) SGD (c) Mini-Batch SGD with varying batch sizes.

[2] The code is available from (REMOVED FOR BLIND REVIEW)

5. The Pegasos algorithm is executed on the entire dataset – this is called the *centralized setting* and serves as a baseline against which the performance of GADGET is evaluated. All three optimization methods (GD, SGD and Mini-Batch) are studied in the context of the centralized setting also.

The following details are relevant:

1. $k = 10$ in the experiments reported in this paper.
2. The accuracy and time reported in Table 2 are averages (and corresponding standard deviations) over all the sites in the network.
3. The user-defined ϵ parameter is set to be 0.001.
4. For Mini-Batch training, the batch size is varied as follows: $b = 10, 100, 1000$.

5.5. Results

5.5.1. Comparison between distributed and centralized algorithms Table 2 summarizes the performance of the distributed primal SVM algorithm (GADGET) against its centralized counterpart. To compute the centralized scores, it is assumed that the entire data set can fit into main memory. For each of the settings – distributed and centralized, we solve the optimization problem using the standard SGD along with its variants – Mini-Batch with varying batch sizes (these are set to 10, 100 and 1000) and a full Gradient Descent (GD). The average accuracy obtained on the test data distributed among sites is reported for the distributed algorithm.

It is observed that in four datasets (Sido, Quantum, Adult, and Protein), the SGD converges the fastest and also has the best accuracy amongst the competing techniques – the only exception being the Protein dataset, wherein the accuracy is much lower. For the Gisette, Covertype and Real-Sim datasets the mini-batch variants with different batch sizes (10 or 100) enables fast convergence with comparable accuracy as that of the centralized algorithm. The performance of the Mini-Batch with small batch size ($b = 10$) closely mimics the performance of SGD, albeit with a smoothing effect while the Mini-Batch with large batch size ($b = 1000$) resembles the performance of GD being fundamentally much slower than the SGD method.

A more detailed discussion of the performance of the distributed algorithm is in order. As briefly introduced in Section 1, the GADGET SVM algorithm is an *anytime* algorithm (Zilberstein (1993)). An implementation of an anytime algorithm typically has a mapping from a set of inputs onto a *set* of outputs. For each input that specifies the problem instance,

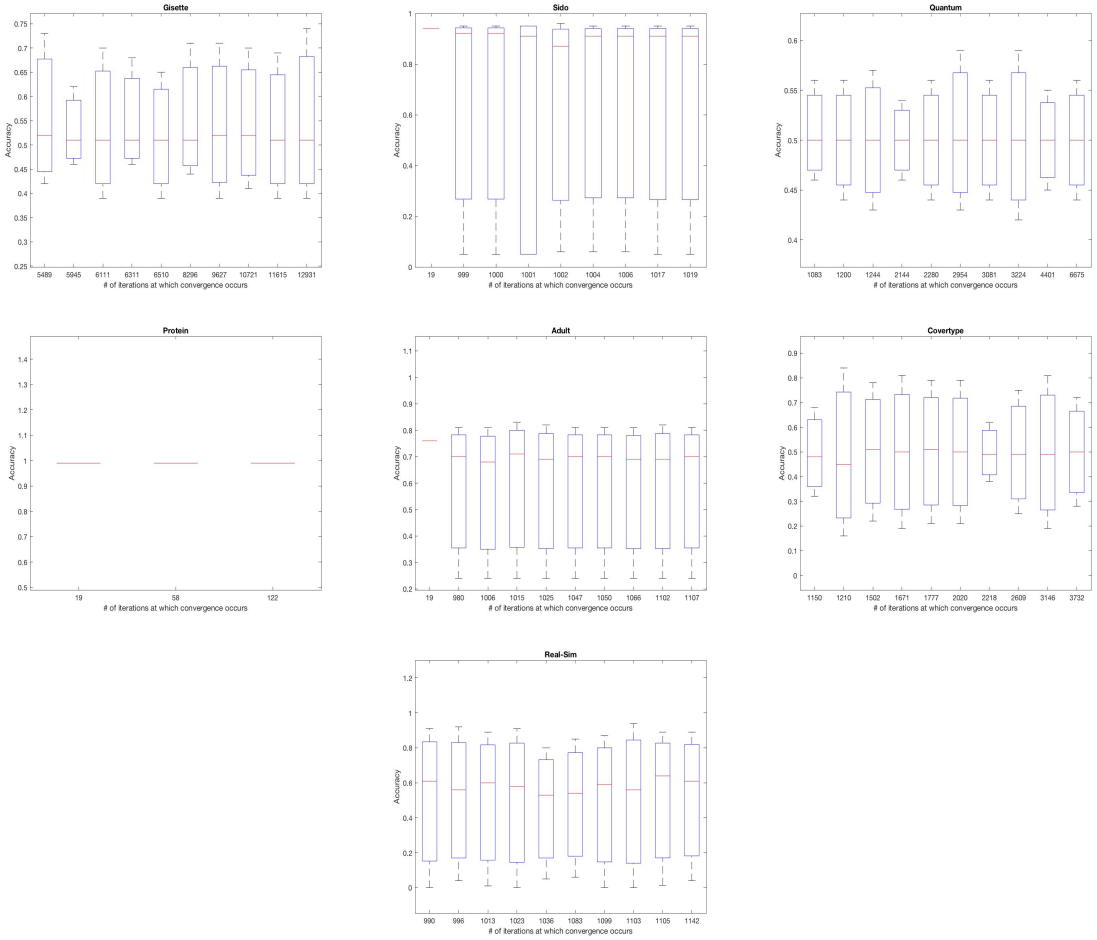

Figure 1 Variation in accuracy of classification at sites until convergence for Gisette, Sido, Quantum, Protein, Adult, Covertype and Real-Sim datasets. The box plot is replaced with a redline if there is no variance to report. Convergence at a site is reported if the norm of the weight vector falls below $\epsilon = 0.001$ in successive iterations.

there is a particular element in the output set that can be regarded as the *correct* solution. For example, in the context of an anytime SVM algorithm, the input can be samples from a dataset of choice; the set of outputs could be the accuracy obtained depending on what part of the data it has seen so far. The utility of the set of outputs comes from the fact that the algorithm can be *interrupted* at any time and still produce results of a certain quality (Hence giving it the name "anytime" algorithm.). In our context, interruption refers to halting the execution of an algorithm when the results are deemed *good enough* by the end user of the system. In the consensus based SVM algorithm, the interruption is provided to a site when the norm of the weight vector in successive iterations falls below a user-defined tolerance. This implies, different sites can "converge" to a solution at different iterations. After convergence, a site can continue to receive examples if so desired. The graph of all

sites is said to have converged when all the sites in it have converged at least once (We estimate the maximum number of iterations required for all sites to converge.). A question that remains unanswered in our discussion so far is "How many successive iterations should be examined (wherein the norm of the weight vector falls below ϵ) for "convergence"?". The results provided in Table 2 consider three successive iterations as this is typically seen to be sufficiently stable in our empirical analysis.

To allow for efficient control of anytime algorithms, the change in performance over time has to be characterized quantitatively. Performance profiles (PPs) (Boddy and Dean (1994), Horvitz (1987), Zilberstein (1996)) are used to describe the expected output quality as a function of execution time T. PPs are constructed empirically by collecting statistics on the performance of an algorithm over many input examples which then forms the quality map of the algorithm. Each point (A, T) represents a situation/instance for which quality A was achieved for runtime T. In this context, a *statistical performance profile* records both upper and lower bounds and expected output quality at time T. The statistical performance profile for GADGET SVM comprising of the minimum, average and maximum accuracy of classification at a given time T is reported in Figures 1 at the time when sites "converge" to a solution. Our results reveal that the statistical performance profile of the Protein data set remains constant (high accuracy) at all sites in the network. For the Gisette, Quantum, Adult and Covertype data sets the accuracy remains between $30-70\%$, with the average performance being above 50% in all cases. Sido and Real-Sim show a large variability in performance with the maximum reaching close to 90%, but the lower bound could be close to zero if the sites have examples that do not help to learn the classification boundary at all. The results emphasize the need to tune and interrupt the anytime algorithm appropriately to collect performance statistics depending on the application requirements.

5.5.2. Comparison of GADGET SVM with state-of-the-art Gossip-based SVM algorithms We compare the performance of GADGET SVM to three state-of-the-art gossip-based distributed SVM algorithms: (a) ASG SVM (Flouri et al. (2009)) (b) Distributed evaluation (Dist-Eval) using gossip (Wang et al. (2010)) and (c) CHOCO-SGD Koloskova et al. (2019b,a). All of these algorithms along with GADGET SVM have been implemented in the Peersim simulator. For the ASG SVM algorithm, we varied the threshold of the selection set between 0.001% and 3% of the local training data. Since increase in threshold value does not provide enhanced classification performance, we report results

in Table 3 with threshold=0.001%. In the case of Dist-Eval, the β parameter of the consensus filter is set to 1 as suggested by Wang et al. (2010). For CHOCO-SGD, we have implemented compression using Quantized SGD (Alistarh et al. (2017)) with parameter $b = 3$ and use it to solve a convex objective, instead of the popular non-convex formulation. We measured the speed-up provided by GADGET SVM using the following formula: $Speed - up_{GADGET} = \frac{\text{Time taken by Algorithm Compared}}{\text{Time taken by GADGET SVM}}$. Our results indicate GADGET SVM is at least 1.5 times faster than ASG SVM on all datasets except Quantum – often several orders of magnitude faster (for e.g. Sido, Protein and Real-sim). GADGET SVM is also at least 10 times faster than Dist-Eval – often several orders of magnitude faster (for e.g. Gisette, Protein, Covertype, Real-sim) on all the datasets examined in this work. This is primarily because, in Dist-Eval, solving the SVM in the dual at a local site in itself is rather compute intensive and contributes to the overall time taken to obtain a prediction. When the gradients are compressed using the CHOCO-SGD algorithm, a speedup is observed in all datasets except Quantum and Covertype datasets for comparable accuracy. Furthermore, an estimate of the size of messages exchanged in each iteration reveals the following – GADGET SVM simply exchanges weights and therefore message complexity is $O(d)$; ASG SVM has a message complexity $\chi \times O(d)$ where χ is dependent on the size of the Selected Set and can be substantially large if a sufficiently large threshold is used – for example, Flouri et al. (2009) use 30% of data in their experiments; Dist-Eval has a message complexity of $O(n_i)$ since it exchanges the predictions at each iteration while CHOCO-SGD has a message complexity of $O(\frac{1}{\sqrt{mT}})$.

