An Improved Training Algorithm for Support Vector Machines

(to appear in the *Proc. of IEEE NNSP'97*, Amelia Island, FL, 24–26 Sep., 1997)

Edgar Osuna
C.B.C.L.
MIT E25-201
Cambridge, MA, 02139
eosuna@ai.mit.edu
tel: (617) 252 1723

Robert Freund
O.R. Center
MIT E40-149A
Cambridge, MA, 02139
rfreund@mit.edu
tel: (617) 253 8997

Federico Girosi
C.B.C.L.
MIT E25-201
Cambridge, MA, 02139
girosi@ai.mit.edu
tel: (617) 253 0548

Abstract

We investigate the problem of training a Support Vector Machine (SVM) [1, 2, 7] on a very large date base (e.g. 50,000 data points) in the case in which the number of support vectors is also very large (e.g. 40,000). Training a SVM is equivalent to solving a linearly constrained quadratic programming (QP) problem in a number of variables equal to the number of data points. This optimization problem is known to be challenging when the number of data points exceeds few thousands. In previous work, done by us as well as by other researchers, the strategy used to solve the large scale QP problem takes advantage of the fact that the expected number of support vectors is small ($< 3,000$). Therefore, the existing algorithms cannot deal with more than a few thousand support vectors. In this paper we present a decomposition algorithm that is guaranteed to solve the QP problem and that does not make assumptions on the expected number of support vectors. In order to present the feasibility of our approach we consider a foreign exchange rate time series data base with 110,000 data points that generates 100,000 support vectors.

1 Introduction

In this paper we consider the problem of training a Support Vector Machine (SVM), a pattern classification algorithm recently developed by V. Vapnik and his team at AT&T Bell Labs. [1, 2, 7]. SVM can be seen as a new way to train polynomial, neural network, or Radial Basis Functions classifiers, based on the idea of *structural risk minimization* rather than *empirical risk* minimization [1]. From the implementation point of view, training a SVM

[1]The name of SVM is due to the fact that one of the outcomes of the algorithm, in addition to the parameters for the classifier, is a set of data points (the "support vectors") which contain, in a sense, all the "relevant" information about the problem.

is equivalent to solving a linearly constrained Quadratic Programming (QP) problem in a number of variables equal to the number of data points. This problem is challenging when the size of the data set becomes larger than a few thousands, which is often the case in practical applications. A number of techniques for SVM training have been proposed [7, 4, 5, 6]. However, many of these strategies take advantage of the following assumptions, or expectations: **1)** The number of support vectors is small, with respect to the the number of data points; **2)** the total number of support vectors does not exceed a few thousands (e.g. < 3.000). Since the ratio between the number of support vectors and the total number of data points (averaged over the probability distribution of the input variables) is an upper bound on the generalization error, the previous assumptions are violated in the following cases: **1)** the problem is "difficult", so that the generalization error will be large and therefore the proportion of support vectors is high, or **2)** the data set is so large (say 300,000) that even if the problem can have small generalization error (say 1%) the number of support vectors will be large (in this case around 3,000).
The algorithm that we present in this paper does not make the above mentioned assumptions. It should be noticed, however, that in the case in which the assumptions above are satisfied the algorithm *does* take advantage of them. The algorithm is similar in spirit to the algorithm that we proposed in [4] (that was limited to deal with few thousands support vectors): it is a decomposition algorithm, in which the original QP problem is replaced by a sequence of smaller problems that is proved to converge to the global optimum. Although the experiments we report in this paper concern a classification problem, the current algorithm can also be used, with minimal modifications, to train the new version of the SVM, that can deal with regression as well as classification.
The plan of the paper is as follows: in the next section we briefly sketch the ideas underlying SVM. Then in section 3 we present our new algorithm, in section 4 we show some results of our implementation on a financial data set with 110,000 data-points with as many as 100,000 support vectors and in section 5 we summarize the paper.

