

# Iterative Inner Solvers for Revised Simplex SVM Training

Eric P. Astor<sup>a</sup>, Winnie J. Lung<sup>b</sup>,  
Ruben Ramirez-Padron, Christopher G. Sentelle, and Dr. Michael Georgiopoulos<sup>c</sup>.

<sup>a</sup> Mathematics & Physics, Swarthmore College, Swarthmore, PA.

<sup>b</sup> Electrical Engineering, Texas A&M University, College Station, TX.

<sup>c</sup> School of Electrical Engineering and Computer Science, University of Central Florida, Orlando, FL.

## ABSTRACT

Support Vector Machine (SVM) training is equivalent to solving a large constrained optimization problem. Much work has been spent on decompositional optimization methods for this problem, but non-decompositional approaches have only recently regained attention. Notably, Sentelle’s work in applying Rusin’s revised simplex method to SVM training demonstrated a significantly shorter training time for complex problems, but requires much more memory than the competing decompositional methods. We propose a revision to Sentelle’s approach, replacing his direct inner solver with an iterative solver: the preconditioned conjugate residual method. Results show an unexpected performance penalty, due to extreme ill-conditioning of the inner problems; avoiding this may require the creation of a specialized functional preconditioner.

**Keywords:** SVM, support vector machine, SVM training, active set, simplex, revised simplex, iterative, ill-conditioned

## 1. INTRODUCTION

Support Vector Machines (SVM’s) are, in their simplest form, a family of binary linear classifiers. Modified SVM’s can and have been used for multi-class and/or nonlinear classification, or even regression, but the underlying theory of their operation remains unchanged from that used for binary and linear SVM’s.

In training, an SVM receives a set of input patterns, each pre-designated as belonging to into one of two classes, and finds a hyperplane separating the patterns according to this classification. In particular, SVM’s are built to find a hyperplane with maximal margin, in the sense of maximizing the distance from the hyperplane to the nearest training data point. This produces the most robust linear classifier, in the sense that if test data exhibits isotropic and position-independent noise, a support vector machine will classify the data correctly at least as often as any other linear classifier. In other words, for most classification problems, an SVM is guaranteed to have the best generalization performance of any linear classifier.

The theory underlying this family of classifiers was first explored by Vapnik [1]. In the course of proving some other properties of the SVM classifier, he showed that SVM training is directly expressible as a quadratic programming problem. Furthermore, Vapnik demonstrated the convenient form of the Wolfe dual for this problem; he noted the “box” form of its inequality constraints, independently limiting the range of each variable, and its dependence on what he called the **support vectors** of the training data. This limited subset of the training data consisted of those data points in error or lying exactly on the margin of the classifying hyperplane. Thus, he not only allowed the application of the many existing quadratic programming techniques, but indicated that the problem as a whole was highly tractable in its dual form. Later, the application of the so-called “kernel trick” to SVM’s allowed their use for non-linear classification problems while retaining their maximal-margin guarantees.

---

Further author information: (Send correspondence to M.G.):

E.P.A.: eastor1@swarthmore.edu    W.J.L.: wlung@neo.tamu.edu

R.R-P.: rramirez@cs.ucf.edu    C.G.S.: csentelle@cfl.rr.com

M.G.: michaelg@mail.ucf.edu

Tel: +1-407-823-5338

The primary obstacle to the use of these classifiers is the large size of the training problem. Many quadratic programming techniques are rendered unusable for any usefully large training dataset. To date, most SVM training has been approached via decompositional methods such as SVM<sup>light</sup> [2] and SMO (Sequential Minimal Optimization) [3]. These methods break down the larger optimization problem into a series of smaller problems, solving these iteratively until the larger problem is solved to the desired precision. As these algorithms never optimize across the entire dataset at one time, they sidestep the large size of SVM training problems, exhibiting excellent speeds for many SVM training problems, but their convergence slows greatly for more complex datasets.

More recently, Shilton et al. [4], Scheinberg [5], and Sentelle [6], among many others, have proposed the use of active set quadratic programming methods to solve the SVM training problem, reducing the problem to one of finding the non-bound support vectors that fall on the classifier’s margin. All of these methods reduce an inequality-constrained optimization problem to a series of smaller equality-constrained optimizations or linear systems. Sentelle, in particular, suggests the use of Rusin’s revised simplex algorithm for quadratic programming [7], coupling its strong guarantees on matrix non-singularity with a modified Cholesky factorization in solving the inner problems of the method.

Sentelle’s algorithm exhibits excellent performance and convergence characteristics. By performing rank one updates on the Cholesky factorization of the inner problems, it successfully maintains  $\Theta(n_s^2)$  iteration time cost, where  $n_s$  is the size of the inner problem, equal to the number of non-bound support vectors (those actually on the classifier’s margin). However, in order to update this factorization, it must store it at all times, and thus has  $\Theta(n_s^2)$  spatial complexity. For complex classification problems,  $n_s$  may sometimes be extremely large, and thus render the problem intractable for reasons of memory consumption.

We studied the possibility of replacing Sentelle’s direct Cholesky-factorization solver with an iterative method for these inner problems. This would eliminate the need for significant additional storage, and could still maintain comparable performance to the original algorithm. We explored various iterative options, focusing primarily on the conjugate gradient and conjugate residual families of algorithms.

Testing demonstrated that the inner problems of the revised simplex algorithm rapidly became ill-conditioned as the candidate solution converged towards the optimal answer; condition numbers on the order of  $10^6$  and  $10^7$  were routinely observed, with the eigenvalue spectrum of these problems almost evenly distributed across the range. This ill-conditioning resulted in poor performance for iterative solvers. Experimentally, we observed a  $\Theta(n_s^3)$  time cost per revised simplex iteration, where we had expected only quadratic complexity.

Standard methods do exist to improve the conditioning in cases such as these, making problems more easily solvable by iterative methods. In fact, a specialized preconditioner (designed for Karush-Kuhn-Tucker problems) improved the condition number by three orders of magnitude. Even so, this preconditioner failed to produce significant improvements in the speed of our algorithm. The particular patterns in which it failed, however, indicate a promising direction for future work, most likely in further functional specializations to the preconditioner.

In this paper, we discuss the construction of an iterative approach to the inner problems found in revised simplex SVM training, detailed as background in our Section 2. In Section 3, we record the particular techniques we used, and lastly, in Sections 4 and 5, we present a detailed analysis of our observed results.

## 2. BACKGROUND

### 2.1 Support Vector Machines (SVM's)

The SVM binary classifier is a supervised learning machine that determines to which of two classes a particular pattern belongs. The training data consists of a set of  $n$  samples  $\{(x_i, y_i) | x_i \in \chi, y_i \in \{+1, -1\}, i = 1, \dots, n\}$ , where the set  $\chi$  denotes a domain of patterns with a finite dimension, and the  $y_i$  components denote the class to which the corresponding  $x_i$  pattern belongs. During training, the SVM classifier seeks a separating hyperplane dividing the patterns in one class from the other; in particular, it maximizes the margin, or distance from the hyperplane to the nearest data point. As Vapnik [1] shows, this can be expressed as the following optimization problem:

$$\begin{aligned} \min_{\mathbf{w}, \xi} \tau(\mathbf{w}, \xi) &= \frac{1}{2} \|\mathbf{w}\|^2 \\ \text{subject to:} & \\ \forall i : y_i (\langle \mathbf{w}, x_i \rangle + b) &\geq 1. \end{aligned}$$

The vector  $\mathbf{w}$  is orthogonal to the hyperplane, with length inversely proportional to the margin, while  $b$  is proportional to the distance from the origin to the hyperplane.

Maximizing the margin produces the most robust linear classifier, in the sense that if test data exhibits isotropic and position-independent noise, an SVM will classify the data correctly at least as often as any other linear classifier. In other words, for the most common sort of variation in the data, an SVM will have the best generalization performance out of all linear classifiers.

