CCIT Report #557 September 2005

Support Vector Machine via Sequential Subspace Optimization

Guy Narkiss guyn@siglab.technion.ac.il
Michael Zibulevsky mzib@ee.technion.ac.il
Department of Electrical Engineering
Technion - Israel Institute of Technology
Haifa 32000, Israel

Abstract

We present an optimization engine for large scale pattern recognition using Support Vector Machine (SVM). Our treatment is based on conversion of soft-margin SVM constrained optimization problem to an unconstrained form, and solving it using newly developed Sequential Subspace Optimization (SESOP) method. SESOP is a general tool for large-scale smooth unconstrained optimization. At each iteration the method minimizes the objective function over a subspace spanned by the current gradient and by directions of few previous steps and gradients. Following an approach of A. Nemirovski, we also include into the search subspace the direction from the starting point to the current point, and a weighted sum of all previous gradients: this provides the worst case optimality of the method. The subspace optimization can be performed extremely fast in the cases when the objective function is a combination of expensive linear mappings with computationally cheap non-linear functions, like in the unconstrained SVM problem. Presented numerical results demonstrate high efficiency of the method.

Keywords: Large-scale optimization, pattern recognition, Support Vector Machine, conjugate gradients, subspace optimization

1. Introduction

The problem of large-scale binary data classification arises in many applications, like recognition of text, hand-written characters, images, medical diagnostics, etc. Quite often, the number of features or examples is very large, say $10^4 - 10^7$ and more, and there is a need for algorithms, for which storage requirement and computational cost per iteration grow not more than linearly in those parameters. One way to treat such problems with SVM (Vapnik, 1998) is to convert a constrained SVM problem into an unconstrained one and solve it with an optimization method having non-expensive iteration cost and storage. An appropriate optimization algorithm of this type is the conjugate gradient (CG) method (Hestenes and Stiefel, 1952; Gill et al., 1981; Shewchuk, 1994). It is known that CG worst case convergence rate for quadratic problems is $O(k^{-2})$ (in terms of objective function calculations), where k is the iteration count. This rate of convergence is independent of the problem size and is optimal, *i.e.* it coincides with the complexity of convex smooth unconstrained optimization (see *e.g.* Nemirovski (1994)). However the standard extensions of CG to nonlinear functions by Fletcher-Reeves and Polak-Ribière (see *e.g.* Shewchuk (1994)) are no longer worst-case optimal.

Nemirovski (1982) suggested a method for smooth unconstrained convex optimization with the optimal worst case convergence rate $O(k^{-2})$. The method consists of sequential minimization of the objective function over subspaces spanned by the following three vectors:

- $\mathbf{d}_k^1 = \mathbf{x}_k - \mathbf{x}_0$, where $\mathbf{x}_k$ – current iterate; $\mathbf{x}_0$ – starting point;
- $\mathbf{d}_k^2 = \sum_{i=0}^{k-1} w_i \mathbf{g}(\mathbf{x}_i)$ – weighted sum of previous gradients with specified weights w_i;
- $\mathbf{g}(\mathbf{x}_k)$ – current gradient.

Note that this method is optimal with respect to the number of subspace minimizations, however the overall number of function/gradient evaluations is suboptimal by a factor of $\log k$. Nemirovski also suggested methods with 2-d and even 1-d subspace optimization instead of 3-d one (Nemirovski and Yudin, 1983). Further progress in this direction was achieved by Nesterov (Nesterov, 1983, 2003; Nemirovski, 1994), who proposed a worst-case optimal algorithm with no line search, which achieves the optimal complexity in terms of function/gradient evaluations. In practical situations, however (in contrast to the worst case), the mentioned methods often behave even poorer than conventional algorithms like non-linear CG or Truncated Newton (TN) (see *e.g.* Gill et al. (1981) for a description of TN).

Our crucial observation is that for many important problems subspace optimization can be performed extremely fast. This happens, for example, when the objective function is a combination of expensive linear mappings with computationally cheap non-linear functions: such a situation is typical in many applications, including SVM-based pattern recognition. Motivated by this observation we tend to increase the dimensionality of the search subspaces and use quite accurate subspace optimization (contrary to the trends of Nemirovski and Nesterov).

SESOP algorithm can be performed in several modes. Using just 2-d subspace optimizations in directions of the current gradient $\mathbf{g}(\mathbf{x}_k)$ and of the previous step $\mathbf{p}_k = \mathbf{x}_k - \mathbf{x}_{k-1}$, we get a method, which coincides with CG, when the problem becomes quadratic. This property is favorable in the proximity of the solution, where the problem has a good quadratic approximation. Globally (in our experience) this method behaves better and is more stable then Polak-Ribière CG. Including two additional Nemirovski directions: $\mathbf{d}_k^1 = \mathbf{x}_k - \mathbf{x}_0$ and $\mathbf{d}_k^2 = \sum_{i=0}^{k-1} w_i \mathbf{g}(\mathbf{x}_i)$ with appropriate weights w_i, we guarantee the worst-case optimality of the method. Including more previous steps and gradients into the optimization subspace helps to further reduce the number of iterations, while moderately increasing the iteration cost.

