Journal of Machine Learning Research 5 (2004) 293-323 Submitted 3/03; Revised 12/03; Published 4/04

A Compression Approach to Support Vector Model Selection

Ulrike von Luxburg ULRIKE.LUXBURG@TUEBINGEN.MPG.DE
Olivier Bousquet OLIVIER.BOUSQUET@TUEBINGEN.MPG.DE
Bernhard Schölkopf BERNHARD.SCHOELKOPF@TUEBINGEN.MPG.DE
Max Planck Institute for Biological Cybernetics
Spemannstrasse 38
72076 Tübingen, Germany

Editor: John Shawe-Taylor

Abstract

In this paper we investigate connections between statistical learning theory and data compression on the basis of support vector machine (SVM) model selection. Inspired by several generalization bounds we construct "compression coefficients" for SVMs which measure the amount by which the training labels can be compressed by a code built from the separating hyperplane. The main idea is to relate the coding precision to geometrical concepts such as the width of the margin or the shape of the data in the feature space. The so derived compression coefficients combine well known quantities such as the radius-margin term R^2/ρ^2, the eigenvalues of the kernel matrix, and the number of support vectors. To test whether they are useful in practice we ran model selection experiments on benchmark data sets. As a result we found that compression coefficients can fairly accurately predict the parameters for which the test error is minimized.

Keywords: Support vector machine, compression coefficient, minimum description length, model selection

1. Introduction

In classification one tries to learn the dependency of labels y on patterns x from a given training set $(x_i, y_i)_{i=1...m}$. We are interested in the connections between two different methods to analyze this problem: a data compression approach and statistical learning theory.
The minimum description length (MDL) principle (cf. Barron et al., 1998, and references therein) states that, among a given set of hypotheses, one should choose the hypothesis that achieves the shortest description of the training data. Intuitively, this seems to be a reasonable choice: by Shannon's source coding theorem (cf. Cover and Thomas, 1991) we know that an efficient code is closely related to the data generating distribution. Moreover, an easy-to-describe hypothesis is less likely to overfit than a more complicated one.

There are several connections between the MDL principle and other learning methods. It can be shown that selecting the hypothesis with the highest posterior probability in a Bayesian setting is equivalent to choosing the hypothesis with the shortest code (cf. Hansen and Yu, 2001). For SVMs, the compression scheme approach of Floyd and Warmuth (1995), which describes the generalization of a learning algorithm by its ability to reduce the training set to a few important points, leads to a bound in terms of the number of support vectors. Combining this result with large margin bounds yields the sparse margin bound of Herbrich et al. (2000). In McAllester (1999), a PAC-Bayesian bound for learning algorithms was derived. For a given prior distribution P on the hypotheses space, it bounds the generalization error essentially by the quantity $-\ln P(U)/m$, where U is the subset of hypotheses consistent with the training examples. Intuitively we can argue that, according to Shannon's theorem, $-\ln P(U)$ corresponds to the length of the shortest code for this subset.

There are statistical learning theory approaches which directly use compression arguments to bound the generalization ability of a classifier, for example the one of Vapnik (1998, sec. 6.2). In this framework, classifiers are used to construct codes that can transmit the labels of a set of training patterns. What those codes essentially do is to tell the receiver which classifier h from a given hypotheses class he should use to reconstruct the training labels from the training patterns (this is described in detail in Section 2). For such a code, the *compression coefficient* $C(h)$ is defined as

$$C(h) := \frac{\text{number of bits to code } y_1, \ldots, y_m \text{ using } h}{m}. \tag{1}$$

The denominator corresponds to the number of bits we need to transmit the uncompressed binary vector $(y_1, \ldots, y_m)$. The numerator tells us how many bits we need to transmit the same information using the code constructed with help of classifier h. Hence, the compression coefficient is a number between 0 and 1 which describes how efficient the code works. Intuitively, if the compression coefficient $C(h)$ is small, h is a "simple" hypothesis which we expect not to overfit too much and hence to have a small generalization error. This belief is supported by the following theorem, which bounds the risk $R(h)$ of a classifier h (i.e., the expected generalization error with respect to the 0-1-loss) in terms of the compression coefficient:

Theorem 1 (Section 6.2 in Vapnik 1998) *With probability at least $1-\eta$ over m random training points drawn iid according to the unknown distribution P, the risk $R(h)$ of classifier h is bounded by*

$$R(h) \leq 2\ln(2)C(h) - \frac{\ln(\eta)}{m}$$

simultaneously for all classifiers h in a finite hypotheses set.

This bound has the disadvantage that it is only valid in the restricted setting where the hypotheses space is finite and independent of the training data.

A different bound that directly works in the coding setting has recently been stated by Blum and Langford (2003). Their setting is slightly different from the one of Vapnik. For a given set of training and test points, the sender constructs a code σ that transmits the labels of both training and test points. R_{test} is then defined as the error which the code σ makes on the given test set.

Theorem 2 (Corollary 3 in Blum and Langford 2003) *With probability at least $1-\eta$ over m random training points and m random test points drawn iid according to the unknown distribution P, and for all codes σ which encode the training labels without error, the error $R_{test}(\sigma)$ on the test set satisfies*

$$R_{test}(\sigma) \leq C(\sigma) - \frac{\ln \eta}{m}.$$

The advantage of this bound is that the problem of data dependency does not occur. It is proved within the coding framework, without assuming a fixed hypotheses space. It is valid for all codes, as long as receiver and sender agree on how the code works before the sender knows the training data. In particular, the codes may depend on the training data in some predefined way, as it will be the case for the codes we are going to construct.

Inspired by the compression coefficient bounds we want to explore whether the connection between statistical learning theory and data compression is only of theoretical interest or whether it can also be exploited for practical purposes. The first part of the work (Section 2) will consist in using hyperplanes learned by SVMs to construct codes for the training labels. Those codes work by encoding the direction of the separating hyperplane. To make them efficient, we will use geometric concepts such as the size of the margin or the shape of the data in the feature space, and sparsity arguments. The main insight of this section is on how to transform those geometrical concepts into an actual code for labels. In the end we obtain compression coefficients that contain quantities already known to be meaningful in the statistical learning theory of SVMs, such as the radius-margin term R^2/ρ^2, the eigenvalues of the kernel matrix, and the number of support vectors. In the second part (Section 3) we then test the model selection performance of our compression coefficients on benchmark data sets. We find that the compression coefficients perform comparable or better than several standard bounds.

Now we want to establish some notation. We assume that the reader is familiar with some basic concepts concerning support vector machines such as a kernel, the margin, and support vectors. In the following, the training data will always consist of m pairs $(x_i, y_i)_{i=1,\dots,m}$ of patterns with labels. The patterns are assumed to live in some Hilbert space (the feature space), and the labels have binary values ± 1. For a given kernel function k we denote the kernel matrix by $K := (k(x_i, x_j))_{i,j=1,\dots,m}$. We will denote its eigenvalues by $\lambda_1, \dots, \lambda_m$, where the λ_i are sorted in non-increasing order. Later on we will also consider the kernel matrix K_{SV} restricted to the span of the support vectors. To define it, let $\{x_{i_1}, \dots, x_{i_s}\} \subset \{x_1, \dots, x_m\}$ the set of support vectors, $SV := \{k \mid x_k \in \operatorname{span}\{x_{i_1}, \dots, x_{i_s}\}\}$ the indices of those training points which are in the subspace spanned by the support vectors. Then the kernel matrix restricted to the span of the support vectors is defined as $K_{SV} := (k(x_i, x_j))_{i,j \in SV}$. In the MDL literature, classifiers are called "hypotheses". In what follows, we use the word "hypothesis" synonymous to "classifier". A hypotheses space is then the space of all possible classifiers we can choose from. The sphere S_R^{d-1} is the surface of a ball with radius R in the space $\mathbb{R}^d$. The function log will always denote the logarithm to the base 2. Code lengths will often be given by some logarithmic term, for instance $\lceil \log m \rceil$. To keep the notations simple, we will omit the ceil brackets and simply write $\log m$.

2. Compression Coefficients for SVMs

The basic setup for the compression coefficient framework is the following. We are given m pairs $(x_i, y_i)_{i=1,\dots,m}$ of training patterns with labels and assume that an imaginary sender and receiver both know the training *patterns*. It will be the task of the sender to transmit the *labels* of the training patterns to the receiver. This reflects the basic structure of a classification problem: we want to predict the labels y for given patterns x. That is, we want to learn something about $P(y|x)$. Sender and receiver are allowed to agree on the details of the code before transmission starts. Then the sender gets the training data and chooses a classifier that separates the training data. He transmits to the receiver which classifier he chose. The receiver can then apply this classifier to the training patterns and reconstruct all the labels.

To understand how this works let us consider a simple example. Before knowing the data, sender and receiver agree on a finite hypotheses space containing k hypotheses $h_1, \dots, h_k$. The sender gets the training patterns and the labels, and the receiver is allowed to look at the training patterns only. Now the sender inspects the training data. For simplicity, let us first assume that one of the hypotheses, say h_7, classifies all training points correctly. In this case the sender transmits "7" to the receiver. The receiver can now reconstruct the labels of the training patterns by classifying them according to hypothesis h_7. Now consider the case where there is no hypothesis that classifies all training patterns correctly. In this case, the receiver cannot reconstruct all labels without error if the sender only transmits the hypothesis. Additionally he has to know which of the training points are misclassified by this hypothesis. In our example, assume that hypothesis 7 misclassifies the training points 3, 14, and 20. The information the sender now transmits is "hypothesis: 7; misclassified points: 3, 14, 20". The receiver can then construct the labels of all patterns according to h_7 and flip the labels he obtained for patterns 3, 14, and 20. After this, he has labeled all training patterns correctly.

This example shows how a classification hypothesis can be used to transmit the labels of training points. In general, a finite, fixed hypothesis space as it was used in the example may not contain a good hypothesis for previously unseen training points and will result in long codes. To avoid this, the sender will have to adapt the hypothesis space to the training points and communicate this to the receiver.

One principle that can already be observed in the example above is the way training errors are handled. As above, the code will always consist of two parts: the first part which serves to describe the hypothesis, and the second part, which tells the receiver which of the training points are misclassified by this hypothesis.

In the following we want to investigate how SVMs can be used for coding the labels. The main part of those codes will consist in transmitting the direction of the separating hyperplane constructed by the SVM. We will always consider the simplified problem where the hyperplane goes through the origin. The hypotheses space consists of all possible directions the normal vector can take. It can be identified with the unit sphere in the feature space. In case of an infinite dimensional feature space, recall that by the representer theorem the solution of an SVM always lies in the subspace spanned by the training points. Thus the normal vector we want to code is a vector in a Hilbert space of dimension at most m (where m is the number of training points). The trick to determine the precision by which we have to code this vector is to interpret the margin in terms of coding precision: Suppose the data are (correctly) classified by a hyperplane with normal vector ω and margin ρ. A fact that often has been observed (e.g., Schölkopf and Smola, 2002, p. 194) is that in case of a large margin, small perturbations of the direction of the hyperplane will not change the classification result on the training points. In compression language this means that we do not have to code the direction of the hyperplane with high accuracy – the larger the margin, the less accurate we have to code. So we will adapt the precision by which we code this direction to the width of the margin. Suppose that all training patterns lie in a ball of radius R around the origin and that they are separated by a hyperplane H through the origin with normal vector ω and margin ρ. Then every "slightly rotated" hyperplane that still lies within the margin achieves the same classification result on all training points as the original hyperplane (cf. Figure 1a). Thus, instead of using the hyperplane with normal vector ω to separate the training points we could use any convenient vector v as normal vector – as long as the corresponding hyperplane still remains inside the margin. In this context, note that rotating a hyperplane by some angle α corresponds to rotating its normal vector by the same angle. We denote the set of normal vectors such that the corresponding hyperplanes still lie inside the margin the *rotation region* of ω. The region in which the corresponding hyperplanes lie will be called the *rotation region of H* (cf. Figures 1a and b).

