An Efficient Active Set Method for SVM Training without Singular Inner Problems

Christopher Sentelle, Georgios C. Anagnostopoulos and Michael Georgiopoulos

Abstract— **Efficiently implemented active set methods have been successfully applied to Support Vector Machine (SVM) training. These active set methods offer higher precision and incremental training at the cost of additional memory requirements when compared to decomposition methods such as Sequential Minimal Optimization (SMO). However, all existing active set methods must deal with singularities occurring within the inner problem solved at each iteration, a problem that leads to more complex implementation and potential inefficiencies. Here, we introduce a revised simplex method, originally introduced by Rusin, adapted for SVM training and show this is an active set method similar to most existing methods with the advantage of maintaining nonsingularity of the inner problem. We compare performance to an existing active set method introduced by Scheinberg and demonstrate an improvement in training times, in some cases. We show our method maintains a slightly simpler implementation and offers advantages in terms of applying iterative methods to alleviate memory concerns. We also show performance of the active set methods when compared to state-of-the-art decomposition implementations such as SVMLight and SMO.**

I. Introduction

EFFICIENTLY implemented active set methods have been introduced for solving the quadratic programming problem associated with the Support Vector Machine (SVM) [1], [2], [3], and [4]. The active set method does not rely on decomposition strategies such as in SMO [5] and SVMLight [6], naturally supports incremental training, and provides solutions with increased numerical accuracy. As pointed out in [7], the active set method is generally ideal for small to medium-sized problems; nevertheless [1] also points out that the active set method is ideal for problems with dense Hessians, as is the case for the SVM problem.

At the core of existing active set implementations, the following generalized system of equations, or inner problem, must be solved at each iteration

$$\begin{pmatrix} -\mathbf{Q_{ss}} & \mathbf{y_s} \\ \mathbf{y_s}^T & 0 \end{pmatrix} \begin{pmatrix} \mathbf{h} \\ g \end{pmatrix} = \begin{pmatrix} \mathbf{c} \\ d \end{pmatrix} \quad (1)$$

where $\mathbf{Q_{ss}}$ represents a subset of the original kernel matrix $\mathbf{Q}$ defined as $Q_{ij} = y_i y_j K(x_i, x_j)$, $K(,)$ is a Mercer kernel, and $\mathbf{y_s}$ is a subset of $\mathbf{y}$ where $\mathbf{y}$ is a $l \times 1$ vector of labels for l data points and $y_i \in \{1, -1\}$. The Mercer kernel, and, therefore, $\mathbf{Q}$ is positive semidefinite. This system can be singular and the amount of rank deficiency depends upon the selected kernel and the selection of entries within the sub-matrix $\mathbf{Q_{ss}}$.

Existing active set implementations employ various mechanisms to contend with rank deficiencies and have been shown to be successful from a practical standpoint. For example, the SVM-QP algorithm introduced in [1] detects rank deficiencies and chooses a prescribed infinite descent direction. In practice, this does not prevent convergence; however, it has the potential to create inefficiencies in the form of additional iterations.

In this paper, we adapt a revised simplex method for quadratic programming, introduced by Rusin [8], to SVM training, which provides theoretical justification for non-singularity of the inner problem solved at each iteration. We will show that the algorithm differs, not in the pricing strategy, but in how the inner problem is formed and solved.

The rest of this paper is organized as follows: In Section II we present the algorithm and discuss the guarantees of non-singularity, experiments and results are presented in Section III and, finally, Section IV concludes the paper.

II. Algorithm

The following soft-margin dual formulation of the SVM optimization problem is typically solved [9]

$$\begin{aligned} \min\ & \frac{1}{2}\alpha^T \mathbf{Q}\alpha - \mathbf{1}^T\alpha \\ \text{s.t. } & \mathbf{y}^T\alpha = 0 \\ & 0 \le \alpha_i \le C\ \forall i \end{aligned}$$

where C is a regularization parameter, $\mathbf{1}^T$ is a vector of all ones, α is a $l \times 1$ vector of Lagrange multipliers associated with l constraints in the primal problem and $\mathbf{Q}$ is the kernel matrix. If the kernel function is positive semi-definite, and, therefore, the matrix $[Q_{ij}]$ is positive semi-definite, then this formulation represents a convex optimization problem.

The problem statement for the revised simplex method for quadratic programming, as discussed in

Christopher Sentelle gratefully acknowledges the support of the College of Engineering & Computer Science and the I2Lab at the University of Central Florida. I would like to thank Ruben Ramirez Padron for his assistance and insightful suggestions.

This work was supported in part by NSF grants: 0341601, 0647018, 0717674, 0717680, 0647120, 0525429, 0806931.

978-1-4244-3553-1/09/$25.00 ©2009 IEEE

Rusin, is of the following form

$$\begin{aligned} \min\ & \mathbf{p}^{\mathrm{T}}\mathbf{x} - \frac{1}{2}\mathbf{x}^{\mathrm{T}}\mathbf{Q}\mathbf{x} \\ \text{s.t. } & \mathbf{A}\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geq 0 \end{aligned} \tag{2}$$

where $\mathbf{p}$ is a $n \times 1$ vector of linear costs, $\mathbf{x}$ is a $n \times 1$ vector representing the variables to be solved, $\mathbf{Q}$ is a $n \times n$ matrix of quadratic costs, $\mathbf{A}$ is a $m \times n$ matrix consisting of m constraints applied to n variables and $\mathbf{b}$ is a $m \times 1$ vector representing the constant offsets for the constraints. In this problem, Rusin assumes the problem is convex and $\mathbf{Q}$ is negative semi-definite. We can adapt the SVM quadratic programming problem into a form compatible with the revised simplex method with the use of slack variables, as follows

