Successive Overrelaxation for Support Vector Machines

O. L. Mangasarian & D. R. Musicant

Abstract— **Successive overrelaxation (SOR) for symmetric linear complementarity problems and quadratic programs [1], [2], [3] is used to train a support vector machine (SVM) [4], [5] for discriminating between the elements of two massive datasets, each with millions of points. Because SOR handles one point at a time, similar to Platt's sequential minimal optimization (SMO) algorithm [6] which handles two constraints at a time and Joachims' SVM^{light} [7] which handles a small number of points at a time, SOR can process very large datasets that need not reside in memory. The algorithm converges linearly to a solution. Encouraging numerical results are presented on datasets with up to 10 million points. Such massive discrimination problems cannot be processed by conventional linear or quadratic programming methods, and to our knowledge have not been solved by other methods. On smaller problems, SOR was faster than SVM^{light} and comparable or faster than SMO.**

Keywords—**Support vector machines, successive overrelaxation, massive data discrimination**

I. Introduction

SUCCESSIVE overrelaxation, originally developed for the solution of large systems of linear equations [8], [9] has been successfully applied to mathematical programming problems [10], [1], [2], [11], [12], [13], some with as many 9.4 million variables [14]. By taking the dual of the quadratic program associated with a support vector machine [4], [5] for which the margin (distance between bounding separating planes) has been maximized with respect to *both* the normal to the planes as well as their location, we obtain a very simple convex quadratic program with bound constraints *only*. This problem is equivalent to a symmetric mixed linear complementarity problem (i.e. with upper and lower bounds on its variables [15]) to which SOR can be directly applied. This corresponds to solving the SVM dual convex quadratic program for one variable at a time, that is computing one multiplier of a potential support vector at a time.

We note that in the Kernel Adatron Algorithm [16], [17], Friess, Cristianini and Campbell propose a similar algorithm which updates multipliers of support vectors one at a time. They also maximize the margin with respect to both the normal to the separating planes as well as their location (bias). However, because they minimize the 2-norm of the constraint violation y of equation (1) and we minimize the 1-norm of y, our dual variables are bounded above in (6) whereas theirs are not. Boser, Guyon and Vapnik [18] also maximize the margin with respect to both the normal to the separating planes as well as their location using a strategy from [19].

In Section II we state our discrimination problem as a classical support vector machine problem (1) and introduce our variant of the problem (4) that allows us to state its dual (6) as an SOR-solvable convex quadratic program with bounds. We show in Proposition II.1 that both problems yield the same answer under fairly broad conditions. In Section III we state our SOR algorithm and establish its linear convergence using a powerful result of Luo and Tseng [3, Proposition 3.5]. In Section IV we give numerical results for problems with datasets with as many as 10 million points. Section V draws some conclusions and points out future directions such as parallel SOR implementations that may lead to the solution of even larger problems.

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 $(x_+)_i = \max\,\{0, x_i\}$, $i = 1, \ldots, n$. For column vectors $x \in R^n$ and $y \in R^m$, $[x; y]$ denotes a column vector in R^{n+m}. The scalar (inner) product of two vectors x and y in the n-dimensional real space R^n will be denoted by $x'y$. For an $m \times n$ matrix A, A_i will denote the ith row of A and $A_{\cdot j}$ will denote the jth column of A. The identity matrix in a real space of arbitrary dimension will be denoted by I, while a column vector of ones of arbitrary dimension will be denoted by e.

II. The Support Vector Machine and its Variant

We consider the problem of discriminating between 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' [4], [5] is given by the following for some $\nu > 0$:

$$\begin{array}{rrcl} \displaystyle\min_{w,\gamma,y} & \nu e'y + \frac{1}{2}w'w & & \\ \text{s.t.} & D(Aw - e\gamma) + y & \geq & e \\ & y & \geq & 0. \end{array} \quad (1)$$

Here w is the normal to the bounding planes:

$$\begin{array}{rcl} x'w - \gamma & = & +1 \\ x'w - \gamma & = & -1. \end{array} \quad (2)$$

The first plane above bounds the class 1 points and the second plane bounds the class -1 points, if the two classes are linearly separable and $y = 0$. If the classes are linearly inseparable then the two planes bound the two classes with a "soft margin" determined by a slack variable $y \in R^m$,

O. L. Mangasarian & D. R. Musicant are with the Computer Sciences Department, University of Wisconsin, 1210 West Dayton Street, Madison, WI 53706. E-mail: olvi@cs.wisc.edu, musicant@cs.wisc.edu.

