Finite Newton Method for Lagrangian Support Vector Machine Classification

Glenn Fung & O. L. Mangasarian
Computer Sciences Department
University of Wisconsin
1210 West Dayton Street
Madison, WI 53706
gfung,olvi@cs.wisc.edu

Abstract

An implicit Lagrangian [19] formulation of a support vector machine classifier that led to a highly effective iterative scheme [18] is solved here by a finite Newton method. The proposed method, which is extremely fast and terminates in 6 or 7 iterations, can handle classification problems in very high dimensional spaces, e.g. over 28,000, in a few seconds on a 400 MHz Pentium II machine. The method can also handle problems with large datasets and requires no specialized software other than a commonly available solver for a system of linear equations. Finite termination of the proposed method is established in this work.

Keywords *classification, Lagrangian support vector machines, Newton method, Myeloma*

1 Introduction

This paper establishes finite termination of a Newton method for the unconstrained minimization of a strongly convex, piecewise quadratic function, underlying a linear or nonlinear kernel classifier [27, 3, 15, 4, 26]. Our piecewise quadratic function is the implicit Lagrangian, first proposed in [19] and utilized in [18] for a highly effective iterative scheme, the Lagrangian Support Vector Machine (LSVM). In order to handle problems with very large dimensional input spaces, we use here instead a fast finite Newton method for finding the unconstrained unique global minimum solution of the implicit Lagrangian associated with the classification problem. The solution is obtained by solving a system of linear equations a finite number of times.

We outline now the contents of the paper. In Section 2 we describe the linear and nonlinear kernel classification problems leading to a dual problem consisting of minimizing a strongly convex function of nonnegative variables. In Section 3 we give our implicit Lagrangian formulation which consists of minimizing an unconstrained strongly convex piecewise quadratic function and establish some properties of the implicit Lagrangian. In Section 4 we give our Newton method with an Armijo stepsize and establish its finite global termination to the unique solution. In Section 5 we give some computational results including those for a gene expression problem with 28,032 variables, solved in seconds, as well as other publicly available datasets.

A word about our notation. All vectors will be column vectors unless transposed to a row vector by a prime superscript $'$. For a vector x in the n-dimensional real space R^n, the *plus function* x_+ is defined as $(x_+)_i = \max\ \{0, x_i\}$, $i = 1, \ldots , n$, while x_* denotes the subgradient of x_+ which is the step function defined as $(x_*)_i = 1$ if $x_i > 0$, $(x_*)_i = 0$ if $x_i < 0$, and $(x_*)_i \in [0, 1]$ if $x_i = 0$, $i = 1, \ldots , n$. Thus, $(x_*)_i$ is any value in the interval $[0, 1]$, when $x_i = 0$ we typically take $(x_*)_i = 0.5$ The scalar (inner) product of two vectors x and y in the n-dimensional real space R^n will be denoted by $x'y$ and the 2-norm of x will be denoted by $\|x\|$. For a matrix $A \in R^{m\times n}$, A_i is the ith row of A which is a row vector in R^n and $\|A\|$ is the 2-norm of A: $\max\limits_{\|x\|=1} \|Ax\|$. A column vector of ones of arbitrary dimension will be denoted by e and the identity matrix of arbitrary order will be denoted by I. For $A \in R^{m\times n}$ and $B \in R^{n\times l}$, the kernel $K(A, B)$ [27, 3, 15] is an arbitrary function which maps $R^{m\times n} \times R^{n\times l}$ into $R^{m\times l}$. In particular, if x and y are column vectors in R^n then, $K(x', y)$ is a real number, $K(x', A')$ is a row vector in R^m and $K(A, A')$ is an $m \times m$ matrix. If f is a real valued function defined on the n-dimensional real space R^n, the gradient of f at x is denoted by $\nabla f(x)$ which is a column vector in R^n and the $n \times n$ matrix of second partial derivatives of f at x is denoted by $\nabla^2 f(x)$. For a piecewise quadratic function such as, $f(x) = \frac{1}{2}||(Ax - b)_+||^2 + \frac{1}{2}x'Px$, where $A \in R^{m\times n}$, $P \in R^{n\times n}$, $P = P'$, P positive semidefinite and $b \in R^m$, the ordinary Hessian does not exist because its gradient, the $n \times 1$ vector $\nabla f(x) = A'(Ax - b)_+ + Px$, is not differentiable. However, one can define its **generalized Hessian** [9, 5, 16] which is the $n \times n$ symmetric positive semidefinite matrix:

$$\partial^2 f(x) = A' diag(Ax - b)_* A + P, \tag{1}$$

where $diag(Ax-b)_*$ denotes an $m \times m$ diagonal matrix with diagonal elements $(A_i x - b_i)_*$, $i = 1, \ldots, m$. The generalized Hessian (1) has many of the properties of the regular Hessian [9, 5, 16] in relation to $f(x)$. If the smallest eigenvalue of $\partial f(x)$ is greater than some positive constant for all $x \in R^n$, then $f(x)$ is a strongly convex piecewise quadratic function on R^n. Throughout this work, the notation := will denote definition.