The paper is organized as follows. In Section 2 we describe the SVM-based pattern recognition and its representation as an unconstrained optimization problem. In Section 3 we introduce the sequential subspace optimization algorithm and discuss its properties. In Section 4 we present a way to conduct an efficient minimization of functions in subspace. Section 5 is devoted to computational experiments. Finally, conclusions are summarized in Section 6.

2. SVM-based Pattern Recognition

Linear SVM Consider a binary classification problem with m training examples, $\{\mathbf{x}_i, y_i\}_{i=1}^m$ where the feature vectors $\mathbf{x}_i \in \mathcal{R}^n$ and the labels $y_i \in \{-1, +1\}$. Our goal is to choose a separating hyperplane from the family of affine functions $f = \mathbf{w}^T\mathbf{x} + b$. The classifier is $\text{sign}(f(\mathbf{x}))$. The parameters to be learnt from the data are $\mathbf{w}$ and b. This methodology is called linear SVM (SVM whose feature space is the same as the input space of the problem). For linearly separable data, we need to solve the following quadratic program

$$\min_{\mathbf{w},b} \|\mathbf{w}\|_2^2 \quad s.t. \quad y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \quad \forall i, \tag{1}$$

which is equivalent to finding the widest strip which separates the sets and does not include any element inside (maximal geometric margin). The width of the strip is

$$\gamma = \frac{2}{\|\mathbf{w}_{opt}\|_2}. \tag{2}$$

For linearly non-separable problems, a soft margin approach is needed (Cortes and Vapnik, 1995). We define the slack margin vector $\xi = [\xi_1, \ldots, \xi_m]^T$ to represent violations of the original constraints,

$$\xi_i \equiv max[0, 1 - y_i f(\mathbf{x}_i)]. \tag{3}$$

The corresponding constrained problem is

$$\min_{\mathbf{w},b} \|\mathbf{w}\|_2^2 + C\sum_i \xi_i^q \tag{4}$$

$$\begin{aligned} s.t. \quad & y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i \\ & \xi_i \geq 0 \quad \forall i, \end{aligned} \tag{5}$$

where C is a regularization parameter, $q = 1$ for a linear penalty on the slacks, and $q = 2$ for a quadratic penalty. Problem (4, 5) is usually referred to as L_2-SVM due to l_2-norm of $\mathbf{w}$. Another approach (see Fung and Mangasarian (2000)) is to minimize the l_1-norm of $\mathbf{w}$. The justification for this approach is that components of $\mathbf{w}$ which do not contribute to the separation will be optimized to zero. This is equivalent to feature selection process of the data, *i.e.* features of the data which are not useful for separation will be ignored. The general expression for L_1 and L_2-SVM is

$$\min_{\mathbf{w},b} \|\mathbf{w}\|_p^p + C\sum_i \xi_i^q \quad s.t.\,(5), \tag{6}$$

with $p = 1$ or 2, and $q = 1$ or 2.

Kernel-Based SVM For specific data sets, an appropriate nonlinear mapping $\mathbf{x} \mapsto \phi(\mathbf{x})$ can be used to embed the original features into a Hilbert feature space $\mathcal{F}$ with inner product $\langle ., . \rangle$. One can seek for an affine classifier on the image $\phi(\mathbf{X}) \subset \mathcal{F}$. Usually, explicit knowledge of $\phi(\mathbf{x})$ is not required, it is sufficient to use the kernel

$$K(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle. \tag{7}$$

The L_2-SVM in the feature space $\mathcal{F} \supset \phi(\mathbf{X})$ is:

$$\min_{\mathbf{w}\in\mathcal{F},b} \langle \mathbf{w}, \mathbf{w} \rangle + C \sum_i \xi_i^q, \qquad q = 1 \text{ or } 2 \tag{8}$$

$$\begin{aligned} s.t. \quad & y_i(\langle \mathbf{w}, \phi(\mathbf{x}_i) \rangle + b) \geq 1 - \xi_i \\ & \xi_i \geq 0 \quad \forall i, \end{aligned} \tag{9}$$