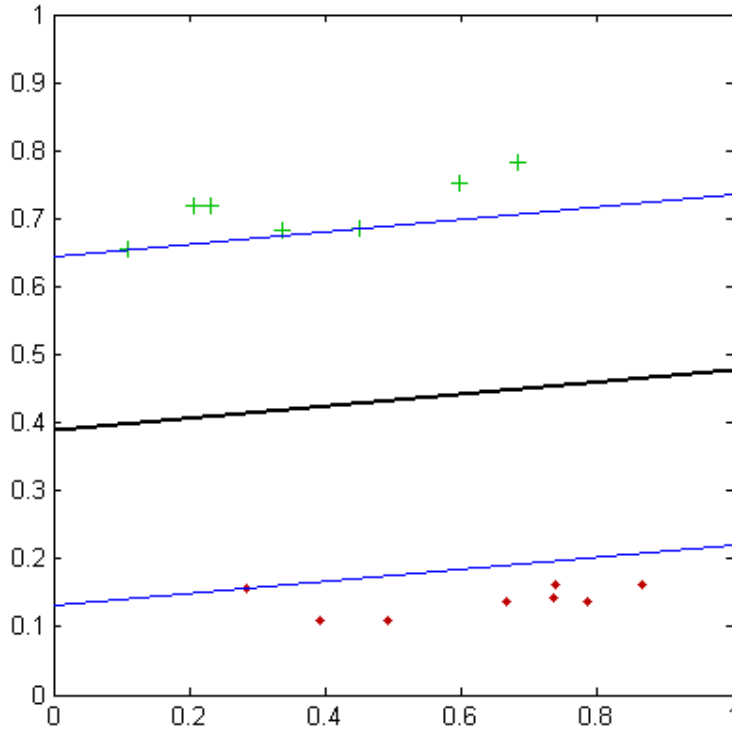


Figure 1. An example of a maximal-margin hyperplane separating two classes

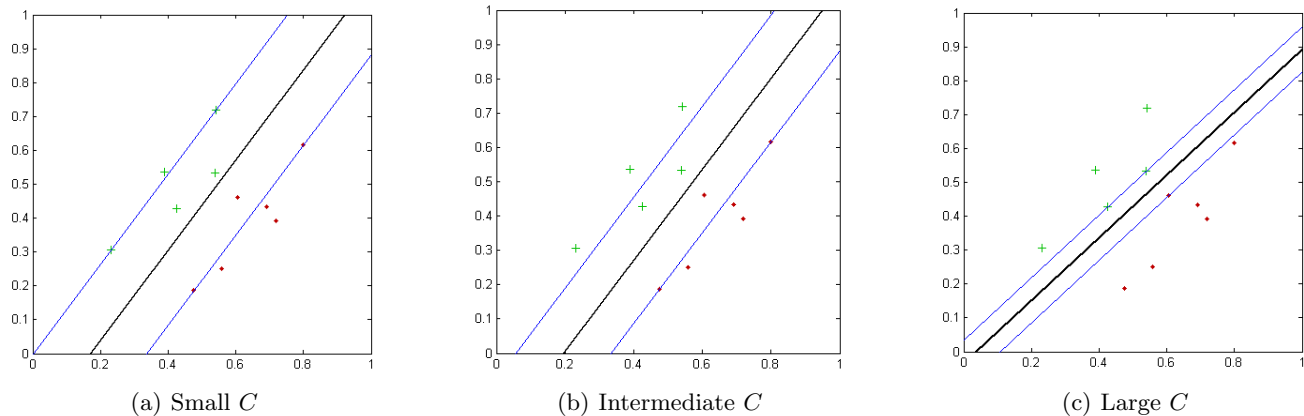


Figure 2. The effects that  $C$  has on classification.

Even if the data is not separable, the SVM can still find a maximal margin, minimal error separating hyperplane by introducing slack variables. This leads to the following primal optimization problem:

$$\begin{aligned} \min_{\mathbf{w}, \xi} \tau(\mathbf{w}, \xi) &= \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^n \xi_i \\ \text{subject to:} & \\ \forall i : y_i (\langle \mathbf{w}, \mathbf{x}_i \rangle + b) &\geq 1 - \xi_i, \\ \forall i : \xi_i &\geq 0. \end{aligned}$$

The constant  $C$  is fixed by the user and indicates the number of misclassified training patterns.  $C$  establishes a trade-off between margin maximization and the acceptance of patterns such that  $(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) < 1$ . The components of  $\xi$  are slack variables that enable non-linearly separable classification problems to be solved. The slack variables allow some data points to be on the wrong side of the margin or even be misclassified. For linearly separable problems, all the slack variables equal zero at the optimal value. This is illustrated in Figure 2(c). For small values of  $C$ , the solution focuses on minimizing  $\|\mathbf{w}\|$  and allows more misclassifications. An example of this is shown in Figure 2(a).

Using Lagrange multipliers, this primal optimization problem can be transformed into its Wolfe dual. This eliminates all parameters except for the Lagrange parameters ( $\alpha_i$ ).

$$\begin{aligned} \max_{\alpha} W(\alpha) &= \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle \\ \text{subject to:} & \\ \forall i : 0 &\leq \alpha_i \leq C, \\ \sum_{i=1}^n \alpha_i y_i &= 0. \end{aligned}$$

This dual problem can be rewritten in matrix form:

$$W(\alpha) = \alpha^T \mathbf{1} - \frac{1}{2} \alpha^T \mathbf{Q} \alpha, \quad (2.1)$$

where

$$Q_{ij} = y_i y_j \langle x_i, x_j \rangle.$$

As the reader may note, in this final problem and through each step of its derivation, our training data (the  $x_i$ 's) are never used directly. They only appear in the context of a dot product ( $\langle x_i, x_j \rangle$ ), serving as a sort of similarity measure. This suggests that we might be able to generalize from this treatment of SVM's, replacing our dot product  $\langle x_i, x_j \rangle$  with some other similarity measure  $k(x_i, x_j)$ . Several sources [8, 9] describe the necessary conditions on this new measure. For our purposes, it will suffice to restrict our attention to the most commonly-used families of kernels: the polynomial kernels, the sigmoidal kernels, and the radial basis function kernels.

The polynomial kernels share the form

$$k(x_i, x_j) = (\langle x_i, x_j \rangle)^r$$

or

$$k(x_i, x_j) = (\langle x_i, x_j \rangle + 1)^r,$$

where  $r$  is a positive integer. Using these kernels proves equivalent to pre-processing our data points, transforming into a "product space" [9] containing all the  $r$ -th order products of the elements of our vectors. This is particularly useful for fields such as vision, where correlations between features are more important than the features themselves. We note that this family includes the linear kernel  $k(x_i, x_j) = \langle x_i, x_j \rangle$ , and so includes ordinary linear SVM's.

The sigmoidal kernels

$$k(x_i, x_j) = \tanh(\kappa \langle x_i, x_j \rangle + \theta),$$

where  $\kappa$  is positive and  $\theta$  is negative, are primarily used for a convenient equivalence. An SVM using such a kernel behaves like a single-layer perceptron classifier, but retains its maximal-margin guarantees, thus arriving at a more error-tolerant result.

The Radial Basis Function (RBF) kernels are based on Gaussian functions, having the form

$$k(x_i, x_j) = \exp\left(-\gamma \|x_i - x_j\|^2\right).$$

These provide a clean way to define similarity in terms of the distance between two vectors, rather than the individual elements they have in common (as the dot product does). An SVM using an RBF kernel is then capable of separating classes which are clustered in their features, even if one class of data is distinguished only by being sufficiently close to a particular point. This situation presents a near-impossible task to linear SVM's; the separating surface between this class and another would be a sphere, which has no approximating hyperplane.

This "kernel trick" is a well-known way of generalizing SVM's to handle non-linear classification, allowing them to distinguish classes with no separating hyperplane. Conveniently, no matter what kernel is chosen, the underlying theory of SVM's remains correct in its original form. Thus, any method of training an ordinary SVM remains just as useful for these non-linear variants, including the revised simplex method described below.

## 2.2 Revised Simplex Method for SVM Training

The simplex method, as applied to quadratic programming (optimization with linear constraints and a quadratic objective function), is a simple approach to constrained optimization. Through a series of transformations, it takes any quadratic programming problem to the form

$$\begin{aligned} \min_{\mathbf{x}} \quad & \frac{1}{2} \mathbf{x}^T \mathbf{Q} \mathbf{x} + \mathbf{p}^T \mathbf{x} \\ \text{subject to:} \quad & \\ & \mathbf{B} \mathbf{x} = \mathbf{b}, \\ & \forall i : x_i \geq 0, \end{aligned}$$

and then performs a series of optimizing steps.

