Reducing the run-time complexity of Support Vector Machines

(To appear in ICPR'98, Brisbane, Australia, August 16-20 1998)

Edgar Osuna and **Federico Girosi**
Center for Biological and Computational Learning
Massachusetts Institute of Technology
Cambridge, MA 02139, USA
e-mail: {eosuna,girosi}@ai.mit.edu

Abstract

The Support Vector Machine (SVM) is a new and promising technique for classification and regression, developed by V. Vapnik and his group at AT&T Bell Labs [2, 9]. The technique can be seen as a new training algorithm for Polynomial, Radial Basis Function and Multi-Layer Perceptron networks. SVMs are currently considered slower at run-time than other techniques with similar generalization performance. In this paper we focus on SVM for classification and investigate the problem of reducing its run-time complexity. We present two relevant results: a) the use of SVM itself as a regression tool to approximate the decision surface with a user-specified accuracy; and b) a reformulation of the training problem that yields the exact same decision surface using a smaller number of basis functions. We believe that this reformulation offers great potential for future improvements of the technique. For most of the selected problems, both approaches give reductions of run-time in the 50-95% range, with no system degradation.

Keywords: Support Vector Machines, Machine Learning, Optimization.

1 Introduction

Support Vector Machines (SVMs) can be seen as an approximate implementation of what Vapnik has defined as Structural Risk Minimization (SRM), an inductive principle that aims at minimizing an upper bound on the generalization error of a model, rather than minimizing the Mean Square Error over the data set. Details on this technique can be found in [9, 2, 3]. A simple intuition about the technique can be obtained in the case of a linear classifier, with linearly separable classes. In this case a SVM will separate the data with an hyperplane (described by $\mathbf{w} \cdot \mathbf{x} + b = 0$) that leaves the maximum margin between the two classes (see fig. 1). It can be shown [9] that maximizing the margin is equivalent to minimizing an upper bound on the generalization error of the classifier, providing a very strong theoretical motivation for this technique. The vector $\mathbf{w}$ that maximizes the margin can be shown to have the form:

$$\mathbf{w} = \sum_{i=1}^{\ell} \lambda_i y_i \mathbf{x}_i$$

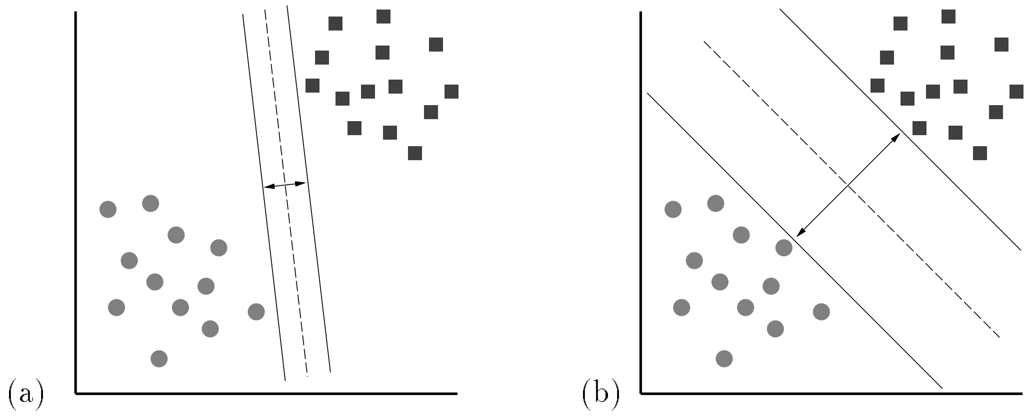

Figure 1: The margin of a hyperplane is defined as the sum of the distances of the hyperplane from the closest points in the two classes. (a) A separating hyperplane with small margin. (b) A Separating Hyperplane with large margin. A better generalization capability is expected from (b).