$$\begin{aligned} \min\ & \begin{pmatrix} -\mathbf{1}^T & \mathbf{0}^T \end{pmatrix}\begin{pmatrix} \alpha \\ \mathbf{s} \end{pmatrix} - \frac{1}{2}\begin{pmatrix} \alpha^T & \mathbf{s}^T \end{pmatrix}\begin{pmatrix} -\mathbf{Q} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{pmatrix}\begin{pmatrix} \alpha \\ \mathbf{s} \end{pmatrix} \\ \text{s.t. } & \alpha^T\mathbf{y} = 0 \\ & \alpha + \mathbf{s} = C\mathbf{1} \\ & \alpha \geq 0, \mathbf{s} \geq 0 \end{aligned} \tag{3}$$

where we have added the vector, $\mathbf{s}$, of slack variables to handle the inequality constraint $\alpha \leq C$. At each iteration of the revised simplex method in Rusin [8], the following problem is solved

$$\begin{pmatrix} -\mathbf{Q_B} & \mathbf{B}^T \\ \mathbf{B} & 0 \end{pmatrix}\begin{pmatrix} \mathbf{h} \\ \mathbf{g} \end{pmatrix} = \begin{pmatrix} \mathbf{q_i} \\ \mathbf{a_i} \end{pmatrix} \tag{4}$$

where $\mathbf{Q_B}$ is the sub-matrix of $\mathbf{Q}$ corresponding to basic variables, $\mathbf{a_i}$ is the column of $\mathbf{A}$ for the variable entering the basis and $\mathbf{q_i}$ is the corresponding column of $\mathbf{Q}$, and the basis matrix, $\mathbf{B}$, is formed from the columns of the constraint matrix, $\mathbf{A}$, corresponding to basic variables. Recall that the basic variables are non-zero variables.

For the SVM formulation, basic variables consist of those components of α or $\mathbf{s}$ which are non-zero. In the case of a non-support vector, $s_i = C$ and $\alpha_i = 0$, for a non-bound support vector, $s_i > 0$ and $\alpha_i > 0$ and for the case of a bound support vector, $s_i = 0$ and $\alpha_i = C$. We can partition the vector α into α_0, α_s and α_c for non-support vectors, non-bound support vectors, and bound support vectors, respectively. Similarily, we have components for $\mathbf{s}$ which are $\mathbf{s_0}$, $\mathbf{s_s}$ and $\mathbf{s_c}$. We define columns of the identity matrix belonging to each type of variable as $\mathbf{I_0}$, $\mathbf{I_s}$ and $\mathbf{I_c}$, where $\mathbf{I_0}$, for example, contains the columns of the identity matrix associated with non-support vectors. The vector $\mathbf{y}$ is also broken down into $\mathbf{y_0}$, $\mathbf{y_s}$ and $\mathbf{y_c}$.

Therefore, the corresponding basis matrix $\mathbf{B}$ for the SVM formulation, consisting of only the columns, where α or $\mathbf{s}$ are non-zero, can be written as

$$\mathbf{B} = \begin{pmatrix} \mathbf{y_s}^T & \mathbf{y_c}^T & \mathbf{0_o}^T & \mathbf{0_s}^T \\ \mathbf{I_s} & \mathbf{I_c} & \mathbf{I_o} & \mathbf{I_s} \end{pmatrix}. \tag{5}$$

Note that $\mathbf{I_s}$ appears twice since both α_s and $\mathbf{s_s}$ components are non-zero for non-bound support vectors. Substituting (5) into (4) and noting from (3) that the effective entries for $\mathbf{Q_B}$ corresponding to the slack variables $\mathbf{s}$ are zero results in the following

$$\begin{pmatrix} -\mathbf{Q_{ss}} & -\mathbf{Q_{sc}} & \mathbf{0} & \mathbf{0} & \mathbf{y_s} & \mathbf{I_s}^T \\ -\mathbf{Q_{cs}} & -\mathbf{Q_{cc}} & \mathbf{0} & \mathbf{0} & \mathbf{y_c} & \mathbf{I_c}^T \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0_o} & \mathbf{I_o}^T \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0_s} & \mathbf{I_s}^T \\ \mathbf{y_s}^T & \mathbf{y_c}^T & \mathbf{0_o}^T & \mathbf{0_s}^T & \mathbf{0} & \mathbf{0} \\ \mathbf{I_s} & \mathbf{I_c} & \mathbf{I_o} & \mathbf{I_s} & \mathbf{0} & \mathbf{0} \end{pmatrix}\begin{pmatrix} \mathbf{h_{s1}} \\ \mathbf{h_c} \\ \mathbf{h_o} \\ \mathbf{h_{s2}} \\ g_\beta \\ \mathbf{g} \end{pmatrix} = \begin{pmatrix} \mathbf{q_{si}} \\ \mathbf{q_{ci}} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{y_i} \\ \mathbf{e_i} \end{pmatrix} \tag{6}$$

where we have separated $\mathbf{h}$ into components $\mathbf{h_s}$, $\mathbf{h_c}$ and $\mathbf{h_o}$ and there are $\mathbf{h_s}$ components, $\mathbf{h_{s1}}$ for the α variables associated with a non-bound support vector and $\mathbf{h_{s2}}$ for the corresponding slack variables. The value, g_β is used to denote the component of $\mathbf{g}$ corresponding to an update in β, the Lagrange multiplier associated with the $\mathbf{y}^T\alpha = 0$ constraint. The vector $\mathbf{e_i}$ is a unit vector with a 1 in the location corresponding to the variable being added to the basis. $\mathbf{Q_{ss}}$ refers to the square portion of $\mathbf{Q}$ for the non-bound support vectors, $\mathbf{Q_{cc}}$ is the square portion for bound support vectors, and $\mathbf{Q_{cs}}$ contains the rows of $\mathbf{Q}$ corresponding to bound support vectors and columns corresponding to non-bound support vectors.

