

A User's Guide to Support Vector Machines

Asa Ben-Hur

Department of Computer Science
Colorado State University

Jason Weston

NEC Labs America
Princeton, NJ 08540 USA

Abstract

The Support Vector Machine (SVM) is a widely used classifier. And yet, obtaining the best results with SVMs requires an understanding of their workings and the various ways a user can influence their accuracy. We provide the user with a basic understanding of the theory behind SVMs and focus on their use in practice. We describe the effect of the SVM parameters on the resulting classifier, how to select good values for those parameters, data normalization, factors that affect training time, and software for training SVMs.

1 Introduction

The Support Vector Machine (SVM) is a state-of-the-art classification method introduced in 1992 by Boser, Guyon, and Vapnik [1]. The SVM classifier is widely used in bioinformatics (and other disciplines) due to its high accuracy, ability to deal with high-dimensional data such as gene expression, and flexibility in modeling diverse sources of data [2].

SVMs belong to the general category of *kernel methods* [4, 5]. A kernel method is an algorithm that depends on the data only through dot-products. When this is the case, the dot product can be replaced by a *kernel function* which computes a dot product in some possibly high dimensional feature space. This has two advantages: First, the ability to generate non-linear decision boundaries using methods designed for linear classifiers. Second, the use of kernel functions allows the user to apply a classifier to data that have no obvious fixed-dimensional vector space representation. The prime example of such data in bioinformatics are sequence, either DNA or protein, and protein structure.

Using SVMs effectively requires an understanding of how they work. When training an SVM the practitioner needs to make a number of decisions: how to preprocess the data, what kernel to use, and finally, setting the parameters of the SVM and the kernel. Uninformed choices may result in severely reduced performance [6]. We aim to provide the user with an intuitive understanding of these choices and provide general usage guidelines. All the examples shown were generated using the PyML machine learning environment, which focuses on kernel methods and SVMs, and is available at <http://pyml.sourceforge.net>. PyML is just one of several software packages that provide SVM training methods; an incomplete listing of these is provided in Section 9. More information is found on the Machine Learning Open Source Software website <http://mloss.org> and a related paper [7].

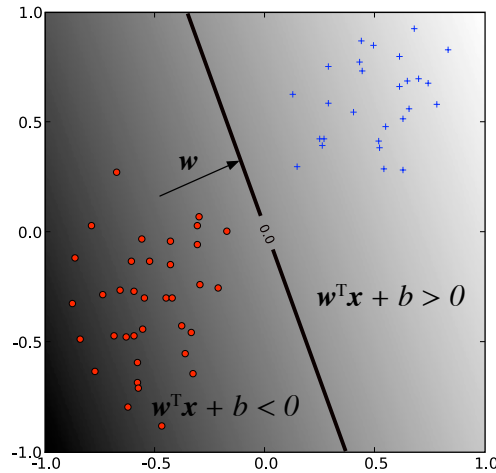


Figure 1: A linear classifier. The decision boundary (points \mathbf{x} such that $\mathbf{w}^T \mathbf{x} + b = 0$) divides the plane into two sets depending on the sign of $\mathbf{w}^T \mathbf{x} + b$.

2 Preliminaries: Linear Classifiers

Support vector machines are an example of a linear two-class classifier. This section explains what that means. The data for a two class learning problem consists of objects labeled with one of two labels corresponding to the two classes; for convenience we assume the labels are $+1$ (positive examples) or -1 (negative examples). In what follows boldface \mathbf{x} denotes a vector with components x_i . The notation \mathbf{x}_i will denote the i^{th} vector in a dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$, where y_i is the label associated with \mathbf{x}_i . The objects \mathbf{x}_i are called *patterns* or *examples*. We assume the examples belong to some set \mathcal{X} . Initially we assume the examples are vectors, but once we introduce kernels this assumption will be relaxed, at which point they could be any continuous/discrete object (e.g. a protein/DNA sequence or protein structure).

A key concept required for defining a linear classifier is the *dot product* between two vectors, also referred to as an *inner product* or *scalar product*, defined as $\mathbf{w}^T \mathbf{x} = \sum_i w_i x_i$. A linear classifier is based on a linear *discriminant function* of the form

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b. \quad (1)$$

The vector \mathbf{w} is known as the *weight vector*, and b is called the bias. Consider the case $b = 0$ first. The set of points \mathbf{x} such that $\mathbf{w}^T \mathbf{x} = 0$ are all points that are perpendicular to \mathbf{w} and go through the origin — a line in two dimensions, a plane in three dimensions, and more generally, a *hyperplane*. The bias b translates the hyperplane away from the origin. The hyperplane

$$\{\mathbf{x} : f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b = 0\} \quad (2)$$

divides the space into two: the sign of the discriminant function $f(\mathbf{x})$ denotes the side of the hyperplane a point is on (see 1). The boundary between regions classified as positive and negative

is called the *decision boundary* of the classifier. The decision boundary defined by a hyperplane is said to be *linear* because it is linear in the input examples (c.f. Equation 1). A classifier with a linear decision boundary is called a linear classifier. Conversely, when the decision boundary of a classifier depends on the data in a non-linear way (see Figure 4 for example) the classifier is said to be non-linear.

3 Kernels: from Linear to Non-Linear Classifiers

In many applications a non-linear classifier provides better accuracy. And yet, linear classifiers have advantages, one of them being that they often have simple training algorithms that scale well with the number of examples [9, 10]. This begs the question: Can the machinery of linear classifiers be extended to generate non-linear decision boundaries? Furthermore, can we handle domains such as protein sequences or structures where a representation in a fixed dimensional vector space is not available?