where the parameters λ_i are found by solving a Quadratic Programming (QP) problem.
The extension of this technique to non-linearly separable classes is simple, but it will not be discussed here. The extension to the case of non-linear decision surfaces is achieved by mapping the original input variables $\mathbf{x}$ in a high (possibly infinite) dimensional space of *features*: $\mathbf{x} \rightarrow \mathbf{z}(\mathbf{x}) = (\phi_1(\mathbf{x}), \ldots, \phi_n(\mathbf{x}), \ldots)$ and then computing a linear separating surface in this new space. It turns out that this approach leads to decision surfaces of the form

$$f(\mathbf{x}) = \theta\left(\sum_{i=1}^{\ell} \lambda_i y_i K(\mathbf{x}; \mathbf{x}_i) + b\right) \tag{1}$$

where θ is the Heaviside function, $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^{l}$ is the data set, with $y_i \in \{-1, 1\}$, λ_i is a set of parameters computed by solving a QP problem, and $K(\mathbf{x}, \mathbf{y})$ is the *kernel* function, which represents the scalar product operation in the feature space: $K(\mathbf{x}, \mathbf{y}) \equiv \mathbf{z}(\mathbf{x}) \cdot \mathbf{z}(\mathbf{y})$. Notice that, according to the definition of K, eq. (1) has the form $f(\mathbf{x}) = \theta(\mathbf{w} \cdot \mathbf{z}(\mathbf{x}) + b)$, and therefore represents a linear separating surface in the feature space. The parameters λ_i are found by solving the following QP problem [9, 2, 3]:

$$\begin{array}{rll} \text{Maximize} \quad W(\mathbf{\Lambda}) & = \mathbf{\Lambda}^T \mathbf{1} - \frac{1}{2}\mathbf{\Lambda}^T D \mathbf{\Lambda} & \\ \text{subject to} & & \\ \mathbf{\Lambda}^T \mathbf{y} & = 0 & \\ 0 \leq \ \lambda_i & \leq C & i = 1 \ldots \ell \end{array} \tag{2}$$

where $(\mathbf{\Lambda})_i = \lambda_i$, $(\mathbf{1})_i = 1$, $D_{ij} = y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$, and C is a positive constant that sets the cost that we are willing to pay for a misclassified data point. The kernel function $K(\mathbf{x}_i, \mathbf{x}_j)$ defines

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 (only for some fixed Θ)

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

the type of decision surface that the machine will build. In table (1) we list some choices of the kernel allowed by the theory of SVM: notice how they lead to well known classifiers, whose decision surfaces are known to have good approximation properties.
Given the nature of the QP problem, it turns out that only a subset of the λ_i will be different from zero, and the corresponding data points $\mathbf{x}_i$ are called *support vectors*, since they the only data points that contribute to determine the optimal separating hyperplane. Intuitively, the support vectors are the data points that lie at the border between the two classes in the feature space. V. Vapnik showed that the generalization error has an upper bound given by the ratio between the number of support vectors and the total number of data points. Therefore, in "easy" problems, in which the classes do not overlap too much in feature space, the number of support vectors will be small respect to the number of data points. Even in this case, however, if the data set is large (say 10^5) the number of support could be easily of the order of thousand, and require the computation of eq. (1) every time a pattern has to be classified. The reduction of the run-time complexity of SVMs can therefore be defined as a series of heuristics or techniques that reduce the computational effort spent in the evaluation of eq. (1).
The outline of the paper is as follows: in section 2 we specify the problem we want to solve and comment on possible approaches to its solution; in section 3 and 4 we offer two possible solutions; in section 5 we present experimental results; and in section 6 we comment on the limitations of our two suggested approaches and present some concluding remarks.

2 Motivation and Statement of the Problem

The number of operations required to compute the solution $f(\mathbf{x})$ (see eq. 1) can easily become the bottleneck of any system that needs to performs a massive number of classifications. Examples of this issue arise in face and people detection systems [6, 5], where SVMs are used to distinguish an object from the background, or in checks and ZIP code readers, where the system is required to spend fractions of a second per check or envelope. Here are some different ways to obtain a speedup:

1. Approximating the solution $f(\mathbf{x})$ by:

$$\hat{f}(\mathbf{x}) = \sum_{i=1}^{\ell'} \gamma_i K(\mathbf{z}_i, \mathbf{x}) + b$$

where $\mathbf{z}_i$ are *synthetic* vectors, which are not necessarily data points anymore, γ_i are weights, and $\ell' \ll \ell$. This approach, which for radial kernels K resembles the technique of "moving centers" [4, 7], has been pursued by C. Burges [1], but the procedure is hard to implement and lacks a principled way for controlling the approximation accuracy.