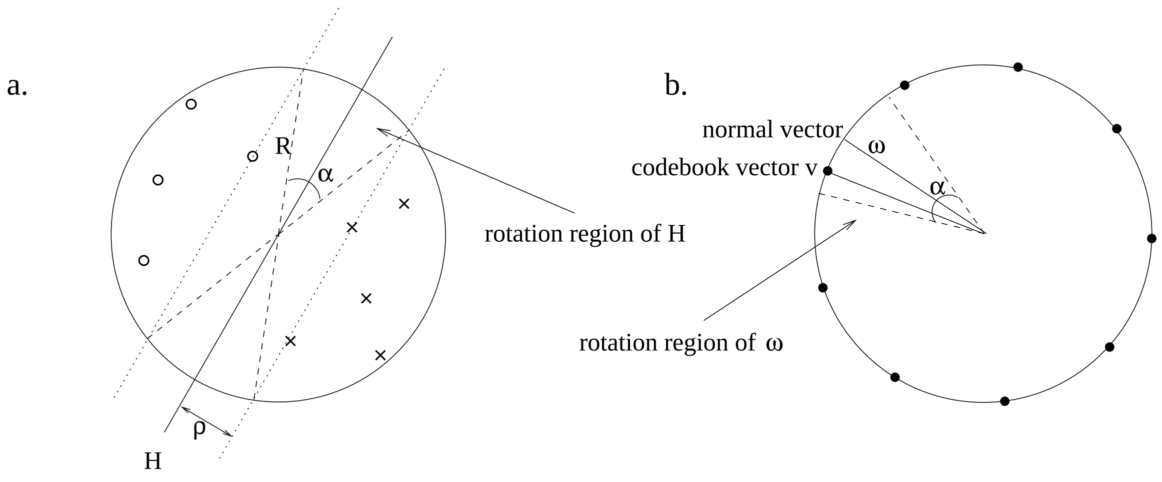

Figure 1: (a) The training points are separated by hyperplane H with margin ρ. If we change the direction of H such that the new hyperplane still lies inside the rotation region determined by the margin (dashed lines), the new hyperplane obtains the same classification result on the training points as H. (b) The hyperplanes in the rotation region of H (indicated by the dashed lines) correspond to normal vectors inside the rotation region of ω. The black points indicate the positions of equidistant codebook vectors on the sphere. The distance between those vectors has to be chosen so small that in each cone of angle α there is at least one codebook vector. In this example, vector v is the codebook vector closest to normal vector ω, and by construction it lies inside the rotation region of ω.

To code the direction of ω, we will construct a discrete set of "codebook vectors" on the sphere. An arbitrary vector will be coded by choosing the closest codebook vector. From the preceding discussion we can see that the set of codebook vectors has to be constructed in such a way that the closest codebook vector for every possible normal vector ω is inside the rotation region of ω (cf. Figure 1b). An equivalent formulation is to construct a set of points on the surface of the sphere such that the balls of radius ρ centered at those points cover the sphere. The minimal number of balls we need to achieve this is called the covering number of the sphere.

Proposition 3 (Covering numbers of spheres) *The number n_d of balls of radius ρ which are required to cover the sphere S_R^{d-1} of radius R in d-dimensional Euclidean space ($d \geq 2$) satisfies*

$$\left(\frac{R}{\rho}\right)^{d-1} \leq n_d \leq 2\left\lceil\frac{R\pi}{\rho}\right\rceil^{d-1}.$$

The constant in the upper bound can be improved, but as we will be interested in log-covering numbers later, this does not make much difference in our application.

Proof We prove the upper bound by induction. For $d = 2$, the sphere is a circle which can be covered with $\lceil 2R\pi/\rho \rceil \leq 2\lceil R\pi/\rho \rceil$ balls. Now assume the proposition is true for the sphere S_R^{d-1}. To construct a ρ-covering on S_R^d we first cover the cylinder $S_R^{d-1} \times [-R\pi/2, R\pi/2]$ with a grid of $\tilde{n}_{d+1} := n_d \cdot \lceil R\pi/\rho \rceil$ points. This grid is a ρ-cover of the cylinder. The grid is then mapped on the sphere such that the one edge of the cylinder is mapped on the north pole, the other edge on the south pole, and the 'equator' of the cylinder is mapped to the equator of the sphere. As the distances between the grid points do not increase by this mapping, the projected points form a ρ-cover of the sphere S_R^d. By the induction assumption, the number of points in this ρ-cover satisfies

$$n_{d+1} \leq \tilde{n}_{d+1} = n_d \cdot \lceil R\pi/\rho \rceil \leq 2\lceil R\pi/\rho \rceil^{d-1} \cdot \lceil R\pi/\rho \rceil = 2\lceil R\pi/\rho \rceil^{d}.$$

We construct a lower bound on the covering number by dividing the surface area of the whole sphere by the area of the part of the surface covered by one single covering ball. The area of this part is smaller than the whole surface of the small ball. So we get a lower bound by dividing the surface area of S_R^{d-1} by the surface area of S_ρ^{d-1}. As the surface area of a sphere with radius R is R^{d-1} times the surface area of the unit sphere we get $(R/\rho)^{d-1}$ as lower bound for the covering number. ■

Now we can explain how the sender will encode the direction of the separating hyperplane. Before getting the data, sender and receiver agree on a procedure on how to determine the centers of a covering of a unit sphere, given the number of balls to use for this covering, and on a way of enumerating these centers in some order. Furthermore, they agree on which kernel the sender will use for his SVM. Both sender and receiver get to see the training patterns, the sender also gets the training labels. Now the sender trains a (soft margin) SVM on the training data to obtain a hyperplane that separates the training patterns with some margin ρ (maybe with some errors). Then he computes the number n of balls of radius ρ one needs to cover the unit sphere in the feature space according to Proposition 3. He constructs such a covering according to the procedure he and the receiver agreed on. The centers of the covering balls form his set of codebook vectors. The sender enumerates the set of codebook vectors in the predefined way from 1 to n. Then he chooses a codebook vector which lies inside the rotation region of the normal vector (this is always possible by construction). We denote its index $i_n \in \{1,...,n\}$. Now he transmits the total number n of codebook vectors and the index i_n of the one he chose. The receiver now constructs the same set of codebook vectors according to the common procedure, enumerates them in the same predefined way as the sender and picks vector i_n. This is the normal vector of the hyperplane he was looking for, and he can now use the corresponding hyperplane to classify the training patterns. In the codes below, we refer to the pair (n, i_n) as *position of the codebook vector*.

When we count how many bits the sender needs to transmit the two numbers n and i_n we have to keep in mind that to decode, the receiver has to know which parts of the binary string belong to n and i_n, respectively. The number n of codebook vectors is given as in Proposition 3, but as it depends on the margin it cannot be bounded independent from the training data. So the sender cannot use a fixed number of bits to encode n. Instead we use a trick described in Cover and Thomas (1991, p. 149): To build a code for the number n, we take the binary representation of n and duplicate every bit. To mark the end of the code we use the string 01. As an example, n = 27 with the binary representation 11011 will be coded as 111100111101. So the receiver knows that the code of n is finished when he comes upon a pair of nonequal bits. We now apply this trick recursively: to code n, we first have to code the length $\log n$ of its binary code and then send the actual bits of the code of n. But instead of coding $\log n$ with the duplication code explained above, we can also first transmit the length $\log\log n$ of the code of $\log n$ and then transmit the code of $\log n$, and so on. At some point we stop this recursive procedure and code the last remaining number with the duplication code described above. This procedure of coding n needs $\log^*(n) := \log n + \log\log n + ...$ bits, where the sum continues until the last positive term (cf. p. 150 in Cover and Thomas, 1991). Having transmitted n, we can send i_n with $\log n$ bits, because i_n is a number between 1 and n and the sender now already knows n. All in all, the sender needs $\log^* n + \log n$ bits to transmit the position of the codebook vector.

The second part of the code deals with transmitting which of the training patterns are misclassified by the hyperplane corresponding to the chosen codebook vector. We have to be careful how we define the misclassified training points in case of a soft margin SVM. For a soft margin SVM it is allowed that some training points lie inside the margin. For those training points which end up inside the rotation region of the hyperplane, our rotation argument breaks down. It cannot be guaranteed that when the receiver uses the hyperplane corresponding to the codebook vector, the points inside the rotation region are classified in the same way as the sender classified them with the original hyperplane. Thus the sender has to transmit which points are inside the rotation region, and he also has to send their labels. All other points can be treated more easily. The sender has to transmit which of the points outside the rotation region were misclassified. The receiver knows that those points will also be misclassified by his hyperplane, and he can flip their labels to get them right. Below we will refer to the part consisting of the information on the points inside the rotation region and on the misclassified points outside the rotation region as *misclassification information*.

The number r of points inside the rotation region is a number between 0 and m, thus we can transmit its binary representation using $\log m$ bits. After transmitting r, the receiver knows how many training points lie inside the region, but not which of them. There are $\binom{m}{r}$ possibilities which of the training points are the points inside the rotation region. Before transmission, sender and receiver agree on an ordering on those possibilities. Now the sender can transmit the index i_r of the one that is the true one. As this is a number between 1 and $\binom{m}{r}$, and as the receiver at this point already knows r, this can be encoded with $\log\binom{m}{r}$ bits. Next the sender can use r bits to send the labels of the r points inside the rotation region. Finally the sender has to transmit the number l of misclassified training points outside the rotation region. It is a number between 1 and $m-r$, thus we can use $\log(m-r)$ bits for this. To transmit which of the vectors are the misclassified ones, we send the index i_l with $\log\binom{m-r}{l}$ bits. All together, we need $\log m + \log\binom{m}{r} + r + \log(m-r) + \log\binom{m-r}{l}$ bits to transmit the misclassification information. For simplicity, we bound this quantity from above by $(r+l+2)\log m + r$.

Now we can formulate our first code:

Code 1 *Sender and receiver agree on the training patterns, a fixed kernel, and on a procedure for choosing the positions of t balls to cover the unit sphere in a d-dimensional Euclidean space. Now the sender trains an SVM with the fixed kernel on the training patterns, determines the size of the margin ρ and the number n of balls he needs to cover the sphere up to the necessary accuracy according to Proposition 3. Furthermore, he determines which of the training patterns lie inside the rotation region and which patterns outside this region are misclassified by the SVM solution. Now he transmits*

- *the position of the codebook vector ($\log^* n + \log n$ bits)*
- *the misclassification information ($(r+l+2)\log m + r$ bits)*

To decode this information, the receiver constructs a covering of the sphere in the feature space with $t := n$ balls according to the common procedure, determines the used codebook vector i, and *constructs a hyperplane using this vector as normal vector. He classifies all training patterns according to this hyperplane, labels the points inside the rotation region as transmitted by the sender, and flips the labels of the misclassified training points outside the rotation region.*

As defined in Equation (1), the compression coefficient of a code is given by the number of bits it needs to transmit all its information, divided by the number m of labels that were transmitted. Hence, according to our computations above, the compression coefficient of Code 1 is given by

$$C_1 = \frac{1}{m}\left(\log^* n + \log n + (r+l+2)\log m + r\right)$$

with $n = 2\left\lceil R\pi/\rho \right\rceil^{d-1}$ according to Proposition 3.

Now we want to refine this code in several aspects. In the construction above we worked with the smallest sphere which contains all the training points. But in practice, the shape of the data in the feature space is typically ellipsoid rather than spherical. This means that large parts of the sphere we used above are actually empty, and thus the code we constructed is not very efficient. Now we want to take into account the shape of the data in the feature space to construct a shorter code. In this setting it will turn out to be convenient to choose the hypotheses space to have the same shape as the data space. The reason for this is the following: When using the rotation argument from above, we observe that in the ellipsoid situation the maximal rotation angle of the hyperplane induced by some fixed margin ρ depends on the actual direction of the hyperplane (cf. Figure 2). This means that the sets of points on a *spherical hypotheses space* which classify the training data in the same way as the hyperplane also have different sizes, depending on the direction of the hyperplane. To construct an optimal set of codebook vectors on the spherical hypotheses space we thus had to cover the sphere with balls of different sizes. Instead of this spherical hypotheses space now consider a hypotheses space which has the same ellipsoid form as the data space. In this case, the sets of vectors which correspond to directions inside the rotation regions can be represented by balls of equal sizes, centered on the surface of the ellipsoid (cf. Figure 2).

