DIRECTSVM: A FAST AND SIMPLE SUPPORT VECTOR MACHINE PERCEPTRON

Danny Roobaert
Computer Vision and Active Perception laboratory
Royal Institute of Technology (KTH), S-100 44 Stockholm, SWEDEN
Tel.: +46 8 790 6646, Fax: +46 8 723 0302
E-mail: roobaert@nada.kth.se

Abstract.
We propose a simple implementation of the Support Vector Machine (SVM) for pattern recognition, that is not based on solving a complex quadratic optimization problem. Instead we propose a simple, iterative algorithm that is based on a few simple heuristics. The proposed algorithm finds high-quality solutions in a fast and intuitively-simple way. In experiments on the COIL database, on the extended COIL database and on the Sonar database of the UCI Irvine repository, DirectSVM is able to find solutions that are similar to these found by the original SVM. However DirectSVM is able to find these solutions substantially faster, while requiring less computational resources than the original SVM.

INTRODUCTION

Support Vector Machines (SVMs) belong to the best-performing learning algorithms available. They have produced remarkable performance in a number of difficult learning tasks without requiring prior knowledge. We mention amongst others the following examples in pattern recognition: handwritten digit recognition [6] and 3D object recognition [1, 10]. The SVM is rooted in Statistical Learning Theory [13] and comes with guarantees on its generalization behavior. The drawback of the SVM however is, that its implementation is complex since a constrained quadratic programming problem needs to be solved. In addition, quadratic programming is rather slow and computationally expensive in order to converge to a solution. Recently, a number of speedup implementations of the quadratic programming algorithm for SVM have been proposed like chunking [8], Sequential Minimal Optimization [9] and SVMlight [5]. Although these approaches formulate real accelerations for the SVM, they are still based on quadratic optimization which hides the way how the algorithm learns incrementally and do not favor geometrical insight in the learning process itself.

In contrast, the learning process of neural networks, and perceptrons in particular, are simpler to understand. They consist of simple elementary processing elements, the abstract neurons, and have relatively simple learning rules that incrementally update the weights of these neurons. Albeit, a disadvantage of neural networks is that the learning behavior, and in particular their *learning capacity*, is often not determined automatically. Concretely, often the number of neurons have to be determined in the fore-hand or alternatively a regularization parameter has to be set by hand. Hence the generalization capacity of neural networks needs to be set by the designer of the neural network empirically.

In this contribution, we want to combine the best of both approaches and unite in one algorithm the theoretical foundation of the SVM and the simple formulation of neural networks. We will call this algorithm DirectSVM. Essentially, the algorithm consist of a single layer perceptron that we augment with the principle of Vapnik-Chervonenkis based learning-capacity control [13], in the same way as in the SVM. The resulting algorithm is not based on quadratic programming, but uses an iterative scheme based on a few simple heuristics.

The paper is organized as follows. First, we give an overview of the original Support Vector Machine algorithm and introduce our proposed DirectSVM algorithm in a conceptual way. In the next section, we propose two concrete implementations of the algorithm: DirectSVM I that operates in the input space and DirectSVM II that operates in a kernel-feature space. In the final sections, we report on the experimental results and conclude.

DIRECTSVM

Support Vector Machine background

For the reader new to Support Vector Machines, we briefly summarize the SVM in this sub-section.

For most learning systems, the designer of the system has to predefine the system's architecture and the system's parameter(s) *ad hoc* and hence he/she needs to control manually the capacity of the approximating functions of the used learning algorithm. In contrast, the SVM approach proposes an algorithm to control the capacity of the system automatically. Statistical Learning Theory [13] provides the theoretical framework for this automatic algorithm as follows. Vladimir Vapnik shows that the learning system's generalization ability is bounded by the learning capacity of the system, which in turn can be expressed by the Vapnik-Chervonenkis (VC) dimension of the system. Using this result, the SVM minimizes the VC-dimension of the system in order to minimize generalization error on future data, a principle called Structural Risk Minimization.