$y \geq 0$, that is:

$$\begin{array}{llllll} x'w & - & \gamma & + & y_i & \geq & +1, \text{ for } x = A_i \text{ and } D_{ii} = +1, \\ x'w & - & \gamma & - & y_i & \leq & -1, \text{ for } x = A_i \text{ and } D_{ii} = -1. \end{array} \tag{3}$$

The one-norm of the slack variable y is minimized with weight ν in (1). The quadratic term in (1), which is twice the reciprocal of the square of the 2-norm distance $\frac{2}{\|w\|_2}$ between the two planes of (2) in the n-dimensional space of $w \in R^n$ for a *fixed* γ, maximizes that distance. In our approach here, which is similar to that of [18], [16], [17], we measure the distance between the planes in the $(n+1)$-dimensional space of $[w;\gamma] \in R^{n+1}$ which is $\frac{2}{\|[w;\gamma]\|_2}$. Thus using twice its reciprocal squared instead, yields our variant of the SVM problem as follows:

$$\begin{array}{lrcl} \min\limits_{w,\gamma,y} & \nu e'y + \frac{1}{2}(w'w+\gamma^2) & & \\ \text{s.t.} & D(Aw - e\gamma) + y & \geq & e \\ & y & \geq & 0. \end{array} \tag{4}$$

The Wolfe duals [20, Section 8.2] to the quadratic programs (1) and (4) are as follows.

$$\max_u \; -\frac{1}{2}u'DAA'Du + e'u, \text{ s.t. } e'Du = 0, \; 0 \leq u \leq \nu e \quad (w = A'Du). \tag{5}$$

$$\max_u \; -\frac{1}{2}u'DAA'Du - \frac{1}{2}u'Dee'Du + e'u, \text{ s.t. } 0 \leq u \leq \nu e$$
$$(w = A'Du, \; \gamma = -e'Du, \; y = (e - D(Aw - e\gamma))_+). \tag{6}$$

We note immediately that the variables (w, γ, y) of the primal problem (4) can be directly computed from the solution u of its dual (6) as indicated. However, *only* w of the primal problem (1) variables can be directly computed from the solution u of its dual (5) as indicated. The remaining variables (γ, y) of (1) can be computed by setting $w = A'Du$ in (1), where u is a solution of its dual (5), and solving the resulting linear program for (γ, y). Alternatively, γ can be determined by minimizing the expression for $e'y = e'(e - D(Aw - e\gamma))_+$ as a function of the single variable γ after w has been expressed as a function of the dual solution u as indicated in (5), that is:

$$\min_{\gamma \in R} e'(e - D(AA'Du - e\gamma))_+). \tag{7}$$

We note that the formulation (6) can be extended to a general nonlinear kernel $K(A, A')$ by replacing AA' by the kernel $K(A, A')$ and the SOR approach can similarly be extended to a general nonlinear kernel. This is fully described in [21].

It is interesting to note that very frequently the standard SVM problem (1) and our variant (4) give the same w. For $1,000$ randomly generated such problems with $A \in R^{40 \times 5}$ and the same ν, only 34 cases had solution vectors w that differed by more than 0.001 in their 2-norm. In fact we can state the following result which gives sufficient conditions that ensure that every solution of (4) is also a solution of (1) for a possibly larger ν.

Proposition II.1: Each solution $(\tilde{w}, \tilde{\gamma}, \tilde{y})$ of (4) is a solution of (1) for a possibly larger value of ν in (1) whenever the following linear system has a solution $\tilde{v}$:

$$A'Dv = 0, \; e'Dv = \tilde{\gamma}, \; v \geq 0, \tag{8}$$

such that

$$e'\tilde{v}(e'\tilde{y} - 1) \leq \tilde{\gamma}^2. \tag{9}$$

We note immediately that condition (9) is automatically satisfied if $e'\tilde{y} \leq 1$. We skip the straightforward proof of this proposition which consists of writing the Karush-Kuhn-Tucker (KKT) conditions [20] for the two problems (1) and (4) and showing that if $(\tilde{w}, \tilde{\gamma}, \tilde{y}, \tilde{u})$ is a KKT point for (4), then $(\tilde{w}, \tilde{\gamma}, \tilde{y}, (\tilde{u} + \tilde{v}))$ is a KKT point for (1) with ν replaced by $\nu + e'\tilde{v}$.