Since slack variables are related to the original variables by $\alpha_i = s_i + C$ and $\mathbf{h_{s1}} = -\mathbf{h_{s2}}$, we can deal with the slack variables, implicitly, from here on out. At each iteration, then, we will either be increasing a component, α_i from 0 or decreasing the component from C. For the case where a non-support vector enters the basis (α_i is increased from 0) the following system of equations is solved.

$$\begin{aligned} \begin{pmatrix} -\mathbf{Q_{ss}} & \mathbf{y_s} \\ \mathbf{y_s}^T & 0 \end{pmatrix}\begin{pmatrix} \mathbf{h_{s1}} \\ g_\beta \end{pmatrix} &= \begin{pmatrix} \mathbf{q_{si}} \\ y_i \end{pmatrix} \\ \mathbf{g_c} &= \mathbf{q_{ci}} - \mathbf{y_c}g_\beta + \mathbf{Q_{cs}}\mathbf{h_{s1}} \\ h_i &= -1 \end{aligned} \tag{7}$$

Note that $\mathbf{h_o} = \mathbf{0}$, $\mathbf{h_c} = \mathbf{0}$, $\mathbf{g_s} = \mathbf{0}$, and $\mathbf{g_o} = \mathbf{0}$. The fact that $h_i = -1$ indicates the corresponding variable α_i is increasing. For the case where a bound support vector enters the basis (α_i is decreased from C) we solve the following

$$\begin{aligned} \begin{pmatrix} -\mathbf{Q_{ss}} & \mathbf{y_s} \\ \mathbf{y_s}^T & 0 \end{pmatrix}\begin{pmatrix} \mathbf{h_{s1}} \\ g_\beta \end{pmatrix} &= \begin{pmatrix} -\mathbf{q_{si}} \\ -y_i \end{pmatrix} \\ \mathbf{g_c} &= -\mathbf{q_{ci}} - \mathbf{y_c}g_0 + \mathbf{Q_{cs}}\mathbf{h_{s1}} \\ h_i &= 1. \end{aligned} \tag{8}$$

Once again, the unspecified components of $\mathbf{h}$, and $\mathbf{g}$ are set to zero and $h_i = 1$ implies α_i is decreasing from C.

To find the pivot variable at each step, we must compute the reduced price of the non-basic variables, which, in our case, correspond to non support vectors where $\alpha_j = 0$ or to bound support vectors where $s_j = 0$.

Per Rusin, the pricing computation is performed as follows

$$\delta_j = p_j - \pi^T \mathbf{a_j} - \mathbf{x}^T \mathbf{q_j}.$$

Using the SVM formulation, we can compute pricing for $\alpha_j = 0$ and $s_j = 0$ as follows, for $\alpha_j = 0$

$$\delta_j = -1 - \pi_0 y_j - \pi_{j+1} - \mathbf{q_{js}}\alpha_\mathbf{s} - \mathbf{q_{jc}}\alpha_\mathbf{c} \tag{9}$$

and as follows, for $s_j = 0$

$$\delta_j = -\pi_{j+1}. \tag{10}$$

From the following equation, in Rusin, we can compute π

$$\mathbf{p_B} - \mathbf{B}\pi - \mathbf{Q_B x_B} = 0,$$

which for the SVM formulation becomes

$$\begin{pmatrix} \mathbf{y_s} & \mathbf{I_s} \\ \mathbf{y_c} & \mathbf{I_c} \\ \mathbf{0_s} & \mathbf{I_s} \\ \mathbf{0_0} & \mathbf{I_0} \end{pmatrix} \begin{pmatrix} \pi_0 \\ \pi \end{pmatrix} - \begin{pmatrix} -\mathbf{Q_{ss}} & -\mathbf{Q_{sc}} & \mathbf{0} & \mathbf{0} \\ -\mathbf{Q_{cs}} & -\mathbf{Q_{cc}} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \alpha_\mathbf{s} \\ \alpha_\mathbf{c} \\ \mathbf{s_s} \\ \mathbf{s_0} \end{pmatrix} = \begin{pmatrix} -\mathbf{1_s} \\ -\mathbf{1_c} \\ \mathbf{0} \\ \mathbf{0} \end{pmatrix}. \tag{11}$$

We immediately observe that $\pi_\mathbf{o} = \mathbf{0}$ and $\pi_\mathbf{s} = \mathbf{0}$ and we can simplify (9) to

$$\delta_\mathbf{o} = -\mathbf{1} - \pi_0 \mathbf{y_0} - \mathbf{Q_{os}}\alpha_\mathbf{s} - \mathbf{Q_{oc}}\alpha_\mathbf{c}. \tag{12}$$

We observe for (10) that we are referencing the components $\pi_\mathbf{c}$ which, from (11) can be solved to give

$$\pi_\mathbf{c} = -\mathbf{1_c} - \pi_0 \mathbf{y_c} - \mathbf{Q_{cs}}\alpha_\mathbf{s} - \mathbf{Q_{cc}}\alpha_\mathbf{c},$$

and, upon substituting this expression for $\pi_\mathbf{c}$ into (10) yields

$$\delta_\mathbf{c} = \mathbf{1_c} + \pi_0 \mathbf{y_c} + \mathbf{Q_{cs}}\alpha_\mathbf{s} + \mathbf{Q_{cc}}\alpha_\mathbf{c}. \tag{13}$$

We can create a pricing vector of size n and compute (12) for non-support vectors and compute (13) for bound support vectors. The reduced cost is, of course, zero for non-bound support vectors. Furthermore, since π is only non-zero for the bound support vectors, and since we only need this for pricing, we can avoid the computation of $\mathbf{g_c}$ in (7) and (8), since this is only used as an update to π. Another observation is that the variable π_0, which is the Lagrange multiplier associated with the equality constraint $\mathbf{y}^T\alpha = 0$, is really the bias, β, from the primal SVM problem, and we can update this using g_β at each iteration.

