Large scale support vector regression for aviation safety

Kamalika Das*, Kanishka Bhaduri†, Bryan L. Matthews‡, Nikunj C. Oza§
*UARC, NASA Ames Research Center, Moffett Field, CA 94035. Email: Kamalika.Das@nasa.gov
†Intuit Inc, 2700 Coast Ave. Mountain View, CA 94043. Email: kanishka.bh@gmail.com
‡SGT Inc., NASA Ames Research Center Moffett Field, CA 94035. Email: Bryan.L.Matthews@nasa.gov
§NASA Ames Research Center Moffett Field, CA 94035. Email: Nikunj.C.Oza@nasa.gov

***Abstract*—Regression problems on massive data sets are ubiquitous in many application domains including the Internet, earth and space sciences, and aviation. Support vector regression (SVR) is a popular technique for modeling the input-output relations of a set of variables under the added constraint of maximizing the margin, thereby leading to a very generalizable and regularized model. However, for a dataset with m training points, it is challenging to build SVR models due to the $O(m^3)$ cost involved in building them. In this paper we propose ParitoSVR — a parallel iterated optimizer for Support Vector Regression in the primal that can be deployed over a network of machines, where each machine iteratively solves a small (sub-)problem based only on the data observed locally and these solutions are then combined to form the solution to the global problem. Our proposed method is based on the Alternating Direction Method of Multipliers (ADMM) optimization technique. Unlike many other existing techniques, ParitoSVR is provably convergent to the results obtained from the centralized algorithm, where the optimization has access to the entire data set. The experimental results show that the algorithm is scalable both with respect to accuracy and time to convergence. We use ParitoSVR to identify flights having anomalous fuel consumption from a large fleet-wide commercial aviation database containing thousands of flights. Along with the algorithmic contributions, this paper also describes the process of deployment of the ADMM-based SVR method on a multicore architecture, namely, the NASA Pleiades supercomputing infrastructure. We have been successful in running ParitoSVR on millions of training data points and hundreds of compute nodes.**

***Keywords*-distributed optimization; support vector regression; aviation;**

I. Introduction

In many application domains, it is important to predict the value of one feature based on certain other measured features. For example, in commercial aviation, it is very important to model the fuel consumption based on input parameters such as aircraft speed, wind speed, control surfaces, engine power, pitch, roll, yaw etc. This is because according to the Air Transportation Association (ATA), fuel is an airline's largest expense at a staggering 17.5 billion gallons per year[1]. Identifying flights with abnormal fuel consumption may help the airlines to do proper maintenance of these aircrafts and save operating costs. One gallon of burnt aviation gasoline results in 8.32 kilograms of carbon emission[2]. Given that the aviation industry already is responsible for approximately 5% of the global radiative forcing (as of 2005) and this green house emission is increasing at an alarming rate [1], keeping fuel consumption under control by monitoring the health of an aircraft can also reduce the carbon footprint of the aviation industry significantly. For such problems, simple linear regression based on minimizing the mean squared error between the true and the predicted values can model the relationship between the input and the target features, and then these regression models can be used to identify those observations which are outside of the normal operating regime.

Support vector machines (SVM) are a class of well-known classification and regression algorithms which are referred to as maximum margin learning methods. They maximize the margin between the inputs and also allow for slack variables and a regularization term, leading to a more generalizable model compared to least squares linear regression. Another advantage of the support vector model is its capability of handling nonlinearity in the data by first projecting the data on to a high dimensional space and then drawing a linear hyperplane in that space, thereby retaining the linear model fitting algorithm's benefit of finding the optimal model for the training set. To avoid this mapping explicitly, support vector machines exploit the kernel trick, in which a matrix is built to capture the relationship among all the inputs, and then a modified optimization problem is solved using the kernel matrix. Both linear and nonlinear support vector methods have been shown to have excellent performance for a variety of tasks.

Support vector machines, however, face computational challenges — learning is generally $O(m^3)$, where m is the number of training points. This scalability issue renders this extremely useful data mining tool useless for large scale applications where m can easily be 10^6 or more. To ease this computational burden, many sophisticated SVM learning methods have been proposed in the literature. For example, SVM^{light} [2] uses the working set principle, decomposition-based methods such as SMO [3] solve the SVM problem by approximating the dual optimization for-

[1]http://www.airlines.org/Energy/Fuels101/Pages/AirlineEnergyQA.aspx

[2]U.S. Energy Information Administration (http://205.254.135.24/oiaf/1605/coefficients.html#tbl2)

mulation and solving a smaller problem at each iteration. More recently, fast algorithms have also been proposed for solving the primal formulation using some variants of finite Newton methods. Many parallel/distributed solutions have also been built relying on some of these techniques, both for SVM and SVR. Some examples are cascade SVM by Graf *et al.* [4] and the PSVM by Chang *et al.* [5]. Most of these techniques are approximate and do not guarantee optimality.

In this paper we propose Parallel Iterated Optimizer for Support Vector Regression in the Primal (ParitoSVR), a new support vector regression algorithm that can be deployed over a network of machines, where each machine solves a small (sub-)problem based only on the data observed locally and these solutions are then combined to form the solution to the global problem. Our proposed method is based on the Alternating Direction Method of Multipliers (ADMM) optimization technique [6], which is parallelizable for separable convex problems, and converges to the exact solution as the centralized version with theoretical guarantees. Our contribution in this paper can be summarized as below:

- We propose an ADMM-based parallel iterated optimizer for solving the primal problem in support vector regression. We demonstrate that the objective function in our formulation is separable and propose a parallel implementation of the above method for addressing large scale applications.
- Unlike existing distributed support vector regression techniques, our proposed method computes the globally optimal solution of the objective function, based on only the locally available data and minimal message passing. We solve the global optimization problem involving all the variables with certain amount of numerical approximation errors resulting from the iterative convergence to the globally optimal solution based on a user-defined threshold.
- The parallel algorithm is provably convergent to the results obtained from the centralized algorithm, where the optimization has access to the entire data set.
- We have implemented and executed the algorithm on NASA Pleiades supercomputer facility[3] and run the algorithm on millions of data points and hundreds of computing nodes. Our experiments on a variety of real-world data sets shows that the proposed method is highly accurate and scales well both with respect to the number of training points and the number of distributed machines.
- We use ParitoSVR to identify flights having anomalous fuel consumption from a large fleet-wide commercial aviation database containing thousands of flights.