We turn now to the principal computational aspect of the paper.

III. Successive Overrelaxation for Support Vector Machines

The main reason for introducing our variant (4) of the SVM is that its dual (6) does not contain an equality constraint as does the dual (5) of (1). This enables us to apply in a straightforward manner the effective matrix splitting methods such as those of [1], [2], [3] that process one constraint of (4) at a time through its dual variable, without the complication of having to enforce an equality constraint at each step on the dual variable u. This permits us to process massive data without bringing it all into fast memory.

If we define

$$H = D[A \quad -e], \;\; L + E + L' = HH', \tag{10}$$

where the nonzero elements of $L \in R^{m \times m}$ constitute the strictly lower triangular part of the symmetric matrix HH', and the nonzero elements of $E \in R^{m \times m}$ constitute the diagonal of HH', then the dual problem (6) becomes the following *minimization* problem after its objective function has been replaced by its negative:

$$\min_u \frac{1}{2}\|H'u\|^2 - e'u, \text{ s.t. } u \in S = \{u \mid 0 \leq u \leq \nu e\}. \tag{11}$$

A necessary and sufficient optimality condition for (11) is the following gradient projection optimality condition [22], [3]:

$$u = (u - \omega E^{-1}(HH'u - e))_\#, \; \omega > 0, \tag{12}$$

where $(\cdot)_\#$ denotes the 2-norm projection on the feasible region S of (11), that is:

$$((u)_\#)_i = \left\{ \begin{array}{ll} 0 & \text{if } u_i \leq 0 \\ u_i & \text{if } 0 < u_i < \nu \\ \nu & \text{if } u_i \geq \nu \end{array} \right\}, \; i = 1, \ldots, m. \tag{13}$$

Our SOR method, which is a matrix splitting method that converges linearly to a point $\bar{u}$ satisfying (12), consists of splitting the matrix HH' into the sum of two matrices as follows:

$$HH' = \omega^{-1}E(B+C), \text{ s.t. } B-C \text{ is positive definite.} \quad (14)$$

For our specific problem we take:

$$B = (I + \omega E^{-1}L),\ C = ((\omega - 1)I + \omega E^{-1}L'),\ 0 < \omega < 2. \quad (15)$$

This leads to the following linearly convergent [3, Equation (3.14)] matrix splitting algorithm:

$$u^{i+1} = (u^{i+1} - Bu^{i+1} - Cu^i + \omega E^{-1}e)_\#, \quad (16)$$

for which

$$B + C = \omega E^{-1}HH',\ B - C = (2 - \omega)I + \omega E^{-1}(L - L'). \quad (17)$$

Note that for $0 < \omega < 2$, the matrix $B + C$ is positive semidefinite and matrix $B - C$ is positive definite. The matrix splitting algorithm (16) results in the following easily implementable SOR algorithm once the values of B and C given in (15) are substituted in (16).

Algorithm III.1: **SOR Algorithm** Choose $\omega \in (0, 2)$. Start with any $u^0 \in R^m$. Having u^i compute u^{i+1} as follows:

$$u^{i+1} = (u^i - \omega E^{-1}(HH'u^i - e + L(u^{i+1} - u^i)))_\#, \quad (18)$$

until $\|u^{i+1} - u^i\|$ is less than some prescribed tolerance.

Remark III.2: The components of u^{i+1} are computed in order of increasing component index. Thus the SOR iteration (18) consists of computing u_j^{i+1} using $(u_1^{i+1}, \ldots, u_{j-1}^{i+1}, u_j^i, \ldots, u_m^i)$. That is, the latest computed components of u are used in the computation of u_j^{i+1}. The strictly lower triangular matrix L in (18) can be thought of as a substitution operator, substituting $(u_1^{i+1}, \ldots, u_{j-1}^{i+1})$ for $(u_1^i, \ldots, u_{j-1}^i)$. Thus, SOR can be interpreted as using each new component value of u immediately after it is computed, thereby achieving improvement over other iterative methods such as gradient methods where all components of u are updated at once.

We have immediately from [3, Proposition 3.5] the following linear convergence result.