2 Linear and Nonlinear Kernel Classification

We describe in this section the fundamental classification problems that lead to minimizing a piecewise quadratic strongly convex function. We consider the problem of classifying m points in the n-dimensional real space R^n, represented by the $m \times n$ matrix A, according to membership of each point A_i in the classes +1 or -1 as specified by a given $m \times m$ diagonal matrix D with ones or minus ones along its diagonal. For this problem, the standard support vector machine with a linear kernel AA' [27, 3] is given by the following quadratic program for some $\nu > 0$:

$$\begin{array}{lrcl} \displaystyle\min_{(w,\gamma,y)\in R^{n+1+m}} & \nu e'y + \frac{1}{2}w'w & & \\ \text{s.t.} & D(Aw - e\gamma) + y & \geq & e \\ & y & \geq & 0. \end{array} \tag{2}$$

As depicted in Figure 1, w is the normal to the bounding planes:

$$\begin{array}{ccccc} x'w & - & \gamma & = & +1 \\ x'w & - & \gamma & = & -1, \end{array} \tag{3}$$

and γ determines their location relative to the origin. The first plane above bounds the class +1 points and the second plane bounds the class -1 points when the two classes are strictly linearly separable, that is when the slack variable $y = 0$. The linear separating surface is the plane

$$x'w = \gamma, \tag{4}$$

midway between the bounding planes (3). If the classes are linearly inseparable then the two planes bound the two classes with a "soft margin" determined by a nonnegative slack variable y, that is:

$$\begin{array}{ccccccccl} x'w & - & \gamma & + & y_i & \geq & +1, & \text{for } x' = A_i \text{ and } D_{ii} = +1, \\ x'w & - & \gamma & - & y_i & \leq & -1, & \text{for } x' = A_i \text{ and } D_{ii} = -1. \end{array} \tag{5}$$

The 1-norm of the slack variable y is minimized with weight ν in (2). The quadratic term in (2), which is twice the reciprocal of the square of the 2-norm distance $\frac{2}{\|w\|}$ between the two bounding planes of (3) in the n-dimensional space of $w \in R^n$ for a *fixed* γ, maximizes that distance, often called the "margin". Figure 1 depicts the points represented by A, the bounding planes (3) with margin $\frac{2}{\|w\|}$, and the separating plane (4) which separates $A+$, the points represented by rows of A with $D_{ii} = +1$, from $A-$, the points represented by rows of A with $D_{ii} = -1$.

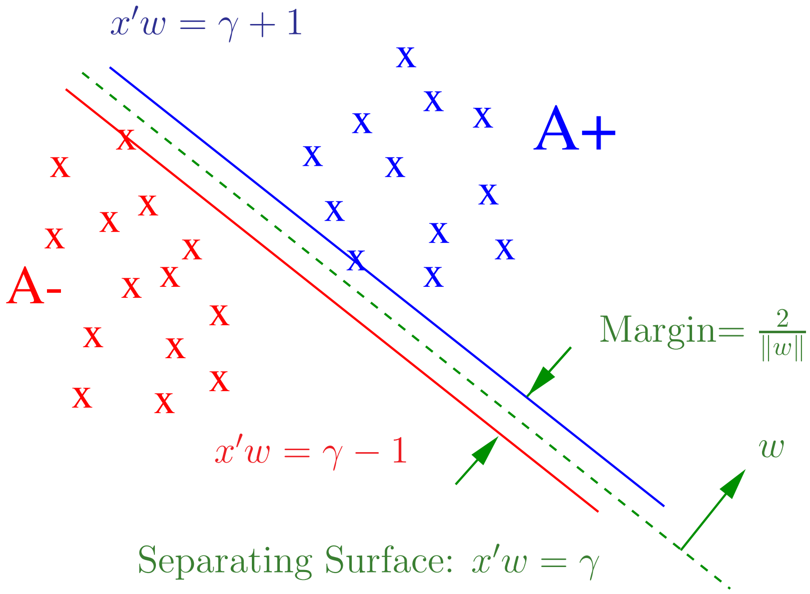

Figure 1: **The bounding planes (3) with margin $\frac{2}{\|w\|}$, and the plane (4) separating $A+$, the points represented by rows of A with $D_{ii} = +1$, from $A-$, the points represented by rows of A with $D_{ii} = -1$.**

In many essentially equivalent formulations of the classification problem [12, 11, 7, 6], the square of 2-norm of the slack variable y is minimized with weight $\frac{\nu}{2}$ instead of the 1-norm of y as in (2). In addition the distance between the planes (3) is measured in the $(n+1)$-dimensional space of $(w, \gamma) \in R^{n+1}$, that is $\frac{2}{\|(w,\gamma)\|}$. Measuring the margin in this $(n+1)$-dimensional space instead of R^n induces strong convexity and has little or no effect in general on the problem as was shown in [17] experimentally. Thus using twice the reciprocal squared of the margin instead, yields our modified SVM problem as follows:

$$
\begin{array}{rrcl}
\min\limits_{(w,\gamma,y)\in R^{n+1+m}} & \frac{\nu}{2}y'y+\frac{1}{2}(w'w+\gamma^2) & & \\
\text{s.t.} & D(Aw-e\gamma)+y & \geq & e \\
 & y & \geq & 0.
\end{array}
\tag{6}
$$

It has been shown computationally [18] that this reformulation (6) of the conventional support vector machine formulation (2) often yields similar results to (2). The dual of this problem is [13]:

$$
\min_{0\leq u\in R^m} \frac{1}{2}u'(\frac{I}{\nu}+D(AA'+ee')D)u-e'u. \tag{7}
$$