The naive way of making a non-linear classifier out of a linear classifier is to map our data from the input space \mathcal{X} to a feature space \mathcal{F} using a non-linear function $\phi : \mathcal{X} \rightarrow \mathcal{F}$. In the space \mathcal{F} the discriminant function is:

$$f(\mathbf{x}) = \mathbf{w}^\top \phi(\mathbf{x}) + b. \quad (3)$$

Example 1 Consider the case of a two dimensional input-space with the mapping

$$\phi(\mathbf{x}) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)^\top,$$

which represents a vector in terms of all degree-2 monomials. In this case

$$\mathbf{w}^\top \phi(\mathbf{x}) = w_1x_1^2 + \sqrt{2}w_2x_1x_2 + w_3x_2^2,$$

resulting in a decision boundary for the classifier, $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b = 0$, which is a conic section (e.g., an ellipse or hyperbola). The added flexibility of considering degree-2 monomials is illustrated in Figure 4 in the context of SVMs.

The approach of explicitly computing non-linear features does not scale well with the number of input features: when applying the mapping from the above example the dimensionality of the feature space \mathcal{F} is quadratic in the dimensionality of the original space. This results in a quadratic increase in memory usage for storing the features and a quadratic increase in the time required to compute the discriminant function of the classifier. This quadratic complexity is feasible for low dimensional data; but when handling gene expression data that can have thousands of dimensions, quadratic complexity in the number of dimensions is not acceptable. Kernel methods solve this issue by avoiding the step of explicitly mapping the data to a high dimensional feature-space. Suppose the weight vector can be expressed as a linear combination of the training examples, i.e. $\mathbf{w} = \sum_{i=1}^n \alpha_i \mathbf{x}_i$. Then:

$$f(\mathbf{x}) = \sum_{i=1}^n \alpha_i \mathbf{x}_i^\top \mathbf{x} + b.$$

In the feature space, \mathcal{F} this expression takes the form:

$$f(\mathbf{x}) = \sum_{i=1}^n \alpha_i \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}) + b.$$

The representation in terms of the variables α_i is known as the *dual* representation of the decision boundary. As indicated above, the feature space \mathcal{F} may be high dimensional, making this trick impractical unless the kernel function $k(\mathbf{x}, \mathbf{x}')$ defined as

$$k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^\top \phi(\mathbf{x}')$$

can be computed efficiently. In terms of the kernel function the discriminant function is:

$$f(\mathbf{x}) = \sum_{i=1}^n \alpha_i k(\mathbf{x}, \mathbf{x}_i) + b. \quad (4)$$

Example 2 Let's go back to the example of $\phi(\mathbf{x}) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)^\top$, and show the kernel associated with this mapping:

$$\begin{aligned} \phi(\mathbf{x})^\top \phi(\mathbf{z}) &= (x_1^2, \sqrt{2}x_1x_2, x_2^2)^\top (z_1^2, \sqrt{2}z_1z_2, z_2^2) \\ &= x_1^2z_1^2 + 2x_1x_2z_1z_2 + x_2^2z_2^2 \\ &= (\mathbf{x}^\top \mathbf{z})^2. \end{aligned}$$

This shows that the kernel can be computed without explicitly computing the mapping ϕ .

The above example leads us to the definition of the degree- d polynomial kernel:

$$k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^\top \mathbf{x}' + 1)^d. \quad (5)$$

The feature space for this kernel consists of all monomials up to degree d , i.e. features of the form: $x_1^{d_1} x_2^{d_2} \cdots x_m^{d_m}$ where $\sum_{i=1}^m d_i \leq d$. The kernel with $d = 1$ is the *linear kernel*, and in that case the additive constant in Equation 5 is usually omitted. The increasing flexibility of the classifier as the degree of the polynomial is increased is illustrated in Figure 4. The other widely used kernel is the Gaussian kernel defined by:

$$k(\mathbf{x}, \mathbf{x}') = \exp(-\gamma \|\mathbf{x} - \mathbf{x}'\|^2), \quad (6)$$

where $\gamma > 0$ is a parameter that controls the width of Gaussian. It plays a similar role as the degree of the polynomial kernel in controlling the flexibility of the resulting classifier (see Figure 5).

We saw that a linear decision boundary can be “kernelized”, i.e. its dependence on the data is only through dot products. In order for this to be useful, the training algorithms needs to be kernelizable as well. It turns out that a large number of machine learning algorithms can be expressed using kernels — including ridge regression, the perceptron algorithm, and SVMs [5, 8].

4 Large Margin Classification

In what follows we use the term *linearly separable* to denote data for which there exists a linear decision boundary that separates positive from negative examples (see Figure 2). Initially we will assume linearly separable data, and later indicate how to handle data that is not linearly separable.

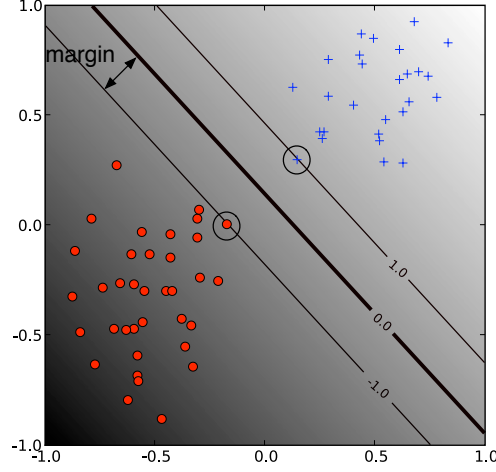


Figure 2: A linear SVM. The circled data points are the *support vectors*—the examples that are closest to the decision boundary. They determine the margin with which the two classes are separated.