Theorem III.3: **SOR Linear Convergence** The iterates $\{u^i\}$ of the SOR Algorithm III.1 converge R-linearly to a solution of $\bar{u}$ of the dual problem (6), and the objective function values $\{f(u^i)\}$ of (6) converge Q-linearly to $f(\bar{u})$. That is for $i \geq \bar{i}$ for some $\bar{i}$:

$$\begin{aligned} \|u^i - \bar{u}\| &\leq \mu\delta^i, \text{ for some } \mu > 0,\ \delta \in (0,1), \\ f(u^{i+1}) - f(\bar{u}) &\leq \tau(f(u^i) - f(\bar{u})), \text{ for some } \tau \in (0,1). \end{aligned} \quad (19)$$

Remark III.4: Even though our SOR iteration (18) is written in terms of the full $m \times m$ matrix HH', it can easily be implemented one row at a time without bringing all of the data into memory as follows for $j = 1, \ldots, m$:

$$u_j^{i+1} = (u_j^i - \omega E_{jj}^{-1}(H_j(\sum_{\ell=1,\, j>1}^{j-1} H_\ell u_\ell^{i+1} + \sum_{\ell=j}^{m} H_l u_\ell^i - 1))_\#. \quad (20)$$

A simple interpretation of this step is that one component of the multiplier u_j is updated at a time by bringing one constraint of (4) at a time.

IV. Numerical Testing

A. Implementation Details

We implemented the SOR algorithm in C++, utilizing some heuristics to speed up convergence. After initializing the u variables to zero, the first iteration consists of a sweep through all the data points. Since we assume that data points generally reside on disk and are expensive to retrieve, we retain in memory all support vectors, that is, constraints of (4) corresponding to nonzero components of u. We only utilize these support vectors for subsequent evaluations, until we can no longer make progress in improving the objective. We then do another sweep through all data points. This large sweep through all data points typically results in larger jumps in objective values than sweeping through the support vectors only, though it takes significantly longer. Moving back and forth between all data points and support vectors works quite well, as indicated by Platt's results on the SMO algorithm[6].

Another large gain in performance is obtained by sorting the support vectors in memory by their u values before sweeping through them to apply the SOR algorithm. Interestingly enough, sorting in either ascending or descending order gives significant improvement over no sorting at all. The experiments described below were all implemented using sorting in descending order when sweeping through all constraints, and sorting in ascending order when sweeping through just the support vectors. This combination yielded the best performance on the University of California at Irvine (UCI) Adult dataset [23].

All calculations were coded to consider three types of data structures: non-sparse, sparse, and binary. The calculations were all optimized to take advantage of the particular structure of the input data.

Finally, the SOR algorithm requires that parameters $\omega \in (0, 2)$ and $\nu > 0$ be set in advance. All the experiments here utilize $\omega = 1.0$ and $\nu = 0.05$. These values showed good performance on the Adult dataset; some of the experimental results presented here could conceivably be improved further by experimenting more with these parameters. Additional experimentation may lead to other values of ω and ν that achieve faster convergence.

B. Experimental Methodology

In order to evaluate the effectiveness of the SOR algorithm, we conducted two types of experiments. One set of experiments demonstrates the performance of the SOR algorithm in comparison with Platt's SMO algorithm [6] and Joachims' SVM^{light} algorithm [7]. We report below on results from running the SOR and SVM^{light} algorithms on the UCI Adult dataset along with the published results from SMO. The SMO experiments are reported to have been carried out on a 266 MHz Pentium II processor running Windows NT 4, using Microsoft's Visual C++ 5.0

compiler. We have run our comparable SOR experiments on a 200 MHz Pentium Pro processor with 64 megabytes of RAM, also running Windows NT 4 and Visual C++ 5.0. We ran our SVM^{light} experiments on the same hardware, but under the Solaris 5.6 operating system using the code available from Joachims' web site [24].

The other set of experiments is directed towards evaluating the efficacy of SOR on much larger datasets. These experiments were conducted under Solaris 5.6, using the GNU EGCS 1.0.3 C++ compiler, and run on the University of Wisconsin Computer Sciences Department Ironsides cluster. This cluster of four Sun Enterprise E6000 machines consists of 16 UltraSPARC II 248 MHz processors and 2 gigabytes of RAM on each node, resulting in a total of 64 processors and 8 gigabytes of RAM.

