Final Report: Parallelizing Regularized Least Squares

Manuel A. Rivas

May 16, 2008

This project was focused on parallelizing regularized least squares - a class of statistical learning algorithms. The aim of the project at the beginning of the course and in the interim report made available at `http://beowulf.csail.mit.edu/~turing` was to parallelize a quadratic programming setup of the type presented in Support Vector Machine and elastic net regularization algorithms along with regression type setups presented in regularized least squares algorithm. By the middle of the term we succeeded in generalizing support vector machines to be handled in Star-P but failed miserably in making an implementation available in the Sicortex machines with any of the algorithms I intended to study. However, I realize that the theoretical properties of Regularized Least Squares gave it a really beautiful setup for parallelization. Something that has not been handled in the literature but that is empirically evaluated with the high level software platform Star-P and the 12 node beowulf cluster from MIT. We observe a 3.6X speedup in the 12 processor beowulf cluster machine using Star-P described in the Performance section of this report.

1 Introduction

Learning from examples is not to memorize but to generalize the setting of our problem. By generalizing our problem we can predict to our advantage the outcome.
Given a set of examples or data $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ our goal is to find a function f such that f is a good prediction of y for any future input of x, i.e. $f(x) = \hat{y}$. We might think that fitting the data would be able to take care of this since it is common to do so. However, for problems that are a bit more complex in the sense that they do not follow the models that one thinks of ahead of time (e.g. linear model to fit y on x) it is not very practical or practical at all. It is important to learn some learning theory since it is quite difficult to carry out applications without any background in the material and to just think of it like a black box.
Here in this page we take advantage of some of the lessons that we have learned while at MIT in the course 9.520 Statistical Learning Theory taught by Prof. Tomaso Poggio, Ryan Rifkin, Jake Bouvrie, and Lorenzo Rosasco . Therefore, some material will be very similar to that outlined in the course. In this report we describe the implementation of Regularized Least Squares several ideas we take into account in the parallelization of the algorithm and then present some plans for future work.
A couple of things to keep in mind. When we refer to **Generalization** we usually mean predicting the function well. Can we estimate the function whereas we can use the empirical or validation error to give us a sense of the prediction error for future use.

1.1 The learning problem

There is an unknown **probability distribution** on the product space $Z = X \times Y$, written $\mu(z) = \mu(x, y)$. We assume that X i s a compact domain in Euclidean space and Y a closed subset of $\mathbf{R}$.

The **training set** $S = \{(x_1, y_1), \ldots, (x_n, y_n)\} = \{z_1, \ldots, z_n\}$ consists of n samples drawn **i.i.d** from μ.

$\mathcal{H}$ is the hypothesis space, a space of functions $f : X \longrightarrow Y$.

A **learning algorithm** is a map $L : Z^n \longrightarrow \mathcal{H}$ that looks at S and selects from $\mathcal{H}$ a function $f_S : x \longrightarrow y$ such that $f_s(x) \approx y$ in a predictive way.

2 Regularized Least Squares

2.1 Reproducing Kernel Hilbert Space

For the purpose of this project a Hilbert space is a possibly infinite dimensional linear space endowed with a dot product. A norm is a nonnegative function $||\cdot||$ such that $\forall f, g \in \mathcal{F}$ (a function space whose elements are functions f) and $\alpha \in \mathbb{R}$. A linear evaluation functional over the Hilbert space of functions $\mathcal{H}$ is a linear function $\mathcal{F}_t : \mathcal{H} \to \mathbb{R}$ that evaluates each function in the space at the point t. A Hilbert space is a Reproducing Kernel Hilbert Space if the evaluation functionals are bounded. IF $\mathcal{H}$ is Reproducing Kernel Hilbert Space, then for erach $t \in X$ there exists, by the Riesz Representation Theorem a function K_t of $\mathcal{H}$ (the representer) with the reproducing property. The reproducing kernel of $\mathcal{H}$ is $K(t, x) = \langle K_t, K_x \rangle_{\mathcal{H}}$. It follows that for every RKHS the reproducing kernel is a positive definite kernel.