While Section 3.2.4 discusses two more closely related gossip-based SVM algorithms (Ormandi et al. (2013) and Kim et al. (2015)), we do not use them for comparison with GADGET SVM. This is primarily due to the following reasons: (a) Ormandi et al. (2013) can be viewed as a special case of the current algorithm when the models being studied are linear and (b) Kim et al. (2015) state that " [their] optimal algorithm is identical to the Wolfe dual formulation of a modified formulation of SVM which is a scaled version of ν-SVM". We surmise that the time complexity of such an algorithm is similar to Dist-Eval presented above (Implemented with C-SVM) since it requires solution of the dual formulation (Both C and ν SVM are implemented in the LibSVM package (Chang and Lin (2011b)). In addition, the message complexity is $\varrho \times O(d)$ where ϱ is the maximum number of extremum points of the convex hull at a site. This implies the following – while

it is possible to learn a non-linear SVM using gossip based protocols, these algorithms are much more compute intensive and require significant communication (including transfer of data) in the network than what is required to learn the linear kernel.

5.5.3. Comparison of GADGET SVM with state-of-the-art SVM algorithms We compare the performance of GADGET SVM to several state-of-the-art SVM algorithms - (a) SVM^{Perf} (Joachims and Yu (2009), Joachims (2006)) (b) LIBLINEAR (Fan et al. (2008)) (c) LIBSVM (Chang and Lin (2011a)) and a pseudo distributed ADMM algorithm(Boyd et al. (2011)). The choice of these algorithms is motivated by the following reasons: GADGET SVM is designed for primal SVM formulations with L_1 loss (hinge) and L_2 regularization. Consequently, we preferred to compare it against state-of-the-art primal solvers with L_1 loss and L_2 regularization wherever available. For LIBLINEAR, a very popular library for large scale linear SVM classification, we compared our results with a dual formulation of L_2-regularized L_1-loss support vector classifier. This is because the primal solver (Newton's method) in LIBLINEAR relies on differentiability and L_1 loss is not differentiable. In the case of LIBSVM, we used a C-SVC (Given instances $x_i, i = 1, \cdots, l$ with labels $y_i \in \{+1, -1\}$ the main task in training C-SVC involves solving the following quadratic optimization problem: $\min_\alpha f(\alpha) = \frac{1}{2}\alpha^T Q\alpha - e^T\alpha$, subject to $0 \leq \alpha_i \leq C, i = 1, \cdots, l, y^T\alpha = 0$, where e is the vector of all 1's, C is the upper bound of all variables, Q is an $l \times l$ symmetric matrix with $Q_{ij} = y_i y_j K(x_i, x_j)$ and $K(x_i, x_j)$ is the Kernel function.) formulation which solves the dual formulation using a quadratic program. Finally, GADGET SVM uses a SGD method for optimization; its performance is also inherently compared against other solvers in state-of-the-art algorithms, such as Sequential Minimal Optimization (SMO) used in LIBSVM, Newton's Method in LIBLINEAR and ADMM.

None of the state-of-the-art SVM algorithms discussed above are distributed; so we simulated a distributed setting as follows: each site executed the algorithm in question on training data available at that site and reported the performance on the test set. The average performance of all the sites is reported in Table 4. This in effect means that these algorithm will execute in a *pseudo distributed* mode i.e. without communication amongst the sites. To the best of our knowledge, there are no known theoretical guarantees on global convergence in these settings for any of the above algorithms, although Dist ADMM has been studied in the consensus setting (Boyd et al. (2011)). Another important point to consider is that GADGET SVM is the only *anytime* algorithm – in other words, the

performance reported for all other algorithms is when the algorithm has converged. In the case of GADGET SVM, convergence is chosen to be the first time when the norm of the weight vector in (three) successive iterations falls below the user defined tolerance (0.001) at each site in the network. The choice of three successive iterations is purely empirical, and can be tuned appropriately for other applications.

Based on the results presented in Table 4 it can be seen that GADGET SVM has significant benefits in time compared to state-of-the-art SVM solvers, an artifact of the distributed nature of the algorithm. For the Sido, Protein and Adult datasets, the time at which the performance metric was obtained by interrupting the algorithm was sufficient to reflect comparable performance to the state-of-the-art algorithms. For the other datasets, Gisette, Quantum, Covertype and Real-Sim more training is likely to enhance performance. Interestingly, the Gisette data, a handwritten digit recognition problem used to separate two classes – those representing the digit 4 and 9, is known to contain a large number of non-informative features which tend to mislead the classifier (Sharma et al. (2017), Guyon et al. (2004)). This is one of the primary reasons for the sub-optimal performance of GADGET SVM on the Gisette data – in other words, GADGET SVM is not designed to handle non-informative features. Furthermore, Quantum, CoverType and Real-Sim are known to be sparse datasets (Hong et al. (2019), Shalev-Shwartz et al. (2007)) and some features contain missing values. GAGDET SVM is not specifically designed to deal with data sparsity, unlike some of the other sparse SVM solutions available in literature (such as Hong et al. (2019), Li et al. (2015), Steinwart (2003)). Furthermore, in terms of accuracy of classification, LIBLINEAR is very accurate on its own. However, the results from LIBLINEAR presented here use the dual formulation of the problem solved using the second-order Newton method (GADGET uses a first order method for solving the optimization problem. A second order method is typically much faster than a first order method – however, computation of the Hessian is not always straightforward, especially for large problems. The Hessian may not even exist.); similarly LIBSVM solves the dual using a quadratic program – thereby warranting careful comparison to the results obtained from GADGET SVM.

When comparing the consensus algorithm against a state-of-the-art ADMM solver (Boyd et al. (2011)), we first note that the ADMM solver is not a completely decentralized one. We follow the guidelines presented in Boyd et al. (2011) and construct subsystems capable of solving small convex problems, where "small" indicates that the problem is solvable using

a serial algorithm. To this end, each site is allowed to solve its own ADMM formulation on its local data partition and the results are averaged over all the sites in the network. Table 7 (in the Appendix B) presents the parameters of the ADMM algorithm used for each dataset. It is important to note that the authors in Boyd et al. (2011) envision a completely decentralized general ADMM solver as follows "In the general form consensus case, [$\cdots$],a decentralized implementation is possible [$\cdots$]; each set of subsystems that share a variable can communicate among themselves directly. In this setting, it can be convenient to think of ADMM as a message-passing algorithm on a graph, where each site corresponds to a subsystem and the edges correspond to shared variables.". However, a system which implements this idea does not exist to date (Personal communication with the authors of Boyd et al. (2011).). Several parallel ADMM solvers are available and popularly used using Message Passing Interface (MPI), Apache Hadoop's Map Reduce framework and their variants. However, the architecture used for such implementations uses a master-slave framework which is drastically different from the peer-to-peer setting studied in this paper. Hence we refrain from comparing performance of the ADMM solver in those architectures with GADGET SVM. It was observed that the consensus based algorithm beats the ADMM algorithm in six of the seven datasets in terms of accuracy. The time taken to converge to a solution is much larger for the ADMM solver than the SGD based algorithm presented here.

These results suggest that the GADGET SVM algorithm can provide accuracy comparable to state-of-the-art solvers (often) much faster and could be very useful in scenarios where getting approximate results fast is much more desirable than obtaining the correct solution using a lot of compute power. This is very useful in distributed and resource constrained environments (such as in networks of IoT devices) where centralization of data may not be an option and distributed algorithms are the norm.

Gisette					
Method	GADGET		Pegasos (Centralized)		Speed-up
	Time (secs)	(Acc. %)	Time (secs)	(Acc.%)	
GD	8111.21 ± 1725.64	0.69 ± 0.16	68801.91	0.96	10617.58
SGD	3.91 ± 2.10	0.65 ± 0.07	1.34	0.61	0.34
Mini-Batch (b=1000)	6.48 ± 1.23	0.59 ± 0.10	10038.11	0.88	1549.09
Mini-Batch (b=100)	$\mathbf{0.53 \pm 0.85}$	$\mathbf{0.71 \pm 0.15}$	59665.88	0.96	112577.13
Mini-Batch (b=10)	3.91 ± 2.1	0.65 ± 0.07	67706.82	0.5	17316.32
Sido					
GD	6.72 ± 4.54	0.92 ± 0.02	4.27	0.94	0.64
SGD	$\mathbf{0.18 \pm 0.51}$	$\mathbf{0.94 \pm 0.009}$	0.008	0.94	0.04
Mini-Batch (b=1000)	44.40 ± 1.42	0.91 ± 0.13	8.56	0.9	0.19
Mini-Batch (b=100)	8.38 ± 5.61	0.92 ± 0.02	0.96	0.92	0.11
Mini-Batch (b=10)	4.99 ± 1.32	0.92 ± 0.04	0.16	0.89	0.03
Quantum					
GD	3.9 ± 2.17	0.5 ± 0.021	1.2	0.48	0.31
SGD	$\mathbf{0.04 \pm 0.049}$	$\mathbf{0.51 \pm 0.034}$	1.18	0.48	29.5
Mini-Batch (b=1000)	0.99 ± 0.44	0.5 ± 0.025	1.26	0.48	1.27
Mini-Batch (b=100)	0.139 ± 0.09	0.049 ± 0.019	1.25	0.48	8.99
Mini-Batch (b=10)	0.046 ± 0.008	0.5 ± 0.029	1.28	0.48	27.82
Protein					
GD	0.05 ± 0.034	0.84 ± 0.32	34.45	0.98	689
SGD	$\mathbf{0.0022 \pm 0.001}$	$\mathbf{0.98 \pm 0.003}$	0.0024	0.99	1.09
Mini-Batch (b=1000)	0.006 ± 0.005	0.88 ± 0.31	0.368	0.977	61.33
Mini-Batch (b=100)	$\mathbf{0.0020 \pm 0.0015}$	$\mathbf{0.89 \pm 0.31}$	0.079	0.98	39.5
Mini-Batch (b=10)	0.0018 ± 0.001	0.89 ± 0.310	0.0039	0.98	2.17
Adult					
GD	2333.06 ± 1081.29	0.77 ± 0.05	24155.12	0.77	10.35
SGD	$\mathbf{37.44 \pm 3.32}$	$\mathbf{0.79 \pm 0.03}$	46.92	0.76	1.25
Mini-Batch (b=1000)	1502.15 ± 789.68	0.77 ± 0.05	709.92	0.75	0.47
Mini-Batch (b=100)	1502.15 ± 789.68	0.77 ± 0.05	709.92	0.75	0.47
Mini-Batch (b=10)	784.78 ± 883.88	0.75 ± 0.03	30.95	0.61	0.04
Covertype					
GD	23.7 ± 1.62	0.49 ± 0.02	340.62	0.46	14.37
SGD	0.092 ± 0.021	0.49 ± 0.022	0.07	0.51	0.76
Mini-Batch (b=1000)	0.75 ± 0.09	0.49 ± 0.02	1.22	0.52	1.63
Mini-Batch (b=100)	0.12 ± 0.016	0.49 ± 0.02	0.09	0.5	0.75
Mini-Batch (b=10)	$\mathbf{0.08 \pm 0.009}$	$\mathbf{0.49 \pm 0.015}$	0.05	0.52	0.625
Real-Sim					
GD	1415.17 ± 498.67	$\mathbf{0.79 \pm 0.03}$	170.16	0.84	0.12
SGD	234.46 ± 35.69	0.73 ± 0.08	605.91	0.72	2.58
Mini-Batch (b=1000)	528.97 ± 367.32	0.75 ± 0.05	416.37	0.68	0.79
Mini-Batch (b=100)	409.13 ± 162.93	0.38 ± 0.14	842.16	0.41	2.06
Mini-Batch (b=10)	$\mathbf{6.67 \pm 1.14}$	$\mathbf{0.27 \pm 0.12}$	37.19	0.13	5.58

Table 2 Comparison of the performance of GADGET SVM and centralized Pegasos on Gisette, Sido, Quantum, Protein, Adult, Covertype, and Real-Sim data using the following metrics: accuracy of classification, time taken for the distributed algorithm versus the centralized, and percentage of speed-up in execution time. The accuracy and time reported for GADGET is the mean over all the sites in the network. Convergence in the consensus algorithm occurs when the norm of the weight vector in successive iterations falls below the user defined tolerance $\epsilon = 0.001$ at each site in the network. The numbers in bold fonts indicate the lowest time to convergence and best accuracy in the distributed setting.