The variables (w,γ) of the primal problem which determine the separating surface (4) are recovered directly from the solution of the dual (7) above by the relations:

$$
w=A'Du,\ \ y=\frac{u}{\nu},\ \ \gamma=-e'Du. \tag{8}
$$

We immediately note that the matrix appearing in the dual objective function is positive definite. We simplify the formulation of the dual problem (7) by defining two matrices as follows:

$$
H=D[A\quad -e],\qquad Q=\frac{I}{\nu}+HH'. \tag{9}
$$

With these definitions the dual problem (7) becomes

$$
\min_{0\leq u\in R^m} f(u):=\frac{1}{2}u'Qu-e'u. \tag{10}
$$

To develop the nonlinear kernel classifier we use the notation of [15]. For $A\in R^{m\times n}$ and $B\in R^{n\times \ell}$, the **kernel** $K(A,B)$ maps $R^{m\times n}\times R^{n\times\ell}$ into $R^{m\times\ell}$. A typical kernel is the Gaussian kernel $\varepsilon^{-\mu\|A_i'-B_{\cdot j}\|^2}$, $i,j=1,\ldots,m$, $\ell=m$, where ε is the base of natural logarithms, while a linear kernel is $K(A,B)=AB$. For a column vector x in R^n, $K(x',A')$ is a row vector in R^m, and the linear separating surface (4) is replaced by the nonlinear surface:

$$
K(x',A')Du=\gamma, \tag{11}
$$

where u is the solution of the dual problem (7) with the linear kernel AA' replaced by the nonlinear kernel product $K(A,A')K(A,A')'$ [15, Equation (8.10)], that is:

$$
\min_{0\leq u\in R^m} \frac{1}{2}u'(\frac{I}{\nu}+D(K(A,A')K(A,A')'+ee')D)u-e'u. \tag{12}
$$

This leads to a redefinition of the matrix Q of (10) as follows:

$$H = D[K(A, A') \quad -e], \qquad Q = \frac{I}{\nu} + HH'. \tag{13}$$

Note that the nonlinear separating surface (11) degenerates to the linear one (4) if we let $K(A, A') = AA'$ and make use of (8).

We describe now a general framework for generating a fast and effective algorithm for solving the quadratic program (10) by solving a system of linear equations a finite number of times.

3 Implicit Lagrangian Formulation

The implicit Lagrangian formulation [18, Equation (17)], [26, Section 10.6.2] consists of replacing the nonnegativity constrained quadratic minimization problem (10) by the equivalent *unconstrained* piecewise quadratic minimization of the implicit Lagrangian $L(u)$:

$$\min_{u \in R^m} L(u) = \min_{u \in R^m} \frac{1}{2}u'Qu - e'u + \frac{1}{2\alpha}(\|(-\alpha u + Qu - e)_+\|^2 - \|Qu - e\|^2), \tag{14}$$

where α is a sufficiently large but finite positive parameter, and the plus function $(\cdot)_+$, defined in the Introduction, replaces negative components of a vector by zeros. Reformulation of the constrained problem (10) as an unconstrained problem (14) is based on ideas [19] of converting the optimality conditions of (10) to an unconstrained minimization problem as follows. Because the Lagrange multipliers of the constraints $u \geq 0$ of (10) turn out to be components of the gradient $Qu - e$ of the objective function, these components of the gradient can be used as Lagrange multipliers in an Augmented Lagrangian [25, 2] formulation of (10) which leads precisely to the unconstrained formulation (14). Our finite Newton method consists of applying Newton's method to this unconstrained minimization problem and showing that it terminates in a finite number of steps at the global minimum. The gradient of $L(u)$ is:

$$\begin{array}{rcl} \nabla L(u) & = & (Qu - e) + \frac{1}{\alpha}(Q - \alpha I)((Q - \alpha I)u - e)_+ - \frac{1}{\alpha}Q(Qu - e) \\ & = & \frac{(\alpha I - Q)}{\alpha}((Qu - e) - ((Q - \alpha I)u - e)_+). \end{array} \tag{15}$$

In order to apply the Newton method we need the $m \times m$ Hessian matrix of second partial derivatives of $L(u)$, which does not exist in the ordinary sense because its gradient, $\nabla L(u)$, is not differentiable. However, a generalized Hessian of $L(u)$ in the sense of [9, 5, 16] exists and is defined as the following $m \times m$ matrix:

$$\partial^2 L(u) = \frac{(\alpha I - Q)}{\alpha}(Q + diag((Q - \alpha I)u - e)_*(\alpha I - Q)), \tag{16}$$

where, as defined in the Introduction, $diag(\cdot)_*$ denotes a diagonal matrix and $(\cdot)_*$ denotes the step function. Our basic Newton step consists of solving the system of m linear equations:

$$\nabla L(u^i) + \partial^2 L(u^i)(u^{i+1} - u^i) = 0, \tag{17}$$

for the unknown $m \times 1$ vector u^{i+1} given a current iterate u^i. We will show in the next section that this iteration coupled with a stepsize, terminates at the global minimum solution to the problem. For that we need the positive definiteness of $\partial^2 L(u)$, which we establish now under the very simple condition that Q is positive definite and that $\alpha > ||Q||$ as follows.

Proposition 3.1 Positive Definiteness of the Generalized Hessian *Let Q be an arbitrary $m \times m$ symmetric positive definite matrix and let $\alpha > ||Q||$. Then:*

(i) The generalized Hessian matrix $\partial^2 L(u)$, defined by (16), is positive definite.

(ii) The generally non-symmetric matrix factor $P(u)$ of $\partial^2 L(u) = \frac{(\alpha I - Q)}{\alpha} P(u)$:

$$P(u) := Q + diag((Q - \alpha I)u - e)_*(\alpha I - Q), \tag{18}$$

is positive definite.

Proof

(i) Since $\alpha > ||Q||$, it follows that $\alpha I - Q$ is positive definite. Hence the product $(\alpha I - Q)Q$ of the two positive definite matrices $\alpha I - Q$ and Q is positive definite [23, Theorem 6.2.1]. Define the diagonal matrix:

$$E(u) := diag((Q - \alpha I)u - e)_*. \tag{19}$$

Since, by the definition of the step function $(\cdot)_*$, each element of $E(u)$ is in the interval $[0, 1]$, it follows that:

$$(\alpha I - Q)E(u)(\alpha I - Q) = (E(u)^{\frac{1}{2}}(\alpha I - Q))^2 \tag{20}$$

is positive semidefinite. Hence the generalized Hessian:

$$\partial^2 L(u) = \frac{(\alpha I - Q)}{\alpha} Q + \frac{(\alpha I - Q)}{\alpha} E(u)(\alpha I - Q), \tag{21}$$

the sum of a positive definite and a positive semidefinite matrix is positive definite.