In each step, the algorithm divides the positivity constraints into the active and inactive sets; those constraints in the inactive set are ignored, while those in the active set are replaced by equality constraints ( $x_i \geq 0$  being replaced by  $x_i = 0$ ). In the context of SVM training, this is equivalent to choosing a possible set of support vectors ( $\alpha_i \neq 0$ ), and from those, choosing which Lagrange multipliers are actively bounded from above ( $\alpha_i = C$ ). The simplex algorithm then allows one bounded variable to move away from its bound, turning an active constraint into an inactive constraint. Thus, each step amounts to an optimization problem in which one variable is changed, while the others change only to maintain the constraint equations  $\mathbf{B} \mathbf{x} = \mathbf{b}$ . This becomes more complex if in changing, one of these other variables would become negative, but for our purposes, this discussion is sufficient.

Rusin’s revised simplex method for quadratic programming [7] conforms to this model, though it differs slightly from the traditional simplex algorithm as designed by Dantzig. These differences provide us with some significant computational advantages. Most usefully for our purposes, Rusin guarantees that the subproblems generated within each step are non-singular, exhibiting unique solutions.

This revised simplex method proves extremely useful for the SVM training problem, as shown by Sentelle [6]. When applied to SVM training, we find that all of our subproblems are of the form

$$\mathbf{A} \mathbf{x} = \mathbf{d},$$

where

$$\mathbf{A} = \begin{bmatrix} -\mathbf{Q}_{ss} & \mathbf{y}_s \\ \mathbf{y}_s^T & 0 \end{bmatrix}.$$

For the purposes of solving these subproblems, all we need to recognize is that  $\mathbf{Q}_{ss}$  is a submatrix of  $\mathbf{Q}$  (as used in Equation 2.1), restricted to the entries corresponding to the current non-bound support vector candidates, while  $\mathbf{y}_s$  is the vector of correct classifications for these data points. Since  $\mathbf{Q}$  is symmetric,  $\mathbf{Q}_{ss}$  is also symmetric, and so  $\mathbf{A}$  is symmetric in turn. From the guarantees of the revised simplex method, we can be sure that  $-\mathbf{Q}_{ss}$  is positive-semidefinite with at most one zero eigenvalue and that  $\mathbf{A}$  is nonsingular and indefinite with exactly one negative eigenvalue. Since  $\mathbf{Q}_{ss}$  is symmetric and nearly positive-definite, we may follow Sentelle [6] in solving these subproblems by means of a cleverly-applied Cholesky factorization. This can be updated cheaply (via a rank one update) after each change, avoiding the need to recompute the factorization from scratch at each step. This leaves us with an iterative algorithm for SVM training with iteration time  $\Theta(n_s^2)$ . However, storing the factorization requires  $\Theta(n_s^2)$  memory - which, for large and complex problems, can render actual computation infeasible.

### 2.3 Iterative Quadratic Optimizers

Sentelle's approach has an  $\Theta(n_s^2)$  spatial complexity associated with its use of rank-one updates to a Cholesky factorization. To reduce our memory consumption, we must avoid this technique altogether.

To avoid direct computation of the matrix inverse or a factorization, we might use an iterative solver on our subproblems. The structure of the subproblems for the revised simplex algorithm makes this particularly promising, being symmetric and non-singular; many of the best iterative algorithms require symmetry and non-singularity. Since, as Sentelle shows, our subproblems can be further divided into positive-semidefinite problems, on which many iterative algorithms perform best, this would seem to be the ideal approach.

In order to design an iterative solver for the linear system

$$\mathbf{Ax} = \mathbf{d},$$

we rewrite the system, instead solving the unconstrained optimization problem

$$\min_{\mathbf{x}} W(\mathbf{x}) = \frac{1}{2} \mathbf{x}^T \mathbf{Ax} - \mathbf{x}^T \mathbf{d}.$$

Solving this is equivalent to solving our original equation  $\mathbf{Ax} = \mathbf{d}$ , as the minimum of this quadratic objective function must occur where

$$\nabla W(\mathbf{x}) = \mathbf{0}.$$

Taking the gradient of our objective function, we see that this is precisely equivalent to

$$\begin{aligned} \mathbf{Ax} - \mathbf{d} &= \mathbf{0}, \\ \mathbf{Ax} &= \mathbf{d}, \end{aligned}$$

as desired.

Iterative optimizers work by taking a series of steps, altering the current proposed solution repeatedly until it converges on the correct solution. To make this process clearer and simplify our optimizer design, we rewrite our objective function in terms of a current proposed solution  $\mathbf{x}_0$ , a step size  $a$ , and a step direction  $\hat{p}$ :

$$\begin{aligned} W'(\mathbf{x}_0, a, \hat{p}) &= \frac{1}{2} (\mathbf{x}_0 + a\hat{p})^T \mathbf{A} (\mathbf{x}_0 + a\hat{p}) - (\mathbf{x}_0 + a\hat{p})^T \mathbf{d} \\ &= \frac{1}{2} a^2 (\hat{p}^T \mathbf{A} \hat{p}) + a \left( \frac{1}{2} \mathbf{x}_0^T \mathbf{A} \hat{p} + \frac{1}{2} \hat{p}^T \mathbf{A} \mathbf{x}_0 - \hat{p}^T \mathbf{d} \right) + \left( \frac{1}{2} \mathbf{x}_0^T \mathbf{A} \mathbf{x}_0 - \mathbf{x}_0^T \mathbf{d} \right) \\ &= \frac{1}{2} a^2 (\hat{p}^T \mathbf{A} \hat{p}) + a (\hat{p}^T (\mathbf{A} \mathbf{x}_0 - \mathbf{d})) + \left( \frac{1}{2} \mathbf{x}_0^T \mathbf{A} \mathbf{x}_0 - \mathbf{x}_0^T \mathbf{d} \right). \end{aligned}$$

In fact, many iterative optimizers divide each step cleanly into two parts: they select a step direction, then determine a step size. To mimic this, we now consider the problem with our initial proposed solution  $\mathbf{x}_0$  and our step direction  $\hat{p}$  both fixed. Our optimum step size must be that value of  $a$  for which the derivative of this restricted objective function is zero:

$$\begin{aligned} 0 &= \frac{\partial W'}{\partial a} \\ &= a (\hat{p}^T \mathbf{A} \hat{p}) + (\hat{p}^T (\mathbf{A} \mathbf{x}_0 - \mathbf{d})), \\ a &= \frac{\hat{p}^T (\mathbf{d} - \mathbf{A} \mathbf{x}_0)}{\hat{p}^T \mathbf{A} \hat{p}}. \end{aligned}$$

Many of the standard iterative optimizers build upon this basic approach. Gradient descent, as specialized for quadratic problems, follows this approach exactly; it sets its search direction in the direction of the negative gradient at each step, greedily choosing the direction that promises the most improvement for the smallest step.

```

input : An  $n \times n$  symmetric matrix  $A$ 
         An  $n$ -vector  $b$ 
output: A solution  $x$  to the system of
         equations  $Ax = b$ 

1  $x_0 \leftarrow 0$ ;
2  $r_0 \leftarrow b$ ;
3  $p_0 \leftarrow b$ ;
4 for  $k \leftarrow 1$  to  $n$  do
5    $\alpha_k \leftarrow \frac{\|r_{k-1}\|^2}{p_k^T A p_k}$ ;
6    $x_k \leftarrow x_{k-1} + \alpha_k p_{k-1}$ ;
7    $r_k \leftarrow r_{k-1} - \alpha_k A p_{k-1}$ ;
8   if  $\|r_k\|$  small enough then
9      $x \leftarrow x_k$ ;
10    return;
11  end
12   $\beta_k \leftarrow \frac{\|r_k\|^2}{\|r_{k-1}\|^2}$ ;
13   $p_k \leftarrow r_k + \beta_k p_{k-1}$ ;
14 end
15  $x \leftarrow x_n$ ;
16 return;

```

(a) Conjugate Gradient

```

input : An  $n \times n$  symmetric matrix  $A$ 
         An  $n$ -vector  $b$ 
output: A solution  $x$  to the system of
         equations  $Ax = b$ 

1  $x_0 \leftarrow 0$ ;
2  $r_0 \leftarrow b$ ;
3  $p_0 \leftarrow b$ ;
4 for  $k \leftarrow 1$  to  $n$  do
5    $\alpha_k \leftarrow \frac{r_{k-1}^T A r_{k-1}}{\|A p_{k-1}\|^2}$ ;
6    $x_k \leftarrow x_{k-1} + \alpha_k p_{k-1}$ ;
7    $r_k \leftarrow r_{k-1} - \alpha_k A p_{k-1}$ ;
8   if  $\|r_k\|$  small enough then
9      $x \leftarrow x_k$ ;
10    return;
11  end
12   $\beta_k \leftarrow \frac{r_k^T A r_k}{r_{k-1}^T A r_{k-1}}$ ;
13   $p_k \leftarrow r_k + \beta_k p_{k-1}$ ;
14 end
15  $x \leftarrow x_n$ ;
16 return;

```

(b) (Unoptimized) Conjugate Residual

Figure 3. Side-by-side comparison of the Conjugate Gradient and Residual methods

Since it takes our derived optimal step at each iteration, the algorithm ensures that the directional derivative  $\frac{\partial W'}{\partial a} = \hat{p} \cdot \nabla W'$  is zero — and so that each search direction is orthogonal to the last. Obviously, as a descent algorithm, this approach can find minima, and by negation of the problem, maxima, but cannot find saddle points in the objective function. Thus, to ensure convergence, the matrix  $\mathbf{A}$  must be definite.