Dataset	GADGET SVM		ASG SVM		Dist-Eval		CHOCO-SGD	
	Time (secs)	Acc (%)	Time (secs) (Speed-up)	Acc (%)	Time (secs)(Speed-up)	Acc (%)	Time (secs)(Speed-up)	Acc (%)
Gisette	3.91 ± 2.10	0.65 ± 0.07	34.71 ± 13.72 (8.87)	0.47 ± 0.08	1296.75 ± 1517.99 (331.65)	0.46 ± 0.005	25.74 ± 0.48 (6.54)	0.52 ± 0.04
Sido	0.18 ± 0.51	0.94 ± 0.009	7.5 ± 2.39 (41.66)	0.96 ± 0.02	18.5 ± 1.70(102.78)	0.99 ± 0.01	3.28 ± 2.73 (18.21)	0.96 ± 0.02
Quantum	1.24 ± 1.7	0.52 ± 0.034	0.46 ± 0.11 (0.37)	0.52 ± 0.05	238 ± 334.73(191.94)	0.68 ± 0.12	0.05 ± 0.01 (0.04)	0.5 ± 0.04
Protein	0.0022 ± 0.001	0.98 ± 0.003	1.54 ± 0.48 (700)	0.99	143.19 ± 100.35 (65086.36)	0.99	0.01 ± 0.0 (4.55)	0.99 ± 0.0
Adult	0.04 ± 0.003	0.79 ± 0.025	0.06 ± 0.02 (1.5)	0.70 ± 0.2	0.43 ± 0.06(10.75)	0.87 ± 0.04	0.06 ± 0.02 (1.45)	0.78 ± 0.03
Covertype	0.81 ± 0.83	0.41 ± 0.27	2.05 ± 0.57 (2.53)	0.69 ± 0.36	601.52 ± 731.85 (742.62)	0.51 ± 0.32	0.07 ± 0.07 (0.08)	0.42 ± 0.41
Real-sim	0.23 ± 0.04	0.73 ± 0.08	6037.47 ± 6373.26 (262.49)	0.38 ± 0.40	2587.39 ± 6367.49 (11249.52)	0.85 ± 0.03	376.91 ± 11.76 (1638.72)	0.46 ± 0.13

Table 3 Comparison of performance of GADGET SVM with state-of-the-art gossip-based algorithms. ASG-SVM refers to Adaptive Selective Gossip(ASG) SVM algorithm (Flouri et al. (2009)), Dist-Eval refers to the algorithm presented by Wang et al. (2010) and CHOCO-SGD refers to the algorithm presented by Koloskova et al. (2019a).

Dataset	GADGET SVM		SVM^{Perf}		$LIBLINEAR$		$LIBSVM$		Dist-ADMM	
	Time (secs)	(Acc. %)	Time (secs)	(Acc.%)	Time (secs)	(Acc. %)	Time (secs)	(Acc. %)	Time (secs)	(Acc. %)
Gisette	3.91 ± 2.10	0.65 ± 0.07	0.078 ± 0.014	0.67 ± 0.125	1.3 ± 0.02	0.98 ± 0.01	7.14 ± 0.11	0.98 ± 0.01	0.88 ± 0.02	0.54
Sido	0.18 ± 0.51	0.94 ± 0.009	0.08 ± 0.013	0.94	0.82 ± 0.01	0.95 ± 0.004	2.10 ± 0.17	0.94	0.84 ± 0.01	0.94
Quantum	1.24 ± 1.7	0.52 ± 0.034	7.37 ± 9.42	0.53 ± 0.017	1.41 ± 0.02	0.62 ± 0.04	2.63 ± 0.09	0.61 ± 0.01	48.86 ± 61.43	0.32 ± 0.02
Protein	0.0022 ± 0.001	0.98 ± 0.003	0.43 ± 0.60	0.98 ± 0.004	1.41 ± 0.42	0.98 ± 0.004	35.73 ± 12.12	0.99	1.48 ± 0.13	0.63 ± 0.32
Adult	0.04 ± 0.003	0.79 ± 0.025	0.006 ± 0.01	0.76	0.08 ± 0.006	0.84 ± 0.006	0.74 ± 0.02	0.83 ± 0.005	25.56 ± 8.77	0.05 ± 0.03
CoverType	0.81 ± 0.83	0.41 ± 0.27	9.916 ± 3.47	0.72	24.86 ± 0.68	0.45 ± 0.08	974.37 ± 83.69	0.4	832.65 ± 918.60	0.59 ± 0.02
Real-Sim	0.23 ± 0.04	0.73 ± 0.08	102.88 ± 4.19	0.82 ± 0.01	18.43 ± 3.83	0.87 ± 0.02	716.96 ± 229.62	100	NA	NA

Table 4 Comparison of the performance of GADGET SVM (SGD), SVM^{Perf}, LIBLINEAR, LIBSVM and pseudo distributed ADMM. All algorithms are executed individually on each site of the network. An NA indicates that the ADMM solver was not able to converge within 86400 secs. For experiments with SVM^{Perf}, $C = 0.0001$ and the 1-slack algorithm (primal) described in Joachims (1999) is used.

Dataset	Lasso		Ridge		Elastic Net	
	Time (secs)	(Acc.%)	Time (secs)	(Acc. %)	Time (secs)	(Acc. %)
Gisette	1.07 ± 0.10	0.98 ± 0.01	11.20 ± 2.45	0.98 ± 0.01	1.16 ± 0.08	0.99 ± 0.01
Sido	1.09 ± 0.08	0.95 ± 0.01	21.89 ± 0.42	0.95 ± 0.01	1.12 ± 0.09	0.95 ± 0.01
Quantum	0.38 ± 0.07	0.7 ± 0.02	0.45 ± 0.01	0.7 ± 0.01	0.38 ± 0.06	0.7 ± 0.02
Protein	12.4 ± 2.94	0.99 ± 0.004	1.51 ± 0.03	0.99 ± 0.002	4.96 ± 0.92	0.99 ± 0.001
Adult	0.59 ± 0.17	0.84 ± 0.01	0.46 ± 0.01	0.84 ± 0.01	0.61 ± 0.15	0.84 ± 0.01
CoverType	3.33 ± 0.13	0.81 ± 0.001	4.95 ± 0.15	0.78 ± 0.07	3.39 ± 0.17	0.81 ± 0.001
Real-Sim	28.66 ± 1.23	0.71 ± 0.02	385.53 ± 205.48	0.61 ± 0.02	30.97 ± 4.47	0.71 ± 0.02

Table 5 Comparison of the performance of GADGET SVM with penalized linear regression – Lasso, Ridge and Elastic Net models. The GADGET SVM results are presented in Table 4 above only for brevity.

Dataset	Random Forest		Boosting		MultiLayer Perceptrons	
	Time (secs)	(Acc.%)	Time (secs)	(Acc. %)	Time (secs)	(Acc. %)
Gisette	0.25 ± 0.12	0.93 ± 0.02	57.97 ± 3.84	0.95 ± 0.03	827.79 ± 37.80	0.72 ± 0.08
Sido	0.14 ± 0.01	0.94 ± 0.01	48.20 ± 2.62	0.94 ± 0.01	353.14 ± 7.69	0.97 ± 0.02
Quantum	0.47 ± 0.04	0.71 ± 0.10	2.24 ± 0.1	0.69 ± 0.01	21.49 ± 0.74	0.5 ± 0.02
Protein	1.00 ± 0.08	0.99 ± 0.0	45.65 ± 110.04	0.99 ± 0.0	78.41 ± 3.91	0.99 ± 0.003
Adult	0.23 ± 0.02	0.83 ± 0.05	11.11 ± 30.38	0.83 ± 0.004	31.44 ± 1.15	0.76 ± 0.003
CoverType	11.94 ± 1.85	0.88 ± 0.03	67.3 ± 0.47	0.8 ± 0.06	292.69 ± 10.69	0.61 ± 0.2
Real-Sim	32.58 ± 2.86	0.54 ± 0.04	2438.98 ± 276.49	0.8 ± 0.20	28284.5 ± 13821.65	0.99 ± 0.008

Table 6 Comparison of the performance of GADGET SVM with Random Forest, Boosting and MultiLayer Perceptron.The GADGET SVM results are presented in Table 4 above only for brevity.

5.5.4. On the utility of using SVMs for empirical analysis The next set of experiments were conducted to justify the use of SVMs as a viable tool for classification, given the existence of several other classification algorithms in literature such as thresholded (penalized) linear regression (Friedman et al. (2010)), random forests (Breiman (2001)), boosting (Schapire and Freund (2012)) and MultiLayer Perceptrons (MLP). Such comparisons are useful to provide the readers with choice of different classification methods. For implementing thresholded (penalized) linear regression we used the glmnet (`https://web.stanford.edu/~hastie/glmnet/glmnet_beta.html`) package in R. For experiments reported here $\alpha = 0.8$. In experiments with Random Forests, the number of trees is kept constant at ten for all datasets. In the AdaBoost algorithm a Decision Stump was used, without resampling. For the construction of an MLP, a learning rate (0.001) with decay, a single hidden layer with ten neurons and sigmoid activation was chosen. Table 5 compares GADGET SVM with Lasso, Ridge and Elastic Net regression methods while Table 6 compares it to Random Forests, Boosting and MultiLayer Perceptrons. It is observed that if the distributed algorithm is interrupted earlier than the convergence time of Lasso, Ridge or Elastic Net models, it produces comparable performance for Sido, Protein, Adult and Real-Sim datasets; however, it is likely to require more training time for Gisette, Quantum and CoverType datasets to reach comparable performance. A similar observation is recorded for experiments with Random Forest and Boosting owing primarily to sparse nature of the datasets discussed earlier. Multilayer Perceptrons, however, provide very different insight – if the problem is non-linear and the data set is very large (such as in Real-Sim) they out-perform most other techniques including SVMs, Random Forests and Boosting; however, they are computationally very expensive and it is often laborious to tune the large parameter space. In contrast, if the problem is linear, they are overkill and much better performance can be obtained with less computation with Random Forests, SVMs and Boosting.