(ii) Since:

$$P(u) = \alpha(\alpha I - Q)^{-1} \partial^2 L(u), \tag{22}$$

is the product of two positive definite matrices, it follows again by [23, Theorem 6.2.1] that $P(u)$ is positive definite. □

We note now that, since both $\nabla L(u)$ and $\partial^2 L(u)$ contain the multiplicative factor $\frac{(\alpha I - Q)}{\alpha}$ which is positive definite, it follows that the Newton iteration (17) can be simplified to:

$$h(u^i) + \partial h(u^i)(u^{i+1} - u^i) = 0, \tag{23}$$

where

$$h(u) := (Qu - e) - ((Q - \alpha I)u - e)_+ = (\frac{\alpha I - Q}{\alpha})^{-1} \nabla L(u), \tag{24}$$

and

$$\partial h(u) := Q + E(u)(\alpha I - Q) = P(u) = (\frac{\alpha I - Q}{\alpha})^{-1} \partial^2 L(u). \tag{25}$$

The simpler iteration (23) will be used in our implementation instead of the equivalent iteration (17).

Another useful tool in our implementation will be the Sherman-Morrison-Woodbury identity [8] when we are classifying large datasets with a *linear* classifier. For such problems we have, with Q defined by (9) and $E(u)$ replaced by E for notational simplicity:

$$\begin{aligned} \partial h(u) &= \alpha E + (I - E)Q = \alpha E + \tfrac{I-E}{\nu} + (I - E)HH' \\ &= F + (I - E)HH' = F(I + F^{-1}(I - E)HH') = F(I + SHH'), \end{aligned} \tag{26}$$

where F and S are defined as the following positive and nonnegative diagonal matrices respectively:

$$F := \alpha E + \frac{I - E}{\nu}, \quad S := F^{-1}(I - E). \tag{27}$$

By using a special case of the Sherman-Morrison-Woodbury identity [8]:

$$(I + KH')^{-1} = I - K(I + H'K)^{-1}H', \tag{28}$$

on the last expression of (26), with $K := SH$, we have:

$$\partial h(u)^{-1} = (I + SHH')^{-1}F^{-1} = (I - SH(I + H'SH)^{-1}H')F^{-1}, \tag{29}$$

where we need to invert the $(n+1) \times (n+1)$ matrix $(I + H'SH)$ instead of the potentially much larger $m \times m$ matrix $(I + SHH')$. This will be the case whenever we generate a linear classifier for problems with $m >> n$.

We turn now to details of the Newton algorithm and its finite termination properties.

4 Finite Newton Classification Method

We first state our Newton algorithm for solving the piecewise quadratic minimization problem (14) for an arbitrary positive definite Q using the simplified iteration (23) together with an Armijo stepsize [1, 12] defined below in order to guarantee finite termination from any starting point.

Algorithm 4.1 Newton Algorithm for (14) *Let $h(u)$ and $\partial h(u)$ be defined by (24) and (25). Start with any $u^0 \in R^m$. For $i = 0, 1, \ldots$:*

(i) Stop if $h(u^i - \partial h(u^i)^{-1}h(u^i)) = 0$.

(ii) $u^{i+1} = u^i - \lambda_i \partial h(u^i)^{-1}h(u^i) = u^i + \lambda_i d^i$,
where $\lambda_i = \max\{1, \frac{1}{2}, \frac{1}{4}, \ldots\}$ is the Armijo stepsize such that:

$$L(u^i) - L(u^i + \lambda_i d^i) \geq -\delta\lambda_i \nabla L(u^i)' d^i, \tag{30}$$

for some $\delta \in (0, \frac{1}{2})$, and d^i is the Newton direction:

$$d^i = -\partial h(u^i)^{-1}h(u^i), \tag{31}$$

obtained by solving (23).

(iii) $i = i + 1$. Go to (i).

We state and prove now our finite termination result for this Newton algorithm. A possible intuitive justification of the finite termination is that as the iterates converge to the unique global solution, the correct quadratic surfaces on whose intersection the solution lies are correctly identified, and hence a single Newton step captures that solution.

Theorem 4.2 Finite Termination of Newton Algorithm *For the symmetric positive definite matrix Q defined by (9) or (13), the sequence $\{u^i\}$ of Algorithm 4.1 terminates at the global minimum solution $\bar{u}$ of (14) and hence that of (10) provided $\alpha > \|Q\|$.*

Proof That the sequence $\{u^i\}$ converges to the global solution $\bar{u}$, for which $h(\bar{u}) = 0$, follows from standard results (e.g.[14, Theorem 2.1, Example 2.1(ii), Example 2.4(iv)]) of unconstrained minimization of a strongly convex function using a Newton direction with an Armijo stepsize. This is exactly what is done in our Algorithm 4.1 above. We now establish finite termination of the sequence $\{u^i\}$ at $\bar{u}$. Our Newton iteration (23) can be written as:

$$(Qu^i - e) - ((Q - \alpha I)u^i - e)_+ + (Q + E(u^i)(\alpha I - Q))(u^{i+1} - u^i) = 0, \tag{32}$$

which we rewrite by subtracting from it the equality $h(\bar{u}) = 0$ satisfied by the solution $\bar{u}$:

$$(Q\bar{u} - e) - ((Q - \alpha I)\bar{u} - e)_+ = h(\bar{u}) = 0. \tag{33}$$

This results in the equivalent iteration:

$$\begin{array}{c} ((Q - \alpha I)\bar{u} - e)_+ - ((Q - \alpha I)u^i - e)_+ \\ -E(u^i)(Q - \alpha I)(u^{i+1} - u^i) + Q(u^{i+1} - \bar{u}) = 0. \end{array} \tag{34}$$

