Training Support Vector Machines with Particle Swarms

U Paquet
Department of Computer Science
University of Pretoria
South Africa
Email: upaquet@cs.up.ac.za

AP Engelbrecht
Department of Computer Science
University of Pretoria
South Africa
Email: engel@driesie.cs.up.ac.za

***Abstract*— Training a Support Vector Machine requires solving a constrained quadratic programming problem. Linear Particle Swarm Optimization is intuitive and simple to implement, and is presented as an alternative to current numeric SVM training methods. Performance of the new algorithm is demonstrated on the MNIST character recognition dataset.**

I. Introduction

Support Vector Machines (SVMs) are a young and important addition to the machine learning toolbox. Having been formally introduced by Boser et al. [1], SVMs have proved their worth – in the last decade there has been a remarkable growth in both the theory and practice of these learning machines.

Training a SVM requires solving a linearly constrained quadratic optimization problem. This problem often involves a matrix with an extremely large number of entries, which make off-the-shelf optimization packages unsuitable. Several methods have been used to decompose the problem, of which many require numeric packages for solving the smaller subproblems.

Particle Swarm Optimization (PSO) is an intuitive and easy-to-implement algorithm from the swarm intelligence community, and is introduced as a new way of training a SVM. Using PSO replaces the need for numeric solvers. A Linear PSO (LPSO) is adapted and shown to be ideal in optimizing the SVM problem.

This paper gives an overview of the SVM algorithm, and explains the main methodologies for training SVMs. PSO is discussed as an alternative method for solving a SVM's quadratic programming problem. Experimental results on character recognition illustrate the convergence properties of the algorithms.

II. Support Vector Machines

Traditionally, a SVM is a learning machine for two-class classification problems, and learns from a set of l N-dimensional example vectors $\mathbf{x}_i$, and their associated classes y_i, i.e.

$$\{\mathbf{x}_1, y_1\}, \ldots, \{\mathbf{x}_l, y_l\} \in \mathbb{R}^N \times \{\pm 1\} \tag{1}$$

The algorithm aims to learn a separation between the two classes by creating a linear decision surface between them. This surface is, however, not created in input space, but rather in a very high-dimensional feature space. The resulting model is nonlinear, and is accomplished by the use of kernel functions. The kernel function k gives a measure of similarity between a pattern $\mathbf{x}$, and a pattern $\mathbf{x}_i$ from the training set. The decision boundary that needs to be constructed is of the form

$$f(\mathbf{x}) = \sum_{i=1}^{l} y_i \alpha_i k(\mathbf{x}, \mathbf{x}_i) + b \tag{2}$$

where the class of $\mathbf{x}$ is determined from the sign of $f(\mathbf{x})$. The α_i are Lagrange multipliers from a primal quadratic programming (QP) problem, and there is an α_i for each vector in the training set. The value b is a threshold. "Support vectors" define the decision surface, and correspond to the subset of nonzero α_i. These vectors can be seen as the "most informative" training vectors.

Training the SVM consists of finding the values of α_i. By defining a Hessian matrix Q such that $(Q)_{ij} = y_i y_j k(\mathbf{x}_i, \mathbf{x}_j)$, training can be expressed as a dual QP problem of solving

$$\max_{\boldsymbol{\alpha}} \; W(\boldsymbol{\alpha}) = \boldsymbol{\alpha}^T \mathbf{1} - \frac{1}{2} \boldsymbol{\alpha}^T Q \boldsymbol{\alpha} \tag{3}$$

subject to one equality constraint

$$\boldsymbol{\alpha}^T \mathbf{y} = 0 \tag{4}$$

and a set of box constraints

$$\begin{aligned} \boldsymbol{\alpha} &\geq \mathbf{0} \\ C\mathbf{1} - \boldsymbol{\alpha} &\geq \mathbf{0} \end{aligned} \tag{5}$$

Training a SVM thus involves solving a linearly constrained quadratic optimization problem.

III. SVM Training Methods

The QP problem is equivalent to finding the maximum of a constrained bowl-shaped objective function. Due to the definition of the kernel function, the matrix Q always gives a convex QP problem, where every local solution is also a global solution [2]. Certain optimality conditions – the Karush-Kuhn-Tucker (KKT) conditions [2] – give conditions determining whether the constrained maximum has been found.