Note that the components $\alpha_\mathbf{s}$ of α corresponding to the non-bound support vectors will alter at every iteration, and, therefore the computation $\mathbf{Q_{cs}}\alpha_\mathbf{s}$ and $\mathbf{Q_{ss}}\alpha_\mathbf{s}$ cannot be avoided at each iteration. However, the quantities $\mathbf{Q_{0c}}\alpha_\mathbf{c}$ and $\mathbf{Q_{sc}}\alpha_\mathbf{c}$ need not be computed at each iteration and can be updated as bound support vectors are added and removed from the problem.

Finally, the revised simplex method must be started from an initial basic feasible solution. The typical strategy is to apply a phase I/phase II approach as described in [7]. In our case, the goal is to find a basis matrix $\mathbf{B}$, which is non-singular and of size $m \times m$, where m is the number of constraints. For the SVM QP problem, we have $m = n + 1$ constraints, where n is the number of training data points. We can include all of the slack variables $\mathbf{s}$ in the set of basic variables and arbitrarily select α_1 to enter the basis. This corresponds to initializing α_1 as a non-bound support vector and all remaining variables as non-support vectors. Having selected α_1 as a basic variable, we can compute the initial values for π and α as follows

$$\begin{aligned} \alpha_i &= 0 \ \forall i \\ s_i &= C \ \forall i \\ \pi_0 &= -y_1 \end{aligned}$$

Since the slack variables are dealt with implicitly and $\pi_0 = \beta$, we have

$$\begin{aligned} \alpha_i &= 0 \ \forall i \\ \beta &= -y_1 \\ I_o = \{2 \ldots N\}, I_s = \{1\}, I_c &= \emptyset \end{aligned}$$

where we have reused the notation I_o, I_s, and I_c to denote index sets for the non-support, non-bound support, and bound support vectors. The details of the algorithm we call SVM-RSQP (SVM-Revised Simplex Quadratic Programming) are found in Algorithm 1.

A. Guarantee of non-singularity

In Rusin [8], Theorems [1-3] provide a guarantee of non-singularity of the basis matrix, (4), and extend quite easily to the matrix in (7) and (8). It can also easily be shown that $\mathbf{Q_{ss}}$ will have at most a single zero eigenvalue, which we show, later, still has to be detected in our algorithm. The important point is that there will be no more than one zero eigenvalue and both (7) and (8) will always have a finite solution.

This algorithm can be seen as an enhancement to the algorithm reported by Scheinberg, SVM-QP, [1]. For this reason, it makes sense to compare the two algorithms in order to highlight the important differences that guarantee non-singularity. First, we note that the pricing step is identical between the two methodologies, and, in fact, the pricing step has no bearing on the maintainenance of non-singularity and speed-up methodologies such as partial pricing, sprint [1], and shrinking [6] are equally applicable to both.

The first important difference is in the initial calculation of the descent direction given an index to a variable being added as a non-bound support vector. In SVM-QP, the optimal solution, given the current set of free variables is computed as

$$\begin{pmatrix} -\mathbf{Q_{ss}} & \mathbf{y_s} \\ \mathbf{y_s}^T & 0 \end{pmatrix} \begin{pmatrix} \mathbf{a_s}^* \\ \beta^* \end{pmatrix} = \begin{pmatrix} -\mathbf{1_s} + \mathbf{Q_{sc}}\alpha_\mathbf{c} \\ -\mathbf{y_c}^T\alpha_\mathbf{c} \end{pmatrix} \tag{14}$$

and subtracted from the current solution to obtain a descent direction,

$$\mathbf{d} = \begin{pmatrix} \alpha - \alpha^* \\ \beta - \beta^* \end{pmatrix}.$$

Algorithm 1: SVM Revised Simplex Training

Input: Dataset $\mathbf{X} \in \Re^{m \times n}$, kernel matrix K, labels, $\mathbf{y}$, stopping criterion tol

Output: α, b

$\alpha_i \leftarrow 0 \ \forall i$
$\beta \leftarrow -y_1$
$I_o \leftarrow \{2 \ldots N\}$, $I_c \leftarrow \emptyset$, $I_s \leftarrow \{1\}$
while *true* **do**
  Calculate the reduced cost, δ_j using (12), (13)
  $i = \arg\min_j \delta_j$
  if $\delta_i \geq tol$ **then**
    stop
  if $i \in I_o$ **then**
    solve $\begin{pmatrix} -\mathbf{Q_{ss}} & \mathbf{y_s} \\ \mathbf{y_s}^T & 0 \end{pmatrix} \begin{pmatrix} \mathbf{h_{s1}} \\ g_\beta \end{pmatrix} = \begin{pmatrix} \mathbf{q_{si}} \\ y_i \end{pmatrix}$
    $\mathbf{h_i} \leftarrow -1$
    $\gamma \leftarrow -q_{ii} - y_i g_\beta + \mathbf{q_{si}^T h_{s1}}$
  else if $i \in I_c$ **then**
    solve $\begin{pmatrix} -\mathbf{Q_{ss}} & \mathbf{y_s} \\ \mathbf{y_s}^T & 0 \end{pmatrix} \begin{pmatrix} \mathbf{h_{s1}} \\ g_\beta \end{pmatrix} = \begin{pmatrix} -\mathbf{q_{si}} \\ -y_i \end{pmatrix}$
    $\mathbf{h_i} \leftarrow 1$
    $\gamma \leftarrow -q_{ii} + y_i g_\beta - \mathbf{q_{si}^T h_{s1}}$
  $I_s \leftarrow I_s \cup \{i\}$
  while $\delta_i < -tol$ **do**
    $\theta \leftarrow \min\left\{ \frac{\alpha_i}{h_j} \mid h_j > 0, \frac{\alpha_j - C}{h_j} \mid h_j < 0 \right\}$
    $\beta \leftarrow \beta - \theta g_\beta$
    $\alpha \leftarrow \alpha - \theta \mathbf{h}$
    $r \leftarrow \arg\min_j \left\{ \frac{\alpha_j}{h_j} \mid h_j > 0, \frac{\alpha_j - C}{h_j} \mid h_j < 0 \right\}$
    if $\gamma \geq 0$ *or* $\theta < \frac{\delta_i}{\gamma}$ **then**
      if $\mathbf{h_r} < 0$ **then**
        $I_c \leftarrow I_c \cup \{r\}$
      else
        $I_o \leftarrow I_o \cup \{r\}$
      $I_s \leftarrow I_s \setminus \{r\}$
      $\delta_i \leftarrow \delta_i - \theta\gamma$
    else
      $\theta \leftarrow \frac{\delta_i}{\gamma}$
      $\delta_i \leftarrow 0$
      break
    if $i \in I_o$ **then**
      solve $\begin{pmatrix} -\mathbf{Q_{ss}} & \mathbf{y_s} \\ \mathbf{y_s}^T & 0 \end{pmatrix} \begin{pmatrix} \mathbf{h_{s1}} \\ g_\beta \end{pmatrix} = \begin{pmatrix} \mathbf{e_i} \\ 0 \end{pmatrix}$
      $\gamma \leftarrow -1$
    else if $i \in I_c$ **then**
      solve $\begin{pmatrix} -\mathbf{Q_{ss}} & \mathbf{y_s} \\ \mathbf{y_s}^T & 0 \end{pmatrix} \begin{pmatrix} \mathbf{h_{s1}} \\ g_\beta \end{pmatrix} = \begin{pmatrix} -\mathbf{e_i} \\ 0 \end{pmatrix}$
      $\gamma \leftarrow -1$

