Large Scale Kernel Regression via Linear Programming

O. L. Mangasarian
Computer Sciences Dept.
University of Wisconsin
1210 West Dayton Street
Madison, WI 53706
olvi@cs.wisc.edu

David R. Musicant
Dept. of Mathematics and Computer Science
Carleton College
One North College Street
Northfield, MN 55057
dmusican@carleton.edu

Abstract

The problem of tolerant data fitting by a nonlinear surface, induced by a kernel-based support vector machine [24], is formulated as a linear program with fewer number of variables than that of other linear programming formulations [21]. A generalization of the linear programming chunking algorithm [2] for arbitrary kernels [13] is implemented for solving problems with very large datasets wherein chunking is performed on *both* data points and problem variables. The proposed approach tolerates a small error, which is adjusted parametrically, while fitting the given data. This leads to improved fitting of noisy data (over ordinary least error solutions) as demonstrated computationally. Comparative numerical results indicate an average time reduction as high as 26.0% over other formulations, with a maximal time reduction of 79.7%. Additionally, linear programs with as many as 16,000 data points and more than a billion nonzero matrix elements are solved.

Keywords: kernel regression, support vector machines, linear programming

1 Introduction

Tolerating a small error in fitting a given set of data, i.e. disregarding errors that fall within some positive ε, can improve testing set correctness over a

standard zero tolerance error fit [23]. Vapnik [24, Section 5.9] makes use of Huber's robust regression ideas [10] by utilizing a robust loss function [24, p 152] with an ϵ-insensitive zone (Figure 1 below) and setting up linear and quadratic programs for solving the problem. Schölkopf et al [18, 19] use quadratic-programming-based support vector machines to automatically generate an ε-insensitive "tube" around the data within which errors are discarded. In [21], Smola et al use a linear programming (LP) based support vector machine approach for ε-insensitive approximation. An LP formulation has a number of advantages over the quadratic programming (QP) based approach given by Vapnik [24], most notably sparsity of support vectors [1, 25, 22] and the ability to use more general kernel functions [13]. In this work we simplify the linear programming formulation of [21] by using a fewer number of variables. This simplification leads to considerably shorter computing times as well as a more efficient implementation of a generalization of the linear programming chunking algorithm of [2] for very large datasets. This generalization consists of chunking *both* data points and problem variables (row and column chunking) which allows us to solve linear programs with more than a billion nonzero matrix elements.

We briefly outline the contents of the paper. In Section 2 we derive and give the theoretical justification of our linear programming formulation (7) based on the loss function of Figure 1 and show (Lemma 2.1) that it is equivalent to the linear program (9) of [21] but with roughly 75% of the number of variables. Proposition 2.2 shows that the tolerated error interval length 2ε varies directly with the size of the parameter $\mu \in [0, 1]$ of our linear program (7), where $\mu = 0$ corresponds to a least 1-norm fit of the data (8) while the value $\mu = 1$ disregards the data and is therefore meaningless. It turns out, that depending on the amount of noise in the data, some positive value of $\mu < 1$ gives a best fit as indicated by the computational results summarized in Table 1. Section 3 implements and tests our linear programming formulation (7) and compares it with the linear programming formulation (9) of Smola et al [21]. Our formulation yielded an average computational time reduction as high as 26.0% on average, with a maximal time reduction of 79.7%. Our chunking implementation solved problems with as many as 16,000 observations. By contrast the largest dataset attempted in [21] contained 506 points.

To summarize, there are three main results in this paper. We present a kernel regression technique as a linear program which runs faster than previous techniques available. We introduce a simultaneous row-column chunking

algorithm to solve this linear program. Finally, we demonstrate that this row-column chunking algorithm can be used to significantly *scale up* the size of problem that a given machine can handle.

A word about our notation. All vectors will be column vectors unless transposed to a row vector by a prime superscript $'$. For a vector x in the d-dimensional real space R^d, the plus function x_+ is defined as $(x_+)_i = \max\{0, x_i\}$, $i = 1, \ldots, d$. The scalar (inner) product of two vectors x and y in the d-dimensional real space R^d will be denoted by $x'y$. For an $\ell \times d$ matrix A, A_i will denote the ith row of A. The identity matrix in a real space of arbitrary dimension will be denoted by I, while a column vector of ones of arbitrary dimension will be denoted by e. For $A \in R^{\ell \times d}$ and $B \in R^{d \times \ell}$, the **kernel** $K(A, B)$ maps $R^{\ell \times d} \times R^{d \times \ell}$ into $R^{\ell \times \ell}$. In particular if x and y are column vectors in R^d then, $K(x', A')$ is a row vector in R^ℓ, $K(x', y)$ is a real number and $K(A, A')$ is an $\ell \times \ell$ matrix. Note that $K(A, A')$ will be assumed to be symmetric, that is $K(A, A') = K(A, A')'$. The 1- 2- and ∞-norms will be denoted by $\|\cdot\|_1$, $\|\cdot\|_2$ and $\|\cdot\|_\infty$, respectively.