The optimal $\mathbf{w}$ belongs to the $\text{Span}(\phi(\mathbf{x}_1), \ldots, \phi(\mathbf{x}_m))$, therefore we can set

$$\mathbf{w} = \sum_{j=1}^{m} \alpha_j y_j \phi(\mathbf{x}_j). \tag{10}$$

Substituting this into (8), we get:

$$\begin{aligned} & \min_{\alpha,b} \sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle + C \sum_i \xi_i^q \\ & s.t. \quad y_i(\sum_{j=1}^{m} \alpha_j y_j \langle \phi(\mathbf{x}_j), \phi(\mathbf{x}_i) \rangle + b) \geq 1 - \xi_i \\ & \qquad \xi_i \geq 0 \quad \forall i, \end{aligned} \tag{11}$$

and using the kernel definition (7):

$$\min_{\alpha,b} \sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j) + C \sum_i \xi_i^q, \qquad q = 1 \text{ or } 2 \tag{12}$$

$$\begin{aligned} s.t. \quad & y_i(\sum_{j=1}^{m} \alpha_j y_j K(\mathbf{x}_j, \mathbf{x}_i) + b) \geq 1 - \xi_i \\ & \xi_i \geq 0 \quad \forall i. \end{aligned} \tag{13}$$

A classifier can be defined as the sign of the following kernel-based affine function:

$$f(\mathbf{x}) = \sum_{i=1}^{m} K(\mathbf{x}, \mathbf{x}_i) y_i \alpha_i + b. \tag{14}$$

Unconstrained SVM formulation In order to use unconstrained optimization techniques, we convert the constrained problem (6) into equivalent unconstrained form. When $q = 1$, we use the following penalty function

$$\tilde{\varphi}_1(t) = \begin{cases} 0 & \text{for } t \leq -1, \\ t + 1 & \text{otherwise,} \end{cases} \tag{15}$$

or equivalently

$$\tilde{\varphi}_1(t) = \frac{1}{2}(|t + 1| + t + 1). \tag{16}$$

When $q = 2$, we set

$$\tilde{\varphi}_2(t) = \begin{cases} 0 & \text{for } t \le -1, \\ (t+1)^2 & \text{otherwise,} \end{cases} \tag{17}$$

The unconstrained representation of (6) is:

$$\min_{\mathbf{w},b} \|\mathbf{w}\|_p^p + C \sum_{i=1}^{m} \tilde{\varphi}_q(-y_i(\mathbf{w}^T \mathbf{x}_i + b)). \tag{18}$$

The kernel SVM (12, 13) can also be represented in the unconstrained form:

$$\min_{\alpha,b} \sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j) + C \sum_{i=1}^{m} \tilde{\varphi}_q(-y_i(\sum_{j=1}^{m} \alpha_j y_j K(\mathbf{x}_j, \mathbf{x}_i) + b)). \tag{19}$$

In the following we use a vector notation

$$\mathbf{y} = [y_1, \cdots, y_m]^T; \quad \tilde{\mathbf{x}}_i = -y_i \mathbf{x}_i; \quad \tilde{\mathbf{X}} = [\tilde{\mathbf{x}}_1, \cdots, \tilde{\mathbf{x}}_m];$$

$$\tilde{\mathbf{K}} - m \times m \text{ matrix with the elements } \tilde{K}_{ij} = y_i y_j K(\mathbf{x}_i, \mathbf{x}_j).$$

We also vectorize generic function notation: $f(\mathbf{s}) = (f(s_1) \dots f(s_N))^T$. In this way the unconstrained linear and kernel SVM (18), (19) can be represented correspondingly as

$$\min_{\mathbf{w},b} \|\mathbf{w}\|_p^p + C\mathbf{1}^T \tilde{\varphi}_q(\tilde{\mathbf{X}}^T \mathbf{w} - b\mathbf{y}); \tag{20}$$

$$\min_{\alpha,b} \ \alpha^T \tilde{\mathbf{K}} \alpha + C\mathbf{1}^T \tilde{\varphi}_q(-\tilde{\mathbf{K}}\alpha - b\mathbf{y}). \tag{21}$$

where $\mathbf{1}$ denotes vector of ones of the corresponding size.

Smoothing the objective function In the cases when parameters p or q are equal to one in (20) or (21), the l_1-norm or the penalty function $\tilde{\varphi}(\cdot)$ are non-smooth functions. In order to use smooth optimization techniques, we smoothly approximate these functions using an approximation of the absolute value. In this work we use

$$|s| \approx \psi_\epsilon(s) = \epsilon\left(\left|\frac{s}{\epsilon}\right| + \frac{1}{|\frac{s}{\epsilon}| + 1} - 1\right), \quad \epsilon > 0, \tag{22}$$

where ϵ is a smoothing parameter. The approximation becomes accurate when $\epsilon \to 0$. This function is faster to compute then other known to us absolute value approximations. Changing the absolute value to $\psi_\epsilon(\cdot)$ in (16), we get a smoothed version of $\tilde{\varphi}_1(\cdot)$:

$$\varphi(t) = \frac{1}{2}(\psi_\epsilon(t+1) + t + 1). \tag{23}$$

Below we will omit indices ϵ and q. The smoothed version of (20) with $p = q = 1$ can be written as

$$\min_{\mathbf{w},b} \mathbf{1}^T \psi(\mathbf{w}) + C\mathbf{1}^T \varphi(\tilde{\mathbf{X}}^T \mathbf{w} - b\mathbf{y}). \tag{24}$$

Note again, that if one chooses L_2-SVM ($p = 2$) or the quadratic penalty on the slacks ($q = 2$), no smoothing is needed in the corresponding terms of (20), (21).

