Support vector classifiers: a first look

David Tax, Dick de Ridder and Robert P.W. Duin
Pattern Recognition Group
Faculty of Applied Physics, Delft University of Technology
Lorentzweg 1, 2628 CJ Delft, The Netherlands
e-mail: {davidt,dick}@ph.tn.tudelft.nl

Keywords: pattern recognition, support vector classifiers, handwritten digit recognition

Abstract

To find a decision boundary in a two-class classification problem, one often tries to find the class probability densities first, before constructing the decision boundary. In support vectors classifiers the input vectors are mapped to a high dimensional feature space and are then separated by the optimal linear hyperplane. Because in this method no probability density is computed, it becomes highly insensitive to the "curse of dimensionality". In this paper we will show some advantages and drawbacks of these support vector classifiers. In our opinion, the support vector classifier is a very promising new classification technique, with a potential for broad application.

1 Introduction

When confronted with a classification problem, one often would like to use as much features as possible to improve the classification result. However, most classifiers suffer from what is called the small sample size effect or peaking effect. That is, there is a certain optimal number of features beyond which performance will only degrade. The reason for this is, that as the dimensionality of the feature space grows, estimating densities of the data becomes harder and harder for a finite number of samples. Only for an infinite number of training samples the error will approach the Bayes error, the minimal error that can be reached. Within certain limits, this error behaviour holds as a function of the classifier complexity too (see figure 1).

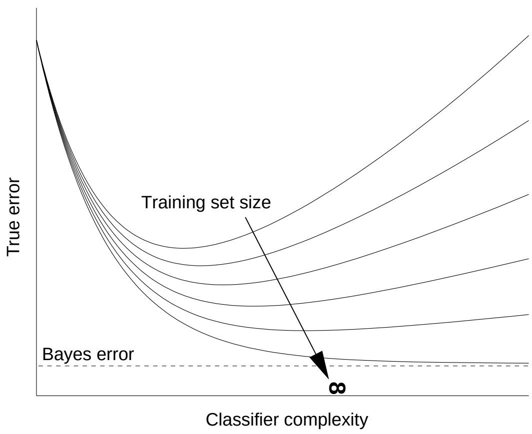

Figure 1: The peaking effect.

Vapnik [Vap95] argued that when a problem has to be solved, one has to try to avoid solving a more general problem as an intermediate step. When a classification boundary between two classes has to be found, the computation of a probability density has to be avoided.

In line with this idea, in 1965 Vapnik [Vap82] proposed a method of finding a hyperplane optimally dividing two classes, which does not depend on a probability estimation. This optimal hyperplane is a linear decision boundary which separates the two classes and leaves the largest margin between the vectors of the two classes (see fig. 2). It was observed that the optimal hyperplane is determined by only a small fraction of the data points, the so-called *support vectors*. Furthermore, it was shown that the probability of making an error depends only on the number of these support vectors (and therefore the complexity of the classifier) and the number of training vectors. In this procedure the dimensionality of the feature space is of no importance, so it becomes highly insensitive to the peaking effect. This method was, however, only fit for separable classes.

To find a non-linear classifier, it is possible to transform the vectors in *input space* to some high-dimensional *feature space*. For example, when classifying digit images (16×16 pixels) using a polynomial classifier of degree d, the feature space is 256^d-dimensional. By this mapping the simple linear classifier transforms into some non-linear decision surface. This allows for more difficult problems to be solved and increases the feature space dimensionality, but hardly influences the classifier complexity. Boser et al. [BV92] showed that it is possible to transform the input without much extra cost for computing the optimal hyperplane in feature space.

In 1995 Cortes and Vapnik [CV95] extended this method for the case of non-separable classes, so this support vector classifier (SVC) became a general tool for solving classification problems.

We believe that the SVC may be one of the most important proposals of the last years. This paper intends to discuss the SVC and its implementation (section 2), some experiments we performed with it (section 3), problems we encountered during the implementation (section 4) and finally, some ideas regarding future research into this classifier (section 5).

In this paper, we refer mostly to the paper by Cortes and Vapnik [CV95], except where indicated.