Having established the utility of our consensus based SVM algorithm on several real-world datasets, we now turn to a specific application – that of predicting failures in a production line – a problem of interest to chocolate manufacturers. This case study is presented next.

6. Case Study on the Bosch Production Line

Bosch, one of the world's largest manufacturing companies uses advanced mechanical components to produce decadent, delicate chocolate soufflés. The parts of the assembly line are subject to close monitoring as they are required to meet high standards of quality and safety. Bosch records data at every step along its assembly lines, they have the ability to apply advanced analytics to improve these manufacturing processes. Each part in the assembly line is assigned a unique id and data about the part is collected. This data is streamed in through sensors which are then required to predict which parts are likely to fail. The goal, therefore, is to predict internal failures using thousands of measurements and tests made for each component along the assembly line. Figure 2 illustrates an architecture of the production line including a primary layer comprising of IoT devices, the protocols and gateway layers and the backend. Our interest lies primarily in the consensus setting in the backend.

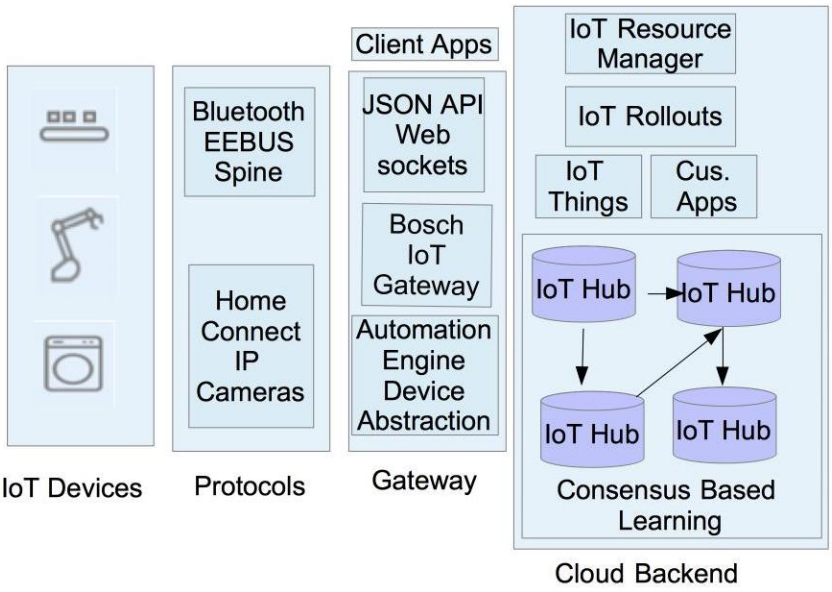

Figure 2 A Consensus Learning Framework on the IoT Hubs for Bosch's Production Line

We test a SVM algorithm for the consensus environment described above using data made available by a Kaggle competition (`https://www.kaggle.com/c/bosch-production-line-performance/overview`). Specifically, our consensus system used 500000 examples to train a model to predict failures and another 500000 to test it.

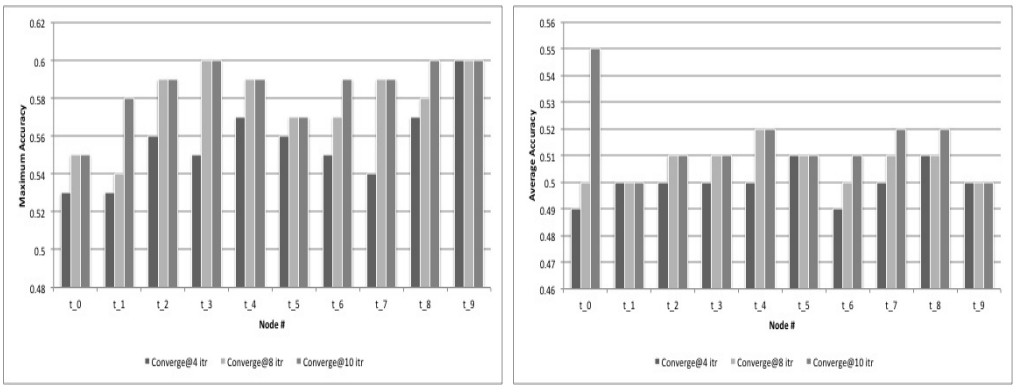

Figure 3 The left hand plot illustrates the maximum accuracy of classification obtained for the Bosch data on one IoT Hub. The right hand plot presents the average accuracy in the Hub. The three vertical lines corresponding to each node shows the results that would have been obtained for different termination criteria i.e. if convergence occurs when successive norm of weight vectors do not change for 4 iterations (Converge@4Itr), 8 iterations (Converge@8Itr) and 10 iterations (Converge@10Itr) respectively.

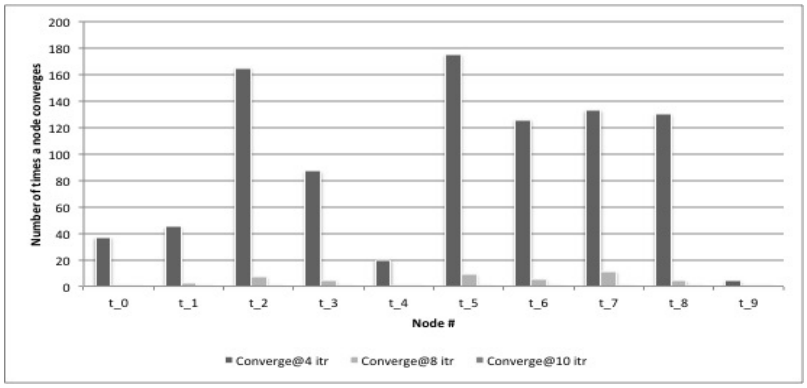

Figure 4 Empirical Analysis of the robustness of convergence – Converge@4 itr, Converge@8 itr and Converge@10 itr.

Each example has 969 numerical features. Features are named according to a convention that indicates the production line, the station on the line, and a feature number – for example, $L3\_S36\_F3939$ is a feature measured on line 3, station 36, and is feature number 3939. There are four lines with fifty one stations in all – the first, second and third lines have one, two and three stations respectively while line four has all the rest of the stations. The data set is highly imbalanced as the number of failures in the system is reasonably low. The training files have 0.6% examples with failures while the test files report 0.59% failures. Figure 5 shows the performance of the system in terms on accuracy of classification and decrease in primal objective value at one of the IoT hubs in the system. The sites are assumed to have converged if the difference in norm of the weight vector falls below $\epsilon = 0.001$ in ten successive iterations. However, this definition of convergence was

set solely from empirical data. We explored what would happen if sites were assumed to have converged after the difference in norm of the weight vector fell below $\epsilon = 0.001$ in four (Converge@4 itr), eight (Converge@8 itr) and ten (Converge@10 itr) iterations respectively. The performance of the system with regard to maximum and average accuracy obtained are reported in Figures 3. It was observed that some sites reach a maximum accuracy of 60% when convergence is observed at eight (Converge@8 itr) or ten (Converge@10 itr) successive iterations of low change in difference of norms of weigh vector. If a more relaxed definition of convergence is permissible for use in the application, say for example, four successive iterations (Converge@4 itr) of low change in difference of norms of weigh vector, the accuracy of classification hovers around the 55% mark. Next, we ran the GADGET SVM algorithm for a fixed number of iterations (10000) and observed the robustness of Converge@4 itr, Converge@8 itr and Converge@10 itr. This was made possible by keeping track of how many times a site "converged" within a fixed time frame (measured by the number of iterations). Figure 4 presents empirical results on the Bosch data. About 60% of sites in the network "converged" more than 80 times with Converge@4 itr thereby indicating that it is not uncommon for sites to reach convergence fast using this metric, but they are fundamentally unstable and can be easily disturbed from their steady state when new information comes along through the gossip protocol. On the other hand, when Converge@10 itr is used, sites tend to remain in their converged state for extended periods of time. Finally, we compare the performance of the consensus based algorithm to several state-of-the-art solvers including SVM^{Perf} and LIBLINEAR on the million examples. Our results indicate that GADGET SVM has an accuracy of 59%, SVM^{Perf} 41% and LIBLINEAR 47%. The consensus based algorithm has a higher accuracy in predicting the failure class than the state-of-the-art algorithms in terms of accuracy of classification. We surmise that the worse performance of the SVM^{Perf} and LIBLINEAR may be linked to the fact that the dataset is highly imbalanced since number of failures in the system is significantly less. Furthermore, the average behavior of GADGET SVM for Converge@10 itr is slightly better than a random classifier (51%). Future work will involve over or under sampling the dataset appropriately (Chawla et al. (2002)) and estimating the accuracy of prediction of the rare class.

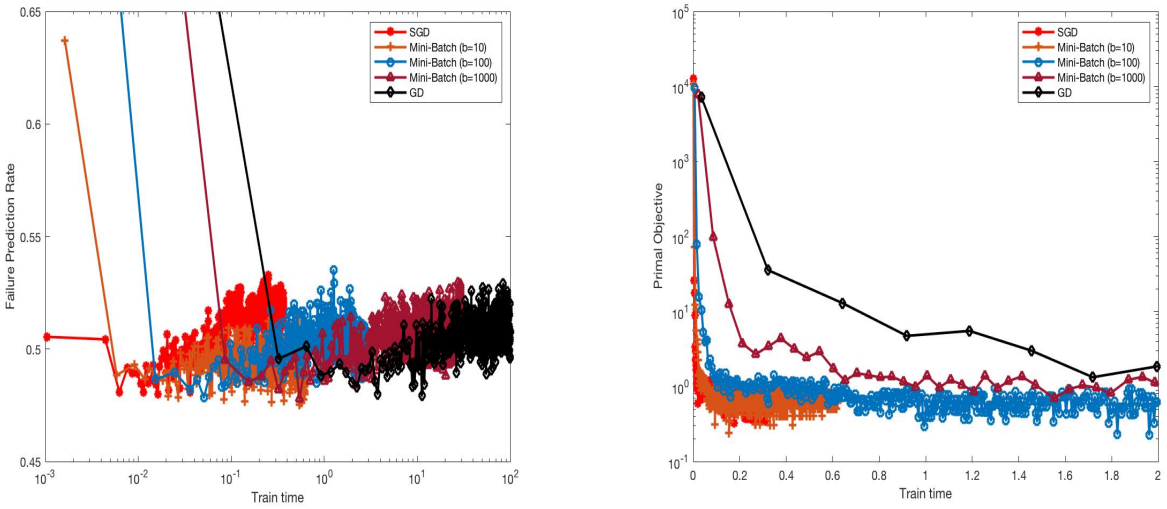

Figure 5 Accuracy of classification and Primal Objective for the Bosch data on one IoT Hub.