Sequential update of the smoothing parameter Whenever a very small value of the smoothing parameter is required for good performance of the classifier, the direct unconstrained optimization may become difficult. In this situation one can use a sequential nested optimization: Starting with a moderate value of ϵ, optimize the objective function to a reasonable accuracy, then reduce ϵ by some factor and perform the optimization again, starting from the currently available solution, and so on... Another alternative is to use the smoothing method of multipliers (Zibulevsky, 1996, 2003), which combines the ideas of Lagrange multipliers with the ideas of smoothing of non-smooth functions, and provides a very accurate solution.

Parameters tuning Adjustment of the regularization parameter C, the smoothing parameter ϵ and the optimization stopping criterion constants is conducted by validation: the data is split into 'training' and 'validation' sets; we optimize (24) on the 'training' set, and count the separation errors on the 'validation' set. This process is repeated using different parameters, and the best combination is chosen. The performance of the final 'trained' classification algorithm is measured on 'test' data, which is not a part of the 'training' or 'validation' sets.

3. Sequential Subspace Optimization (SESOP) algorithm

In this section we present SESOP algorithm (Narkiss and Zibulevsky, 2005), which is a general method for smooth unconstrained optimization. In the following we will use it as an optimization engine for the unconstrained SVM.

For an unconstrained smooth minimization problem

$$\min_{\mathbf{x}\in\mathcal{R}^n} f(\mathbf{x}). \tag{25}$$

SESOP performs an iterative sequence of optimizations of the objective function f over affine subspaces spanned by previous steps and gradients, as described below. We will define the algorithm with its various modes, and discuss its properties. A detailed complexity analysis appears in (Narkiss and Zibulevsky, 2005).

3.1 Construction of subspace structure

In order to define the subspace structure, denote the following sets of directions:

1. **Current gradient**: $\mathbf{g}(\mathbf{x}_k)$ - the gradient at the k'th point $\mathbf{x}_k$.
2. **Nemirovski directions**:

$$\begin{aligned} \mathbf{d}_k^{(1)} &= \mathbf{x}_k - \mathbf{x}_0 \\ \mathbf{d}_k^{(2)} &= \sum_{i=0}^{k} w_i \mathbf{g}(\mathbf{x}_i), \end{aligned} \tag{26}$$

where w_k is defined by

$$w_k = \begin{cases} 1 & \text{for } k = 0 \\ \frac{1}{2} + \sqrt{\frac{1}{4} + w_{k-1}^2} & \text{for } k > 0. \end{cases} \tag{27}$$

3. **Previous directions**:

$$\mathbf{p}_{k-i} = \mathbf{x}_{k-i} - \mathbf{x}_{k-i-1}, \quad i = 0, \ldots, s_1. \tag{28}$$

4. **Previous gradients**:

$$\mathbf{g}_{k-i}, \quad i = 1, \ldots, s_2. \tag{29}$$

The mandatory direction 1 and any subset of directions 2 - 4 can be used to define the subspace structure. We will discuss possible considerations for several constellations.

3.2 Algorithm summary

Let $\mathbf{D}$ be a matrix of the chosen M (column) directions described in Subsection 3.1, and α a column vector of M coefficients. At every iteration we find a new direction $\mathbf{D}\alpha$ in the subspace spanned by the columns of $\mathbf{D}$. The algorithm is summarized as follows:

1. Initialize $\mathbf{x}_k = \mathbf{x}_0$, $\mathbf{D} = \mathbf{D}_0 = \mathbf{g}(\mathbf{x}_0)$.
2. Normalize the columns of $\mathbf{D}$.
3. Find
$$\alpha^* = \underset{\alpha}{\operatorname{argmin}}\, f\big(\mathbf{x}_k + \mathbf{D}\alpha\big). \tag{30}$$
4. Update current iterate:
$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{D}\alpha^*. \tag{31}$$
5. Update matrix $\mathbf{D}$ according to the chosen set of subspace directions in Subsection 3.1.
6. Repeat steps 2 - 5 until convergence.