The conjugate gradient method, used both to optimize quadratic systems ( $\frac{1}{2}\alpha_s^T \mathbf{A}\alpha_s + \mathbf{d}^T \alpha_s$ ) and to solve linear systems ( $\mathbf{A}\mathbf{x} = \mathbf{d}$ ), is a clever improvement on gradient descent. Pseudocode can be found in Figure 3(a). Its only difference from gradient descent, as an algorithm, is that it  $\mathbf{A}$ -orthogonalizes (makes conjugate) each new search direction with respect to all of the previous directions. Conveniently, due to the relation of the search directions, this can be accomplished with reference only to the previous search direction and the residual, in a constant-time computation.

$\mathbf{A}$ -orthogonalization ensures that the algorithm will terminate within  $n$  iterations, where  $n$  is the dimension of the space; for revised simplex SVM training,  $n$  is equal to the number of candidate non-bound support vectors, and so is sometimes written  $n_s$ . Due to the use of the problem’s matrix  $\mathbf{A}$  in the algorithm’s “orthogonalization” and the properties of conjugacy, this algorithm can be proven to converge before termination, giving it a powerful guarantee gradient descent lacks. However, it fails if  $\mathbf{A}$  is indefinite. Not only is this algorithm, at its core, a descent algorithm, but  $\mathbf{A}$ -orthogonalization becomes a suspect step once  $\mathbf{A}$  no longer defines an inner product.

To address this problem, Luenberger invented the conjugate residual method [10], for which pseudocode is provided in Figure 3(b). He focused his attention on the linear system  $\mathbf{A}\alpha_s + \mathbf{d} = 0$ . Assuming this system is solvable ( $\mathbf{A}$  non-singular), its solution is also a stationary point of the quadratic system  $\frac{1}{2}\alpha_s^T \mathbf{A}\alpha_s + \mathbf{d}^T \alpha_s$ . To solve the linear system with an iterative method, he followed an approach almost identical to the conjugate gradient method, but used a different objective function

$$\begin{aligned}
 E(\alpha_s) &= \|\mathbf{A}\alpha_s + \mathbf{d}\|^2 \\
 &= \alpha_s^T \mathbf{A}^2 \alpha_s + 2(\mathbf{A}\mathbf{d})^T \alpha_s + d.
 \end{aligned}$$

Since  $\mathbf{A}^2$  is positive-definite for any non-singular symmetric matrix  $\mathbf{A}$ , this algorithm is guaranteed to converge, with the same basic properties as the conjugate gradient method. In fact, Luenberger proves that the residuals  $-\mathbf{d} - \mathbf{A}\alpha_s$  form a  $\mathbf{A}$ -orthogonal sequence, and so must reach zero in at most  $n$  iterations.

## 2.4 Preconditioning

We leave the analysis of the convergence of iterative methods, such as the conjugate gradient and conjugate residual methods, to better sources [8, 11]; the topic is mathematically extremely complex. However, we can summarize the convergence guarantees briefly: the conjugate gradient method (and to some degree, conjugate residual) requires more iterations to converge if the eigenvalues of the problem’s matrix are evenly distributed over a wide range, and fewer iterations to converge if the eigenvalues are instead distributed in a few tight clusters. A clever change of variables can maintain the symmetry of the matrix, leaving it solvable by the conjugate gradient and residual methods, while improving the eigenvalue distribution. The net effect of such a change is that, rather than depending on the eigenvalue distribution of  $\mathbf{A}$ , our algorithm now depends on the eigenvalue distribution of  $\mathbf{M}^{-1}\mathbf{A}$ , where the **preconditioner**  $\mathbf{M}$  represents the change of variables.

Clearly, if we choose an easily inverted preconditioner  $\mathbf{M}$  for which  $\mathbf{M}^{-1}\mathbf{A}$  has a good eigenvalue distribution, we may dramatically improve the performance of our algorithm. In addition, we must also choose our preconditioner carefully, as our ability to preserve the symmetry of our matrix (and thus the applicability of conjugate gradient) relies on having a factorizable preconditioner  $\mathbf{M} = \mathbf{C}^T\mathbf{C}$ . These restrictions, in practice, come into conflict, often opposing each other in the matrix manipulations they would prescribe. For example, if we choose  $\mathbf{M} = \mathbf{A}$ ,  $\mathbf{M}^{-1}\mathbf{A}$  would simply be the identity matrix, with all eigenvalues equal to 1. This is an ideal case for the conjugate gradient and residual methods, both of which are then guaranteed to converge in a single iteration. However, inverting  $\mathbf{A}$  is equivalent to solving the system of equations we seek to solve in the first place, and so this saves us no computational effort.

The problem of finding good preconditioners is, unfortunately, quite difficult; the requirements on the preconditioner are quite clear, but the structure of the problem provides very few guidelines on how one might construct such a matrix for any given system of equations. Among the preconditioners most often used are more easily inverted approximations of  $\mathbf{A}$  (such as a matrix containing only the diagonal elements of  $\mathbf{A}$ ), inverses of approximations to  $\mathbf{A}^{-1}$  (used implicitly, as  $\mathbf{M}^{-1}$  is constructed directly), and what are called **functional** preconditioners. These functional preconditioners are designed to take advantage of knowledge external to the problem regarding the form of the system of equations. In particular, for problems of the form

$$\begin{bmatrix} \mathbf{X} & \mathbf{B} \\ \mathbf{B}^T & \mathbf{0} \end{bmatrix} \mathbf{x} = \mathbf{d},$$

(known as KKT [Karush-Kuhn-Tucker] problems), there exist known strong preconditioners that leverage the structure of this matrix; some such preconditioners are discussed in Lukšan and Vlček [12]. The attentive reader may have noted that the inner problems we seek to solve are also of this form; in fact, as our experiments have shown, one of these preconditioners provides the best improvement in our expected performance, with significant advantages over other standard preconditioning techniques.

## 3. PROPOSED APPROACH

Following the same pattern used to precondition the conjugate gradient method, we were able to design a preconditioned conjugate residual method to serve as the inner solver in our revised simplex SVM training routine. Pseudocode for this algorithm is included as an appendix to this paper. Unfortunately, due to the nature of the transforms required to precondition a symmetry-based method, this algorithm can only be proven to converge for symmetric positive-definite preconditioners.

Our SVM training algorithm is based on Sentelle’s implementation of the revised simplex method for SVM training [6]. Beginning with his C++ code, we made slight modifications to replace his inner solver, based on a maintained Cholesky factorization, with our preconditioned conjugate residual method. We also took full advantage of the ability of the conjugate residual method to operate on an indefinite matrix; we replaced Sentelle’s inner step, in which he solves two systems of equations involving  $\mathbf{Q}_{ss}$ , with a step that only solves a single system of equations involving  $\mathbf{A}$ .

We ran tests without preconditioning, as well as testing several standard preconditioning techniques as applied to our KKT problem. We tried the naïve diagonal preconditioner (a diagonal matrix containing the diagonal elements of our matrix  $\mathbf{A}$ ), an incomplete Cholesky factorization, and the KKT matrix preconditioner used by Bonettini and Ruggiero [13] (originally advanced by Lukšan and Vlček [12] as their  $C_4$ ). This last technique uses a preconditioner of the form

$$\mathbf{M} = \begin{bmatrix} -\overline{\mathbf{Q}} & \mathbf{y}_s \\ \mathbf{y}_s^T & 0 \end{bmatrix}$$

for a problem with matrix

$$\mathbf{A} = \begin{bmatrix} -\mathbf{Q}_{ss} & \mathbf{y}_s \\ \mathbf{y}_s^T & 0 \end{bmatrix},$$

where  $\overline{\mathbf{Q}}$  is an easily-inverted approximation of  $\mathbf{Q}_{ss}$ . In our case, as in most cases found in the literature, we choose  $\mathbf{Q}$  to be a diagonal approximation of  $\mathbf{Q}$ , as no obvious patterns were present in our tested kernel matrices.

