5. Introduction to Support Vector Machines and Kernel Methods

Facundo Bromberg Ph.D.
Laboratorio DHARMa Dept. de Sistemas de Informacin
U. Tecnolgica Nacional - Facultad Regional Mendoza - Argentina
fbromberg@frm.utn.edu.ar
http://dharma.frm.utn.edu.ar

Agosto 2012

Contents

The present chapter was constructed using material from several sources. Mainly, it is based on chapter 6 of [1], a great introduction to Kernel methods and SVMs by C. Campbell, and a great presentation by John Shawe-Taylor available online (in both video and pdf forms) at [1] (in english). The main benefit of these presentation is its aim toward intuition rather than technical details.

1 Support Vector Machines

In this section we explore the *Support Vector Machines* (**SVM**) algorithm. This algorithm, as well as *Neural Networks*, overcome some shortcomings of the *perceptron* algorithm. Inspired by models of natural neural networks, Rosenblat introduces the *perceptron* in [4], a mathematical model of a neuron that is algorithmically efficient. Later, as explained in chapter 4 of [1], the perceptron algorithm was understood as a generalization of linear discriminant classifiers of the form

$$y(\mathbf{x}) = \mathbf{w}^T.\phi(\mathbf{x}) + b$$

where the input vector $\mathbf{x} = \{x_1, \cdots, x_D\}$ was projected into a vector $\phi = \{\phi_1, \cdots, \phi_M\} \in \mathcal{F}$ on a feature space $\mathcal{F}$. This way, a training data set with input vectors $S = \{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$ and target values $t_1, \cdots, t_N$, $t_n \in \{-1, 1\}$ that is not linearly separable may be rendered into a linearly separable set $\{\phi(\mathbf{x}), \cdots, \phi_N(\mathbf{x})\}$ (and same target values) in the projected space. Let's see an example

Example 1. *The example considers the transformation of* $\mathbf{x} = (x_1, x_2)$ *into* $\phi = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$. *As before, the separation surface must satisfy* $\mathbf{w}^t.\phi(\mathbf{x}) + b = 0$, *that is,* $w_1x_1^2 + w_2\sqrt{2}x_1x_2 + w_3x_2^2 + b = 0$. *As shown before, the latter is the plane equation in 3D (ϕ-space), but in x-space is a conic section equation* $Ax_1^2 + Bx_1x_2 + Cx_2^2 + Dx_1 + Ex_2 + F = 0$, *with* $D = E = 0$, $A = w_1$, $B = w_2\sqrt{2}$, $C = w_3$, *and* $F = b$ *which corresponds to a circle, ellipse, parabola, or hyperbola depending on the values of* A, B, C *and* F, *i.e., the values of* $\mathbf{w}$ *and* b.

This is shown graphically in figure (1), where a dataset that is non-separable linearly, can be separated linearly in ϕ-space (right), corresponding to a circle on the $\mathbf{x}$*-space (left).*

The projection into a feature space has the enormous advantage of generalizing linear disciminant algorithms to non-linear discriminant algorithms.

[1] videolectures.net/stw07_taylor_kmsl/

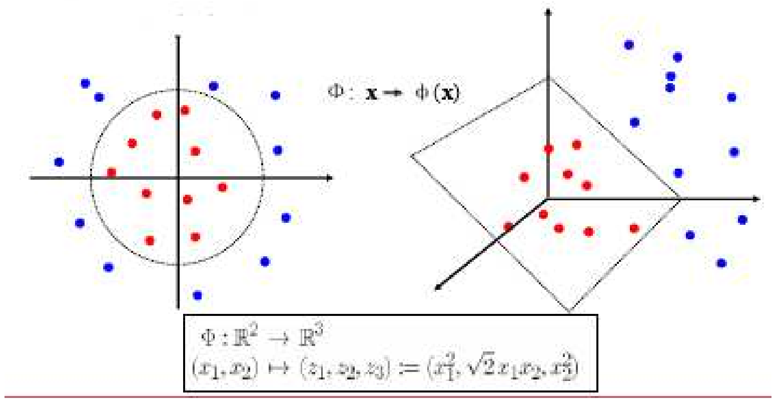
$\Phi : \mathbf{x} \to \phi(\mathbf{x})$
$\Phi : \mathbb{R}^2 \to \mathbb{R}^3$
$(x_1, x_2) \mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt{2}x_1x_2, x_2^2)$

Figure 1: A sample that is non-linearly separable in $\mathbf{x}$-space (left), is separated linearly in ϕ-space (right); corresponding to a non-linear separation surface (a circle) in $\mathbf{x}$-space (left). corresponding to a circle on the $\mathbf{x}$-space (left).

