Extracting Support Data for a Given Task

Bernhard Schölkopf[*] **Chris Burges**[†] **Vladimir Vapnik**

AT&T Bell Laboratories
101 Crawfords Corner Road
Holmdel NJ 07733
USA
schoel@big.att.com cjcb@big.att.com vlad@big.att.com

Abstract

We report a novel possibility for extracting a small subset of a data base which contains all the information necessary to solve a given classification task: using the Support Vector Algorithm to train three different types of handwritten digit classifiers, we observed that these types of classifiers construct their decision surface from strongly overlapping small ($\approx 4\%$) subsets of the data base. This finding opens up the possibility of compressing data bases significantly by disposing of the data which is not important for the solution of a given task.

In addition, we show that the theory allows us to predict the classifier that will have the best generalization ability, based solely on performance on the training set and characteristics of the learning machines. This finding is important for cases where the amount of available data is limited.

Introduction

Learning can be viewed as inferring regularities from a set of training examples. Much research has been devoted to the study of various learning algorithms which allow the extraction of these underlying regularities. No matter how different the outward appearance of these algorithms is, they all must rely on intrinsic regularities of the data. If the learning has been successful, these intrinsic regularities will be captured in the values of some parameters of a learning machine; for a polynomial classifier, these parameters will be the coefficients of a polynomial, for a neural net they will be the weights and biases, and for a radial basis function classifier they will be weights and centers. This variety of different representations of the intrinsic regularities, however, conceals the fact that they all stem from a common root.

In the present study, we explore the *Support Vector Algorithm*, an algorithm which gives rise to a number of different types of pattern classifiers. We show that the algorithm allows us to construct different classifiers (polynomial classifiers, radial basis function classifiers, and neural networks) exhibiting similar performance and relying on almost identical subsets of the training set, their *support vector sets*. In this sense, the support vector set is a stable characteristic of the data.

In the case where the available training data is limited, it is important to have a means for achieving the best possible generalization by controlling characteristics of the learning machine. We use a bound of statistical learning theory (Vapnik, 1995) to predict the degree which yields the best generalization for polynomial classifiers.

In the next Section, we follow Vapnik (1995), Boser, Guyon & Vapnik (1992), and Cortes & Vapnik (1995) in briefly recapitulating this algorithm and the idea of Structural Risk Minimization that it is based on. Following that, we will present experimental results obtained with support vector machines.

[*]permanent address: Max-Planck-Institut für biologische Kybernetik, Spemannstraße 38, 72076 Tübingen, Germany

[†]supported by ARPA under ONR contract number N00014-94-C-0186

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

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

(the index set Λ not necessarily being a subset of $\mathbf{R}^n$) 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 is such a technique. It 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

$$R(\alpha) \le R_{emp}(\alpha) + \Phi(\frac{h}{\ell}, \frac{\log(\eta)}{\ell}) \tag{1}$$

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 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 (1), 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 $\{f_\alpha : \alpha \in \Lambda\}$,

$$S_1 \subset S_2 \subset \ldots \subset S_n \subset \ldots, \tag{2}$$

with the corresponding VC–dimensions satisfying

$$h_1 \le h_2 \le \ldots \le h_n \le \ldots$$

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 (1)) is minimal.

The Support Vector Algorithm

A Structure on the Set of Hyperplanes. Each particular choice of a structure (2) 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 uniquely to a pair $(\mathbf{w}, b) \in \mathbf{Z} \times \mathbf{R}$ if we additionally require

$$\min_{i=1,\ldots,r} |(\mathbf{w} \cdot \mathbf{x}_i) + b| = 1. \tag{3}$$

Each hyperplane corresponds to a decision function constructed in the following way: we first find the smallest ball $B_{\mathbf{x}_1,\ldots,\mathbf{x}_r} = \{\mathbf{x} \in \mathbf{Z} : \|\mathbf{x} - \mathbf{a}\| < R\}$ ($\mathbf{a} \in \mathbf{Z}$) containing the points $\mathbf{x}_1, \ldots, \mathbf{x}_r$. Then we define a decision function $f_{\mathbf{w},b}$ by

$$f_{\mathbf{w},b} : B_{\mathbf{x}_1,\ldots,\mathbf{x}_r} \to \{\pm 1\},$$
$$f_{\mathbf{w},b} = \operatorname{sgn}((\mathbf{w} \cdot \mathbf{x}) + b). \tag{4}$$