Now we want to determine the shape of the ellipsoid containing the data points in the feature space. The lengths of the principal axes of this ellipse can be described in terms of the eigenvalues of the kernel matrix:

Proposition 4 (Shape of the data ellipse) *For given training patterns $(x_i)_{i=1,\dots,m}$ and kernel k, let $\lambda_1,\dots,\lambda_d$ be the eigenvalues of the kernel matrix $K = (k(x_i,x_j))_{i,j=1,\dots,m}$. Then all training patterns are contained in an ellipse with principal axes of lengths $\sqrt{\lambda_1},\dots,\sqrt{\lambda_d}$ in the feature space.*

Proof The trick of the proof is to interpret the eigenvectors of the kernel matrix, who originally live in $\mathbb{R}^d$, as vectors in the feature space. Let $H_m := \operatorname{span}\{\delta_{x_i} | i = 1,\dots,m\}$ the subspace of the feature space spanned by the training examples. It is endowed with the scalar product $\langle \delta_{x_i}, \delta_{x_j}\rangle_K = k(x_i,x_j)$. Let $(e_i)_{i=1,\dots,m}$ the canonical basis of $\mathbb{R}^m$ and $\langle\cdot,\cdot\rangle_{\mathbb{R}^m}$ the Euclidean scalar product. Define the mapping $T : \mathbb{R}^m \to H_m,\ e_i \mapsto \delta_{x_i}$. For $u = \sum_{i=1}^m u_i e_i,\ v = \sum_{j=1}^m v_j e_j \in \mathbb{R}^m$ we have

$$\langle Tu, Tv\rangle_K = \langle \sum_{i=1}^m u_i \delta_{x_i}, \sum_{j=1}^m v_j \delta_{x_j}\rangle_K = \sum_{i,j=1}^m u_i v_j \langle \delta_{x_i}, \delta_{x_j}\rangle_K = \sum_{i,j=1}^m u_i v_j k(x_i,x_j) = u'Kv\,. \tag{2}$$

Let $v_1,\dots,v_d \in \mathbb{R}^m$ be the normalized eigenvectors of the matrix K corresponding to the eigenvalues $\lambda_1,\dots,\lambda_d$, i.e., $Kv_i = \lambda_i v_i$ and $\langle v_i, v_j\rangle_{\mathbb{R}^m} = \delta_{ij}$. From Equation (2) we can deduce

$$\langle Tv_i, Tv_j\rangle_K = v_i'Kv_j = v_i'\lambda_j v_j = \lambda_j \langle v_i, v_j\rangle_{\mathbb{R}^m} = \lambda_i \delta_{ij},$$

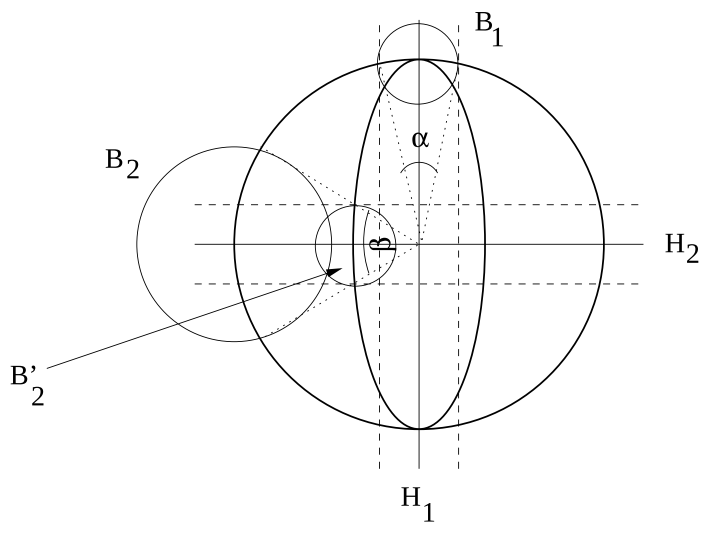

Figure 2: In this figure, the ellipse represents the data domain and the large circle represents a spherical hypotheses space. Consider two hyperplanes H_1 and H_2 with equal margin. The balls B_1 resp. B_2 indicate the sets of hypotheses that yield the same classification result as H_1 resp. H_2. The sizes of those balls depend on the direction of the hyperplane with respect to the ellipse. In the example shown, B_1 is smaller than B_2, hence H_1 must be coded with higher accuracy than H_2. Note that if we use the ellipse itself as hypotheses space, we have to consider the balls B_1 and B_2' which are centered on the surface of the ellipse and have equal size.

in particular $\|Tv_i\|_K = \sqrt{\lambda_i}$. Furthermore we have

$$\langle \delta_{x_i}, Tv_j \rangle_K = \langle \delta_{x_i}, \sum_{l=1}^{m} (v_j)_l \delta_{x_l} \rangle_K = \sum_{l=1}^{m} (v_j)_l k(x_i, x_l) = (Kv_j)_i = (\lambda_j v_j)_i = \lambda_j (v_j)_i.$$

Altogether we can now see that in the feature space, all data points δ_{x_i} lie in the ellipse whose principal axes have direction Tv_j and length $\sqrt{\lambda_j}$ because the ellipse equation is satisfied:

$$\sum_{j=1}^{d} \left(\frac{\langle \delta_{x_i}, \frac{Tv_j}{\|Tv_j\|_K} \rangle_K}{\sqrt{\lambda_j}} \right)^2 = \sum_j ((v_j)_i)^2 \leq 1.$$

Here the last equality follows from the fact that $\sum_j ((v_j)_i)^2$ is the Euclidean norm of a row vector of the orthonormal matrix containing the eigenvectors $(v_1, \ldots, v_d)$. ■

Now that we know the shape of the ellipse, we have to find out how many balls of radius ρ we need to cover its surface. As surfaces of ellipses in high dimensions are complicated to deal with, we simplify our calculation. Instead of covering the surface of the ellipse, we will cover the ellipse completely. This means that we use one extra dimension (volume instead of area), but in high dimensional spaces this does not make much difference, especially if some of the axes are very small. Computing a rough bound on the covering numbers of an ellipse is easy:

Proposition 5 (Covering numbers of ellipses) *The number n of balls of radius ρ which are required to cover a d-dimensional ellipse with principal axes $c_1,\dots,c_d$ satisfies*

$$\prod_{i=1}^{d} \frac{c_i}{\rho} \le n \le \prod_{i=1}^{d} \left\lceil \frac{2c_i}{\rho} \right\rceil .$$

Proof The smallest parallelepiped containing the ellipse has side lengths $2c_1,\dots,2c_d$ and can be covered with a grid of $\prod_{i=1}^{d} \lceil 2c_i/\rho \rceil$ balls of radius ρ. This gives an upper bound on the covering number.

To obtain a lower bound we divide the volume of the ellipse by the volume of one single ball. Let v_d be the volume of a d-dimensional unit ball. Then the volume of a d-dimensional ball of radius ρ is $\rho^d v_d$ and the volume of an ellipse with axes $c_1,\dots,c_d$ is given by $v_d \prod_{i=1}^{d} c_d$. So we need at least $\prod_{i=1}^{d}(c_i/\rho)$ balls. ■

Now we can formulate the refined code:

Code 2 *This code works analogously to Code 1, the only difference is that sender and receiver work with a covering of the data ellipse instead of a covering of the enclosing sphere.*

The compression coefficient of Code 2 is

$$C_2 = \frac{1}{m}\left(\log^* n + \log n + (r+l+2)\log m + r\right),$$

with $n = \prod_{i=1}^{d} \lceil 2\sqrt{\lambda_i}/\rho \rceil$ according to Propositions 4 and 5.

It is interesting to notice that the main complexity term in the compression coefficient is the logarithm of the number of balls needed to cover the region of interest of the hypotheses space. So the shape of the bounds we obtain is very similar to classical bounds based on covering numbers in statistical learning theory. This is not so surprising since we explicitly approximate our hypotheses space by covers, but there is a somewhat deeper connection. Indeed, when we construct our code, we consider all normal vectors in a certain region as equivalent with respect to the labeling they give of the data. This means that we define a metric on the set of possible normal vectors which is related to the induced Hamming distance on the data (that is the natural distance in the "coordinate projections" of our function class on the data). Hence, when we adapt the size of the balls to the direction in hypotheses space (in Figure 2), we actually say that Hamming distance 1 on the data translates into a certain radius. Hence, we are led to build covers in this induced distance which is exactly the distance which is used in classical covering number bounds for classification. So the compression approach gives another motivation, of information theoretic flavor, for considering that the right measure of the capacity of a function class is the metric entropy of its coordinate projections.

Both compression coefficients we derived so far implicitly depend on the dimension d of the feature space in which we code the hyperplane. Above we always used $d = m$ as the solution of an SVM always lives in the subspace spanned by the training examples. But as the solution even lies in the subspace spanned by the support vectors, an easy dimension reduction can be achieved by working in this subspace. The procedure then works as follows: the sender trains the SVM and determines the support vectors and the margin ρ. The ellipse that we have to consider now is the ellipse determined by the kernel matrix K_{SV} restricted to the linear span of the support vectors (cf. notations at the end of Section 1). The reason for this is that we are only interested in what happens if we slightly change the direction of the normal vector *within* the subspace spanned by the support vectors.

To let the receiver know the subspace he is working in, the sender has to transmit which of the training patterns are support vectors. This part of the code will be called *support vector information*. As the number s of support vectors is between 0 and m, the sender first codes s with $\log m$ bits and then the index i_s of the actual support vectors among the $\binom{m}{s}$ possibilities with $\log \binom{m}{s}$ bits. So the support vector information can be coded with $\log m + \log \binom{m}{s} \leq (s+1)\log m$ bits. After submitting the information about the support vectors, the code proceeds analogously to Code 2.

Code 3 *Sender and receiver agree on the training patterns, the kernel, and the procedure of covering an ellipsoid. After training an SVM, the sender transmits*

- *the support vector information ($(s+1)\log m$ bits),*
- *the position of the codebook vector ($\log^* n + \log n$ bits),*
- *the misclassification information ($(r+l+2)\log m + r$ bits).*

To decode, the receiver constructs the hypotheses space consisting of the data ellipse projected on the subspace spanned by the support vectors. He covers this ellipse with n balls and chooses the vector representing the normal vector of the hyperplane. Then he labels the training patterns by first projecting them into the subspace and then classifying them according to the hyperplane. Finally, he deals with the misclassified training points as in the codes before.

The compression coefficient of Code 3 is given as

$$C_3 = \frac{1}{m}\left(\log^* n + \log n + (r+l+s+3)\log m + r\right),$$

with $n \leq \prod_{i=1}^{s} \lceil 2\sqrt{\gamma_i}/\rho \rceil$ according to Propositions 4 and 5. Here γ_i denote the eigenvalues of the restricted kernel matrix K_{SV}.

A further dimension reduction can be obtained with the following idea: It has been empirically observed that on most data sets the axes of the data ellipse decrease fast for large dimensions. Once the axis in one direction is very small, we want to discard this dimension by projecting in a lower dimensional subspace using kernel principal component decomposition (cf. Schölkopf and Smola, 2002). A projection P will be allowed if the image $P(\omega)$ of the normal vector ω is still within the rotation region induced by the margin ρ. In this case we construct codebook vectors for $P(\omega)$ in the lower dimensional subspace. We have to make sure that the vector representing $P(\omega)$ is still contained in the original rotation region.

In more detail, this approach works as follows: First we train the SVM and get the normal vector ω and margin ρ. For convenience, we now normalize ω to length R by $\omega_0 := \frac{\omega}{||\omega||}R$. After normalizing, we know that a vector v still lies inside the rotation region if $\|\omega_0 - v\| \leq \rho$. Now we perform a kernel PCA of the training data in the feature space. For $d_P \in \{1,...,m\}$ let P be the projection on the subspace spanned by the first d_P eigenvectors. To determine whether we are allowed to perform projection d_P we have to check whether $\|P(\omega_0) - \omega_0\| \leq \rho$. If not, the hyperplane corresponding to $P(\omega_0)$ is not within the rotation region any more, and we are not allowed to make this projection. Otherwise, we are still within the rotation region after projecting ω_0, so we can discard the last $m - d_P$ dimensions. In this case, we call P a *valid* projection. We then can encode the projected normal vector $P(\omega_0)$ in an d_P-dimensional subspace. As $P(\omega_0)$ is not in the center of the rotation region any more, we have to code its direction more precisely now. We have to ensure that the codebook vector v for $P(\omega_0)$ still is in the original rotation region, that is $\|v - \omega_0\| \leq \rho$. Define

$$c_P := \frac{1}{\rho}\|\omega_0 - P(\omega_0)\|$$

(note that for a valid projection, $c_P \in [0,1]$), and choose the radius r of the covering balls as $r = \rho\sqrt{1-c_P^2}$. Then we have

$$\|\omega_0 - v\|^2 \leq \|\omega_0 - P(\omega_0)\|^2 + \|P(\omega_0) - v\|^2 \leq c_P^2\rho^2 + (1-c_P^2)\rho^2 = \rho^2 \,.$$

Thus the codebook vector v is still within the allowed distance of ω_0.

All in all our procedure now works as follows: The sender trains an SVM. From now on he works in the subspace spanned by the support vectors only. In this subspace, he performs the PCA and determines the smallest d_P such that the projection on the subspace of d_P dimensions is a valid projection.The principal axes of the ellipse in the subspace are now given by the first d_P eigenvalues of the restricted kernel matrix K_{SV}, and we have to construct a covering of this ellipse with covering radius $r = \rho\sqrt{1-c_P^2}$. Then the code proceeds as before. The number d_P will be called the *projection information* and can be encoded with $\log m$ bits.