7. Conclusions

We presented a distributed consensus based algorithm for SVMs, for approximately minimizing the objective function of a linear SVM using the primal formulation. The algorithm uses a gossip-based protocol to communicate amongst distributed sites. We derived theoretical bounds to show that the algorithm is guaranteed to converge and presented empirical results on seven publicly available real-world data sets. Our results indicate that the performance of the distributed algorithm is comparable to state-of-the-art centralized counterparts (such as Pegasos) and pseudo distributed variants including SVM^{Perf}, LIBLINEAR, LIBSVM and an ADMM solver. It is also 1.5 times faster than state-of-the-art gossip based algorithms for distributed SVMs on all datasets studied in this paper. The consensus algorithm was then used to predict failures in mechanical components in a chocolate manufacturing process using a large data set of more than a million points. This case study showed that businesses can use the IoT device not just for collecting data but also for the purpose of prediction and learning using the consensus based framework. More generally, the quote from the McKinsey Global Institute Report stated earlier in the article which laments the use of IoT devices only as data collecting tools without being useful for predictive analytics in business enterprises can change with successful adoption of these new technologies.

There are several directions for future work including extension to multi-class variants of SVMs, testing the resilience to site failures, impact of the underlying network structure on

the convergence of the algorithm and development of distributed gossip-based algorithms for non-linear SVMs. We hope to address these and other related problems in subsequent papers.

References

Agarwal A, Duchi JC (2011) Distributed delayed stochastic optimization. *Advances in Neural Information Processing Systems*, 873–881.

Alistarh D, Grubic D, Li J, Tomioka R, Vojnovic M (2017) Qsgd: Communication-efficient sgd via gradient quantization and encoding. *Advances in Neural Information Processing Systems 30*, 1709–1720.

Bawa M, Garcia-Molina H, Gionis A, Motwani R (2003) Estimating aggregates on a peer-to-peer network. Technical Report 2003-24, Stanford InfoLab.

Bekkerman R, Bilenko M, Langford J (2011) *Scaling Up Machine Learning: Parallel and Distributed Approaches* (New York, NY, USA: Cambridge University Press).

Bertsekas DP, Tsitsiklis JN (1997) *Parallel and Distributed Computation: Numerical Methods.* (Athena Scientific).

Bliman PA, Ferrari-Trecate G (2008) Average consensus problems in networks of agents with delayed communications. *Automatica* 44(8):1985–1995.

Blondel VD, Hendrickx JM, Olshevsky A, Tsitsiklis JN (2005) Convergence in multi-agent coordination, consensus and flocking. *Proc. of IEEE CDC* 2996–3000.

Blot M, Picard D, Thome N, Cord M (2019) Distributed optimization for deep learning with gossip exchange. *Neurocomputing* 330:287–296.

Boddy M, Dean TL (1994) Deliberation scheduling for problem solving in time-constrained environments. *Artif. Intell.* 67(2):245–285.

Bottou L, Bousquet O (2011) The tradeoffs of large scale learning. Sra S, Nowozin S, Wright SJ, eds., *Optimization for Machine Learning*, 351–368 (MIT Press).

Bottou L, Chapelle O, DeCoste D, Weston J (2007) *Large-Scale Kernel Machines (Neural Information Processing)* (The MIT Press).

Boyd S, Ghosh A, Prabhakar B, Shah D (2005) Gossip algorithms: Design, analysis and applications. *Proc. of IEEE INFOCOM* 3:1653–1664.

Boyd S, Ghosh A, Prabhakar B, Shah D (2006) Randomized gossip algorithms. *IEEE/ACM Trans. Netw.* 14(SI):2508–2530.

Boyd S, Parikh N, Chu E, Peleato B, Eckstein J (2011) Distributed optimization and statistical learning via the alternating direction method of multipliers. *Found. Trends Mach. Learn.* 3(1):1–122.

Breiman L (2001) Random forests. *Mach. Learn.* 45(1):5–32.

Caragea C, Caragea D, Honavar V (2005) Learning support vector machines from distributed data sources. *AAAI*, 1602–1603.

Carli R, Fagnani F, Speranzon A, Zampieri S (2007) Average consensus on networks with transmission noise or quantization. *Proc. of European Control Conference* .

Cecchini M, Aytug H, Koehler GJ, Pathak P (2010) Detecting management fraud in public companies. *Management Science* 56(7):1146–1160.

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

Chang CC, Lin CJ (2011b) LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology* 2:27:1–27:27.

Chang KW, Hsieh CJ, Lin C (2008) Coordinate descent method for large-scale l2-loss linear support vector machines. *J. Mach. Learn. Res.* 9:1369–1398.

Chapelle O (2007) Training a support vector machine in the primal. *Neural Comput.* 19(5):1155–1178.

Chawla NV, Bowyer KW, Hall LO, Kegelmeyer WP (2002) Smote: Synthetic minority over-sampling technique. *J. Artif. Int. Res.* 16(1):321–357.

Daily J, Vishnu A, Siegel C, Warfel T, Amatya V (2018) Gossipgrad: Scalable deep learning using gossip communication based asynchronous gradient descent. *CoRR* abs/1803.05880.

Datta S, Bhaduri K, Giannella C, Wolff R, Kargupta H (2006) Distributed data mining in peer-to-peer networks. *IEEE Internet Computing special issue on Distributed Data Mining.* 10:18–26.

Demers A, Greene D, Hauser C, Irish W, Larson J, Shenker S, Sturgis H, Swinehart D, Terry D (1987) Epidemic algorithms for replicated database maintenance. *n Proc. of the ACM Symposium on Principles of Distributed Computing* 1–12.

Dimakis A, Sarwate AD, Wainwright MJ (2006) Geographic gossip: efficient aggregation for sensor networks. *International Conference on Information Processing in Sensor Networks*, 69–76.

Dimakis AG, Kar S, Moura JMF, Rabbat MG, Scaglione A (2010) Gossip algorithms for distributed signal processing. *Proceedings of the IEEE* 98(11):1847–1864.

Duchi J, Agarwal A, Wainwright M (2012) Dual Averaging for Distributed Optimization: Convergence Analysis and Network Scaling. *IEEE Transactions on Automatic Control* 57(3):592–606.

Duchi J, Singer Y (2009) Efficient online and batch learning using forward backward splitting. *J. Mach. Learn. Res.* 10:2899–2934.

Dutta H, Srinivasan A (2018) Consensus-based modeling using distributed feature construction with ILP. *Machine Learning* 107(5):825–858.

Fadda E, Gobbarto L, Perboli G, Rosano M, R T (2018) Waste collection in urban areas: A case study. *INFORMS Journal of Applied Analytics* 48(4).

Fan R, Chang K, Hsieh C, Wang X, Lin C (2008) LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research* 9:1871–1874.

Flouri K, Beferull-Lozano B, Tsakalides P (2006) Training a support vector machine based classifier in distributed sensor networks. *European Signal Processing Conference*.

Flouri K, Beferull-Lozano B, Tsakalides P (2009) Optimal gossip algorithm for distributed consensus svm training in wireless sensor networks. *International Conference on Digital Signal Processing*, 1–6.

Forero PA, Cano A, Giannakis GB (2010) Consensus-based distributed support vector machines. *J. Mach. Learn. Res.* 99:1663–1707.

Freund Y, Schapire RE (1998) Large margin classification using the perceptron algorithm. *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*, 209–217, COLT' 98.

Friedman J, Hastie T, Tibshirani R (2010) Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software* 33(1):1–22.

Gong L, Jwo W, Tang K (1997) Using on-line sensors in statistical process control. *Management Sc.* 43(7):1017–1028.

Graf HP, Cosatto E, Bottou L, Durdanovic I, Vapnik V (2005) Parallel support vector machines: The cascade svm. *In Advances in Neural Information Processing Systems*, 521–528 (MIT Press).

Guyon I, Gunn S, Hur AB, Dror G (2004) Result analysis of the nips 2003 feature selection challenge. *Proceedings of the 17th International Conference on Neural Information Processing Systems*, 545–552, NIPS'04.

Hazan T, Man A, Shashua A (2008) A parallel decomposition solver for svm: Distributed dual ascend using fenchel duality. *IEEE Conference on Computer Vision and Pattern Recognition*, 1–8.

Hensel C, Dutta H (2009) Gadget svm: A gossip-based sub-gradient svm solver. *In Proc. of International Conference on Machine Learning (ICML), Numerical Mathematics in Machine Learning Workshop* .

Hong B, Zhang W, Liu W, Ye J, Cai D, He X, Wang J (2019) Scaling up sparse support vector machines by simultaneous feature and sample reduction. *Journal of Machine Learning Research* 20(121):1–39.

Horvitz EJ (1987) Reasoning about beliefs and actions under computational resource constraints. *Proceedings of the Third Conference on Uncertainty in Artificial Intelligence*, 429–447, UAI'87.

Jadbabaie A, Lin J, Morse AS (2003) Coordination of groups of mobile autonomous agents using nearest neighbor rules. *IEEE Transactions on Automatic Control* 48(6):988–1001.

Joachims T (1998) Text categorization with support vector machines: Learning with many relevant features. *Proc. of European Conference on Machine Learning (ECML)*, 137–142 (Berlin).

Joachims T (1999) Making large-scale support vector machine learning practical. *Advances in Kernel Methods*, 169–184 (Cambridge, MA, USA: MIT Press).

Joachims T (2006) Training linear svms in linear time. *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, 217–226, KDD '06.

Joachims T, Yu CN (2009) Sparse kernel svms via cutting-plane training. *Machine Learning* 76(2-3):179–193.

Kargupta H, Chan Pe (2000) *"Advances in Distributed and Parallel Knowledge Discovery"* (Menlo Park, CA: AAAI press).

Kargupta H, Sarkar K, Gilligan M (2010) Minefleet: An overview of a widely adopted distributed vehicle performance data mining system. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 37–46.

Keerthi SS, Chapelle O, DeCoste D (2006) Building support vector machines with reduced classifier complexity. *Journal of Machine Learning Research* 7:1493–1515.

Keerthi SS, DeCoste D (2005) A modified finite newton method for fast solution of large scale linear svms. *J. Mach. Learn. Res.* 6:341–361.

Kempe D, Dobra A, Gehrke J (2003a) Gossip-based computation of aggregate information. *In Proc. of the IEEE Symposium on Foundations of Computer Science* 482–491.

Kempe D, Dobra A, Gehrke J (2003b) Gossip-based computation of aggregate information. *Foundations of Computer Science, 2003. Proceedings. 44th Annual IEEE Symposium on*, 482 – 491.

Kempe D, McSherry F (2008) A decentralized algorithm for spectral analysis. *J. Comput. Syst. Sci.* 74(1):70–83.

Kim W, Stankovi? MS, Johansson KH, Kim HJ (2015) A distributed support vector machine learning over wireless sensor networks. *IEEE Transactions on Cybernetics* 45(11):2599–2611.