The possibility of introducing a structure on the set of hyperplanes is based on the fact (Vapnik, 1995) that the set $\{f_{\mathbf{w},b} : \|\mathbf{w}\| \le A\}$ has a VC-dimension h satisfying

$$h \le R^2 A^2. \tag{5}$$

Note. Dropping the condition $\|\mathbf{w}\| \le A$ leads to a set of functions whose VC–dimension equals $N+1$, where N is the dimensionality of $\mathbf{Z}$. Due to $\|\mathbf{w}\| \le A$, 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 are given a set of examples $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell)$, $\mathbf{x}_i \in \mathbf{R}^N, y_i \in \{-1, +1\}$, and 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 (3) implies

$$y_i((\mathbf{w} \cdot \mathbf{x}_i) + b) \ge 1, \quad i = 1, \ldots, \ell. \tag{6}$$

In many practical applications, a separating hyperplane does not exist. To allow for the possibility of examples violating (6), Cortes & Vapnik (1995) introduce slack variables

$$\xi_i \ge 0, \quad i = 1, \ldots, \ell, \tag{7}$$

to get

$$y_i((\mathbf{w} \cdot \mathbf{x}_i) + b) \ge 1 - \xi_i, \quad i = 1, \ldots, \ell. \tag{8}$$

The support vector approach to minimizing the guaranteed risk bound (1) consists in the following: minimize

$$\Phi(\mathbf{w}, \xi) = (\mathbf{w} \cdot \mathbf{w}) + \gamma \sum_{i=1}^{\ell} \xi_i \tag{9}$$

subject to the constraints (7) and (8). According to (5), minimizing the first term amounts to minimizing the VC–dimension of the learning machine, thereby minimizing the second term of the bound (1). 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 (1). 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 one can show that the solution has an expansion

$$\mathbf{w} = \sum_{i=1}^{\ell} y_i \alpha_i \mathbf{x}_i, \qquad (10)$$

with nonzero coefficients α_i only for the cases where the corresponding example $(\mathbf{x}_i, y_i)$ precisely meets the constraint (8). These $\mathbf{x}_i$ are called *Support Vectors*, and (10) is the *Support Vector Expansion*. All the remaining examples $\mathbf{x}_j$ of the training set are irrelevant: their constraint (8) is satisfied automatically (with $\xi_j = 0$), and they do not appear in the support vector expansion. Although the solution $\mathbf{w}$ is unique, the coefficients α_i are not. They can be found by solving the following quadratic programming problem: maximize

$$W(\alpha) = \sum_{i=1}^{\ell} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{\ell} y_i y_j \alpha_i \alpha_j (\mathbf{x}_i \cdot \mathbf{x}_j) \qquad (11)$$

subject to

$$0 \le \alpha_i \le \gamma, \quad i = 1, \ldots, \ell, \quad \text{and} \quad \sum_{i=1}^{\ell} \alpha_i y_i = 0. \qquad (12)$$

By linearity of the dot product, the decision function (4) can thus be written as

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

So far, we have described linear decision surfaces. These are not appropriate for all tasks. 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 (11) then requires the computation of dot products $(\phi(\mathbf{x}) \cdot \phi(\mathbf{x}_i))$ in a high-dimensional space. Under certain conditions (Boser, Guyon & Vapnik, 1992), these expensive calculations can be reduced significantly by using a 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

$$f(\mathbf{x}) = \mathrm{sgn}\left(\sum_{i=1}^{\ell} y_i \alpha_i \cdot K(\mathbf{x}, \mathbf{x}_i) + b\right). \qquad (13)$$

Experimental Results

In our experiments, we used the support vector algorithm with standard quadratic programming techniques to construct three different types of classifiers. This was done by choosing three different functions K in the decision function (13) and in the function to be maximized under the constraint (12),

$$W(\alpha) = \sum_{i=1}^{\ell} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{\ell} y_i y_j \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j). \qquad (14)$$

- Homogeneous polynomial classifiers:
  $K(\mathbf{x}, \mathbf{x}_i) = (\mathbf{x} \cdot \mathbf{x}_i)^{\text{degree}}$
- Radial Basis Function (RBF) classifiers:
  $K(\mathbf{x}, \mathbf{x}_i) = \exp\left(-\|\mathbf{x} - \mathbf{x}_i\|^2/\sigma^2\right)$
- Neural networks:
  $K(\mathbf{x}, \mathbf{x}_i) = \tanh(\kappa \cdot (\mathbf{x} \cdot \mathbf{x}_i) - \Theta)$

All the results reported in this paper were obtained with $\gamma = 10$ (cf. (9)). We used a US postal database of 9300 handwritten digits (7300 for training, 2000 for testing) (cf. LeCun et al., 1990). Each digit is a 16×16 vector with entries between -1 and 1. Preprocessing consisted in smoothing with a Gaussian kernel of width $\sigma = 0.75$.