In (14), $\mathbf{Q_{ss}}$ includes the newly added variable identified immediately after pricing. In (7), for example, we compute the descent direction $\mathbf{h}$ and g_β directly, and as result, $\mathbf{Q_{ss}}$ does not contain the newly added variable. It can be shown, quite easily, however, that these two computations are equivalent, if we choose the component of α_s^* corresponding to the newly added variable in (14) to be -1 and assume complementarity of the remaining basic variables, i.e., $\delta_s = \mathbf{0}$.

In both algorithms, the inner problem is repeatedly solved to find a direction of descent, and variables are removed from the set of non-bound support vectors that reach 0 or C, until the optimal solution for the newly added variable is found. In the case of SVM-QP, each time a variable is removed, (14) is re-solved to produce a new descent direction. On the other hand, SVM-RSQP solves the following after removing a variable and forming a new $\mathbf{Q_{ss}}$ matrix.

$$\begin{pmatrix} -\mathbf{Q_{ss}} & \mathbf{y_s} \\ \mathbf{y_s}^T & 0 \end{pmatrix} \begin{pmatrix} \mathbf{h_{s1}} \\ \beta \end{pmatrix} = \begin{pmatrix} \pm\mathbf{e_i} \\ 0 \end{pmatrix}$$

In this respect, SVM-QP and SVM-RSQP are quite different. To highlight the differences, we can rewrite (14) using $\mathbf{h} = \alpha_s - \alpha_s^*$ and $g_\beta = \beta - \beta^*$ as follows

$$\begin{pmatrix} -\mathbf{Q_{ss}} & \mathbf{y_s} \\ \mathbf{y_s}^T & 0 \end{pmatrix} \begin{pmatrix} \mathbf{h} \\ g_\beta \end{pmatrix} = \begin{pmatrix} -\mathbf{1_s} + \mathbf{Q_{sc}}\alpha_\mathbf{c} + \mathbf{Q_{ss}}\alpha_\mathbf{s} - \beta\mathbf{y_s} \\ 0 \end{pmatrix}.$$

The expression $-\mathbf{1_s} + \mathbf{Q_{sc}}\alpha_\mathbf{c} + \mathbf{Q_{ss}}\alpha_\mathbf{s} - \beta\mathbf{y_s}$ represents the pricing value for non-bound support vectors. If complementarity were maintained, this expression would equal zero and only a trivial solution to g and $\mathbf{h}$ would be available unless $\mathbf{Q_{ss}}$ were singular. In fact, complementarity is not maintained in the case of SVM-QP until the inner iterations, represented by the inner **while** loop in SVM-RSQP, have completed and the pricing step is again reached. SVM-RSQP, on the other hand, maintains complementarity for all of the non-bound support vectors with the exception of the newly added variable, during the inner iterations. The maintenance of the complementarity conditions by SVM-RSQP is an integral part of Theorem 2 in Rusin, which provides a guarantee of non-singularity as variables are removed from the basis.

B. Solution of the Inner Problem

When $\mathbf{Q_{ss}}$ is strictly positive definite, we can simply solve (7) or (8) as follows and as introduced in [1]

$$\begin{pmatrix} -\mathbf{Q_{ss}} & \mathbf{y_s} \\ \mathbf{y_s}^T & 0 \end{pmatrix} \begin{pmatrix} \mathbf{h_{s1}} \\ g_\beta \end{pmatrix} = \begin{pmatrix} \mathbf{u} \\ v \end{pmatrix},$$

which gives us a set of equations

$$\begin{aligned} -\mathbf{Q_{ss}}\mathbf{h_{s1}} + \mathbf{y_s} g_\beta &= \mathbf{u} \\ \mathbf{y_s}^T \mathbf{h_{s1}} &= v. \end{aligned}$$

We can multiply the top equation by $\mathbf{y_s}^T\mathbf{Q_{ss}}^{-1}$ and add the bottom equation to obtain

$$\mathbf{y_s}^T\mathbf{Q_{ss}}^{-1}\mathbf{y_s}g_\beta = \mathbf{y_s}^T\mathbf{Q_{ss}}^{-1}\mathbf{u} + v.$$

Solving for g_β we obtain

$$g_\beta = \frac{\mathbf{y_s}^T\mathbf{Q_{ss}}^{-1}\mathbf{u} + v}{\mathbf{y_s}^T\mathbf{Q_{ss}}^{-1}\mathbf{y_s}}.$$