2 Support vector classifiers

2.1 The optimal hyperplane

The optimal hyperplane is defined to be a plane separating two separable classes in such a way that its margin is as wide as possible. All vectors have to lie either on or outside of the margin (fig. 2). The discrimination function is then defined in such a way that for vectors lying on the margin on one side its output is -1 or smaller, for vectors on the other side 1 or greater.

If we denote the training vectors by $\mathbf{x}_i$, $i = 1, ..., l$ with corresponding labels $y_i \in \{-1, 1\}$ then this translates to

$$y_i(\mathbf{w}_o \cdot \mathbf{x}_i + b_o) \geq 1, \;\; i = 1, ..., l \qquad (1)$$

if the optimal hyperplane is

$$\mathbf{w}_o \cdot \mathbf{x} + b_o = 0 \qquad (2)$$

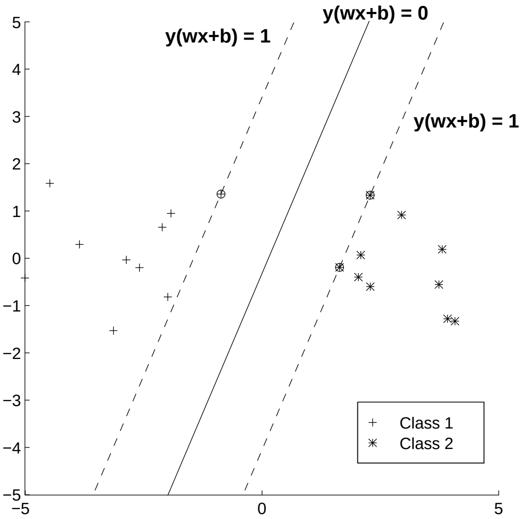

Figure 2: An optimal hyperplane. Support vectors are indicated by an **o**.

in which $\mathbf{w}_o$ can be written as

$$\mathbf{w}_o = \sum_{i=1}^{l} \alpha_{o_i} y_i \mathbf{x}_i \qquad (3)$$

with $\alpha_{o_i} \geq 0$. The discrimination function for a certain vector $\mathbf{z}$ is therefore written as

$$f(\mathbf{z}) = \sum_{i=1}^{l} \alpha_{o_i} y_i (\mathbf{z} \cdot \mathbf{x}_i) + b_o \qquad (4)$$

In fact, since only the support vectors contribute to the optimal hyperplane, most α_{o_i} will be zero. Support vectors are those vectors for which the equality in eqn. 1 holds.

It can be shown (see [CV95]) that to find the optimal set of weights α_{o_i}, the following expression has to be maximized:

$$W(\mathbf{\Lambda}) = \mathbf{\Lambda}^T \mathbf{1} - \frac{1}{2} \mathbf{\Lambda}^T \mathbf{D} \mathbf{\Lambda} \qquad (5)$$

w.r.t. $\mathbf{\Lambda} = (\alpha_1, ..., \alpha_l)$, subject to constraints:

$$\mathbf{\Lambda} \geq \mathbf{0} \qquad (6)$$

$$\mathbf{Y}^T \mathbf{\Lambda} = 0 \qquad (7)$$

In eqn. 5, $\mathbf{D}$ is a matrix containing dot products of the training vectors, multiplied by their labels:

$$\mathbf{D}_{ij} = y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) \qquad (8)$$

Equation 5 is a quadratic programming problem with linear constraints, which can be solved using more or less standard algorithms.

2.2 The soft margin hyperplane

The concept of the optimal hyperplane developed in the previous section is only fit for the separable case. To make it a generally applicable tool, it should be able to handle cases in which the classes overlap.