To get the ten-class classifiers for the digit recognition problem, 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.

Performance for Various Types of Classifiers

The results for the three different functions K are summarized in Table 1. They should be compared with values achieved on the same database with a five-layer neural net (LeCun et al., 1990), 5.1%, a two-layer neural net, 5.9%, and the human performance, 2.5% (Bromley & Säckinger, 1991).

Table 1: Performance for three different types of classifiers, constructed with the support vector algorithm by choosing different functions K in (13) and (14). Given are raw errors (i.e. no rejections allowed) on the test set. The normalization factor $c = 1.04$ in the sigmoid case is chosen such that $c \cdot \tanh(2) = 1$. For each of the ten-class-classifiers, we also show the average number of support vectors of the ten two-class-classifiers.

polynomial: $K(\mathbf{x}, \mathbf{y}) = ((\mathbf{x} \cdot \mathbf{y})/256)^{\text{degree}}$

degree	1	2	3	4	5	6
raw error/%	8.9	4.7	4.0	4.2	4.5	4.5
av. # of SVs	282	237	274	321	374	422

RBF: $K(\mathbf{x}, \mathbf{y}) = \exp\left(-\|\mathbf{x} - \mathbf{y}\|^2/(256\,\sigma^2)\right)$

σ^2		1.0	0.8	0.5	0.2	0.1
raw error/%		4.7	4.3	4.4	4.4	4.5
av. # of SVs		234	235	251	366	722

sigmoid: $K(\mathbf{x}, \mathbf{y}) = 1.04 \tanh(2(\mathbf{x} \cdot \mathbf{y})/256 - \Theta)$

Θ		0.9	1.0	1.2	1.3	1.4
raw error/%		4.8	4.1	4.3	4.4	4.8
av. # of SVs		242	254	278	289	296

The similar performance for the three different functions K suggests that the choice of the set of decision

Table 2: Performance of the classifiers with degree predicted by the VC–bound. Each row describes one two–class–classifier separating one digit (stated in the first column) from the rest. The remaining columns contain: degree: the degree of the best polynomial as predicted by the described procedure, parameters: the dimensionality of the high dimensional space, which is also the maximum possible VC–dimension for linear classifiers in that space, $h_{estim.}$: the VC dimension estimate for the actual classifiers, which is much smaller than the number of free parameters of linear classifiers in that space, 1 – 7: the numbers of errors on the test set for polynomial classifiers of degrees 1 through 7. The table shows that the decribed procedure chooses polynomial degrees which are optimal or close to optimal.

	chosen classifier			errors on the test set for degrees 1 – 7						
digit	degree	parameters	$h_{estim.}$	1	2	3	4	5	6	7
0	3	$2.8 \cdot 10^6$	547	36	14	$\boxed{11}$	11	11	12	17
1	7	$1.5 \cdot 10^{13}$	95	17	15	14	11	10	9	$\boxed{10}$
2	3	$2.8 \cdot 10^6$	832	53	32	$\boxed{28}$	26	28	27	32
3	3	$2.8 \cdot 10^6$	1130	57	25	$\boxed{22}$	22	22	22	23
4	4	$1.8 \cdot 10^8$	977	50	32	32	$\boxed{30}$	30	29	33
5	3	$2.8 \cdot 10^6$	1117	37	20	$\boxed{22}$	24	24	26	28
6	4	$1.8 \cdot 10^8$	615	23	12	12	$\boxed{15}$	17	17	19
7	5	$9.5 \cdot 10^9$	526	25	15	12	10	$\boxed{11}$	13	14
8	4	$1.8 \cdot 10^8$	1466	71	33	28	$\boxed{24}$	28	32	34
9	5	$9.5 \cdot 10^9$	1210	51	18	15	11	$\boxed{11}$	12	15

functions is less important than capacity control in the chosen type of structure. This phenomenon is well–known for the Parzen window method for density estimation: there, the choice of the Kernel function is less critical than the choice of the appropriate value of the bandwidth parameter for a given amount of data.