Solving the QP problem for real-world problems can prove to be very difficult: The matrix Q has a dimension equal to the number of training examples. A training set of 60,000 vectors gives rise to a matrix Q with 3.6 billion elements, which does not fit into the memory of a standard computer. For large learning tasks, off-the-shelf optimization packages and techniques for general quadratic programming quickly become intractable in their memory and time requirements. A

number of other approaches, which allow for fast convergence and small memory requirements, even on large problems, have been invented:

Chunking

The chunking algorithm is based on the fact that the non-support vectors play no role in the SVM decision boundary. If they are removed from the training set of examples, the SVM solution will be exactly the same.

Chunking has been suggested by V. Vapnik in [14], and breaks the large QP problem down into a number of smaller problems.

A QP routine is used to optimize the Lagrangian on an arbitrary subset of data. After this optimization, the set of nonzero α_i (the current support vectors) are retained, and all other data points ($\alpha_i = 0$) are discarded. At every subsequent step, chunking solves the QP problem that consists of all nonzero α_i, plus some of the α_i that violates the KKT conditions. After optimizing the subproblem, data points with $\alpha_i = 0$ are again discarded. This procedure is iterated until the KKT conditions are met, and the margin is maximized.

The size of the subproblem varies, but tends to grow with time. At the last step, chunking has identified and optimized all the nonzero α_i, which correspond to the set of all the support vectors. Thus the overall QP problem is solved.

Although this technique of reducing the Q matrix's dimension from the number of training examples to approximately the number of support vectors makes it suitable to large problems, even the reduced matrix may not fit into memory.

Decomposition

Decomposition methods are similar to chunking, and were introduced by E. Osuna in [8], [9]. The large QP problem is broken down into a series of smaller subproblems, and a numeric QP optimizer solves each of these problems. It was suggested that one vector be added and one removed from the subproblem at each iteration, and that the size of the subproblems should be kept fixed. The motivation behind this method is based on the observation that as long as at least one α_i violating the KKT conditions is added to the previous subproblem, each step reduces the objective function and maintains all of the constraints. In this fashion the sequence of QP subproblems will asymptotically converge. For faster practical convergence, researchers add and delete multiple examples.

While the strategy used in chunking takes advantage of the fact that the expected number of support vectors is small ($< 3,000$), decomposition allows for training arbitrarily large data sets.

Another decomposition method was introduced by T. Joachims [3]. Joachim's method is based on the gradient of the objective function. The idea is to pick α_i for the QP subproblem such that the α_i form the steepest possible direction of ascent on the objective function, where the number of nonzero elements in the direction is equal to the size of the QP subproblem. As in Osuna's method, the size of the subproblem remains fixed.

Solving each subproblem requires a numeric quadratic optimizer.

Sequential Minimal Optimization

The most extreme case of decomposition is Sequential Minimal Optimization (SMO) – where the smallest possible optimization problem is solved at each step [11]. Because the α_i must obey the linear equality constraint, two α_i is chosen to be jointly optimized. No numerical QP optimization is necessary, and after an analytic solution, the SVM is updated to reflect the new optimal values.

With the exception of SMO, a numeric QP library is needed for training a SVM. An intuitive and alternative approach is to use PSO to optimize each decomposed subproblem. The PSO algorithm is easy to implement, and certain properties of the LPSO make it ideal for the type of problem posed by SVM training.

IV. Particle Swarm Optimization

PSO [4] is similar in spirit to birds migrating in a flock toward some destination, where the intelligence and efficiency lies in the cooperation of an entire flock.

PSO differs from traditional optimization methods in that a population of potential solutions is used in the search. The direct fitness information instead of function derivatives or related knowledge is used to guide the search. Particles collaborate as a population, or swarm, to reach a collective goal, for example maximizing an n-dimensional objective function f. Each particle has memory of the best solution that it has found, called its *personal best*. A particle's traversal of the search space is influenced by its personal best and the best solution found by a neighborhood of particles.

There is thus a sharing of information that takes place. Particles profit from the discoveries and previous experience of other particles during the exploration and search for higher objective function values. There exists a great number of schemes in which this information sharing can take place. One of two sociometric principles is usually implemented. The first, called *gbest* (global best) PSO, conceptually connects all the particles in the population to one another, so that the very best performance of the entire population – the *global best* – influences each particle. The second, called *lbest* (local best), creates a neighborhood for each individual comprising of itself and some fixed number of its nearest neighbors. Since SVM training requires solving a convex problem, the *gbest* version is implemented in this paper.

