MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY
and
CENTER FOR BIOLOGICAL AND COMPUTATIONAL LEARNING
DEPARTMENT OF BRAIN AND COGNITIVE SCIENCES

A.I. Memo No. 1599 December, 1996
C.B.C.L. Paper No. 142

Comparing Support Vector Machines with Gaussian Kernels to Radial Basis Function Classifiers

B. Schölkopf, K. Sung, C. Burges, F. Girosi, P. Niyogi, T. Poggio, V. Vapnik

This publication can be retrieved by anonymous ftp to publications.ai.mit.edu.

Abstract

The Support Vector (SV) machine is a novel type of learning machine, based on statistical learning theory, which contains polynomial classifiers, neural networks, and radial basis function (RBF) networks as special cases. In the RBF case, the SV algorithm automatically determines centers, weights and threshold such as to minimize an upper bound on the expected test error.

The present study is devoted to an experimental comparison of these machines with a classical approach, where the centers are determined by k–means clustering and the weights are found using error backpropagation. We consider three machines, namely a classical RBF machine, an SV machine with Gaussian kernel, and a hybrid system with the centers determined by the SV method and the weights trained by error backpropagation. Our results show that on the US postal service database of handwritten digits, the SV machine achieves the highest test accuracy, followed by the hybrid approach. The SV approach is thus not only theoretically well–founded, but also superior in a practical application.

This report describes research done at the Center for Biological and Computational Learning, the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology, and at AT&T Bell Laboratories (now AT&T Research, and Lucent Technologies Bell Laboratories). Support for the Center is provided in part by a grant from the National Science Foundation under contract ASC–9217041. BS thanks the M.I.T. for hospitality during a three–week visit in March 1995, where this work was started. At the time of the study, BS, CB, and VV were with AT&T Bell Laboratories, NJ; KS, FG, PN, and TP were with the Massachusetts Institute of Technology. KS is now with the Department of Information Systems and Computer Science at the National University of Singapore, Lower Kent Ridge Road, Singapore 0511; CB and PN are with Lucent Technologies, Bell Laboratories, NJ; VV is with AT&T Research, NJ. BS was supported by the Studienstiftung des deutschen Volkes; CB was supported by ARPA under ONR contract number N00014-94-C-0186. We thank A. Smola for useful discussions.
Please direct correspondence to Bernhard Schölkopf, bs@mpik-tueb.mpg.de, Max–Planck–Institut für biologische Kybernetik, Spemannstr. 38, 72076 Tübingen, Germany.

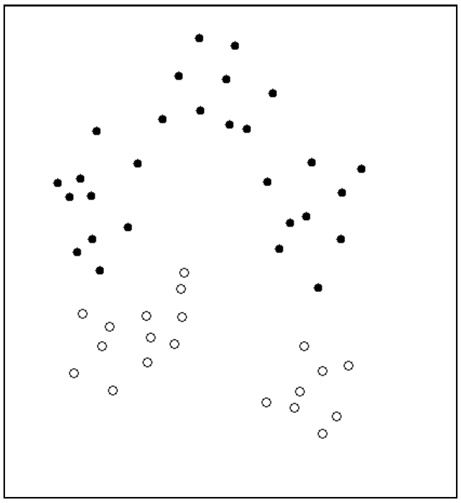

Figure 1: A simple 2–dimensional classification problem: find a decision function separating balls from circles. The box, as in all following pictures, depicts the region $[-1, 1]^2$.

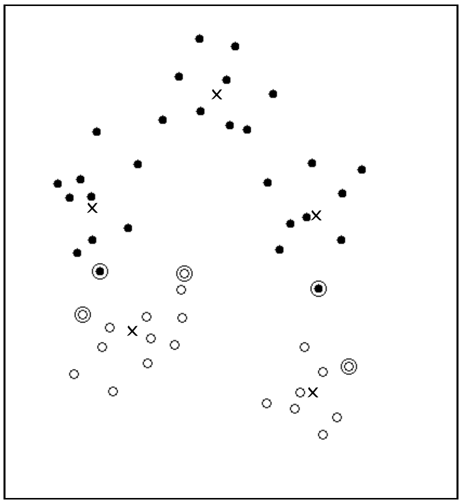

Figure 2: RBF centers automatically found by the Support Vector algorithm (indicated by extra circles), using $c_i = 1$ for all i (cf. Eq. 1). The number of SV centers accidentally coincides with the number of identifiable clusters (indicated by crosses found by k–means clustering with $k = 2$ and $k = 3$ for balls and circles, respectively) but the naive correspondence between clusters and centers is lost; indeed, 3 of the SV centers are circles, and only 2 of them are balls. Note that the SV centers are chosen with respect to the classification task to be solved.