Predicting the Optimal Decision Functions

Given a set of training data, for some choices of the nested set of decision functions (2) one can estimate the second term on the right hand side of Eq. (1). We will call the latter term the "VC-confidence". In these cases, since one can also measure the empirical risk $R_{emp}(\alpha)$, one can actually compute upper bounds on the expected risk $R(\alpha)$ (i.e. the expected error on the test set). We can try to use these bounds to predict which set of decision functions will give the best performance on test data, without the use of any validation or test sets. Clearly this could be very useful in situations where the amount of available data is very limited.

To test this idea we concentrated on the polynomial classifiers with $K(\mathbf{x}, \mathbf{y}) = (((\mathbf{x} \cdot \mathbf{y}) + 1)/256)^{\text{degree}}$. For these decision functions, the nested structure (2) is implemented by varying the degree of the polynomial. We estimated the VC dimension h by making the assumption that the bound (5) is met, that is,

$$h \approx h_{estim.} = R^2 \|\mathbf{w}\|^2.$$

In this case, the right hand side of Equation (1) is dominated by the VC-confidence, which is minimized when the VC-dimension is minimized. Now $\|\mathbf{w}\|$ is determined by the support vector algorithm; so in order to estimate h, we need to compute R.

Recall that R is the radius of the smallest sphere enclosing the training data in the high dimensional space (the space in which K corresponds to a dot product). We formulate the problem as follows: Minimize R^2 subject to $\|X_i - X^*\|^2 \leq R^2$ where X^* is the (to be determined) position vector of the center of the sphere. This is a well known quadratic programming problem. We use the objective function $R^2 - \sum_i \lambda_i (R^2 - (X_i - X^*)^2)$ and vary R and X^* to get $X^* = \sum_i \lambda_i X_i$ and the Wolfe dual problem: Maximize

$$\sum_i \lambda_i \cdot (X_i \cdot X_i) - \sum_{i,j} \lambda_i \lambda_j \cdot (X_i \cdot X_j)$$

subject to

$$\sum_i \lambda_i = 1, \ \lambda_i \geq 0,$$

where the λ_i are Lagrange multipliers. As in the support vector algorithm, this problem has the property

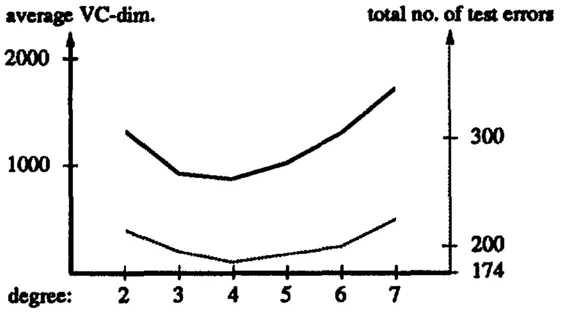

Figure 1: Average VC–dimension (solid) and total number of test errors of the ten two–class-classifiers (dotted) for polynomial degrees 2 through 7 (for degree 1, R_{emp} is comparably big, so the VC–dimension alone is not sufficient for predicting R, cf. (1)). The baseline on the error scale, 174, corresponds to the total number of test errors of the ten best two–class–classifiers out of the degrees 2 through 7. The graph shows that the VC–dimension allows us to predict that degree 4 yields the best overall performance of the two–class-classifiers on the test set. This is not necessarily the case for the performances of the ten–class-classifiers, which are built from the two–class-classifier outputs before applying the sgn functions. The fact that indeed our ten–class-classifier with degree 3 so far performs better that the one with degree 4 (see above) leads us to believe that the latter can be improved by using a more principled way of doing ten–class–classification.

that the X_i appear only in dot products, so as before one can compute the dot products in the high dimensional space by replacing $(X_i \cdot X_j)$ by $K(\mathbf{x}_i, \mathbf{x}_j)$ (where the $\mathbf{x}_i$ live in the low dimensional space, and the X_i in the high dimensional space), provided the K functions satisfy certain positivity conditions. In this way, we computed the radius of the minimal enclosing sphere for all the training data for polynomial classifiers of degrees 1 through 7.

We then trained a binary polynomial classifier for each digit, for degrees 1 through 7. Using the obtained values for our approximation to h, we can predict, for each digit, which degree polynomial will give the best generalization performance. We can then compare this prediction with the actual polynomial degree which gives the best performance on the test set. The results are shown in Table 2; cf. also Fig. 1. The above method for predicting the optimal classifier functions gives good results. In four cases the theory predicted the correct degree; in the other cases, the predicted degree gave performance close to the best possible one.