#### 4. EXPERIMENTAL RESULTS

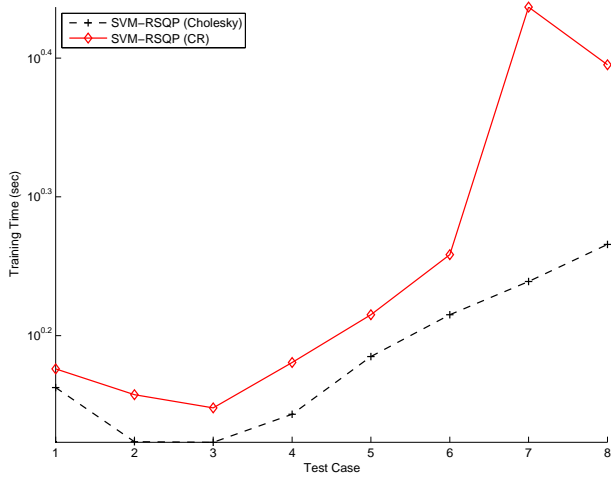
Preliminary testing showed that all preconditioners above failed to improve our code’s performance, and in fact worsened its numerical stability. This led to failures to converge, and so increased execution time. Therefore, all tests below were run with no preconditioning.

Our test cases, datasets and parameters, are taken from Sentelle [6]. Descriptions of the datasets in question are attached as Appendix B. For all linear kernel training tests, these cases are listed from left to right in increasing order of the training parameter  $C$ . For the radial basis function (RBF) kernels, test cases are listed first in increasing order of the kernel parameter  $\gamma$  (the inverse square of the standard deviation), then in order of increasing  $C$ . The parameter ranges tested are listed in Table 4, while the results of these tests are shown in Figures 4–10.

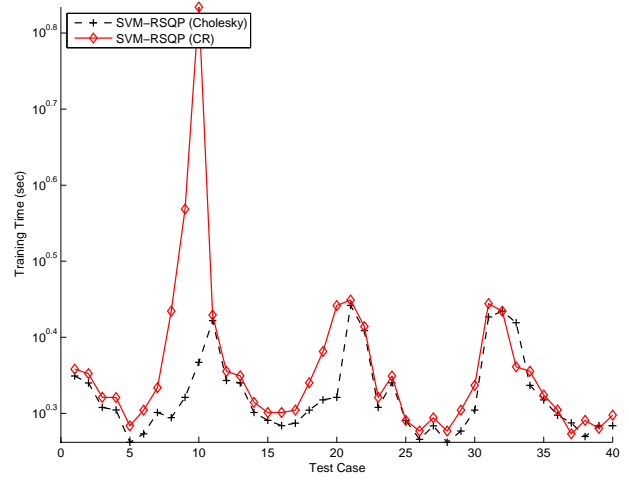
#### 5. DISCUSSION

As our results make clear, the effects of an iterative inner solver on the speed of revised simplex SVM training are highly sensitive to the training dataset used. Extremely long training times often indicate failures to converge, where our results differ significantly from the accepted answers. Most of these cases also exhibit exceedingly poor conditioning; a typical example has a near-uniform eigenvalue distribution, with a condition number on the order of  $10^6$  or higher. These observations together lead us to suspect that our problems in these borderline cases are caused by accumulated numerical error.

Despite the dependency of our training time on the training parameters and dataset, our experimental results show that except in a few cases, which are mostly those for which the algorithm fails to converge, the number of revised simplex iterations required to optimize the system is nearly independent of our choice of inner solver. Instead, we must look at the complexity of our subproblems. The size of our subproblems is directly determined by the number of candidate non-bound support vectors (those with  $0 < \alpha_i < C$ ) in that iteration, and so tends to correlate with the number of non-bound support vectors found in our eventual solution. Observing training time versus the number of non-bound support vectors on a log-log scale (Figure 11), we see a “cloud” at smaller numbers of non-bound support vectors. Profiling reveals that, for problems of this scale, both the iterative and direct inner solvers take a very small fraction of the total execution time, hiding a difference in performance. For higher numbers of non-bound support vectors, we see the training time for the direct-solver implementation remaining roughly constant (as the fraction of time in the inner solver remains small), while our conjugate residual implementation begins to stand out.