1 Introduction

Consider Fig. 1. Suppose we want to construct a radial basis function classifier

$$D(\mathbf{x}) = \operatorname{sgn}\left(\sum_{i=1}^{\ell} w_i \exp\left(-\frac{\|\mathbf{x}-\mathbf{x}_i\|^2}{c_i}\right) + b\right) \qquad (1)$$

(b and c_i being constants, the latter positive) separating balls from circles, i.e. taking different values on balls and circles. How do we choose the *centers* $\mathbf{x}_i$? Two extreme cases are conceivable:

The first approach consists in choosing the centers for the two classes separately, irrespective of the classification task to be solved. The classical technique of finding the centers by some clustering technique (*before* tackling the classification problem) is such an approach. The weights w_i are then usually found by either error backpropagation (Rumelhart, Hinton, & Williams, 1986) or the pseudo–inverse method (e.g. Poggio & Girosi, 1990).

An alternative approach (Fig. 2) consists in choosing as centers points which are *critical* for the classification task at hand. Recently, the *Support Vector Algorithm* was developed (Boser, Guyon & Vapnik 1992, Cortes & Vapnik 1995, Vapnik 1995) which implements the latter idea. It is a general algorithm, based on guaranteed risk bounds of statistical learning theory, which in particular allows the construction of radial basis function classifiers. This is done by simply choosing a suitable kernel function for the SV machine (see Sec. 2.2). The SV training consists of a quadratic programming problem which can be solved efficiently and for which we are guaranteed to find a global extremum. The algorithm automatically computes the number and location of the above centers, the weights w_i, and the threshold b, in the following way: by the use of a suitable kernel function (in the present case, a Gaussian one), the patterns are mapped nonlinearly into a high–dimensional space. There, an optimal separating hyperplane is constructed, expressed in terms of those examples which are closest to the decision boundary (Vapnik 1979). These are the *Support Vectors* which correspond to the centers in input space.

The goal of the present study is to compare real–world results obtained with k–means clustering and classical RBF training to those obtained with the centers, weights and threshold automatically chosen by the Support Vector algorithm. To this end, we decided to undertake a performance study combining expertise on the Support Vector algorithm (AT&T Bell Laboratories) and classical radial basis function networks (Massachusetts Institute of Technology). We report results obtained on a US postal service database of handwritten digits.

We have organized the material as follows. In the next Section, we describe the algorithms used to train the different types of RBF classifiers used in this paper. Following that, we present an experimental comparison of the approaches. We conclude with a discussion of our findings.

2 Different Ways of Constructing a Radial Basis Function Classifier

We describe three radial basis function systems, trained in different ways. In Sec. 2.1, we discuss the first system trained along more classical lines. In the following section (2.2), we discuss the Support Vector algorithm, which constructs an RBF network whose parameters (centers, weights, threshold) are automatically optimized. In Sec. 2.3, finally, we use the Support Vector algorithm merely to choose the centers of the RBF network and then optimize the weights separately.

2.1 Classical Spherical Gaussian RBFs:

We begin by first describing the classical Gaussian RBF system. A d-dimensional spherical Gaussian RBF network with K centers has the mathematical form

$$
\begin{aligned}
g(\vec{x}) &= \sum_{i=1}^{K} w_i \mathcal{G}_i(\vec{x}) + b \\
&= \sum_{i=1}^{K} w_i \frac{1}{(2\pi)^{d/2}\sigma_i^d} \exp\left(-\frac{\|\vec{x}-\vec{c_i}\|^2}{2\sigma_i^2}\right) + b
\end{aligned}
$$

where $\mathcal{G}_i$ is the i^{th} Gaussian basis function with center $\vec{c_i}$ and variance σ_i^2. The weight coefficients w_i combine the Gaussian terms into a single output value and b is a bias term. In general, building a Gaussian RBF network for a given learning task involves (1) determining the total number of Gaussian basis functions to use for each output class and for the entire system, (2) locating the Gaussian basis function centers, (3) computing the cluster variance for each Gaussian basis function, and (4) solving for the weight coefficients and bias in the summation term. One can implement a binary pattern classifier on input vectors $\vec{x}$ as a Gaussian RBF network by defining an appropriate output threshold that separates the two pattern classes.