Implementation notes:

1. The choice of the subspace dimension M, is a trade off between the increase in computational cost per iteration and the possible decrease in iterations number. Using just 2-d subspace optimizations in directions of the current gradient $\mathbf{g}(\mathbf{x}_k)$ and of the previous step $\mathbf{p}_k$, we get a method, which coincides with CG, when the problem becomes quadratic. This property is favorable in the proximity of the solution, where the problem has a good quadratic approximation. Also globally (in our experience) this method behaves better and is more stable then Polak-Ribière CG.
2. Including two additional Nemirovski directions (26), we guarantee the worst-case optimality of the method (when solving concrete non-worst case classes of problems, these two directions may not bring significant improvement, therefore can be omitted after careful testing). Including more previous steps and gradients into the optimization subspace helps to further reduce the number of iterations, while moderately increasing the iteration cost. The guiding principle is that the dimension M of the search subspace should not be higher than a few tens or maybe hundreds of directions.

3. For the first $M-1$ iterations, some directions may be a combination of other directions, or may not exist. We take advantage of this fact, to decrease the size of matrix $\mathbf{D}$ for these iterations, and reduce the computation load. After more than M iterations, the size of $\mathbf{D}$ does not change.

4. **Preconditioning** A common practice for optimization speedup is to use a preconditioner matrix. One can use a preconditioned gradient $\mathbf{Mg}(\mathbf{x}_k)$ instead of $\mathbf{g}(\mathbf{x}_k)$, where the matrix $\mathbf{M}$ approximates the inverse of the Hessian at point $\mathbf{x}_k$. There is a trade off between the pre-conditioner calculation time, and the optimization runtime saving.

5. **Newton method in subspace optimization** Basically SESOP is a first order method, which can work using only first order derivatives. In many cases the objective function is twice differentiable, but the Hessian cannot be used due to memory and other limitations. However, in the subspace optimization (step 3 of the algorithm), we often use Newton method because of the small size of this auxiliary problem.

4. Reduced computations for subspace minimization

Consider a function of the form

$$f(\mathbf{x}) = \varphi(\mathbf{Ax}) + \psi(\mathbf{x}). \tag{32}$$

Such functions are very common in many applications. The multiplications $\mathbf{Ax}$ and $\mathbf{A}^T\mathbf{y}$ are usually the most computationally expensive operations for calculating the function f and its gradient. Our aim is to construct an optimization algorithm which will avoid such operations whenever possible. It is worthwhile emphasizing that we "change the rules" for comparison of computation load between different optimization algorithms. Instead of the common method of counting the number of function and gradient calculations, we will count the number of matrix-vector multiplications. Methods based on subspace optimization often iterate the multi-dimensional minimizer incrementally in the form

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \sum_{i=1}^{M} \alpha_i \mathbf{r}_i, \tag{33}$$

where the coefficients α_i, the directions $\mathbf{r}_i$ and the number of directions M are determined according to the specific optimization scheme. Such a framework allows us to save a large part of the matrix-vector multiplications originally needed for calculation of the objective function value (32). The term $\mathbf{Ax}_{k+1}$ can be broken into

$$\begin{aligned} \mathbf{Ax}_{k+1} &= \mathbf{Ax}_k + \mathbf{A}\sum_{i=1}^{M} \alpha_i \mathbf{r}_i \\ &= \mathbf{Ax}_k + \sum_{i=1}^{M} \alpha_i \mathbf{Ar}_i \\ &= \mathbf{v}_0 + \sum_{i=1}^{M} \mathbf{v}_i \alpha_i, \end{aligned} \tag{34}$$

where $\mathbf{v}_i = \mathbf{A}\mathbf{r}_i$. For each new direction $\mathbf{r}_i$ we need to calculate and save one vector ($\mathbf{v}_i$). Total memory requirement is M directions $\mathbf{r}_i$, M matrix-vector multiplications results $\mathbf{v}_i$ and one accumulative vector term $\mathbf{v}_0$. Obviously, as the data dimension n increases, the complexity reduction from using (34) is more significant. For line search operation along a single direction, or subspace minimization along several directions, there is no need to perform any matrix-vector multiplication, since the function and its gradient with respect to α are gained using the pre-calculated set of vectors $\mathbf{v}_i$.

Computational complexity of outer and inner iteration Consider for example problem (24). Assume that the absolute value smooth approximation function requires $K = 5$ arithmetical operations, so that the penalty function (23) costs $k = K + 3 = 8$ operations.

It is easy to see that the gradient of the objective function (24) requires $nK + mn + km + nM + m$ operations (using fast computations described above). Each outer iteration requires up to $n(m + 3)$ additional arithmetical operation for the update of the direction matrix $\mathbf{D}$. If the data is significantly sparse, the amount of operations drops accordingly.

At the inner iteration of the subspace optimization (30), the basic calculation is usually function and gradient evaluation with respect to α. The cost of function evaluation is $n + nK + m + mk + nM + m$ operations, and gradient evaluation in subspace requires $nM+nK+mM+mk$ additional operations. When Newton method is used for the subspace optimization, calculation of the Hessian with respect to α will require $nK+nM^2+mk+mM^2$ operations.