[3]http://www.nas.nasa.gov/hecc/resources/pleiades.html

II. Background

Our ParitoSVR algorithm uses as a building block two components: (1) a parallelizable optimization technique known as Alternating Direction Method of Multipliers (ADMM), and (2) the primal optimization formulation of the support vector regression (SVR) problem. In this section, we discuss these two topics before using these to develop our ParitoSVR algorithm.

A. Alternating direction method of multipliers (ADMM)

Alternating Direction Method of Multipliers (ADMM) [6] is a decomposition algorithm for solving separable convex optimization problems of the form:

$$\begin{aligned} \min_{\mathbf{x},\mathbf{y}} \quad & G_1(\mathbf{x}) + G_2(\mathbf{y}) \\ \text{subject to} \quad & A\mathbf{x} - \mathbf{y} = 0, \quad \mathbf{x} \in \mathbb{R}^n, \quad \mathbf{y} \in \mathbb{R}^m \end{aligned} \tag{1}$$

where $A \in \mathbb{R}^{m \times n}$ and G_1 and G_2 are convex functions. The method works as follows. First, the augmented Lagrangian of the actual objective is formed:

$$L_\rho(\mathbf{x}, \mathbf{y}, \mathbf{z}) = G_1(\mathbf{x}) + G_2(\mathbf{y}) + \mathbf{z}^T(A\mathbf{x} - \mathbf{y}) + \rho/2 \left\| A\mathbf{x} - \mathbf{y} \right\|_2^2$$

where ρ is a positive constant called the penalty parameter and $\mathbf{z}$ is the dual variable. ADMM iterations can then be written as:

$$\begin{aligned} \mathbf{x}^{t+1} &= \min_{\mathbf{x}} \left\{ G_1(\mathbf{x}) + (\mathbf{z}^t)^T A\mathbf{x} + \rho/2 \left\| A\mathbf{x} - \mathbf{y}^t \right\|_2^2 \right\} \\ \mathbf{y}^{t+1} &= \min_{\mathbf{y}} \left\{ G_2(\mathbf{y}) - (\mathbf{z}^t)^T \mathbf{y} + \rho/2 \left\| A\mathbf{x}^{t+1} - \mathbf{y} \right\|_2^2 \right\} \\ \mathbf{z}^{t+1} &= \mathbf{z}^t + \rho \left(A\mathbf{x}^{t+1} - \mathbf{y}^{t+1} \right) \end{aligned}$$

This is an iterative technique where the superscript t denotes the iteration number, and the initial vectors $\mathbf{y}^0$ and z^0 can be chosen arbitrarily. ADMM can be written in a different form (known as the scaled form) by combining the linear and quadratic terms of the Lagrangian as follows:

$$\mathbf{z}^T(A\mathbf{x} - y) + \rho/2 \left\| (A\mathbf{x} - \mathbf{y}) \right\|_2^2 = \rho/2 \left\| (A\mathbf{x} - \mathbf{y}) + (1/\rho)\mathbf{z} \right\|_2^2 - 1/(2\rho) \left\| \mathbf{z} \right\|_2^2$$

Now defining the scaled dual variable $\mathbf{p} = (1/\rho)\mathbf{z}$, the iterations of ADMM become:

$$\begin{aligned} \mathbf{x}^{t+1} &= \min_{\mathbf{x}} \left\{ G_1(\mathbf{x}) + \rho/2 \left\| A\mathbf{x} - \mathbf{y}^t + \mathbf{p}^t \right\|_2^2 \right\} \\ \mathbf{y}^{t+1} &= \min_{\mathbf{y}} \left\{ G_2(\mathbf{y}) + \rho/2 \left\| A\mathbf{x}^{t+1} - \mathbf{y} + \mathbf{p}^t \right\|_2^2 \right\} \\ \mathbf{p}^{t+1} &= \mathbf{p}^t + A\mathbf{x}^{t+1} - \mathbf{y}^{t+1} \end{aligned}$$

ADMM effectively decouples the $\mathbf{x}$ and $\mathbf{y}$ updates such that parallel execution becomes possible. In a distributed computing framework, this becomes ever more interesting since each computing node can now solve a (smaller) subproblem in $\mathbf{x}$ independently, and then, these solutions can be efficiently gathered to compute the consensus variable $\mathbf{y}$ and the dual variable $\mathbf{p}$. It has been argued in the literature that ADMM is slow to converge especially when very high precision is desired. However, ADMM converges within a few iterations when moderate precision is good enough. This can be particularly useful for many large scale problems, similar to the ones we consider in this paper.

Critical to the working and convergence of the ADMM method is the termination criterion. The primal and dual residuals at each iteration are:

$$r_p^{t+1} = A\mathbf{x}^{t+1} - \mathbf{y}^{t+1} \quad \text{(primal residual)}$$
$$r_d^{t+1} = \rho A^T(\mathbf{y}^{t+1} - \mathbf{y}^t) \quad \text{(dual residual)}$$

A reasonable termination criterion is when the primal and the dual residuals are below some tolerance *i.e.*

$$\left\|r_p^{t+1}\right\|_2 \leq \epsilon_p \quad \text{and} \quad \left\|r_d^{t+1}\right\|_2 \leq \epsilon_d.$$

where ϵ_p and ϵ_d are the primal and dual feasibility tolerances defined as,

$$\epsilon_p = \epsilon_1\sqrt{m} + \epsilon_2 \max\left(\left\|A\mathbf{x}^{t+1}\right\|_2, \left\|\mathbf{y}^{t+1}\right\|_2\right)$$
$$\epsilon_d = \epsilon_1\sqrt{n} + \epsilon_2 \left\|A^T\mathbf{p}^{t+1}\right\|_2.$$

where ϵ_1 and ϵ_2 are user-specified thresholds for precision. These precisions are problem dependent; we have specified the values for our experiments in Section V.