2 The Support Vector Regression Problem

We consider a given dataset of ℓ points in d dimensional real space R^d represented by the matrix $A \in R^{\ell \times d}$. Associated with each point A_i is a given observation, a real number y_i, $i = 1, \ldots, \ell$. We wish to approximate $y \in R^\ell$ by some linear or nonlinear function of the matrix A with linear parameters, such as the simple linear approximation:

$$Aw + be \approx y, \tag{1}$$

where $w \in R^d$ and $b \in R^1$ are parameters to be determined by minimizing some error criterion. If we consider w to be a linear combination of the rows of A, i.e. let $w = A'\alpha$, $\alpha \in R^\ell$, then we have:

$$AA'\alpha + be \approx y. \tag{2}$$

This immediately suggests the much more general idea of replacing the linear kernel AA' by some arbitrary nonlinear kernel $K(A, A') : R^{\ell \times d} \times R^{d \times \ell} \longrightarrow R^{\ell \times \ell}$ that leads to the following approximation, which may be nonlinear in A but linear in α:

$$K(A, A')\alpha + be \approx y. \tag{3}$$

We will measure the error in (3) by a vector $s \in R^\ell$ defined by:

$$-s \leq K(A, A')\alpha + be - y \leq s, \tag{4}$$

which we modify by tolerating a small fixed positive error ε, possibly to overlook errors in the observations y, as follows:

$$\begin{array}{rcccc} -s & \leq & K(A, A')\alpha + be - y & \leq & s \\ & & e\varepsilon & \leq & s \end{array} \tag{5}$$

We now drive the error s no lower than the fixed (for now) tolerance ε by minimizing the 1-norm of the error s together with the 1-norm of α for complexity reduction or stabilization. This leads to the following constrained optimization problem with positive parameter C:

$$\begin{array}{rcccc} & & \min\limits_{(\alpha,b,s)} \ \dfrac{1}{\ell}\|\alpha\|_1 + \dfrac{C}{\ell}\|s\|_1 & & \\ \text{s.t.} \quad -s & \leq & K(A, A')\alpha + be - y & \leq & s \\ & & e\varepsilon & \leq & s, \end{array} \tag{6}$$

which can be represented as a linear program. For a linear kernel and a *fixed* tolerance ε, this is essentially the model proposed in [23], which utilizes a 2-norm instead of a 1-norm. We now allow ε to be a nonnegative *variable* in the optimization problem above that will be driven to some positive tolerance error determined by the size of the parameter μ. By making use of linear programming perturbation theory [14] we parametrically maximize ε in the objective function with a positive parameter μ to obtain our basic linear programming formulation of the nonlinear support vector regression (SVR) problem:

$$\begin{array}{rcccc} & & \min\limits_{(\alpha,b,s,\varepsilon,a)} \ \dfrac{1}{\ell}e'a + \dfrac{C}{\ell}e's - C\mu\varepsilon & & \\ \text{s.t.} \quad -s & \leq & K(A, A')\alpha + be - y & \leq & s \\ 0 & \leq & e\varepsilon & \leq & s, \\ -a & \leq & \alpha & \leq & a. \end{array} \tag{7}$$

For $\mu = 0$ this problem is equivalent to the classical stabilized least 1-norm error minimization of:

$$\min_{(\alpha,b)} \ \|\alpha\|_1 + C\|K(A, A')\alpha + be - y\|_1. \tag{8}$$

For positive values of μ, the zero-tolerance ε of the stabilized least 1-norm problem is increased monotonically to $\varepsilon = \|y\|_\infty$ when $\mu = 1$. (See Proposition 2.2 below.) Components of the error vector s are not driven below this increasing value of the tolerated error ε. Thus, as μ increases from 0 to 1, ε increases correspondingly over the interval $[0, \|y\|_\infty]$. Correspondingly, an increasing number of error components $(K(A, A')\alpha + be - y)_i$, $i = 1, \ldots, \ell$, fall within the interval $[-\varepsilon, \varepsilon]$ until all of them do so when $\mu = 1$ and $\varepsilon = \|y\|_\infty$. (See Proposition 2.2 below.)

One might criticize this approach by pointing out that we have merely substituted one experimental parameter (ε in (6)) for another parameter (μ in (7)). This substitution, however, has been shown both theoretically and experimentally to provide tighter control on training errors [24, 19]. It has also been suggested that in many applications, formulation (7) may be more robust [19].

It turns out that our linear program (7) is equivalent to the one proposed by Smola et al [21]:

$$\begin{array}{lccc} \min\limits_{(\alpha^1,\alpha^2,b,\xi^1,\xi^2,\varepsilon)} & \frac{1}{\ell}e'(\alpha^1+\alpha^2) + \frac{C}{\ell}e'(\xi^1+\xi^2) + C(1-\mu)\varepsilon & & \\ \text{s.t.} \quad -\xi^2 - e\varepsilon & \leq & K(A,A')(\alpha^1-\alpha^2) + be - y & \leq \xi^1 + e\varepsilon \\ & & \alpha^1,\ \alpha^2,\ \xi^1,\ \xi^2,\ \varepsilon & \geq 0, \end{array} \tag{9}$$

with $4\ell+2$ variables compared to our $3\ell+2$ variables. We now show that the two problems are equivalent. Note first that at optimality of (9) we have that ${\alpha^1}'\alpha^2 = 0$. To see this, let $\alpha = \alpha^1 - \alpha^2$. Consider a particular component α_i at optimality. If $\alpha_i > 0$, then $\alpha_i^1 = \alpha_i$ and $\alpha_i^2 = 0$ since we minimize the sum of α^1 and α^2 in the objective function. By a similar argument, if $\alpha_i < 0$, then $\alpha_i^2 = -\alpha_i$ and $\alpha_i^1 = 0$. Hence for $\alpha = \alpha^1 - \alpha^2$ it follows that $|\alpha| = \alpha^1 + \alpha^2$. We thus have the following result.