Concretely the Support Vector Machine for a two-pattern classification problem starts from a linear classifier, a hyper-plane. The VC-dimension of this linear classifier is adapted by introducing the notion of a margin of this classifier to the closest opposite-class training points to this hyper-plane. Linear classifiers with a large margin have a smaller VC-dimension and hence better generalization properties than a linear classifier with a small margin. The SVM searches searches as Optimal Hyper-Plane (OHP), the hyper-plane with maximal margin.

In brief (we refer to the literature for a full exposition), given training data $(\mathbf{x}_1, y_1), ..., (\mathbf{x}_l, y_l)$, $\mathbf{x}_i \in \Re^N, y_i \in \{-1, 1\}$ the OHP, $\mathbf{w} \cdot \mathbf{x} + b = 0$, can be found by minimizing:

$$\frac{1}{2}||\mathbf{w}||^2 \qquad \text{constrained by } y_i((\mathbf{x}_i \cdot \mathbf{w} + b) \geq 1 \quad \text{for } i = 1, \ldots, l$$

In order to solve this optimization problem, the method of Lagrange multipliers can be used *i.e.* one has to find the saddle point of the Lagrangian: $L = \frac{1}{2}||\mathbf{w}||^2 - \sum_{i=1}^{l} \alpha_i y_i (\mathbf{x}_i \cdot \mathbf{w} + b) + \sum_{i=1}^{l} \alpha_i$ with α_i the non-negative Lagrange multipliers. This is equivalent to finding the saddle point in the dual formulation, by maximizing: $L_{\text{dual}} = -\frac{1}{2} \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) + \sum_{i=1}^{l} \alpha_i$ with constraints $0 \leq \alpha_i$, for $i = 1 \ldots l$ and $\sum_{i=1}^{l} \alpha_i y_i = 0$. This is a quadratic programming problem. The solution of this problem, $\mathbf{w}^*$, can be written as a linear combination: $\mathbf{w}^* = \sum_{i=1}^{l} \alpha_i y_i \mathbf{x}_i$.

The $\mathbf{x}_i$ for which $\alpha_i \neq 0$ are defined as the support vectors, since only they determine the OHP. Geometrically, the support vectors correspond to the previously mentioned closest points to the OHP.

Linear SVMs can be generalized to non-linear SVMs by first mapping the input $\phi(\mathbf{x})$ to a high-dimensional feature space (say with dimension M) and to search the OHP in that new space. However, there is no need to compute this mapping explicitly within the above formulation [2]. Since only scalar products are calculated in the input space, this scalar product can be replaced by a kernel $K(\mathbf{x}, \mathbf{y}) = \sum_{j=1}^{M} \lambda_j \phi_j(\mathbf{x}) \phi_j(\mathbf{y})$ (under certain conditions) to realize this mapping implicitly. Using such a kernel, the separation function $f(\mathbf{x})$ can be written in input space as:

$$f(\mathbf{x}) = \operatorname{sgn}\left(\sum_{i=1}^{l} \alpha_i K(\mathbf{x}, \mathbf{x}_i) + b\right)$$

Note that by using this kernel method, one is able to classify data in a high-dimensional feature space, while requiring only computations in the (low-dimensional) input space. Hence avoiding "the curse of dimensionality". Note that the dimensionality M of the feature space can be very high. *E.g.* if the input space is of 256 dimensions and, using for example the polynomial kernel $K(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y})^d$, results in a feature space of dimension $M \approx 33.000$ if $p = 2$, and $M \approx 10^6$ if $p = 3$, etc...

One can extend the SVM with the possibility to cope with misclassified data, the so called soft-margin was first introduced in [3]. The SVM implements this aspect by bounding $\alpha_i < C$.

Multi-class pattern recognition systems can be obtained by combining two-class SVMs. We will use the 1-against-1 scheme for constructing a multi-class classifier *i.e.* we construct $\frac{n(n-1)}{2}$ two-class machines (with n the number of objects). Each machine is trained as a classifier for one class against another class. In order to classify test data, pair-wise competition between all the machines is performed and in analogy with a tennis tournament, each winner competes against another winner until a single winner remains. This final winner determines the class of the test data.