B. Support vector regression in the primal

Support vector machine [7] is a powerful tool for a wide variety of regression and classification tasks, yielding good predictive performance on many datasets. In this section, we present a brief introduction to support vector regression. For details, interested readers can refer to the citations above.

Give m data tuples (training set) $\mathcal{D} = (\mathbf{x}_i, y_i)_{i=1}^m$, where $\mathbf{x}_i \in \mathbb{R}^n$ is the input and $y_i \in \mathbb{R}$ is the corresponding output or target. To obtain a linear predictor, SVR solves the following optimization problem:

$$\min_{\mathbf{w}\in\mathbb{R}^n, b\in\mathbb{R}} \left[\frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^m(\xi_i^+ + \xi_i^-)\right]$$

subject to

$$\mathbf{w}\cdot\mathbf{x}_i + b - y_i \leq \xi_i^+ + \xi_i^-$$
$$y_i - \mathbf{w}\cdot\mathbf{x}_i + b \leq \xi_i^+ + \xi_i^-, \text{and } \xi_i^+ \geq 0, \xi_i^- \geq 0,$$

$\forall i \in \{1:m\}$, where

- C is a positive constant,
- $f(\mathbf{x}) = \mathbf{w}\cdot\mathbf{x} + b$ is the learned hyperplane, and
- ξ^+, ξ^- are the slack variables.

This is called the primal formulation of linear SVR. The unconstrained optimization problem is:

$$\min_{\mathbf{w},b}\left[\lambda||\mathbf{w}||^2 + \sum_{i=1}^m \ell_\varepsilon(\mathbf{w}\cdot\mathbf{x}_i + b - y_i)\right] \tag{2}$$

where $\lambda = \frac{1}{2C}$ and ℓ_ε is the *ε-insensitive loss function* defined as, $\ell_\varepsilon(r) = \max(|r| - \varepsilon, 0)$. This is a convex optimization problem which can be solved using convex optimization solvers such as CVX[4]. Eqn. 2 is the linear SVR model, which uses the primal objective function without any explicit feature mapping. Note that, in the above equation, we can easily remove the bias term b, by adding a column of 1's to the data matrix $\mathcal{D}$ and then adjusting the dimensionality of $\mathbf{w}$ accordingly.

In the next section we show how to build SVR models for very large datasets using distributed computing via the ADMM technique.

[4]http://cvxr.com/cvx/

III. PARITOSVR FORMULATION

For ParitoSVR algorithm setup, we assume that the training data is distributed among N client processors (nodes) $P_1, \ldots, P_N$ with a central machine P_0 acting as the server or collector. The dataset at machine P_j, denoted by D_j, consists of m_j data points *i.e.* $D_j = \left\{\mathbf{x}_i^{(j)}, y_i^{(j)}\right\}_{i=1}^{m_j}$. It is assumed that the datasets are disjoint: $D_i \bigcap D_j = \emptyset$ and $\bigcup_{j=1}^N D_j = D$, where D is the total (global) data set. The goal is to learn a linear support vector regression model on D without exchanging all of the data among all the nodes.

Given Eqn. 2, the optimization problem is now:

$$\min_{\mathbf{w}}\left[\sum_{i=1}^m \ell_\varepsilon(\mathbf{w}\cdot\mathbf{x}_i - y_i) + \lambda||\mathbf{w}||^2\right]$$
$$\Leftrightarrow \min_{\mathbf{w}}\left[\sum_{j=1}^N\sum_{i=1}^{m_j} \ell_\varepsilon\left(\mathbf{w}\cdot\mathbf{x}_i^{(j)} - y_i^{(j)}\right) + \lambda\left\|\mathbf{w}\right\|^2\right]$$

The inner sum can be computed by each node independently (assuming that $\mathbf{w}$ is known). We next write it in a form such that it is decoupled across the nodes:

$$\min_{\mathbf{w}_1,\ldots,\mathbf{w}_N,\mathbf{z}}\left[\sum_{j=1}^N\sum_{i=1}^{m_j} \ell_\varepsilon\left(\mathbf{w}_j\cdot\mathbf{x}_i^{(j)} - y_i^{(j)}\right) + \lambda\left\|\mathbf{z}\right\|^2\right] \tag{3}$$
$$\text{subject to} \quad \mathbf{w}_j = \mathbf{z}$$

In the ADMM decomposition, each node can solve its local problem using its own data and optimization variable and then coordinate the results across the nodes to drive them into consensus. The nodes update the consensus variable $\mathbf{z}$ iteratively, based on their local data and scatter-gather operations on $\mathbf{z}$ until they converge to the same result.

Theorem 3.1: The ADMM update rules for support vector regression primal optimization are:

$$\mathbf{w}_j^{t+1} = \min_{\mathbf{w}_j}\left\{\sum_{i=1}^{m_j} \ell_\varepsilon\left(\mathbf{w}_j\cdot\mathbf{x}_i^{(j)} - y_i^{(j)}\right) + \frac{\rho}{2}\left\|\mathbf{w}_j - \mathbf{z}^t - \mathbf{u}_j^t\right\|_2^2\right\}$$
$$\mathbf{z}^{t+1} = \min_{\mathbf{z}}\left\{\lambda\left\|\mathbf{z}\right\|_2^2 + \frac{N\rho}{2}\left\|\mathbf{z} - \overline{\mathbf{w}}^{t+1} - \overline{\mathbf{u}}^t\right\|_2^2\right\}$$
$$\mathbf{u}_j^{t+1} = \mathbf{u}_j^t + \mathbf{w}_j^{t+1} - \mathbf{z}^{t+1}$$

where $\mathbf{u} \in \mathbb{R}^n$ is the (scaled) dual variable and $\overline{\mathbf{w}}^{t+1}$ and $\overline{\mathbf{u}}^{t+1}$ are the averages of the variables over all the nodes.