Lemma 2.1 LP Equivalence *The linear programming formulations (7) and (9) are equivalent.*

Proof Define the deviation error:

$$d = K(A, A')\alpha + be - y. \tag{10}$$

By noting that at an optimal solution of (9), $\xi^1 = \max\{d - e\varepsilon, 0\} = (d - e\varepsilon)_+$ and $\xi^2 = \max\{-d - e\varepsilon, 0\} = (-d - e\varepsilon)_+$, the linear program (9) can be

written as:

$$\min_{(\alpha,b,\varepsilon,d)} \quad \frac{1}{\ell}\|\alpha\|_1 + \frac{C}{\ell}(e'(d - e\varepsilon)_+ + e'(-d - e\varepsilon)_+) + C(1-\mu)\varepsilon, \tag{11}$$

or equivalently

$$\min_{(\alpha,b,\varepsilon,d)} \quad \frac{1}{\ell}\|\alpha\|_1 + \frac{C}{\ell}e'(|d| - e\varepsilon)_+ + C(1-\mu)\varepsilon. \tag{12}$$

On the other hand, note that at optimality the variable s of the linear program (7) is given by:

$$s = \max\{|d|, e\varepsilon\} = e\varepsilon + \max\{|d| - e\varepsilon, 0\} = e\varepsilon + (|d| - e\varepsilon)_+. \tag{13}$$

Thus the linear program (7) becomes, upon noting that $\frac{e'e}{\ell} = 1$:

$$\min_{(\alpha,b,\varepsilon,d)} \quad \frac{1}{\ell}\|\alpha\|_1 + \frac{C}{\ell}e'(|d| - e\varepsilon)_+ + C(1-\mu)\varepsilon, \tag{14}$$

which is identical to (12).□

Note that (14) is equivalent to:

$$\min_{(\alpha,b,\varepsilon,d)} \quad \frac{1}{\ell}\|\alpha\|_1 + \frac{C}{\ell}(\|(|d| - e\varepsilon)_+\|_1 + \ell(1-\mu)\varepsilon), \tag{15}$$

and Figure 1 depicts the loss function $(\|(|d| - e\varepsilon)_+\|_1 + \ell(1-\mu)\varepsilon)$ of (15) as a function of a one dimensional error d with $\ell = 1$. The loss function is being minimized by the equivalent linear programs (7) and (9). The linear program (7) generates a tolerance $\varepsilon \in [0, \|y\|_\infty]$ corresponding to the parameter $\mu \in [0, 1]$. Within the increasing-size interval $[-\varepsilon, \varepsilon]$, depicted in Figure 1, an increasing number of errors corresponding to the constraints of (7) fall, i.e.:

$$-\varepsilon \leq (K(A, A')\alpha + be - y)_i \leq \varepsilon, \quad i \in I \subset \{1, \ldots, \ell\}.$$

By using linear programming perturbation theory we can give a useful interpretation of the role of the parameter $(1-\mu)$ appearing in the formulations above. We note first that the parameter μ lies in the interval $[0, 1]$. This is so because for $\mu > 1$ the objective function of the linear program (9) is unbounded below (let ε go to ∞), while for negative μ it is evident from the linear program (7) that $\varepsilon = 0$ and we revert to the stabilized least one norm formulation (8).

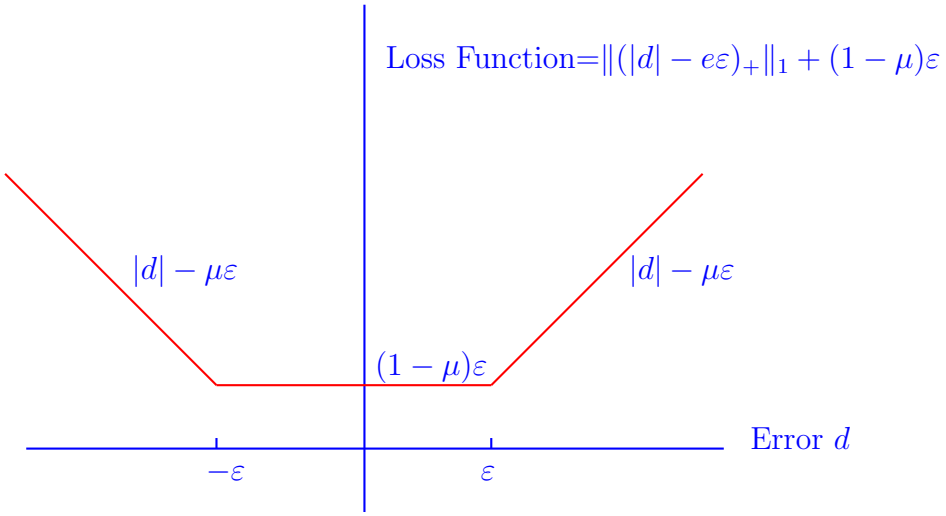

Figure 1: One Dimensional Loss Function Minimized

Proposition 2.2 Perturbation parameter interpretation

(i) For $\mu = 0$ the linear program (7) is equivalent to the classical stabilized least 1-norm approximation problem (8).