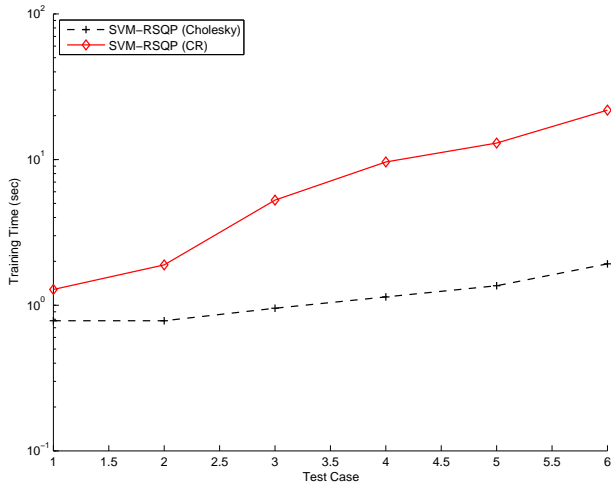


(a) Linear kernel

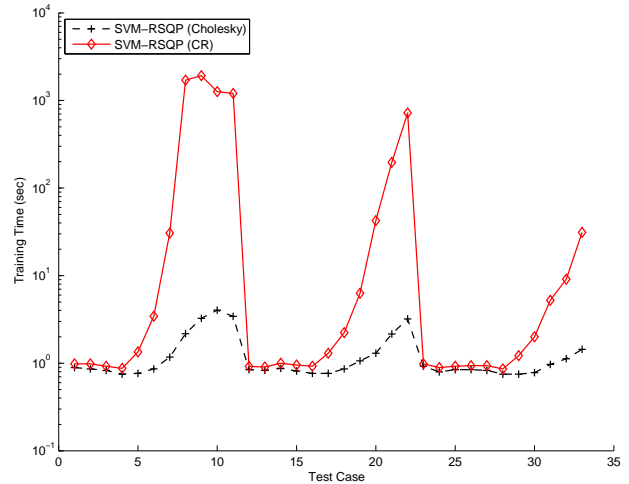


(b) RBF kernel

Figure 4. Training time comparisons for the abalonev2 dataset

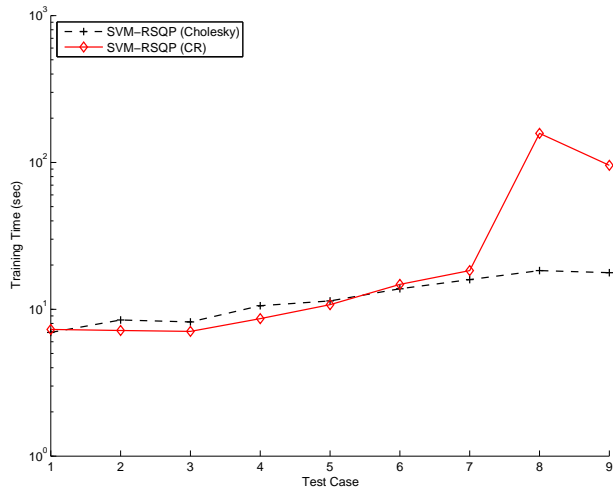


(a) Linear kernel

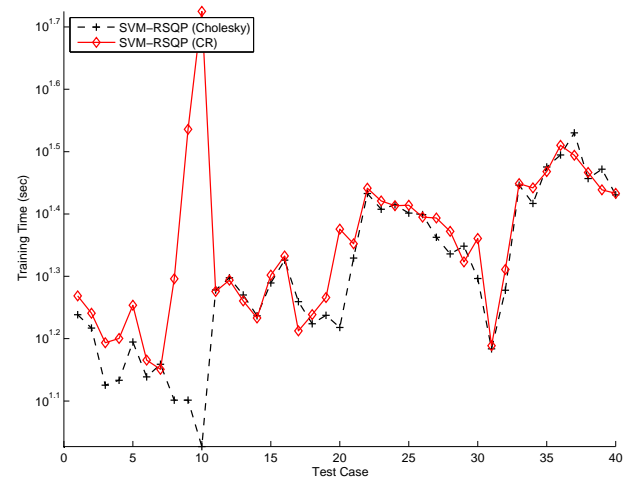


(b) RBF kernel

Figure 5. Training time comparisons for the adult-1a dataset

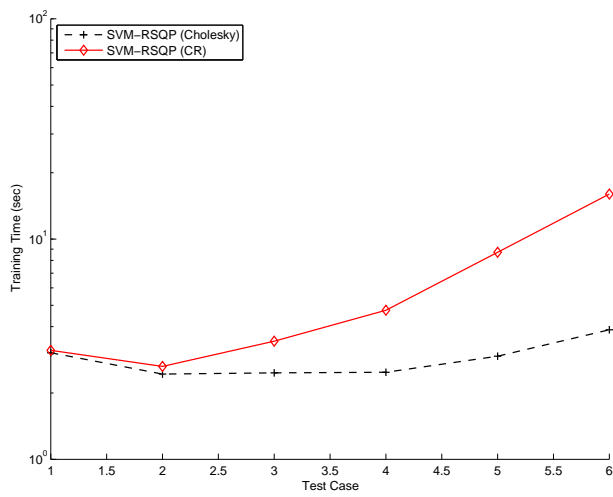


(a) Linear kernel

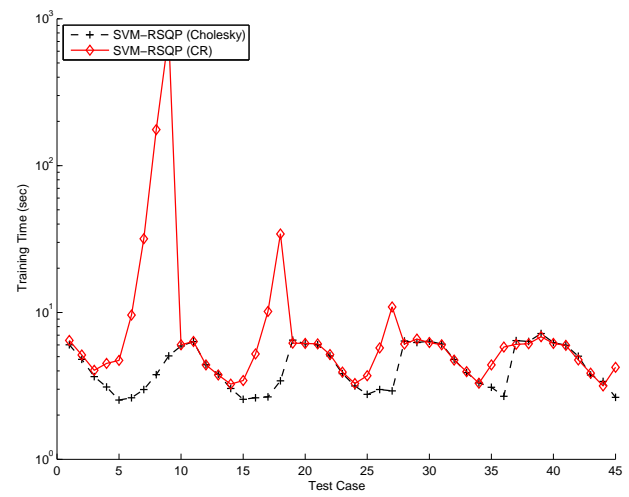


(b) RBF kernel

Figure 6. Training time comparisons for the letter-g dataset  
 Note: RBF kernel test cases 9 and 10 both failed to converge