2 Support Vector Machines

In this section we briefly sketch the SVM algorithm and its motivation. A more detailed description of SVM can be found in [7] (chapter 5) and [2].
We start from the simple case of two linearly separable classes. We assume that we have a data set $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^{\ell}$ of labeled examples, where $y_i \in \{-1, 1\}$, and we wish to select, among the infinite number of linear classifiers

that separate the data, one that minimizes the generalization error, or at least an upper bound on it (this is the idea underlying the structural risk minimization principle [7]). V. Vapnik showed [7] that the hyperplane with this property is the one that leaves the maximum margin between the two classes [1], where the margin is defined as the sum of the distances of the hyperplane from the closest point of the two classes.

If the two classes are non-separable the SVM looks for the hyperplane that maximizes the margin and that, at the same time, minimizes a quantity proportional to the number of misclassification errors. The trade off between margin and misclassification error is controlled by a positive constant C that has to be chosen beforehand. In this case it can be shown that the solution to this problem is a linear classifier $f(\mathbf{x}) = \text{sign}(\sum_{i=1}^{\ell} \lambda_i y_i \mathbf{x}^T \mathbf{x}_i + b)$ whose coefficients λ_i are the solution of a QP problem, defined over the hypercube $[0, C]^\ell$, whose precise statement will be given in section 3 (see eq. 1). Since the quadratic form is minimized in the hypercube $[0, C]^\ell$, the solution will have a number of coefficients λ_i exactly equal to zero. Since there is a coefficient λ_i associated to each data point, only the data points corresponding to non-zero λ_i (the "support vectors") will influence the solution. Intuitively, the support vectors are the data points that lie at the border between the two classes. It is then clear that a small number of support vectors indicates that the two classes can be well separated.

This technique can be extended to allow for non-linear decision surfaces. This is done by projecting the original set of variables $\mathbf{x}$ in a higher dimensional *feature space*: $\mathbf{x} \in R^d \Rightarrow \mathbf{z}(\mathbf{x}) \equiv (\phi_1(\mathbf{x}), \ldots, \phi_n(\mathbf{x})) \in R^n$ (where n is possibly infinite) and by formulating the linear classification problem in the feature space. Vapnik proves that there are certain choices of features ϕ_i for which the solution has the following form:

$$f(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^{\ell} \lambda_i y_i K(\mathbf{x}, \mathbf{x}_i) + b\right)$$

where $K(\mathbf{x}, \mathbf{y})$ is a symmetric positive definite kernel function that depends on the choice of the features and represent the scalar product in the feature space. In table (1) we list some choices of the kernel function proposed by Vapnik: notice how they lead to well known classifiers, whose decision surfaces are known to have good approximation properties.

3 Training a Support Vector Machine

In this section we present a decomposition algorithm that, without making assumptions on the expected number of support vectors, allows us to train a

Kernel Function	Type of Classifier
$K(\mathbf{x}, \mathbf{x}_i) = \exp(-\|\mathbf{x} - \mathbf{x}_i\|^2)$	Gaussian RBF
$K(\mathbf{x}, \mathbf{x}_i) = (\mathbf{x}^T \mathbf{x}_i + 1)^d$	Polynomial of degree d
$K(\mathbf{x}, \mathbf{x}_i) = \tanh(\mathbf{x}^T \mathbf{x}_i - \Theta)$	Multi Layer Perceptron

Table 1: Some possible kernel functions and the type of decision surface they define.

SVM on a large data set by solving a sequence of smaller QP problems. The two key issues to be considered are:

1. **Optimality Conditions**: These conditions allow us to decide computationally whether the problem has been solved optimally at a particular iteration of the original problem. Section 3.1 states and proves optimality conditions for the QP given by (1).

2. **Strategy for Improvement**: If a particular solution is not optimal, this strategy defines a way to improve the cost function and is frequently associated with variables that violate optimality conditions. This strategy will be stated in section 3.2.

Using the results of sections 3.1 and 3.2 we will then formulate our decomposition algorithm in section 3.3.