Proof: We can write the augmented Lagrangian for the objective function in Eqn. 3 as:

$$L_\rho = \sum_{j=1}^N\left[\sum_{i=1}^{m_j} \ell_\varepsilon\left(\mathbf{w}_j\cdot\mathbf{x}_i^{(j)} - y_i^{(j)}\right) + \lambda\left\|\mathbf{z}\right\|^2 + \mathbf{m}_j^T(\mathbf{w}_j - \mathbf{z}) + \frac{\rho}{2}\left\|\mathbf{w}_j - \mathbf{z}\right\|_2^2\right]$$

where $\mathbf{m}_j \in \mathbb{R}^n$ is the dual variable. From Section II-A we know that each ADMM iteration consists of alternating between minimizing the primal and consensus variables ($\mathbf{w}$ and $\mathbf{z}$ respectively), followed by an update of the dual variable (m in this case).

$$\begin{aligned}
\mathbf{w}_j^{t+1} &= \min_{\mathbf{w}_j} \left\{ \sum_{i=1}^{m_j} \ell_\varepsilon \left(\mathbf{w}_j \cdot \mathbf{x}_i^{(j)} - y_i^{(j)} \right) \right. \\
&+ \left. \mathbf{m}_j^T (\mathbf{w}_j - \mathbf{z}) + \frac{\rho}{2} \left\| \mathbf{w}_j - \mathbf{z} \right\|_2^2 \right\} \\
\mathbf{z}^{t+1} &= \min_{\mathbf{z}} \left\{ \lambda \left\| \mathbf{z} \right\|_2^2 + \sum_{j=1}^{N} \left[(-\mathbf{m}_j^k)^T \mathbf{z} + \frac{\rho}{2} \left\| \mathbf{w}_j^{t+1} - \mathbf{z} \right\|_2^2 \right] \right\} \\
\mathbf{m}_j^{t+1} &= \mathbf{m}_j^t + \rho \left(\mathbf{w}_j^{t+1} - \mathbf{z}^{t+1} \right)
\end{aligned}$$

Let $\mathbf{u}_j = \frac{1}{\rho} \mathbf{m}_j$ be the scaled dual variable. Then the $\mathbf{w}$-update can be rewritten as,

$$\mathbf{w}_j^{t+1} = \min_{\mathbf{w}_j} \left\{ \sum_{i=1}^{m_j} \ell_\varepsilon \left(\mathbf{w}_j \cdot \mathbf{x}_i^{(j)} - y_i^{(j)} \right) + \frac{\rho}{2} \left\| \mathbf{w}_j - \mathbf{z}^t + \mathbf{u}_j^t \right\|_2^2 \right\}$$

For the $\mathbf{z}$ update, let $\overline{\mathbf{w}^{t+1}} = \frac{1}{N} \sum_{j=1}^{N} \mathbf{w}_j^{t+1}$ and $\overline{\mathbf{u}^t} = \frac{1}{N} \sum_{j=1}^{N} \mathbf{u}_j^t$. Then the $\mathbf{z}$-update can be rewritten as:

$$\mathbf{z}^{t+1} = \frac{N\rho}{2\lambda + N\rho} \left(\overline{\mathbf{w}^{t+1}} + \overline{\mathbf{u}^t} \right) \tag{4}$$

Finally, the dual variable update can be simplified as,

$$\mathbf{u}_j^{t+1} = \mathbf{u}_j^t + \mathbf{w}_j^{t+1} - \mathbf{z}^{t+1} \tag{5}$$

■

The $\mathbf{w}$ update can be executed in parallel for each machine. It involves solving a convex optimization problem in $n+1$ variables at each node. This solution depends only on the data available at that partition. The $\mathbf{z}$ update step involves computing the average of the $\mathbf{w}$ and $\mathbf{u}$ vectors in order to combine the results from the different partitions. Critical to the working of ADMM is the convergence criteria. The primal and dual residuals can be written as:

$$r_p^t = \left\| \mathbf{w}^t - \mathbf{z}^t \right\|_2^2 \qquad r_d^t = \left\| \rho(\mathbf{z}^t - \mathbf{z}^{t-1}) \right\|$$

Also, given the thresholds ϵ_{pri} and ϵ_{dual}, the primal and dual thresholds can be written as,

$$\epsilon_{pri} = \epsilon_{abs}\sqrt{m} + \epsilon_{rel} \max(\left\| \mathbf{w} \right\|, \left\| \mathbf{z} \right\|) \quad \text{and}$$
$$\epsilon_{dual} = \epsilon_{abs}\sqrt{m} + \rho\epsilon_{rel} \left\| \mathbf{u} \right\|.$$

The iterations terminate when $r_p^t < \epsilon_{pri}$ and $r_d^t < \epsilon_{dual}$.

The pseudo code of ParitoSVR is presented in Alg. 1, Alg. 2, and Alg. 3. Alg. 1 is the driver which calls the **ADMM_SVR** routine to split the data into N chunks. It then calls the **RunDistributedJob** function in parallel for these N subproblems to find the $\mathbf{w}$ minimizer. Then it aggregates the results and updates the $\mathbf{z}$ and $\mathbf{u}$ variables. This process is repeated until the primal and dual residuals fall below the thresholds or the total number of iterations exceed MAXITER.