2. Approximating the solution $f(\mathbf{x})$ by:

$$\hat{f}(\mathbf{x}) = \sum_{i=1}^{\ell'} \gamma_i K(\mathbf{x}_i, \mathbf{x}) + b$$

where $\mathbf{x}_i$ is still a support vector with weight γ_i, but $\ell' \ll \ell$. This will be our first approach and it will be described in detail in section 3.

3. If possible, rewriting the solution $f(\mathbf{x})$ as:

$$f(\mathbf{x}) = \sum_{i=1}^{\ell'} \gamma_i y_i K(\mathbf{x}_i, \mathbf{x}) + b$$

where $\mathbf{x}_i$ are still data points (but not necessarily support vectors according to the traditional definition) with weight γ_i, but $\ell' \ll \ell$. Our hope in this approach is to find an alternative more *efficient* representation of the separating hyperplane in problems where its expansion is not unique. This will be our second approach and it will be described in detail in section 4.

2.1 The class of problems we approach

Before characterizing explicitly the kind of problems we are trying to solve, we want to present the following example: Training the Ripley data set [1] (250 data points, 2 dimensions, not linearly separable) with a linear SVM yields roughly 90 support vectors (actually 89 for C=100). This means that the separating hyperplane $\mathbf{w} = \sum_{i=1}^{\ell} \lambda_i y_i \mathbf{x}_i$ is defined using 90 non-zero coefficients. Figure 2 presents the data set, the support vectors and the separating hyperplane. This representation for $\mathbf{w}$ is strikingly inefficient, since we are using a linear combination of 90 vectors to define a vector $\mathbf{w}$ that lives in $\Re^2$. Although the example given is a simple linear classifier in $\Re^2$, it is representative of the problems which are encountered in more complex examples involving nonlinear decision surfaces and input spaces with higher dimensionality.

Why is this happening?

[1] Available at ftp://markov.stats.ox.ac.uk/pub/neural/papers/

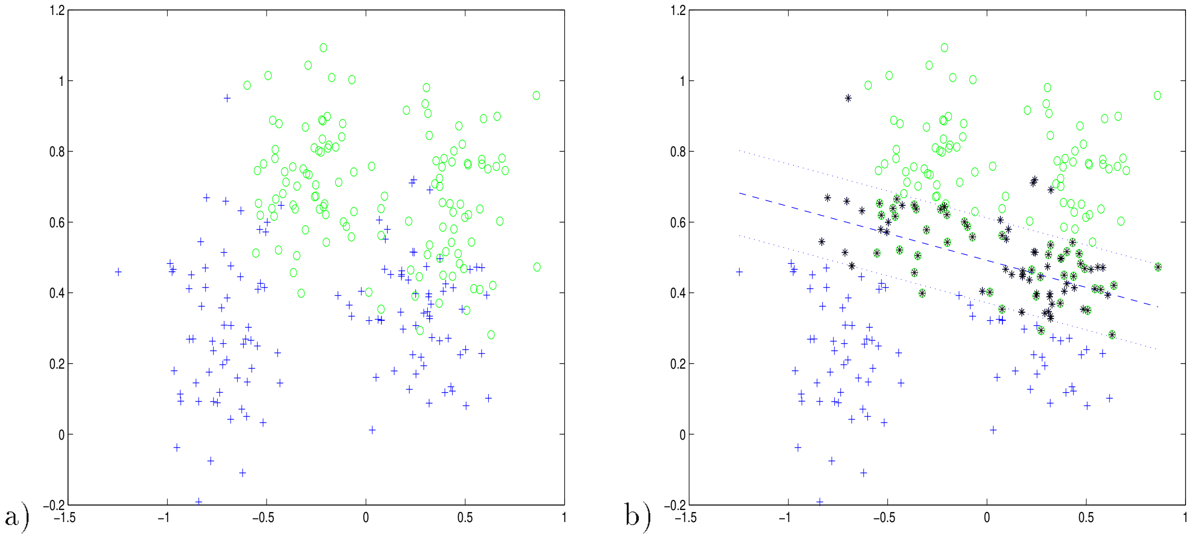

Figure 2: (a) The Ripley data set. (b) The optimal separating hyperplane and its support vectors. The dotted lines above and below the hyperplane depict the ± 1 range around the separating surface.