Let i indicate a particle's index in the swarm. In a *gbest* PSO each of the s particles $\mathbf{p}_i$ fly through the n-dimensional search space $\mathbb{R}^n$ with a velocity $\mathbf{v}_i$, which is dynamically adjusted according to its own previous best solution $\mathbf{z}_i$ and the previous best solution $\widehat{\mathbf{z}}$ of the entire swarm.

In the original PSO [4], each particle's velocity adjustments are calculated separately for each component in its position

vector. By calculating velocity adjustments as linear combinations of position vectors, equality constraints on the objective function can easily be met.

Equality Constraints and the Linear PSO

The Linear PSO (LPSO) was introduced by [10] to constrain the movement of a swarm to a linear hyperplane in $\mathbb{R}^n$. LPSO differs from the original PSO, since velocity updates are calculated as a linear combination of position and velocity vectors. The particles of a LPSO interact and move according to the following equations

$$\mathbf{v}_i^{(t+1)} = w\mathbf{v}_i^{(t)} + c_1 r_1^{(t)}[\mathbf{z}_i^{(t)} - \mathbf{p}_i^{(t)}] + c_2 r_2^{(t)}[\widehat{\mathbf{z}}^{(t)} - \mathbf{p}_i^{(t)}] \quad (6)$$

$$\mathbf{p}_i^{(t+1)} = \mathbf{v}_i^{(t+1)} + \mathbf{p}_i^{(t)} \quad (7)$$

where $r_1^{(t)}$, $r_2^{(t)} \sim UNIF(0,1)$ are random numbers between zero and one. These numbers are scaled by acceleration coefficients c_1 and c_2, where $0 \leq c_1, c_2 \leq 2$, and w is an inertia weight [12]. It is possible to clamp the velocity vectors by specifying upper and lower bounds on $\mathbf{v}_i$, to avoid too rapid movement of particles in the search space.

When the objective function f needs to be maximized subject to constraints $A\mathbf{p} = \mathbf{b}$, the swarm should be constrained to fly through hyperplane P. With A being a $m \times n$ matrix and $\mathbf{b}$ a m-dimensional vector, $P = \{\mathbf{p} \mid A\mathbf{p} = \mathbf{b}\}$ defines the set of feasible solutions to the constrained problem, and each point in P will be a feasible point.

It was shown in [10] that if the initial swarm lies in P, LPSO will force the particles to fly only in feasible directions, and the swarm will optimize within the search space P.

Premature convergence is overcome by using a version of van den Bergh's Guaranteed Convergence Particle Swarm Optimizer [13]. In this algorithm, the velocity updates for the global best particle is changed to force it to search for a better solution in an area around the position of that particle. Let τ be the index of the global best particle, such that $\mathbf{z}_\tau = \widehat{\mathbf{z}}$. The velocity update equation for particle τ is changed to

$$\mathbf{v}_\tau^{(t+1)} = -\mathbf{p}_\tau^{(t)} + \widehat{\mathbf{z}}^{(t)} + w\mathbf{v}_\tau^{(t)} + \rho^{(t)}\boldsymbol{\upsilon}^{(t)} \quad (8)$$

where $\rho^{(t)}$ is some scaling factor, and $\boldsymbol{\upsilon}^{(t)} \sim UNIF(-1,1)^n$ is a random n-dimensional vector with the property that $A\boldsymbol{\upsilon}^{(t)} = \mathbf{0}$, or $\boldsymbol{\upsilon}^{(t)}$ lies in the null space of A.

The LPSO algorithm [10] is summarized below:

1) Set the iteration number t to zero. Initialize the swarm S of s particles such that the position $\mathbf{p}_i^{(0)}$ of each particle meets $A\mathbf{p}_i^{(0)} = \mathbf{b}$.
2) Evaluate the performance $f(\mathbf{p}_i^{(t)})$ of each particle.
3) Compare the personal best of each particle to its current performance, and set $\mathbf{z}_i^{(t)}$ to the better performance, i.e.