2.2 The Regularization Setting

We are given a Reproducing Kernel Hilbert Space $\mathcal{H}$ with a positive semidefinite kernel function k of which it can be a number of different kernel functions.

1. linear: $k(X_i, X_j) = X_i^t X_j$
2. polynomial: $k(X_i, X_j) = (X_i^t X_j + 1)^d$
3. gaussian: $k(X_i, X_j) = exp\left(-\frac{||X_i - X_j||^2}{\sigma^2}\right)$

We define the kernel matrix K to satisfy $K_{ij} = k(X_i, X_j)$. Recall, our goal is to find the function $f \in \mathcal{H}$ that minimizes the weighted sum of the total square loss and the Reproducing Kernel Hilbert Space Norm

$$\min_{f \in \mathcal{H}} \frac{1}{2} \sum_{i=1}^{n} (f(X_i) - Y_i)^2 + \frac{\lambda}{2} ||f||_K^2 \tag{1}$$

The loss function we use in this scenario is $V(x) = (f(x) - y)^2$ which makes sense for regression, but can still be used for classification where it makes no sense but still works.

2.3 Representer Theorem

The representer theorem guarantees that the solution to (1) can be written as

$$f(\cdot) = \sum_{i=1}^{n} c_i k(X_i, \cdot)$$

for some $c \in \mathbb{R}^n$. Hence, minimizing over the possibly infinite dimensional Hilbert space boils down to minimizing over $\mathbb{R}^n$. We can therefore rewrite (1) as

$$\min_{c \in \mathbb{R}^{\ltimes}} \frac{1}{2}||Y - Kc||_2^2 + \frac{\lambda}{2}||f||_K^2.$$

If we consider a function of the form

$$f(\cdot) = \sum_{i=1}^{n} c_i k(X_i, \cdot),$$

for such a function,

$$\begin{aligned} ||f||_K^2 &= \langle f, f \rangle_K \\ &= \left\langle \sum_{i=1}^{n} c_i k(X_i, \cdot), \sum_{j=1}^{n} c_j k(X_j, \cdot) \right\rangle_K \\ &= \sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j k(X_i, X_j) \\ &= c^t K c \end{aligned}$$

Now,

$$\frac{1}{2}||Y - Kc||_2^2 + \frac{\lambda}{2} c^t K c$$

is convex in c. We can find its minimum by setting the gradient with respect to 0:

$$\begin{aligned} -K(Y - Kc) + \lambda K c &= 0 \\ (K + \lambda I)c &= Y \\ c &= (K + \lambda I)^{-1} Y \end{aligned}$$

where Y is our set of training values. The solution exists and is unique for any $\lambda > 0$. If we define $G(\lambda) = K + \lambda I$ (as in our code), the prediction at a new test point X_* is:

$$\begin{aligned} f(X_*) &= \sum c_i k\left(X_i, X_*\right) \\ &= k\left(X, X_*\right)^t c \\ &= Y^t G^{-1} k\left(X, X_*\right). \end{aligned}$$

We really just need to solve a single linaer system. The matrix $K + \lambda I$ is symmetric positive definite so Cholesky factorization is the appropriate algorithm to take care of this. In our implementation we set λ ahead of time. However, we can find $c(\lambda)$ for many λ's essentially free since from the eigendecomposition $K = Q \wedge Q^t$ where $\wedge$ is diagonal with $\wedge_{ii} \geq 0$ and $QQ^t = I$. Hence,

$$\begin{aligned} G &= K + \lambda I \\ &= Q \wedge Q^t + \lambda I \\ &= Q\left(\wedge + \lambda I\right) Q^t, \end{aligned}$$

$\implies G^{-1} = Q\left(\wedge + \lambda I\right)^{-1} Q^t$. This is $O(n^3)$ time to solve one dense linear system or to compute the eigendecomposition. However, given Q and $\wedge$ we can find $c(\lambda)$ in $O(n^2)$ time:

$$c(\lambda) = Q(\wedge + \lambda I)^{-1} Q^t \lambda,$$

and we note that $(\wedge + \lambda I)$ is diagonal which results in finding $c(\lambda)$ for many λ's essentially for free.
A very peculiar thing about the Representer Theorem is that thus far it has been shown to work great under the Regularization in L^2 norm. This type of setup is called Tikhonov Regularization. However, when you add the L^1 norm or use the L^1 norm by itself nothing seems to work.