For this purpose, eqn. 1 is adapted to allow for errors:

$$y_i(\mathbf{w}_o \cdot \mathbf{x}_i + b_o) \geq 1 - \xi_i, \; i = 1, ..., l \quad (9)$$

$$\xi_i \geq 0, \; i = 1, ..., l \quad (10)$$

and the following function can be minimized:

$$\tfrac{1}{2}\mathbf{w}^2 + C\mathcal{F}(\textstyle\sum_{i=1}^{l} \xi_i^\sigma) \quad (11)$$

Note that, for very small σ, the sum of ξ_i^σ counts the number of errors. However, the smallest value of σ for which the quadratic programming problem still has a unique solution is 1. Also, the choice of $\mathcal{F}$ is still free. The only demand on the function $\mathcal{F}$ is that it is a convex function with $\mathcal{F}(0) = 0$. In [CV95], $\mathcal{F}(u) = u^2$ is chosen to simplify calculations. In fact, for $\mathcal{F}(u) = \mathcal{F}'(u^d)$ with $d > 2$ the quadratic programming problem becomes a convex problem and can be less easily solved.

The quadratic programming problem (eqn. 5) now becomes a dual one:

$$W(\mathbf{\Lambda}, \delta) = \mathbf{\Lambda}^T\mathbf{1} - \frac{1}{2}\left[\mathbf{\Lambda}^T\mathbf{D}\mathbf{\Lambda} + \frac{\delta}{C}\right] \quad (12)$$

w.r.t. $\mathbf{\Lambda} = (\alpha_1, ..., \alpha_l)$ and δ, subject to constraints:

$$\mathbf{0} \leq \mathbf{\Lambda} \leq \delta\mathbf{1} \quad (13)$$

$$\mathbf{Y}^T\mathbf{\Lambda} = 0 \quad (14)$$

This can be rewritten as:

$$\mathbf{\Lambda}' = \begin{pmatrix} \mathbf{\Lambda} \\ \delta \end{pmatrix} \quad (15)$$

$$W(\mathbf{\Lambda}') = \mathbf{\Lambda}'^T \begin{pmatrix} \mathbf{1} \\ 0 \end{pmatrix} - \frac{1}{2}\left[\mathbf{\Lambda}'^T \begin{pmatrix} \mathbf{D} & \mathbf{0} \\ \mathbf{0}^T & \frac{1}{C} \end{pmatrix} \mathbf{\Lambda}'\right] \quad (16)$$

subject to constraints:

$$\begin{pmatrix} -\mathbf{I} & \mathbf{1} \\ \mathbf{0}^T & 1 \end{pmatrix} \mathbf{\Lambda}' \geq 0 \quad (17)$$

$$\begin{pmatrix} \mathbf{Y}^T & 0 \end{pmatrix} \mathbf{\Lambda}' = 0 \quad (18)$$

where $\mathbf{I}$ is an $l \times l$ identity matrix.

The only remaining parameter now is C, which controls the trade-off between the complexity of the decision function on the one hand (since it forms an upper bound on the value of the α_i) and number of errors on the other hand.

2.3 Dot products in feature space

As was stated in section 1, normally a non-linear decision surface is obtained by transforming the input vectors to a (high-dimensional) feature space and calculating a classifier in that space. It was also shown that this can be problematic, since these feature spaces can have quite large dimensionalities.

However, it can be shown [CV95] that, since the SVC's discrimination function is based on dot products of vectors and support vectors only and linear in the support vectors, there is no need to actually transform the input vectors to feature vectors before taking the dot product. Instead, the dot product of two vectors can be performed in input space after which the transformation can be applied to the resulting scalar.

Furthermore, the dot product can be replaced by a generalized dot product $K(\mathbf{x}, \mathbf{y})$: any function satisfying Mercer's theorem (see [CV95]).

The only equations that change are eqn. 8, which becomes

$$\mathbf{D}_{ij} = y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \quad (19)$$

and the discrimination function $f(\mathbf{z})$ (eqn. 4):

$$f(\mathbf{z}) = \sum_{i=1}^{l} \alpha_{o_i} y_i K(\mathbf{z}, \mathbf{x}_i) + b_o \quad (20)$$

In this way, arbitrarily complex decision surfaces can be calculated. Some examples are:

- A polynomial classifier of degree d:

  $$K(\mathbf{z}, \mathbf{x}_i) = (\mathbf{z} \cdot \mathbf{x}_i + 1)^d \quad (21)$$

- A radial basis function neural network with kernel width σ, of which the necessary number of kernels and their positions are found by the algorithm:

  $$K(\mathbf{z}, \mathbf{x}_i) = \exp\left(-\frac{|\mathbf{z} - \mathbf{x}_i|^2}{\sigma^2}\right) \quad (22)$$

- A 2-layer feedforward neural network[1] of which the number of hidden units and all weights are determined by the algorithm:

  $$K(\mathbf{z}, \mathbf{x}_i) = \tanh(c_1(\mathbf{z} \cdot \mathbf{x}_i) + c_2) \quad (23)$$

For each of these functions, some parameters have to be chosen in advance (e.g., the degree d).