Input: D, ε, λ, ρ, N, $MaxIter$
Output: $\mathbf{w}$ of the SVR model
Initialization: Initialize $\mathbf{z}^0$, $\mathbf{u}^0$

Split D into $D_1, \ldots, D_N$;
Call **ADMM_SVR**($D_1, \ldots, D_N$, ε, λ, ρ, N, $MaxIter$)

Algorithm 1: ParitoSVR

Procedure **ADMM_SVR**($D_1, \ldots, D_N$, ε, λ, ρ, N, $MaxIter$)
forall the *t=1 to* $MaxIter$ **do**
  forall the *j=1 to* N **do**
    $\mathbf{w}_j^t$ = **RunDistributedJob**(D_j, ε, λ, ρ, $\mathbf{u}^{t-1}$, $\mathbf{z}^{t-1}$);
  end
  $\mathbf{z}^t = \frac{N\rho}{2\lambda+N\rho} \left(\overline{\mathbf{w}^t} + \overline{\mathbf{u}^t} \right)$;
  forall the *j=1 to* N **do**
    $\mathbf{u}_j^t = \mathbf{u}_j^{t-1} + \mathbf{w}_j^t - \mathbf{z}^t$;
  end
  if $r_p^t < \epsilon_{pri}$ *and* $r_d^t < \epsilon_{dual}$ **then**
    break;
  end
end
Return $\overline{\mathbf{w}}$

Algorithm 2: Procedure ADMM_SVR

IV. Analysis

In this section we present the message complexity, running time and convergence analysis of ParitoSVR.

A. Message complexity of ParitoSVR

In each iteration, the distributed nodes compute $\mathbf{w}$ based on last iterations $\mathbf{u}$ and $\mathbf{z}$. Hence, we need to send $\mathbf{u}$ and $\mathbf{z}$ to the compute nodes, resulting in $2n$ message complexity, assuming each vector is in $\mathbb{R}^n$. As $\mathbf{w}$ needs to be broadcast by the compute nodes, per node messages increase to $3n$. Hence the overall message complexity is $MaxIter \times 3nN$. Note that this is independent of the number of training data points or m.

B. Running time of ParitoSVR

The total running time is comprised of three components. (1) The $\mathbf{w}$-update which needs iterating over all m training examples once for computing the inner product with the n dimensional $\mathbf{w}$ vector. The complexity of this step is $O(mn/N)$ (per node). The worst case number of iterations in the CVX toolbox has a theoretical bound of $\sqrt{n}\log(1/\alpha)$, where α is a very small ($\approx 10^{-9}$) constant. The cost per

Procedure **RunDistributedJob**($\mathcal{D}$, ε, λ, ρ, $\mathbf{u}^{t-1}$, $\mathbf{z}^{t-1}$)
$\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^{m_j}$

$$\mathbf{w} = \min_{\mathbf{w}} \left\{ \sum_{i=1}^{m_j} \ell_\varepsilon \left(\mathbf{w} \cdot \mathbf{x}_i - y_i \right) + \frac{\rho}{2} \left\| \mathbf{w} - \mathbf{z}^{t-1} + \mathbf{u}^{t-1} \right\|_2^2 \right\}$$

Return $\mathbf{w}$

Algorithm 3: Procedure RunDistributedJob

iteration is a polynomial of the problem dimension using interior point method ([8], page 8). The overall complexity of this step is poly$(n)\sqrt{n}\log(1/\alpha)$. (2) $\mathbf{z}$ update requires us to compute the average of the n dimensional $\mathbf{w}$ and $\mathbf{u}$ vectors which costs $O(Nn)$, and (3) $\mathbf{u}$ update is an $O(n)$ computation. Therefore, the overall running time is $O(t\max\{poly(n)\sqrt{n}\log(1/\alpha), mn/N\})$, where t is the number of iterations of the ParitoSVR algorithm. It should be noted that both centralized and distributed ParitoSVR both need to be run for the same number of iterations to achieve convergence.

C. Convergence of ParitoSVR

Due to space constraints, formal convergence analysis is outside the scope of this paper.

Informally, let $G_1(\mathbf{w}) = \ell_\varepsilon(\mathbf{w}\cdot\mathbf{x}_i - y_i)$ and $G_2(\mathbf{w}) = ||\mathbf{w}||^2$ be closed, proper, and convex functions of $\mathbf{w}$. Then, convergence of ParitoSVR is guaranteed [6], thereby implying:

- primal and dual residual approaches 0 *i.e.* $\left\|r_p^t\right\|_2^2 \to 0$ and $\left\|r_d^t\right\|_2^2 \to 0$ as $k \to \infty$
- the objective function approaches the optimal value
- dual variable approaches feasibility

V. Experiments

In this section we demonstrate the performance of the ParitoSVR algorithm on a variety of real life datasets.

Setup: All experiments to understand the behavior of ParitoSVR algorithm were run on NASA's Pleiades Supercomputer[5] with the following hardware and software configuration. Each of the worker nodes are based on the Intel Sandy Bridge architecture with dual 8 core 2.6 GHz processors and with 32 GB of memory. All nodes' operating systems are running SGI ProPack for Linux kernel version 3.0. The nodes are connected to a cluster file system mass storage device as well as a network file system for home drives via InfiniBand with a bandwidth of 56 Gb/sec. The local nodes' /tmp drive mount is 16 GB, which can be used to store local files for quick access once the tasks begin. Pleiades utilizes a PBS scheduler to coordinate the large amount of jobs submitted by hundreds of users. For solving the convex problems at each iteration, we have used Matlab convex optimization toolbox[6].

We have run our experiments on five datasets[7], four of which have been previously used in [10]. The fifth dataset is a proprietary aircraft performance data from a large commercial airline company. The description of the datasets along with the parameters used for our study of the algorithms are given in Table I. Unless otherwise stated, for each dataset, we have used 75% of the data for training and 25% for testing.