DirectSVM algorithm

The idea of the DirectSVM is to find the parameters of the optimal hyper-plane (OHP) without solving the quadratic optimization problem mentioned above. Instead the algorithm is based on the observation that the OHP is determined by the support vectors only, and more precisely, the OHP is determined only by the centers of some of the opposite-class support vectors. Hence the main task of the algorithm is to find the set of the latter support vectors (called candidate support vectors). Two heuristics are central in finding this set of candidate support vectors in an incremental way:

Heuristic I: the two data-points of opposite class that are in distance *closest to each other* are the most probable support vectors and hence select them as initial two candidate support vectors. If not, the next-closest pairs of points of opposite class are candidate support vectors, etc...

Heuristic II: increase the set of candidate support vectors, by searching for the training data-point that *maximally violates* the current orientation of the hyper-plane. Make this data point member of the set of candidate support vectors, by turning the orientation of the hyper-plane in such a way that the latter data-point becomes equally distant from the hyper-plane as the two initial candidate support vectors.

Based on these heuristics, the algorithm can be formulated conceptually as follows:

1. *Initialization:* The origin point of the hyper-plane is the center of the two closest points of opposite class. The latter are also the initial two candidate support vectors. The initial orientation of the hyper-plane is orthogonal to the orientation of the line connecting these two points. The margin of the hyper-plane is always the orthogonal distance of one of these points to the hyper-plane. If the algorithm needs to be reinitialized, use the pair of next-closest data-points of opposite class that have not been used yet.

2. *Update:* Choose the point with the largest violation with respect to this hyper-plane and make it a new candidate support vector as follows. Rotate the hyper-plane precisely as much as necessary, such that the rotated hyper-plane runs through the center of, this new candidate support vector and the initial candidate support vector of opposite class. The actual update is the latter rotation, subtracted by the orthogonal

projections of all previous rotations. If the update is zero, then a linear dependency has been discovered. If the number of updates is equal to the number of training examples or exceeds the dimensionality of the data-space then reinitialize, else do next update.

3. *End criterium:* If no point violates the hyper-plane or does not lie within its margin (within some relaxation parameter C), then the OHP is found.

This algorithm can be formulated explicitly in the input space (DirectSVM I) or in the dual formulation *i.e.* by expressing every data point in function of the training data (DirectSVM II). We describe both variant implementations.

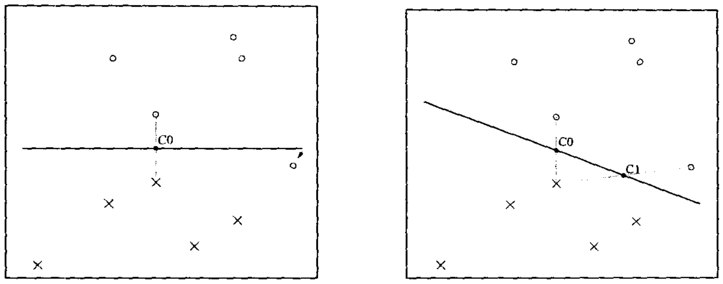

Figure 1: Illustration of (Left) the *initialization* of the DirectSVM algorithm and (Right) the orientation of the hyper-plane after the *first update* in a 2D dimensional input space. Stars and circles represent training points from the two classes that have to be separated. The full line is the current hyper-plane.

IMPLEMENTATIONS

DirectSVM I: learning in the input space

We can write the ortho-normal vector determining the OHP as $\mathbf{w}$, and an origin point contained by the OHP as $\mathbf{c}_0$. The decision function using this OHP can then be written as: $f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} - \mathbf{w} \cdot \mathbf{c}_0$. Note that the OHP can be interpreted as an abstract neuron with $\mathbf{w}$ its weights and $-\mathbf{w} \cdot \mathbf{c}_0$ the bias term, determining its "hard" (non-linear) threshold. The pseudo-code of DirectSVM I can by found in Table 1. The first update of the hyper-plane is illustrated in Fig. 1. The only parameter of the algorithm ($C < 1$) controls the "softness" of the hyper-plane. Typically for the experiments performed, a "hard" hyper-plane ($C = 1 - \epsilon$) gave the best results.