4.1 The Geometric Margin

In this section we define the notion of a margin. For a given hyperplane we denote by \mathbf{x}_+ (\mathbf{x}_-) the closest point to the hyperplane among the positive (negative) examples. The *norm* of a vector \mathbf{w} denoted by $\|\mathbf{w}\|$ is its length, and is given by $\sqrt{\mathbf{w}^\top \mathbf{w}}$. A unit vector $\hat{\mathbf{w}}$ in the direction of \mathbf{w} is given by $\mathbf{w}/\|\mathbf{w}\|$ and has $\|\hat{\mathbf{w}}\| = 1$. From simple geometric considerations the margin of a hyperplane f with respect to a dataset \mathcal{D} can be seen to be:

$$m_{\mathcal{D}}(f) = \frac{1}{2} \hat{\mathbf{w}}^\top (\mathbf{x}_+ - \mathbf{x}_-), \quad (7)$$

where $\hat{\mathbf{w}}$ is a unit vector in the direction of \mathbf{w} , and we assume that \mathbf{x}_+ and \mathbf{x}_- are equidistant from the decision boundary i.e.

$$\begin{aligned} f(\mathbf{x}_+) &= \mathbf{w}^\top \mathbf{x}_+ + b = a \\ f(\mathbf{x}_-) &= \mathbf{w}^\top \mathbf{x}_- + b = -a \end{aligned} \quad (8)$$

for some constant $a > 0$. Note that multiplying our data points by a fixed number will increase the margin by the same amount, whereas in reality, the margin hasn't really changed — we just changed the “units” with which we measure it. To make the geometric margin meaningful we fix the value of the decision function at the points closest to the hyperplane, and set $a = 1$ in Eqn. (8). Adding the two equations and dividing by $\|\mathbf{w}\|$ we obtain:

$$m_{\mathcal{D}}(f) = \frac{1}{2} \hat{\mathbf{w}}^\top (\mathbf{x}_+ - \mathbf{x}_-) = \frac{1}{\|\mathbf{w}\|}. \quad (9)$$

4.2 Support Vector Machines

Now that we have the concept of a margin we can formulate the maximum margin classifier. We will first define the hard margin SVM, applicable to a linearly separable dataset, and then modify it to handle non-separable data. The maximum margin classifier is the discriminant function that maximizes the geometric margin $1/\|\mathbf{w}\|$ which is equivalent to minimizing $\|\mathbf{w}\|^2$. This leads to the following constrained optimization problem:

$$\begin{aligned} & \underset{\mathbf{w}, b}{\text{minimize}} && \frac{1}{2} \|\mathbf{w}\|^2 \\ & \text{subject to:} && y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 \quad i = 1, \dots, n. \end{aligned} \quad (10)$$

The constraints in this formulation ensure that the maximum margin classifier classifies each example correctly, which is possible since we assumed that the data is linearly separable. In practice, data is often not linearly separable; and even if it is, a greater margin can be achieved by allowing the classifier to misclassify some points. To allow errors we replace the inequality constraints in Eqn. (10) with

$$y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i \quad i = 1, \dots, n,$$

where $\xi_i \geq 0$ are *slack variables* that allow an example to be in the margin ($0 \leq \xi_i \leq 1$, also called a margin error) or to be misclassified ($\xi_i > 1$). Since an example is misclassified if the value of its slack variable is greater than 1, $\sum_i \xi_i$ is a bound on the number of misclassified examples. Our objective of maximizing the margin, i.e. minimizing $\frac{1}{2} \|\mathbf{w}\|^2$ will be augmented with a term $C \sum_i \xi_i$ to penalize misclassification and margin errors. The optimization problem now becomes:

$$\begin{aligned} & \underset{\mathbf{w}, b}{\text{minimize}} && \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^n \xi_i \\ & \text{subject to:} && y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0. \end{aligned} \quad (11)$$

The constant $C > 0$ sets the relative importance of maximizing the margin and minimizing the amount of slack. This formulation is called the *soft-margin SVM*, and was introduced by Cortes and Vapnik [11]. Using the method of Lagrange multipliers, we can obtain the *dual* formulation which is expressed in terms of variables α_i [11, 5, 8]:

$$\begin{aligned} & \underset{\alpha}{\text{maximize}} && \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n y_i y_j \alpha_i \alpha_j \mathbf{x}_i^\top \mathbf{x}_j \\ & \text{subject to:} && \sum_{i=1}^n y_i \alpha_i = 0, \quad 0 \leq \alpha_i \leq C. \end{aligned} \quad (12)$$

The dual formulation leads to an expansion of the weight vector in terms of the input examples:

$$\mathbf{w} = \sum_{i=1}^n y_i \alpha_i \mathbf{x}_i. \quad (13)$$

The examples \mathbf{x}_i for which $\alpha_i > 0$ are those points that are on the margin, or within the margin when a soft-margin SVM is used. These are the so-called *support vectors*. The expansion in terms of the support vectors is often sparse, and the level of sparsity (fraction of the data serving as support vectors) is an upper bound on the error rate of the classifier [5].

The dual formulation of the SVM optimization problem depends on the data only through dot products. The dot product can therefore be replaced with a non-linear kernel function, thereby

performing large margin separation in the feature-space of the kernel (see Figures 4 and 5). The SVM optimization problem was traditionally solved in the dual formulation, and only recently it was shown that the primal formulation, Equation (11), can lead to efficient kernel-based learning [12]. Details on software for training SVMs is provided in Section 9.

5 Understanding the Effects of SVM and Kernel Parameters

Training an SVM finds the large margin hyperplane, i.e. sets the parameters α_i and b (c.f. Equation 4). The SVM has another set of parameters called *hyperparameters*: The soft margin constant, C , and any parameters the kernel function may depend on (width of a Gaussian kernel or degree of a polynomial kernel). In this section we illustrate the effect of the hyperparameters on the decision boundary of an SVM using two-dimensional examples.

We begin our discussion of hyperparameters with the soft-margin constant, whose role is illustrated in Figure 3. For a large value of C a large penalty is assigned to errors/margin errors. This is seen in the left panel of Figure 3, where the two points closest to the hyperplane affect its orientation, resulting in a hyperplane that comes close to several other data points. When C is decreased (right panel of the figure), those points become margin errors; the hyperplane’s orientation is changed, providing a much larger margin for the rest of the data.