For evaluating the performance of our algorithms, we have measured the RMSE metric on the test dataset defined as:

$$\frac{1}{\max\limits_{i=1:m} y_i}\sqrt{\frac{1}{m}\sum_{i=1}^{m}(f(\mathbf{x}_i)-y_i)}$$

We have compared the performance (RMSE and running time) of our ParitoSVR algorithm with a centralized SVR algorithm using CVX for optimization and having access to all the data. The reason we have not directly compared our results with standard SVR solvers is due to two reasons: (1) most state-of-the-art SVM solvers such as $LIBSVM$ and SVM^{Light} solve the dual of the optimization problem, while we solve the primal problem, and (2) the other primal solvers such as [11][12] use a differentiable approximation to the original loss function of Vapnik [7] for using a gradient-based Newton solver. In our ADMM formulation, we do not require the loss functions to be differentiable. Nevertheless, we will refer to the accuracy results published in earlier papers whenever appropriate. We do not report the number of support vectors since they are very similar to the ones published in the literature.

ParitoSVR performance: In this section we demonstrate the performance of ParitoSVR on various datasets. In our first experiment, we show the behavior of ParitoSVR on a synthetic dataset. Fig. 1(a) shows the sample dataset generated from a linear model following $y = \mathbf{w}\times\mathbf{x}+noise$, where $\mathbf{w}$ is the weight of the regression model. We have used 2 nodes in this experiment and, for each node, chosen a different $\mathbf{w}$ vector so that each node sees a different data distribution. The data of the two nodes are shown in two different colors (circle and plus markers). Also shown in the figure are the models (straight lines) formed by node 1 at different iterations of ParitoSVR algorithm. As seen, the algorithm updates the model as iterations proceed. For comparison, we have also plotted the centralized model formed by the union of all the data. As clearly shown, ParitoSVR model in the final iteration is very close to the model formed by the centralized model. Note that, at convergence, all nodes have the same model. Fig. 1(b) shows the primal and dual residuals (r_p and r_d) in solid blue lines and the primal and dual thresholds ϵ_{pri} and ϵ_{dual} in red dotted lines. Fig. 1(c) shows the objective values for different iterations of the algorithm. For this experiment, we have used the following values of the parameters: $\varepsilon = 10, \lambda = 1, \epsilon_{pri} = 10^{-4}, \epsilon_{dual} = 10^{-2}$.

In the next set of experiments, we report the variation of RMSE and execution time for different datasets in Table I. The parameters specific to each dataset are shown in Table I. The other parameters are set to: $\epsilon_{abs} = 0.01$, $\epsilon_{rel} = 0.001$, $\rho = 0.8$, and $\mu = 1.5$. In order to evaluate the performance

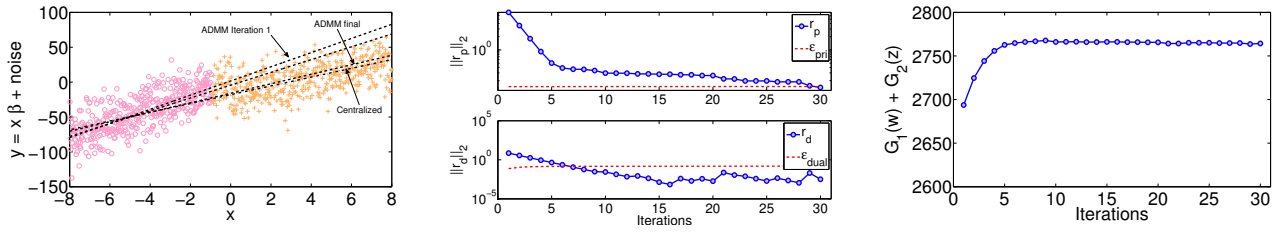
[5]http://www.nas.nasa.gov/hecc/

[6]http://cvxr.com/cvx/

[7]http://www.liaad.up.pt/~ltorgo/Regression/DataSets.html

Dataset	Description	Size	Parameters
Elevators	Predict the elevator position of an F16 aircraft based on other attributes	16559 × 18	λ=30, $\varepsilon = 10^{-8}$
Kinematics	Predict the distance of the end-effector of a 8-link robot arm from a fixed target. We have used the highly non-linear and medium noisy version.	8192 × 32	λ=1, $\varepsilon = 0.25$
Friedman[9]	This dataset is generated by first generating the values of 10 attributes, $X_1, \ldots, X_{10}$ each independently $\in [0, 1]$. The target variable Y is generated as $10\sin(\pi X_1 X_2) + 20(X_2 - 0.5)^2 + 10X_4 + 5X_5 + N(0, 1)$	40768 × 10	λ=0.1, $\varepsilon = 0.25$
CPU activity	Given a number of computer systems activity measures, the task is predict the the portion of time the CPUs run in user mode	8192 × 8	λ=1, $\varepsilon = 7$
CarrierX data	This dataset is from a fleet of commercial aircrafts. The task is to predict the amount of fuel consumed on an aircraft.	4.4 million × 29	λ=10, $\varepsilon = 1$

Table I
DESCRIPTION OF THE DATASETS AND PARAMETER VALUES.



(a) Synthetic dataset (b) Primal and dual residuals (c) Objective function

Figure 1. (a) Synthetic dataset. Also shown are the models formed by node 1 as the algorithm progresses. (b) Primal and dual residuals (blue solid line) for different iterations of ParitoSVR algorithm. The red dotted lines show the ϵ_{pri} and ϵ_{dual}. (c) Change in objective value with different iterations.

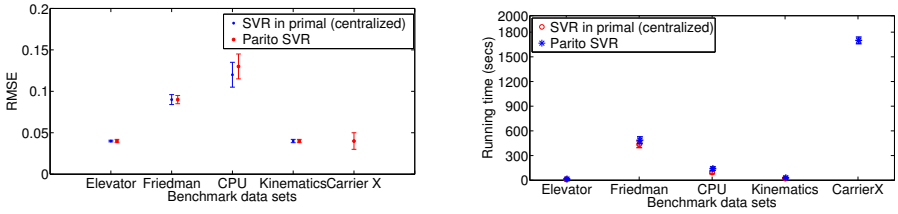
of ParitoSVR under distributed setting, we have compared the algorithm to a centralized execution which has access to all the data and is run on a single machine. Therefore, in each subplot of Figure 2, we have 2 graphs – centralized execution and distributed execution. We have only used 2 nodes for each of the distributed experiments since the dataset sizes are reasonably small. As shown in the RMSE plot, both the executions have similar RMSE values for all the datasets (all close to 0.1), indicating very good model fit.