$$\mathbf{z}_i^{(t)} = \begin{cases} \mathbf{z}_i^{(t-1)} & \text{if } f(\mathbf{p}_i^{(t)}) \leq f(\mathbf{z}_i^{(t-1)}) \\ \mathbf{p}_i^{(t)} & \text{if } f(\mathbf{p}_i^{(t)}) > f(\mathbf{z}_i^{(t-1)}) \end{cases} \quad (9)$$

4) Set the global best $\widehat{\mathbf{z}}^{(t)}$ to the position of the particle with the best performance within the swarm, i.e.

$$\widehat{\mathbf{z}}^{(t)} \in \{\mathbf{z}_1^{(t)}, \mathbf{z}_2^{(t)}, \ldots, \mathbf{z}_s^{(t)}\} \mid f(\widehat{\mathbf{z}}^{(t)}) = \max\{f(\mathbf{z}_1^{(t)}), f(\mathbf{z}_2^{(t)}), \ldots, f(\mathbf{z}_s^{(t)})\} \quad (10)$$

5) Change the velocity vector for each particle according to equation (6).
6) Move each particle to its new position, according to equation (7).
7) Let $t := t + 1$.
8) Go to step 2, and repeat until convergence.

The LPSO algorithm is sufficient to optimize the SVM objective function subject to the linear equality constraint (4). The box constraints (5) are easily handled by initializing all particles $\mathbf{p}_i$ to lie inside the hypercube defined by the constraints, and restricting their movement to this hypercube. When a particle is moving outside the boundary of the hypercube, its velocity vector is scaled with some factor such that all components of its position lie either inside the hypercube, or on its boundary.

The practical side of using LPSO, as well as the training algorithm, is discussed in the following section.

V. Training the SVM

Using LPSO to solve the SVM QP problem requires criteria for optimality, a way to decompose the QP, and a way to extend LPSO to optimize the SVM subproblem.

Since Q is a positive semi-definite matrix (the kernel function used is positive semi-definite), and the constraints are linear, the Karush-Kuhn-Tucker (KKT) conditions are necessary and sufficient for optimality [2]. A solution $\boldsymbol{\alpha}$ of the QP problem, as stated in equalities (3) – (5), is an optimal solution if the following relations hold for each α_i:

$$\begin{aligned} \alpha_i = 0 &\Rightarrow y_i f(\mathbf{x}_i) \geq 1 \\ 0 < \alpha_i < C &\Rightarrow y_i f(\mathbf{x}_i) = 1 \\ \alpha_i = C &\Rightarrow y_i f(\mathbf{x}_i) \leq 1 \end{aligned} \quad (11)$$

where i is the index of an example vector from the training set.

Decomposing the QP problem involves choosing a subset, or "working set," of variables for optimization. The working set, called set B, is created by picking q sub-optimal variables from all l α_i. The working set of variables is optimized while keeping the remaining variables (called set N) constant. The general decomposition algorithm works as follows:

1) While the KKT conditions for optimality are violated:
   a) Select q variables for the working set B. The remaining $l - q$ variables (set N) are fixed at their current value.
   b) Use LPSO to optimize $W(\boldsymbol{\alpha})$ on B.
   c) Return the optimized α_i from B to the original set of variables.
2) Terminate and return $\boldsymbol{\alpha}$.

A concern in the above algorithm is to select the optimal working set. The decomposition method presented here is due to [3], and works on the method of feasible directions. The idea is to find the steepest feasible direction $\mathbf{d}$ of ascent on the objective function W as defined in equation (3), under the requirement that only q components be nonzero. The α_i corresponding to these q components will be included in the

working set. Finding an approximation to $\mathbf{d}$ is equivalent to solving

$$\begin{aligned} \text{Maximise} \quad & \nabla W(\boldsymbol{\alpha})^T \mathbf{d} \\ \text{subject to} \quad & \mathbf{y}^T \mathbf{d} = 0 \\ & d_i \geq 0 \qquad \text{if } \alpha_i = 0 \\ & d_i \leq 0 \qquad \text{if } \alpha_i = C \\ & d_i \in \{-1, 0, 1\} \\ & |\{d_i : d_i \neq 0\}| = q \end{aligned}$$

For $\mathbf{y}^T \mathbf{d}$ to be equal to zero, the number of elements with sign matches between d_i and y_i must be equal to the number of elements with sign mismatches between d_i and y_i. Also, $\mathbf{d}$ should be chosen to maximize the direction of ascent $\nabla W(\boldsymbol{\alpha})^T \mathbf{d}$. This is achieved by first sorting the training vectors in increasing order according to $y_i \nabla W(\boldsymbol{\alpha})_i$. Assuming q to be even, a "forward pass" selects $\frac{q}{2}$ examples from the front of the sorted list, and a "backward pass" selects $\frac{q}{2}$ examples from the back. A full explanation of this method is given by P. Laskov in [5].