We first look at the effect of varying degrees of separability on the performance of the SOR algorithm for a dataset of 50,000 data points. We do this by varying the fraction of misclassified points in our generated data, and measure the corresponding performance of SOR. A tuning set of 0.1% is held out so that generalization can be measured as well. We use this tuning set to determine when the SOR algorithm has achieved 95% of the true separability of the dataset.

For the SMO experiments, the datasets are small enough in size so that the entire dataset can be stored in memory. These differ significantly from larger datasets, however, which must be maintained on disk. A disk-based dataset results in significantly larger convergence times, due to the slow speed of I/O access as compared to direct memory access. The C++ code is therefore designed to easily handle datasets stored either in memory or on disk. Our experiments with the UCI Adult dataset were conducted by storing all points in memory. For all other experiments, we kept the dataset on disk and stored only support vectors in memory.

We next ran SOR on the same dataset of 1 million points in R^{32} which was used in evaluating the LPC (Linear Programming Chunking) algorithm [25]. The previous LPC work measured how long it took the algorithm to converge to a stable objective. We consider this here as well; we also monitor training and tuning set accuracy. In order to do this, we remove 1% of the data (10,000 data points) to serve as a tuning set. This tuning set is not used at all in the SOR algorithm. Instead, we use it to monitor the generalizability of the separating plane which we produce. We chose 1% in order to balance the fact that we want the tuning set to be a significant fraction of the training set, yet it must be entirely stored in memory in order to be utilized efficiently.

Finally, we continue in a similar fashion and evaluate the success of SOR on a dataset one order of magnitude larger, namely consisting of 10 million points in R^{32}. We created the data by generating 10 million uniformly distributed random non-sparse data points, and a random separating hyperplane to split the data into two distinct classes. In order to make the data not linearly separable, we intentionally mislabeled a fraction of the points on each side of the plane in order to obtain a training set with a minimum prescribed separability of the data.

We note that a problem of this magnitude might be solvable by using a sample of the data. Preliminary experiments have shown that sampling may be less effective on datasets which are less separable. A proper experiment on the effect of sampling would require running SOR to convergence on various subsets of a massive dataset, and comparing test set accuracies. While we have not carried out this time-intensive procedure, we are considering further directions using sampling (see Conclusion).

C. Experimental Results

Table 1 shows the effect of training set separability on the speed of convergence of the SOR algorithm on a 50,000-point dataset in R^{32} to 95% correctness on a 5,000-point tuning set. We note that it takes about four times as long for SOR to converge for a training set that is 80% separable as it does for a set that is 99.5% separable. These results are also depicted in Figure 1. We note that there are many real world datasets that are 95%-99% separable [26], [23].

Table 2 illustrates running times, numbers of support vectors, and iterations for the SOR, SMO, and SVM^{light} algorithms using the UCI Adult dataset. Testing set accuracies are also included for SOR and SVM^{light}. We quote the SMO numbers from [6], and adopt the similar convention of showing both bound-constrained support vectors (those where $u_i = \nu$) and non-bound-constrained support vectors (those where $0 < u_i < \nu$). Note that the "iterations" column takes on very different meanings for the three algorithms. The SMO work defines an iteration as a single optimization process taking place between two data points. SVM^{light} defines an iteration as solving an optimization problem over a a small number of constraints. For SOR we define an iteration to be a complete sweep through the data. By "data" we mean either all the data points on disk, or all the support vectors in memory, depending on the type of iteration. We observe that although we ran our experiments on a **slower** processor, the larger datasets converged almost twice as fast under SOR as they did under SMO. These results are seen in Figure 2. Finally, we see that SOR converged much more quickly than SVM^{light} while still producing similar test set accuracies.

We note the underlying optimization problems (1) and (4) for SMO and SOR respectively are both strictly convex in the variable w. Hence this variable, which determines the orientation of the separating planes, is unique in both the SMO and SOR solutions. However these solutions may be different unless conditions of Proposition II.1 hold. Even when the solution to these two problems are the same theoretically, different implementation details of the two different algorithms can lead to different approximate solutions. Similar comments apply when comparing SOR with SVM^{light}.