1. The Karush-Kuhn-Tucker optimality conditions of the training problem mandate for any misclassified training point to have $\lambda_i = C$ (also points falling within the margin, but correctly classified will have $\lambda_i = C$).

2. Any non-zero λ_i contributes in the expansion of $\mathbf{w}$, and cannot be removed from the training set without affecting the solution.

3. Since the data set is non-separable, all the training errors are present in the expansion.

Moreover, the following approaches:

- Removing known errors from the data-set before training,

- removing errors from the expansion after training, or

- training, removing the errors from the obtained set of support vectors, and retraining,

will not, in general, give the same separating hyperplane, since some of these errors are actually needed in the expansion of $\mathbf{w}$ and also need to be taken into account when penalizing non-separability. Therefore, we want to approach problems in which the number of support vectors with $\lambda_i = C$ is *excessively* high. This is a problem that appears in particular when the data set is non-separable, and becomes more relevant with the increase of noise, degree of non-separability and size of the training set.

3 First approach: Using SVRM

The solution of SVM consists of a linear combination of kernels, one for each support vector. One way to simplify the solution is to approximate it with a linear combination of a *subset* of the kernels. This can be accomplished by using Support Vector Regression Machines (SVRM). SVRM is the natural extension of SVM to the problem of function approximation, and it can be seen as the problem of finding the minimum of a functional of the following form:

$$H[f] = \sum_{i=1}^{l} |y_i - f(\mathbf{x}_i)|_\epsilon + \frac{1}{2C}\phi[f]$$

where ϕ is a smoothness functional and $|x|_\epsilon$ is Vapnik's ϵ-insensitive cost function defined below:

$$| x |_\epsilon = \begin{cases} 0 & \text{if } \; | x | < \epsilon \\ | x | - \epsilon & \text{otherwise.} \end{cases} \tag{3}$$

The effect of the cost function $|x|_\epsilon$ is to make the technique robust to outliers and insensitive to error below a certain threshold (ϵ). The minimum of this functional has always the form $f(\mathbf{x}) = \sum_{i=1}^{l} \alpha_i K(\mathbf{x}, \mathbf{x}_i)$, where K is a kernel that depends on the particular choice of the smoothness functional ϕ, and the coefficients α_i are found by solving a QP problem very similar to the one of SVM for classification. The solution is typically *sparse*, that is, only some coefficients α_i will be different from zero, and their number is critically controlled by the parameter ϵ, which also controls the accuracy of the approximation. The relationship between SVRM and sparse approximation has been studied in [3]. Our approach consists in using this approximation scheme to approximate the function $f(\mathbf{x}) = \sum_{i=1}^{l} \lambda_i y_i K(\mathbf{x}, \mathbf{x}_i)$ which is obtained solving SVM for classification, and to obtain a sparse version of it, which uses a much smaller number of support vectors. While many different choices of kernels K are available in SVRM, we choose the same kernel that is used in the classification stage, for the following reasons:

1. ϵ-accuracy can be chosen arbitrarily small and a perfect approximation can always be achieved, provided that the parameter C is big enough. This is an important property if we consider the approximation quality more important than speed.

2. Properties of the kernel functions let us estimate the distance $\|\mathbf{w} - \hat{\mathbf{w}}\|_2^2$ between the "true" separating hyperplane in feature space ($\mathbf{w}$) and its sparse approximation $\hat{\mathbf{w}}$.

3. Since the kernel used initially during training has already built a linear hyperplane in feature space, using the same kernel takes advantage of that same linearization.

The Algorithm:
The formal algorithm can be stated as:

1. Train the SV classifier using the kernel and parameters of choice. This step defines $f(\mathbf{x}) = \sum_{i=1}^{\ell} \lambda_i y_i K(\mathbf{x}_i, \mathbf{x}) + b$.
2. Run SVMR on the data points $(\mathbf{x}_i, f(\mathbf{x}_i))$, where $\mathbf{x}_i$ is a support vector obtained in step 1. This will provide a function of the same form, but with a smaller number of support vectors.Use in this step a high value for the parameter C, the same kernel used in (1), and the desired ϵ-accuracy.

The results obtained with this approach can be seen in section 5.