3 Parallelization

Support Vector Machines are widely used for classification and regression and are well known for their good generalization performance. Their limitation is a fast increase in compute and memory requirements with the number of training vectors leaving many problems out of their reach. On the other hand, Regularized Least Squares computation is faster and have really nice properties as shown above. However, they also have the large memory requirement since you are solving a system of linear equations and basically end up taking inverse of a matrix. However, a Cholesky Factorization algorithm from SCALAPACK eases the constraint somewhat nicely and since Star-P models most of its parallelized matlab computation calls after SCALAPACK it seems quite the neat solution to our problem. A straightforward way of accelerating Regularized Square is by accelerating the kernel computation and the solution to the system of linear equation. The kernel computation can be easily parallelized and we attempt to parallelize only this step in sequential code made available. The Cholesky factorization is also easily parallelized, well not easily but it is taken care of by the Star-P software environment. It is extremely convenient to have the power to just add the *p at then of the variables that are being infected throughout your program and Star-P takes care of the rest. We compare the speed and acceleration of serial Regularized Least Squares with Parallel Regularized Least Squares.

4 Results

Figure 4 demonstrates the generalization abilitiy of our regression with regularization networks. We solve our regularized least squares problem with $(32, 64, 128, 256, 512, 1024, 2048)$ data points drawn *i.i.d.* from $f(x) = x^3$ with white noise.

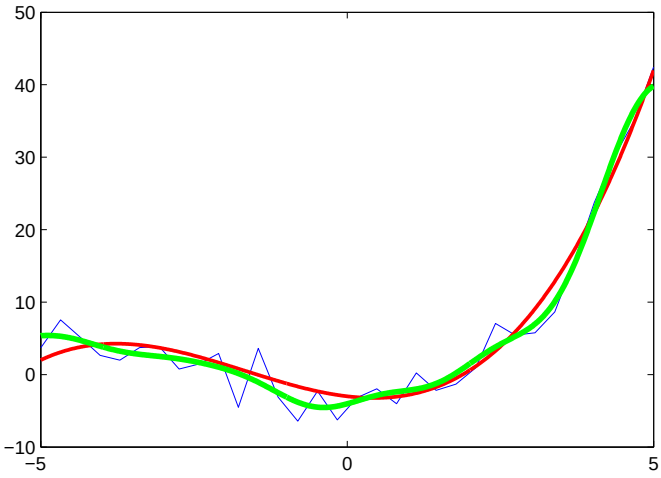
5 Performance

The following plot 5 demonstrates the overall performance gain from parallelizing our sequential MATLAB code in STAR-P with 12 processors. We intend to vary the number of processors to get a feeling for the type of speedup we experience by doing so (expect it to be a linear speedup). This example clearly demonstrates Keppner's model with Star-P.

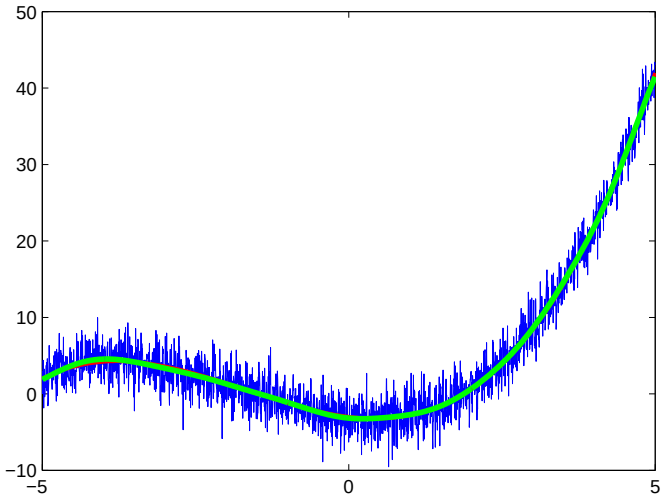
6 Future Work

6.1 Spread-Kernel SVMs

We intend to use the parallelization approaches outlined by Durdanovic, Cosatto, and Graf (2008) in an implementation of the parallelized algorithm Spread-Kernel SVM in the Sicortex machines.