The SOR algorithm performed quite well on the 1 million point dataset of [25]. Previous experiments with the LPC algorithm converged to a solution in 231.3 hours [25]. SOR converged in only 9.7 hours. This vast improvement may

True Separability	CPU Minutes to: 95% Separability Final Convergence	Iterations to: 95% Separability Final Convergence
80.0%	303.0 498.4	301,500 559,898
85.0%	83.3 334.6	76,000 445,213
90.0%	51.9 260.9	63,000 443,868
95.0%	39.2 230.6	62,500 351,266
98.5%	15.2 178.3	24,500 228,788
99.5%	4.5 127.8	9,000 129,535

Table 1: Effect of dataset separability on SOR performance on a 50,000-point dataset in R^{32}

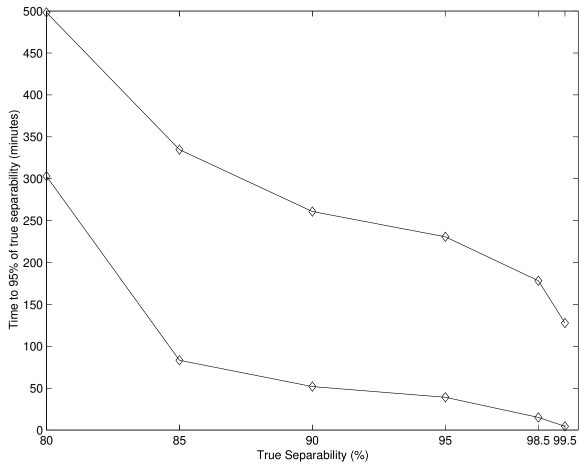

Fig. 1. **Effect of dataset separability on SOR performance on a 50,000-point dataset in R^{32}: Time to convergence (top curve) and time to 95% of true separability (bottom curve) versus true training set separability**

be partially due to the fact that we used a less stringent criterion for convergence in the SOR algorithm than in LPC; like SMO, we required that at least one u value change by more than 10^{-3} in order to proceed with another iteration. Furthermore (as in SMO), an attempt at updating a dual variable is made only if the associated primal constraint violates the KKT conditions by more than a particular tolerance (also 10^{-3}). In order to verify the quality of the solution that we obtained, we looked at training set and tuning set accuracy as described above. Both training set and tuning set accuracy achieved levels of 99.7% after 18 minutes. Though the algorithm continued to run for many hours, no further improvement in training and tuning accuracy was seen. We note that LPC did not review training or tuning set accuracy, and could possibly use a similar criterion for early stopping. These results are summarized in Table 3.

Finally, Table 4 shows SOR performance on a 99.9% linearly separable dataset of 10 million data points. After 14.3 hours, 95% of the true separability of the dataset was achieved.

Training Set Size	CPU Sec	Non-Bound SVs	Bound SVs	Iter*	Test Set Accuracy
	SOR SMO SVM^l	SOR SMO SVM^l	SOR SMO SVM^l	SOR SMO SVM^l	SOR SMO** SVM^l
1605	0.3 0.4 5.4	49 42 44	635 633 634	924 3230 292	84.06 84.25
2265	1.2 0.9 10.8	51 47 49	930 930 929	1142 4635 383	84.24 84.43
3185	1.4 1.8 21.0	52 57 62	1221 1210 1208	962 6950 732	84.23 84.40
4781	1.6 3.6 43.2	65 63 67	1794 1791 1788	1819 9847 865	84.28 84.47
6414	4.1 5.5 87.6	64 61 63	2379 2370 2370	1799 10669 956	84.30 84.43
11221	18.8 17.0 306.6	81 79 85	4085 4079 4078	2642 17128 1625	84.37 84.68
16101	24.8 35.3 667.2	86 67 77	5852 5854 5849	2995 22770 2234	84.62 84.83
22697	31.3 85.7 1425.6	103 88 109	8209 8209 8199	6061 35822 3960	85.06 85.17
32561	83.9 163.6 2184.0	201 149 166	11550 11558 11554	4702 44774 5075	84.96 85.05

Table 2: SOR, SMO [6], and SVM^{light} [7] comparisons on the Adult dataset in R^{123}.
***SOR, SMO, and SVM^{light} iterations are not directly comparable. See Section IV-C.**
****SMO test set accuracies on the Adult dataset have not been published. Oral communications with John Platt have suggested that SMO shows similar accuracies to those shown here for SOR and SVM^{light}.**

