2004

Active Set Support Vector Regression

David R. Musicant
Carleton College

Alexander Feinberg

Active Set Support Vector Regression

David R. Musicant and Alexander Feinberg

***Abstract*— We present ASVR, a new active set strategy to solve a straightforward reformulation of the standard support vector regression problem. This new algorithm is based on the successful ASVM algorithm for classification problems, and consists of solving a finite number of linear equations with a typically large dimensionality equal to the number of points to be approximated. However, by making use of the Sherman-Morrison-Woodbury formula, a much smaller matrix of the order of the original input space is inverted at each step. The algorithm requires no specialized quadratic or linear programming code, but merely a linear equation solver which is publicly available. ASVR is extremely fast, produces comparable generalization error to other popular algorithms, and is available on the web for download.**

***Index Terms*— regression, active set, support vector.**

I. Introduction

SUPPORT vector regression (SVR) is a powerful technique for predictive data analysis [1], [2] with many applications to varied areas of study. For example, regression is used in biological contexts to model disease onset and intensity as influenced by various behaviors and environmental conditions [3]. Support vector regression has been used for diverse application areas such as drug discovery [4], civil engineering [5], and sunspot frequency prediction [6]. In this work, we present a new active set strategy to solve a straightforward reformulation of the standard support vector regression problem. This new algorithm, based on the successful ASVM algorithm for classification problems [7], consists of solving a finite number of linear equations with a typically large dimensionality equal to the number of points to be approximated. However, by making use of the Sherman-Morrison-Woodbury formula, a much smaller matrix of the order of the original input space is inverted at each step. The algorithm requires no specialized quadratic or linear programming code, but merely a linear equation solver which is publicly available.

Key to our approach are the following two changes to the standard linear SVR problem:

- Regularize the regression plane with respect to both orientation (w) and location relative to the origin (b). See (6) below. Such an approach has been successfully used in a number of classification methodologies, such as ASVM [7], LSVM [8], and SSVM [9].
- Minimize the regression error (ξ and $\hat{\xi}$) using the 2-norm squared instead of the conventional 1-norm. See (6). Such an approach is common in the context of traditional least squares regression [10].

These simple, but fundamental changes, lead to a considerably simpler dual problem with only nonnegativity constraints and a simple complementarity condition.

In Section II of the paper we begin with the standard SVR formulation and its dual and then give our formulation and its simpler dual. We corroborate with solid computational evidence that our simpler formulation does not compromise on generalization ability as evidenced by numerical tests in Section IV on public datasets. See Table I. Section III gives our active set support vector regression (ASVR) Algorithm 1 which consists of repeatedly solving systems of linear equations. By invoking the Sherman-Morrison-Woodbury (SMW) formula (1) we need only invert an $(n+1) \times (n+1)$ matrix where n is the dimensionality of the input space. This is a key feature of our approach that allows us to solve problems with millions of points by merely inverting much smaller matrices of the order of n. Section IV describes our numerical results which indicate that the ASVR formulation has a tenfold testing correctness that is as good as the ordinary SVR formulation, and has the capability of quickly and accurately solving massive problems with millions of points that are difficult to solve with standard SVR methods.

There is a considerable amount of previous literature related to our work. Numerous software packages available to the community do support vector regression [6], [11]–[15]. The Sherman-Morrison-Woodbury formula has been used by many support vector machine approaches, such as ASVM [7], LSVM [8], and PSVM [16], as well as in an interior point approach by Ferris and Munson [17]. Williams and Seeger make use of the SMW formula in applying the Nyström method to Gaussian process classification and regression [18]. In addition to our own previous work on active set methods [7], Burges has also used an active set method for the support vector classification problem [19]. We note that an *active set* computational strategy bears no relation to *active learning*. Evgeniou, Pontil, and Poggio [20] provide a general theory for considering different support vector machine formulations. Within this framework, our approach can be seen to contain elements of both standard support vector regression and regularization networks.

The support vector machine regression problem differs from the support vector machine classification problem in a few fundamental ways [2], [21]. The goal of the classification problem is to make a binary decision, i.e. to choose in which of two classes a given point should be classified. The goal of the regression problem, on the other hand, is to approximate a function. The solution to a support vector regression problem is a function that accepts a data point and returns a continuous

This work was supported by a grant from the Howard Hughes Medical Institute and by additional funding from Carleton College.

1045-9227/04$20.00 © 2004 IEEE

value. The support vector regression problem also allows for a "zone of insensitivity" defined typically by a parameter ε. Suykens and Vandewalle [22] have considered an approach to support vector classification by treating it as a regression problem, where the function to be approximated takes on the value 1 for one class and -1 for the other. As with our approach, Suykens and Vandewalle use a quadratic loss function.

We now describe our notation and provide some background material. All vectors will be column vectors unless transposed to a row vector by a prime $'$. For a vector $x \in R^n$, x_+ denotes the vector in R^n with all of its negative components set to zero. The notation $A \in R^{m\times n}$ will signify a real $m \times n$ matrix. For such a matrix A' will denote the transpose of A and A_i will denote the i-th row of A. A vector of ones or zeroes in a real space of arbitrary dimension will be denoted by e or 0, respectively. The identity matrix of arbitrary dimension will be denoted by I. We shall employ the MATLAB "dot" notation [23] to signify application of a function to all components of a matrix or a vector. For example if $A \in R^{m\times n}$, then $A_\bullet^2 \in R^{m\times n}$ will denote the matrix obtained by squaring each element of A. For two vectors x and y in R^n, $x \perp y$ denotes orthogonality, that is $x'y = 0$. For $u \in R^m$, $Q \in R^{m\times m}$ and $B \subset \{1, 2, \ldots, m\}$, u_B denotes $u_{i\in B}$, Q_B denotes $Q_{i\in B}$ and Q_{BB} denotes a principal submatrix of Q with rows $i \in B$ and columns $j \in B$. The notation $\operatorname{argmin}_{x\in S} f(x)$ denotes the set of minimizers in the set S of the real-valued function f defined on S. The 2-norm of a matrix Q will be denoted by $\|Q\|_2$. A special case of the Sherman-Morrison-Woodbury (SMW) formula [24] will be utilized:

$$(\frac{I}{C} + HH')^{-1} = C(I - H(\frac{I}{C} + H'H)^{-1}H'), \qquad (1)$$

where C is a positive number and H is an arbitrary $m \times k$ matrix. This formula enables us to invert a large $m \times m$ matrix by merely inverting a smaller $k \times k$ matrix.