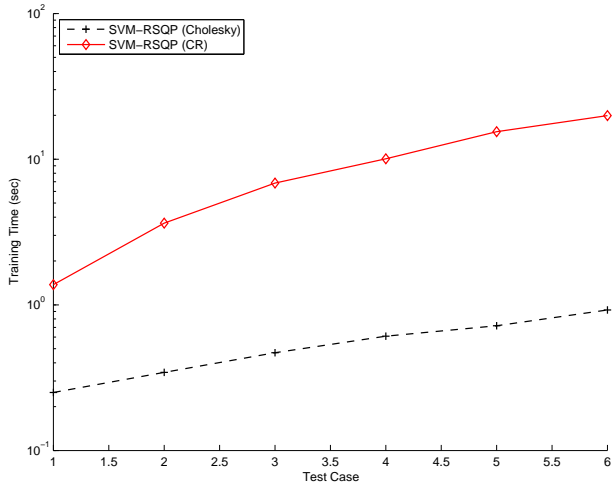


(a) Linear kernel

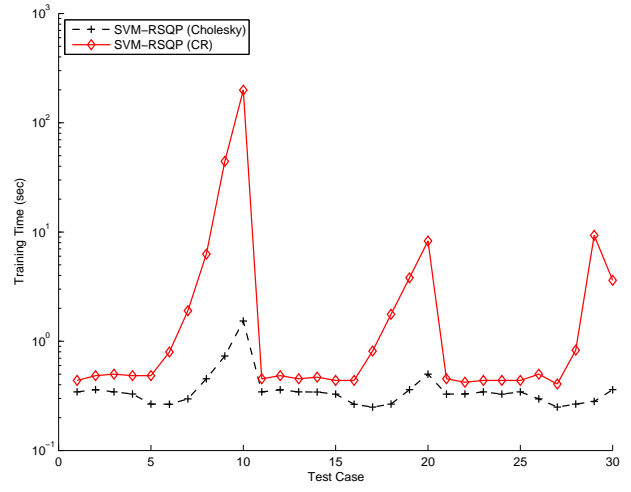


(b) RBF kernel

Figure 7. Training time comparisons for the spam dataset

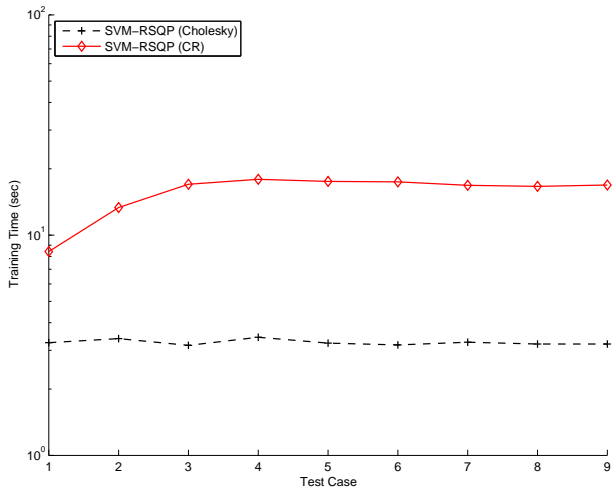


(a) Linear kernel

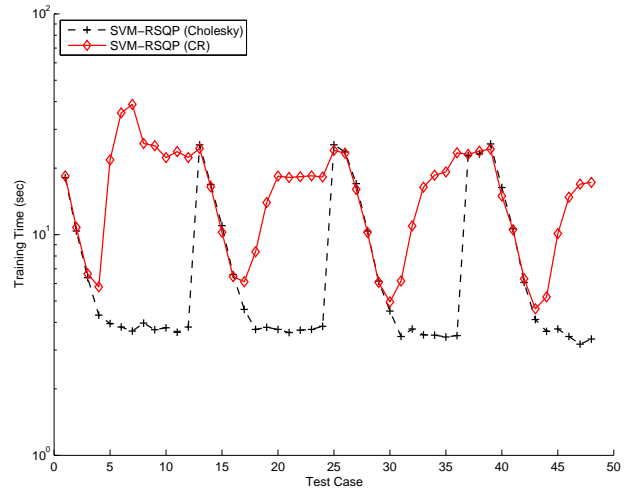


(b) RBF kernel

Figure 8. Training time comparisons for the splice dataset

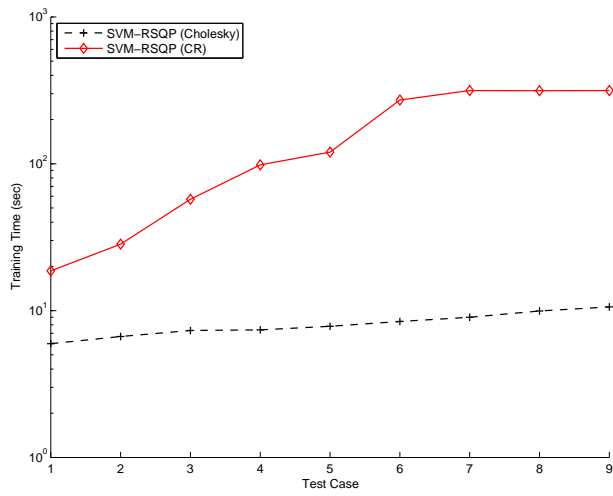


(a) Linear kernel

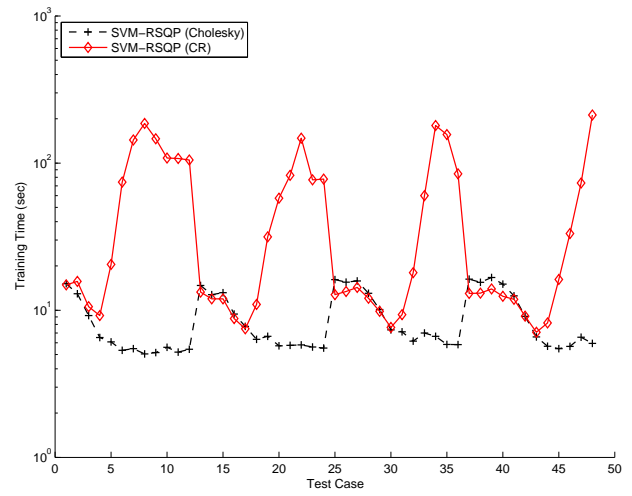


(b) RBF kernel

Figure 9. Training time comparisons for the ocr-0 dataset



(a) Linear kernel



(b) RBF kernel

Figure 10. Training time comparisons for the ocr-9 dataset

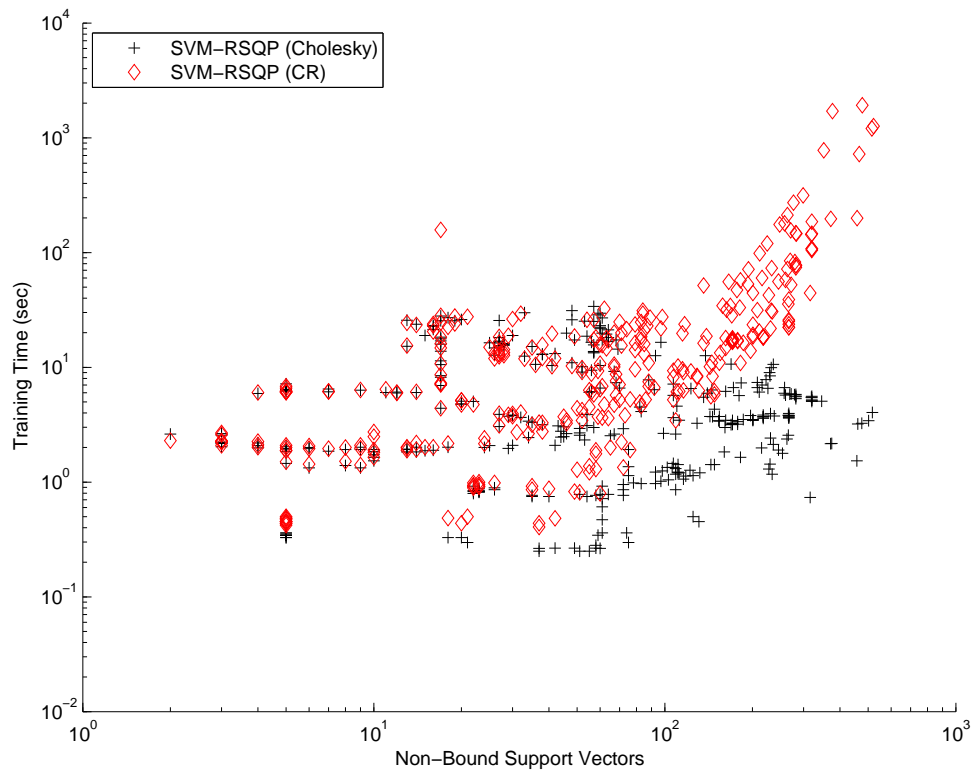


Figure 11. Training time as a function of the number of non-bound support vectors

Table 1. Range of  $C$  and  $\gamma$  parameters for each dataset

Dataset	Kernel	Range of Parameters*
abalone	linear	$\log_2(C) = [-3 : 2 : 11]$
abalone	rbf	$\log_2(C) = [-5 : 2 : 13], \log_2(\gamma) = [-5 : -2 : -11]$
adult-1a	linear	$\log_2(C) = [-3 : 2 : 7]$
adult-1a	rbf	$\log_2(C) = [-5 : 2 : 15], \log_2(\gamma) = [-7 : -2 : -11]$
letter-g	linear	$\log_2(C) = [-3 : 2 : 13]$
letter-g	rbf	$\log_2(C) = [-5 : 2 : 13], \log_2(\gamma) = [-5 : -2 : -11]$
spam	linear	$\log_2(C) = [-3 : 2 : 7]$
spam	rbf	$\log_2(C) = [-5 : 2 : 11], \log_2(\gamma) = [-3 : -2 : -11]$
splice	linear	$\log_2(C) = [-3 : 2 : 7]$
splice	rbf	$\log_2(C) = [-5 : 2 : 13], \log_2(\gamma) = [-11 : -2 : -15]$
ocr-9	linear	$\log_2(C) = [-3 : 2 : 13]$
ocr-9	rbf	$\log_2(C) = [-5 : 2 : 17], \log_2(\gamma) = [-9 : -2 : -15]$
ocr-0	linear	$\log_2(C) = [-3 : 2 : 13]$
ocr-0	rbf	$\log_2(C) = [-5 : 2 : 17], \log_2(\gamma) = [-9 : -2 : -15]$

\* The parameter ranges are given in the form  $[L : S : U]$ , where L is the lower limit, S is the step size, and U is the upper limit.

Fitting a log-log line (a power law) to our test cases requiring more than 100 non-bound support vectors (Figure 12), we see that our training times are an excellent fit to a power law with exponent equal to 3.0581. In other words, the time taken by our CR inner solver appears to be cubic in the size of our subproblems. Since each iteration of our inner solver takes  $O(n_s^2)$  time (dominated by the matrix-vector multiplication required at each step), this implies that our inner solver requires  $O(n_s)$  iterations to converge on the average subproblem we encounter. In fact, the convergence behavior of the conjugate residual method is well-studied; for ill-conditioned problems with a uniform eigenvalue distribution, the algorithm is expected to take exactly  $n_s + 1$  iterations to successfully converge. Clearly, the ill-conditioned nature of the subproblems of the revised simplex method for SVM training is the greatest obstacle for improving the performance of a low-memory implementation using an iterative inner solver.

In the context of our preliminary results, this final observation is very surprising; after all, preconditioning proved to be unhelpful in improving our training times. A more detailed inspection of the subproblems in question revealed that even the best preconditioner (from Bonettini and Ruggiero [13]) could only improve our condition number (and thus our convergence bounds) from  $10^6$  to  $10^3$ , even for small problems ( $n_s \approx 20$ ). The CR method is expected to converge in a number of iterations at best linear in the smaller of the condition number or the size of the problem, and so these preconditioners are insufficient to overcome the severe ill-conditioning of our inner problems.

## 6. CONCLUSIONS

In light of our results, we conclude that the revised simplex method with an iterative inner solver may be a viable solution to the problem of SVM training. Admittedly, Sentelle’s revised simplex SVM training method [6], using a direct solver (based on a Cholesky factorization), is clearly superior in terms of training time; where Sentelle’s Cholesky factorization solver takes time quadratic in the number of non-bound support vectors, the conjugate residual method appears to take time proportional to  $n_s^3$ . However, our iterative inner solver avoids Sentelle’s  $O(n_s^2)$  space requirement. With kernel caching, our method requires only  $O(n_s)$  memory. In some environments, this dramatic reduction in space requirements may be worth the penalty in training speed.

Our results also show a potentially promising approach for further research. The increased time requirement of our algorithm is entirely due to the ill-conditioning of the inner problems; the conjugate residual method requires at least  $O(n_s^2)$  time, but this leaves significant room for improvement. Preliminary tests indicate that the revised simplex SVM training inner problems are resistant to structural preconditioning, such as the KKT preconditioner of Lukšan and Vlček [12], but a functional preconditioner, more specialized to the SVM training problem, may be more effective in improving training speed.