[1] This function is a valid general dot product function only for certain (unspecified) values of c_1 and c_2.

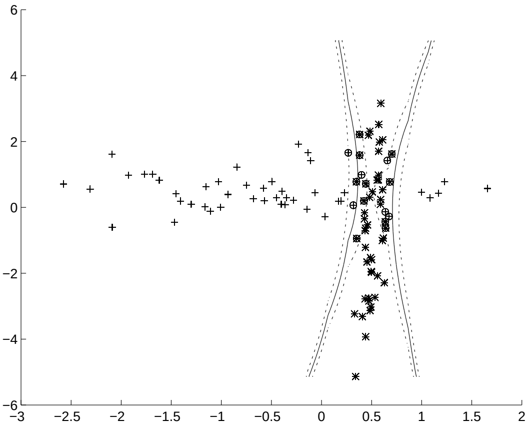

Figure 3: The decision boundary using a polynomial dot product of degree two.

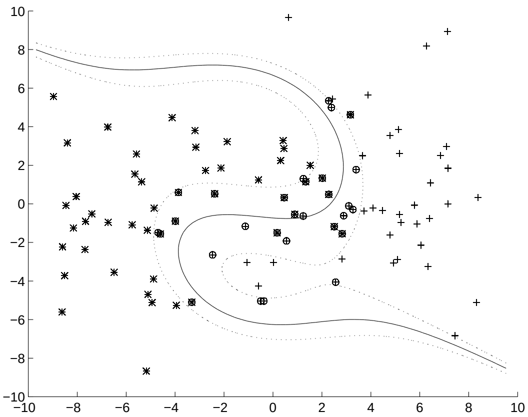

Figure 4: The decision boundary using a polynomial inproduct of degree three.

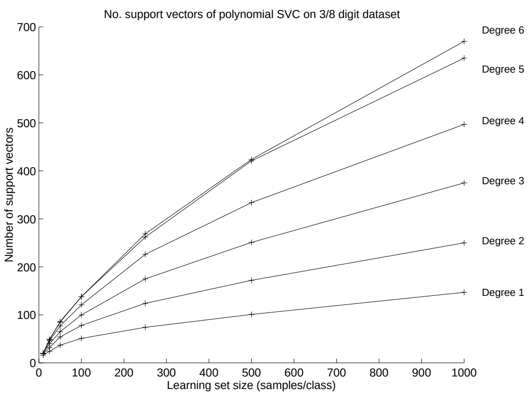

Figure 5: The number of support vectors on a two-digit problem (distinguish between 3 and 8).

3 Experiments

In this section some small artificial classification problems (which can be visualized) and a real-life problem will be treated. We implemented the soft margin hyperplane algorithm in Matlab, using QLD (see section 7) to solve the quadratic programming problem (eqn. 16-18).

3.1 Two dimensional problems

In figure 2 a polynomial of degree one (i.e., a linear function) is used to discriminate between two classes. These classes are separable and the margins are as wide as possible. Note that in this case all support vectors are on the lines $y_i(\mathbf{w}_o\mathbf{x}_i + \mathbf{b}_o) = 1$.

In figures 3 and 4 polynomials of degree 2 and 3 were used. The data in these figures cannot be separated without errors. The support vectors were now not only on the borders of the class, but the data points which were classified wrong also become support vectors. In figure 4 the strength of support vectors becomes apparent. With only 18 support vectors, the two classes could be separated with only two errors.