Code 4 *Sender and receiver agree on the training patterns, the kernel, the procedure of covering ellipsoids, and on how to perform kernel PCA. The sender trains the SVM and chooses a valid projection on some subspace. He transmits*

- *the support vector information ($(s+1)\log m$ bits),*
- *the projection information ($\log m$ bits),*
- *the position of the codebook vector ($\log^* n + \log n$ bits),*
- *the misclassification information ($(r+l+2)\log m + r$ bits).*

To decode, the receiver constructs the hypotheses space consisting of the ellipse in the subspace spanned by the support vectors. Then he performs a PCA in this subspace and projects the hypotheses space on the subspace spanned by the first d_P principal components. He covers the remaining ellipse with n balls and continues as in the codes before.

The compression coefficient of this code is given by

$$C_4 = \frac{1}{m}\left(\log^* n + \log n + (r+l+s+4)\log m + r\right),$$

with $n \leq \prod_{i=1}^{d_P} \lceil 2\sqrt{\gamma_i}/(\rho\sqrt{1-c_P^2}\,)\rceil$, c_P as described above, and γ_i the eigenvalues of the restricted kernel matrix K_{SV}.

So far we always considered codes which use the direction of the hyperplane as hypothesis. A totally different approach is to reduce the data by transmitting support vectors and their labels. Then the receiver can train his own SVM on the support vectors and will get the same result as the sender. Note that in this case, we not only have to transmit which of the vectors are support vectors as in the other codes, but also the labels of the support vectors. On the other hand we have the advantage that we do not have to treat the points inside the rotation region separately as they are support vectors anyway. The misclassification information only consists in the misclassified points which are not support vectors. This simple code works as follows:

Code 5 *Sender and receiver agree on training patterns and a kernel. The sender sends*

- *the support vector information ($(s+1)\log m$ bits),*
- *the labels of the support vectors (s bits),*
- *the information on the misclassified points outside the rotation region ($(l+1)\log m$ bits).*

To decode this information, the receiver trains an SVM with the support vectors as training set. Then he computes the classification result of this SVM for the remaining training patterns and flips the labels of the misclassified non-support vector training points.

This code has a compression coefficient of

$$C_5 = \frac{1}{m}\left((s+l+2)\log m + s\right).$$

3. Experiments

To test the utility of the derived compression coefficients for applications, we ran model selection experiments on different artificial and real world data sets. We used all data sets in the benchmark data set repository compiled and explained in detail in Rätsch et al. (2001). The data sets are available at http://ida.first.fraunhofer.de/projects/bench/benchmarks.htm. Most of the data sets in this repository are preprocessed versions of data sets originating from the UCI, Delve, or STATLOG repositories. In particular, all data sets are normalized. The data sets are called banana (5300 points), breast cancer (277 points), diabetis (768 points), flare-solar (1066 points), german (1000 points), heart (2700 points), image (2310 points), ringnorm (7400 points), splice (3175 points), thyroid (215 points), titanic (2201 points), twonorm (7400 points), waveform (5100 points). To be consistent with earlier versions of this manuscript we also used the data sets abalone (4177 points), Wisconsin breast cancer (683 points) from the UCI repository, and the US postal handwritten digits data set (9298 points). In all experiments, we first permuted the whole data set and divided it into as many disjoint training subsets of sample size $m = 100$ or 500 as possible. We centered each training subset in the feature space as described in Section 14.2. of Schölkopf and Smola (2002) and then used it to train soft margin SVMs with Gaussian kernels. The test error was computed on the training subset's complement.

For different choices of the soft margin parameter $C \in [10^0, \dots, 10^5]$ and the kernel width $\sigma \in [10^{-2}, \dots, 10^3]$ we computed the compression coefficients and chose the parameters where the compression coefficients were minimal. Note that as we centered the data in the feature space, the radius R can be approximately computed as the maximum distance of the centered training points to the origin. We compared the test errors corresponding to the chosen parameters to the ones obtained by model selection criteria from different generalization bounds. To state those bounds we denote the empirical risk of a classifier by $\mathrm{R}_{\mathrm{emp}}$ and the true risk of a classifier by $\mathrm{R}_{\mathrm{true}}$. Note that the definition of the risks varies slightly for the different bounds, for example with respect to the used loss function. We refer to the cited papers for details. The bounds we consider are the following:

- The radius-margin bound of Vapnik (1998). We here cite the version stated in Bartlett and Shawe-Taylor (1999): with probability at least $1-\delta$,

  $$\mathrm{R}_{\mathrm{true}} \leq \mathrm{R}_{\mathrm{emp}} + \sqrt{\frac{c}{m}\left(\frac{R^2}{\rho^2}\log^2 m - \log\delta\right)}.$$

  The quantity we compute in the experiments is $\mathrm{R}_{\mathrm{emp}} + \sqrt{(R^2\log m)/(\rho^2 m)}$.

- The rescaled radius-margin bound of Chapelle and Vapnik (2000). It uses the shape of the training data in the feature space to refine the classical radius margin bound. In this case, the quantity R^2/ρ^2 in the above bound is replaced by

  $$\sum_{k=1}^{m} \lambda_k^2 \max_{i=1,\dots,m} A_{ik}^2 \Big(\sum_{j=1,\dots,m} A_{jk} y_j \alpha_j \Big)^2,$$

  where A is the matrix of the normalized eigenvectors of the kernel matrix K, λ_k are the eigenvalues of the kernel matrix, and α the coefficients of the SVM solution.

- The trace bound of Bartlett and Mendelson (2001) which contains the eigenvalues of the kernel matrix: with probability at least $1-\delta$,

  $$\mathrm{R}_{\mathrm{true}} \leq \mathrm{R}_{\mathrm{emp}} + \mathit{Rademacher} + \sqrt{\frac{8\ln(2/\delta)}{m}},$$

  where the Rademacher complexity is given by $\mathit{Rademacher} \leq \sum_{i=1}^{m} \sqrt{\lambda_i/m}$. The quantity we compute in the experiments is $\mathrm{R}_{\mathrm{emp}} + \mathit{Rademacher}$.

- The compression scheme bound of Floyd and Warmuth (1995) using the sparsity of the SVM solution: with probability at least $1-\delta$,

  $$\mathrm{R}_{\mathrm{true}} \leq \frac{1}{m-s}\left(\ln\binom{m}{s} + \ln\frac{m^2}{\delta}\right),$$

  where s is the number of support vectors. In the experiments we computed the quantity $\ln\binom{m}{s}/(m-s)$. Note that if $s=m$ this bound is infinite. We omitted those cases from the plots.

- The sparse margin bound of Herbrich et al. (2000) which uses the size of the margin and the sparsity of the solution: with probability at least $1-\delta$,

  $$\mathrm{R}_{\mathrm{true}} \leq \frac{1}{m-s}\left(\kappa\ln\frac{em}{\kappa} + \ln\frac{m^2}{\delta}\right),$$

  where $\kappa = \min(\lceil \frac{R^2}{\rho^2}+1\rceil, d+1)$. For our experiments we compute the quantity $\frac{1}{m-s}\left(\kappa\ln\frac{em}{\kappa}\right)$. In case $s=m$ this bound is infinite and we omit those cases from the plots.

- The span estimate of Vapnik and Chapelle (2000), cf. also Opper and Winther (2000). This bound is different from all the other bounds as it estimates the leave-one-out error of the classifier. To achieve this, it bounds for each support vector how much the solution would change if this particular support vector were removed from the training set.

All experimental results shown below were obtained with training set size $m = 100$; the results for $m = 500$ are comparable. The interested reader can study those results, as well as many more plots which we cannot show here because they would use too much space, on the webpage http://www.kyb.tuebingen.mpg.de/bs/people/ule.

The goal of our first experiment is to use the compression coefficients to select the kernel width σ. In Figure 3 we study which of the five compression coefficients achieves the best results on this task. The plots in this figure were obtained as follows. For each training set, each parameter C and each compression coefficient we chose the kernel width σ for which the compression coefficient was minimal. Then we evaluated the test error for the chosen parameters on the test set and computed the mean over all runs on the different training sets. Plotted are the means of these test errors versus parameter C, as well as the means of the minimal true test errors.

We can observe that for nearly all data sets, the compression coefficients C_2, C_3, and C_4 yield better results than the simple coefficients C_1 and C_5. This can be explained by the fact that C_1 and C_5 only use one part of information (the size of the margin or the number of support vectors, respectively), while the other coefficients combine several parts of information (margin, shape of data, number of support vectors). When we compare coefficients C_3 and C_4 we observe that they have nearly identical values. This indicates that the gain we obtain by projecting into a smaller subspace using kernel PCA is outweighed by the additional information we have to transmit about this projection. As C_3 is simpler to compute, we thus prefer C_3 to C_4. From now on we want to evaluate the results of the most promising compression coefficients C_2 and C_3.

In Figure 4 we compare compression coefficients C_2 and C_3 to all the other model selection criteria. The plots were obtained in the same way as the ones above. We see that for most data sets, the performance of C_2 is rather good, and in most cases it is better than C_3. It performs nearly always comparable or better than the standard bounds, in particular it is nearly always better than the widely-used radius margin bound. Among all bounds, C_2 and the span bound achieve the best results. Comparing those two bounds shows that no one is superior to the other one: C_2 is better than the span bound on five data sets (abalone, banana, diabetis, german, usps), the span bound is better than C_2 on five data sets (image, ringnorm, splice, thyroid, waveform), and they achieve similar results on six data sets (breast-cancer, flare-solar, heart, titanic, twonorm, wisconsin).

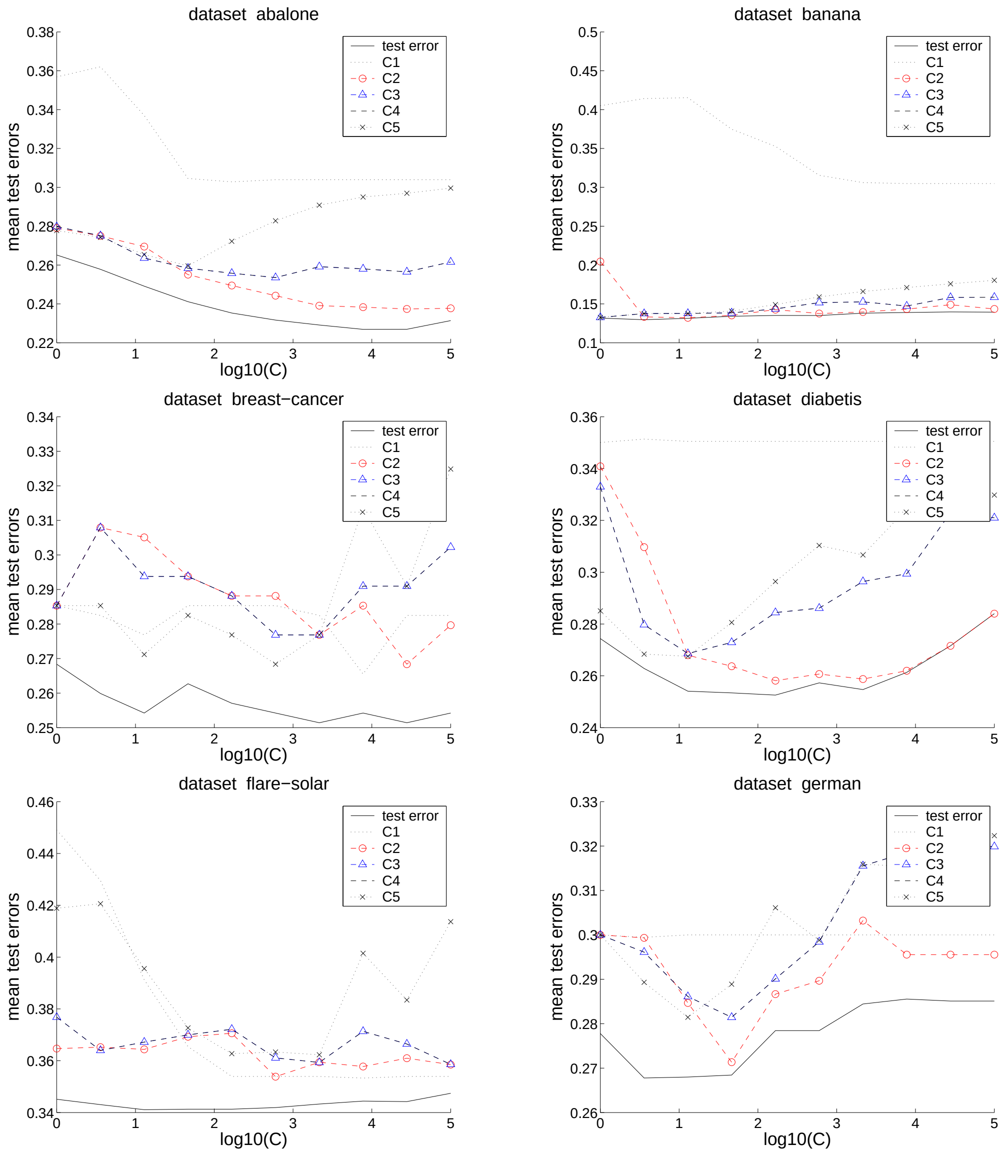

Figure 3: Comparison among the compression coefficients. For each training set, each soft margin parameter C and each compression coefficient we chose the kernel width σ for which the compression coefficient was minimal and evaluated the test error for the chosen parameters. Plotted are the mean values of the test errors over the different training sets, as well as the means of the true minimal test errors (this figure is continued on the next page).