II. Linear Support Vector Regression

We consider a given dataset of m points in n-dimensional real space, represented by the $m \times n$ matrix $A \in R^{m\times n}$. Associated with each point A_i is a given observation, a real number $y_i, i = 1, ..., m$. We wish to approximate $y \in R^m$ by a function of A of the form:

$$y \approx Aw + be, \qquad (2)$$

where $w \in R^n$ and $b \in R^1$ are parameters to be determined by minimizing some error criterion. For this problem the standard ε-loss insensitive support vector regression problem (SVR) with a linear kernel [1], [2] is given by the following quadratic program with parameters $C > 0, \varepsilon > 0$:

$$\begin{array}{ll} \min\limits_{w,b,\xi,\hat{\xi}} & \frac{1}{2}\|w\|_2^2 + C(e'\xi + e'\hat{\xi}) \\ \text{s.t.} & Aw + be - y \le \varepsilon e + \xi \\ & y - Aw - be \le \varepsilon e + \hat{\xi} \\ & \xi \ge 0, \hat{\xi} \ge 0 \end{array} \qquad (3)$$

The regression surface, which determines approximate values for y, is then given as:

$$\hat{y} = x'w + b \qquad (4)$$

The objective function in (3) contains two competing goals. The linear term $e'\xi + e'\hat{\xi}$ represents the error made in not fitting some of the data exactly. The quadratic term $\frac{1}{2}\|w\|_2^2$ is a regularization term, and is present to help avoid overfitting [2]. Since these two terms cannot typically both be simultaneously minimized, the parameter C indicates how much emphasis is to be placed on one goal versus the other.

The dual to the standard quadratic linear SVR (3) is the following [21]:

$$\begin{array}{ll} \min\limits_{\alpha,\hat{\alpha}} & \frac{1}{2}(\hat{\alpha} - \alpha)'AA'(\hat{\alpha} - \alpha) - y'(\hat{\alpha} - \alpha) + \varepsilon e'(\hat{\alpha} + \alpha) \\ \text{s.t.} & e'(\hat{\alpha} - \alpha) = 0 \\ & 0 \le \hat{\alpha}, \alpha \le C \end{array} \qquad (5)$$

w and b in the primal formulation can be expressed easily in terms of α and $\hat{\alpha}$. This dual formulation is the one most commonly used by currently available software [6], [15]. However, our alternative formulation to this problem is simpler, which leads to the ASVR algorithm which is able to solve the problem considerably faster. To arrive at our formulation, we make two modifications to (3). First, we change the error terms to be quadratic. This is a similar approach to that taken by traditional least-squares methods, and also has been used both in the context of support vector regression [21] and classification [22]. Second, we add b^2 to the objective as well. This approach has often been used successfully in classification problems [8], [16], [25], and results in a much more straightforward dual problem that can be solved by our ASVR algorithm. Our new formulation is thus given as:

$$\begin{array}{ll} \min\limits_{w,b,\xi,\hat{\xi}} & \frac{1}{2}\|w\|_2^2 + \frac{1}{2}C(\|\xi\|_2^2 + \|\hat{\xi}\|_2^2) + \frac{1}{2}b^2 \\ \text{s.t.} & Aw + be - y \le \varepsilon e + \xi \\ & y - Aw - be \le \varepsilon e + \hat{\xi} \end{array} \qquad (6)$$

Note that since the objective now contains quadratic loss terms for ξ and $\hat{\xi}$, the nonnegativity constraints on ξ and $\hat{\xi}$ are no longer necessary. To see this, suppose that a given component of the optimal values for ξ or $\hat{\xi}$ were negative. Setting this component to zero would result in a better (smaller) objective value, and would only increase the right hand side of the constraints in (6). Therefore none of the components of ξ or $\hat{\xi}$ can be negative at optimality, and the nonnegativity constraints are unnecessary.

Observe that this ASVR formulation differs noticeably from traditional least-squares regression. The regularization term $\frac{1}{2}\|w\|_2^2$ in the objective and the ε insensitivity term in the constraints capture the essence of the traditional support vector regression formulation.

The dual to (6) is:

$$\begin{array}{ll} \min\limits_{\alpha,\hat{\alpha}} & \frac{1}{2}(\hat{\alpha} - \alpha)'(AA' + ee')(\hat{\alpha} - \alpha) - y'(\hat{\alpha} - \alpha) \\ & \quad + \varepsilon e'(\hat{\alpha} + \alpha) + \frac{1}{2C}(\hat{\alpha}'\hat{\alpha} + \alpha'\alpha) \\ \text{s.t.} & \hat{\alpha}, \alpha \ge 0 \end{array} \qquad (7)$$

The dual formulation (7) is significantly more straightforward than (5) as it is missing both the upper bounds on the variables and the equality constraint. It is these simplifications that allow us to obtain the high speed and simplicity of the ASVR algorithm.