5. Numerical experiments

We compared the performance of several algorithms for problem (24) with several large-scale data sets. The algorithms used were

1. SESOP with various numbers of directions;
2. Polak-Ribière nonlinear conjugate gradients;
3. Truncated Newton;
4. Nesterov method (Nesterov, 1983, 2003).

In our experiments the best line search methods were cubic interpolation for CG, and backtracking search with Armijo rule for all other methods, see for example (Gill et al., 1981). Newton method was used for subspace minimization in SESOP algorithm.

The parameters for comparison between algorithms are number of iterations, normalized to two matrix-vector multiplications per iteration, as shown in Table 1, and computational time [1].

Notation: SESOPi is an abbreviation for SESOP using directions of i previous steps. When we do not include Nemirovski directions in our scheme, we use the notation SESOPi^-. CG_{fast} means CG with reduced computations for line search minimization, as explained in Section 4. All data sets and their description are available at the UCI repository (Hettich

1. Experiments were conducted in MATLAB, on Intel® Xenon™ CPU 2.8GHz core with 2Gb memory, running Linux

Method	**Matrix-vector mult. per iteration**
SESOP	2
Conjugate Gradient	2
Truncated Newton	2 per inner CG iter. + 2 per outer Newton iter.
Nesterov	3

Table 1: Number of heavy operations for different optimization methods.

Data set	**Num. of features**	**Num. of training vectors**	**Num. of validation vectors**	**Num. of test vectors**	**Features sparsity** [%]	**Error** [%]
Arcene	10000	50	50	100	54	14
Gisette	5000	1750	1750	3500	18	4
Dexter	20000	150	150	300	0.47	15
Dorothea	100000	400	400	350	0.91	7.42
Internet_ads	1558	1093	1094	1093	1	3.93
Madelon	500	500	500	1300	100	45

Table 2: Data sets short description. Error percentage was measured on the test set.

et al., 1998) and NIPS "feature selection challenge 2003" (Guyon and Gunn., 2003). A short summary of these data sets and our L_1-SVM error percentage is presented in Table 2.

Figures 1,2 show the progress with iterations in objective function and in validation error, for the data set 'Internet_ads'. As we see, SESOP8 provides few times faster decrease of the objective function comparing to CG and TN. Also the validation error is reduced and stabilized by SESOP8 much earlier.

Tables 3,4 present number of iterations and runtime of different algorithms for several data sets. We have used two stopping criteria: norm of the gradient $\|\nabla f\| \leq 10^{-5}$, and stabilization of the validation set error ('good results'). Nesterov method was significantly slower than the other methods, and is not presented in here. SESOP has solved all the problems, while CG failed to converge in 5000 iterations at data set "Madelon". The most stable performance was demonstrated by SESOP8, which converged on average about twice faster then CG, and about 10 times faster then TN (see Table 4).

We also have conducted some experiments with diagonal preconditioning, but they did not show any benefit.

6. Conclusions

We have demonstrated that SESOP algorithm is an efficient tool for solving SVM problem in unconstrained formulation. The main advantages of SESOP are optimal worst-case complexity for smooth convex unconstrained problems, low memory requirements and low computation load per iteration. Our numerical experiments were restricted to the smoothed

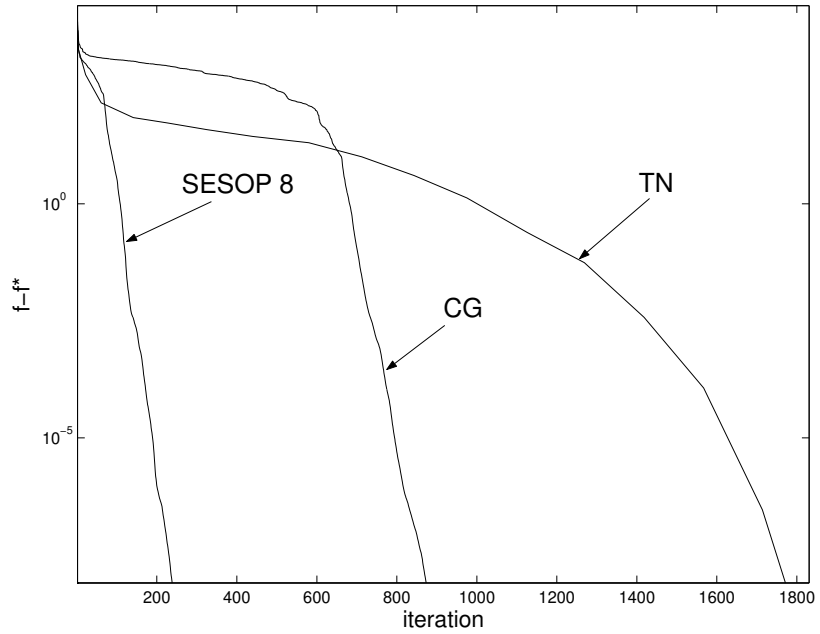

Figure 1: Accuracy of the objective function $f(\mathbf{x}_k) - f_{\text{optimal}}$ with iterations for data set 'Internet_ads' [log scale]