It is necessary to rewrite the objective function (3) as a function that is only dependent on the working set. Let $\boldsymbol{\alpha}$ be split into two sets $\boldsymbol{\alpha}_B$ and $\boldsymbol{\alpha}_N$. If $\boldsymbol{\alpha}$, $\mathbf{y}$ and Q are appropriately rearranged, we have

$$\boldsymbol{\alpha} = \begin{bmatrix} \boldsymbol{\alpha}_B \\ \boldsymbol{\alpha}_N \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} \mathbf{y}_B \\ \mathbf{y}_N \end{bmatrix}, \quad Q = \begin{bmatrix} Q_{BB} & Q_{BN} \\ Q_{NB} & Q_{NN} \end{bmatrix}$$

Since only $\boldsymbol{\alpha}_B$ is going to be optimized, W is rewritten in terms of $\boldsymbol{\alpha}_B$. If terms that do not contain $\boldsymbol{\alpha}_B$ are dropped, the optimization problem remains essentially the same. Also, since Q is symmetric, with $Q_{BN} = Q_{NB}^T$, the problem is to find:

$$\max_{\boldsymbol{\alpha}_B} W(\boldsymbol{\alpha}_B) = \boldsymbol{\alpha}_B^T \mathbf{1} - \frac{1}{2} \boldsymbol{\alpha}_B^T Q_{BB} \boldsymbol{\alpha}_B - \boldsymbol{\alpha}_B^T Q_{BN} \boldsymbol{\alpha}_N \quad (12)$$

subject to

$$\begin{aligned} \boldsymbol{\alpha}_B^T \mathbf{y}_B + \boldsymbol{\alpha}_N^T \mathbf{y}_N &= 0 \\ \boldsymbol{\alpha}_B &\geq \mathbf{0} \\ C\mathbf{1} - \boldsymbol{\alpha}_B &\geq \mathbf{0} \end{aligned} \quad (13)$$

Implementing Particle Swarm Optimization

When the decomposition algorithm starts, a feasible solution that satisfies the linear constraint $\boldsymbol{\alpha}^T \mathbf{y} = 0$, with constraints $0 \leq \alpha_i \leq C$ also met, is needed. The initial solution is constructed in the following way:

Let c be some real number between 0 and C, and γ some positive integer less than both the number of positive examples ($y_i = +1$) and negative examples ($y_i = -1$) in the training set. Randomly pick a total of γ positive examples, and γ negative examples, and initialize their corresponding α_i to c. By setting all other α_i to zero, the initial solution will be feasible.

The value 2γ gives the total number of initial support vectors, and since these initial support vectors are a randomly chosen guess, it is suggested that the value of γ be kept small.

TABLE I

INFLUENCE OF DIFFERENT WORKING SET SIZES ON THE FIRST 20,000 ELEMENTS OF THE MNIST DATASET

Working set size	Working set selections	Time	SVs
4	8,782	02:17:43	1,631
6	8,213	03:11:40	1,637
8	7,502	03:51:24	1,639
10	10,023	06:27:06	1,648
12	9,667	07:26:23	1,652

In optimizing the q-dimensional subproblem, LPSO requires that all particles be initialized such that $\boldsymbol{\alpha}_B^T \mathbf{y}_B + \boldsymbol{\alpha}_N^T \mathbf{y}_N = 0$ is met. This is done as follows:

1) Set each particle in the swarm to the q-dimensional vector $\boldsymbol{\alpha}_B$.
2) Add a random q-dimensional vector $\boldsymbol{\delta}$ satisfying $\mathbf{y}_B^T \boldsymbol{\delta} = 0$ to each particle, under the condition that the particle will still lie in the hypercube $[0, C]^q$.

Initializing the swarm in this way ensures that the initial swarm lies in the set of feasible solutions $P = \{\mathbf{p} \mid A\mathbf{p} = -\boldsymbol{\alpha}_N^T \mathbf{y}_N\}$, allowing the flight of the swarm to be defined by feasible directions.