Moreover, from literature [10], it is evident that our results are very similar to those obtained by running *LIBSVM* and SVM^{Light} on the same datasets, under similar settings. For the distributed setting, running time consists of the total time which includes the job submission time, the network latency and the time CVX takes to solve each subproblem. The running time plot in Fig. 2 presents an interesting phenomenon. For each dataset, the execution time of the distributed setting is greater than the corresponding centralized setting. Although each distributed execution solves a smaller subproblem, the total time is greater due to job overhead and network latency.

Scalability of ParitoSVR: In order to assess the scalability of ParitoSVR, we have run experiments using the CarrierX dataset. Specifically, we are interested in analyzing the effects of varying the (1) number of computing cores, (2) number of data points per core, and (3) both. We run each experiment 5 times and report the mean and the variance of those running times. Fig. 3 presents the results. In Fig. 3(a), the running time shows a linear growth as the number of cores is increased while keeping the number of data points fixed at 250 per core. This trend is primarily due to the overhead of computation, including network latency, and the cost to aggregate the information back at the central core. Fig. 3(b) shows the effect of running time on increasing the total number of flights, while keeping the total number of cores fixed. The running time shows super-linear growth. This is expected, since with increasing number of flights to process, the per core computation load increases. Finally, in Fig. 3(c) we show the effect of increasing both the number of cores and the data points per core. As expected, the running time decreases, as the number of cores increases. Although there are two competing forces at play here, viz. increasing number of cores vs. increasing size of dataset at each core, we still see a decreasing trend in running time due to greater benefits of increasing the number of cores.

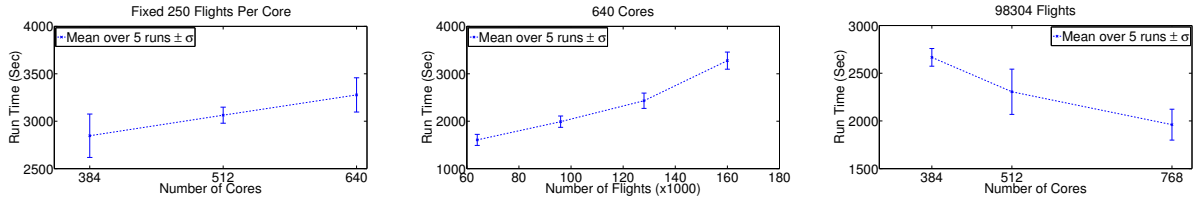
Figure 4 shows the NMSE on a hold out set of 500 flights, for different runs of the algorithm. In each run, we varied the size of the training dataset (from 64K to 160K flights) while keeping the test data fixed. The number of cores for this experiment were fixed at 640. As we see in the figure, the NMSE is primarily invariant, demonstrating excellent accuracy of the algorithm.

Anomaly detection on CarrierX dataset: We use ParitoSVR algorithm to detect anomalous fuel consumption in



(a) RMSE (b) Running time

Figure 2. RMSE and execution time of ParitoSVR for different datasets both for centralized and distributed setting. For each dataset, test set size is 25% of the size of the corresponding dataset. As shown here, RMSE values of the centralized and distributed runs of ParitoSVR are comparable. The overall running time of ParitoSVR in the distributed setting shows some increase due to network latency, while the execution time becomes smaller, since smaller sub problem is solved at each node.



(a) Time for varying number of cores, fixed data points per core (b) Time for varying size of data per node, fixed number of cores (c) Time for varying size of data per node and core

Figure 3. Scalability results of ParitoSVR on CarrierX dataset.

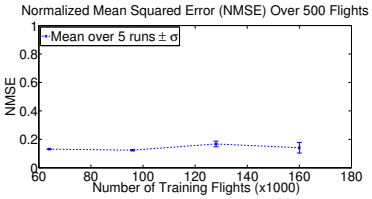


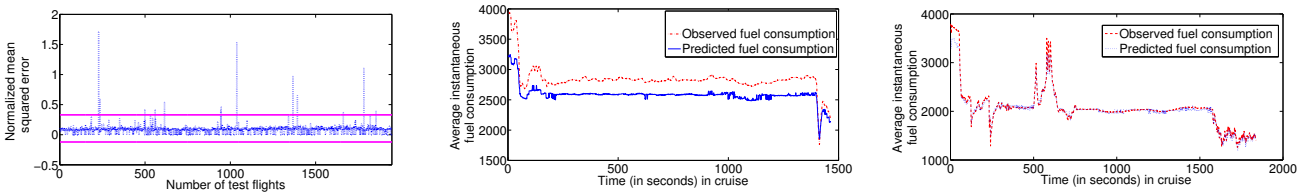
Figure 4. NMSE on test dataset for different runs of ParitoSVR algorithm. Each run is for a specific size of the training dataset.

a commercial aircraft. We model the average fuel flow as a function of 29 different parameters that measure system parameters such as lateral and longitudinal acceleration, roll and pitch angle, air pressure, and velocity, as well as external parameters such as wind speed and direction. Since the goal is to track the fuel consumption of particular tail numbers, we have used all 1500 flights ($\approx$ 4.5 million training instances) for a specific tail number for a particular year for training, and tested subsequent years' flights for predicting fuel consumption. Flights for which the mean squared errors of the predicted instantaneous fuel consumption fall outside the 3σ boundary of the average mean squared error, are tagged anomalous (σ is the standard deviation of the mean predictions). Out of approximately 1800 flights for a test year, 14 flights were determined to be anomalous. Figure 5(a) shows the mean squared errors for each of the flights in blue and the 3σ bounds in green. The instantaneous fuel flow for the top ranked anomalous flight among these 14 flights is shown in Figure 5(b). The red graph depicting observed fuel flow is significantly higher than the predicted fuel consumption, shown in blue. The outlier flight was verified by domain experts to be truly anomalous.