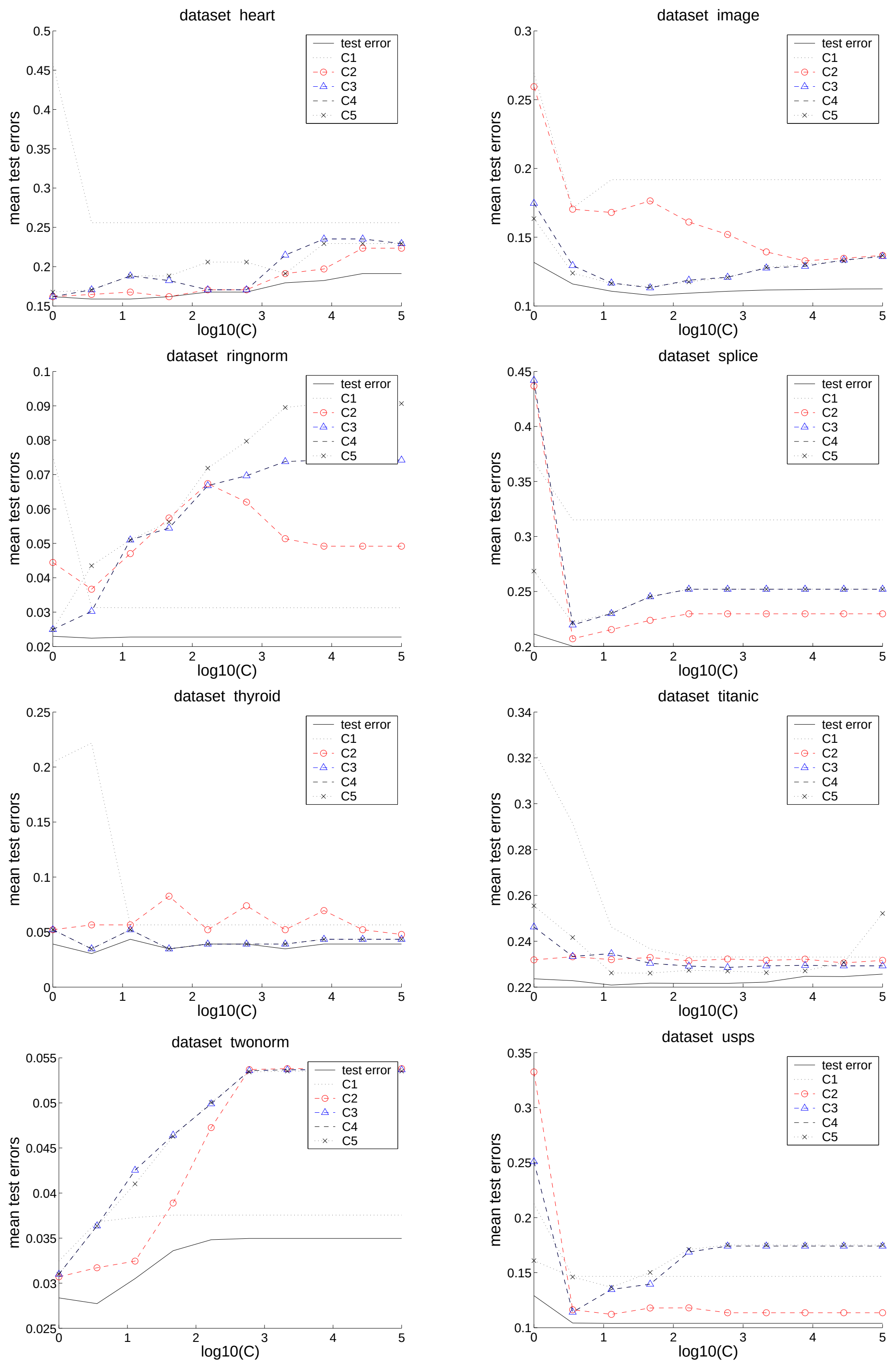

Figure 3, continued

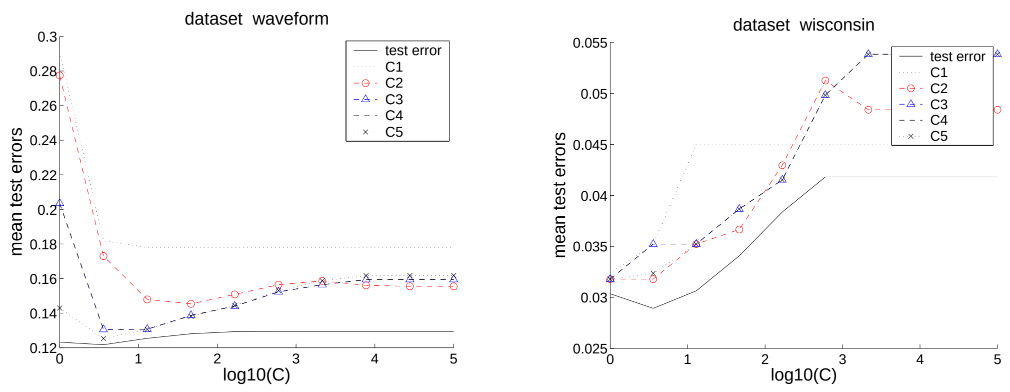

Figure 3, continued

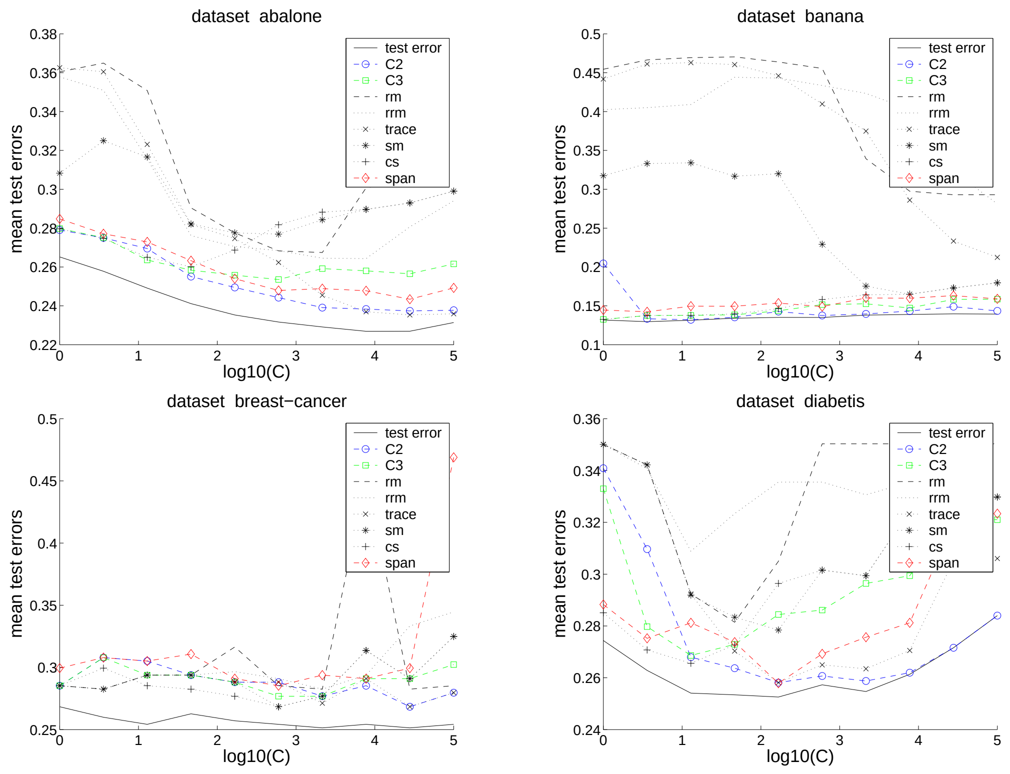

Figure 4: Comparison between C_2, C_3, and the other bounds. For each training set, each soft margin parameter C and each bound we chose the kernel width σ for which the bound was minimal and evalutated the test error for the chosen parameters. Plotted are the mean values of the test errors over the different training sets. In the legend we use the abbreviations rm = radius margin bound, rrm = rescaled radius margin bound, sm = sparse margin bound, and cs = compression scheme bound (continued on the next page).