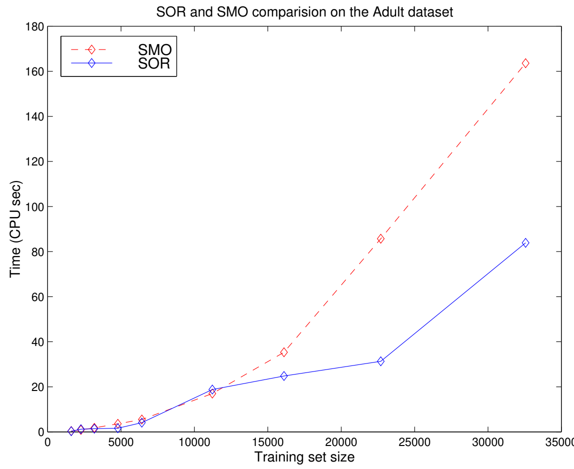

Fig. 2. **SOR and SMO comparison on the Adult dataset in R^{123}**

Training Set Size	CPU hrs	CPU hrs to 99.7% Accuracy
	SOR	SOR
	LPC	LPC
1,000,000	9.7	0.3
	231.1	—

Table 3: SOR and LPC [25] comparison on the 1 million point dataset in R^{32}

Training Set Size	Time to 95% separability	Iterations
10,000,000	14.3 hours	10,000

Table 4: SOR applied to 10 million point dataset in R^{32}

V. Conclusion

We have proposed a versatile iterative method, successive overrelaxation, for the solution of extremely large discrimination problems using support vector machines. The method converges considerably faster than other methods that require the presence of a substantial amount of the data in memory. We have solved problems that cannot be directly handled by conventional methods of mathematical programming. The proposed method scales up with no changes and can be easily parallelized by using techniques already implemented [27], [14]. Future work includes multicategory discrimination and nonlinear discrimination via kernel methods and successive overrelaxation. We also plan to use SOR in conjunction with sampling methods to choose appropriate sample sizes.

Acknowledgements

This research is supported by National Science Foundation Grants CCR-9729842 and CDA-9623632, and by Air Force Office of Scientific Research Grant F49620-97-1-0326 as Mathematical Programming Technical Report 98-18, November 1998.

References

[1] O. L. Mangasarian, "Solution of symmetric linear complementarity problems by iterative methods," *Journal of Optimization Theory and Applications*, vol. 22, no. 4, pp. 465–485, Aug. 1977.

[2] O. L. Mangasarian, "On the convergence of iterates of an inexact matrix splitting algorithm for the symmetric monotone linear complementarity problem," *SIAM Journal on Optimization*, vol. 1, pp. 114–122, 1991.

[3] Z.-Q. Luo and P. Tseng, "Error bounds and convergence analysis of feasible descent methods: A general approach," *Annals of Operations Research*, vol. 46, pp. 157–178, 1993.

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

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

[6] J. Platt, "Sequential minimal optimization: A fast algorithm for training support vector machines," in *Advances in Kernel Methods - Support Vector Learning*, Bernhard Schölkopf, Christopher J. C. Burges, and Alexander J. Smola, Eds. 1999, pp. 185–208, MIT Press, http://www.research.microsoft.com/~jplatt/smo.html.

[7] T. Joachims, "Making large-scale support vector machine learning practical," in *Advances in Kernel Methods - Support Vector Learning*, Bernhard Schölkopf, Christopher J. C. Burges, and Alexander J. Smola, Eds. 1999, pp. 169–184, MIT Press.

[8] J. M. Ortega and W. C. Rheinboldt, *Iterative Solution of Nonlinear Equations in Several Variables*, Academic Press, 1970.

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

[10] C. W. Cryer, "The solution of a quadratic programming problem using systematic overrelaxation," *SIAM Journal on Control and Optimization*, vol. 9, pp. 385–392, 1971.

[11] J.-S. Pang, "More results on the convergence of iterative methods for the symmetric linear complementarity problem," *Journal of Optimization Theory and Applications*, vol. 49, pp. 107–134, 1986.

[12] O. L. Mangasarian and R. De Leone, "Parallel gradient projection successive overrelaxation for symmetric linear complementarity problems," *Annals of Operations Research*, vol. 14, pp. 41–59, 1988.

[13] W. Li, "Remarks on convergence of matrix splitting algorithm for the symmetric linear complementarity problem," *SIAM Journal on Optimization*, vol. 3, pp. 155–163, 1993.