(ii) For all $\mu \in [0, \bar{\mu}]$ for some fixed $\bar{\mu} \in (0, 1]$, we obtain a fixed solution of the stabilized least 1-norm approximation (8) as in (i) with the ADDITIONAL *property of maximizing ε, the least error component. Hence ε is fixed over the interval $\mu \in [0, \bar{\mu}]$.*

(iii) For all $\mu \in [\hat{\mu}, 1]$ for some fixed $\hat{\mu}$ in [0,1), we obtain a fixed solution of

$$\min_{(\alpha,b,\varepsilon,d)} \quad \frac{1}{\ell}\|\alpha\|_1 + \frac{C}{\ell}e'(|d| - e\varepsilon)_+ \tag{16}$$

with the ADDITIONAL *property of minimizing ε, the least error component. Hence ε is fixed over the interval $\mu \in [\hat{\mu}, 1]$.*

(iv) For $\mu \in [0, 1]$ we get a monotonically nondecreasing least error component ε dependent on μ until the value $\mu = 1$ is reached whereat all error components have been trapped within the interval $[-\varepsilon, \varepsilon]$ with the trivial solution: $\alpha = 0$, $b = 0$, $\varepsilon = \|y\|_\infty$, $s = e\varepsilon$.

Proof

Figure 2: Visual representation of Proposition 2.2. The interval $[0, 1]$ is shown, with $\bar{\mu}$ and $\hat{\mu}$ indicated as well. The Roman numerals indicate which part of Proposition 2.2 is relevant.

(i) Because $a \geq 0$, $s \geq 0$ and ε is absent from the objective function of (7), ε can be set to zero and the resulting problem is that of minimizing the stabilized classical least 1-norm approximation (8).

(ii) We first note that the linear program (7) is solvable for all $\mu \in [0, 1]$ because its objective function is bounded below by zero on its nonempty feasible region as follows:

$$\begin{array}{rcl} \frac{1}{\ell}e'a + \frac{C}{\ell}e's - C\mu\varepsilon & = & \frac{1}{\ell}e'a + \frac{C}{\ell}e'(s - \mu e\varepsilon) \\ & \geq & \frac{1}{\ell}e'a + \frac{C}{\ell}e'(s - e\varepsilon) \geq 0. \end{array} \tag{17}$$

The desired result is then a direct consequence of [14, Theorem 1] which states that for all sufficiently small perturbations of a linear programming objective function by another possibly nonlinear function, a solution of the linear program is obtained that in addition optimizes the perturbation function as well. This translates here to obtaining a solution of the stabilized least 1-norm approximation problem (8) that also maximizes ε.

(iii) We observe that the linear program (7) is equivalent to the linear program (12), as shown in Lemma 2.1. We therefore obtain result (iii) by a similar application of linear programming perturbation theory as in (ii). Thus for $\mu = 1$ the linear program (12) is equivalent to (16) and for μ sufficiently close to 1, that is $\mu \in [\hat{\mu}, 1]$ for some $\hat{\mu} \in [0, 1)$, we get a solution of (16) that *also* minimizes ε.

(iv) Let $0 \leq \mu^1 < \mu^2 \leq 1$. Let the two points $(\alpha^1, b^1, s^1, \varepsilon^1, a^1)$ and $(\alpha^2, b^2, s^2, \varepsilon^2, a^2)$ be corresponding solutions of (7) for the values of μ of μ^1 and μ^2 respectively. Because both of the two points are feasible

for (7), without regard to the value of μ which appears in the objective function only, it follows by the optimality of each point that:

$$\frac{1}{\ell}e'a^1 + \frac{C}{\ell}e's^1 - C\mu^1\varepsilon^1 \leq \frac{1}{\ell}e'a^2 + \frac{C}{\ell}e's^2 - C\mu^1\varepsilon^2,$$

$$\frac{1}{\ell}e'a^2 + \frac{C}{\ell}e's^2 - C\mu^2\varepsilon^2 \leq \frac{1}{\ell}e'a^1 + \frac{C}{\ell}e's^1 - C\mu^2\varepsilon^1.$$

Adding these two inequalities gives:

$$C(\mu^2 - \mu^1)\varepsilon^1 \leq C(\mu^2 - \mu^1)\varepsilon^2.$$

Dividing the last inequality by the positive number $C(\mu^2 - \mu^1)$, gives $\varepsilon^1 \leq \varepsilon^2$.

To establish the last statement of (iv) we note from (17) that the objective function of (7) is nonnegative. Hence for $\mu = 1$, the point $(\alpha = a = 0,\ b = 0,\ \varepsilon = \|y\|_\infty,\ s = e\varepsilon)$ is feasible and renders the nonnegative objective function zero, and hence must be optimal. Thus $\varepsilon = \|y\|_\infty$. $\square$

We turn now to computational implementation of our SVR linear program (7) and its numerical testing.