4 Second approach: Reformulating the Problem

Traditionally, SVM are derived by first deriving the constrained optimization problem of finding the optimal separating hyperplane that has maximum margin (the *primal* problem), and then studying its *dual* problem, which has the form (2). One common reason for studying the problem in the dual setting is that in this way one avoids the definition and computation of the parameters of a hyperplane in a possibly infinite dimensional feature space, because the solution only depends on the scalar products in the feature space, which can be easily computed by the kernel K. However, it is also possible to solve the primal formulation of the problem using only the kernel K. In fact, one can show that the primal problem, which is the dual of the problem currently in use (eq. 2) can also be written as:

$$
\begin{array}{lllll}
\text{Minimize} & F(\mathbf{\Lambda}, b, \mathbf{\Xi}) & = \frac{1}{2}\mathbf{\Lambda}^T Q \mathbf{\Lambda} + C\sum_{i=1}^{n} \xi_i & & \\
\text{subject to} & & & & \\
 & y_i(\sum_{j=1}^{n} \lambda_j y_j K(\mathbf{x}_i, \mathbf{x}_j) + b) & \geq 1 - \xi_i & i = 1 \ldots n & (4) \\
 & \xi_i, \lambda_i & \geq 0 & i = 1 \ldots n & \\
 & b & free & &
\end{array}
$$

This problem formulation, which from now on we refer to as the *primal reformulation*, has the same initial structure as the original primal formulation, but incorporates the kernel K and has a natural and clear interpretation:

- $\mathbf{\Lambda}^T Q \mathbf{\Lambda}$ can be shown to be proportional to the inverse of the margin in feature space (and therefore we want to minimize it);
- The constraint $y_i(\sum_{j=1}^{n} \lambda_j y_j K(\mathbf{x_i}, \mathbf{x_j}) + b) \geq 1 - \xi_i$ models how well the data point $\mathbf{x}_i$ is classified.
- ξ_i captures for data point i, its *degree of separability*, and *pays* a linear penalty C in the cost function.

One can show that the minimum of this problem gives the same separating surface of the classical approach, but uses a (possibly) different expression for $\mathbf{w}$ (work along these lines has also been done in [8]). It is not guaranteed that the new representation of the hyperplane is more sparse than the classical solution. However, experimentally, this approach works very well for two reasons: first, the coefficients are not *forced* to the upper bound for misclassifications, since they are not Lagrange multipliers anymore; and second, starting with $\mathbf{\Lambda} = \mathbf{0}$ as initial solution helps in keeping at 0 level unnecessary points in the expansion.

4.1 Possible Improvements

A clear advantage of this primal reformulation when compared to the training problem typically solved , is that we can include in the cost function certain attractor terms in order to benefit certain types of expansion. For example:

- We can include a small penalty of the form $\sum_{i=1}^{l} |\lambda_i|$, which has been used before in other techniques to enforce sparse representations.
- We can include a small penalty for using points that do not meet the separability constraints exactly. This will bring the set of coefficients to be a subset of the support vectors obtained through the current training problem. Since the coefficients are not Lagrange multipliers anymore, the values can be drastically different.
- We can include a small penalty for using errors, etc.

We want to remark that these penalties should be *small* so that the essence of the cost function is altered in the minimal way, that is, these small penalties are just coding a *preference* among all the possible linear combinations. A scheme in which one gradually reduces these penalties to zero is also possible.
The results obtained with this approach can be seen in section 5.

5 Experimental Results

Three datasets were selected for our experiments, and two different runs were performed with each one of them. Each run corresponds to a different kernel and/or parameter setting. The problems selected, the information regarding the kernel parameters, number of support vectors and training performance is given in table 2. The training performance is presented to give the reader an idea of the separability of the data.
The results obtained with the SV regression approach can be found on table 3. Notice that reduction percentages are above 50% in all runs except electrons (2). Notice also the relationship between the ϵ-approximation quality and the number of vectors obtained.

Database	# Data points	Kernel	C	Tr. Perf.	# SVs	$\|\|w\|\|$
skin (1)	1600	pol 2	100	91.94	587	12.669,36
skin (2)	1600	pol 5	100	95.81	227	58.005,72
electrons (1)	2000	pol 2	100	89.20	611	2.082,35
electrons (2)	2000	rbf σ =2	100	91.60	554	10.210,16
ripley (1)	250	linear	100	86.40	89	71,80
ripley (2)	250	rbf σ =1	100	89.60	77	643,11

Table 2: Problems selected for our experiments.