Kimeldorf G, Wahba G (1971) Some results on tchebycheffian spline functions. *Journal of Mathematical Analysis and Applications* 33(1):82 – 95.

Koloskova A, Lin T, Stich SU, Jaggi M (2019a) Decentralized deep learning with arbitrary communication compression. *CoRR* abs/1907.09356.

Koloskova A, Stich S, Jaggi M (2019b) Decentralized stochastic optimization and gossip algorithms with compressed communication. *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, 3478–3487 (Long Beach, California, USA).

Konecny J, Richatrik P (2018) Randomized distributed mean estimation: Accuracy vs. communication. *Frontiers in Applied Mathematics and Statistics* 4:62.

Lee S, Stolpe M, Morik K (2012) Separable approximate optimization of support vector machines for distributed sensing. *Machine Learning and Knowledge Discovery in Databases*, 387–402.

Lesser V, Pavlin J, Durfee E (1988) Approximate processing in real-time problem solving. *AI Magazine* 9(1):49–61.

Li M, Zhang T, Chen Y, Smola AJ (2014) Efficient mini-batch training for stochastic optimization. *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 661–670, KDD '14.

Li X, Wang H, Gu B, Ling CX (2015) Data sparseness in linear svm. *Proceedings of the 24th International Conference on Artificial Intelligence (IJCAI)*, 3628–3634.

Li Z, Fang X, Bai X, Sheng ORL (2017) Utility-based link recommendation for online social networks. *Management Science* 63(6):1938–1952.

Lin Y, Han S, Mao H, Wang Y, Dally WJ (2017) Deep gradient compression: Reducing the communication bandwidth for distributed training. URL `ICLR`.

Lu Y, Roychowdhury V, Vandenberghe L (2008) Distributed parallel support vector machines in strongly connected networks. *Neural Networks, IEEE Transactions on* 19(7):1167–1178.

Lynch NA (1996) *Distributed Algorithms* (San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.).

Mangasarian OL (2002) A finite newton method for classification. *Optimization Methods and Software* 17(5):913–929.

Manyika J, Chui M, Bisson P, Woetzel J, Dobbs R, Bughin J, Aharon D (2015) The internet of things: Mapping the value beyond the hype. Technical report, McKinsey Global Institute.

Menon AK (2009) Large-scale support vector machines: Algorithms and theory. Technical report, University of California, San Diego.

Mouaddib Ai, Zilberstein S (1995) Knowledge-based anytime computation. *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI)*, 775–781 (Montreal, Canada), URL `http://rbr.cs.umass.edu/shlomo/papers/MZijcai95.pdf`.

Narayanan H (2007) Geographic gossip on geometric random graphs via affine combinations. *Proc. of the ACM Symposium on Principles of Distributed Computing*, 388–389, PODC '07 (ACM).

Nedic A, Ozdaglar A (2009) *Cooperative Distributed Multi-Agent Optimization*, chapter 10, 340–384 (Cambridge University Press).

Nedić A, Ozdaglar A (2009) Distributed subgradient methods for multi-agent optimization. *IEEE Transactions on Automatic Control* 54(1):48–61.

Nedić A, Ozdaglar A, Parrilo PA (2010) Constrained consensus and optimization in multi-agent networks. *Automatic Control, IEEE Transactions on* 55(4):922–938.

Ormandi R, Hegedus I, Jelasity M (2013) Gossip learning with linear models on fully distributed data. *Concurrency and Computation: Practice and Experience* 25(4):556–571.

Osuna E, Freund R, Girosi F (1997a) An improved training algorithm for support vector machines. *Proc. of IEEE NNSP*, 130–136 (Amelia Island, FL).

Osuna E, Freund R, Girosi F (1997b) Training support vector machines: an application to face detection. *Computer Vision and Pattern Recognition*, 130–136.

Provost F, Fawcett T (2013) *Data Science for Business* (New York, NY, USA: OReilly).

Rajaraman A, Ullman J (2011) *Mining of Massive Datasets* (New York, NY, USA: Cambridge University Press).

Ram SS, Nedic A, Veeravalli V (2010) Distributed stochastic subgradient projection algorithms for convex optimization. *Journal of Opt. Theory and Applications.* 147(3):516–545.

Sai J, Wang B, Wu B (2016) Bppgd: Budgeted parallel primal gradient descent kernel svm on spark. *IEEE International Conference on Data Science in Cyberspace (DSC)*, 74–79.

Scardapane S, Fierimonte R, Di Lorenzo P, Panella M, Uncini A (2016) Distributed semi-supervised support vector machines. *Neural Netw.* 80(C):43?52.

Schapire RE, Freund Y (2012) *Boosting: Foundations and Algorithms* (The MIT Press).

Shah D (2009) *Gossip algorithms. Foundations and Trends in Networking* (Now Publishers Inc).

Shalev-Shwartz S, Singer Y, Srebro N (2007) Pegasos: Primal estimated sub-gradient solver for svm. *Proceedings of the International Conference on Machine learning*, 807–814, ICML '07 (New York, NY, USA: ACM).

Sharma N, Verlekar P, Ashary R, Khaled M, Zhiquan S (2017) Regularization and feature selection for large dimensional data. *CoRR* abs/1712.01975.

Simester D, Timoshenko A, Zoumpoulis SI (2019) Targeting prospective customers: Robustness of machine-learning methods to typical data challenges. *Management Science, Forthcoming* 0(0):0.

Simon HA (1982) *Models of Bounded Rationality, Volume 2.* (Cambridge, Mass.: MIT Press).

Singer Y, Srebro N (2007) Pegasos: Primal estimated sub-gradient solver for svm. *ICML*, 807–814.

Staats BR, Dai H, Hofmann D, Milkman KL (2017) Motivating process compliance through individual electronic monitoring: An empirical examination of hand hygiene in healthcare. *Management Sc.* 63(5):1563–1585.

Steinwart I (2003) Sparseness of support vector machines. *J. Mach. Learn. Res.* 4:1071–1105.

Stolpe M, Bhaduri K, Das K (2016) *Distributed Support Vector Machines: An Overview*, 109–138 (Springer International Publishing).

Sundhar RS, Nedic A, Veeravalli VV (2010) Distributed stochastic subgradient projection algorithms for convex optimization. *J. Optim. Theory Appl.* 147(3):516–545.

SVM-SGD (Website) http://leon.bottou.org/projects/sgd.

Syed NA, Liu H, Sung KK (1999) Handling concept drifts in incremental learning with support vector machines. *Proc. of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 317–321 (New York, NY, USA: ACM).

Takáč M, Bijral A, Richtárik P, Srebro N (2013) Mini-batch primal and dual methods for svms. *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, III–1022–III–1030, ICML'13.

Tanenbaum AS, Steen M (2006) *Distributed Systems: Principles and Paradigms (2nd Edition)* (Upper Saddle River, NJ, USA: Prentice-Hall, Inc.).

Tang H, Gan S, Zhang C, Zhang T, Liu J (2018) Communication compression for decentralized training. *Advances in Neural Information Processing Systems 31*, 7652–7662.

Tortonesi M, Stefanelli C, Benvegnu E, Ford K, Suri N, Linderman M (2012) Multiple-uav coordination and communications in tactical edge networks. *Communications Magazine, IEEE* 50(10):48–55.

Tsitsiklis J (1984) *Problems in Decentralized Decision Making and Computation.* Ph.D. thesis, Department of EECS, MIT.

Tsitsiklis JN, Bertsekas DP, Athans M (1986) Distributed asynchronous deterministic and stochastic gradient optimization algorithms. *In Proceedings. of IEEE Transactions on Automatic Control*, volume 31.

Wang D, Zheng J, Zhou Y, Li J (2010) A scalable support vector machine for distributed classification in ad hoc sensor networks. *Neurocomput.* 74(1?3):394?400.

Wangni J, Wang J, Liu J, Zhang T (2018) Gradient sparsification for communication-efficient distributed optimization. *Advances in Neural Information Processing Systems 31*, 1299–1309.

Wardell DG, Mosokowitz H, Plante RD (1992) Control charts in the presence of data correlation. *Management Sc.* 38:1084–1105.

Wen W, Xu C, Yan F, Wu C, Wang Y, Chen Y, Li H (2017) Terngrad: Ternary gradients to reduce communication in distributed deep learning. *Advances in Neural Information Processing Systems 30*, 1509–1519.

Yang Z, Bajwa WU (2016) Rd-svm: A resilient distributed support vector machine. *International Conference on Acoustics, Speech and Signal Processing.*

Yu HF, Hsieh CJ, Chang KW, Lin CJ (2012) Large linear classification when data cannot fit in memory. *ACM Trans. Knowl. Discov. Data* 5(4).

Zadorojniy A, Wasserkrug S, Lipets V (2017) Ibm cognitive technology helps aqualia to reduce costs and save resources in wastewater treatment. *INFORMS Journal on Applied Analytics* 47(5).

Zaki M, Ho CT (2000) *Large Scale Parallel Data Mining*, volume 1759 (Lecture Notes in Computer Science: State of the Art Survey).

Zhang DJ, Dai H, Dong L, Wu Q, Guo L, Liu X (2019) The value of pop-up stores on retailing platforms: Evidence from a field experiment with alibaba. *Management Sc.* 65(11):5143–5151.

Zhang H, Li J, Kara K, Alistarh D, Liu J, Zhang G (2017) ZipML: Training linear models with end-to-end low precision, and a little bit of deep learning. *Proceedings of the International Conference on Machine Learning*, volume 70, 4035–4043.

Zhang J, Jin R, Yang Y, Hauptmann A (2003) Modified logistic regression: An approximation to svm and its applications in large-scale text categorization. *Proceedings of the International Conference on Machine Learning*, volume 2, 888–895.

Zhang T (2004) Solving large scale linear prediction problems using stochastic gradient descent algorithms. *Proceedings of the International Conference on Machine learning*, 116– (New York, NY, USA: ACM).

Zilberstein S (1993) *Operational Rationality through Compilation of Anytime Algorithms*. Ph.D. thesis, Computer Science Division, University of California Berkeley.

Zilberstein S (1996) Using anytime algorithms in intelligent systems. *AI Magazine* 17(3):73–83.

Appendix

A. Analytical Results

Before analyzing the GADGET algorithm, the notion of strong convexity and a sub-differential needs to be introduced. These are essential tools for obtaining bounds on the convergence rate of the algorithm in addition to understanding the convergence of the projected sub-gradient method and the network error. We will prove a lemma about the SVM loss function and then apply the relative error bounds of Push-Sum to obtain the true convergence rate.

Informally, a strong convex function is a function who's gradient is always changing, or equivalently the Hessian is always positive definite. Unfortunately, the SVM objective function is not differentiable and our analysis must rely on the following, more general, definition of strong convexity using sub-differentials. The following is the formal definition of strong convexity.