The time-complexity of the algorithm scales in the order of $O(Nm^2)$ with N the dimensionality of the space and m the number of updates. This scaling factor is obtained as follows. Since at every update orthogonalization has to be performed with all previous updates, the number of computational steps for all updates is $\frac{m(m-1)}{2}$. Given that every computational step scales with N, gives the former time-complexity. Hence we can conclude: since the

```
(1) Choose an origin point $\mathbf{c_0} = \frac{\mathbf{x}^+ + \mathbf{x}^-}{2}$
    *Heuristic I: choose the pair $(\mathbf{x}^+, \mathbf{x}^-)$ that is closest/next closest to each other*
    Choose initial $\mathbf{w_0}$ such that $\mathbf{w_0} \cdot (\mathbf{x}^+ - \mathbf{x}^-) = 0$

(2) Calc $f(\mathbf{x_i}) = \mathbf{w_k} \cdot \mathbf{x_i} - \mathbf{w_k} \cdot \mathbf{c_0} \quad \forall i = 1 \dots l$
(3) if $[y_i f(\mathbf{x_i})] \geq C f(\mathbf{x}^+) \ \forall x_i$ , then "Best hyper-plane found"
    Else choose a $x_*$ with $[y_* f(\mathbf{x_*})] < f(\mathbf{x}^+)$
    *Heuristic II: choose the $x_*$ with minimal $[y_i f(\mathbf{x_i})]$*
    if $(y_* > 0)$ then $\mathbf{c_k} = \frac{\mathbf{x}^- + \mathbf{x_*}}{2}$ else $\mathbf{c_k} = \frac{\mathbf{x}^+ + \mathbf{x_*}}{2}$
    $\mathbf{r_k} = \mathbf{c_k} - \mathbf{c_0}$ and $\mathbf{r}'_\mathbf{k} = \frac{\mathbf{r_k}}{\|\mathbf{r_k}\|}$
    $\mathbf{\Delta w} = \left(\mathbf{w}'_\mathbf{k-1} \cdot \mathbf{r}'_\mathbf{k}\right) \mathbf{r}'_\mathbf{k}$ and $\mathbf{\Delta w}^\perp = \mathbf{\Delta w} - \sum_{\mathbf{m=1}}^{\mathbf{k-1}} \left(\mathbf{r}'_\mathbf{m} \cdot \mathbf{r}'_\mathbf{k}\right) \mathbf{r}'_\mathbf{m}$
    $\mathbf{w_k} = \mathbf{w}'_\mathbf{k-1} - \mathbf{\Delta w}^\perp$ and $\mathbf{w}'_\mathbf{k} = \frac{\mathbf{w_k}}{\|\mathbf{w_k}\|}$
    If $[y_* f(\mathbf{x_*})] < 0$ after update then go to (1)
(4) repeat (2) and (3) maximally $\min\{N, l\}$ times.

(5) go to (1), or if maximal trials is attained: "Approximative hyper-plane found"
```

Table 1: *DirectSVM I* pseudo code: DirectSVM in input space. **c** are centers of opposite class candidate support vectors and **r** is the amount of rotation that the hyper-plane has to perform in order to take into account the new candidate support vector.

number of updates is less or in the order of the number of support vectors, and the number of support vectors is a decreasing fraction of the number of training vectors, the time-complexity of the algorithm increases slower than quadratic in the number of training vectors. Note that the original SVM algorithm scales typically worse than quadratic and scales up to *cubic* in the number of training examples, and hence DirectSVM I has a much better scaling behavior.

We point out that in quite a number of cases this implementation of DirectSVM can be useful, since quite a lot of pattern recognition tasks can be solved with a linear classifier only. If the input space dimensionality is larger or comparable to the number of training examples, the learning data will be most likely linearly-separable (except in a few pathological cases only) and regardless of the amount of prior knowledge of the task available, a linear maximal-margin classifier will, almost always, also be the best classifier in these cases. Note that application domains with high-dimensional input spaces are quite common. For example many problems in perception have input spaces that have dimensions in the order of a million or more.