3.2 Handwritten digit recognition

In table 1 the performance of an SVC is shown on a digit recognition problem. In this problem, 16×16 pixel images of handwritten digits 3 and 8 have to be separated (see [RHD96]). We trained several SVC's using training sets of different sizes, as well as a k-nearest neighbour classifier (for which the optimal k was chosen using cross-validation), a shared weights neural network, the LeNotre architecture (see [RHD96]) and a standard feed-forward neural network with one hidden layer. Classification performance was measured on a test set containing 1000 digits per class.

It can be observed that the performance does not deteriorate with increasing degree of the polynomial (and thus the dimensionality of the feature space). This indicates that using more features does not degrade the performance of the SVC. Note also that the SVC (with degree larger than one) outperforms all other classifiers. This is even more impressive when one realizes that the SVC does not include any a priori knowledge, such as shift-invariant feature detection, like the LeNotre network.

Train size	SVC with polynomial of degree						k-NN	LeNotre	256:256:2
	1	2	3	4	5	6			ANN
10	6.50	6.95	7.35	8.20	9.25	10.40	9.65	11.00	8.70
25	6.45	5.35	5.35	5.45	5.70	6.25	8.80	10.30	8.00
50	3.90	3.35	3.30	3.60	4.70	5.60	5.50	6.90	4.15
100	3.10	2.00	1.70	1.85	2.10	2.20	3.90	5.00	3.30
250	3.10	1.75	1.65	1.65	1.55	1.95	2.25	3.15	2.90
500	3.00	1.20	1.05	0.95	1.00	1.05	1.55	2.40	2.65
1000	2.95	0.75	0.80	0.70	0.70	0.60	1.30	2.25	2.25

Table 1: The performance, in % error on an independent test set, on a two digit problem (distinguish between 3 and 8). The influence of the degree of the polynomial on the performance.

Also, the computation time requirements of the algorithm are fair. The training time is somewhat longer than LeNotre, but shorter than the standard neural network. The time required for classifying is much lower however, since it only depends on the (low) number of support vectors.

In figure 5 the number of support vectors for different degree of polynomial dot product is shown. Although the number of free parameters explodes with increasing degree (e.g., the number of features of a 6^{th} degree polynomial of a 256-dimensional input vector is about $4 \cdot 10^{11}$, while the first degree polynomial results in just 256 features), the number of support vectors grows only with a factor of about 6.

4 Some caveats

Although in the SVC the number of the support vectors and their coefficients are obtained automatically, some parameters still have to be chosen beforehand. These are the dimensionality of the feature space (by the choice of the type of mapping to this feature space) and the parameter C (see eqn. 16) which indicates how severely errors have to be punished. These choices highly influence how well the SVC can classify the data and how well it generalizes. When the dimensionality of the feature space is too low, the data may not be separable by a linear hyperplane; when errors are punished too much, the SVC can overfit the training data.

Fortunately, in our experiments the SVC did not seem to suffer too much from overfitting problems due to a too high dimensional feature space. Because the training time does not depend on the dimensionality of the feature space (only on the input space, which is fixed), choosing a high dimensional feature space is a good idea.

The second problem is that the maximization procedure requires that $\mathbf{D}$ (eqn. 12) is positive semi-definite. When this is not the case, we use $\mathbf{D}' = \mathbf{D} + \varepsilon\mathbf{I}$, with ε chosen as small as possible to avoid changing the problem too much, to make $\mathbf{D}'$ positive semi-definite. In most cases ε can stay as small as 10^{-12}, but in hard real-life problems with mappings to high dimensional feature spaces, ε can become too large. In these cases the problem is changed too much and cannot be solved in this way.

A third problem is that the divide-and-conquer technique proposed by Vapnik does not work for classification problems with large overlap. Vapnik argues that a large classification problem (i.e., with a large number of training points) can be divided in several smaller problems and solved incrementally. The training data is divided in small batches and the support vectors for the first training portion are calculated. These support vectors are added to the second batch and from this new portion new support vectors are obtained. This process is continued untill the whole dataset is covered.

This method works in case of separable data, where a data point which is no support vector in one batch can never become a support vector for the complete problem. In the case of non-separable data a batch can be a very bad representation of the data. Then a bad hyperplane is found and data points will be thrown away which are in fact support vectors for the complete problem. Since the memory space requirement of the algorithm is quadratic in the number of data points, this limits the size of the classification problems which can be solved.