The variables (w, b) of the primal problem which determine the regression surface (4) can be obtained from the solution of the dual problem as:

$$w = A'(\hat{\alpha} - \alpha), \quad b = e'(\hat{\alpha} - \alpha) \tag{8}$$

The objective of formulation (7) can be simplified further by observing that $\hat{\alpha}'\alpha = 0$ at optimality. To verify this, suppose that for an optimal choice of $\hat{\alpha}$ and α there exists an index i such that $\hat{\alpha}_i > 0$ and $\alpha_i > 0$. Then define $\beta = min(\hat{\alpha}_i, \alpha_i)$, and redefine $\hat{\alpha}_i = \hat{\alpha}_i - \beta$ and $\alpha_i = \alpha_i - \beta$. All of the $(\hat{\alpha} - \alpha)$ terms in the objective of (7) remain unchanged, but the $(\hat{\alpha} + \alpha)$ and $(\hat{\alpha}'\hat{\alpha} + \alpha'\alpha)$ terms are reduced. This contradicts the supposition that $\hat{\alpha}$ and α are optimal. Therefore, we add to (7) the complementarity constraint $\hat{\alpha} \perp \alpha$. With this added constraint, we can safely add $-2\hat{\alpha}'\alpha$ to the last term of the objective in (7):

$$\begin{array}{ll} \min\limits_{\alpha,\hat{\alpha}} & \frac{1}{2}(\hat{\alpha} - \alpha)'(AA' + ee')(\hat{\alpha} - \alpha) - y'(\hat{\alpha} - \alpha) \\ & +\varepsilon e'(\hat{\alpha} + \alpha) + \frac{1}{2C}(\hat{\alpha}'\hat{\alpha} - 2\hat{\alpha}'\alpha + \alpha'\alpha) \\ \text{s.t.} & 0 \le \hat{\alpha} \perp \alpha \ge 0 \end{array} \tag{9}$$

This simplifies to:

$$\begin{array}{ll} \min\limits_{\alpha,\hat{\alpha}} & \frac{1}{2}(\hat{\alpha} - \alpha)'(AA' + ee' + \frac{I}{C})(\hat{\alpha} - \alpha) \\ & -y'(\hat{\alpha} - \alpha) + \varepsilon e'(\hat{\alpha} + \alpha) \\ \text{s.t.} & 0 \le \hat{\alpha} \perp \alpha \ge 0 \end{array} \tag{10}$$

We immediately note that the matrix $AA' + ee' + \frac{I}{C}$ appearing in the dual objective function is positive definite. This fact, combined with the facts that there is no equality constraint and no upper bound on the dual variables α and $\hat{\alpha}$, lead us to our simple finite active set algorithm which requires nothing more sophisticated than inverting an $(n+1) \times (n+1)$ matrix at each iteration in order to solve the dual problem (10).

III. ASVR (Active Set Support Vector Regression) Algorithm

The algorithm consists of partitioning the dual variables $\hat{\alpha}$ and α into nonbasic and basic variables. The nonbasic variables are those which are set to zero. The values of the basic variables are determined by finding the gradient of the objective function of (10) with respect to these variables, setting this gradient equal to zero, and solving the resulting linear equations for the basic variables. If any basic variable takes on a negative value after solving the linear equations, it is set to zero and becomes nonbasic. This is the essence of the algorithm.

With that in mind, we define a combined dual variable u as:

$$u = \hat{\alpha} - \alpha \tag{11}$$

At the beginning of each iteration of the ASVR algorithm, we consider as nonbasic all variables that are set equal to 0. We then consider the basic set as two subsets: one corresponding to the positive $\hat{\alpha}$ values, and one corresponding to the positive α values.

$$\begin{array}{ll} \hat{B} = \{i| & u_i > 0\} \\ B = \{i| & u_i < 0\} \\ N = \{i| & u_i = 0\} \end{array} \tag{12}$$

Finally, we define the entire basic set as those corresponding to both positive $\hat{\alpha}$ and α:

$$\mathcal{B} = \mathcal{B}(\hat{B}, B) = \hat{B} \cup B \tag{13}$$

The crux of the algorithm consists of optimizing only over those variables which are basic. We therefore make the following simplifying definitions:

$$\begin{array}{c} H = [A \quad e], \qquad M = \frac{I}{C} + HH', \\ q_{\mathcal{B},i} = \begin{cases} -y_i + \varepsilon e & \text{if } \ i \in \hat{B} \\ -y_i - \varepsilon e & \text{if } \ i \in B \\ 0 & \text{otherwise} \end{cases} \end{array} \tag{14}$$

With these definitions, we can rewrite (10), holding the nonbasic variables fixed at zero, as:

$$\begin{array}{c} \min\limits_{u_{\mathcal{B}}} \ f_{\mathcal{B}}(u_{\mathcal{B}}) = \frac{1}{2}u'_{\mathcal{B}}M_{\mathcal{B}\mathcal{B}}u_{\mathcal{B}} + q'_{\mathcal{B}}u_{\mathcal{B}} \\ \text{s.t.} \quad u_{\hat{B}} \ge 0, \ u_B \le 0 \end{array} \tag{15}$$

Note that the complementarity constraint in (10) has not been written explicitly in (15). This is because our choices for $\hat{B}$ and B always ensure that this condition holds true implicitly. More specifically, as seen in Algorithm 1:

- We initialize $\hat{B} = \{1, ..., m\}$ and $B = \emptyset$. This corresponds to assuming that $\alpha = 0$ initially, and hence the complementarity condition holds.
- At every "regular" step of the algorithm (see (A) and (B) in Algorithm 1) the cardinality of $\mathcal{B}$ is reduced by moving indices to the nonbasic set. Each regular ASVR step therefore preserves the complementarity condition.
- When progress can no longer be made via the regular ASVR approach, a gradient projection step is taken. This step does not preserve the complementarity condition, but complementarity can be immediately restored afterwards via a shift.

The basic ASVR approach is to find the global unconstrained minimum for the objective function of (15), and then to project the solution back onto the feasible region. This can be formalized as:

$$\begin{array}{c} \bar{u}_{\mathcal{B}} = -M^{-1}_{\mathcal{B}\mathcal{B}}q_{\mathcal{B}} \\ u_{\hat{B}} = (\bar{u}_{\hat{B}})_+ \\ u_B = (-\bar{u}_B)_+ \\ u_N = 0 \end{array} \tag{16}$$

Now that we have a new value for the variable u, we determine the new basic set and repeat.