DirectSVM II: learning in a kernel-feature space

If one wants to build more complex classifiers than linear hyper-planes, one can map the data into a higher-dimensional feature space (with dimension M) and then search a hyper-plane in this mapped space. In analogy with the SVM, this mapping can be performed implicitly and computationally efficient by the use of the kernel method [2]. This kernel method requires the formulation of the OHP in terms of the (mapped) training data only (*i.e.* this is the dual formulation). Concretely, the normal vector of the OHP can be rewritten as a linear combination of the training vectors as follows:

```
(1) Choose an origin point $\mathbf{C_0} = (0, \ldots, 0)$ except for $C_0^{x^+} = C_0^{x^-} = \frac{1}{2}$.
    *Heuristic I: choose the pair $(\mathbf{x}^+, \mathbf{x}^-)$ that is closest/next closest to each other in feature space*
    Choose initial $\mathbf{W_0} = (0, \ldots, 0)$ except for $W_0^{x^+} = \frac{1}{2}$ and $W_0^{x^-} = -\frac{1}{2}$.
(2) Calc $f(\mathbf{x_i}) = \sum_a^l W_k^a K(x_a, x_i) - \sum_{a,b}^l W_k^a C_0^b K(x_a, x_b) \quad \forall i = 1 \ldots l$
(3) if $[y_i f(\mathbf{x_i})] \geq C f(\mathbf{x}^+) \ \forall x_i$ , then "Best hyper-plane found"
    Else choose a $x_*$ with $[y \cdot f(\mathbf{x_*})] < f(\mathbf{x}^+)$
    *Heuristic II: choose the $x_*$ with minimal $[y_i f(\mathbf{x_i})]$*
    $\mathbf{C_k} = (0, \ldots, 0)$ except $C_k^{x^*} = \frac{1}{2}$ and if $(y_* > 0)$ then $C_k^{x^-} = \frac{1}{2}$ else $C_k^{x^+} = \frac{1}{2}$
    $\mathbf{R_k} = \mathbf{C_k} - \mathbf{C_0}$ and $\mathbf{R}_\mathbf{k}' = \frac{\mathbf{R_k}}{\|\mathbf{R_k}\|}$
    $\mathbf{\Delta W} = \left(\sum_{a,b}^l W_{k-1}'^a R_k'^b K(x_a, x_b)\right) \mathbf{R}_\mathbf{k}'$ & $\mathbf{\Delta W}^\perp = \mathbf{\Delta W} - \sum_{m=1}^{k-1} \left(\sum_{a,b}^l R_m'^a R_k'^b K(x_a, x_b)\right) \mathbf{R}_\mathbf{m}'$
    $\mathbf{W_k} = \mathbf{W_{k-1}} - \mathbf{\Delta W}^\perp$ & $\mathbf{W_k} = \frac{\mathbf{W_k}}{\|\mathbf{W_k}\|}$
    If $[y_* f(\mathbf{x_*})] < 0$ after update then go to (1)
(4) repeat (2) and (3) maximally $\min\{M, l\}$ times.
(5) go to (1), or if maximal trials is attained: "Approximative hyper-plane found"
```

Table 2: *DirectSVM II* pseudo code: DirectSVM in kernel-feature space.

$\mathbf{w} = \sum_{i=1}^l w_i \mathbf{x}_i$. We can equivalently write the OHP in a feature space as a linear combination of the mapped training vectors (we use capitals for all vectors written in the dual notation): $\mathbf{W} = \sum_{i=1}^l W_i \phi(\mathbf{x_i})$. Hence the OHP in feature space is determined by its coefficients $\mathbf{W} = (W_0, \ldots, W_l)$ and its origin can be determined by the coefficients $\mathbf{O} = (O_0, \ldots, O_l)$. An update consist of modifying some of the coefficients of $\mathbf{W}$ in a similar way as in DirectSVM I. The pseudo code for DirectSVM II can be found in Table 2.