For faster convergence, the vector $\boldsymbol{v}^{(t)}$ used to adjust the global best particle, can be chosen as an approximation to the partial derivative $\nabla W(\boldsymbol{\alpha}_B)$, subject to $\mathbf{y}_B^T \boldsymbol{v}^{(t)} = 0$.

VI. EXPERIMENTAL RESULTS

The SVM training algorithm presented in this paper was tested on the MNIST dataset [7]. The MNIST dataset is a database of optical characters, and consists of a training set of 60,000 handwritten digits. Each digit is a 28 by 28 pixel gray-level image, equivalent to a 784-dimensional input vector. Each pixel corresponds to an integer value in the range of 0 (white) to 255 (black)

For training a SVM on the MNIST dataset, the character '8' was used to represent the positive examples, while the remaining digits defined the negative examples. Training was done with a polynomial kernel of degree five:

$$k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j + 1)^5 \quad (14)$$

Due to the size of the dot product between two images, raised to the fifth power, the pixel values were scaled to the range $[0, 0.1]$. This gives Lagrange multiplies α_i that are easier for the LPSO to handle. (The kernel function of two unscaled black images would be $(784 \times 255^2 + 1)^5$, while the kernel function of the scaled versions gives a more practical $(784 \times 0.01 + 1)^5 \approx 835$).

For an optimal solution to be found in the following PSO experiments, the KKT conditions needed to be satisfied within an error threshold of 0.02 from the right hand side of equations (11). Optimization of the working set terminated when the KKT conditions on the working set were met within an error

TABLE II
SCALABILITY: TRAINING ON THE MNIST DATASET

MNIST elements	PSO Working set selections	PSO time	PSO SVs	SMO time	SMO SVs	SVM^{light} time	SVM^{light} SVs
10,000	3,898	00:29:49	1,022	00:01:29	1,032	00:02:02	1,034
20,000	8,782	02:17:43	1,631	00:06:14	1,647	00:10:43	1,641
30,000	12,428	04:50:11	1,988	00:13:22	2,012	00:23:04	2,001
40,000	15,725	08:14:26	2,353	00:22:46	2,355	00:41:09	2,367
50,000	22,727	15:05:09	2,728	01:46:38	2,740	01:31:48	2,726
60,000	25,914	20:54:15	3,025	04:38:11	3,043	08:01:05	3,026

of 0.001, or when the swarm has optimized for a hundred iterations.

The following parameters defined the experimental PSO: By letting $\gamma = 10$, a total of 20 initial support vectors were chosen to start the algorithm. The swarm size s used in each experiment was 10, while the inertia weight w was set to 0.7. The acceleration coefficients c_1 and c_2 were both set to 1.4 [13]. Since the objective function is constrainted by a set of box constraints, the velocity vectors were not clamped. For each experiment the upper bound C was kept at 100.0.

The PSO training algorithm was written in Java, and does not make use of caching and shrinking methods to optimize its speed. The sparsity of input data is used to speed up the evaluation of kernel functions. All experiments were preformed on a 1.00 GHz AMD Duron processor.

Experimental results show successful and accurate training on the MNIST database. The influence of different working set sizes on the LPSO training algorithm, its scalability, as well as its relation to other SVM training algorithms, were examined.

Influence of working set sizes

Experiments on different working set sizes were done on the first 20,000 elements of the MNIST database. Results are shown in Table I, and indicate that a working set of size $q = 4$ gives the fastest convergence time and fewest support vectors. A working set of size 2 can be solved analytically, as is true in the case of SMO. The results in Table I are not necessarily an indication of the speed of the PSO on the working set, as selection of the working set also burdens the speed of the algorithm (the $\frac{q}{2}$ greatest and least values of $y_i \nabla W(\boldsymbol{\alpha})_i$ need to be selected from a list of thousands).

Scalability of the PSO approach

Scalability of the PSO algorithm was tested by training on the first 10,000, 20,000, etc. examples from the MNIST dataset, as shown in Table II. In each case a working set of size 4 was used. The experimental results indicate that the PSO training algorithm shows quadratic scalability, and scales as $\sim l^{2.1}$.

Comparison to other algorithms

In Table II, the PSO approach is compared to SMO and a decomposition method, SVM^{light} [3]. WinSVM was developed by C. Longbin [6] from the SVM^{light} source code, and was used as an implementation of SMO. Unlike these methods, the current PSO algorithm does not make use of caching and shrinking to optimize its speed.