In this first system, we implement each individual digit recognizer as a spherical Gaussian RBF network, trained with a classical RBF algorithm. Given a specified number of Gaussian basis functions for each digit class, the algorithm separately computes the Gaussian centers and variances for each of the 10 digit classes to form the system's RBF kernels. The algorithm then solves for an optimal set of weight parameters between the RBF kernels and each output node to perform the desired digit recognition task. The training process constructs all 10 digit recognizers in parallel so one can reuse the same Gaussian basis functions among the 10 digit recognizers. To avoid overfitting the available training data with an overly complex RBF classifier connected to every Gaussian kernel, we use a "bootstrap" like operation that selectively connects each recognizer's output node to only a "relevant" subset of all basis functions. The idea is similar to how we choose relevant "near-miss" clusters for each individual digit recognizer in the original system. The training procedure proceeds as follows (for further details, see Sung, 1996):

1. The first training task is to determine an appropriate number k of Gaussian kernels for each digit class. This information is needed to initialize our clustering procedure for computing Gaussian RBF kernels. We opted for using the same numbers of Gaussian kernels as the ones automatically computed by the SV algorithm (see Table 1).
2. Our next task is to actually compute the Gaussian kernels for each digit class. We do this by separately performing classical k–means clustering (see e.g. Lloyd, 1982) on each digit class in the US postal service (USPS) training database. Each clustering operation returns a set of Gaussian centroids and their respective variances for the given digit class. Together, the Gaussian clusters from all 10 digit classes form the system's RBF kernels.
3. For each single-digit recognizer, we build an *initial* RBF network using only Gaussian kernels from its target class, using error backpropagation to train the weights. We then separately collect all the false positive mistakes each initial digit recognizer makes on the USPS training database.
4. In the final training step, we augment each initial digit recognizer with additional Gaussian kernels from outside its target class to help reduce misclassification errors. We determine which Gaussian kernels are "relevant" for each recognizer as follows: For each *false positive* mistake the initial recognizer makes during the previous step, we look up the misclassified pattern's actual digit class and include the nearest Gaussian kernel from its class in the "relevant" set. The *final* RBF network for each single-digit recognizer thus contains every Gaussian kernel from its target class, and several "relevant" kernels from the other 9 digit classes, trained by error backpropagation. Because our final digit recognizers have fewer weight parameters than a naive system that fully connects all 10 recognizers to every Gaussian kernel, we expect our system to generalize better on new data.

2.2 The Support Vector Machine

Structural Risk Minimization. For the case of two–class pattern recognition, the task of *learning from examples* can be formulated in the following way: given a set of functions

$$\{f_\alpha : \alpha \in \Lambda\}, \quad f_\alpha : \mathbf{R}^N \to \{-1, +1\}$$

and a set of examples

$$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell), \quad \mathbf{x}_i \in \mathbf{R}^N, y_i \in \{-1, +1\},$$

each one generated from an unknown probability distribution $P(\mathbf{x}, y)$, we want to find a function f_{α^*} which provides the smallest possible value for the risk

$$R(\alpha) = \int |f_\alpha(\mathbf{x}) - y| \, dP(\mathbf{x}, y).$$

The problem is that $R(\alpha)$ is unknown, since $P(\mathbf{x}, y)$ is unknown. Therefore an *induction principle* for risk minimization is necessary.