The time required to execute this implementation is in the order of $O(m^3)$, with m the number of updates required. This estimate is based on the following: all updates with their orthogonalization require $\frac{m(m-1)}{2}$ computational steps. Every computational step involves computations with previous support vectors and if one uses sparse coding, m calculations have to be performed. Following the same path of reasoning as in DirectSVM I, this implementation scales better than cubic in the number of training examples. This is comparable to the original SVM algorithm and hence actual speed-up has to be determined experimentally and may depend on the particular dataset used.

EXPERIMENTAL RESULTS

We compare our results with the results obtained with the standard implementation of the original SVM developed at Royal Holloway[12]. We also compare with the Nearest Neighbor Classifier (NNC) as a reference classifier. All execution times were measured on a SUN workstation with a 143 Mhz UltraSPARC processor.

Results for DirectSVM I are on the Columbia Image Library (COIL) database [7] and on the extended COIL database [11]. The former is one of the few *standard* databases available, containing data of dimensionality larger than the total number of training data available. Results for DirectSVM II are on the well-known Sonar database of Gorman & Sejnowski [4], now part of the UCI repository of machine learning databases. We are also in the process of testing the learning algorithms on handwritten digit recognition databases, but at the moment of writing these results are not yet available.

COIL database

The COIL database consists of images of 100 three-dimensional objects each taken from 72 different views. The color images were sub-sampled and presented as data-points in a 768 dimensional space. We varied the difficulty of the problem by varying the number of views per object shown during training. The task is to recognize 30 different objects from unseen views (views not presented during training). Average performance (misclassified views) is shown in Table 3. One can see that the performance of SVM and DirectSVM is very similar. The computational requirements for the experiments are different however (Table 5): DirectSVM performs the tasks substantially faster and with less memory requirements than SVM.

	#training views/single object				
	36	12	8	4	2
DirectSVM	0.0	0.4	2.0	7.8	20.5
SVM	0.0	0.3	1.7	7.8	20.5
NNC	0.0	1.4	3.7	11.2	22.5

Table 3: *Test-set error-rates (erroneously classified unseen views) on the COIL 3D object image database* while increasing (from left to right) the demands on the generalization ability of the learning algorithms.

Extended COIL database

The extended COIL database [11] contains the COIL images of 3D objects on a variety of different backgrounds. We use the BW training scheme[11], consisting of training every view of the 3D objects once with a black background and once with a white background, in order to make the classifier background-invariant. The other experiment variables are the same as in previous experiment and we keep the number of training views per single object at a constant 36. Also here DirectSVM performs the tasks with similar accuracy as SVM (Table 4), but DirectSVM is faster and uses much less computational resources (Table 5).

UCI Sonar database

The Sonar database consists of 208 measurements of sonar data of respectively rocks and mines. The data is represented in a 60-dimensional space. The training set consists of 104 data-points, while the other 104 data-points are used for testing the system. We used the polynomial kernel: $K(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y})^d$ and vary its degree d. The best performance (squares) of DirectSVM

	Extended COIL: object images with unseen backgrounds				
	scene+original	scene+black	scene+white	grey	noise
DirectSVM	2.8	5.7	1.8	0.0	0.5
SVM	3.0	5.5	1.7	0.0	0.4
NNC	25.1	15.5	22.5	75.7	73.5

Table 4: *Test-set error-rates on the extended COIL database*: object recognition with different backgrounds, all using the BW training scheme for learning background-invariance.

	Training		Testing
	Time	Memory	Time per image
DirectSVM	2 min $\rightarrow$ 9 min	2 MB $\rightarrow$ 3 MB	10 ms
SVM	8 min $\rightarrow$ 42 min	202 MB $\rightarrow$ 396 MB	550 ms
NNC	0	1 MB $\rightarrow$ 2 MB	80ms $\rightarrow$ 160 ms

Table 5: Approximate *required computational resources* for training and testing on the COIL database (left of the arrow) and on the extended COIL database (right of the arrow). The latter experiment requires two times larger training sets.

and SVM are identical (see Table 6). Performance reported [4] on this database for the NNC classifier is 82.7% (18 errors). The time required to perform this experiment was 1.1 seconds for DirectSVM which is faster than the 2.5 seconds for SVM.