Given that $\mathbf{Q_{ss}}$ is non-singular, and, therefore, positive definite, a Cholesky decomposition can be formed $\mathbf{Q_{ss}} = \mathbf{R}^T\mathbf{R}$, leading to the following expression

$$g_\beta = \frac{\mathbf{r_1}^T\mathbf{r_2} + v}{\mathbf{r_1}^T\mathbf{r_2}},$$

where

$$\mathbf{r_1} = \mathbf{R}^{-T}\mathbf{y_s}$$
$$\mathbf{r_2} = \mathbf{R}^{-T}\mathbf{u}$$

and, once we solve for g_β, we can obtain $\mathbf{h_{s1}}$

$$\mathbf{R}\mathbf{h_{s1}} = \mathbf{r_1}g_\beta - \mathbf{r_2}.$$

In the case where $\mathbf{Q_{ss}}$ is singular upon adding a new variable to the basis, we can alter the Cholesky factorization, subject to permutation, as follows,

$$\mathbf{Q_{ss}} = \begin{pmatrix}\mathbf{R}^T & \mathbf{0}\\ \mathbf{r}^T & 0\end{pmatrix}\begin{pmatrix}\mathbf{R} & \mathbf{r}\\ \mathbf{0} & 0\end{pmatrix}.$$

When forming the Cholesky factorization $\mathbf{Q_{ss}}$, if the last column is linearly dependent on the remaining columns, a zero will appear in the last diagonal entry. Note that $\mathbf{Q_{ss}}$ will never have more than a single zero eigenvalue since the basis matrix is guaranteed to be non-singular. With this new factorization, we solve the following

$$\begin{pmatrix}-\mathbf{R}^T\mathbf{R} & -\mathbf{R}^T\mathbf{r} & \mathbf{y_{s,r}}\\ -\mathbf{r}^T\mathbf{R} & -\mathbf{r}^T\mathbf{r} & y_{s,0}\\ \mathbf{y_{s,r}}^T & y_{s,0} & 0\end{pmatrix}\begin{pmatrix}\mathbf{h_{s1,r}}\\ h_{s1,0}\\ g_\beta\end{pmatrix} = \begin{pmatrix}\mathbf{u_r}\\ u_0\\ v\end{pmatrix}$$

where $y_{s,0}$ is the last component of $\mathbf{y_s}$ and $\mathbf{y_{s,r}}$ consists of the remaining components and similarly for u_0 versus $\mathbf{u_r}$ and $h_{s1,0}$ versus $\mathbf{h_{s1,r}}$. It is straight forward to show the following

$$g_\beta = \frac{\mathbf{r}^T\mathbf{r_1} - u_0}{\mathbf{r}^T\mathbf{r_2} - y_{s,0}}$$
$$h_{s1,0} = \frac{\mathbf{r_2}^T\mathbf{r_1} + v - \mathbf{r_2}^T\mathbf{r_2}g_\beta}{y_{s,0} - \mathbf{r_2}^T\mathbf{r}}$$
$$\mathbf{R}\mathbf{h_{s1,r}} = \mathbf{r_2}g_\beta - \mathbf{r_1} - \mathbf{r}h_{s1,0}$$

where $\mathbf{r_1}$ and $\mathbf{r_2}$ are defined as

$$\mathbf{r_1} = \mathbf{R}^{-T}\mathbf{u_r}$$
$$\mathbf{r_2} = \mathbf{R}^{-T}\mathbf{y_{s,r}}.$$

As the active set method progresses, a single row and column are added to the basis matrix at a time. As a result, rather than recalculating the Cholesky factorization, a rank-one update of an existing factorization can be performed as described in [1] and [10].

III. Experiments

In this research, we chose to compare the revised simplex method (SVM-RSQP) with the active set method implemented by Scheinberg (SVM-QP) and two state-of-the-art SVM training algorithms, SVMLight and LIBSVM.

The active set methods tend to provide more accurate solutions than the decomposition methods. During testing, our observation was that raising the tolerance did little to change the accuracy of the overall solution. As a result, we set the accuracy tolerance for all algorithms to 10^{-6} to enable a fair comparison between the algorithms, although a much smaller tolerance, typically 10^{-3} may be sufficient for classification accuracy. All algorithms employ shrinking and 500 MBytes for kernel caching. SVM-QP was downloaded from https://projects.coin-or.org/SVM-QP and converted from Fortran to C++ to allow a fair comparison with SVM-RSQP, also implemented in C++. Both SVM-QP and SVM-RSQP were configured to use sprinting [1]. The following datasets were used for comparison: **adult-1a** obtained from http://research.microsoft.com/ jplatt/smo.html, **abalone**, **letter-g**, **spam**, and **splice** obtained from UCI [11], and **ocr-0** and **ocr-9** obtained from the United States Postal Service (USPS) OCR dataset [12].

In **adult-1a**, 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. For the **abalone** dataset, age was thresholded at 10 rings (1.5 rings per year) to create a binary classification problem, The **letter** dataset from UCI was converted to a binary classification problem by choosing to classify the letter "G" from the remaining letters to create the **letter-g** data set and similarly, **ocr-0** and **ocr-9** were modifications of the **ocr** dataset to detect the digit "0" or "9" from the remaining digits. For all of the datasets, all numerical attributes were scaled between -1 and 1.

Each dataset was tested using both the RBF kernel and the linear kernel with an appropriate range of values for C and for the RBF kernel parameter, γ, where the RBF kernel is defined as the classical Gaussian kernel, $exp(-\gamma\|x_i - x_j\|_2^2)$. In practice, it is not uncommon to perform training across a range of parameters (grid search) to find the parameters yielding the highest test performance. In light of this, we compare the total training time for the set of parameters. The range of parameters were not optimized for each of the individual data sets, and, therefore, results might differ if apriori knowledge is included in the grid search. However, we also compare training times for the optimal set of parameters discovered during the grid search. Table I gives the range of parameters employed for each of the kernel types.