Results similar to Table I indicate that SVM^{light} gives the fastest rate of convergence with a working set size $q = 8$, which is used in Table II's comparison.

Experimental results show SMO scaling as $\sim l^{2.8}$, and SVM^{light} scaling as $\sim l^{3.0}$. Both these algorithms are substantially faster than training a SVM with PSO on the MNIST dataset, but the PSO approach shows better scaling abilities ($\sim l^{2.1}$). Due to the fact that the PSO training algorithm starts with a very small set of possible support vectors, with all other α_i set to zero, the PSO method consistently finds a few support vectors less than the other approaches.

The main drawback from the current PSO approach is its slow performance times, but from this initial study many optimizations can be implemented on both decomposition and PSO methods.

VII. CONCLUSION

It was shown that a PSO can be used to train a SVM. Some properties of LPSO make it particularly useful to solve the SVM constrained QP problem. The PSO algorithm is simple to implement, and does not require any background of numerical methods. Accurate and scalable training results were shown on the MNIST dataset, with the PSO algorithm finding fewer support vectors and better scalability than other approaches. Although the algorithm is simple, its speed poses a practical bottleneck, which can be improved from this initial study.

ACKNOWLEDGMENT

The financial assistance of the National Research Foundation towards this research is hereby acknowledged. Opinions expressed in this paper and conclusions arrived at, are those of the authors and not necessarily to be attributed to the National Research Foundation.

REFERENCES

[1] B.E. Boser, I.M. Guyon, and V.N. Vapnik, "A training algorithm for optimal margin classifiers," in D. Haussler, editor, *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*, pages 144-152, Pittsburgh, PA, 1992. ACM Press.

[2] R. Fletcher, *Practical Methods of Optimization*. John Wiley and Sons, Inc., 2nd edition, 1987.

[3] T. Joachims, "Making large-scale SVM learning practical," in *Advances in Kernel Methods – Support Vector Learning*, B. Schölkopf, C.J.C Burges, and A.J. Smola, editors, pages 169-184. MIT Press, Cambridge, MA, 1999.

[4] J. Kennedy and R.C. Eberhart, "Particle swarm optimization," in *Proceedings of the IEEE International Conference on Neural Networks, IV*, pages 1942-1948, 1995.

[5] P. Laskov, "Feasible direction decomposition algorithms for training support vector machines," in *Machine Learning, Volume 46*, N. Cristianini, C. Campbell, and Chris Burges, editors, pages 315-349, 2002.

[6] C. Longbin, http://liama.ia.ac.cn/PersonalPage/lbchen/, Institute of Automation, Chinese Academy of Sciences (CASIA).

[7] MNIST Optical Character Database at AT&T Research, http://yann.lecun.com/exdb/mnist/.

[8] E. Osuna, R. Freund, and F. Girosi, "Support vector machines: Training and applications," A.I. Memo AIM-1602, MIT A.I. Lab, 1996.

[9] E. Osuna, R. Freund, and F. Girosi, "An improved training algorithm for support vector machines," in *Neural Networks for Signal Processing VII – Proceedings of the 1997 IEEE Workshop*, J. Principe, L. Gile, N. Morgan, and E. Wilson, editors, pages 276-285. IEEE, New York, 1997.

[10] U. Paquet and A.P. Engelbrecht, "Particle swarms for equality-constrained optimization," submitted to *IEEE Transactions on Evolutionary Computation*.

[11] J. Platt, "Fast training of support vector machines using sequential minimal optimization," in *Advances in Kernel Methods – Support Vector Learning*, B. Schölkopf, C.J.C Burges, and A.J. Smola, editors, pages 185-208. MIT Press, Cambridge, MA, 1999.

[12] Y.H. Shi and R.C. Eberhart, "A modified particle swarm optimizer," in *IEEE International Conference on Evolutionary Computation*, Anchorage, Alaska, 1998.

[13] F. van den Bergh, *An analysis of particle swarm optimizers*, PhD Thesis, Department of Computer Science, University of Pretoria, 2002.

[14] V. Vapnik, *Estimation of Dependences Based on Empirical Data [in Russian]*, Nauka, Moscow, 1979. (English translation: Springer Verlag, New York, 1982.)