Database	Regression						Primal Ref.	
	$\epsilon = 10^{-2}$			$\epsilon = 10^{-6}$				
	$\frac{\|\|w-\hat{w}\|\|}{\|\|w\|\|}$	# vec.	Red. %	$\frac{\|\|w-\hat{w}\|\|}{\|\|w\|\|}$	# vec.	Red. %	# vec.	Red. %
skin (1)	1.4×10^{-3}	11	98.13	1.6×10^{-10}	25	95.74	6	98.98
skin (2)	1.2×10^{-3}	10	95.59	6.1×10^{-7}	75	66.96	57	74.89
electrons (1)	6.1×10^{-3}	40	93.45	2.9×10^{-9}	47	92.30	44	92.80
electrons (2)	6.6×10^{-3}	297	46.38	5.1×10^{-7}	554	0.0	521	5.96
ripley (1)	2.6×10^{-5}	3	96.63	2.5×10^{-12}	3	96.63	2	97.75
ripley (2)	4.6×10^{-3}	14	76.82	2.4×10^{-7}	33	57.14	34	55.84

Table 3: Results obtained using the SV regression and primal reformulation approaches. The column "# vec." reflects the number of vectors that are now present in the expansion. This number corresponds with a percentage reduction indicated in the column "Red. %".

The results obtained with the primal reformulation are also shown in table 3. As it was the case with the SV regression approach, reduction percentages are above 50% in all runs except electrons (2).

6 Limitations and Final Remarks

Although we think that our solutions have been successful for the kind of problem we decided to approach, there are 2 clear limitations that must be discussed:

1. Neither solution will reduce the run-time complexity of the classifier when all or most of the coefficients are strictly between the bounds (i.e. $0 < \lambda_i < C$). This situation tends to occur when the data is highly (if not totally) separable, and worsens as the dimensionality of the feature space grows. Up to date, the best we can do in cases like this is to use the reduced set method of Burges [1].

2. Although a delayed column generation algorithm can be devised for decomposing and solving our primal reformulation, memory limitations make it prohibitive for large data sets (beyond 10.000). Alternatives to this problem are still an open area of research.

We feel that both approaches have attacked and solved the problem of excessive number of support vectors with active upper bound (i.e. $\lambda = C$). The SV regression approach is useful since it can be applied as an *after-training* filter, it is not limited by memory requirements, and offers accuracy control of the approximation through the use of the parameter ϵ. The primal reformulation offers a *one-step* training approach that is not an approximation, but in fact gives exactly the same hyperplane obtained by training the current QP problem. Together with the possible improvements suggested in section 4.1, this technique can be made to enforce certain properties of the points used in the linear combination, and in fact, can be made to give a subset of the original support vectors without any loss of accuracy or approximation error. This primal reformulation has a very natural structure and therefore is suitable for improvements in the technique.

As a last comment, we must remark that throughout the paper we have only considered reducing the run-time complexity of the SV classifier. The proposed methods can easily be adapted to reducing the complexity of SV regression machines as well.

References

[1] C. Burges. Simplified support vector decision rules. *Procedings of the 13th International Conference on Machine Learning*, pages 71–77, 1996.

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

[3] F. Girosi. An equivalence between sparse approximation and Support Vector Machines. *Neural Computation*, 1998. (in press).

[4] J. Moody and C. Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1(2):281–294, 1989.

[5] M. Oren, C. Papageorgiou, P. Sinha, E. Osuna, and T. Poggio. Pedestrian detection using wavelet templates. In *Proc. Computer Vision and Pattern Recognition*, pages 193–199, Puerto Rico, June 16–20 1997.

[6] E. Osuna, R. Freund, and F. Girosi. Training support vector machines: an application to face detection. In *Proc. Computer Vision and Pattern Recognition*, Puerto Rico, June 16–20 1997.

[7] T. Poggio and F. Girosi. Regularization algorithms for learning that are equivalent to multilayer networks. *Science*, 247:978–982, 1990.

[8] M. Pontil and A. Verri. Properties of support vector machines. *Neural Computation*, 1998. (in press).

[9] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, 1995.