DEFINITION 1. A function $f : \Re^d \to \Re$ is λ-strongly convex if $\forall x, y \in \Re^d$ and $\forall g \in \partial f(x)$,
$f(y) \geq f(x) + \langle g, y - x\rangle + \frac{\lambda}{2}||x - y||^2$.

DEFINITION 2. The *sub-differential* of $f(x)$, denoted $\partial f(x)$, is the set of all tangent lines which can be drawn under $f(x)$. Moreover, each vector $v \in \partial f(x)$ is called a *sub-gradient* of $f(x)$.

Lemma 1 *If $L_g(w)$ is the global average loss of the vector w, then we have*
$||L_g(w_1) - L_g(w_2)|| \leq R||w_1 - w_2||_2$.

Proof: The global hinge loss is equal to $\frac{1}{n}\sum_{i=1}^{n} \max\{0, 1 - y_i\langle x_i, w\rangle\}$; Using triangle inequality we have $\|L_g(w_1) - L_g(w_2)\|_2 \leq \frac{1}{n}\sum_{i=1}^{n} \|\max\{0, 1 - y_i\langle x_i, w_1\rangle\} - \max\{0, 1 - y_i\langle x_i, w_2\rangle\}\|_2$. Notice that if one of the max functions is zero (assume without loss of generality $1 - y_i\langle x_i, w_2\rangle \leq 0$), the difference of loss equation (for a particular feature) is less or equal to $\|(1 - y_i\langle x_i, w_1\rangle) - (1 - y_i\langle x_i, w_2\rangle)\|_2$. Further observing that if both losses are non-zero, we also arrive at this same equation, the difference in global loss simplifies to $\frac{1}{n}\sum_{i=1}^{n} \|y_i\langle x_i, w_2\rangle - y_i\langle x_i, w_1\rangle\|_2 \leq \frac{1}{n}\sum_{i=1}^{n} \|\langle x_i, w_2 - w_1\rangle\|_2$. Finally, using the Cauchy-Schwartz inequality, we obtain our desired result. □

Lemma 2 *As presented by Kempe and McSherry (2008), let $\mathbf{v}_{t,i}$ be the $k \times 1$ vector held by the site i after the t^{th} iteration of Push-Vector, $w_{t,i}$ its weight at that time, and $\mathbf{v}$ the correct vector. Define $M = \sum_i |\mathbf{v}_{0,i}|$ to be the vector whose (r,1) entry is the sum of the absolute values of the initial vector $\mathbf{v}_{0,i}$ at all sites i. Then, for any ϵ, the approximation error is $||\frac{\mathbf{v}_{(t,i)}}{w_{(t,i)}} - \mathbf{v}||_2 \leq \epsilon||M||_2$, after t=$O(\tau_{mix} \cdot \log\frac{1}{\epsilon})$*

Proof: This proof appears in Lemma 2 of Kempe and McSherry (2008). □

Theorem 2 *Assuming that $||\mathbf{w}^{(t)} - \hat{\mathbf{w}}_i^{(t)}|| < \epsilon$, we have*

$$||\mathbf{w}^{(t+\frac{1}{2})} - \hat{\mathbf{w}}_i^{(t+\frac{1}{2})}|| \leq (1 - \lambda\alpha^{(t)})(\epsilon + \frac{N\epsilon_1}{\lambda}) + \alpha^{(t)}(\frac{R\epsilon}{N} + \epsilon_2) \tag{7}$$

where ϵ_2 is the relative error in approximating average global loss through Push-Sum of loss on each site and ϵ_1 is the relative error in approximating the weighted sum of the weight (support) vector estimated at each site i.e. $\frac{\sum_i n_i \hat{\mathbf{w}}_i^{(t)}}{N}$

Proof:

$$\begin{aligned}||\mathbf{w}^{(t+\frac{1}{2})} - \hat{\mathbf{w}}_i^{(t+\frac{1}{2})}|| &= ||(1-\lambda\alpha^{(t)})\mathbf{w}^{(t)} + \alpha^{(t)}\frac{L^{(t)}}{N} - (1-\lambda\alpha^{(t)})PS(n_i\hat{\mathbf{w}}_i^{(t)}, B) - \alpha^{(t)}PS(\hat{L}_i^{(t)}, B)|| \\ &\leq (1-\lambda\alpha^{(t)})(||\mathbf{w}^{(t)} - \frac{\sum_i n_i\hat{\mathbf{w}}_i^{(t)}}{N}|| + ||\frac{\sum_i n_i\hat{\mathbf{w}}_i^{(t)}}{N} - PS(n_i\hat{\mathbf{w}}_i^{(t)}, B)||) \\ &+ \alpha^{(t)}(||\frac{L^{(t)}}{N} - \frac{\sum_i \hat{L}_i^{(t)}}{N}|| + ||\frac{\sum_i \hat{L}_i^{(t)}}{N} - PS(\hat{L}_i^{(t)}, B)||) \\ &\leq (1-\lambda\alpha^{(t)})(\epsilon + \frac{N\epsilon_1}{\lambda}) + \alpha^{(t)}(\frac{R\epsilon}{N} + \epsilon_2)\end{aligned} \tag{8}$$

where we have used Lemma 2, Lemma 3, and the following:

$||\mathbf{w}^{(t)} - \frac{\sum_i n_i\hat{\mathbf{w}}_i^{(t)}}{N}|| = ||\sum_i \frac{n_i\mathbf{w}^{(t)}}{N} - \frac{\sum_i n_i\hat{\mathbf{w}}_i^{(t)}}{N}|| \leq \sum_i \frac{n_i\epsilon}{N} = \epsilon$

$$||\frac{\sum_i n_i\hat{\mathbf{w}}_i^{(t)}}{N} - PS(n_i\hat{\mathbf{w}}_i^{(t)}, B)|| \leq \epsilon_1 \frac{1}{N^2}\sum_{k=1}^{K}\left(\sum_{i=1}^{N}|n_i\hat{\mathbf{w}}_i^{(t+\frac{1}{2})}(k)|\right)^2$$

$$\leq \frac{\sum_i n_i^2}{N^2}\sum_{i=1}^{N}\sum_{k=1}^{K}(\hat{\mathbf{w}}_i^{(t+\frac{1}{2})}(k))^2 \leq \frac{\sum_i n_i^2}{N^2}\frac{N}{\lambda} \leq \frac{N}{\lambda}$$

$$||\frac{\sum_i \hat{L}_i^{(t)}}{N} - PS(\hat{L}_i^{(t)}, B)|| \leq \frac{\epsilon_2}{N}\sqrt{\sum_{i=1}^{N}(\hat{L}_i^{(t)})^2} \leq \epsilon_2$$

□

Lemma 3 *As stated in Shalev-Shwartz et al. (2007), let $f_1, \cdots, f_T$ be a sequence of λ-strongly convex functions. Let B be a closed convex set and define $\prod_B(w) = arg\ min_{\acute{w}\in B}||w - \acute{w}||$. Let $w_1, \cdots, w_{T+1}$ be a sequence of vectors such that $w_1 \in B$ and for $t \geq 1$, $w_{t+1} = \prod_B(w_t - \alpha_t\nabla_t)$, where ∇_t belongs to the sub-gradient set of f_t at w_t and $\alpha_t = \frac{1}{\lambda t}$. Assume that for all t, $||\nabla_t|| < c$. Then it can be shown that,*

$$\langle \mathbf{w}^{(t)} - \mathbf{u}, \nabla_t\rangle \leq \frac{||\mathbf{w}^{(t)} - \mathbf{u}||^2 - ||\mathbf{w}^{(t+1)} - \mathbf{u}||^2}{2\alpha^{(t)}} + \frac{\alpha^{(t)}}{2}c^2$$

Proof: The proof appears in Lemma 1 of Shalev-Shwartz et al. (2007). We restate it here for the sake of completeness. Let $\acute{w}_t$ denote $w_t - \alpha_t\nabla_t$. Since w_{t+1} is the projection of $\acute{w}_t$ onto B, and $u \in B$, we have that $||\acute{w}_t - u||^2 - ||\acute{w_{t+1}} - u||^2 \geq ||\acute{w}_t - u||^2 - ||\acute{w}_t - u||^2 = 2\alpha_t\langle \mathbf{w}^{(t)} - \mathbf{u}, \nabla_t\rangle - \alpha_t^2||\nabla_t||^2$. Rearranging the previous equation, and using the fact that $||\nabla_t|| < c$, we have,

$$\langle \mathbf{w}^{(t)} - \mathbf{u}, \nabla_t\rangle \leq \frac{||\mathbf{w}^{(t)} - \mathbf{u}||^2 - ||\mathbf{w}^{(t+1)} - \mathbf{u}||^2}{2\alpha^{(t)}} + \frac{\alpha^{(t)}}{2}c^2$$

□

Theorem 3 *Assume that the conditions in Theorem 2 hold. Then it can be shown that*

$$f(\bar{\mathbf{w}}_i) - f(\mathbf{w}^\star) \leq \frac{2c}{\sqrt{\lambda}} + \frac{c^2 log(T)}{2T\lambda} + \frac{2}{\sqrt{\lambda}}\left(\frac{\gamma R}{\sqrt{\lambda}} + \gamma R\right)$$

Proof: From Algorithm 2, we have,

$$\tilde{\mathbf{w}}_i^{(t+\frac{1}{2})} = (1-\lambda\alpha^{(t)})n_i\hat{\mathbf{w}}_i^{(t)} + \alpha^{(t)}\hat{L}_i^{(t)} \tag{9}$$

Summing over iterations $t = 1, 2, \ldots, T$ we define,

$$\bar{\mathbf{w}}^{(t+\frac{1}{2})} := \sum_{i=1}^{m} \frac{\hat{\mathbf{w}}_i^{(t+\frac{1}{2})}}{N} = \tilde{\bar{\mathbf{w}}}^{(t)} - \alpha^{(t)}\left(\lambda\tilde{\bar{\mathbf{w}}}^{(t)} - \hat{L}_g^{(t)}\right) \tag{10}$$

where $\tilde{\bar{\mathbf{w}}}^{(t)} = \frac{\sum_{i=1}^{m} n_i\hat{\mathbf{w}}_i^{(t)}}{N}$, $\hat{L}_g^{(t)} = \frac{1}{N}\sum_{i=1}^{m}\hat{L}_i^{(t)}$. Note that $||\mathbf{w}^\star|| \leq \frac{1}{\sqrt{\lambda}}$ and $||\hat{\mathbf{w}}_i^{(t)}|| \leq \frac{1}{\sqrt{\lambda}}$ from the algorithm. Also, we have

$$||\tilde{\bar{\mathbf{w}}}^{(t)}|| = ||\frac{\sum_{i=1}^{m} n_i\hat{\mathbf{w}}_i^{(t)}}{N}|| \leq \sum_{i=1}^{m}\frac{n_i}{N}||\hat{\mathbf{w}}_i^{(t)}|| \leq \sum_{i=1}^{m}\frac{n_i}{\sqrt{\lambda}N} = \frac{1}{\sqrt{\lambda}} \tag{11}$$