In order to ensure that the algorithm converges and terminates, a few additional safeguards need to be put in place in order to allow us to invoke the Moré-Toraldo finite termination result [26]. These safeguards can be found in Algorithm 1. Specifically, if the "regular" ASVR step as described above does not make any progress towards the objective, we use a line search step. If the line search fails as well, we free all

nonbasic variables and do a gradient projection step which is guaranteed to improve the objective. In order to do so, we define the following notation:

$$\tilde{u} = \begin{bmatrix} \hat{\alpha} \\ \alpha \end{bmatrix}, \quad \tilde{q} = \begin{bmatrix} -y + \varepsilon e \\ y + \varepsilon e \end{bmatrix}$$
$$Q = \begin{bmatrix} M & -M \\ -M & M \end{bmatrix} \tag{17}$$

With these definitions the dual problem (10) becomes

$$\min_{0 \le \tilde{u} \in R^{2m}} f(\tilde{u}) = \frac{1}{2}\tilde{u}'Q\tilde{u} + \tilde{q}'\tilde{u} \quad \text{s.t.} \quad 0 \le \hat{\alpha} \perp \alpha \ge 0 \tag{18}$$

It is on this formulation of the problem that we use a gradient projection step. See Algorithm 1 for details. As noted earlier, this step does not necessarily preserve complementarity. We can restore complementarity, however, with a shift as follows. If there exists an $i \in \{1, ..., m\}$ such that $\tilde{u}_i > 0$ and $\tilde{u}_{i+m} > 0$ (i.e., such that $\hat{\alpha}_i > 0$ and $\alpha_i > 0$), we define $\beta = min(\tilde{u}_i, \tilde{u}_{i+m})$ and then redefine

$$\tilde{u}_i = \tilde{u}_i - \beta, \quad \tilde{u}_{i+m} = \tilde{u}_{i+m} - \beta \tag{19}$$

As described following (8), this process simultaneously restores the complementarity condition and improves the objective function.

Another key feature of the algorithm is a computational one that makes use of the SMW formula. This feature allows us to invert an $(n+1)\times(n+1)$ matrix at each step instead of a much bigger matrix of order $m \times m$. It should be understood that whenever Algorithm 1 refers to inverting a matrix, it should be done with the SMW formula and hence only an $(n+1)\times(n+1)$ matrix is inverted. (For simplicity we speak of "inverting" a matrix whereas in fact we are only solving a system of $n+1$ linear equations in $n+1$ unknowns which take less time to solve.)

Algorithm 1: ASVR (Active Set Support Vector Regression)

Initialize $\hat{B} = \mathcal{B} = \{1, ..., m\}$, $B = \{\}$.
Initialize $N = \{m+1, ..., 2m\}$.
Initialize done = false.
Initialize $u_{\hat{B}} = (-M_{\mathcal{B}\mathcal{B}}^{-1} q_{\mathcal{B}})_+$, $u_N = 0$.
Initialize $\tilde{u} = [u_+; \ (-u)_+]$.

While (not done)
  Set $\tilde{u}^{prev} = \tilde{u}$.
  Define $\hat{B} = \{i \mid u_i > 0\}$, $B = \{i \mid u_i < 0\}$.
  Define $N = \{i \mid u_i = 0\}$, $\mathcal{B} = \hat{B} \cup B$.

  // Try heuristic step (A).
  Set $\bar{u}_{\mathcal{B}} = -M_{\mathcal{B}\mathcal{B}}^{-1} q_{\mathcal{B}}$, $u_{\hat{B}} = (\bar{u}_{\hat{B}})_+$.
  Set $u_B = (-\bar{u}_B)_+$, $u_N = 0$.
  Set $\tilde{u} = [u_+; \ (-u)_+]$.

  If $f(\tilde{u}) \ge f(\tilde{u}^{prev})$, then

    // Try line search step (B).
    Set $\tilde{u} = \tilde{u}^{prev}$.
    Set $u = \tilde{u}_{\{1,...,m\}} - \tilde{u}_{\{m+1,...,2m\}}$.
    Set $u_{\mathcal{B}} = \arg min_{0 \le \lambda \le 1} \{ f_{\mathcal{B}}(u_{\mathcal{B}} + \lambda(\bar{u}_{\mathcal{B}} - u_{\mathcal{B}})) \mid u_{\hat{B}} + \lambda(\bar{u}_{\hat{B}} - u_{\hat{B}}) \ge 0 \ \ and \ \ u_B + \lambda(\bar{u}_B - u_B) \le 0 \}$.
    Set $u_N = 0$.
    Set $\tilde{u} = [u_+; \ (-u)_+]$.

    If $f(\tilde{u}) \ge f(\tilde{u}^{prev})$, then

      // Do gradient projection step, freeing up all
      // variables (C).
      Set $\tilde{u} = \tilde{u}^{prev}$.
      Repeat
        Set $\tilde{u} = \arg min_{0 \le \lambda \le 1} \ f(\tilde{u} - \lambda(\tilde{u} - (Q\tilde{u} + \tilde{q}))_+)$
      until $f(\tilde{u}) < f(\tilde{u}^{prev})$.