3 Numerical Testing

We conducted two different kinds of experiments to illustrate the effectiveness of our formulation of the support vector regression problem. We first show the effectiveness of our linear programming formulation (7) when compared to the linear programming formulation (9) by using both methods on some large datasets. We then implement our method with a chunking methodology in order to demonstrate regression on massive datasets.

All experiments were run on Locop2, which is one of the machines associated with the University of Wisconsin Computer Sciences Department Data Mining Institute. Locop2 is a Dell PowerEdge 6300 server powered with four 400 MHz Pentium II Xeon processors, four gigabytes of memory, 36 gigabytes of disk space, and the Windows NT Server 4.0 operating system. All linear programs were solved with the state-of-the-art CPLEX solver [12].

We used the Gaussian radial basis kernel [24, 5] in our experiments, namely

$$K(A, A') = \exp(-\gamma \|A_i - A_j\|_2^2), i, j = 1, \ldots, \ell, \tag{18}$$

where γ is a small positive number specified in Table 1.

3.1 Comparison of Methods

Our linear programming formulation (7) for handling support vector regression will be referred to hereafter as the MM method. The experiments in this section compare MM to the linear programming formulation (9) given by Smola, Schölkopf, and Rätsch in [21], which will be referred to as the SSR method. We implemented the bookkeeping for both these methods in the MATLAB environment [15]. The actual linear programs were solved by the CPLEX [12] solver, using a homegrown MATLAB executable "mex" interface file [16] to CPLEX. This interface allows us to call CPLEX from MATLAB as if it were a native MATLAB function. In order to do a fair comparison, both the MM method and the SSR methods were implemented *without* the chunking ideas which we describe in Section 3.2.

Three datasets were used for testing the methods. The first dataset, Census, is a version of the US Census Bureau "Adult" dataset, which is publicly available from Silicon Graphics' website [3]. This dataset contains nearly 300,000 data points with 11 numeric attributes, and is used for predicting income levels based on census attributes. The second dataset, Comp-Activ, was obtained from the Delve website [8]. This dataset contains 8192 data points and 25 numeric attributes. We implemented the "cpuSmall prototask", which involves using twelve of these attributes to predict what fraction of a CPU's processing time is devoted to a specific mode ("user mode"). The third dataset, Boston Housing, is a fairly standard dataset used for testing regression problems. It contains 506 data points with 12 numeric attributes, and one binary categorical attribute. The goal is to determine median home values, based on various census attributes. This dataset is available at the UCI repository [17].

We present testing set results from tenfold cross-validation on the above datasets. We randomly partitioned each dataset into ten equal parts, and ran the algorithm ten times on 90% of this dataset. Each time, a different one of the ten segments was held out to serve as a test set. We used this test set to measure the generalization capability of the learning method by

measuring relative error. For an actual vector of values y and a predicted vector $\hat{y}$, the relative error as a percentage was determined as:

$$\frac{\|\hat{y} - y\|_2}{\|y\|_2} \times 100 \tag{19}$$

The test set errors shown in Table 1 are averages over tenfold cross-validation of this relative error.

For the Boston Housing dataset, all points in the dataset were used. For the Census and Comp-Activ datasets, a randomly selected subset of 2000 points was utilized for these experiments. The parameter C of the linear programs (7) and (9), and the parameter γ in the kernel (18) were chosen by experimentation to yield best performance. The parameter μ of (7) and (9) was varied as described below. We also observed in our experiments that tolerant training yields stronger improvements over standard least 1-norm fitting when there is a significant amount of noise in the training set. We therefore added Gaussian noise of mean zero and standard deviation σ to all training sets. Our parameter settings are shown in Table 1 below along with the results.

For each dataset, we began by setting $\mu = 0$ and doing the regression. This corresponds to doing a standard least 1-norm fit. Increasing μ for values near $\mu = 0$ typically increases fitting accuracy. Note that it would not make sense to start the experiments at $\mu = 1$, as Part (iv) of Proposition 2.2 demonstrates that the solution at $\mu = 1$ is trivial ($\varepsilon = 0$, $b = 0$).

We therefore started at $\mu = 0$ and raised μ at increments of 0.1 until we saw a degradation in testing set accuracy. The "total time" column in Table 1 reflects for a single fold the total time required to solve all problems we posed, from $\mu = 0$ up to our stopping point for μ. While it is true that varying both C and μ simultaneously might produce even more accurate results than those that we show, such experiments would be significantly more time consuming. Moreover, the experiments that we do perform demonstrate clearly that introducing ε into the regression problem can improve results over the classical stabilized least 1-norm problem. We applied the same experimental method to both the MM and the SSR methods, so as to make a fair comparison.

Table 1 summarizes the results for this group of numerical tests. We make the following points:

(i) The MM method is faster on all problems, by as much as 26.0% on average and 79.7% maximally, than the SSR method.

(ii) Test set error bottoms out at an intermediate positive value of the tolerated error ε. The size of the tolerated error ε is monotonically increasing with the parameter μ of the linear programs (7) and (9).