The straightforward approach to minimize the *empirical risk*

$$R_{emp}(\alpha) = \frac{1}{\ell} \sum_{i=1}^{\ell} |f_\alpha(\mathbf{x}_i) - y_i|$$

turns out not to guarantee a small actual risk (i.e. a small error on the training set does not imply a small error on a test set), if the number ℓ of training examples is limited. To make the most out of a limited amount of data, novel statistical techniques have been developed during the last 25 years. The *Structural Risk Minimization* principle (Vapnik, 1979) is based on the fact that for the above learning problem, for any $\alpha \in \Lambda$ with a probability of at least $1-\eta$, the bound

$$R(\alpha) \leq R_{emp}(\alpha) + \Phi(\frac{h}{\ell}, \frac{\log(\eta)}{\ell}) \tag{2}$$

holds, Φ being defined as

$$\Phi(\frac{h}{\ell}, \frac{\log(\eta)}{\ell}) = \sqrt{\frac{h\left(\log\frac{2\ell}{h}+1\right) - \log(\eta/4)}{\ell}}.$$

The parameter h is called the VC–dimension of a set of functions. It describes the capacity of a set of functions implementable by the learning machine. For binary classification, h is the maximal number of points k which can be separated into two classes in all possible 2^k ways by using functions of the learning machine; i.e. for each possible separation there exists a function which takes the value 1 on one class and -1 on the other class.

According to (2), given a fixed number ℓ of training examples one can control the risk by controlling two quantities: $R_{emp}(\alpha)$ and $h(\{f_\alpha : \alpha \in \Lambda'\})$; Λ' denoting some subset of the index set Λ. The empirical risk depends on the *function* chosen by the learning machine (i.e. on α), and it can be controlled by picking the right α. The VC–dimension h depends on the *set of functions* $\{f_\alpha : \alpha \in \Lambda'\}$ which the learning machine can implement. To control h, one introduces a structure of nested subsets $S_n := \{f_\alpha : \alpha \in \Lambda_n\}$ of $\{f_\alpha : \alpha \in \Lambda\}$,

$$S_1 \subset S_2 \subset \ldots \subset S_n \subset \ldots, \tag{3}$$

with the corresponding VC–dimensions satisfying

$$h_1 \leq h_2 \leq \ldots \leq h_n \leq \ldots$$

For a given set of observations $(\mathbf{x}_1, y_1), ..., (\mathbf{x}_\ell, y_\ell)$ the *Structural Risk Minimization principle* chooses the function $f_{\alpha_\ell^n}$ in the subset $\{f_\alpha : \alpha \in \Lambda_n\}$ for which the guaranteed risk bound (the right hand side of (2)) is minimal.

The remainder of this section follows Schölkopf, Burges & Vapnik (1995) in briefly reviewing the Support Vector algorithm. For details, the reader is referred to (Vapnik, 1995).

A Structure on the Set of Hyperplanes. Each particular choice of a structure (3) gives rise to a learning algorithm. The Support Vector algorithm is based on a structure on the set of hyperplanes. To describe it, first note that given a dot product space $\mathbf{Z}$ and a set of vectors $\mathbf{x}_1, \ldots, \mathbf{x}_r \in \mathbf{Z}$, each hyperplane $\{\mathbf{x} \in \mathbf{Z} : (\mathbf{w} \cdot \mathbf{x}) + b = 0\}$ corresponds to a canonical pair $(\mathbf{w}, b) \in \mathbf{Z} \times \mathbf{R}$ if we additionally require

$$\min_{i=1,\ldots,r} |(\mathbf{w} \cdot \mathbf{x}_i) + b| = 1. \tag{4}$$

Let $B_{\mathbf{x}_1,\ldots,\mathbf{x}_r} = \{\mathbf{x} \in \mathbf{Z} : ||\mathbf{x} - \mathbf{a}|| < R\}$ $(\mathbf{a} \in \mathbf{Z})$ be the smallest ball containing the points $\mathbf{x}_1, \ldots, \mathbf{x}_r$, and

$$f_{\mathbf{w},b} = \operatorname{sgn}((\mathbf{w} \cdot \mathbf{x}) + b) \tag{5}$$

the decision function defined on these points. The possibility of introducing a structure on the set of hyperplanes is based on the result (Vapnik, 1995) that the set $\{f_{\mathbf{w},b} : ||\mathbf{w}|| \leq A\}$ has a VC-dimension h satisfying

$$h \leq R^2 A^2. \tag{6}$$

Note. Dropping the condition $||\mathbf{w}|| \leq A$ leads to a set of functions whose VC–dimension equals $N+1$, where N is the dimensionality of $\mathbf{Z}$. Due to $||\mathbf{w}|| \leq A$, we can get VC–dimensions which are much smaller than N, enabling us to work in very high dimensional spaces.