      // Adjust $\tilde{u}$ to preserve complementarity (D).
      For all $1 \le i \le m$ where $\tilde{u}_i > 0$ and $\tilde{u}_{i+m} > 0$
        Set $\beta = min(\tilde{u}_i, \tilde{u}_{i+m})$.
        Set $\tilde{u}_i = \tilde{u}_i - \beta$.
        Set $\tilde{u}_{i+m} = \tilde{u}_{i+m} - \beta$.
      End for

    End if

  End if

  // Test for global optimality (E).
  If $0 \le \tilde{u} \perp Q\tilde{u} + \tilde{q} \ge 0$ (within a tolerance)
    Set done = true.
  End if

End while

The heuristic step (A) in Algorithm 1, while not guaranteed to improve the objective, often works well. Since we use the SMW formula to invert the matrix $M_{\mathcal{B}\mathcal{B}}$, we only handle a system with $n+1$ columns where n is the number of features (columns) in the original dataset. The inversion thus takes a fixed amount of time at each iteration. Furthermore, we note that if the heuristic step were not used we would need to invert the matrix anyway for the line search step (B). Therefore, the time spent on inverting the matrix should really be thought of as "shared" between these two steps. The only other work done by the heuristic step (A) is in checking to see if it actually succeeded in improving the objective, which consists of evaluating the objective at the heuristic solution.

The line search step (B) is the traditional active set approach, designed to move one constraint at a time from the basic set to the nonbasic set. It is invoked when the heuristic in step (A) fails to improve the objective. Since step (B) can leverage the matrix inverse computed for step (A), the only additional work necessary is to solve an equation for a single variable. This, like the rest of step (A) apart from the inverse, is considerably faster than computing the inverse itself. However, the line search step (B) only eliminates one index at a time from the basic set. After this index has been removed from the basic set, another iteration and thus another inverse is required.

When the heuristic step (A) works well, on the other hand, it may remove a considerable number of indices from the basic set in one single heuristic step.

Finally, the gradient projection iteration (C) is guaranteed to converge to the global solution of (18) [27, pp 223-225] and is placed here to ensure that the strict inequality $f(\tilde{u}) < f(\tilde{u}^{prev})$ eventually holds as required in [26]. Similarly, the gradient projection step ensures that the function value does not increase when it remains on the same face, in compliance with [26, Algorithm BCQP(b)]. In our implementation we first do a variant of (C) where we do not yet free all the variables, in an attempt to make further progress on the face at hand. If no progress can be made, then we invoke step (C) as shown in Algorithm 1. Each individual gradient projection step (C) involves repeatedly solving an equation for a single variable until progress on the objective has been made. Each individual equation solution, therefore, takes approximately the same time as a single line search step (B). However, the number of iterations required in (C) may be many, and is not easily predictable. Step (C) also does not require the matrix inverse that was computed for steps (A) and (B). However, we only invoke this step after the other two have failed, so this particular advantage is not realized. Attempting step (C) before the other two could theoretically negate the need to compute an inverse in rare circumstances, but most of the time the number of iterations that step (C) needs to make is large enough that it makes sense to try the other approaches first.

The test for global optimality (E) is a direct application of the KKT optimality conditions [28] for (10) with the complementarity condition $0 \leq \hat{\alpha} \perp \alpha \geq 0$ relaxed. Note that this extra complementarity condition is initialized to be true, and is never violated throughout the algorithm except via the gradient projection step (C). If the gradient projection step does cause a violation of this condition, it is immediately corrected in the adjustment step (D).

ASVR is extended to solve regression problems with positive semidefinite nonlinear kernels in the same manner as LSVM. [8]. For $A \in R^{m \times n}$ and $B \in R^{n \times \ell}$, the kernel $K(A, B)$ maps $R^{m \times n} \times R^{n \times \ell}$ into $R^{m \times \ell}$. A typical kernel is the polynomial kernel $K(A, B) = (AB)_{\bullet}^{d}$, where d is an integer greater than or equal to 1. We therefore substitute a kernel in the definition for M in (14) to obtain

$$M = \frac{I}{C} + K(H, H') \tag{20}$$

Everything else in Algorithm 1 remains identical, apart from actually classifying points once α and $\hat{\alpha}$ have been determined. The parameters w and b no longer exist. Instead, we use an analogous approach to classification kernels [8] and predict the regression value associated with a particular point x to be

$$\hat{y} = K([x \quad 1], H')(\hat{\alpha} - \alpha) \tag{21}$$

The price paid for this extension is that large datasets can be handled with the efficiency of the linear case only if the inner product terms of the kernel [29, Equation (3)] are explicitly known, which in general they are not. Reduced kernel techniques [30], [31] can be beneficial here. Regardless, ASVR may be a useful tool for regression with nonlinear kernels because of its simplicity. ASVR does not require any specialized quadratic or linear programming code, but merely a linear equation solver which is publicly available.

IV. Numerical Implementation and Comparisons

We implemented ASVR in C++. The GNU g++-2.91.66 compiler under Red Hat Linux 6.2 [32] was used for all experiments, on a machine containing four 700 MHz Pentium III Xeon processors and a maximum of 2 gigabytes of memory available per process. We ensured that all experiments were isolated, i.e. no other users were using the machine at the same time. Our software does not make any use of parallel processing capabilities. We wrote all the code ourselves except for the linear equation solver, for which we used CLAPACK [33], [34].

Our stopping condition for ASVR comes directly from the test for global optimality (E) in Algorithm 1. Since this optimality condition really consists of three simultaneous checks ($\tilde{u} \geq 0$, $\tilde{u} \perp Q\tilde{u} + \tilde{q}$, $Q\tilde{u} + \tilde{q} \geq 0$), we need a convenient metric of how much this condition is violated for a particular choice of $\tilde{u}$. We therefore use the error bound residual $\|\tilde{u} - (\tilde{u} - Q\tilde{u} - \tilde{q})_+\|$. This residual is explained in more detail by Mangasarian and Ren [35], but can intuitively be seen to be zero when global optimality condition (E) is satisfied. We choose to terminate our algorithm when this error bound residual goes below 0.1. We also used a tolerance of 10^{-4} to determine at each iteration which points remained in the basic set. Any point with a dual variable less than this tolerance was moved to the nonbasic set.