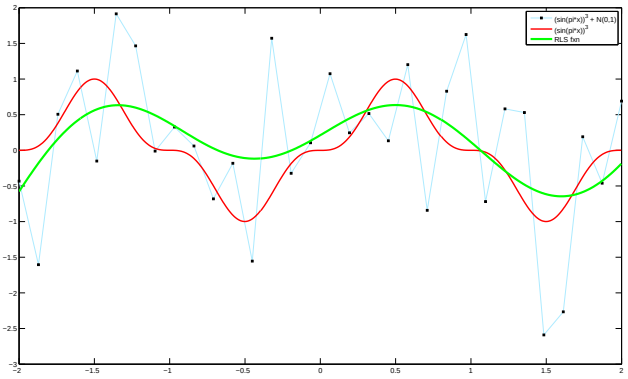


(a) 32 data points Regularized Least Squares Regression

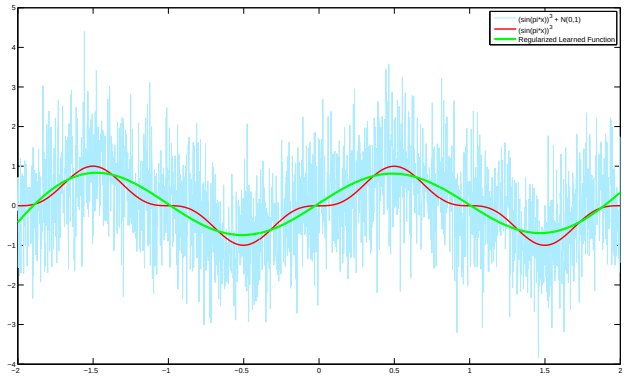


(b) 2048 data points Regularized

Figure 1: Regression Regularization Results for: $0.2x^3 + x^2 - x - 3$



(a) 32 data points Regularized Least Squares Regression



(b) 2048 data points Regularized Least Squares Regression

Figure 2: Regression Regularization Results for: $(sin(\pi x))^3$

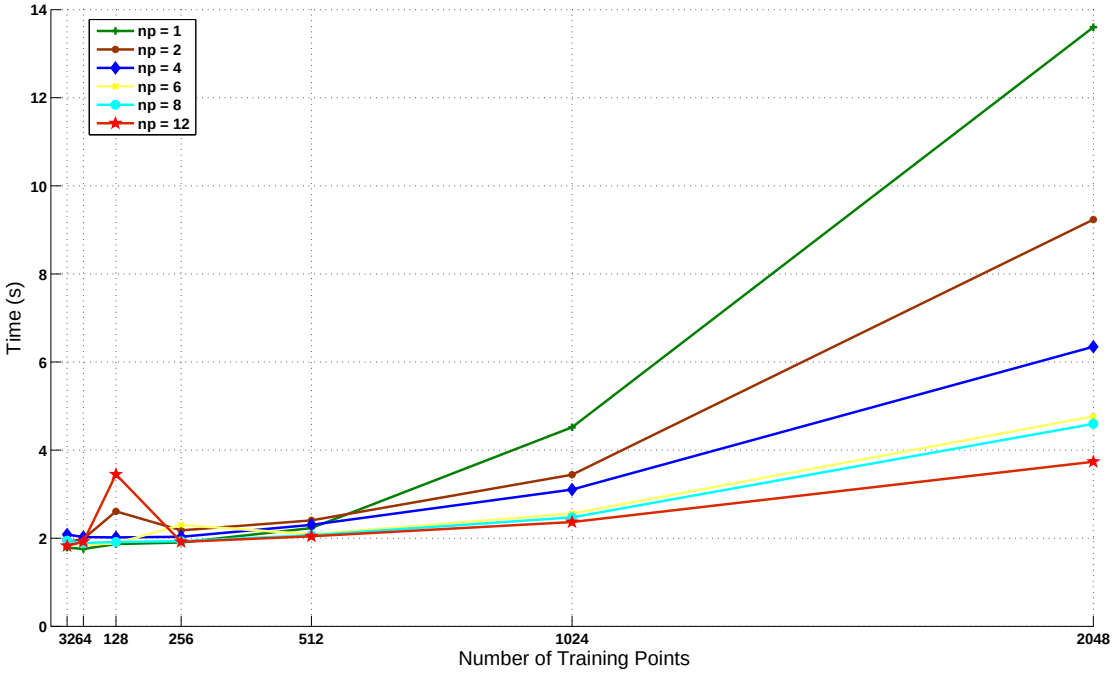


Figure 3: Time needed to execute an iteration of Regularized Least Squares with training points $= 32, 64, 128, 256, 512, 1024, 2048$.