We do not report on training or testing accuracy since, unlike many other neural network technolo-

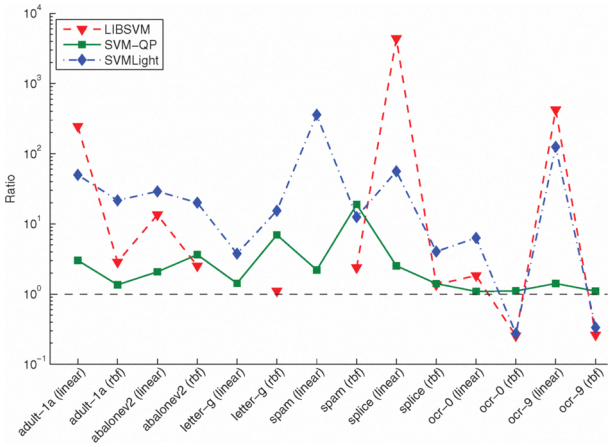

Fig. 1: Ratio of the total training time. The numerator of this ratio is the total training time of the specific algorithm used (LIBSVM, SVM-QP, or SVM Light) while the denominator of this ratio is the total training time for the SVM-RSQP algorithm. Consequently, the ratio corresponding to the SVM-RSQP algorithm is shown as a flat curve at the level 1.0. A missing symbol corresponding to an algorithm indicates that the corresponding algorithm did not converge properly (e.g., see LIBSVM for the letter-g dataset). The ratio scale is logarithmic.

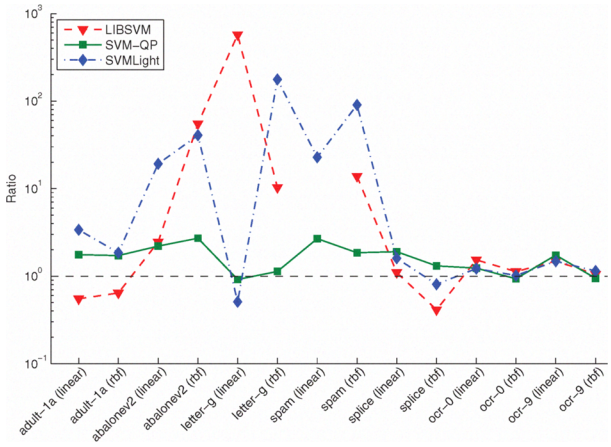

Fig. 2: Ratio of the training time for optimal settings. The numerator of this ratio is the time required to run the algorithm (LIBSVM, SVM-QP, or SVMLight) with its optimal parameter settings, while the denominator of this ratio is the time required to run the SVM-RSQP algorithms with its optimal parameter settings. Consequently, this ratio for the SVM-RSQP algorithm is a straight line at the level of 1.0. The ratio scale is logarithmic.

gies, different training algorithms do not yield different solutions in terms of the separating hyperplane and margin width. The convexity of the optimization problem provides for a single, global minimum (albeit semidefiniteness may yield multiple solutions for the same minimum). Since various optimization algorithms will only vary in their numerical stability, convergence properties, and precision, we can readily compare performance by directly observing the final solutions along with training times.

TABLE I: Range of testing parameters for C and γ

Kernel	Range of Parameters
linear	$log_2(C) = [-3 : 2 : 13]$
rbf	$log_2(C) = [-5 : 2 : 17], log_2(\gamma) = [-5 : -2 : -15]$

Figure 1 depicts the total training time required for performing a grid search while Figure 2 depicts the training time for the optimal settings yielding the best generalization performance.

The first observation we make is that LIBSVM fails to converge when performing the grid search on the **letter-g** and **spam** datasets when applying the linear kernel and fails to converge, with the optimal setting for C and the linear kernel for the **spam** dataset. In the case of **letter-g**, with a linear kernel and $C = 8$, we note LIBSVM takes on the order of 10,000 seconds to converge. For **spam**, LIBSVM did not converge properly for a value of C greater than 2.0 although the training time at $C = 2.0$ is 1.64 seconds.

Of course, restricting the parameter search space during the grid search may alleviate issues with convergence for these datasets and allowing a lower termination criterion may help with the **spam** and **letter-g** datasets for the optimal settings. However, this shows that the active set methods do, indeed, provide increased accuracy, speed, and numerical stability when compared to SMO.

SVM-RSQP performs worse than both SVMLight and LIBSVM for the **ocr-0** and **ocr-9** datasets, with the rbf kernel, in terms of total training time. This poor performance occurs when the number of non-bound support vectors begins to increase dramatically at $\gamma = 2^{-5}$. In this situation, the time taken per iteration is dominated by the solution of the inner problem rather than the pricing step, which is normally the case. A similar scenario appears to occur for the **adult-1a** and **splice** data sets when training with the optimal parameters. Overall, it appears that SVM-RSQP is never more than a factor of about 3 to 4 times slower than LIBSVM and SVMLight for the chosen data sets and chosen tolerance while it is often orders of magnitude faster in other cases.

When comparing SVM-RSQP with SVM-QP, we observe SVM-RSQP performs better or is comparable

to SVM-QP in terms of training time. SVM-RSQP is approximately 10 times faster than SVM-QP for the **spam** and **letter-g** datasets when using the rbf kernel. Looking closer at the **spam** dataset, we notice that SVM-RSQP is typically no more than 1 to 2 times faster than SVM-QP with the exception of two cases for high values of C where SVM-QP failed to properly converge and eventually terminates after failing to converge within 27,000 iterations. A similar phenomenon occurs for the **letter-g** dataset for $C \geq 512$ and $\gamma \leq 3 \times 10^{-5}$ where SVM-QP fails to converge within a large number of expended iterations. On average, however, SVM-RSQP appears to be no more than 3 times faster for the remaining cases. For the optimal settings, SVM-RSQP is no more than 3 times faster, on average, and has comparable performance with SVM-QP in the worst case.