However, a naïve use of the projection trick may produce very bad results. First, it is hard to come up with good mappings ϕ. The artificial neural networks (**ANNs**) approach is to parametrize the mappings ϕ, allowing them to be adapted during learning together with the weights $\mathbf{w}$. The discovery of the backpropagation algorithm was the final result necessary to recover the interest in linear discriminants. This lead to a great amount of work resulting in many practical algorithms in the family of ANNs. These algorithms, however, present several shortcomings that are succesfully addressed by support vector machines:

- ANNs requires a non-linear optimization of the error function, resulting in several local minima, and thus a potencially long sequence of iterations. Morever, since it is never known when a minima is the global one, the algorithm may terminate too soon with a sub-optimal minima.

- The backpropagation algorithm may have a low cost ($O(W)$, with W the total number of weights), but this is a price that has to be paid during each out of many iterations, with the algorithm having a long sequence of iterations due to the non-linearity of the error function.

- Over-fitting resulting from too many hidden units may be overcome only empirically by evaluating a validation set. This results in smaller training sets, and may result in a pour prunning criteria when the validation set is not iid, or is not a good representation of the underlying distribution.

The advantages of the SVM algorithm are many-fold:

- It results in a quadratic (convex) optimization, which consists of a single minima. This reduces considerably the complexity of the optimization, and guarantees the result to be the global minima.
- It has a dual representation that allows the *kernel trick*. This further reduces the amount of computations due to several results that allow efficient computations of the kernel (more later).
- Over-fitting is minimized by taking advantage of theoretical results of computational learning theory that assures that neither a validation set nor prunning are required. That is, the technique assures a minimization of the generalization error, regardless of the dimensionality M.

SVMs belongs to the family of classifiers that requires remembering the examples S (other members are the Parzen window and nearest neighbor classifier). However, as it will be later shown, the SVMs algorithm requires the memorization of only a sparse set of examples (the so called *support vectors*). This results in a runtime for classification that is competitive when compared to ANNs.

One major limitation of SVMs is they are fundamentally two-class classifiers and the problem of multi-class classification is still an open problem. However, support vector machines are useful for regression, classification and novelty detection, but because it is a decision machine it does not provide posterior probabilities (see [1], §1.5.4 for a discussion on the advantages of determining probabilities). An alternative is known as the *relevance vector machine*, which on top of providing posterior probabilistic outputs, it results tipically in much sparser solutions than SVMs (see [1], §7.2).

In what follows we discuss SVMs for binary classification and omit multiclass SVMs and SVMs for regression (explained in detail in [1], §7.1.3 and 7.1.4). Since we rely heavily on Lagrange multipliers, the reader is encouraged to review these topics in appendix E of [1].

1.1 Support Vector Machines for binary classification

In the context of binary classification we introduce here important concepts and theoretical results that are pertinent to SVMs in general. As usual, we consider the linear model

$$y(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) + b \tag{1}$$

and a training data set $\mathcal{T} = \{(\mathbf{x}_n, t_n)\}_{n=1}^N$, containing, for each data point $\mathbf{x}_n$, a class label t_n. For convenience, we assume the encoding $t \in \{1, -1\}$ for the class label. Also, we assume this training data set is linearly separable in feature (ϕ) space (see [1], section 7.1.1, for an SVM framework that relax this assumption). Under these assumptions, the problem consists on finding the linear separator $\{\mathbf{w}^*, b^*\}$ that correctly separates the training set, that is, it results, for every $(\mathbf{x}_n, t_n)$ in the dataset, that $y(\mathbf{x}_n) \geq 0$ whenever $t_n = 1$, and $y(\mathbf{x}_n) < 0$ whenever $t_n = -1$. For later convenience, we note this can be summarized as $t_n y(\mathbf{x}_n) \geq 0$. To illustrate, Figure 2 shows several possible linear discriminant solutions for a binary classification problem.