We show now that this Newton iteration is satisfied uniquely (since $\partial h(u^i)$ is nonsingular) by $u^{i+1} = \bar{u}$ when u^i is sufficiently close to $\bar{u}$ and hence the Newton iteration terminates at $\bar{u}$ at step (i) of Algorithm 4.1. Setting $u^{i+1} = \bar{u}$ in (34) and canceling the last term $Q(\bar{u} - \bar{u}) = 0$, gives:

$$((Q - \alpha I)\bar{u} - e)_+ - ((Q - \alpha I)u^i - e)_+ - E(u^i)(Q - \alpha I)(\bar{u} - u^i) = 0. \tag{35}$$

We verify now that that this equation is indeed satisfied when u^i is sufficiently close to $\bar{u}$ by looking at each component j, $j = 1, \dots, m$ of the equation (35). We consider the the following nine possible combinations determined by the vector function $r(u)$ appearing in (35) defined as:

$$r_j(u) := ((Q - \alpha I)u - e)_j, \quad j = 1, \dots, m. \tag{36}$$

Noting that every element of the diagonal matrix $E(u)$ defined by (19), that is $E_{jj}(u) = (r_j(u))_*$, $j = 1, \dots, m$, is in the interval $[0, 1]$, we have:

(i) $r_j(\bar{u}) > 0,\ r_j(u^i) > 0$:
$(Q_j - \alpha I_j)\bar{u} - 1 - (Q_j - \alpha I_j)u^i + 1 - 1 \cdot (Q_j - \alpha I_j)(\bar{u} - u^i) = 0$

(ii) $r_j(\bar{u}) > 0,\ r_j(u^i) = 0$: Cannot occur when u^i is sufficiently close to $\bar{u}$.

(iii) $r_j(\bar{u}) > 0,\ r_j(u^i) < 0$: Cannot occur when u^i is sufficiently close to $\bar{u}$.

(iv) $r_j(\bar{u}) = 0,\ r_j(u^i) > 0$:
$0 - (Q_j - \alpha I_j)u^i + 1 - 1 \cdot (Q_j - \alpha I_j)(\bar{u} - u^i) = 1 - Q_j\bar{u} + \alpha\bar{u}_j = -r_j(\bar{u}) = 0$

(v) $r_j(\bar{u}) = 0,\ r_j(u^i) = 0$:
$0 + 0 - [0,1] \cdot ((Q_j - \alpha I_j)(\bar{u} - u^i) - e + e) = [0,1] \cdot (r_j(\bar{u}) - r_j(u^i)) = 0$

(vi) $r_j(\bar{u}) = 0,\ r_j(u^i) < 0$:
$0 + 0 - 0 \cdot (Q_j - \alpha I_j)(\bar{u} - u^i) = 0$

(vii) $r_j(\bar{u}) < 0,\ r_j(u^i) > 0$: Cannot occur when u^i is sufficiently close to $\bar{u}$.

(viii) $r_j(\bar{u}) < 0,\ r_j(u^i) = 0$: Cannot occur when u^i is sufficiently close to $\bar{u}$.

(ix) $r_j(\bar{u}) < 0,\ r_j(u^i) < 0$:
$0 + 0 - 0 \cdot (Q_j - \alpha I_j)(\bar{u} - u^i) = 0$

Consequently for u^i is sufficiently close to $\bar{u}$, the Newton iteration is uniquely satisfied by $\bar{u}$, and hence, terminates at $\bar{u}$. $\square$

We turn now to our computational results.

5 Numerical Experience

Our numerical tests and comparisons were carried out on a dataset with a high dimensional input space and a moderate number of data points, as well as on more conventional datasets where the dimensionality of the input space is considerably smaller than the number of data points. All our computations were performed on the University of Wisconsin Data Mining Institute "locop1" machine, which utilizes a 400 Mhz Pentium II and allows a maximum of 2 Gigabytes of memory for each process. This computer runs on Windows NT server 4.0, with MATLAB 6 installed [20].

Because of the simplicity of our algorithm, we give below a simple MATLAB implementation of the algorithm without the Armijo stepsize, which does not seem to be needed in most applications. Although this is merely an empirical observation in the present case, it considerably simplifies our MATLAB Code 5.1. However, it has also been shown [16, Theorem 3.6] that under a well conditioned assumption, not generally satisfied here, the proposed Newton method indeed terminates in a finite number of steps without an Armijo stepsize.

Code 5.1 NSVM: Finite Newton LSVM Code

```
function [w,gamma]=nsvm(A,d,nu);
% NSVM:linear and nonlinear classification without Armijo
% INPUT: A, D, nu. OUTPUT: w, gamma
% [w, gamma] = nsvm(A,d,nu);
[m,n]=size(A);iter=0;
u=zeros(m,1);e=ones(m,1);
H=[diag(d)*A -d]; Q=speye(m)/nu+H*H';
alpha=1.1*((1/nu)+(norm(H',2)^2));
hu=-max(((Q*u-e)-alpha*u),0)+Q*u-e;
while norm(hu)>10^(-5)
    iter=iter+1
    star=sign(max(((Q-alpha*eye(m))*u-e),0));
    dhu=sparse((eye(m)-diag(star))*Q+alpha*diag(star));
    delta=dhu\hu;
    unew=u-delta;
    u=unew;
    hu=-max(((Q*u-e)-alpha*u),0)+Q*u-e;
end

 w=A'*(d.*u);gamma=-sum(d.*u);

return
```