The Support Vector Algorithm. Now suppose we want to find a decision function $f_{\mathbf{w},b}$ with the property $f_{\mathbf{w},b}(\mathbf{x}_i) = y_i$, $i = 1, \ldots, \ell$. If this function exists, canonicality (4) implies

$$y_i((\mathbf{w} \cdot \mathbf{x}_i) + b) \geq 1, \quad i = 1, \ldots, \ell. \tag{7}$$

In many practical applications, a separating hyperplane does not exist. To allow for the possibility of examples violating (7), Cortes & Vapnik (1995) introduce slack variables

$$\xi_i \geq 0, \quad i = 1, \ldots, \ell, \tag{8}$$

to get

$$y_i((\mathbf{w} \cdot \mathbf{x}_i) + b) \geq 1 - \xi_i, \quad i = 1, \ldots, \ell. \tag{9}$$

The Support Vector approach to minimizing the guaranteed risk bound (2) consists in the following: minimize

$$\Phi(\mathbf{w}, \xi) = (\mathbf{w} \cdot \mathbf{w}) + \gamma \sum_{i=1}^{\ell} \xi_i \tag{10}$$

subject to the constraints (8) and (9). According to (6), minimizing the first term amounts to minimizing the VC–dimension of the learning machine, thereby minimizing the second term of the bound (2). The term $\sum_{i=1}^{\ell} \xi_i$, on the other hand, is an upper bound on the number of misclassifications on the training set — this controls the empirical risk term in (2). For a suitable positive constant γ, this approach therefore constitutes a practical implementation of Structural Risk Minimization on the given set of functions.

Introducing Lagrange multipliers α_i and using the Kuhn–Tucker theorem of optimization theory, the solution can be shown to have an expansion

$$\mathbf{w} = \sum_{i=1}^{\ell} y_i \alpha_i \mathbf{x}_i, \tag{11}$$

with nonzero coefficients α_i only for the cases where the corresponding example $(\mathbf{x}_i, y_i)$ precisely meets the constraint (9). These $\mathbf{x}_i$ are called *Support Vectors.* All the remaining examples $\mathbf{x}_i$ of the training set are irrelevant: their constraint (9) is satisfied automatically (with $\xi_i = 0$), and they do not appear in the expansion (11). The coefficients α_i are found by solving the following quadratic programming problem: maximize

$$W(\alpha) = \sum_{i=1}^{\ell} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{\ell} y_i y_j \alpha_i \alpha_j (\mathbf{x}_i \cdot \mathbf{x}_j) \tag{12}$$

subject to

$$0 \leq \alpha_i \leq \gamma, \quad i = 1, \ldots, \ell, \quad \text{and} \quad \sum_{i=1}^{\ell} \alpha_i y_i = 0. \tag{13}$$

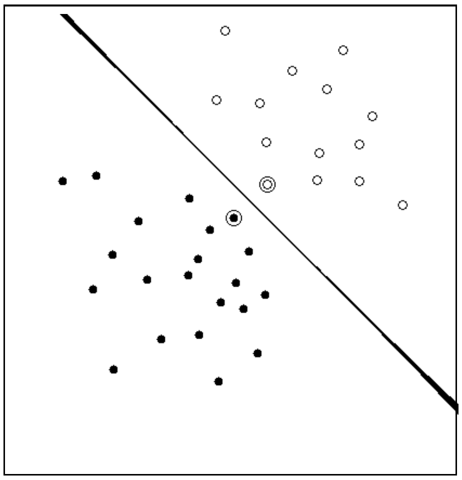

Figure 3: A simple two–class classification problem as solved by the Support Vector algorithm ($c_i = 1$ for all i; cf. Eq. 1). Note that the RBF centers (indicated by extra circles) are closest to the decision boundary.