	Kernel polynomial degree (d)							
	1	2	3	4	5	6	7	8
DirectSVM	26	15	12	15	$\boxed{10}$	11	11	12
SVM	25	13	12	11	11	$\boxed{10}$	$\boxed{10}$	11

Table 6: The number of errors on the test set of the *sonar database*. The test-set size was 104. Note that the best performance (squares) of DirectSVM and SVM are identical, although the degree of the polynomial kernel at which it is obtained is different.

CONCLUSION

The DirectSVM algorithm has been proposed that implements the Support Vector Machine without the need to code and solve a complex quadratic programming problem. Instead, DirectSVM follows a iterative update scheme that is based on a few intuitively-simple heuristics. Experimental results on the COIL, the extended COIL, and the UCI Sonar database, indicate that DirectSVM is able to find solutions that are similar as the original SVM, but requiring substantially less computational resources and training time.

Acknowledgments
D.R. wants to thank Wendy Li for proof-reading the manuscript. D.R. is supported by a TMR grant of the European Commission (ERBFMBICT961126).

REFERENCES

[1] Blanz, V., Schölkopf, B., Bülthoff, H., Burges, C., Vapnik, V. and Vetter, T., "Comparison of view-based object recognition algorithms using realistic 3D models", *Proc. Int. Conf. on Artificial Neural Networks 1996 (ICANN96)*, 251-256.

[2] Boser, B., Guyon, I. and Vapnik, V., "A training algorithm for optimal margin classifiers", *5th Ann. Workshop on Computational Learning Theory*, Pittsburgh ACM, 144-152, 1992.

[3] Cortes, C., Vapnik, V., "Support-Vector Networks", *Machine Learning* **20**, 1-25.

[4] Gorman, R. P., and Sejnowski, T. J., "Analysis of Hidden Units in a Layered Network Trained to Classify Sonar Targets", *Neural Networks* **1**, 75-89, 1988.

[5] Joachims, T., "Making Large-Scale Support Vector Machine Learning Practical" in *Advances in Kernel Methods*, Schölkopf, B., Burges, C., Smola, A., eds., 169-184, 1999.

[6] LeCun, Y., Jackel, L., Bottou, L., Brunot, A., Cortes, C., Denker, J., Drucker, H., Guyon, I., Müller, U., Säckinger, E., Simard, P. and Vapnik, V., "Comparison of learning algorithms for handwritten digit recognition", *Proc. Int. Conf. on Artificial Neural Networks 1995 (ICANN95)*, 53-60.

[7] Nene, S. A., Nayar, S. K. and Murase, H., "Columbia Object Image Library (COIL-100)", Techn. Rep. No. CUCS-006-96, dept. Comp. Science, Columbia University, 1996.

[8] Osuna, E., Freund, R., Girosi, F., "An improved training algorithm for support vector machines", *Proc. IEEE Neural Netw. for Sign. Proc. NNSP97*, 276-285, 1997.

[9] Platt, J., "Sequential Minimal Optimisation: a fast algorithm for training support vector machines", Techn. Rep. MSR-TR-98-14, Microsoft Research, 1998.

[10] Pontil, M. and Verri, A. "Support Vector Machines for 3D Object Recognition", *IEEE Trans. on Pattern Analysis & Machine Intelligence* **20**, 637-646, 1998.

[11] Roobaert, D., "Improving the Generalisation of Linear Support Vector Machines: an Application to 3D Object Recognition with Cluttered Background", *Proc. SVM Workshop at Int. Joint Conf. Artif. Intell. (IJCAI99)*, Stockholm, Sweden, August 1999.

[12] Saunders, C., Stitson, M.O., Weston, J., Bottou, L., Schölkopf, B. and Smola, A., "Support Vector Machine - Reference Manual", CSD-TR-98-03, Royal Holloway Techn. Rep., 1998.

[13] Vapnik, V. N., *Statistical Learning Theory*, in series Adaptive & Learning Systems for Signal Proc., Comm. & Control, John Wiley & Sons, New-York, 1998.