[14] R. De Leone and M. A. Tork Roth, "Massively parallel solution of quadratic programs via successive overrelaxation," *Concurrency: Practice and Experience*, vol. 5, pp. 623–634, 1993.

[15] S. P. Dirkse and M. C. Ferris, "MCPLIB: A collection of nonlinear mixed complementarity problems," *Optimization Methods and Software*, vol. 5, pp. 319–345, 1995, ftp://ftp.cs.wisc.edu/tech-reports/reports/94/tr1215.ps.

[16] T.-T. Friess, N. Cristianini, and C. Campbell, "The kernel-adatron algorithm: A fast and simple learning procedure for support vector machines," in *Machine Learning Proceedings of the Fifteenth International Conference (ICML'98)*, Jude Shavlik, Ed., San Francisco, 1998, pp. 188–196, Morgan Kaufmann, http://svm.first.gmd.de/papers/FriCriCam98.ps.gz.

[17] T.-T. Friess, "Support vector neural networks: The kernel adatron with bias and soft margin," Tech. Rep., Department of Automatic Control and Systems Engineering, University of Sheffield, Sheffield, England, 1998, Revised Version: www.brunner-edv.com/friess/.

[18] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, D. Haussler, Ed., Pittsburgh, PA, July 1992, pp. 144–152, ACM Press.

[19] V. N. Vapnik, *Estimation of Dependences Based on Empirical Data*, Springer, New York, 1982.

[20] O. L. Mangasarian, *Nonlinear Programming*, McGraw–Hill, New York, 1969, Reprint: SIAM Classic in Applied Mathematics 10, 1994, Philadelphia.

[21] O. L. Mangasarian and David R. Musicant, "Data discrimination via nonlinear generalized support vector machines," Tech. Rep. 99-03, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, March 1999, ftp://ftp.cs.wisc.edu/math-prog/tech-reports/99-03.ps.Z.

[22] B. T. Polyak, *Introduction to Optimization*, Optimization Software, Inc., Publications Division, New York, 1987.

[23] P. M. Murphy and D. W. Aha, "UCI repository of machine learning databases," Tech. Rep., Department of Information and Computer Science, University of California, Irvine, 1992, www.ics.uci.edu/~mlearn/MLRepository.html.

[24] T. Joachims, "Svm^{light}," 1998, www-ai.informatik.uni-dortmund.de/FORSCHUNG/VERFAHREN/SVM_LIGHT/svm_light.eng.html.

[25] P. S. Bradley and O. L. Mangasarian, "Massive data discrimination via linear support vector machines," Tech. Rep. 98-05, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, May 1998, ftp://ftp.cs.wisc.edu/math-prog/tech-reports/98-03.ps.Z.

[26] Chunhui Chen and O. L. Mangasarian, "Hybrid misclassification minimization," *Advances in Computational Mathematics*, vol. 5, no. 2, pp. 127–136, 1996, ftp://ftp.cs.wisc.edu/math-prog/tech-reports/95-05.ps.Z.

[27] R. De Leone, O. L. Mangasarian, and T.-H. Shiau, "Multi-sweep asynchronous parallel successive overrelaxation for the nonsymmetric linear complementarity problem," *Annals of Operations Research*, vol. 22, pp. 43–54, 1990.

Olvi Mangasarian received his Ph.D. in Applied Mathematics from Harvard University, and worked for eight years as a Mathematician for Shell Oil Company in California before coming to the University of Wisconsin, where he is now John von Neumann Professor of Mathematics and Computer Sciences. His main research interests are in mathematical programming, machine learning and data mining. He has had a long-term interest in breast cancer diagnosis and prognosis problems. A breast cancer diagnostic system based on his work is in current use at University of Wisconsin Hospital. He is the author of the book "Nonlinear Programming" and associate editor of three optimization journals. His recent papers are available at www.cs.wisc.edu/∼olvi.

David Musicant received B.S. degrees in both Mathematics and Physics from Michigan State University, then earned an M.A. in Mathematics and an M.S. in Computer Sciences from the University of Wisconsin. He spent three years in the consulting industry as a Technical Operations Research Consultant for ZS Associates, and as a Senior Consultant for Icon InfoSystems, both in Chicago. David is now pursuing a Ph.D. in Computer Sciences at the University of Wisconsin, with interests in applying data mining to massive datasets. His recent papers are available at www.cs.wisc.edu/∼musicant.