					μ									
Dataset	Parameters		Train size / Test size		0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	Total Time (sec)	Time Improvement
Census	C	100,000	2000	Test set error	5.10%	4.74%	4.52%	4.24%	4.16%	3.99%	3.79%	3.91%		Max
	γ	0.01	1000	ε	0.00	0.02	0.05	0.08	0.11	0.14	0.17	0.21		79.7%
	σ	0.2		SSR time (sec)	980	935	814	623	540	504	403	287	5086	Avg
				MM time (sec)	199	294	351	392	480	573	710	766	3765	26.0%
Comp-	C	100	2000	Test set error	6.60%	6.32%	6.16%	6.09%	6.01%	5.88%	5.60%	5.79%		Max
Activ	γ	0.01	200	ε	0.00	3.09	7.06	11.20	15.43	19.87	24.90	30.65		65.7%
	σ	30		SSR time (sec)	1364	1286	1015	1045	917	747	601	629	7604	Avg
				MM time (sec)	468	660	664	859	815	983	1043	1040	6533	14.1%
Boston	C	1,000,000	481	Test set error	14.69%	14.62%	14.23%	13.71%	13.59%	13.77%				Max
Housing	γ	0.0001	25	ε	0.00	0.42	1.37	2.31	3.23	4.20				52.0%
	σ	6		SSR time (sec)	36	34	28	25	24	23			170	Avg
				MM time (sec)	17	23	24	23	26	27			140	17.6%

C, μ: Parameters of the linear programs (7) and (9)

γ: Parameter of the Gaussian kernel (18)

σ: Standard deviation of introduced noise

ε: Variable in linear programs (7) and (9) which indicates tolerated error (see Figure 1). Its size is directly proportional to the parameter μ.

Empty columns under μ indicate experiments were not needed due to degradation of testing set accuracy.

Table 1: Tenfold cross-validation results for MM and SSR methods.

Dataset	$\bar{\mu}$	$1 - \hat{\mu}$
Census	0.051	5×10^{-7}
Comp-Activ	0.01	1×10^{-5}
Boston Housing	0.005	1×10^{-7}

Table 2: Experimental values for $\bar{\mu}$ and $\hat{\mu}$.

Finally, we ran some experiments to determine the values of $\bar{\mu}$ and $\hat{\mu}$ as in Proposition 2.2. We perturbed μ from 0 and from 1, and observed when the appropriate objective function value began to change. Table 2 shows the results for a single fold from each of the data sets under consideration. Note that Table 1 shows that the optimal values for μ are typically in the 0.4-0.6 range, and therefore are not included in either the intervals $[0, \bar{\mu}]$ or $[\hat{\mu}, 1]$. This yields the interesting result that the best solution to linear program (7) is a Pareto optimal solution that optimizes a weighted sum of the terms in the objective function. In other words, the solution is not subject to interpretations (ii) or (iii) of Proposition 2.2. Nevertheless, Proposition 2.2 gives a new interpretation of what kind of solution we get for values of μ close to 0 or close to 1, which in some instances may lead to the best testing set correctness.

3.2 Massive Datasets via Chunking

The Linear Programming Chunking method (LPC) has proven to be a useful approach in applying support vector machines to massive datasets by chunking the constraints that correspond to data points [2]. Other support vector machine research has utilized chunking as well [6]. We describe below how to chunk both the constraints and variables, which allows the solution of considerably larger problems than those considered previously. We first present a brief review of how the basic LPC is implemented. We then discuss the optimizations and improvements that we made.

The basic chunking approach is to select a subset of the constraints of the linear program (7) under consideration. When optimization on this first chunk of constraints is complete, a second chunk is created by combining all active constraints (those with positive Lagrange multipliers) from the first chunk and adding to it new unseen constraints. This process is repeated, looping a finite number of times through all constraints of the linear program until the objective function remains constant. This approach yields a non-

decreasing objective value that will eventually terminate at a solution of the original linear program [2].

To make our chunking implementation simpler, we apply the variable substitutions $s = t + e\varepsilon$ and $\alpha = \alpha^1 - \alpha^2, \alpha^1 \geq 0, \alpha^2 \geq 0$, which results in a linear program that is solved faster:

$$\begin{array}{cccccc} \min\limits_{(\alpha^1,\alpha^2,b,t,\varepsilon)} & & \frac{1}{\ell}e'(\alpha^1+\alpha^2) + \frac{C}{\ell}e't + C(1-\mu)\varepsilon & & & \\ \text{s.t.} \quad -t - e\varepsilon & \leq & K(A,A')(\alpha^1-\alpha^2) + be - y & \leq & t + e\varepsilon \\ & & \alpha^1,\ \alpha^2,\ t & \geq & 0, \end{array} \tag{20}$$

Formulation (20) looks similar to (9), but it has an important distinction. The two variables ξ^1 and ξ^2 have been replaced by the single variable t. This version of our problem is simpler for chunking than (7), since we do not need to be concerned with chunking the $e\varepsilon \leq s$ constraint.

One difference between our approach and the LPC in [2] is that each support vector has associated with it two constraints, namely the first two inequalities in (20). For a given support vector, only one of the two constraints associated with it has a nonzero multiplier (except possibly for a trivial case involving equalities). Our code tracks both constraints for a given support vector, and only retains from chunk to chunk those constraints with positive multipliers.