Kernel parameters also have a significant effect on the decision boundary. The degree of the polynomial kernel and the width parameter of the Gaussian kernel control the flexibility of the resulting classifier (Figures 4 and 5). The lowest degree polynomial is the linear kernel, which is not sufficient when a non-linear relationship between features exists. For the data in Figure 4 a degree-2 polynomial is already flexible enough to discriminate between the two classes with a sizable margin. The degree-5 polynomial yields a similar decision boundary, albeit with greater curvature.

Next we turn our attention to the Gaussian kernel: $k(\mathbf{x}, \mathbf{x}') = \exp(-\gamma \|\mathbf{x} - \mathbf{x}'\|^2)$. This expression is essentially zero if the distance between \mathbf{x} and \mathbf{x}' is much larger than $1/\sqrt{\gamma}$; i.e. for a fixed \mathbf{x}' it is localized to a region around \mathbf{x}' . The support vector expansion, Equation (4) is thus a sum of Gaussian “bumps” centered around each support vector. When γ is small (top left panel in Figure 5) a given data point \mathbf{x} has a non-zero kernel value relative to any example in the set of support vectors. Therefore the whole set of support vectors affects the value of the discriminant function at \mathbf{x} , resulting in a smooth decision boundary. As γ is increased the locality of the support vector expansion increases, leading to greater curvature of the decision boundary. When γ is large the value of the discriminant function is essentially constant outside the close proximity of the region where the data are concentrated (see bottom right panel in Figure 5). In this regime of the γ parameter the classifier is clearly overfitting the data.

As seen from the examples in Figures 4 and 5 the γ parameter of the Gaussian kernel and the degree of polynomial kernel determine the flexibility of the resulting SVM in fitting the data. If this complexity parameter is too large, overfitting will occur (bottom panels in Figure 5).

A question frequently posed by practitioners is “which kernel should I use for my data?” There are several answers to this question. The first is that it is, like most practical questions in machine learning, data-dependent, so several kernels should be tried. That being said, we typically follow the following procedure: Try a linear kernel first, and then see if we can improve on its performance using a non-linear kernel. The linear kernel provides a useful baseline, and in many bioinformatics applications provides the best results: The flexibility of the Gaussian and polynomial kernels often

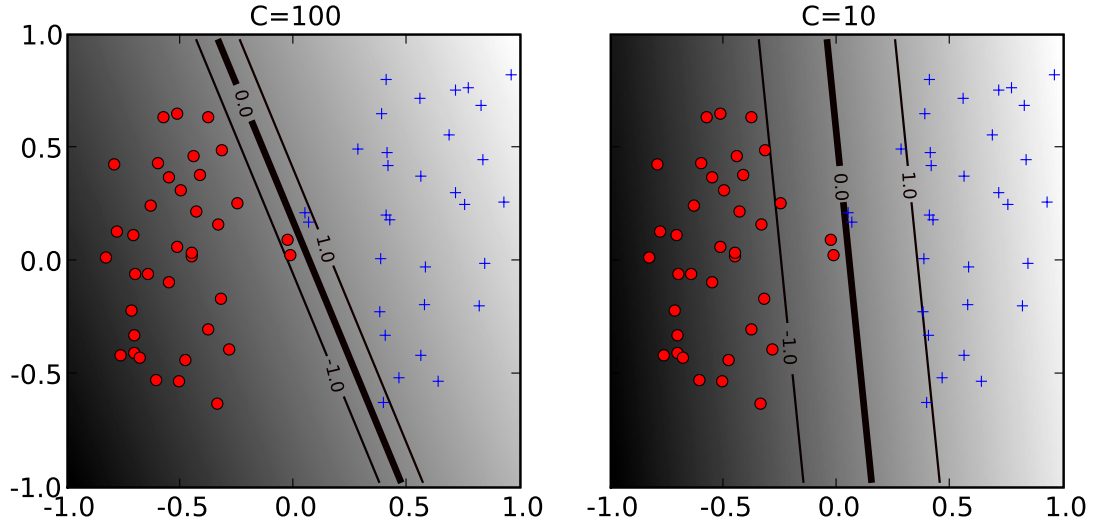


Figure 3: The effect of the soft-margin constant, C , on the decision boundary. A smaller value of C (right) allows to ignore points close to the boundary, and increases the margin. The decision boundary between negative examples (red circles) and positive examples (blue crosses) is shown as a thick line. The lighter lines are on the margin (discriminant value equal to -1 or $+1$). The grayscale level represents the value of the discriminant function, dark for low values and a light shade for high values.

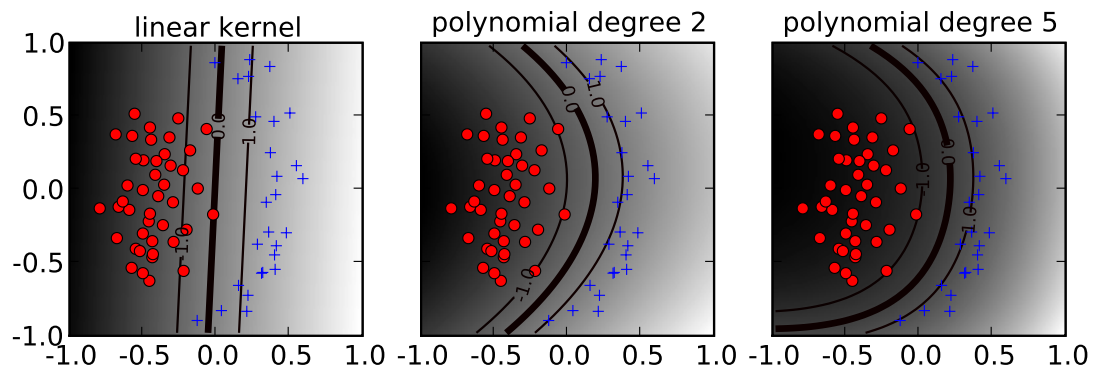


Figure 4: The effect of the degree of a polynomial kernel. Higher degree polynomial kernels allow a more flexible decision boundary. The style follows that of Figure 3.