The first set of experiments is designed to show that our reformulation of the SVR problem (6) and its associated algorithm ASVR yield similar performance to standard SVR techniques (3). For four publicly available datasets, we performed ten-fold cross-validation in order to compare test set errors between our formulation as implemented via ASVR, and the standard formulation as implemented via SVMTorch [6]. We initially tried comparing to mySVM [15] as well, but we quickly discovered that mySVM ran considerably more slowly than SVMTorch and so we abandoned the mySVM experiments apart from the smallest dataset. In all cases we normalized the data, as this generally seems to help the algorithms converge more quickly. We chose to use the default termination error criteria in SVMTorch of 1×10^{-2} and mySVM of 1×10^{-4}, since these were less stringent criteria for large datasets than the one we used for ASVR. This is because the criterion we used for ASVR is an aggregate over the errors for all points, whereas the SVMTorch and mySVM criteria reflect a minimum error threshold for each point. In all experiments, we measured generalization capability of the algorithm 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:

$$\frac{\|\hat{y} - y\|_2}{\|y\|_2} \times 100 \tag{22}$$

The training and test set errors that we report are averages over ten-fold cross-validation of this relative error. Likewise,

the CPU time and iteration counts that we present are ten-fold averages as well.

We also used ten-fold cross-validation with a hold-out tuning set in order to find optimal parameter settings for C and ε. For each of the ten folds for a given dataset, 10% of the data was removed as a test set and not used again until after optimal choices for C and ε were determined. From the other 90%, we removed 10% of that for use as a hold-out tuning set. We then tried a variety of choices for C and ε, training on our ten training sets and averaging our error rates on the ten tuning sets. We chose C and ε to maximize average performance on the tuning sets. C and ε were then fixed, and used to present the training and test results shown. It should be noted that we conducted this process independently for each algorithm. Since ASVR is based on a different formulation than the other methods, the optimal choices of C and ε are in general different than the optimal choices for the other algorithms. Likewise, since mySVM and SVMTorch both obtain approximate solutions using different algorithms, and since a range of C and ε values tend to give approximately the same "best" results, the experimental optimal values of these parameters are often different between mySVM and SVMTorch.

For each algorithm and for each dataset we optimized C and ε by starting with an exponential grid search, exhaustively trying all combinations of C from $\{10^{-6}, 10^{-5}, 10^{-4}, ..., 10^{6}\}$ and ε from $\{0.1, 1, 10\}$. Once optimal choices for C and ε were found by measuring average performance on hold-out tuning sets, we used the optimal combination of C and ε as a starting seed for the MATLAB [23] optimization tool *fminsearch*. This MATLAB function attempts to minimize a nonlinear multidimensional function by using a Nelder-Mead simplex method [36]. This only guarantees that we find a local minimum error, but finding a global minimum for C and ε is intractable given the complexity of the formulations. We note that the same optimization methodology was consistently used for all algorithms.

Four datasets were used for the first round of experiments. The first 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 [37]. The second dataset, Comp-Activ, was obtained from the Delve website [38]. This dataset contains 8192 data points and 25 numeric attributes. We implemented the "cpu prototask", which involves using 21 of these attributes to predict what fraction of a CPU's processing time is devoted to a specific mode ("user mode"). The third dataset, Census 30k, is a version of the US Census Bureau "Adult" dataset, which is publicly available from Silicon Graphics' website [39]. This "Adult" dataset contains nearly 300,000 data points with 11 numeric attributes, and is used for predicting income levels based on census attributes. We used a subset of 30,000 points to run these experiments. The fourth dataset, Census-House, was also obtained from the Delve website [38]. This dataset, also derived from US Census Bureau data, is used for predicting median home prices based on other demographic information. We used Census-House to evaluate ASVR on a dataset with considerably more features than the others. The original version of this dataset contained 137 features, although some of these features had constant values. We removed these features, leaving 121 of them remaining. We used a random subset of 5000 points so that we could run our experiments in a reasonable period of time.

The results from these experiments are shown in Table I. For the Housing dataset, ASVR generalizes similarly to SVMTorch and runs in comparable time. For the Comp-Activ dataset, ASVR generalizes better than SVMTorch but runs noticeably slower. We point out that with non-optimal parameter settings ASVR achieves generalization performance comparable with SVMTorch with running time faster than SVMTorch. Both sets of parameters for the Comp-Activ dataset are shown in Table I. On the Census 30k dataset, ASVR generalizes better than SVMTorch in faster time. Finally, on the Census-House dataset, we do see that ASVR runs considerably more slowly than SVMTorch. This is due to the relatively high dimensionality of this dataset.

Our second experiment shows that ASVR has the potential to perform well on massive datasets. We created an "easy-to-fit" synthetic dataset of 2 million points in ten dimensions by starting off with an exact linear fit, then adding Gaussian noise to the observations. The optimal error rate on this dataset was known, since we generated the dataset ourselves from a known regression function. ASVR achieved a close approximation to this error rate in about 60 seconds with a single iteration. SVMTorch, however, did not terminate on this simple dataset for a variety of parameter settings, even after running for several hours. For a simple but large dataset such as this one, the heuristics that ASVR uses are particularly well suited. The decomposition approach that SVMTorch uses, however, seems to have considerable difficulty.

We next examine how the running time for ASVR depends on the dimensionality of the dataset. Since we make use of the Sherman-Morrison-Woodbury identity repeatedly throughout the algorithm (step A, Algorithm 1), we expect that performance will slow significantly as dimensionality grows. We tested this in a straightforward manner by creating a series of synthetic datasets in the same manner described above, each with 1000 rows and varied numbers of features. As with the earlier experiments, we optimized C and ε via performance on a tuning set. The results for these experiments on our synthetic data are shown in Table II. As expected, and as confirmed further by the results seen in Table I, ASVR performs at its best on datasets with lower dimensionality. Likewise, ASVR performs quite well on datasets with large numbers of points. We additionally point out that use of the SMW identity is not required for our algorithm. If a dataset contains more columns than rows, it would make sense to avoid the SMW identity and invert the matrix directly.