We further note that the MATLAB code above not only works for a linear classifier, but also for a nonlinear classifier as well [15, Equations (1), (10)]. In the nonlinear case, the matrix $K(A, A')$ is used as input instead of A, and the pair $(\hat{u}, \gamma)$, where $\hat{u} = K(A, A')Du$, is returned instead of (w, γ). The nonlinear separating surface is then given by (11) as:

$$K(x, A')\hat{u} - \gamma = 0. \tag{37}$$

Our numerical testing and comparisons were carried out on the high dimensional Multiple Myeloma dataset available at:

$$http://lambertlab.uams.edu/publicdata.htm,$$

and processed by by David Page and his colleagues [24]. Further tests and comparisons were also carried out on six moderately dimensioned, publicly available datasets [21, 22].

We describe our tests and comparisons now.

5.1 Multiple Myeloma dataset

Multiple Myeloma is cancer of the plasma cell. The plasma cell normally produces antibodies that destroy foreign bodies such as bacteria. As a product of the Myeloma disease the plasma cells get out of control and produce a tumor. These tumors can grow in several sites, usually in the soft middle part of bone, the bone marrow. When these tumors appear in multiples sites they are called Multiple Myeloma. A detailed description of the process used to obtain the data can be found in [24].

5.1.1 Description of the dataset

The data consists of 105 data points, 74 of the points representing newly-diagnosed multiple Myeloma patients while 31 points represent 31 healthy donors. Each data point represents measurements taken from 7008 genes using plasma cells samples from the patients. For each one of the 7008 genes there are two measurements. One measurement is called Absolute Call (AC) and takes on one of three nominal values: A (Absent), M (Marginal) or P (Present). The other measurement, the average difference (AD), is a floating point number that can be either positive or negative. Since each one of the 7008 AC features takes on nominal values from the set $\{A, M, P\}$, a real valued representation is needed to utilize our classifier which requires an input of real numbers. Thus, each nominal value is mapped into a three dimensional binary vector depending on the value that is being represented. This simple and widely used "1 of N" mapping scheme for converting nominal attributes into real-valued attributes is illustrated in Figure 2. Once this simple conversion is applied to the dataset, the AC feature space is transformed from a 7008-dimensional space with nominal values A, M, P into a $7008 \times 3 = 21024$ real-valued dimensional space. Adding the numerical AD feature for each of the 7008 genes results in each data point being transformed to a point in R^{28032}, with 21024 coming from the AC transformation mentioned above and 7008 from the AD values. This makes this dataset very interesting for our method, since a main objective of this paper is to show that our proposed algorithm can quickly classify points in very high dimensional spaces.

$$\mathtt{A} \longrightarrow \boxed{\mathtt{1}}\boxed{\mathtt{0}}\boxed{\mathtt{0}}$$

$$\mathtt{M} \longrightarrow \boxed{\mathtt{0}}\boxed{\mathtt{1}}\boxed{\mathtt{0}}$$

$$\mathtt{P} \longrightarrow \boxed{\mathtt{0}}\boxed{\mathtt{0}}\boxed{\mathtt{1}}$$

Figure 2: **Real-valued representation of the AC features set** $\{A, M, P\}$ **.**

5.1.2 Numerical comparisons

Performance of our Newton SVM (**NSVM**) algorithm on the Myeloma dataset, in terms of speed and generalization ability, is first compared with two publicly available SVM solvers: LSVM [18] and SVM^{light} version 5.0 [10]. The comparison with LSVM was carried out because both LSVM and NSVM minimize the same unconstrained differentiable convex implicit Lagrangian function (14) but using entirely different techniques. Reported times for LSVM here differ from the ones reported in [18] because the calculation time for the matrix H of (9) is considered as input time in [18], whereas here it is counted as part of the computational time. The other algorithm included in our comparisons, SVM^{light}, solves a different optimization problem with a classification error measured using the 1-norm instead of the 2-norm. This solver was included in our experiments because it is widely cited in the literature and is often used as a benchmark for SVM classification algorithms. Termination criteria for all methods was set to 0.001 which is the default for SVM^{light}. We outline some of the results of our comparative testing.

- Both NSVM and SVM^{light} obtained 100% *leave-one-out correctness (looc)*. However, we note that SVM^{light} did not perform as well with the default value of its parameter C which determines the trade-off between empirical and generalization errors. In order to find an optimal value for both ν (NSVM and LSVM) and C (SVM^{light}) the following tuning procedure was employed on each fold:

- A random tuning set of the the size of 10% of the training data was chosen and separated from the training set.
- Several SVMs were trained on the remaining 90% of the training data using values for C or ν equal to 2^i where $i = -12, \dots, 0, \dots, 12$.
- The value of C or ν that gave the best SVM correctness on the tuning set was chosen.
- A final SVM was trained using the chosen value of C and ν and all the training data. The resulting SVM was tested on the testing data.

- The average cpu time required by our algorithm for the leave-one-out correctness (*looc*) computations was 4.11 seconds per case and total time for all cases was 432.40 seconds. This outperformed the SVM^{light} cpu times of 27.83 seconds average per case and total *looc* time of 2922.15 seconds.
- LSVM failed and reported an out of memory error.

These results are summarized in Table 1 below.