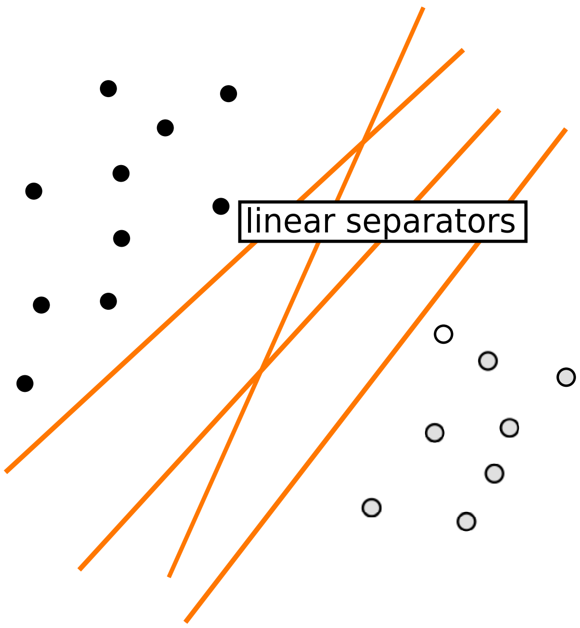

Figure 2: A single training set may have several possible linear separators (in orange).

We already solved this problem using the Perceptron algorithm. The solution found by this algorithm, however, depends on the (arbitrary) initial values chosen for $\mathbf{w}$ and b, as well as on the order in which the data points are processed. This results in infinitely many possible solutions. In terms of linear discriminants, these are all possible linear boundaries that separates the training set into positive and negative examples, as exemplified by Figure 2 .

Intuitivelly, one can imagine that some of the solutions may be better generalizers with those further appart from any example are more likely to correctly classify new examples, as examples of the same class are more likely to be closely located. This is exactly what SVMs, output the solution that maximizes the distance to its closest example. Before formalizing this, let's reinforce the major problem of overfitting that may occur with linear discriminants.

For that we first present an auxiliary theorem

Theorem 1. *Given $M + 1$ examples in general position* [2] *in an M dimensional space, we can generate every possible classification with a thresholded linear function.*

The theorem is exemplified for $M = 2$ in Figure 3 where it is shown that all possible dichotomies of the $M + 1 = 3$ data points can be linearly separated (whenever they are in general position, i.e., no subset lies in a line).

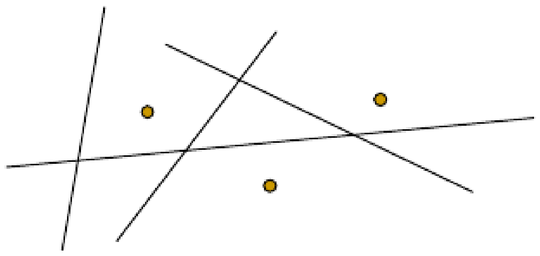

Figure 3: Demonstration of Theorem 1 for 2D. We can see that all dichotomies can be linearly separated. linearly separable.

Therefore, when M, the dimension of the ϕ-space is chosen to high, in particular, when $M = N - 1$, with N the number of data points, the theorem holds. In that case, the fact that all possible dichotomies are linearly separable is a clear sign of overfitting, as discussed in Section 3 (Aprendizaje requiere bias) of the Introduction chapter, There we discussed that in such cases, no learning (and thus generalization) is possible at all, that is, overfitting is at its maximum; decreasing as the space of possible separators is restricted more and more.

[2] i.e., no subset lies in a linear sub-space

SVMs overcome this problem by selecting the linear separator $\{\mathbf{w}^*, b^*\}$ that is further appart from its closest example. The distance to its closes example is called the *margin* of the linear separator, and is denoted by γ (see Figure (4)). Results from computational learning theory (see [2], [5]) guarantees that the generalization error is minimized by *maximizing* the *margin* γ. Let's reproduce here the main theorem:

Theorem 2. *Suppose examples are drawn independentely according to a distribution whose support is contained in a ball in $\mathcal{R}^D$, centered at the origin, of radius R. If hypothesis $h \in H$ succeed in correctly classifiying m such examples by a canonical hyperplane, then with confidence $1 - \delta$ the generalization error $err(h)$ will be bound from above:*

$$err(h) < \frac{2}{m}\left(\frac{64R^2}{\gamma^2}\log\left(\frac{\gamma e m}{8R^2}\right) + \log\left(\frac{4}{\delta}\right)\right)$$

provided $64R^2/\gamma^2 < m$ (i.e., sufficiently large m).

Some comments about this bound:

- It is not dependent on the dimensionality of the space (M), and
- It is reduced by maximizing the margin γ.
- It requires the data to be linearly separable. It does not provide any guarantees if the opposite is true.