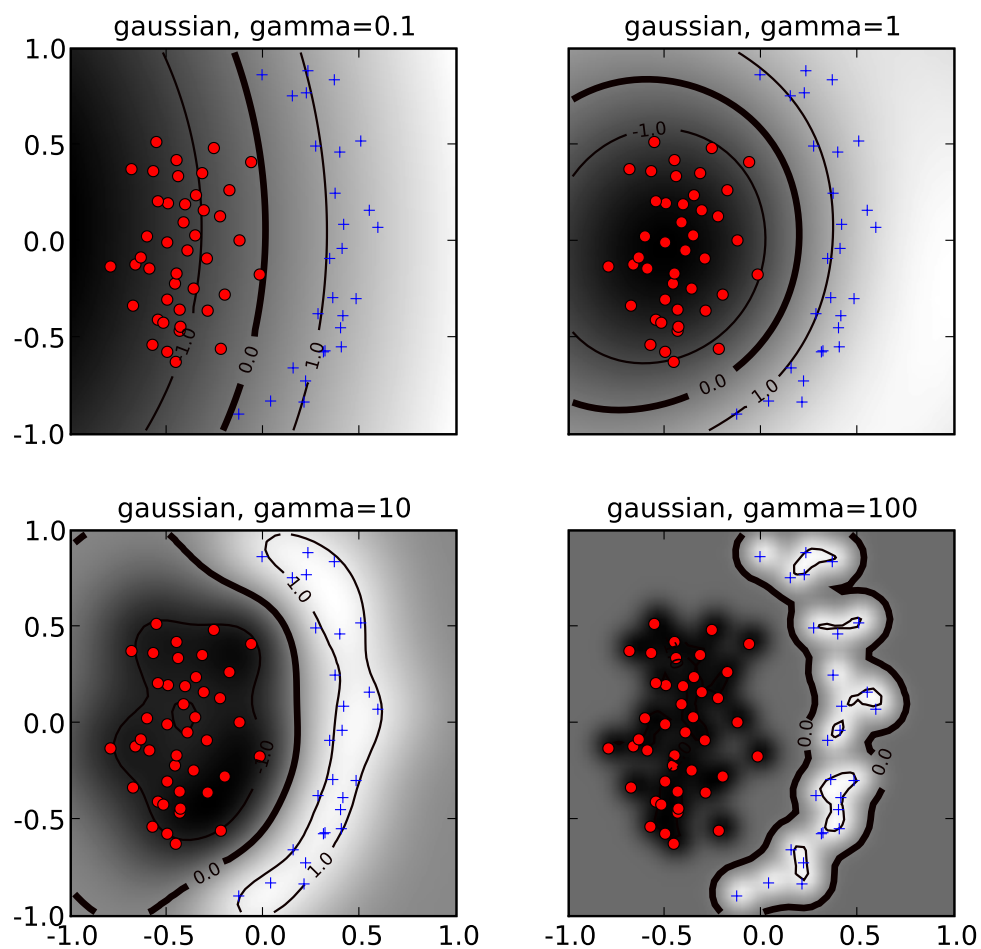


Figure 5: The effect of the inverse-width parameter of the Gaussian kernel (γ) for a fixed value of the soft-margin constant. For small values of γ (upper left) the decision boundary is nearly linear. As γ increases the flexibility of the decision boundary increases. Large values of γ lead to overfitting (bottom). The figure style follows that of Figure 3.

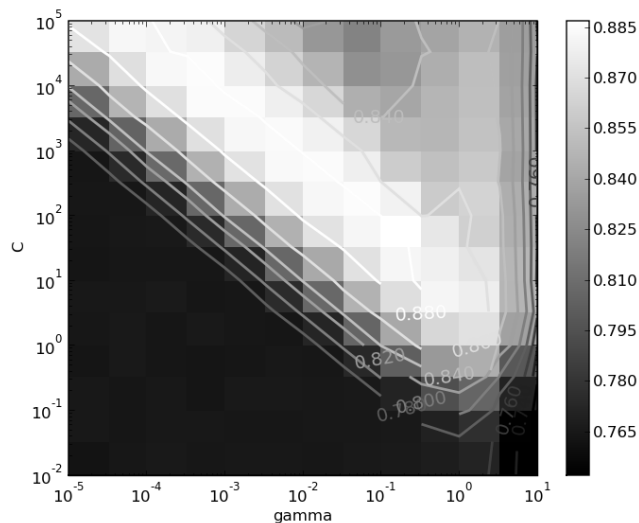


Figure 6: SVM accuracy on a grid of parameter values.

leads to overfitting in high dimensional datasets with a small number of examples, microarray datasets being a good example. Furthermore, an SVM with a linear kernel is easier to tune since the only parameter that affects performance is the soft-margin constant. Once a result using a linear kernel is available it can serve as a baseline that you can try to improve upon using a non-linear kernel. Between the Gaussian and polynomial kernels, our experience shows that the Gaussian kernel usually outperforms the polynomial kernel in both accuracy and convergence time.

6 Model Selection

The dependence of the SVM decision boundary on the SVM hyperparameters translates into a dependence of classifier accuracy on the hyperparameters. When working with a linear classifier the only hyperparameter that needs to be tuned is the SVM soft-margin constant. For the polynomial and Gaussian kernels the search space is two-dimensional. The standard method of exploring this two dimensional space is via grid-search; the grid points are generally chosen on a logarithmic scale and classifier accuracy is estimated for each point on the grid. This is illustrated in Figure 6. A classifier is then trained using the hyperparameters that yield the best accuracy on the grid.