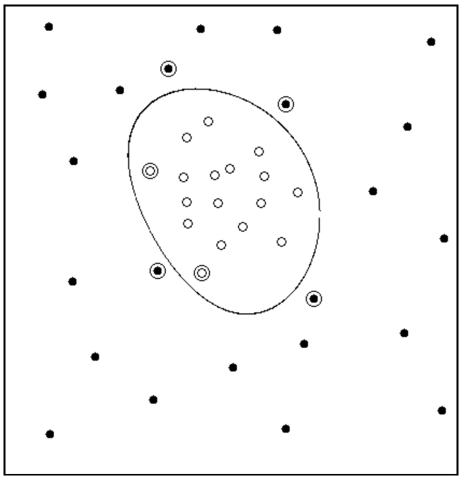

Figure 4: Two–class classification problem solved by the Support Vector algorithm ($c_i = 1$ for all i; cf. Eq. 1).

By linearity of the dot product, the decision function (5) can thus be written as

$$f(\mathbf{x}) = \operatorname{sgn}\left(\sum_{i=1}^{\ell} y_i\alpha_i \cdot (\mathbf{x}\cdot\mathbf{x}_i) + b\right).$$

So far, we have described linear decision surfaces. To allow for much more general decision surfaces, one can first nonlinearly transform the input vectors into a high–dimensional feature space by a map ϕ and then do a linear separation there. Maximizing (12) then requires the computation of dot products $(\phi(\mathbf{x})\cdot\phi(\mathbf{x}_i))$ in a high–dimensional space. In some cases, these expensive calculations can be reduced significantly by using a suitable function K such that

$$(\phi(\mathbf{x})\cdot\phi(\mathbf{x}_i)) = K(\mathbf{x},\mathbf{x}_i).$$

We thus get decision functions of the form

$$f(\mathbf{x}) = \operatorname{sgn}\left(\sum_{i=1}^{\ell} y_i\alpha_i \cdot K(\mathbf{x},\mathbf{x}_i) + b\right). \qquad (14)$$

In practise, we need not worry about conceiving the map ϕ. We will choose a K which is the Kernel of a positive Hilbert–Schmidt operator, and Mercer's theorem of functional analysis then tells us that K corresponds to a dot product in some other space (see Boser, Guyon & Vapnik, 1992). Consequently, everything that has been said above about the linear case also applies to nonlinear cases obtained by using a suitable kernel K instead of the Euclidean dot product. We are now in a position to explain how the Support Vector algorithm can construct radial basis function classifiers: we simply use

$$K(\mathbf{x},\mathbf{x}_i) = \exp\left(-\|\mathbf{x}-\mathbf{x}_i\|^2/c\right) \qquad (15)$$

(see Aizerman, Braverman & Rozonoer, 1964). Other possible choices of K include

$$K(\mathbf{x},\mathbf{x}_i) = (\mathbf{x}\cdot\mathbf{x}_i)^d,$$

yielding polynomial classifiers ($d \in \mathbf{N}$), and

$$K(\mathbf{x},\mathbf{x}_i) = \tanh(\kappa\cdot(\mathbf{x}\cdot\mathbf{x}_i) + \Theta)$$

for constructing neural networks.

Interestingly, these different types of SV machines use largely the same Support Vectors; i.e. most of the centers of an SV machine with Gaussian kernel coincide with the weights of the polynomial and neural network SV classifiers (Schölkopf, Burges & Vapnik 1995).

To find the decision function (14), we have to maximize

$$W(\alpha) = \sum_{i=1}^{\ell}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{\ell} y_i y_j \alpha_i\alpha_j K(\mathbf{x}_i,\mathbf{x}_j) \qquad (16)$$

under the constraint (13). To find the threshold b, one takes into account that due to (9), for Support Vectors $\mathbf{x}_j$ for which $\xi_j = 0$ we have

$$\sum_{i=1}^{\ell} y_i\alpha_i \cdot K(\mathbf{x}_j,\mathbf{x}_i) + b = y_j.$$

Finally, we note that the Support Vector algorithm has been empirically shown to exhibit good generalization ability (Cortes & Vapnik, 1995). This can be further improved by incorporating invariances of a problem at hand, as with the Virtual Support Vector method of generating artificial examples from the Support Vectors (Schölkopf, Burges, & Vapnik, 1996). In addition, the decision rule (14), which requires the computation of dot products between the test example and all Support Vectors, can be sped up with the reduced set technique (Burges, 1996). These methods have led to substantial improvements for polynomial Support Vector machines (Burges & Schölkopf, 1996), and they are directly applicable also to RBF Support Vector machines.