TABLE II: Performance for Forest Covertype

Algorithm	Time	Iter	NSV[a]	NBSV[b]	bias
LIBSVM	31,396	359,336,873	38021	36772	-11.167
SVMLight	-	-	-	-	-[c]
SVM-QP	12,145	166,961	38051	36765	-11.825594
SVM-RSQP	6,839	117,331	38053	36767	-11.414945

[a] Number of support vectors
[b] Number of bound support vectors
[c] SVMLight failed to converge
[d] Since we are comparing numerical optimization techniques training/testing accuracy is not reported

We are also interested in the performance of the new algorithm for large datasets. Here, we chose to explore performance using the **Forest Covertype** dataset. We modified the dataset to extract types 1 and 2 for a binary classification problem (these types are the overwhelming majority of instances within the data) and 50,000 samples were randomly drawn from each class to create a total of 100,000 data points. A grid search was performed to find the optimal settings for C and the kernel (RBF kernel) and those settings were applied here. These results are reported in Table II. We note that SVM-RSQP is faster than the remaining algorithms including the SVM-QP algorithm. In the case of SVM-QP, we see that SVM-RSQP is taking a considerable fewer number of iterations to converge, suggesting that the singularities encountered by SVM-QP may be creating some inefficiencies in terms of convergence.

In summary, active set methods such as SVM-RSQP provide more accurate solutions and can outperform decomposition algorithms in certain scenarios as observed, here. However, the active set methods must store a factorization which has a memory requirement of $O(n_s^2)$, where n_s is the number of non-bound support vectors. This can be of concern for very large datasets where the number of non-bound support vectors can be large, even if the fraction is small. In addition, both active set methods show reduced performance when compared to the decomposition methods in scenarios where the number of non-bound support vectors are a large fraction of the total number of support vectors, although this was minimal for the cases considered. These cases often represent scenarios where reduced generalization performance will occur.

SVM-RSQP, however, tends to exhibit slightly better efficiency in terms of convergence when compared to SVM-QP and appears to be more numerically stable in some scenarios where SVM-QP fails to converge. While in some cases SVM-RSQP does not appear to have much advantage from a practical standpoint, SVM-RSQP is slightly easier to implement as the additional cases where singularities occur do not have to be handled. An additional advantage is that SVM-RSQP offers the possibility of implementing a simpler iterative method to alleviate the memory concerns.

IV. Conclusions

In this work, we derived a new, efficient active set algorithm for SVM training based upon the Revised Simplex method for quadratic programming introduced by Rusin [8] which provides a guarantee of non-singularity of the basis matrix solved during each iteration. This is unlike existing active set methods which must contend with the singularities. These algorithms have successfully demonstrated they can deal with these singularities in a practical manner and still provide proofs of convergence; however, there may still exist some level of inefficiency as we demonstrated when comparing training times between SVM-RSQP and SVM-QP. In addition, our new algorithm algorithm is slightly easier to implement as it no longer contains the logic for detecting singularities and producing a prescribed infinite descent direction (we still detect the single zero eigenvalue case, but the solution is still finite and other methods could be employed to obviate this as well).

We have shown that SVM-RSQP is competitive with SVMLight and LIBSVM, at least with the settings and datasets chosen. Of course, from a practical standpoint, data scaling, apriori selection of the range of parameters for C and γ, and proper selection of the stopping criterion can be employed to improve the timing for LIBSVM. For example, lowering the tolerance for the **Forest Covertype** dataset does not sacrifice training and testing accuracy while providing training times commensurate with SVM-RSQP. However, SVM-RSQP still remains more robust from a numerical optimization standpoint with less likelihood of running into scenarios where convergence fails.

While the active set methods offer improved precision and incremental training, they do not solve the memory constraint issue for which the decomposition methods were originally invented [13]. However, for very large datasets, an iterative method, such as conjugate gradient, could be employed to solve the inner

problem. The inner problem associated with the active set methods are indefinite, but in our case are non-singular. As a result, our algorithm offers the possibility of applying a null-space method to create a positive definite system suitable for the conjugate gradient method. Overall, SVM-RSQP should offer more possibilities than other active set methods in terms of implementing an iterative method to address memory requirements, a topic of future research.

References

[1] K. Scheinberg, "An efficient implementation of an active set method for svms," *Journal of Machine Learning Research*, vol. 7, pp. 2237–2257, December 2006.

[2] 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.

[3] S. V. N. Vishwanathan, A. J. Smola, and M. N. Murty, "Simplesvm," in *ICML*, 2003, pp. 760–767.

[4] O. L. Mangasarian and D. R. Musicant, "Active support vector machine classification," in *NIPS*, 2000, pp. 577–583.

[5] J. Platt, "Making large-scale support vector machine learning practical," in *Advances in Kernel Methods: Support Vector Machines*, A. S. B. Schölkopf, C. Burges, Ed. MIT Press, Cambridge, MA, 1998.

[6] T. Joachims, "Making large-scale support vector machine learning practical," in *Advances in Kernel Methods: Support Vector Machines*, A. S. B. Schölkopf, C. Burges, Ed. MIT Press, Cambridge, MA, 1998.

[7] J. Nocedal and S. J. Wright, *Numerical Optimization*, 1st ed., ser. Operations Research. Springer, 1999.

[8] M. H. Rusin, "A revised simplex method for quadratic programming," *SIAM Journal on Applied Mathematics*, vol. 20, no. 2, pp. 143–160, March 1971.

[9] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*. Cambridge University Press, 2000.

[10] G. H. Golub and C. F. V. Loan, *Matrix Computations*, 3rd ed. Baltimore and London: The John Hopkins University Press, 1996.

[11] A. Asuncion and D. Newman, "UCI machine learning repository," 2007. [Online]. Available: http://www.ics.uci.edu/~mlearn/MLRepository.html

[12] J. J. Hull, "A database for handwritten text recognition research," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 5, pp. 550–554, May 1994.

[13] E. Osuna, R. Freund, and F. Girosi, "Training support vector machines: An application to face detection," in *CVPR*. San Juan, Puerto Rico: 1997 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'97), June 1997, pp. 130–136.