The accuracy landscape in Figure 6 has an interesting property: there is a range of parameter values that yield optimal classifier performance; furthermore, these equivalent points in parameter space fall along a “ridge” in parameter space. This phenomenon can be understood as follows. Consider a particular value of (γ, C) . If we decrease the value of γ , this decreases the curvature of the decision boundary; if we then increase the value of C the decision boundary is forced to curve to accommodate the larger penalty for errors/margin errors. This is illustrated in Figure 7 for two dimensional data.

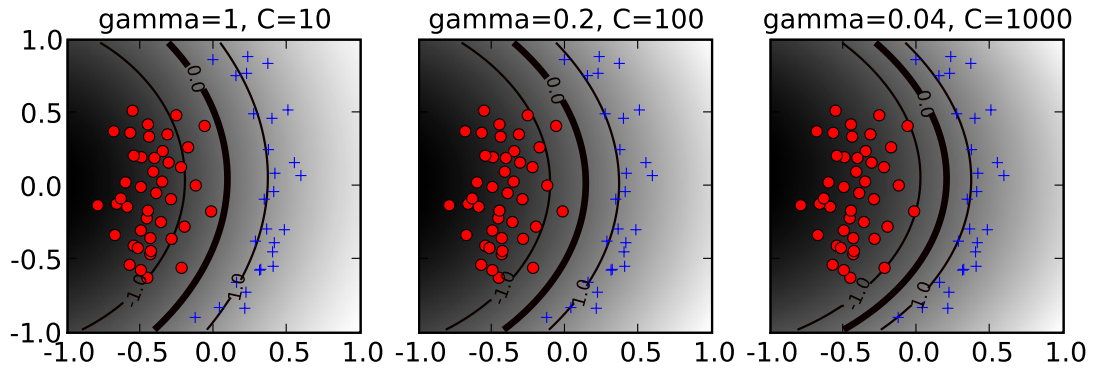


Figure 7: Similar decision boundaries can be obtained using different combinations of SVM hyperparameters. The values of C and γ are indicated on each panel and the figure style follows Figure 3.

7 SVMs for Unbalanced Data

Many datasets encountered in bioinformatics and other areas of application are unbalanced, i.e. one class contains a lot more examples than the other. Unbalanced datasets can present a challenge when training a classifier and SVMs are no exception — see [13] for a general overview of the issue. A good strategy for producing a high-accuracy classifier on imbalanced data is to classify any example as belonging to the majority class; this is called the majority-class classifier. While highly accurate under the standard measure of accuracy such a classifier is not very useful. When presented with an unbalanced dataset that is not linearly separable, an SVM that follows the formulation Eqn. 11 will often produce a classifier that behaves similarly to the majority-class classifier. An illustration of this phenomenon is provided in Figure 8.

The crux of the problem is that the standard notion of accuracy (the success rate, or fraction of correctly classified examples) is not a good way to measure the success of a classifier applied to unbalanced data, as is evident by the fact that the majority-class classifier performs well under it. The problem with the success rate is that it assigns equal importance to errors made on examples belonging to the majority class and errors made on examples belonging to the minority class. To correct for the imbalance in the data we need to assign different costs for misclassification to each class. Before introducing the balanced success rate we note that the success rate can be expressed as:

$$P(\text{success}|+)P(+) + P(\text{success}|-)P(-),$$

where $P(\text{success}|+)$ ($P(\text{success}|-)$) is an estimate of the probability of success in classifying positive (negative) examples, and $P(+)$ ($P(-)$) is the fraction of positive (negative) examples. The balanced success rate modifies this expression to:

$$BSR = (P(\text{success}|+) + P(\text{success}|-))/2,$$

which averages the success rates in each class. The majority-class classifier will have a balanced-success-rate of 0.5. A balanced error-rate is defined as $1 - BSR$. The BSR , as opposed to the standard success rate, gives equal overall weight to each class in measuring performance. A similar effect is obtained in training SVMs by assigning different misclassification costs (SVM soft-margin constants) to each class. The total misclassification cost, $C \sum_{i=1}^n \xi_i$ is replaced with two terms, one for each class:

$$C \sum_{i=1}^n \xi_i \longrightarrow C_+ \sum_{i \in I_+} \xi_i + C_- \sum_{i \in I_-} \xi_i,$$

where C_+ (C_-) is the soft-margin constant for the positive (negative) examples and I_+ (I_-) are the sets positive (negative) examples. To give equal overall weight to each class we want the total penalty for each class to be equal. Assuming that the number of misclassified examples from each class is proportional to the number of examples in each class, we choose C_+ and C_- such that

$$C_+ n_+ = C_- n_-,$$

where n_+ (n_-) is the number of positive (negative) examples. Or in other words:

$$\frac{C_+}{C_-} = \frac{n_-}{n_+}.$$

This provides a method for setting the ratio between the soft-margin constants of the two classes, leaving one parameter that needs to be adjusted. This method for handling unbalanced data is implemented in several SVM software packages, e.g. LIBSVM [14] and PyML.