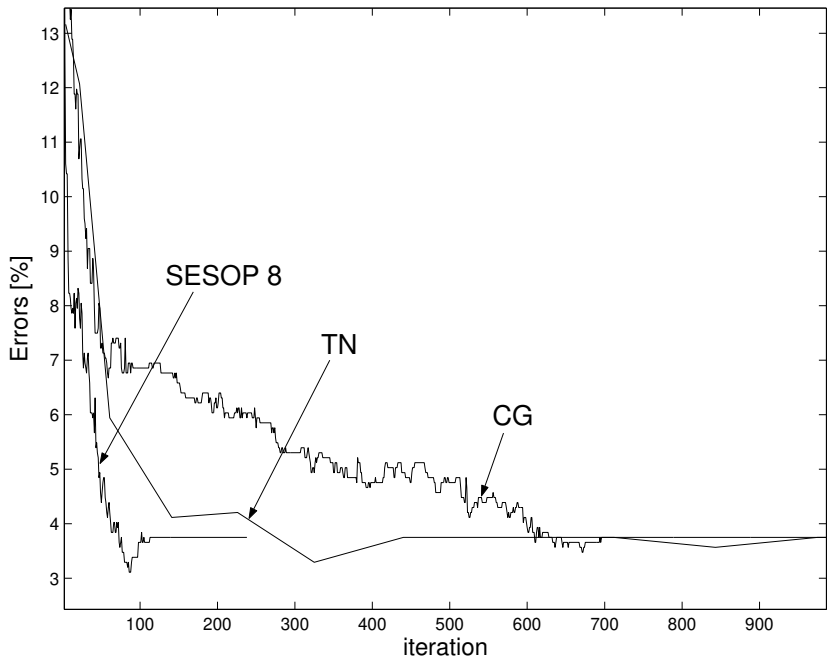

Figure 2: Validation error with iterations for data set 'Internet_ads'

		Convergence		Good results	
Data set	**Method**	**iter**	**time**	**iter**	**time**
Arcene	SESOP0	601	68.22	118	11.39
	SESOP1$^-$	148	10.61	63	5.62
	SESOP1	261	17.59	128	8.76
	SESOP8	57	6.36	20	3.5
	SESOP32	24	4.16	14	2.97
	CG	174	22.24	111	12.86
	CG$_{fast}$	164	10.01	91	4.66
	TN	6039	28.53	5890	27.58
Gisette	SESOP0	377	49	138	17.28
	SESOP1$^-$	96	13.07	46	7.74
	SESOP1	232	27.46	119	14.44
	SESOP8	74	13.54	34	8.35
	SESOP32	49	13.08	27	8.32
	CG	158	298.77	119	225.41
	CG$_{fast}$	196	24.47	121	14.23
	TN	33827	2153.97	33750	2147.72
Dexter	SESOP0	56	9.03	11	2.71
	SESOP1$^-$	26	4.98	10	2.61
	SESOP1	52	7.72	18	3.38
	SESOP8	22	6.15	8	2.52
	SESOP32	21	6.35	8	2.46
	CG	61	55.14	22	19.79
	CG$_{fast}$	61	13.02	22	4.36
	TN	268	11.76	220	9.59
Internet_ads	SESOP0	3131	667.02	940	149.56
	SESOP1$^-$	381	54.5	225	32.42
	SESOP1	599	86.83	420	60.19
	SESOP8	237	39.06	110	20.18
	SESOP32	180	43.7	82	24.17
	CG	750	367.58	414	131.83
	CG$_{fast}$	1035	163.98	696	99.93
	TN	2378	39.24	1210	20.65
Dorothea	SESOP0	901	1566.04	19	21.9
	SESOP1$^-$	189	182.44	15	18.49
	SESOP1	248	219.35	12	11.83
	SESOP8	156	216.2	12	19.53
	SESOP32	144	409.9	12	19.69
	CG	246	1663.11	7	22.19
	CG$_{fast}$	246	293.22	7	4.36
	TN	849	248.35	362	110.64
Madelon	SESOP0	∞	∞	2402	58.22
	SESOP1$^-$	478	5.53	162	2.06
	SESOP1	4493	164.79	3552	112.02
	SESOP8	334	4.23	115	1.82
	SESOP32	232	5.53	79	2.46
	CG	∞	∞	∞	∞
	CG$_{fast}$	∞	∞	∞	∞
	TN	35170	69.77	5000	11.06

Table 3: Iterations and CPU runtime [sec] to convergence ($\|\nabla f\| \leq 10^{-5}$), and to 'good results' (stabilization of the error rate on the validation set); ∞ – no convergence in 5000 iterations.