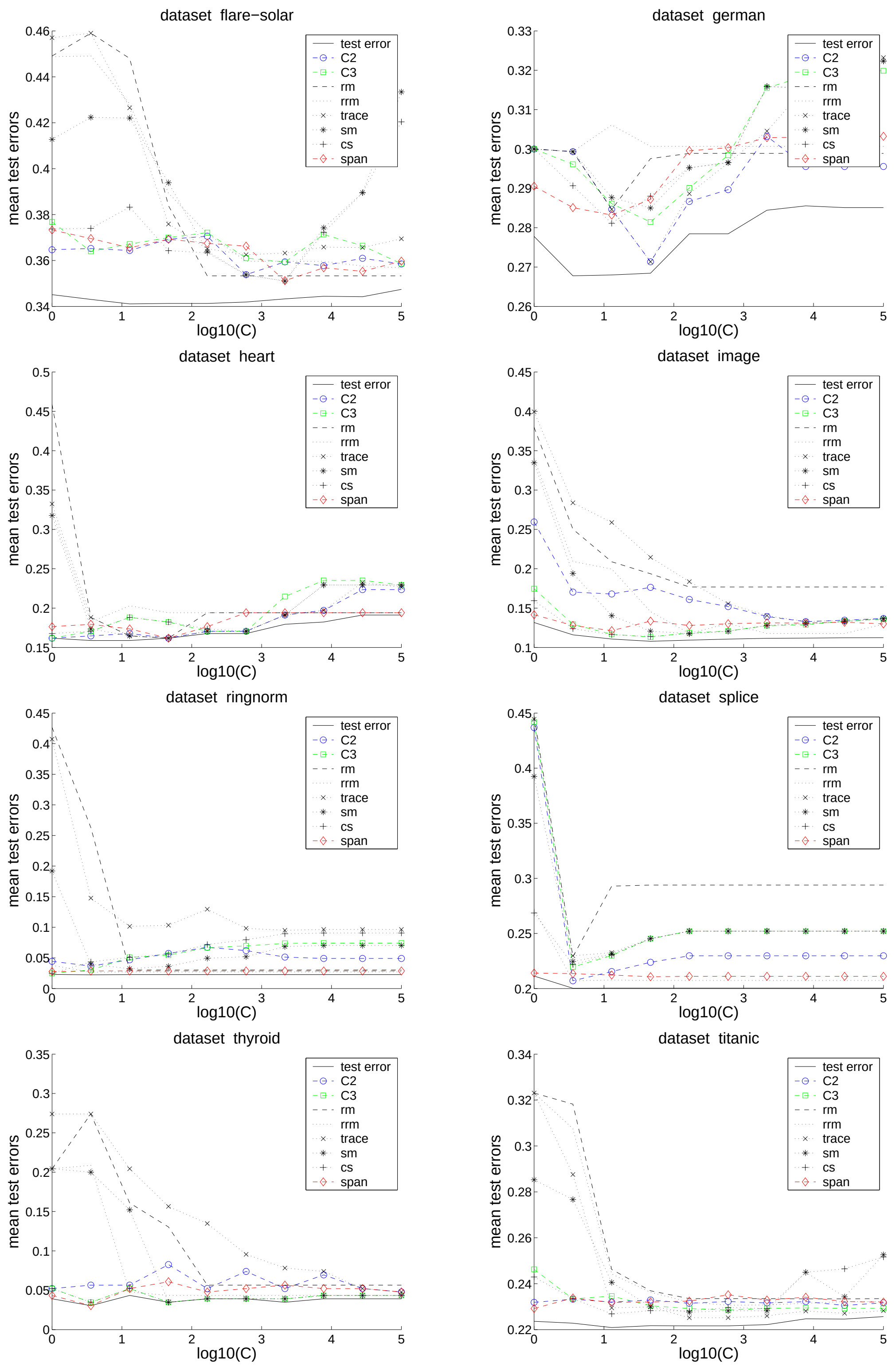

Figure 4, continued

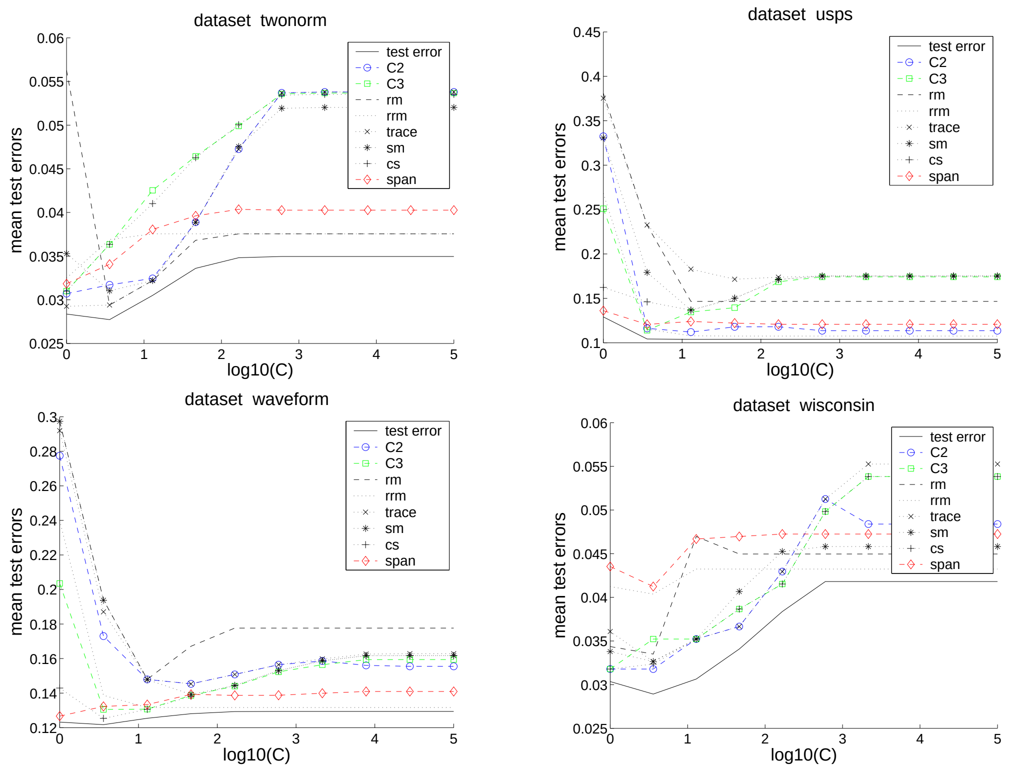

Figure 4, continued

The goal of the next experiment was not only to select the kernel width σ, but to find the best values for the kernel width σ and the soft margin parameter C simultaneously. Its results can be seen in Table 1, which was produced as follows. For each training set we chose the parameters σ and C for which the respective bounds were minimal, and evaluated the test error for the chosen parameters. Then we computed the means over the different training runs. The first column contains the mean value of the minimal test error. The other columns contain the offset by which the test errors selected by the different bounds are worse than the optimal test error. On most data sets, the radius margin bound, the rescaled radius margin bound, and the sparse margin bound perform rather poorly. Often their results are worse than those of the other bounds by one order of magnitude. Among the other bounds, C_2 and the span bound are the two superior bounds. Between those two bounds, there is a tendency towards the span bound in this experiment: C_2 beats the span bound on 6 data sets, the span bound beats C_2 on 10 data sets. Thus C_2 does not perform as good as the span bound, but it gets close.

To explain the good performance of the compression coefficients we now want to analyze their properties in more detail. As the compression coefficients are a sum of several terms it is a natural question which of the terms has the largest influence on the actual value of the sum. To answer this question we look at Figure 5, where we plotted the different parts of C_3: the term $(s+1)\log m$ cor-

data set	test error	C_2	C_3	span	rm	rrm	trace	sm	cs
abalone	0.224	0.017	0.036	0.029	0.137	0.134	0.012	0.084	0.072
banana	0.124	0.021	0.024	0.020	0.347	0.278	0.141	0.202	0.047
breast-cancer	0.251	0.062	0.065	0.209	0.034	0.034	0.028	0.034	0.042
diabetis	0.247	0.012	0.049	0.025	0.103	0.103	0.059	0.103	0.080
flare-solar	0.339	0.026	0.027	0.020	0.120	0.110	0.030	0.101	0.101
german	0.263	0.037	0.042	0.026	0.037	0.037	0.060	0.037	0.044
heart	0.156	0.015	0.015	0.021	0.303	0.168	0.071	0.159	0.053
image	0.105	0.074	0.028	0.023	0.275	0.235	0.031	0.183	0.028
ringnorm	0.021	0.054	0.052	0.007	0.405	0.017	0.075	0.178	0.068
splice	0.198	0.039	0.053	0.016	0.249	0.035	0.054	0.084	0.053
thyroid	0.026	0.030	0.017	0.026	0.178	0.178	0.022	0.178	0.017
titanic	0.220	0.010	0.010	0.012	0.103	0.103	0.008	0.065	0.041
twonorm	0.027	0.016	0.026	0.007	0.029	0.006	0.027	0.009	0.026
usps	0.103	0.075	0.071	0.020	0.278	0.141	0.072	0.190	0.072
waveform	0.118	0.047	0.040	0.019	0.179	0.120	0.045	0.179	0.042
wisconsin	0.028	0.007	0.011	0.015	0.006	0.013	0.027	0.005	0.008

Table 1: Model selection results for selecting the kernel width σ and the soft margin parameter C simultaneously. For each training set and each bound we chose the parameters σ and C for which the bound was minimal, and evaluated the test error for the chosen parameters. Shown are the mean values of the test errors over the different training sets. The first column contains the value of the test error. The other columns contain the offset by which the test errors achieved by the different bounds are worse than the optimal test error.

responding to the support vector information, the term $\log^* n + \log n$ corresponding to the position of the codebook vector, the term $(r+1)\log m + r$ corresponding to the points inside the rotation region, and the term $(l+1)\log m$ corresponding to the information on the misclassified points outside the rotation region. For each fixed soft margin parameter C we chose the kernel width σ where the value of the compression coefficient C_3 is minimal. For this value of σ, we plotted the means of the different terms. Here we only show the plots for compression coefficient C_3 (as C_3 has more different terms than C_2) on the first six data sets (in alphabetical order). The plots on the other data sets, as well as the plots for C_2, are very similar to the ones we show. We find that all terms (except the term "misclass. inside" which is negligible) are of the same order of magnitude, with no term consistently dominating the other ones. This behavior is attractive as it shows that all different parts of information have substantial influence on the value of the compression coefficient.

Finally we want to study the shapes of the curves of the different bounds. As we use the values of the bounds to predict the qualitative behavior of the test error it is important that the shapes of the bounds' curves are similar to the shape of the test error curve. In Figure 6 we plot those shapes for the compression coefficients C_2, C_3, and the other bounds versus the kernel width σ. We show the plots for every second value of C (to cover the whole range of values of C we used) and for the first six data sets (in alphabetical order). First of all we can observe that the value of the compression coefficient is often larger than 1. This is also the case for several of the other bounds, and is due to the fact that most bounds only yield nontrivial results for very large sample sizes. This need not be a problem for applications as we use the bounds to predict the qualitative behavior of the test error, not the quantitative one. Secondly, the compression scheme and the sparse margin bound suffer from the fact that they only attain finite values when the number of support vectors is smaller than the number of training vectors. Among all bounds, only C_2, C_3 and the span bound seem to be able to predict the shape of the test error curve.

The main conclusions we can draw from all experimental results is that in all three tasks (predicting the shape of the test error curve, choosing parameter σ, choosing parameters σ and C) the span bound and compression coefficient C_2 have the best performance among all bounds, where none of the two bounds is clearly superior to the other one. The latter fact is also remarkable for the following reason. All considered bounds apart from the span bound use the capacity of the model class to bound the expected risk of the classifier. The span bound on the other hand is a clever way of computing an upper bound on the leave-one-out error of the classifier, which is known to be an almost unbiased estimator of the true risk. Thus the methods by which those bounds are derived are intrinsically different. Our results now show that the bounds derived by studying the size of the model class can achieve results in practice that are comparable to using the state of the art span bound.

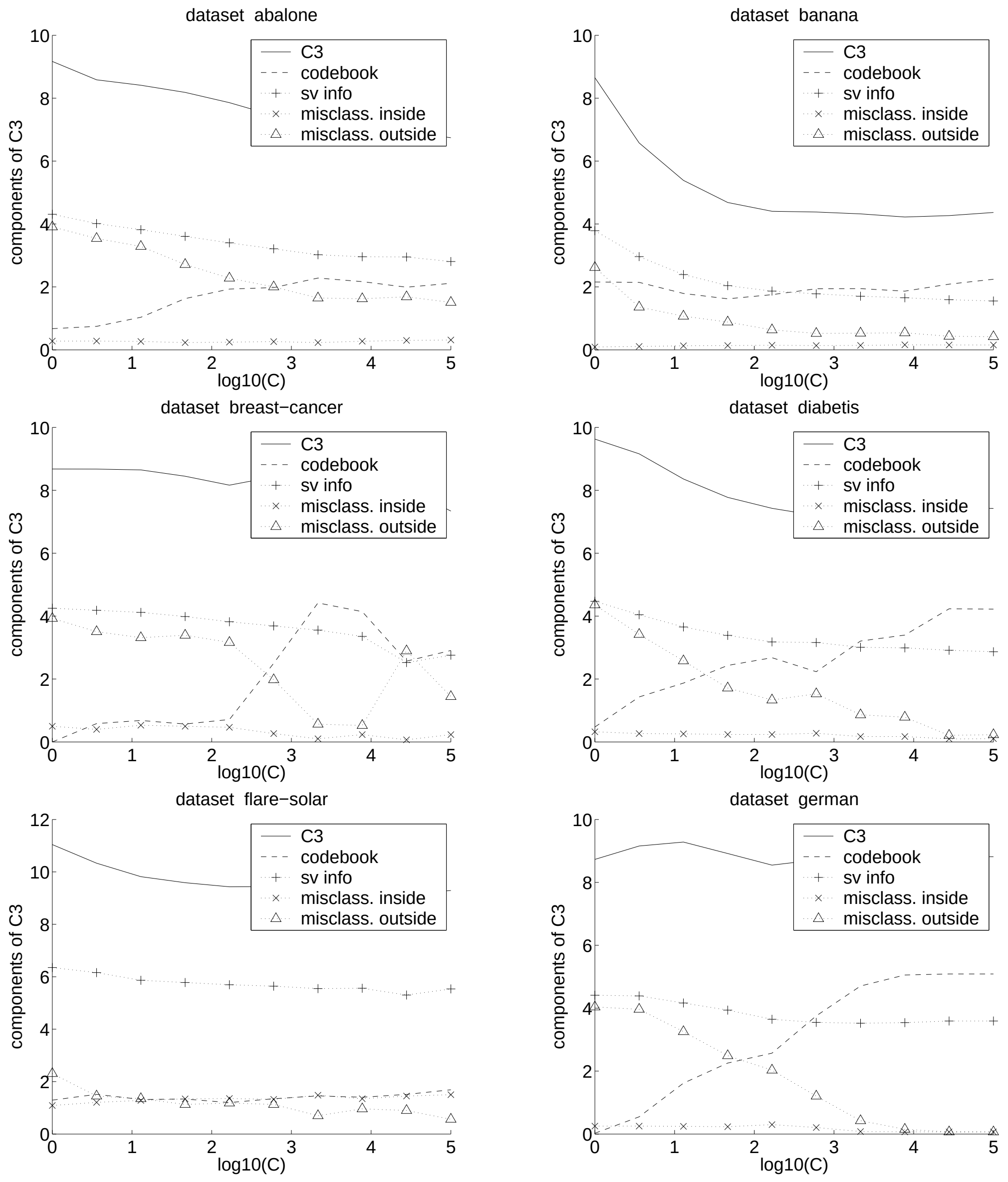

Figure 5. Here we study the relationship between the different components of C_3. We plot the lengths of the codes for the support vector information (“sv info”), the position of the codebook vector (“codebook”), the information on the points inside the rotation region (“misclass. inside”), and the information about the misclassified points outside the rotation region (“misclass. outside”).

4. Conclusions

We derived five compression coefficients for SVMs which combine information on the geometry of the training data in the feature space with information about geometry and sparsity of the classifier. In our model selection experiments it turned out that the compression coefficients can be readily used to predict the parameters where the test error is small. Our favorite compression coefficient is

C_2 because it is easy to compute and yields good results in the experiments. The results it achieves are comparable to those of the state of the art span bound. The theoretical justification for using compression coefficients are the generalization bounds we cited in Section 2. They were proved in an abstract coding theoretic setting. We now derived methods to apply these bounds in practical applications. This shows that the connection between information theory and learning can be exploited in every-day machine learning applications.

Acknowledgements

We thank the anonymous reviewers for comments that significantly improved the quality of the manuscript.