2.3 A Hybrid System: SV Centers Only

The previous section discusses how one can train RBF like networks using the Support Vector algorithm. This

Digit Class	0	1	2	3	4	5	6	7	8	9
# of SVs	274	104	377	361	334	388	236	235	342	263
# of pos. SVs	172	77	217	179	211	231	147	133	194	166

Table 1: Numbers of centers (Support Vectors) automatically extracted by the Support Vector machine. The first row gives the total number for each binary classifier, including both positive and negative examples; in the second row, we only counted the positive SVs. The latter number was used in the initialization of the k–means algorithm, cf. Sec. 2.1.

digit	0	1	2	3	4	5	6	7	8	9
classical RBF	20	16	43	38	46	31	15	18	37	26
full SVM	16	8	25	19	29	23	14	12	25	16
SV centers only	9	12	27	24	32	24	19	16	26	16

Table 2: Two–class-classification: numbers of test errors (out of 2007 test patterns) for the three systems described in Sections 2.1 – 2.3.

involves the choice of an appropriate kernel function K and solving the optimization problem in the form of Eq. (16). The Support Vector algorithm thus automatically determines the centers (which are the Support Vectors), the weights (given by $y_i \alpha_i$), and the threshold b for the RBF machine.

To assess the relative influence of the automatic SV center choice and the SV weight optimization, respectively, we built another RBF system, constructed with centers that are simply the Support Vectors arising from the SV optimization, and with the weights trained separately.

3 Experimental Results

Toy examples. What are the Support Vectors? They are elements of the data set that are "important" in separating the two classes from each other. In general, the Support Vectors with zero slack variables (see Eq. 8) lie on the boundary of the decision surface, as they precisely satisfy the inequality (9) in the high–dimensional space. Figures 3 and 4 illustrate that for the used Gaussian kernel this is also the case in input space.

This raises an interesting question from the point of view of interpreting the structure of trained RBF networks. The traditional view of RBF networks has been one where the centers were regarded as "templates" or stereotypical patterns. It is this point of view that leads to the clustering heuristic for training RBF networks. In contrast, the Support Vector machine posits an alternate point of view, with the centers being those examples which are critical for a given classification task.

US Postal Service Database. We used the USPS database of 9300 handwritten digits (7300 for training, 2000 for testing), collected from mail envelopes in Buffalo (cf. LeCun et al., 1989). Each digit is a 16×16 vector with entries between -1 and 1. Preprocessing consisted in smoothing with a Gaussian kernel of width $\sigma = 0.75$. The Support Vector machine results reported in the following were obtained with $\gamma = 10$ (cf. (10)) and $c = 0.3 \cdot 16 \cdot 16$ (cf. (15)).[1] In all experiments, we used the Support Vector algorithm with standard quadratic programming techniques (conjugate gradient descent).

Two–class classification. Table 1 shows the numbers of Support Vectors, i.e. RBF centers, extracted by the SV algorithm. Table 2 gives the results of binary classifiers separating single digits from the rest, for the systems described in Sections 2.1, 2.2, and 2.3.

Ten–class classification. For each test pattern, the arbitration procedure in all three systems simply returns the digit class whose recognizer gives the strongest response.[2] Table 3 shows the 10-class digit recognition error rates for our original system and the two RBF-based systems.

The fully automatic Support Vector machine exhibits the highest test accuracy. Using the Support Vector algorithm to choose an appropriate number and corresponding centers for the RBF network is also better than the baseline procedure of choosing the centers by a clustering heuristic. It can be seen that in contrast to the k–means cluster centers, the centers chosen by the Support Vector algorithm allow zero training error rates.