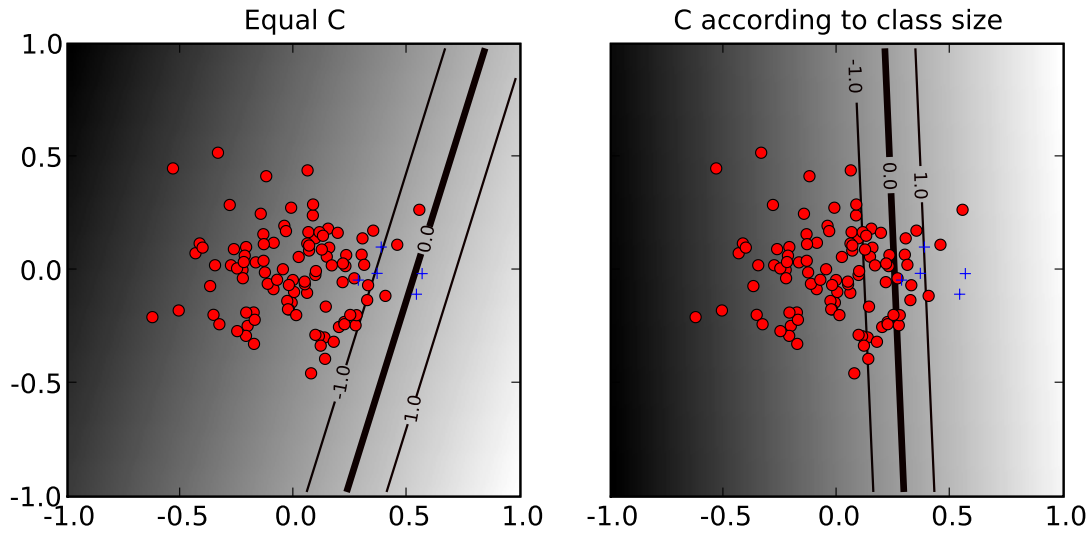


Figure 8: When data is unbalanced and a single soft-margin is used, the resulting classifier (left) will tend to classify any example to the majority-class. The solution (right panel) is to assign a different soft-margin constant to each class (see text for details). The figure style follows that of Figure 3.

8 Normalization

Large margin classifiers are known to be sensitive to the way features are scaled [14]. Therefore it is essential to normalize either the data or the kernel itself. This observation carries over to kernel-based classifiers that use non-linear kernel functions: The accuracy of an SVM can severely degrade if the data is not normalized [14]. Some sources of data, e.g. microarray or mass-spectrometry data require normalization methods that are technology-specific. In what follows we only consider normalization methods that are applicable regardless of the method that generated the data.

Normalization can be performed at the level of the input features or at the level of the kernel (normalization in feature space). In many applications the available features are continuous values, where each feature is measured in a different scale and has a different range of possible values. In such cases it is often beneficial to scale all features to a common range, e.g. by *standardizing* the data (for each feature, subtracting its mean and dividing by its standard deviation). Standardization is not appropriate when the data is sparse since it destroys sparsity since each feature will typically have a different normalization constant. Another way to handle features with different ranges is to bin each feature and replace it with indicator variables that indicate which bin it falls in.

An alternative to normalizing each feature separately is to normalize each example to be a unit vector. If the data is explicitly represented as vectors you can normalize the data by dividing each vector by its norm such that $\|\mathbf{x}\| = 1$ after normalization. Normalization can also be performed at the level of the kernel, i.e. normalizing in feature-space, leading to $\|\Phi(\mathbf{x})\| = 1$ (or equivalently $k(\mathbf{x}, \mathbf{x}) = 1$). This is accomplished using the *cosine* kernel which normalizes a kernel $k(\mathbf{x}, \mathbf{x}')$ to:

$$k_{\text{cosine}}(\mathbf{x}, \mathbf{x}') = \frac{k(\mathbf{x}, \mathbf{x}')}{\sqrt{k(\mathbf{x}, \mathbf{x})k(\mathbf{x}', \mathbf{x}')}}. \quad (14)$$

Note that for the linear kernel cosine normalization is equivalent to division by the norm. The use of the cosine kernel is redundant for the Gaussian kernel since it already satisfies $K(\mathbf{x}, \mathbf{x}) = 1$. This does not mean that normalization of the input features to unit vectors is redundant: Our experience shows that the Gaussian kernel often benefits from it. Normalizing data to unit vectors reduces the dimensionality of the data by one since the data is projected to the unit sphere. Therefore this may not be a good idea for low dimensional data.

9 SVM Training Algorithms and Software

The popularity of SVMs has led to the development of a large number of special purpose solvers for the SVM optimization problem [15]. One of the most common SVM solvers is LIBSVM [14]. The complexity of training of non-linear SVMs with solvers such as LIBSVM has been estimated to be quadratic in the number of training examples [15], which can be prohibitive for datasets with hundreds of thousands of examples. Researchers have therefore explored ways to achieve faster training times. For linear SVMs very efficient solvers are available which converge in a time which is linear in the number of examples [16, 17, 15]. Approximate solvers that can be trained in linear time without a significant loss of accuracy were also developed [18].

There are two types of software that provide SVM training algorithms. The first type are specialized software whose main objective is to provide an SVM solver. LIBSVM [14] and SVM^{light} [19] are two popular examples of this class of software. The other class of software are machine learning libraries that provide a variety of classification methods and other facilities such as methods

for feature selection, preprocessing etc. The user has a large number of choices, and the following is an incomplete list of environments that provide an SVM classifier: Orange [20], The Spider (<http://www.kyb.tuebingen.mpg.de/bs/people/spider/>), Elefant [21], Plearn (<http://plearn.berlios.de/>), Weka [22], Lush [23], Shogun [24], RapidMiner [25], and PyML (<http://pyml.sourceforge.net>). The SVM implementation in several of these are wrappers for the LIBSVM library. A repository of machine learning open source software is available at <http://mloss.org> as part of a movement advocating distribution of machine learning algorithms as open source software [7].

10 Further Reading

We focused on the practical issues in using support vector machines to classify data that is already provided as features in some fixed-dimensional vector space. In bioinformatics we often encounter data that has no obvious explicit embedding in a fixed-dimensional vector space, e.g. protein or DNA sequences, protein structures, protein interaction networks etc. Researchers have developed a variety of ways in which to model such data with kernel methods. See [2, 8] for more details. The design of a good kernel, i.e. defining a set of features that make the classification task easy, is where most of the gains in classification accuracy can be obtained.