Finally, we implemented a rough prototype in MATLAB [23] of ASVR with kernels using the techniques described at the end of Section III. Table III shows accuracy and running time for the Boston Housing dataset for both a linear and a quadratic kernel. C and ε were again optimized through the use of a hold-out tuning set. We have re-run the linear results

TABLE I
ASVR COMPARED WITH CONVENTIONAL SVMTORCH ON PUBLICLY AVAILABLE DATASETS.

Dataset	Algorithm	C	ε	Train Error	Test Error	CPU Time (secs)	Iters
Housing	SVMTorch	1	0.01	20.48%	19.80%	0.19	3840.7
(506 x 12)	ASVR	0.25	0.015	19.28%	19.44%	0.50	96.5
	mySVM	1.6	0.014	20.45%	19.77%	2.22	
Comp-Activ	SVMTorch	0.099	10	12.07%	11.97%	3.55	2905.9
(8192 x 21)	ASVR	0.004	0.1	11.76%	11.86%	2.41	18.9
	ASVR	1	0.1	11.15%	11.27%	31.9	329.5
Census 30k	SVMTorch	1000	0.1	27.77%	27.36%	5021.0	1271250.0
(30k x 10)	ASVR	0.017	0.19	25.16%	24.99%	4345.9	7748.3
Census-House	SVMTorch	1000	0.1	8.96%	9.12%	792.3	4422460.7
(5k x 121)	ASVR	0.1	0.1	8.03%	9.03%	3308.5	3095.4

ASVR error rates are comparable to SVMTorch in these examples. mySVM performance is shown for the first experiment as well (though mySVM did not output an iteration count). Iterations have significantly different meanings in each algorithm, and are provided here for reference only. Train error, test error, CPU time, and iteration counts are averages over ten-fold cross-validation. For the Comp-Activ dataset, we show ASVR twice, with different parameter settings. The first row shows comparable error rates to SVMTorch, with faster running time. The second row shows improved error rates, at the expense of running time. ASVR is thus seen in these examples to be comparable to or better than SVMTorch in running time and accuracy, with the exception of the Census-House high dimensional dataset shown at the bottom.

TABLE II
PERFORMANCE OF ASVR ON SYNTHETIC DATASETS WITH 1000 ROWS, WITH INCREASING DIMENSIONALITY.

number of features	C	ε	Train Error	Test Error	CPU Time (secs)	Iters
10	0.19	0	23.06%	23.70%	0.83	77.3
50	0.10	0.10	23.02%	24.68%	2.58	96.2
100	0.11	0.10	8.48%	9.75%	25.3	231.1
200	0.10	0.11	9.00%	11.63%	117.8	220.2
500	0.10	0.10	11.63%	27.90%	531.3	67.5

The use of the Sherman-Morrison-Woodbury identity in computing inverses is evident in the running time as the dimensionality increases. These results, and the ones in the previous table, suggest that ASVR is a superior algorithm on datasets with smaller dimensionality.

here using our MATLAB prototype code so that timing can be properly compared. We note that this prototype code does not use the SMW identity for computing inverses, is not optimized, and does not implement all heuristics in precisely the same detail as the earlier C++ code. Despite these caveats, these results suggest that ASVR can be successfully used with a nonlinear kernel.

V. CONCLUSION

We have demonstrated that an active set algorithm can be used effectively for the support vector regression problem. ASVR is fast, finite, simple, and capable of performing support vector regression on datasets with millions of points. ASVR requires nothing more complex than a commonly available linear equation solver for solving small systems with few variables even for massive datasets. Future work includes implementing kernel reduction techniques [30], [31] to allow ASVR to be used efficiently with nonlinear kernels.

ASVR can be downloaded from `http://www.mathcs.carleton.edu/faculty/dmusican/asvr`, and is free for research and academic purposes.

REFERENCES

[1] V. Cherkassky and F. Mulier, *Learning from Data - Concepts, Theory and Methods*. New York: John Wiley & Sons, 1998.
[2] V. N. Vapnik, *The Nature of Statistical Learning Theory*. New York: Springer, 1995.
[3] K. J. Rothman and S. Greenland, *Modern Epidemiology*. Lippincott Williams and Wilkins Publishers, 1998.
[4] A. Demiriz, K. Bennett, C. Breneman, and M. Embrechts, "Support vector machine regression in chemometrics," *Computing Science and Statistics*, 2001.
[5] Y. B. Dibike, S. Velickov, and D. Solomatine, "Support vector machines: Review and applications in civil engineering," in *AI Methods in Civil Engineering Applications*, O. Schleider and A. Zijderveld, Eds., Cottbus, 2000, pp. 45–58.

TABLE III
PERFORMANCE OF PROTOTYPE KERNELIZED ASVR ON BOSTON HOUSING DATA.

polynomial degree	C	ε	Train Error	Test Error	CPU Time (secs)	Iters
1	0.25	0.015	19.28%	19.44%	11.38	43.2
2	10	1	9.75%	15.60%	20.34	284.40

[6] R. Collobert and S. Bengio, "SVMTorch: Support vector machines for large-scale regression problems," *Journal of Machine Learning Research*, vol. 1, no. 1, pp. 143–160, February 2001. [Online]. Available: http://www.jmlr.org

[7] O. L. Mangasarian and D. R. Musicant, "Active set support vector machine classification," in *Advances in Neural Information Processing Systems 13*, T. K. Leen, T. G. Dietterich, and V. Tresp, Eds. MIT Press, 2001, pp. 577–583.

[8] ——, "Lagrangian support vector machines," *Journal of Machine Learning Research*, vol. 1, pp. 161–177, 2001. [Online]. Available: http://www.jmlr.org

[9] Y.-J. Lee and O. L. Mangasarian, "SSVM: A smooth support vector machine," *Computational Optimization and Applications*, vol. 20, pp. 5–22, 2001.