3.1 Optimality Conditions

The QP problem that we have to solve in order to train a SVM is the following [1, 2, 7]:

$$\begin{array}{llll} \underset{\Lambda}{\text{Minimize}} & W(\boldsymbol{\Lambda}) & = -\boldsymbol{\Lambda}^T \mathbf{1} + \frac{1}{2}\boldsymbol{\Lambda}^T D \boldsymbol{\Lambda} & \\ \text{subject to} & & & \\ & \boldsymbol{\Lambda}^T \mathbf{y} & = 0 & (\mu) \\ & \boldsymbol{\Lambda} - C\mathbf{1} & \leq \mathbf{0} & (\boldsymbol{\Upsilon}) \\ & -\boldsymbol{\Lambda} & \leq \mathbf{0} & (\boldsymbol{\Pi}) \end{array} \tag{1}$$

where $(\mathbf{1})_i = 1$, $D_{ij} = y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$, μ, $\boldsymbol{\Upsilon}^T = (\upsilon_1, \ldots, \upsilon_\ell)$ and $\boldsymbol{\Pi}^T = (\pi_1, \ldots, \pi_\ell)$ are the associated Kuhn-Tucker multipliers. The choice of the kernel K is left to the user, and it depends on the decision surfaces one expects to work best. Since D is a positive semi-definite matrix (the kernel function K is positive definite), and the constraints in (1) are linear, the Kuhn-Tucker, (KT) conditions are necessary and sufficient for optimality. The KT conditions are as follows:

$$\begin{array}{ll}
\nabla W(\mathbf{\Lambda}) + \mathbf{\Upsilon} - \mathbf{\Pi} + \mu \mathbf{y} & = \mathbf{0} \\
\mathbf{\Upsilon}^T(\mathbf{\Lambda} - C\mathbf{1}) & = 0 \\
\mathbf{\Pi}^T \mathbf{\Lambda} & = 0 \\
 & \\
\mathbf{\Upsilon} & \geq \mathbf{0} \\
\mathbf{\Pi} & \geq \mathbf{0} \\
 & \\
\mathbf{\Lambda}^T \mathbf{y} & = 0 \\
\mathbf{\Lambda} - C\mathbf{1} & \leq \mathbf{0} \\
-\mathbf{\Lambda} & \leq \mathbf{0}
\end{array} \tag{2}$$

In order to derive further algebraic expressions from the optimality conditions (2), we assume the existence of some λ_i such that $0 < \lambda_i < C$, and consider the three possible values that each component of $\mathbf{\Lambda}$ can have:

1. **Case:** $0 < \lambda_i < C$
   From the first three equations of the KT conditions we have:

$$(D\mathbf{\Lambda})_i - 1 + \mu y_i = 0 \tag{3}$$

   Using the results in [2] and [7] one can show that this implies that $\mu = b$.

2. **Case:** $\lambda_i = C$
   From the first three equations of the KT conditions we have:

$$(D\mathbf{\Lambda})_i - 1 + \upsilon_i + \mu y_i = 0 \tag{4}$$

   It is useful to define the following quantity:

$$g(\mathbf{x}_i) = \sum_{j=1}^{\ell} \lambda_j y_j K(\mathbf{x}_i, \mathbf{x}_j) + b \tag{5}$$

   Using the fact that $\mu = b$ and requiring the KT multiplier υ_i to be positive one can show that the following conditions should hold:

$$y_i g(\mathbf{x}_i) \ \leq \ 1 \tag{6}$$

3. **Case:** $\lambda_i = 0$
   From the first three equations of the KT conditions we have:

$$(D\mathbf{\Lambda})_i - 1 - \pi_i + \mu y_i = 0 \tag{7}$$

By applying a similar algebraic manipulation as the one described for case 2, we obtain

$$y_i g(\mathbf{x}_i) \;\geq\; 1 \tag{8}$$

3.2 Strategy for Improvement

The optimality conditions derived in the previous section are essential in order to devise a decomposition strategy that guarantees that at every iteration the objective function is improved. In order to accomplish this goal we partition the index set in two sets B and N, where the set B is called the *working set*. Then we decompose $\mathbf{\Lambda}$ in two vectors $\mathbf{\Lambda}_B$ and $\mathbf{\Lambda}_N$, keeping fixed $\mathbf{\Lambda}_N$ and allowing changes only in $\mathbf{\Lambda}_B$, thus defining the following subproblem:

$$\begin{array}{lll}
\underset{\Lambda_B}{\text{Minimize}} & W(\mathbf{\Lambda}_B) & = -\mathbf{\Lambda}_B^T\mathbf{1} + \frac{1}{2}\left[\mathbf{\Lambda}_B^T D_{BB}\mathbf{\Lambda}_B + \mathbf{\Lambda}_B^T D_{BN}\mathbf{\Lambda}_N + \right. \\
 & & \left. +\mathbf{\Lambda}_N^T D_{NB}\mathbf{\Lambda}_B + \mathbf{\Lambda}_N^T D_{NN}\mathbf{\Lambda}_N\right] - \mathbf{\Lambda}_N^T\mathbf{1} \\
\text{subject to} & & \\
 & \mathbf{\Lambda}_B^T\mathbf{y}_B + \mathbf{\Lambda}_N^T\mathbf{y}_N & = 0 \\
 & \mathbf{\Lambda}_B - C\mathbf{1} & \leq \mathbf{0} \\
 & -\mathbf{\Lambda}_B & \leq \mathbf{0}
\end{array} \tag{9}$$

where $(\mathbf{1})_i = 1$, $D_{\alpha\beta}$ is such that $D_{ij} = y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$, with $i \in \alpha, j \in \beta$, and C is a positive constant. Using this decomposition we notice that:

- The terms $-\mathbf{\Lambda}_N^T\mathbf{1} + \frac{1}{2}\mathbf{\Lambda}_N^T D_{NN}\mathbf{\Lambda}_N$ are constant within the defined subproblem.
- Since $K(\mathbf{x}, \mathbf{y})$ is a symmetric kernel, the computation of $\mathbf{\Lambda}_B^T D_{BN}\mathbf{\Lambda}_N + \mathbf{\Lambda}_N^T D_{NB}\mathbf{\Lambda}_B$ can be replaced by $2\mathbf{\Lambda}_B^T\mathbf{q}_{BN}$, where:

$$(\mathbf{q}_{BN})_i \;=\; y_i \sum_{j \in N} \lambda_j y_j K(\mathbf{x}_i, \mathbf{x}_j) \qquad i \in B \tag{10}$$

  This is a very important simplification, since it allows us to keep the size of the subproblem independent of the number of fixed variables $\mathbf{\Lambda}_N$, which translates into keeping it also independent of the number of support vectors.

- We can replace *any* λ_i, $i \in B$, with *any* λ_j, $j \in N$ (i.e. there is no restriction on their value), without changing the cost function or the feasibility of both the subproblem and the original problem.

- If the subproblem is optimal before such a replacement, the new subproblem is optimal if and only if λ_j satisfies the Optimality Conditions for the appropriate case (3 cases described above).

The previous statements lead to the following more formal propositions:

Proposition 3.1 *(“**Build down**”): moving a variable from B to N leaves the cost function unchanged, and the solution is feasible in the subproblem.*

Proof: Let $B' = B \setminus \{k\}$ and $N' = N \cup \{k\}$. Then:

$$
\begin{aligned}
W(\mathbf{\Lambda}_B, \mathbf{\Lambda}_N) &= -\sum_{i\in B}\lambda_i - \sum_{i\in N}\lambda_i + \frac{1}{2}\left[\sum_{i,j\in B}\lambda_i\lambda_j D_{ij} + 2\sum_{i\in B}\lambda_i\sum_{j\in N}\lambda_j D_{ij} + \right. \\
&\quad \left. + \sum_{i,j\in N}\lambda_i\lambda_j D_{ij}\right] \\
&= -\sum_{i\in B'}\lambda_i - \lambda_k - \sum_{i\in N}\lambda_i + \frac{1}{2}\left[\sum_{i,j\in B'}\lambda_i\lambda_j D_{ij} + 2\lambda_k\sum_{i\in B'}\lambda_i D_{ik} + \right. \\
&\quad \left. + 2\lambda_k\sum_{j\in N}\lambda_j D_{jk} + 2\sum_{i\in B'}\lambda_i\sum_{j\in N}\lambda_j D_{ij} + \lambda_k^2 D_{kk} + \sum_{i,j\in N}\lambda_i\lambda_j D_{ij}\right] \\
&= -\sum_{i\in B'}\lambda_i - \sum_{i\in N'}\lambda_i + \frac{1}{2}\left[\sum_{i,j\in B'}\lambda_i\lambda_j D_{ij} + 2\sum_{i\in B'}\lambda_i\sum_{j\in N'}\lambda_j D_{ij} + \right. \\
&\quad \left. + \sum_{i,j\in N'}\lambda_i\lambda_j D_{ij}\right] \\
&= W(\mathbf{\Lambda}_{B'}, \mathbf{\Lambda}_{N'})
\end{aligned}
$$

The solution $(\mathbf{\Lambda}_{B'}, \mathbf{\Lambda}_{N'})$ is feasible in the subproblem since:

$$
\begin{aligned}
0 &= \mathbf{\Lambda}_B^T\mathbf{y}_B + \mathbf{\Lambda}_N^T\mathbf{y}_N \\
&= \mathbf{\Lambda}_{B'}^T\mathbf{y}_{B'} + \lambda_k y_k + \mathbf{\Lambda}_N^T\mathbf{y}_N \\
&= \mathbf{\Lambda}_{B'}^T\mathbf{y}_{B'} + \mathbf{\Lambda}_{N'}^T\mathbf{y}_{N'}
\end{aligned}
$$

and the bound constraints are always unaffected.

Proposition 3.2 *(“**Build up**”): moving a variable that violates the optimality conditions from N to B gives a strict improvement in the cost function when the subproblem is re-optimized.*

Proof: This is a direct consequence of Proposition 3.1 and the fact that Kuhn-Tucker conditions are necessary and sufficient for optimality.

3.3 The Decomposition Algorithm

Using the results of the previous sections we are now ready to formulate our decomposition algorithm:

1. Arbitrarily choose $|B|$ points from the data set.
2. Solve the subproblem defined by the variables in B.
3. While there exists some $j \in N$, such that:
   - $\lambda_j = 0$ and $g(\mathbf{x}_j)y_j < 1$
   - $\lambda_j = C$ and $g(\mathbf{x}_j)y_j > 1$
   - $0 < \lambda_j < C$ and $g(\mathbf{x}_j)y_j \neq 1$,

   replace any λ_i, $i \in B$, with λ_j and solve the new subproblem given by:

$$\begin{array}{lll} \underset{\Lambda_B}{\text{Minimize}} & W(\boldsymbol{\Lambda}_B) & = -\boldsymbol{\Lambda}_B^T \mathbf{1} + \frac{1}{2}\boldsymbol{\Lambda}_B^T D_{BB} \boldsymbol{\Lambda}_B + \boldsymbol{\Lambda}_B^T \mathbf{q}_{BN} \\ \text{subject to} & & \\ & \boldsymbol{\Lambda}_B^T \mathbf{y}_B + \boldsymbol{\Lambda}_N^T \mathbf{y}_N & = 0 \\ & \boldsymbol{\Lambda}_B - C\mathbf{1} & \leq \mathbf{0} \\ & -\boldsymbol{\Lambda}_B & \leq \mathbf{0} \end{array} \tag{11}$$

   where:

$$(\mathbf{q}_{BN})_i = y_i \sum_{j \in N} \lambda_j y_j K(\mathbf{x}_i, \mathbf{x}_j) \qquad i \in B \tag{12}$$

Notice that we have omitted the constant term $-\boldsymbol{\Lambda}_N^T \mathbf{1} + \frac{1}{2}\boldsymbol{\Lambda}_N^T D_{NN} \boldsymbol{\Lambda}_N$ in the cost function, and that according to Proposition 3.2, this algorithm will strictly improve the objective function at each iteration and therefore will not cycle. Since the objective function is bounded ($W(\boldsymbol{\Lambda})$ is convex quadratic and the feasible region is bounded), the algorithm must converge to the global optimal solution in a finite number of iterations.