After having defined a set of features it is instructive to perform *feature selection*: remove features that do not contribute to the accuracy of the classifier [26, 27]. In our experience feature selection doesn't usually improve the accuracy of SVMs. Its importance is mainly in obtaining better understanding of the data—SVMs, like many other classifiers, are “black boxes” that do not provide the user much information on why a particular prediction was made. Reducing the set of features to a small salient set can help in this regard. Several successful feature selection methods have been developed specifically for SVMs and kernel methods. The Recursive Feature Elimination (RFE) method for example, iteratively removes features that correspond to components of the SVM weight vector that are smallest in absolute value; such features have less of a contribution to the classification and are therefore removed [28].

SVMs are two-class classifiers. Solving multi-class problems can be done with multi-class extensions of SVMs [29]. These are computationally expensive, so the practical alternative is to convert a two-class classifier to a multi-class. The standard method for doing so is the so-called one-vs-the-rest approach where for each class a classifier is trained for that class against the rest of the classes; an input is classified according to which classifier produces the largest discriminant function value. Despite its simplicity, it remains the method of choice [30].

Acknowledgements

The authors would like to thank William Noble for comments on the manuscript.

References

- [1] B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In D. Haussler, editor, *5th Annual ACM Workshop on COLT*, pages 144–152, Pittsburgh, PA, 1992. ACM Press.

- [2] B. Schölkopf, K. Tsuda, and J.P. Vert, editors. *Kernel Methods in Computational Biology*. MIT Press series on Computational Molecular Biology. MIT Press, 2004.
- [3] W.S. Noble. What is a support vector machine? *Nature Biotechnology*, 24(12):1564–1567, 2006.
- [4] J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge UP, Cambridge, UK, 2004.
- [5] B. Schölkopf and A. Smola. *Learning with Kernels*. MIT Press, Cambridge, MA, 2002.
- [6] Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin. A practical guide to support vector classification. Technical report, Department of Computer Science, National Taiwan University, 2003.
- [7] S. Sonnenburg, M.L. Braun, C.S. Ong, L. Bengio, S. Bottou, G. Holmes, Y. LeCun, K Müller, F Pereira, C.E. Rasmussen, G. Rätsch, B. Schölkopf, A. Smola, P. Vincent, J. Weston, and R.C. Williamson. The need for open source software in machine learning. *Journal of Machine Learning Research*, 8:2443–2466, 2007.
- [8] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines*. Cambridge UP, Cambridge, UK, 2000.
- [9] T. Hastie, R. Tibshirani, and J.H. Friedman. *The Elements of Statistical Learning*. Springer, 2001.
- [10] C.M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2007.
- [11] C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20:273–297, 1995.
- [12] O. Chapelle. Training a support vector machine in the primal. In L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, editors, *Large Scale Kernel Machines*. MIT Press, Cambridge, MA., 2007.
- [13] F. Provost. Learning with imbalanced data sets 101. In *AAAI 2000 workshop on imbalanced data sets*, 2000.
- [14] C-C. Chang and C-J. Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at <http://www.csie.ntu.edu.tw/~cjlin/libsvm>.
- [15] L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, editors. *Large Scale Kernel Machines*. MIT Press, Cambridge, MA., 2007.
- [16] T. Joachims. Training linear SVMs in linear time. In *ACM SIGKDD International Conference On Knowledge Discovery and Data Mining (KDD)*, pages 217 – 226, 2006.
- [17] V. Sindhwani and S. S. Keerthi. Large scale semi-supervised linear svms. In *29th annual international ACM SIGIR Conference on Research and development in information retrieval*, pages 477–484, New York, NY, USA, 2006. ACM Press.
- [18] A. Bordes, S. Ertekin, J. Weston, and L. Bottou. Fast kernel classifiers with online and active learning. *Journal of Machine Learning Research*, 6:1579–1619, September 2005.

- [19] T. Joachims. Making large-scale support vector machine learning practical. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods: Support Vector Machines*. MIT Press, Cambridge, MA, 1998.
- [20] J. Demsar, B. Zupan, and G. Leban. *Orange: From Experimental Machine Learning to Interactive Data Mining*. Faculty of Computer and Information Science, University of Ljubljana, 2004.
- [21] K. Gawande, C. Webers, A. Smola, S.V.N. Vishwanathan, S. Gunter, C.H. Teo, J.Q. Shi, J. McAuley, L. Song, and Q. Le. ELEFANT user manual (revision 0.1). Technical report, NICTA, 2007.
- [22] I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2nd edition edition, 2005.
- [23] L. Bottou and Y. Le Cun. *Lush Reference Manual*, 2002.
- [24] S. Sonnenburg, G. Raetsch, C. Schaefer, and B. Schoelkopf. Large scale multiple kernel learning. *Journal of Machine Learning Research*, 7:1531–1565, 2006.
- [25] I. Mierswa, M. Wurst, R. Klinkenberg, M. Scholz, and T. Euler. YALE: Rapid prototyping for complex data mining tasks. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2006.
- [26] I. Guyon, S. Gunn, M. Nikravesh, and L. Zadeh, editors. *Feature extraction, foundations and applications*. Springer Verlag, 2006.
- [27] I. Guyon and A. Elisseeff. Special issue on variable and feature selection. *Journal of Machine Learning Research*, 3, March 2003.
- [28] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik. Gene selection for cancer classification using support vector machines. *Machine Learning*, 46(1-3):389–422, 2002.
- [29] J. Weston and C. Watkins. Multi-class support vector machines. *Royal Holloway Technical Report CSD-TR-98-04*, 1998.
- [30] R. Rifkin and A. Klautau. In defense of one-vs-all classification. *Journal of Machine Learning Research*, 5:101–141, 2004.