We make a further computational improvement involving upper bounds of the multipliers. This is motivated by looking at the dual problem of (20), which can be written as

$$\begin{array}{cccc} \max\limits_{(u,v)} & y'(u-v) & & \\ \text{s.t.} \quad K(u-v) & \leq & \frac{e}{\ell} \\ K(-u+v) & \leq & \frac{e}{\ell} \\ e'(u-v) & = & 0 \\ u+v & \leq & \frac{C}{\ell}e \\ e'(u+v) & \leq & C(1-\mu) \\ u,\ v & \geq & 0, \end{array} \tag{21}$$

where u and v are the multipliers associated with the constraints that we use for chunking. The constraint $u + v \leq \frac{C}{\ell}e$ indicates that these multipliers have an upper bound of $\frac{C}{\ell}e$. Experimentation has shown us that constraints with multipliers u_i or v_i at this upper bound have a strong likelihood of remaining at that upper bound in future chunks. This observation has been

made in the context of other support vector machine algorithms as well [4]. We therefore use this knowledge to our advantage. When optimization over a chunk of constraints is complete, we find all active constraints in that chunk that have multipliers at this upper bound. In the next chunk, we constrain the multiplier to be at this bound. This way the constraint remains active, but CPLEX does not have to spend resources determining this multiplier value. This frees up memory to add more new constraints to the chunk. This constraint on the multiplier is removed when the constraint selection procedure loops again through the entire constraint list, and tries to re-add this constraint to the problem. Our software actually represents the primal version of the linear program (20), as experimentation has shown that this solves faster than the dual. We therefore fix multipliers at this upper bound value by taking all associated constraints and adding them together to create a single collapsed constraint.

One difficulty in using the approach presented thus far is that as the number of data points increases, both the number of constraints and the number of variables increases. This is due to the kernel $K(A, A')$. The vectors α^1 and α^2, which are variables in the optimization problem, grow as data points are added to the problem. This means that the number of points that can fit in memory *decreases* as the problem size increases overall. This somewhat limits the success of chunking. A sufficiently large problem has so many variables that even a small number of support vectors cannot be represented in memory.

We therefore present a new approach for dealing with this issue: we chunk not only on the rows of the linear program, i.e. the constraints, but also on the columns of the linear program, i.e. the variables. We have observed that at optimality, most values of α^1 and α^2 are zero. For a given set of constraints, we choose a small subset of the α^1 and α^2 values to actually participate in the the linear program. All others are fixed at zero. This drastically reduces the size of the chunk, as the chunk width is no longer determined by the number of points in the entire problem, but merely by the number of nonzero α^1 and α^2. We solve this small problem, retain all α^1 and α^2 values which are nonzero, and proceed to chunk on another set of columns. For a fixed set of constraints, we loop repeatedly over the columns in the linear program until we terminate for that particular set of rows. Then another chunk of rows is selected as described above, and the process begins again. The column chunking is similar to the column generation method of Gilmore-Gomory and Dantzig-Wolfe [9, 7].

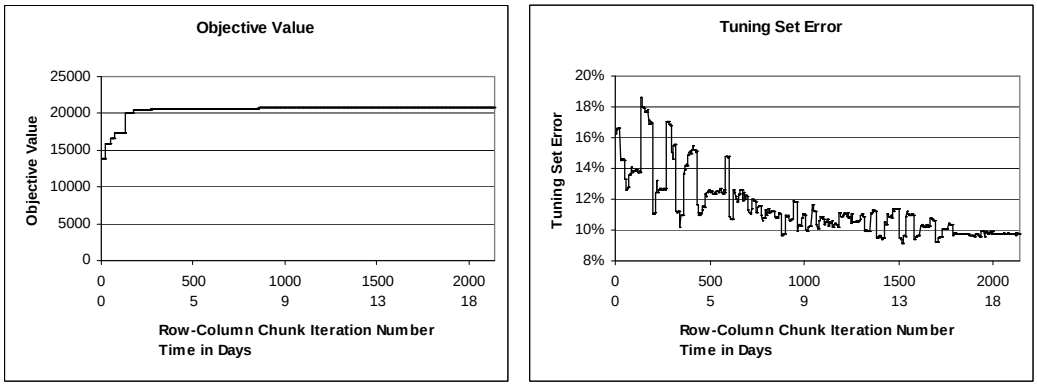

Figure 3: Objective function value and tuning set error versus row-column iteration and time in days.

We used this chunking approach on a subset of the Census dataset consisting of 16,000 data points. This results in a linear program with 32,000 rows and over 32,000 non-sparse columns, thereby containing over *one billion* nonzero values. Values for C and γ were chosen the same as shown in Table 1. We also chose $\sigma = 1$ and $\mu = 0.9$, and picked the initial chunk size to be 5000 rows and 2500 columns.

This massive problem was solved in 18.8 days, terminating with 1621 support vectors and 33 nonzero components of α^1 and α^2. Approximately 90 chunks, or 10 sweeps through the dataset, were required to reach termination. Figure 3 depicts the objective function value and tuning set error for each row-column chunk.

The final tuning set error achieved by this fit was 9.8% on a tuning set of 1000 points. We note that at termination of the first chunk, the tuning set error was at 16.2%. This indicates that a large sample of points can prove useful in getting better solutions.