Comparison of the Support Vector Sets for Different Classifiers

In the remainder of the paper we shall use the optimal parameters according to Table 1 in studying the support vector sets for the three different types of classifiers. Table 3 shows that all three classifiers use around 250 support vectors per two–class–classifier (less than 4% of the training set). The total number of different support vectors of the ten–class-classifiers is around 1600. The reason why it is less than 2500 (ten times the above 250) is the following: one particular vector that has been used as a positive example (i.e. $y_i = +1$ in (13)) for digit 7, say, might be a good negative example ($y_i = -1$) for digit 1.

Table 3: First row: total number of different support vectors of three different ten–class-classifiers (i.e. number of elements of the union of the ten two–class-classifier support vector sets) obtained by choosing different functions K in (13) and (14); second row: average number of support vectors per two–class-classifier.

	Polynomial	RBF	Sigmoid
total # of SVs	1677	1498	1611
average # of SVs	274	235	254

Tables 4–6 show how many elements the support vector sets of the different classifiers have in common. As mentioned above, the support vector expansion (10) is not unique. Depending on the way the quadratic programming problem is solved, one can get different expansions and therefore different support vector sets. We used the same quadratic programming algorithm and the same ordering of the training set in all three cases.

Table 4: Percentage of the support vector set of [column] contained in the support set of [row]; ten–class classifiers.

	Polynomial	RBF	Sigmoid
Polynomial	100	93	94
RBF	83	100	87
Sigmoid	90	93	100

Table 5: Percentage of the support vector set of [column] contained in the support set of [row]; only one classifier (digit 7) each

	Polynomial	RBF	Sigmoid
Polynomial	100	84	93
RBF	89	100	92
Sigmoid	93	86	100

In describing the support vector algorithm, we have shown that the support vector set contains all the information a given classifier needs for constructing the decision function. Due to the overlap in the support vector sets of different classifiers, one can even train classifiers on support vector sets of *another* classifier. Table 7 shows that this leads to results comparable to those after training on the whole database.

Table 6: Comparison of all three support vector sets at a time. For each of the (ten–class) classifiers, % intersection gives the fraction of its support vector set shared with both the other two classifiers. Out of a total of 1834 different support vectors, 1355 are shared by all three classifiers; an additional 242 is common to two of the classifiers.

	Polynomial	RBF	Sigmoid	intersection	shared by 2	union
no. of supp. vectors	1677	1498	1611	1355	242	1834
% intersection	81	90	84	100	—	—

Table 7: Training classifiers on the support vector sets of other classifiers leads to performances on the test set which are as good as the results for training on the full data base (shown are numbers of errors on the 2000–element test set, for two–class classifiers separating digit 7 from the rest). Additionally, the results for training on a random subset of the data base of size 200 are displayed.

classifier	trained on	Poly–support	RBF–support	Sigmoid–support	full data base	random subset
	size	*178*	*189*	*177*	*7291*	*200*
Polynomial		13	13	12	13	23
RBF		17	13	17	15	27
sigmoid		15	13	13	15	25

Conclusion

We have shown that

- the support vector algorithm allows the construction of various learning machines, all of which are performing similarly well,
- we were able to predict the degree of polynomial classifiers which leads to high generalization ability, and
- for the considered classification problem (the US postal handwritten digit database) the three constructed learning machines (polynomial classifiers, neural nets, radial basis function classifiers) construct their decision functions from highly overlapping subsets of the training set.

Considering that in our experiments these *support vector sets* constituted less than 4% of the whole training set (for binary classification), the latter point indicates the possibility of a substantial data compression. Whether this holds true for various other real–life problems is an open and interesting question.

References

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.

Bromley, J.; and Säckinger, E. 1991. Neural–network and k–nearest–neighbor classifiers. Technical Report 11359–910819–16TM, AT&T.

Cortes, C.; and Vapnik, V. 1995. Support Vector Networks. To appear in *Machine Learning*.

Le Cun, Y.; Boser, B.; Denker, J. S.; Henderson, D.; Howard, R. E.; Hubbard, W.; and Jackel, L. J. 1990. Handwritten digit recognition with a back-propagation network. *Advances in Neural Information Processing Systems, Vol. 2*, 396-404, Morgan Kaufman.

Vapnik, V. 1995. *The Nature of Statistical Learning Theory.* Springer Verlag, forthcoming.