VI. Related work

Support vector classification and regression techniques are a class of maximum margin methods. They are posed as convex optimization problems in the primal and as quadratic programming (QP) problems in the dual. Typically, the solution to these problems scales poorly with respect to the number of training points. Researchers have proposed efficient algorithms for solving both the primal *e.g.* [11] and dual problems *e.g.* [2].

Several researchers have also developed methods for parallelizing or distributing the optimization problem in SVRs. This is particularly useful when the datasets are large and the computation cannot be executed on a single machine. The cascade SVM by Graf *et al.* [4] uses a cascade of binary SVMs arranged in an inverted binary tree topology to train a global model. The input dataset is first split into chunks and then a set of SVMs is learned in parallel. The SVs of any two pairs of SVMs in the earlier layer is combined and then a new SVM model is learned in this step. Hazan *et al.* [13] presents a method for parallel SVM learning based on the parallel Jacobi block update scheme derived from the convex conjugate Fenchel duality. In a related method, Lu *et al.* [14] have proposed techniques in which the computation is done by the local nodes and then the central node performs aggregation of the results. In their method, SVMs are learned



(a) Squared error for all flights (b) Outlier flight fuel consumption (c) Normal flight fuel consumption

Figure 5. Fuel flow study on CarrierX dataset. Fig. (a) shows squared error for all test flights, the 3-σ bound and flights which cross the threshold. Fig. (b) shows the observed and predicted fuel flow of top ranked anomalous flight. Fig. (c) shows the same for a normal flight.

at each node independently and then the SVs are passed onto the other nodes for updating the models of the other nodes.

The work most related to the algorithm proposed here is the consensus SVM method discussed by the *et al.* [15]. There are several major differences between their work and ours. Firstly, the goal of [15] is to build a global SVM model in a sensor network without any central authority. As a result, they develop an asynchronous algorithm in which messages are exchanged only among the neighboring nodes. In contrast, our goal is to build a single SVR model over a very large dataset and to achieve this we assume a client-server topology. Secondly, in their work, they assume that each node has m_j number of data points, of which $\mathbf{x}_1, \ldots, \mathbf{x}_\ell$ are common to all nodes. The ADMM algorithm discussed in [15] ensures that at termination, each node achieves a consensus only in these entries of the kernel matrix. In our case, each node has a partition of the kernel matrix and our goal is to achieve consensus on all the entries at termination.

VII. Conclusion

In this paper we have proposed a parallel iterated optimizer which solves support vector regression in the primal. Our formulation is parallelizable among a number of computing nodes connected to a central computing node. In every iteration of the ParitoSVR algorithm, each node solves sub problems using its local data and then these partial results are aggregated to form the solution to the final problem. Theoretical and experimental results show that our algorithm is accurate and scalable, ideal for large scale deployment. As future work, we plan to develop asynchronous version of this problem for peer-to-peer architectures.

Acknowledgements

The material is based upon work supported by the ARMD seedling fund from NASA under Prime Contract Number NAS2-03144 awarded to the University of California, Santa Cruz, University Affiliated Research Center.

References

[1] J.E.Penner, D.H.Lister, D.J.Griggs, D.J.Dokken, and M.McFarland, *Aviation and the Global Atmosphere - IPCC 1999*. Cambridge University Press, 1999.

[2] T. Joachims, *Making large-scale support vector machine learning practical*. MIT Press, 1999, pp. 169–184.

[3] J. C. Platt, *Fast training of support vector machines using sequential minimal optimization*. MIT, 1999, pp. 185–208.

[4] H. Graf, E. Cosatto, L. Bottou, I. Durdanovic, and V. Vapnik, "Parallel Support Vector Machines: The Cascade SVM," in *Proceedings of NIPS'04*, 2004.

[5] E. Chang, K. Zhu, H. Wang, H. Bai, J. Li, Z. Qiu, and H. Cui, "PSVM: Parallelizing Support Vector Machines on Distributed Computers," in *Proceedings of NIPS'07*, 2007.

[6] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers," *Found. and Trends in Mach. Learn.*, vol. 3, no. 1, pp. 1–122, 2011.

[7] V. V. N. Vapnik, *The Nature of Statistical Learning Theory*. Springer-Verlag New York, Inc., 1995.

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

[9] J. Friedman, "Multivariate Adaptive Regression Splines," *The Annals of Statistics*, vol. 19, pp. 1–67, 1991.

[10] R. Collobert and S. Bengio, "SVMTorch: Support Vector Machines for Large-Scale Regression Problems," *JMLR*, vol. 1, pp. 143–160, 2001.

[11] S. Keerthi and D. DeCoste, "A Modified Finite Newton Method for Fast Solution of Large Scale Linear SVMs," *J. Mach. Learn. Res.*, vol. 6, pp. 341–361, 2005.

[12] L. Bo, L. Wang, and L. Jiao, "Recursive Finite Newton Algorithm for Support Vector Regression in the Primal," *Neural Comput.*, vol. 19, pp. 1082–1096, 2007.

[13] T. Hazan, A. Man, and A. Shashua, "A Parallel Decomposition Solver for SVM: Distributed dual ascend using Fenchel Duality," in *CVPR'08*, 2008, pp. 1 –8.

[14] Y. Lu, V. P. Roychowdhury, and L. Vandenberghe, "Distributed Parallel Support Vector Machines in Strongly Connected Networks," *IEEE Transactions on Neural Networks*, vol. 19, no. 7, pp. 1167–1178, 2008.

[15] P. Forero, A. Cano, and G. Giannakis, "Consensus-Based Distributed Support Vector Machines," *J. Mach. Learn. Res.*, vol. 99, pp. 1663–1707, 2010.