[10] N. R. Draper and H. Smith, *Applied Regression Analysis*, 3rd ed. New York, NY: John Wiley and Sons, 1998.

[11] C.-W. Hsu and C.-J. Lin, "BSVM Software," 2001. [Online]. Available: http://www.csie.ntu.edu.tw/~cjlin/bsvm/

[12] D. Bhattacharyya, "LEARNSC Software," 2001. [Online]. Available: http://www.support-vector.ws/html/downloads.html

[13] C.-C. Chang and C.-J. Lin, "LIBSVM Software," 2003. [Online]. Available: http://www.csie.ntu.edu.tw/~cjlin/libsvm/

[14] S. R. Gunn, "MATLAB SVM Toolbox," 1998. [Online]. Available: http://www.isis.ecs.soton.ac.uk/resources/svminfo/download.php

[15] S. Rüping, "mySVM," September 2001. [Online]. Available: http://www-ai.cs.uni-dortmund.de/SOFTWARE/MYSVM

[16] G. Fung and O. L. Mangasarian, "Proximal support vector machine classifiers," in *Proceedings KDD-2001: Knowledge Discovery and Data Mining, August 26-29, 2001, San Francisco, CA*, F. Provost and R. Srikant, Eds. New York: Association for Computing Machinery, 2001, pp. 77–86.

[17] M. C. Ferris and T. S. Munson, "Interior point methods for massive support vector machines," *SIAM Journal on Optimization*, vol. 13, pp. 783–804, 2003.

[18] C. K. I. Williams and M. Seeger, "Using the Nyström method to speed up kernel machines," in *Advances in Neural Information Processing Systems*, T. K. Leen, T. G. Dietterich, and V. Tresp, Eds., vol. 13. MIT Press, 2001, pp. 682–688.

[19] C. J. C. Burges, "A tutorial on support vector machines for pattern recognition," *Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 121–167, 1998.

[20] T. Evgeniou, M. Pontil, and T. Poggio, "Regularization networks and support vector machines," *Advances in Computational Mathematics*, vol. 13, 2000.

[21] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines*. Cambridge: Cambridge University Press, 2000.

[22] J. A. K. Suykens and J. Vandewalle, "Least squares support vector machine classifiers," *Neural Processing Letters*, vol. 9, no. 3, pp. 293–300, 1999.

[23] The MathWorks, Inc., *Using MATLAB*. Natick, MA: The MathWorks, Inc., 1984-2002.

[24] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed. Baltimore, Maryland: The Johns Hopkins University Press, 1996.

[25] D. R. Musicant, "ASVM Software: Active set support vector machine classification software," 2000. [Online]. Available: http://www.cs.wisc.edu/dmi/asvm

[26] J. J. Moré and G. Toraldo, "Algorithms for bound constrained quadratic programs," *Numerische Mathematik*, vol. 55, pp. 377–400, 1989.

[27] D. P. Bertsekas, *Nonlinear Programming*, 2nd ed. Belmont, MA: Athena Scientific, 1999.

[28] O. L. Mangasarian, *Nonlinear Programming*. Philadelphia, PA: SIAM, 1994.

[29] ——, "Generalized support vector machines," in *Advances in Large Margin Classifiers*, A. Smola, P. Bartlett, B. Schölkopf, and D. Schuurmans, Eds. Cambridge, MA: MIT Press, 2000, pp. 135–146.

[30] Y.-J. Lee and O. L. Mangasarian, "RSVM: Reduced support vector machines," in *Proceedings of the First SIAM International Conference on Data Mining*, Chicago, IL, April 2001.

[31] A. Smola and B. Schölkopf, "Sparse greedy matrix approximation for machine learning," in *Proceedings of the Seventeenth International Conference on Machine Learning*, P. Langley, Ed. Morgan Kaufmann, 2000.

[32] Red Hat Inc., "Red Hat Linux for Intel processors." [Online]. Available: http://www.redhat.com

[33] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK User's Guide*, 3rd ed. Philadelphia, Pennsylvania: SIAM, 1999. [Online]. Available: http://www.netlib.org/lapack

[34] CLAPACK, "f2c'ed version of LAPACK." [Online]. Available: http://www.netlib.org/clapack

[35] O. L. Mangasarian and J. Ren, "New improved error bounds for the linear complementarity problem," *Mathematical Programming*, vol. 66, pp. 241–255, 1994.

[36] J. C. Lagarias, J. A. Reeds, M. H. Wright, and P. E. Wright, "Convergence properties of the Nelder-Mead simplex method in low dimensions," *SIAM Journal of Optimization*, vol. 9, no. 1, pp. 112–147, 1988.

[37] P. M. Murphy and D. W. Aha, "UCI repository of machine learning databases," 1992. [Online]. Available: http://www.ics.uci.edu/~mlearn/MLRepository.html

[38] Delve, "Data for evaluating learning in valid experiments." [Online]. Available: http://www.cs.utoronto.ca/~delve

[39] United States Census Bureau, "Census dataset," http://reality.sgi.com/ronnyk.

Dave Musicant received B.S. degrees in both mathematics and physics from Michigan State University, then earned an M.A. degree in mathematics and an M.S. degree in computer sciences from the University of Wisconsin–Madison. He completed his Ph.D. degree in computer sciences at the University of Wisconsin–Madison in 2000, and is now an assistant professor of computer science at Carleton College. His research interests are in support vector machines and text mining.

Alex Feinberg received his B.A. degree in Computer Science from Carleton College in Northfield, Minnsota, in 2002. He is currently working as a programmer for SpeechGear, Inc., a technology startup also based in Northfield. His interests include machine learning, computer graphics, and collision detection. His web page is located at `http://www.steelwalrus.net`.