Table 1: **NSVM, SVM^{light} & LSVM: *leave-one-out correctness (looc)* and total running times using a linear classifier for the Myeloma dataset. Best results are in bold. oom stands for "out of memory".**

Data Set $m \times n$ (*points* $\times$ *dimensions*)	NSVM looc Time (Sec.)	SVM^{light} looc Time (Sec.)	LSVM looc Time (Sec.)
Myeloma 105 $\times$ 28032	**100.0%** **432.40**	**100.0%** 2922.15	oom oom

5.2 Six publicly available datasets

Even though our algorithm is primarily intended for datasets with very high dimensional input space and a moderate number of points, it also performed very well on more conventional datasets where the opposite is true. To exhibit this fact we tested our algorithm on six publicly available datasets.

Five from the UCI Machine Learning Repository [21]: Ionosphere, Cleveland Heart, Pima Indians, BUPA Liver and Housing. The sixth dataset is the Galaxy Dim dataset available at [22]. The dimensionality and size of each dataset is given in Table 2.

5.2.1 Numerical comparisons using a linear classifier

In this set of experiments we used a linear classifier to compare our method NSVM with LSVM and SVM^{light} on the six datasets mentioned above. Because $m >> n$ for these datasets, it was preferable to use (29) in solving the Newton iteration (23) and inverting an $(n+1) \times (n+1)$ matrix instead of an $m \times m$ matrix. The complexity of the original NSVM for a linear kernel is $O(km^3) + O(m^2n)$, where k is the number of iterations of NSVM and the term $O(m^2n)$ reflects the time for computing the matrix Q of of (9). By using the Sherman-Morrison-Woodbury formula, the complexity changes to $O(kn^3) + O(m^2n)$, which is obviously preferable when $m >> n$. However, the explicit calculation of the matrix Q of equation (9) which is of the order $O(m^2n)$ is very time consuming when m is large. Hence, instead of explicitly calculating Q and then performing the matrix-vector product Qu, needed in computing $h(u)$, we calculate:

$$Qu = (\frac{I}{\nu} + HH')u = \frac{1}{\nu}u + H(H'u), \tag{38}$$

where $H'u$ is calculated first then $H(H'u)$ is calculated next. This simple but effective algebraic manipulation changes the complexity of the algorithm to be linear in m that is, $O(kn^3) + O(mn)$. This makes our algorithm very fast even when $m >> n$ but n is relatively small.

The values for the parameters C and ν were again calculated using the same tuning procedure explained in section 5.1.2

As shown in Table 2, the correctness of the three methods was very similar, but the execution time including ten-fold cross validation for NSVM was less for all the datasets tested.

5.2.2 Numerical comparisons using a nonlinear classifier

In order to show that our algorithm can also be used to find nonlinear classifiers, we chose three datasets from the UCI Machine Learning Repository for which it is known that a nonlinear classifier performs better that a linear classifier. We used NSVM, LSVM and SVM^{light} in order to find a nonlinear classifier based on the Gaussian kernel:

$$(K(A,B))_{ij} = \varepsilon^{-\mu\|A_i{}' - B_{\cdot j}\|^2},\ i = 1\ldots, m,\ j = 1\ldots, k, \tag{39}$$

where $A \in R^{m\times n}$, $B \in R^{n\times k}$ and μ is a positive constant. The value of μ in the Gaussian kernel and the value of ν in NSVM and LSVM and C in SVM^{light} were chosen all by tuning from the set of values 2^i with i an integer ranging from -12 to 12 following the same procedure described in section 5.1.2. Because the nonlinear kernel matrix is square and since both NSVM and LSVM preform better on rectangular matrices, we also used a rectangular kernel formulation as described in the Reduced SVM (RSVM) [11]. This resulted in as good or better correctness and much faster running times. The size of the random sample used to calculate the rectangular kernel was 10% of the size of the original dataset in all cases. We refer to these variations of NSVM and LSVM as Reduced NSVM and Reduced LSVM respectively. The results are summarized in Table 3 for these nonlinear classifiers.

6 Conclusion

We have presented a fast and finitely terminating Newton method for solving a fundamental classification problem of data mining and machine learning. The method is simple and fast and can be applied to problems with a very large dimensional input space, which is often the case on problems related to analysis of gene expression microarray data. Even though the method is intended to be applied to problems with very large dimensional input space, it showed a excellent performance in other problems as well. Computational testing on a variety of real-world test problems demonstrate the effectiveness of the proposed method.

Acknowledgments

The research described in this Data Mining Institute Report 02-01, February 2002, was supported by National Science Foundation Grants CCR-9729842 and CDA-9623632, by Air Force Office of Scientific Research Grant F49620-00-1-0085 and by the Microsoft Corporation. We are grateful to our colleague David Page for making available to us the Multiple Myeloma dataset as well as a preprint of his joint paper [24]. We are also grateful to three anonymous referees for helpful constructive suggestions. Revised August 2002.

References

[1] L. Armijo. Minimization of functions having Lipschitz-continuous first partial derivatives. *Pacific Journal of Mathematics*, 16:1–3, 1966.

[2] D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, Belmont, MA, second edition, 1999.

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

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

[5] F. Facchinei. Minimization of SC^1 functions and the Maratos effect. *Operations Research Letters*, 17:131–137, 1995.

[6] G. Fung and O. L. Mangasarian. Incremental support vector machine classification. Technical Report 01-08, Data Mining Institute, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, September 2001. ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/01-08.ps.

[7] G. Fung and O. L. Mangasarian. Proximal support vector machine classifiers. In F. Provost and R. Srikant, editors, *Proceedings KDD-2001: Knowledge Discovery and Data Mining, August 26-29, 2001, San Francisco, CA*, pages 77–86, New York, 2001. Asscociation for Computing Machinery. ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/01-02.ps.

[8] G. H. Golub and C. F. Van Loan. *Matrix Computations*. The John Hopkins University Press, Baltimore, Maryland, 3rd edition, 1996.