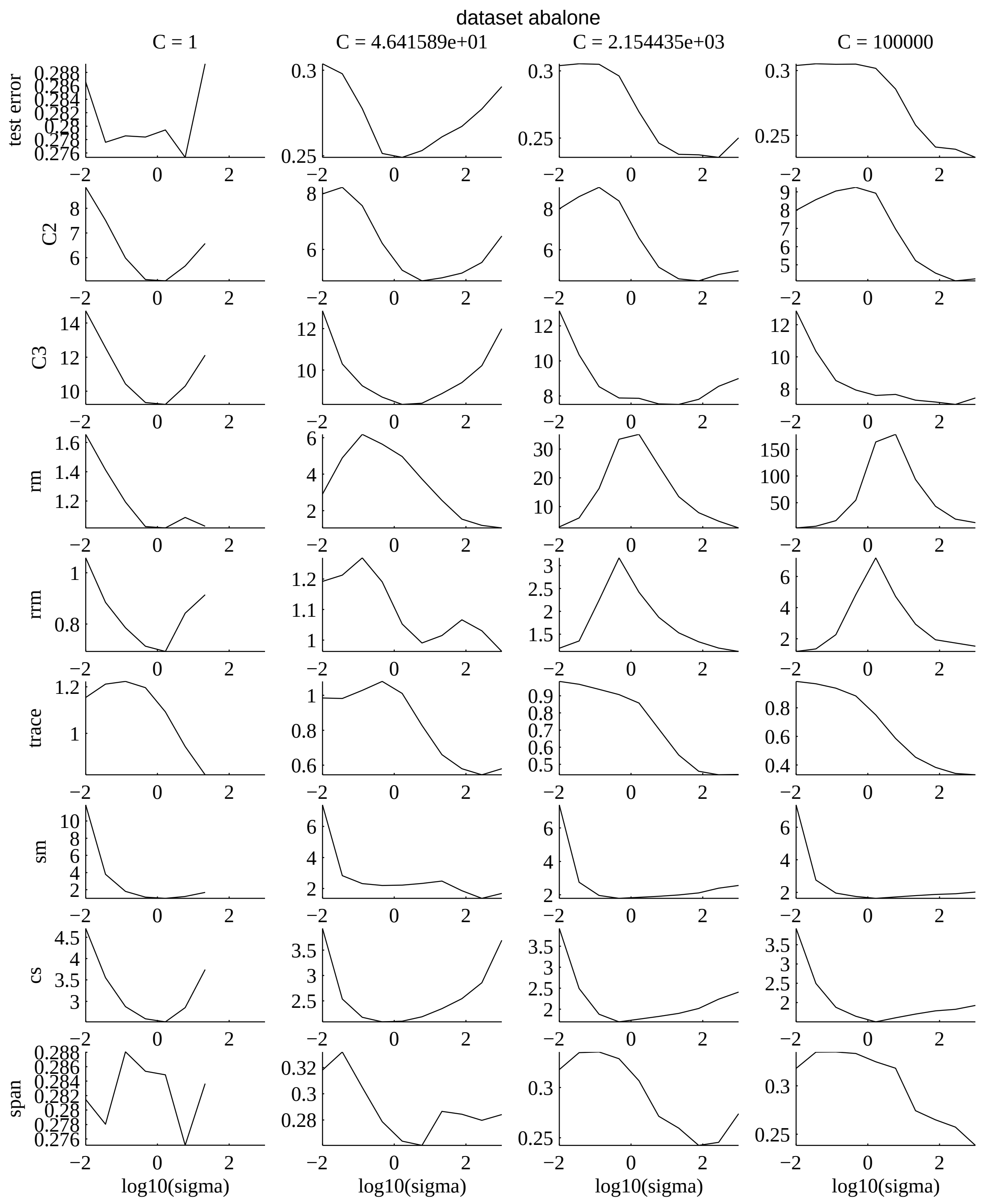

Figure 6: Shapes of curves. Plotted are the mean values of the bounds themselves over the different training runs versus the kernel width σ, for fixed values of the soft margin parameter C.