We now formalize the definition of the margin γ:

Definition 1. *The* margin *γ of a linear separator $\{\mathbf{w}, b\}$ and training dataset $\mathcal{T}$ is the smallest distance between the decision boundary of the linear separator, and any of the samples in $\mathcal{T}$. Denoting the distance of $\mathbf{x}_n$ to the decision boundary by $d_\perp(\mathbf{x}_n)$, we have that*

$$\gamma = \min_n d_\perp(\mathbf{x}_n)$$

This is illustrated in Figure (4).

We thus have that the solution according to the SVM approach is:

$$\{\mathbf{w}^*, b^*\} = \arg\max_{\mathbf{w},b}\{\gamma\} = \arg\max_{\mathbf{w},b}\{\min_n [d_\perp(\mathbf{x}_n)]\} \qquad (2)$$

$$t_n y(x_n) \geq 0, \qquad n = 1, \ldots, N,$$

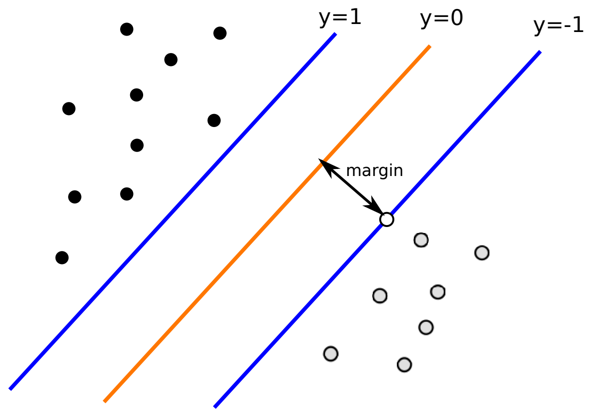

Figure 4: The margin is defined as the smallest distance between the decision boundary (in red) and any of the samples (adapted from [1]).

where the condition $t_n y(x_n) \geq 0$ guarantees $\{\mathbf{w}^*, b^*\}$ is indeed a solution, i.e., it separates correctly the training dataset. To proceed we recall the perpendicular distance $d_\perp(\mathbf{x})$ of some point $\mathbf{x}$ to the separation boundary defined by $y(\mathbf{x}) = 0$ is

$$d_\perp(\mathbf{x}) = \frac{|y(\mathbf{x})|}{\|\mathbf{w}\|} = \frac{|\mathbf{w}^T\mathbf{x} + b|}{\|\mathbf{w}\|}, \tag{3}$$

resulting in

$$\begin{aligned} &\{\mathbf{w}^*, b^*\} = \arg\max_{\mathbf{w},b} \left\{ \frac{1}{\|\mathbf{w}\|} \min_n \left(t_n(\mathbf{w}^T\phi(\mathbf{x}_n) + b) \right) \right\} \\ &t_n y(x_n) \geq 0, \qquad n = 1, \ldots, N, \end{aligned} \tag{4}$$

where $|y(\mathbf{x}_n)|$ was replaced by $t_n y(\mathbf{x}_n)$, since, from the constrains, they are both equal. Also, we have taken the factor $1/\|\mathbf{w}\|$ out of the optimization over n because $\mathbf{w}$ is the same for all n's.

To further simplify this quantity we note that according to Eq.(3). the distance from any point $\mathbf{x}_n$ to the decision surface remains unchanged if we rescale both $\mathbf{w}$ and b by the same constant κ, i.e. $\mathbf{w} \to \kappa\mathbf{w}$ and $b \to \kappa b$. We can then set

$$t_n \left(\mathbf{w}^T\phi(\mathbf{x}_n) + b \right) = 1 \tag{5}$$

for the point $\mathbf{x}_n$ that is closest to the surface. In that case, it will hold for all other points that

$$t_n \left(\mathbf{w}^T \phi(\mathbf{x}_n) + b \right) \geq 1, \qquad n = 1, \cdots, N. \tag{6}$$

(because they are further away from the surface).

With this rescaling, the minimum of the minimization of Eq. (4) is 1, and thus Eq. (4) becomes

$$\{\mathbf{w}^*, b^*\} = \arg\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2$$

subject to the constraints given by Eq. (6); where the factor 1/2 is included for later convenience, and the change from $\|w\|$ to $\|w\|^2$ does not change the minimization, being $\|w\|$ a monotonally increasing function.

The resulting optimization problem is an instance of a *quadratic programming* problem, i.e., a minimization of a *quadratic* function subject to a set of *linear inequality* constraints. Note that the parameter b, altought dissapeared from Eq. (4), appears in the constraints.