As more machines or processors are introduced improving the caching efficiency dominates the improvement in time (speedup) and a strong super linear accelartion can be observed this is also taken into account in this project and our code.

6.2 Elastic Net Regularization

It seems the solution proposed by Durdanovic, Cosatto, and Graf extends to many quadratic programming type problems. Hence, I intend to implement a parallel version of elastic-net regularization algorithms when the SVM is complete and demonstrate performance.

7 Implementation

Code and documentation of the Regularized Least Squares algorithm is available at: `http://beowulf.csail.mit.edu/~turingene/s`

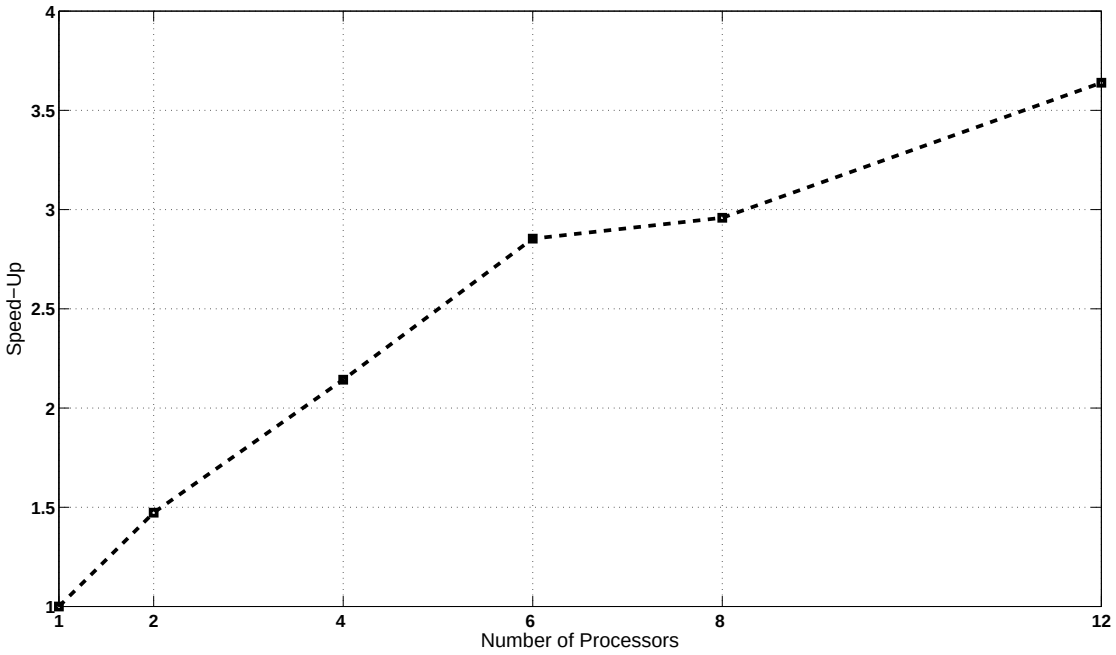


Figure 4: Speed gained with Parallel Regularized Least Squares (n = 1,. . . ,12 processors)