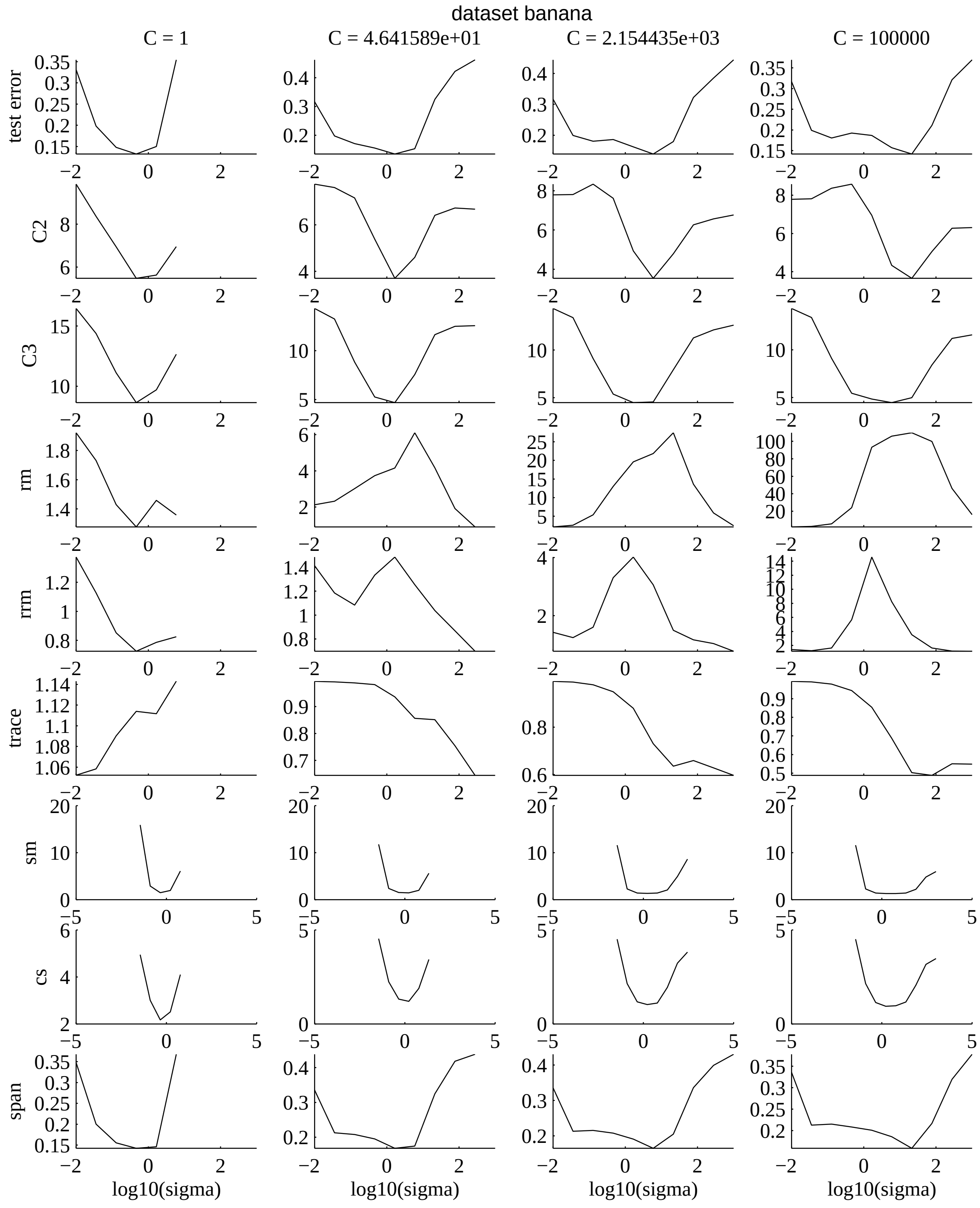

Figure 6, continued

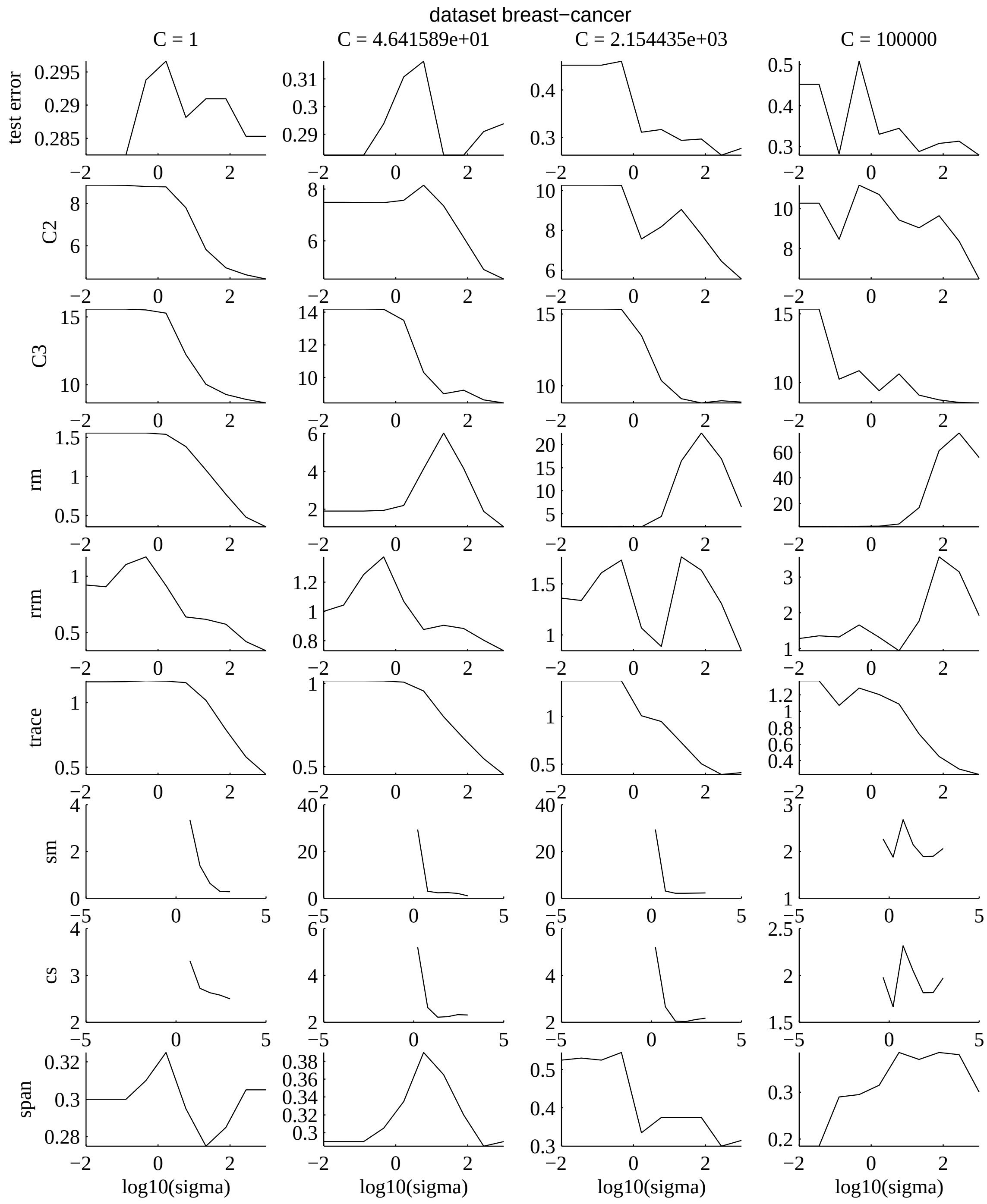

Figure 6, continued

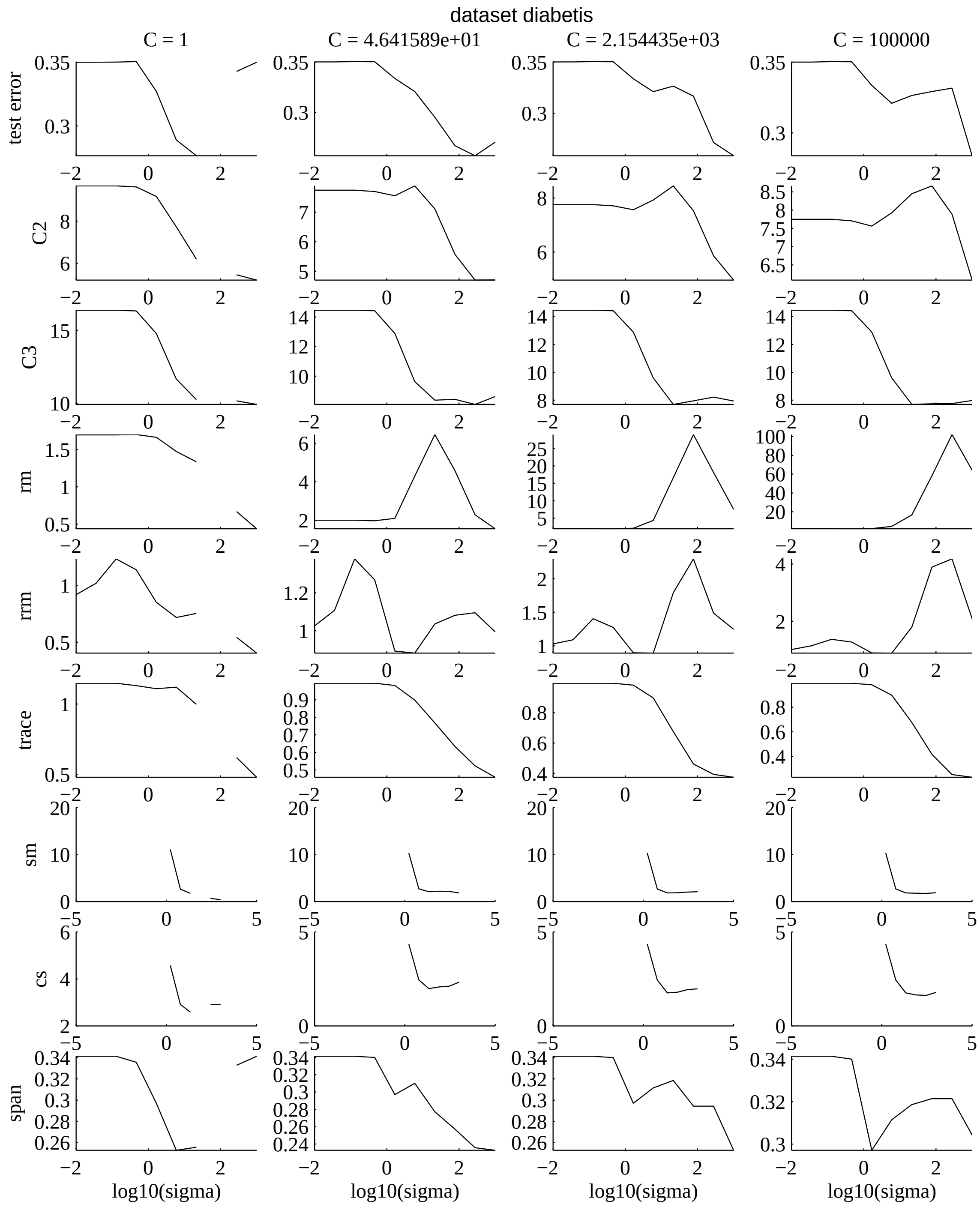

Figure 6, continued

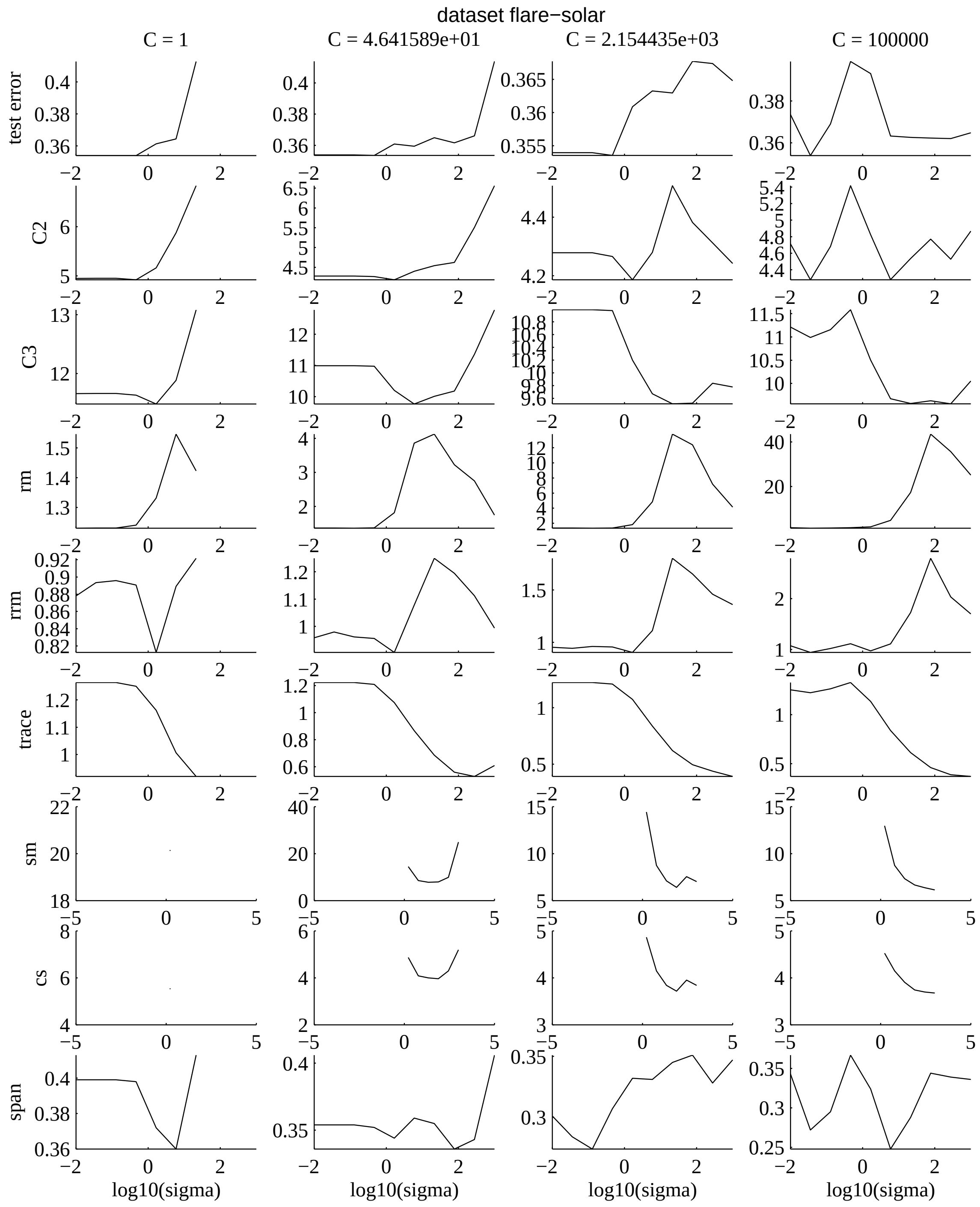

Figure 6, continued

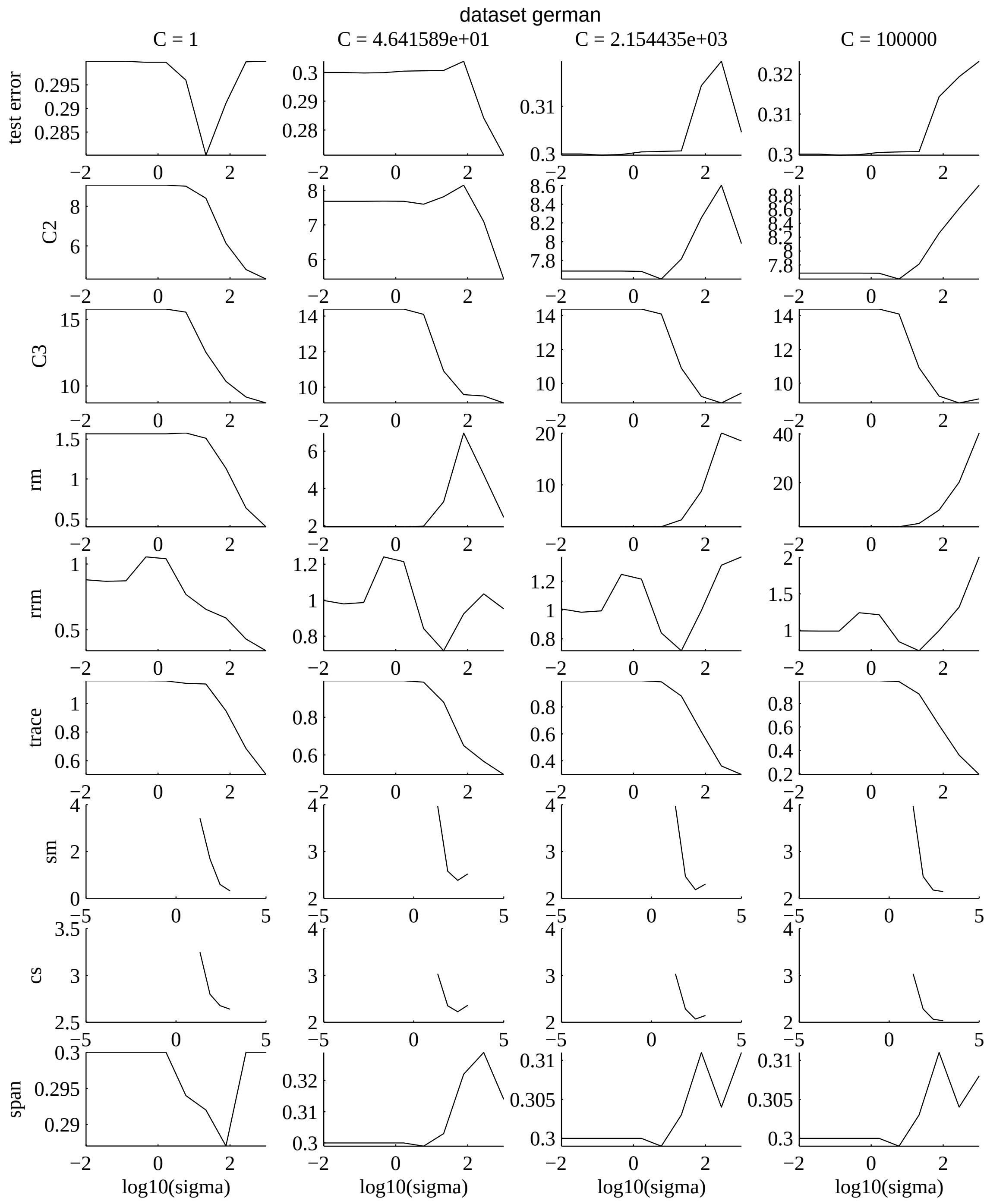

Figure 6, continued

References

A. Barron, J. Rissanen, and B. Yu. The minimum description length principle in coding and modeling. *IEEE Transactions on Information Theory*, 44(6):2743 – 2760, 1998.

P. Bartlett and S. Mendelson. Rademacher and Gaussian complexities: Risk bounds and structural results. In D. Helmbold and B. Williamson, editors, *Proceedings of the 14th annual conference on Computational Learning Theory*, pages 273–288, 2001.

P. L. Bartlett and J. Shawe-Taylor. Generalization performance of support vector machines and other pattern classifiers. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 43–54. MIT Press, 1999.

A. Blum and J. Langford. PAC-MDL bounds. In B. Schölkopf and M.K. Warmuth, editors, *Learning Theory and Kernel Machines*, pages 344–357. 16th Annual Conference on Learning Theory, Springer, 2003.

O. Chapelle and V. Vapnik. Model selection for support vector machines. In S. A. Solla, T. K. Leen, and K.-R. Müller, editors, *Advances in Neural Information Processing Systems 12*. MIT Press, 2000.

T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley, 1991.

S. Floyd and M. K. Warmuth. Sample compression, learnability, and the Vapnik-Chervonenkis dimension. *Machine Learning*, 21(3):269–304, 1995.

M. H. Hansen and B. Yu. Model selection and the principle of minimum description length. *Journal of the American Statistical Association*, 96(454):746–774, 2001.

R. Herbrich, T. Graepel, and J. Shawe-Taylor. Sparsity vs. large margins for linear classifiers. In N. Cesa-Bianchi and S. Goldman, editors, *Proceedings of the Thirteenth Annual Conference on Computational Learning Theory*, pages 304–308. Morgan Kaufmann, 2000.

D. McAllester. Some PAC–Bayesian theorems. *Machine Learning*, 37(3):355–363, 1999.

M. Opper and O. Winther. Gaussian processes and SVM: Mean field and leave-one-out. In A. J. Smola, P. L. Bartlett, B. Schölkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 311–326. MIT Press, 2000.

G. Rätsch, T. Onoda, and K.-R. Müller. Soft margins for AdaBoost. *Machine Learning*, 42(3): 287–320, March 2001.

B. Schölkopf and A. Smola. *Learning with Kernels*. MIT Press, 2002.

V. Vapnik. *Statistical Learning Theory*. Wiley, 1998.

V. Vapnik and O. Chapelle. Bounds on error expectation for support vector machines. *Neural Computation*, 12(9):2013–2036, 2000.