5 Outlook

The SVC can be extended in a very interesting way. The equations in section 2 show that the dot product $K(\mathbf{x}, \mathbf{y})$ can be replaced by some similarity measure $S(\mathbf{x}, \mathbf{y})$. The only demand on such a measure S would be that it would give us a symmetric, positive semi-definite matrix $\mathbf{D}$. In that case the $S(\mathbf{x}, \mathbf{y})$ does not have to be a mapping from input space to feature space, but can be freely chosen to obtain an optimal classification performance. For example, one could use some non-linear measurement of the difference between two images, or - in cases where no hard measure can be defined - export opinions on the differences between samples. This would result in a *feature-less* classification technique: no features are used, only similarities or dissimilarities between samples.

Furthermore it is possible to incorporate prior knowledge of the problem. A well-known technique to obtain transformation invariance is to extend the training data by artificial data which is obtained from the original data by applying the desired transformation. Schölkopf et al. [SBV96] showed that transforming not all training data, but only the support vectors, gives the same result, thereby avoiding a large increase of the size of the training set.

6 Conclusion

We have shown the strengths and some weaknesses of the support vector classifiers. The SVC generalizes very well and because it does not have to estimate a probability density, it becomes highly insensitive to the "curse of dimensionality". The SVC is also very versatile; the procedure allows a free choice for the transformation to the high dimensional feature space by choosing $K(\mathbf{x}, \mathbf{y})$.

Still, some drawbacks remain: some parameters have to be chosen beforehand (C, the error weighing function $\mathcal{F}$, the dot product function) and in the case of overlapping classes, the classification can be limited in practice by the size of the problem.

Especially the choices of the dot product function $K(\mathbf{x}, \mathbf{y})$ and the constant C are important. The SVC will always try to minimize the number of errors it makes on the training set, but may adapt the classifier too much to the actual training data. $K(\mathbf{x}, \mathbf{y})$ determines how much freedom (i.e., dimensions) the SVC has to adapt itself; the parameter C controls in how far the discrimination function is adapted to avoid any error. Therefore, the SVC still does not solve the problem of finding an optimally generalizing classifier, given a certain dataset.

7 Acknowledgments

We would like to express our gratitude to A.J. Quist, of the Statistics, Stochastic and Operations Research department of the faculty of Applied Mathematics at the Delft University of Technology. He helped us solve the problems we encountered when implementing the numerical routines in the algorithm. Also, we would like to thank M.J.D. Powell and K. Schittkowski for their QLD algorithm, which was kindly supplied by A.L. Tits in the CFSQP package (by C.T. Lawrence, J.L. Zhou and A.L. Tits). Finally, we thank A. Hoekstra and A. Ypma for some fruitful discussions.

This work was partly supported by the Foundation for Applied Sciences (STW), the Foundation for Computer Science in the Netherlands (SION) and the Dutch Organization for Scientific Research (NWO).

References

[BV92] Guyon, I. Boser, B.E. and Vapnik, V.N. A training algorithm for optimal margin classifiers. In *Proc. of the* 5^{th} *annual workshop of computational learning theory, 5*, pages 144–152. ACM, 1992.

[CV95] Cortes, C. and Vapnik, V. Support-vector networks. *Machine Learning*, 20:273–297, 1995.

[RHD96] Ridder, D. de, Hoekstra, A., and Duin, R.P.W. Feature extraction in shared weights neural networks. In Kerckhoffs, E.J.H. et al., editor, *Proc. of the* 2^{nd} *annual conference of the ASCI, Lommel, Belgium, June 5-7, 1996*, pages 289–294, Delft, 1996. ASCI.

[SBV96] Schölkopf, B., Burges, C., and Vapnik, V. Incorporating invariances in support vector learning machines. In Von der Malsburg, C. et al., editor, *Artificial Neural Networks - ICANN 96*, 1996. 1.

[Vap82] Vapnik, V. *Estimation of dependencies based on empirical data.* Springer-Verlag, New York, 1982.

[Vap95] Vapnik, V. *The nature of statistical learning theory.* Springer-Verlag, Berlin, 1995.