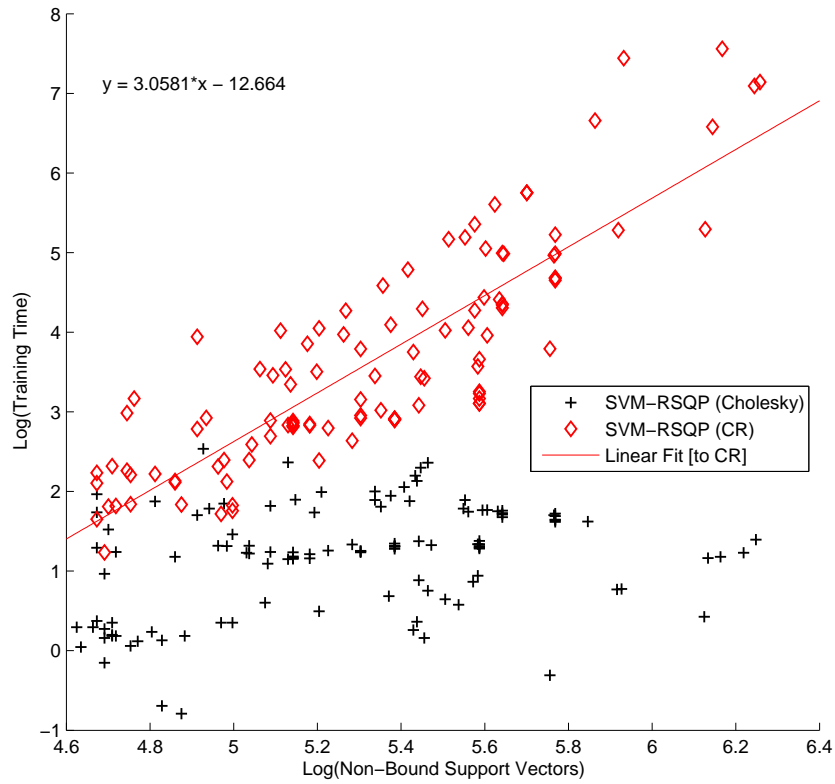


Figure 12. Training time as a function of the number of non-bound support vectors

### ACKNOWLEDGMENTS

This material is based upon work/research supported in part by the National Science Foundation under Grant No. 0647120 and Grant No. 0647018. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. The author Ruben Ramirez-Padron gratefully acknowledges the support of the College of Engineering and Computer Science and the I2Lab Fellowship at the University of Central Florida for the research presented here.

## REFERENCES

- [1] V. N. Vapnik, *The Nature of Statistical Learning Theory*, 1st ed. New York: Springer-Verlag, 1995.
- [2] T. Joachims, “Making large-scale support vector machine learning practical,” in *Advances in Kernel Methods: Support Vector Machines*, B. Schölkopf, C. Burges, and A. Smola, Eds. Cambridge, MA: MIT Press, 1998, ch. 11.
- [3] J. Platt, “Fast training of support vector machines using sequential minimal optimization,” in *Advances in Kernel Methods: Support Vector Machines*, B. Schölkopf, C. Burges, and A. Smola, Eds. Cambridge, MA: MIT Press, 1998, ch. 12.
- [4] A. Shilton, M. Palaniswami, D. Ralph, and A. C. Tsoi, “Incremental training of support vector machines,” *IEEE Transactions on Neural Networks*, vol. 16, no. 1, pp. 114–131, January 2005.
- [5] K. Scheinberg, “An efficient implementation of an active set method for SVMs,” *Journal of Machine Learning Research*, vol. 7, pp. 2237–2257, 2006.
- [6] C. Sentelle, “An adapted revised simplex method for SVM training,” Unpublished, 2008.
- [7] M. H. Rusin, “A revised simplex method for quadratic programming,” *SIAM Journal on Applied Mathematics*, vol. 20, no. 2, pp. 143–160, March 1971.
- [8] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed. New York: Springer, 2006.
- [9] B. Schölkopf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Cambridge, MA: MIT Press, 2002.
- [10] D. G. Luenberger, “The conjugate residual method,” *SIAM Journal on Numerical Analysis*, vol. 7, no. 3, pp. 390–398, September 1970.
- [11] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd ed. Philadelphia, PA: SIAM, 2003.
- [12] L. Lukšan and J. Vlček, “Indefinitely preconditioned inexact newton method for large sparse equality constrained non-linear programming problems,” *Numerical Linear Algebra with Applications*, vol. 5, no. 3, pp. 219–247, May/June 1998.
- [13] S. Bonettini and V. Ruggiero, “Some iterative methods for the solution of a symmetric indefinite kkt system,” *Computational Optimization and Applications*, vol. 38, no. 1, pp. 3–25, September 2007.
- [14] A. Asuncion and D. Newman, “UCI machine learning repository,” 2007. [Online]. Available: <http://www.ics.uci.edu/~mllearn/MLRepository.html>

## APPENDIX A. PRECONDITIONED CONJUGATE RESIDUAL PSEUDOCODE

```

input : An  $n \times n$  symmetric matrix  $A$ 
         An  $n \times n$  symmetric matrix  $M^{-1}$ 
         ( $M$  being our preconditioner)
         An  $n$ -vector  $b$ 
output: A solution  $x$  to the system of
         equations  $Ax = b$ 

1  $x_0 \leftarrow 0$ ;
2  $r_0 \leftarrow b$ ;
3  $p_0 \leftarrow M^{-1}b$ ;
4  $z_0 \leftarrow AM^{-1}r_0$ ;
5  $w_0 \leftarrow Ap_0$ ;
6  $\phi_0^2 \leftarrow b^T M^{-1}b$ ;
7  $\mu_0 \leftarrow r_0^T M^{-1}z_0$ ;
8 for  $k \leftarrow 1$  to  $n$  do
9    $\alpha_k \leftarrow \frac{\mu_{k-1}}{w_{k-1}^T M^{-1}w_{k-1}}$ ;
10   $x_k \leftarrow x_{k-1} + \alpha_k p_{k-1}$ ;
11   $r_k \leftarrow r_{k-1} - \alpha_k w_{k-1}$ ;
12   $\phi_k^2 \leftarrow r_k^T M^{-1}r_k$ ;
13  if  $|\phi_k^2|$  small enough then
14     $x \leftarrow x_k$ ;
15    return;
16  end
17   $z_k \leftarrow AM^{-1}r_k$ ;
18   $\mu_k \leftarrow r_k^T M^{-1}z_k$ ;
19   $\beta_k \leftarrow \frac{\mu_k}{\mu_{k-1}}$ ;
20   $p_k \leftarrow M^{-1}r_k + \beta_k p_{k-1}$ ;
21   $w_k \leftarrow z_k + \beta_k w_{k-1}$ ;
22 end
23  $x \leftarrow x_n$ ;
24 return;

```

## APPENDIX B. DESCRIPTIONS OF DATASETS TESTED

All descriptions are based on Sentelle’s descriptions [6], with minor editing.

**adult-1a** The goal of the adult data set is to predict household incomes that are greater than \$50K per year. There are 123 features, of which approximately 12 are non-zero, and 1605 entries. Due to the large number of zero features, this data set benefits from employing sparse kernel evaluations as described by both Joachims [2] and Platt [3]. This dataset is a UCI repository [14] dataset.

**abalonev2** This data set contains measurements from abalone which are then used to determine age. There are eight features, one being categorical (sex) and the rest numerical. For the purposes of this research the sex attribute (male/female/infant) was mapped to a set of binary attributes (1, 0, 0), (0, 1, 0), and (0, 0, 1), respectively. The age was thresholded at ten rings (1.5 rings per year) to create a binary classification problem. This dataset was also obtained from the UCI repository [14].

**letter-g** This dataset is derived from the Letter Recognition dataset at the UCI repository [14]. The original task was to identify the 26 capital letters in the English alphabet. The characters were derived from 20 various fonts, randomly distorted, and rendered into a pixel-based format. A total of 16 different features were then measured and reported. This dataset is configured so as to create a binary classification problem, distinguishing the letter “G” from all other letters.

**OCR-0** This dataset is based upon the USPS (United States Postal Service) Optical Character Recognition (OCR) dataset. The original task was to recognize hand-written digits between 0 and 9. There are 256 features for each example (7291 examples) which are scaled between 0 and 255. This dataset is configured so as to create a binary classification problem, distinguishing the digit “0” from all other digits.

**OCR-9** Based on the same dataset as OCR-0, this dataset is configured so as to create a binary classification problem, distinguishing the digit “9” from all other digits.

**spam** The spam dataset contains 57 features and 4601 examples for the task of distinguishing spam email from normal email. Example features consist of percentage of content containing 48 different words, percentage of content containing 6 different characters, and statistics regarding run-lengths of capital letters. This is a UCI repository [14] dataset.

**splice** The splice dataset contains 60 features and 3190 examples. The original task was to determine if a particular location within a DNA sequence is a splice junction, and if so, whether the junction is an exon/intron (EI) boundary or an intron/exon (IE) boundary. The 60 features of each pattern represent the base-pairs starting at index -30 and ending at index +30, relative to the current location, and are each one of four letters:  $\{a, g, t, c\}$ . For this test, each of these “letters” was converted to an integer quantity; the patterns were then scaled to fit each feature into the range  $[-1, 1]$ . This dataset is configured so as to reduce the classification problem to determining whether or not the current location is a splice junction. This is a UCI repository [14] dataset.