[9] J.-B. Hiriart-Urruty, J. J. Strodiot, and V. H. Nguyen. Generalized hessian matrix and second-order optimality conditions for problems with C^{L1} data. *Applied Mathematics and Optimization*, 11:43–56, 1984.

[10] T. Joachims. Making large-scale support vector machine learning practical. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, pages 169–184. MIT Press, 1999.

[11] Y.-J. Lee and O. L. Mangasarian. RSVM: Reduced support vector machines. Technical Report 00-07, Data Mining Institute, Computer

Sciences Department, University of Wisconsin, Madison, Wisconsin, July 2000. Proceedings of the First SIAM International Conference on Data Mining, Chicago, April 5-7, 2001, CD-ROM Proceedings. ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/00-07.ps.

[12] Yuh-Jye Lee and O. L. Mangasarian. SSVM: A smooth support vector machine. *Computational Optimization and Applications*, 20:5–22, 2001. Data Mining Institute, University of Wisconsin, Technical Report 99-03. ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/99-03.ps.

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

[14] O. L. Mangasarian. Parallel gradient distribution in unconstrained optimization. *SIAM Journal on Control and Optimization*, 33(6):1916–1925, 1995. ftp://ftp.cs.wisc.edu/tech-reports/reports/1993/tr1145.ps.

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

[16] O. L. Mangasarian. A finite Newton method for classification problems. Technical Report 01-11, Data Mining Institute, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, December 2001. ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/01-11.ps.*Optimization Methods and Software,* to appear.

[17] O. L. Mangasarian and D. R. Musicant. Successive overrelaxation for support vector machines. *IEEE Transactions on Neural Networks*, 10:1032–1037, 1999. ftp://ftp.cs.wisc.edu/math-prog/tech-reports/98-18.ps.

[18] O. L. Mangasarian and D. R. Musicant. Lagrangian support vector machines. *Journal of Machine Learning Research*, 1:161–177, 2001. ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/00-06.ps.

[19] O. L. Mangasarian and M. V. Solodov. Nonlinear complementarity as unconstrained and constrained minimization. *Mathematical Programming, Series B*, 62:277–297, 1993.

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

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

[22] S. Odewahn, E. Stockwell, R. Pennington, R. Humphreys, and W. Zumach. Automated star/galaxy discrimination with neural networks. *Astronomical Journal*, 103(1):318–331, 1992.

[23] J. M. Ortega. *Numerical Analysis, A Second Course*. Academic Press, 1972.

[24] D. Page, F. Zhan, J. Cussens, M. Waddell, J. Hardin, B. Barlogie, and J. Shaughnessy,Jr. Comparative data mining for microarrays: A case study based on multiple myeloma. Technical report, Department of Biostatistics and Medical Informatics, University of Wisconsin, Madison, Wisconsin, January 2002.

[25] R. T. Rockafellar. Augmented Lagrange multiplier functions and duality in nonconvex programming. *SIAM Journal on Control*, 12:268–285, 1974.

[26] A. Smola and B. Schölkopf. *Learning with Kernels*. MIT Press, Cambridge, MA, 2002.

[27] V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, New York, second edition, 2000.

Table 2: **NSVM, SVM^{light} & LSVM: Training correctness, ten-fold testing correctness and ten-fold training times using a LINEAR classifier. NSVM and LSVM parameter ν and SVM^{light} parameter C, all chosen by tuning. Best results are in bold.**

Data Set $m \times n$ $(points \times dimensions)$	NSVM Train Test Time (Sec.)	SVM^{light} Train Test Time (Sec.)	LSVM Train Test Time (Sec.)
Ionosphere 351×34	**93.2%** **89.8%** **0.95**	92.0 % 88.3 % 2.3	**93.2%** **89.8%** 1.49
BUPA Liver 345×6	**70.3%** **70.2%** **0.19**	70.1% 69.3% 5.17	**70.3%** **70.2%** 0.61
Pima Indians 768×8	**77.7%** 77.0% **0.48**	77.4% **77.1%** 3.87	**77.7%** **77.0%** 2.04
Cleveland Heart 297×13	**87.3%** **86.3%** **0.31**	87.1% 85.9% 0.88	**87.3%** **86.3%** 0.83
Housing 506×13	87.2% **86.6%** **0.58**	**87.6%** 85.8% 5.57	87.2% **86.6%** 1.53
Galaxy Dim 4192×14	**95.0%** **95.3%** **7.16**	91.3% 91.2% 15.94	**95.0%** **95.3%** 76.67

Table 3: **NSVM, Reduced NSVM, SVM^{light}, LSVM & Reduced LSVM: Training correctness, ten-fold testing correctness and ten-fold training times using a NONLINEAR classifier. Best results are in bold.**

Data Set $m \times n$ $(points \times dimensions)$	Ionosphere 351×34	BUPA Liver 345×6	Cleveland Heart 297×13
NSVM			
Train	**96.1**	75.7	**87.6**
Test	95.0	73.1	86.8
Time (Sec.)	23.27	25.54	17.51
Reduced NSVM			
Train	**96.1**	76.4	86.8
Test	94.5	73.9	**87.1**
Time (Sec.)	**0.88**	**0.67**	**0.53**
SVM^{light}			
Train	94.4	**77.2**	87.1
Test	**96.0**	**74.2**	85.9
Time (Sec.)	2.42	2.74	0.74
LSVM			
Train	**96.1**	75.7	**87.6**
Test	95.0	73.1	86.8
Time (Sec.)	23.76	27.01	12.98
Reduced LSVM			
Train	**96.1**	75.1	87.1
Test	94.5	73.1	86.2
Time (Sec.)	2.09	1.81	1.09