	Convergence				Good results			
Method	**iter**		**time**		**iter**		**time**	
SESOP0	∞	(5066)	∞	(2359.3)	3628	(1226)	261	(202.8)
SESOP1$^-$	1318	(840)	271.1	(265.6)	5210	(359)	68.9	(66.88)
SESOP1	5885	(1392)	523.7	(358.9)	4249	(697)	210.6	(98.6)
SESOP8	880	(546)	285.54	(281.31)	299	(184)	55.9	(54.08)
SESOP32	650	(418)	482.72	(477.19)	222	(143)	60.07	(57.61)
CG	∞	(1389)	∞	(2406.8)	∞	(673)	∞	(412.1)
CG$_{fast}$	∞	(1702)	∞	(504.7)	∞	(937)	∞	(127.5)
TN	78531	(43361)	2552	(2482)	46432	(41432)	2327	(2315)

Table 4: Total iterations/time (over Table 3). The numbers in brackets are the totals without including the data set 'Madelon'.

version of L_1-SVM with linear penalty on slack variables, however we believe that other forms of SVM (L_2; kernel-based; having quadratic penalty on slacks) can be approached with SESOP. Our optimism is partially supported by good results with this method in other fields, like computerized tomography and image denoising with Basis Pursuit, reported in (Narkiss and Zibulevsky, 2005).
Unconstrained optimization is a building block for many constrained optimization techniques, which makes SESOP a promising candidate for embedding into many existing solvers.

Acknowledgments

We are grateful to Arkadi Nemirovski for his most useful advice and support. Our thanks to Dori Peleg for the references to pattern recognition data sets. This research has been supported in parts by the "Dvorah" fund of the Technion and by the HASSIP Research Network Program HPRN-CT-2002-00285, sponsored by the European Commission. The research was carried out in the Ollendorff Minerva Center, funded through the BMBF.

References

C. Cortes and V.N. Vapnik. Support vector networks. *Machine Learning*, 20:273–297, 1995.

Glenn Fung and O. L. Mangasarian. Data selection for support vector machine classifiers. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 64–70, 2000.

Philip E. Gill, Walter Murray, and Margareth H. Wright. *Practical Optimization*. Academic Press, 1981.

I. Guyon and S. Gunn. NIPS feature selection challenge. 2003. `http://www.nipsfsc.ecs.soton.ac.uk/datasets/`.

Magnus R. Hestenes and Eduard Stiefel. Methods of conjugate gradients for solving linear systems. *J. Res. Natl. Bur. Stand.*, 49:409–436, 1952.

S. Hettich, C.L. Blake, and C.J. Merz. UCI repository of machine learning databases. *University of California, Irvine, Dept. of Information and Computer Sciences*, 1998. http://www.ics.uci.edu/∼mlearn/MLRepository.html.

Guy Narkiss and Michael Zibulevsky. Sequential subspace optimization method for large-scale unconstrained optimization. Technical report, Technion - The Israel Institute of Technology, faculty of Electrical Engineering, 2005.

A. Nemirovski. Orth-method for smooth convex optimization (in russian). *Izvestia AN SSSR, Ser. Tekhnicheskaya Kibernetika (the journal is translated to English as Engineering Cybernetics. Soviet J. Computer & Systems Sci.)*, 2, 1982.

A. Nemirovski. *Efficient methods in convex programming.* Technion - The Israel Institute of Technology, faculty of Industrial Engineering and Managments, 1994.

A. Nemirovski and D. Yudin. Information-based complexity of mathematical programming (in russian). *Izvestia AN SSSR, Ser. Tekhnicheskaya Kibernetika (the journal is translated to English as Engineering Cybernetics. Soviet J. Computer & Systems Sci.)*, 1, 1983.

Yu. Nesterov. A method for unconstrained convex minimization problem with the rate of convergence $o(1/n^2)$ (in russian). *Doklady AN SSSR (the journal is translated to English as Soviet Math. Docl.)*, 269(3):543–547, 1983.

Yu. Nesterov. Smooth minimization of non-smooth functions. CORE discussion paper, Universitè catholique de Louvain, Belgium, 2003.

J. Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical report, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1994.

Vladimir Vapnik. *Statistical learning theory.* Wiley-Interscience, New York, 1998.

M. Zibulevsky. Smoothing method of multipliers for sum-max problems. Technical report, Dept. of Elec. Eng., Technion, 2003. http://ie.technion.ac.il/˜mcib/.

Michael Zibulevsky. *Penalty/Barrier Multiplier Methods for Large-Scale Nonlinear and Semidefinite Programming.* PhD thesis, Technion – Israel Institute of Technology, 1996. http://ie.technion.ac.il/˜mcib/.