Finally, we note that the techniques in this paper are certainly applicable to practitioners who use less powerful machines. On a smaller machine, a smaller chunk size can be used. Chunking will significantly increase the problem size that any machine can handle, when compared to the capabilities of such a machine without chunking.

4 Conclusion

A new formulation of a linear program to perform support vector regression can be solved considerably faster than other linear programming formulations of the same problem, while producing identical test set generalization capability. Row-column chunking is a new approach presented which has been shown to successfully handle massive nonlinear regression problems with more than a billion entries. Future work includes generalizing to loss functions other than that depicted in Figure 1. For example, we plan to consider the Huber M-estimator loss function [11] which is quadratic for small errors and linear for large errors by using quadratic programming. Extension to larger problems will also be considered using parallel processing for both linear and quadratic programming formulations.

Acknowledgements

The research described in this Data Mining Institute Report 99-02, August 1999, was supported by National Science Foundation Grants CCR-9729842 and CDA-9623632, by Air Force Office of Scientific Research Grant F49620-97-1-0326 and by the Microsoft Corporation.

References

[1] K. P. Bennett. Combining support vector and mathematical programming methods for induction. In Schölkopf et al. [20], pages 307–326.

[2] P. S. Bradley and O. L. Mangasarian. Massive data discrimination via linear support vector machines. *Optimization Methods and Software*, 13:1–10, 2000. ftp://ftp.cs.wisc.edu/math-prog/tech-reports/98-03.ps.

[3] US Census Bureau. Adult dataset. Publicly available from: www.sgi.com/Technology/mlc/db/.

[4] C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[5] V. Cherkassky and F. Mulier. *Learning from Data - Concepts, Theory and Methods*. John Wiley & Sons, New York, 1998.

[6] C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20:273–279, 1995.

[7] G. B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations Research*, 8:101–111, 1960.

[8] Delve. Data for evaluating learning in valid experiments. http://www.cs.utoronto.ca/∼delve/.

[9] P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting stock problem. *Operations Research*, 9:849–859, 1961.

[10] P. J. Huber. Robust estimation of location parameter. *Annals of Mathematical Statistics*, 35:73–101, 1964.

[11] P. J. Huber. *Robust Statistics*. John Wiley, New York, 1981.

[12] ILOG CPLEX Division, Incline Village, Nevada. *ILOG CPLEX 6.5 Reference Manual*, 1999.

[13] O. L. Mangasarian. Generalized support vector machines. In A. Smola, P. Bartlett, B. Schölkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 135–146, Cambridge, MA, 2000. MIT Press. ftp://ftp.cs.wisc.edu/math-prog/tech-reports/98-14.ps.

[14] O. L. Mangasarian and R. R. Meyer. Nonlinear perturbation of linear programs. *SIAM Journal on Control and Optimization*, 17(6):745–752, November 1979.

[15] MATLAB. *User's Guide*. The MathWorks, Inc., Natick, MA 01760, 1994-2001. http://www.mathworks.com/products/matlab/usersguide.shtml.

[16] MATLAB. *Application Program Interface Guide*. The MathWorks, Inc., Natick, MA 01760, 1997.

[17] P. M. Murphy and D. W. Aha. UCI repository of machine learning databases, 1992. www.ics.uci.edu/∼mlearn/MLRepository.html.

[18] B. Schölkopf, P. Bartlett, A. Smola, and R. Williamson. Support vector regression with automatic accuracy control. In L. Niklasson, M. Boden, and T. Ziemke, editors, *Proceedings of the 8th International Conference on Artificial Neural Networks*, pages 111–116, Berlin, 1998. Springer Verlag. Available at `http://www.kernel-machines.org/publications.html`.

[19] B. Schölkopf, P. Bartlett, A. Smola, and R. Williamson. Shrinking the tube: a new support vector regression algorithm. In M. S. Kearns, S. A. Solla, and D. A. Cohn, editors, *Advances in Neural Information Processing Systems 11*, pages 330–336, Cambridge, MA, 1999. MIT Press. Available at `http://www.kernel-machines.org/publications.html`.

[20] B. Schölkopf, C. Burges, and A. Smola (editors). *Advances in Kernel Methods: Support Vector Machines*. MIT Press, Cambridge, MA, 1999.

[21] A. Smola, B. Schölkopf, and G. Rätsch. Linear programs for automatic accuracy control in regression. In *Ninth International Conference on Artificial Neural Networks*, Conference Publications No. 470, pages 575–580, London, 1999. IEE. Available at `http://www.kernel-machines.org/publications.html`.

[22] A. J. Smola. *Learning with Kernels*. PhD thesis, Technische Universität Berlin, Berlin, Germany, 1998.

[23] W. N. Street and O. L. Mangasarian. Improved generalization via tolerant training. *Journal of Optimization Theory and Applications*,

96(2):259–279, February 1998. ftp://ftp.cs.wisc.edu/math-prog/tech-reports/95-11.ps.

[24] V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, New York, 1995.

[25] J. Weston, A. Gammerman, M. Stitson, V. Vapnik, V. Vovk, and C. Watkins. Support vector density estimation. In Schölkopf et al. [20], pages 293–306.