Assuming that c is the maximum sub-gradient, and using the fact that increment of a continuous function over an interval is always less than its maximum subgradient times the length of the interval, we have

$$f(\hat{\mathbf{w}}_i^{(t)}) - f(\tilde{\bar{\mathbf{w}}}^{(t)}) \leq c||\hat{\mathbf{w}}_i^{(t)} - \tilde{\bar{\mathbf{w}}}^{(t)}||_2 \tag{12}$$

where f represents Eq.(1). Using λ-strong convexity of f from definition 1 with $x = \tilde{\bar{\mathbf{w}}}^{(t)}, y = \mathbf{w}^\star$, and $g = \lambda\tilde{\bar{\mathbf{w}}}^{(t)} - L_g(\tilde{\bar{\mathbf{w}}}^{(t)})$ we have

$$f(\tilde{\bar{\mathbf{w}}}^{(t)}) - f(\mathbf{w}^\star) + \frac{\lambda}{2}||\tilde{\bar{\mathbf{w}}}^{(t)} - \mathbf{w}^\star||^2 \leq \langle\lambda\tilde{\bar{\mathbf{w}}}^{(t)} - L_g(\tilde{\bar{\mathbf{w}}}^{(t)}), \tilde{\bar{\mathbf{w}}}^{(t)} - \mathbf{w}^\star\rangle \tag{13}$$

Summing over t in Eq. 12 and subtracting $f(\mathbf{w}^\star)$ on both sides we have

$$\sum_{t=1}^{T} f(\hat{\mathbf{w}}_i^{(t)}) - f(\mathbf{w}^\star) \leq c\sum_{t=1}^{T}||\hat{\mathbf{w}}_i^{(t)} - \tilde{\bar{\mathbf{w}}}^{(t)}|| + \sum_{t=1}^{T} f(\tilde{\bar{\mathbf{w}}}^{(t)}) - f(\mathbf{w}^\star) \tag{14}$$

Using Eq.13 to replace $f(\tilde{\bar{\mathbf{w}}}^{(t)}) - f(\mathbf{w}^\star)$ in Eq.14 gives

$$\begin{aligned}\sum_{t=1}^{T} f(\hat{\mathbf{w}}_i^{(t)}) - f(\mathbf{w}^\star) \leq c\sum_{t=1}^{T}||\hat{\mathbf{w}}_i^{(t)} - \tilde{\bar{\mathbf{w}}}^{(t)}|| + \sum_{t=1}^{T}\langle\lambda\tilde{\bar{\mathbf{w}}}^{(t)} - L_g(\tilde{\bar{\mathbf{w}}}^{(t)}), \tilde{\bar{\mathbf{w}}}^{(t)} - \mathbf{w}^\star\rangle \\ - \sum_{t=1}^{T}\frac{\lambda}{2}||\tilde{\bar{\mathbf{w}}}^{(t)} - \mathbf{w}^\star||^2\end{aligned} \tag{15}$$

Now using Lemma 3 with $\mathbf{w}^{(t)} = \tilde{\bar{\mathbf{w}}}^{(t)}, \mathbf{u} = \mathbf{w}^\star, \nabla_t = \lambda\tilde{\bar{\mathbf{w}}}^{(t)} - \hat{L}_g^{(t)}$, we have

$$\langle\lambda\tilde{\bar{\mathbf{w}}}^{(t)} - \hat{L}_g^{(t)}, \tilde{\bar{\mathbf{w}}}^{(t)} - \mathbf{w}^\star\rangle \leq \frac{||\tilde{\bar{\mathbf{w}}}^{(t)} - \mathbf{w}^\star||^2 - ||\tilde{\bar{\mathbf{w}}}^{(t+1)} - \mathbf{w}^\star||^2}{2\alpha^{(t)}} + \frac{\alpha^{(t)}}{2}c^2 \tag{16}$$

Now, using the fact that $\langle a-b,c\rangle = \langle (a-d)-(b-d),c\rangle = \langle a-d,c\rangle - \langle b-d,c\rangle$, we have

$$\langle \lambda\tilde{\tilde{\mathbf{w}}}^{(t)} - L_g(\tilde{\tilde{\mathbf{w}}}^{(t)}), \tilde{\tilde{\mathbf{w}}}^{(t)} - \mathbf{w}^\star\rangle = \langle \lambda\tilde{\tilde{\mathbf{w}}}^{(t)} - \hat{L}_g^{(t)}, \tilde{\tilde{\mathbf{w}}}^{(t)} - \mathbf{w}^\star\rangle + \langle \hat{L}_g^{(t)} - L_g(\tilde{\tilde{\mathbf{w}}}^{(t)}), \tilde{\tilde{\mathbf{w}}}^{(t)} - \mathbf{w}^\star\rangle \tag{17}$$

Now, $||\tilde{\tilde{\mathbf{w}}}^{(t)} - \mathbf{w}^\star|| \leq ||\tilde{\tilde{\mathbf{w}}}^{(t)}|| + ||\mathbf{w}^\star|| \leq \frac{2}{\sqrt{\lambda}}$. Notice that $||L_g(\tilde{\tilde{\mathbf{w}}}^{(t)}) - \hat{L}_g^{(t)}|| = ||L_g(\tilde{\tilde{\mathbf{w}}}^{(t)}) - \frac{1}{N}\sum_{i=1}^m \hat{L}_i^{(t)}||$. Because each global loss approximation is within a γ-radius ball of the global loss of some w vector which in-turn is within a γ-radius ball of $\tilde{\tilde{\mathbf{w}}}^{(t)}$, this in turn is less than or equal to $||L_g(\tilde{\tilde{\mathbf{w}}}^{(t)}) - (1-\gamma)L_g((1+\gamma)\tilde{\tilde{\mathbf{w}}}^{(t)})||$. Now, using lemma 2 we have the last term $\leq ||L_g(\tilde{\tilde{\mathbf{w}}}^{(t)}) - L_g((1+\gamma)\tilde{\tilde{\mathbf{w}}}^{(t)})|| + \gamma R$. Again applying lemma 2 to the first term, we get it $\leq \frac{\gamma R}{\sqrt{\lambda}} + \gamma R$. Using this bound and Cauchy-Schwartz inequality, we have $\langle \hat{L}_g^{(t)} - L_g(\tilde{\tilde{\mathbf{w}}}^{(t)}), \tilde{\tilde{\mathbf{w}}}^{(t)} - \mathbf{w}^\star\rangle \leq ||\tilde{\tilde{\mathbf{w}}}^{(t)} - \mathbf{w}^\star||||L_g(\tilde{\tilde{\mathbf{w}}}^{(t)}) - \hat{L}_g^{(t)}|| \leq \frac{2}{\sqrt{\lambda}}(\frac{\gamma R}{\sqrt{\lambda}} + \gamma R)$. Using this and replacing the first term of Eq. 17 with bound in Eq. 16, we have

$$\begin{aligned}\langle \lambda\tilde{\tilde{\mathbf{w}}}^{(t)} - L_g(\tilde{\tilde{\mathbf{w}}}^{(t)}), \tilde{\tilde{\mathbf{w}}}^{(t)} - \mathbf{w}^\star\rangle \leq \frac{||\tilde{\tilde{\mathbf{w}}}^{(t)} - \mathbf{w}^\star||^2 - ||\tilde{\tilde{\mathbf{w}}}^{(t+1)} - \mathbf{w}^\star||^2}{2\alpha^{(t)}} + \frac{\alpha^{(t)}}{2}c^2 \\ + \frac{2}{\sqrt{\lambda}}(\frac{\gamma R}{\sqrt{\lambda}} + \gamma R)\end{aligned} \tag{18}$$

Using this bound in Eq. 15, and expanding the sum over t, we get

$$\begin{aligned}\sum_{t=1}^T f(\hat{\mathbf{w}}_i^{(t)}) - f(\mathbf{w}^\star) \leq c\sum_{t=1}^T ||\hat{\mathbf{w}}_i^{(t)} - \tilde{\tilde{\mathbf{w}}}^{(t)}|| + (\frac{1}{2\alpha^{(1)}} - \frac{\lambda}{2})||\tilde{\tilde{\mathbf{w}}}^{(1)} - \mathbf{w}^\star||^2 \\ - \left(\frac{1}{2\alpha^{(T)}}\right)||\tilde{\tilde{\mathbf{w}}}^{(T+1)} - \mathbf{w}^\star||^2 + \frac{c^2}{2}\sum_{t=1}^T \alpha^{(t)} \\ + \sum_{t=2}^T \left(\frac{1}{2\alpha^{(t)}} - \frac{1}{2\alpha^{(t-1)}} - \frac{\lambda}{2}\right)||\tilde{\tilde{\mathbf{w}}}^{(t)} - \mathbf{w}^\star||^2 + \frac{2T}{\sqrt{\lambda}}(\frac{\gamma R}{\sqrt{\lambda}} + \gamma R)\end{aligned} \tag{19}$$

Substituting in for $\alpha^{(t)} = \frac{1}{\lambda t}$, we find that the second term and the fifth term is zero. Fourth term is bounded by $\frac{c^2 log(T)}{2\lambda}$. We ignore the third term. Using the fact that $||\hat{\mathbf{w}}_i^{(t)}||$, $||\mathbf{w}^\star||$, and $||\tilde{\tilde{\mathbf{w}}}^{(t)}||$ are $\leq \frac{1}{\sqrt{\lambda}}$, the first term is bounded by $\frac{2cT}{\sqrt{\lambda}}$. Thus simplifying the Eq. 19 and diving by T, we get

$$\sum_{t=1}^T f(\hat{\mathbf{w}}_i^{(t)}) - f(\mathbf{w}^\star) \leq \frac{2c}{\sqrt{\lambda}} + \frac{c^2 log(T)}{2T\lambda} + \frac{2}{\sqrt{\lambda}}\left(\frac{\gamma R}{\sqrt{\lambda}} + \gamma R\right)$$

Using the property of f being a convex function, we have

$$f(\bar{\mathbf{w}}_i) - f(\mathbf{w}^\star) \leq \frac{2c}{\sqrt{\lambda}} + \frac{c^2 log(T)}{2T\lambda} + \frac{2}{\sqrt{\lambda}}\left(\frac{\gamma R}{\sqrt{\lambda}} + \gamma R\right) \tag{20}$$

□

B. Supplements to Empirical Results

The results presented in Section 5.5.3 compares GADGET SVM with state-of-the-art SVM solvers. One such algorithm studied is a pseudo-distributed version of ADMM. Table 7 presents the parameters used in our experiments.

Data	Lambda	Rho	Alpha
Gisette	1000	0.7	2.4
Sido	1000	0.7	10.5
Quantum	1	1	1
Protein	0.001	10000	1.2
Adult	1000	0.8	1.4
Covertype	0.0001	0.000001	1
Real-sim	NA	NA	NA

Table 7 Parameters of the ADMM Algorithm used in experiments.