4 Summary and Discussion

The Support Vector algorithm provides a principled way of choosing the number and the locations of RBF centers. Our experiments on a real–world pattern recognition problem have shown that *compared to a corresponding number of centers chosen by k–means, the centers chosen by the Support Vector algorithm allowed a training error of zero*, even if the weights were trained by classical RBF methods. Our interpretation of this finding is that the Support Vector centers are specifically

[1] The SV machine is rather insensitive to different choices of c: for all values in $0.1, 0.2, \ldots, 1.0$, the performance is about the same (in the area of $4\% - 4.5\%$).

[2] In the Support Vector case, we constructed ten two–class classifiers, each trained to separate a given digit from the other nine, and combined them by doing the ten–class classification according to the maximal output (before applying the sgn function) among the two–class classifiers.

USPS Database	Classification Error Rate		
	Clustered Centers	S.V. centers	Full S.V.M.
Training (7291 patterns)	1.7%	0.0%	0.0%
Test (2007 patterns)	6.7%	4.9%	4.2%

Table 3: 10-class digit recognition error rates for three RBF classifiers constructed with different algorithms. The first system is a more classical one choosing its centers by a clustering heuristic. The other two are the Gaussian RBF-based systems we trained, one with the Support Vectors were chosen to be the centers and the second where the entire network was trained using the Support Vector algorithm.

chosen for the classification task at hand, whereas k–means does not care about picking those centers which will make a problem separable.

In addition, *the SV centers yielded lower test error rates than k–means.* It is interesting to note that using SV centers, while sticking to the classical procedure for training the weights, improved training and test error rates by approximately the same margin (2 per cent). In view of the guaranteed risk bound (2), this can be understood in the following way: the improvement in test error (risk) was solely due to the lower value of the training error (empirical risk); the confidence term (the second term on the right hand side of (2)), depending on the VC–dimension and thus on the norm of the weight vector (Eq. 6), did not change, as we stuck to the classical weight training procedure. However, when we also trained the weights with the Support Vector algorithm, we minimized the norm of the weight vector (see Eq. 10) and thus the confidence term, while still keeping the training error zero. Thus, consistent with (2), *the Support Vector machine achieved the highest test accuracy of the three systems.*

References

[1] Aizerman, M.; Braverman, E.; and Rozonoer, L. 1964. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, **25**:821-837.

[2] Boser, B. E.; Guyon, I. M.; and Vapnik, V. 1992. A training algorithm for optimal margin classifiers. *Fifth Annual Workshop on Computational Learning Theory*, Pittsburgh ACM 144–152.

[3] Burges, C. J. C. 1996. Simplified Support Vector Decision Rules. 13th International Conference on Machine Learning.

[4] Burges, C.J.C., and Schölkopf, B. 1996. Improving the Accuracy and Speed of Support Vector Machines. To appear in: Advances in Neural Information Processing Systems 9.

[5] Cortes, C.; and Vapnik, V. 1995. Support Vector Networks. *Machine Learning*, **20**:273 – 297.

[6] Le Cun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., Jackel, L. J. 1989. Backpropagation applied to handwritten zip code recognition. *Neural Computation* **1**: 541 – 551.

[7] Lloyd, S. P. 1982. Least squares quantization in PCM. *IEEE Trans. Information Theory* IT-28:129 – 137.

[8] Poggio, T., & Girosi, F. 1990. Networks for approximation and learning. Proc. IEEE, **78**:1481 – 1497.

[9] Rumelhart, D. E., Hinton, G. E., & Williams, R. J. 1986. Learning representations by back–propagating errors. *Nature*, **323**:533 – 536.

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

[11] Schölkopf, B., Burges, C.J.C., Vapnik, V. 1996. Incorporating Invariances in Support Vector Learning Machines. In C. von der Malsburg, W. von Seelen, J. C. Vorbrüggen, and B. Sendhoff, editors, *Artificial Neural Networks — ICANN'96*, pages 47 – 52, Berlin. (Springer Lecture Notes in Computer Science, Vol. 1112.)

[12] Sung, K. 1995. Learning and Example Selection for Object and Pattern Detection. Ph.D. Thesis, Massachusetts Institute of Technology.

[13] Vapnik, V. 1979. *Estimation of Dependences Based on Empirical Data*, [in Russian] Nauka, Moscow; English translation: Springer–Verlag, New York, 1982.

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