4 Implementation and Results

We have implemented the decomposition algorithm using MINOS 5.4 [3] as the solver of the sub-problems. We tested our technique on a problem known for being "difficult": a foreign exchange rate time series that was used in the 1992 Santa Fe Institute Time Series Competition, in which we looked at the sign of the change of the time series, rather than its value. We considered data sets of increasing sizes, up to 110,000 points, obtaining up to 100,000 support vectors. Figure 4 shows the relationship between training times, number of data points and number of support vectors in our experiments. The training time on a SUN Sparc 20 with 128 Mb of RAM ranged from 3 hours for 10,000 support vectors to 48 hours for 40,000 support vectors. The results that we obtain are comparable to the results reported in [8] using a Neural Networks approach, where generalization errors around 53% were reported. The purpose of this experiment was not to benchmark SVM's on this specific problem, but to show that its use in a problem with as many as 100,000 support vectors is computationally tractable.

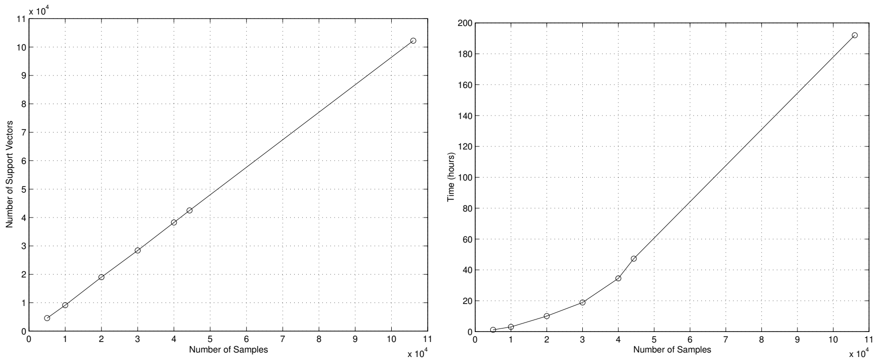

Figure 1: (a)Number of support vectors Vs. number of data points. (b) Training time Vs. number of data points.

5 Summary and Conclusions

In this paper we have presented a novel decomposition algorithm that can be used to train Support Vector Machines on large data sets that contain a large number of support vectors. The current version of the algorithm has been tested with a data set of 110,000 data points and 100,000 support vectors on a machine with 40 Mb of RAM. No attempts to optimize and speed up the algorithm have been made yet. We believe that this algorithm starts to meet the increasing need to deal with data sets where both the number of data points and the number of support vectors are of the order of 10^5. Problems with these characteristics are likely to be found in the area of financial markets, where lots of data maybe available but little generalization error is expected.

Acknowledgements

The authors would like to thank Sayan Mukherjee for his help in collecting and preprocessing the data.

References

[1] B.E. Boser, I.M. Guyon, and V.N. Vapnik. A training algorithm for optimal margin classifier. In *Proc. 5th ACM Workshop on Computational Learning Theory*, pages 144–152, Pittsburgh, PA, July 1992.

[2] C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20:1–25, 1995.

[3] B. Murtagh and M. Saunders. Large-scale linearly constrained optimization. *Mathematical Programming*, 14:41–72, 1978.

[4] E. Osuna, R. Freund, and F. Girosi. Support vector machines: Training and applications. A.I. Memo 1602, MIT A. I. Lab., 1997.

[5] M. Schmidt. Identifying speakers with support vectors networks. In *Proceedings of Interface '96*, Sydney, July 1996.

[6] B. Schölkopf, C. Burges, and V. Vapnik. Extracting support data for a given task. In U.M. Fayyad and R. Uthurusamy, editors, *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*, Menlo Park, CA, 1995. AAAI Press.

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

[8] X. Zhang. Non-linear predictive models for intra-day foreign exchange trading. Technical report, PHZ Partners, August 1993.