As explained in appendix E of [1] and elsewhere, a constraint optimization problem with inequality constraints can be solved by minimizing a lagrangian function of the form

$$L(\mathbf{w}, b, \mathbf{a}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{n=1}^{N} a_n \left\{ t_n \left(\mathbf{w}^T \pi(\mathbf{x}_n) + b \right) - 1 \right\} \tag{7}$$

where $\mathbf{a} = (a_1, \cdots, a_N)^T$, and where the a_n's are positive reals called *largrange multipliers*, the minus sign between the two terms implies a minimization, and each of the lagrange multipliers is multiplying the constraints (curly brakets). To minimize the largangian, one must set the derivatives of $L(\mathbf{w}, b, \mathbf{a})$ with respect to $\mathbf{w}$, b and $\mathbf{a}$ to zero. For $\mathbf{w}$ and b we obtain

$$\mathbf{w} = \sum_{n=1}^{N} a_n t_n \phi(\mathbf{x}_n) \tag{8}$$

$$0 = \sum_{n=1}^{N} a_n t_n \tag{9}$$

In order to obtain the values for the w_i's and t_n we must solve the third equation, the one resulting from equating the derivative of $L(\mathbf{w}, b, \mathbf{a})$ with respect to $\mathbf{a}$ to zero. However, instead of taking that route we re-incorporate the above value of $\mathbf{w}$ into $L(\mathbf{w}, b, \mathbf{a})$ to obtain

$$\begin{aligned}
\widetilde{L}(\mathbf{a}) &= \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{n=1}^{N} a_n \left\{ t_n \left(+ b \right) - 1 \right\} \\
&= \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{n=1}^{N} a_n t_n \mathbf{w}^T \phi(\mathbf{x}_n) - b - \sum_{n=1}^{N} a_n t_n + \sum_{n=1}^{N} a_n \\
&= \frac{1}{2}\mathbf{w}^T \sum_{n=1}^{N} a_n t_n \phi(\mathbf{x}_n) - \mathbf{w}^T \sum_{n=1}^{N} a_n t_n \phi(\mathbf{x}_n) + \sum_{n=1}^{N} a_n \\
&= \sum_{n=1}^{N} a_n - \frac{1}{2}\mathbf{w}^T \sum_{n=1}^{N} a_n t_n \phi(\mathbf{x}_n) \\
&= \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m \phi^T(\mathbf{x}_n)\phi(\mathbf{x}_m) \\
&= \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m), \qquad (10)
\end{aligned}$$

subject to the constrains

$$\begin{aligned}
a_n &\geq 0 \quad , \qquad n = 1 \ldots, N \\
\sum_{n=1}^{N} a_n t_n &= 0.
\end{aligned}$$

where we denoted $\phi^T(\mathbf{x}_n)\phi(\mathbf{x}_m)$ by $k(\mathbf{x}_n, \mathbf{x}_m)$. Note that given as input two input vectors $\mathbf{x}_n$ and $\mathbf{x}_m$ it produces a real number, and thus can be seen as a function of these inputs. Such a function of two vectors that results from first applying a non-linear vector function (i.e., ϕ) to each of the input vectors (i.e., $\mathbf{x}$) and then computing the inner product between the resulting vectors is called in mathematics a *kernel* function. Having the input vectors $\mathbf{x}_n$ and feature vectors ϕ showing up as a kernel (i.e., as an inner product) has several benefits that we will discuss below. These benefits are common to any learning problem whose optimization depends on the ϕ's and $\mathbf{x}$'s solely through inner products between the feature vectors ϕ, i.e., through kernels. The alternative representation obtained above for the margin maximization problem in terms of kernels can be obtained for many other learning problems (see chapter 6 of [1]). It therefore receives a special name, the *dual representation.*.

1.2 Evaluation of SVMs

At this point, the solution of the dual problem (fromalized by $\widetilde{L}(\mathbf{a})$ is the value of $\mathbf{a}$. Given this solution, how do we classify some unseen input vector? As usual, all we have to do is evaluate the sign of $y(\mathbf{x})$. However, we have the solution to the dual problem, i.e., the a_n's. All we have to do is therefore substitute the solution for $\mathbf{w}$ of Eq. (8) into Eq. (1) to obtain

$$y(\mathbf{x}) = \sum_{n=1}^{N} a_n t_n k(\mathbf{x}, \mathbf{x}_n) + b \tag{11}$$

First note that in order to evaluate $y(\mathbf{x})$ as shown above we must first obtain a solution for the parameter b. Let us assume we have it for now, and postpone to the next section the discussion on how it is calculated.

The above equation makes explicit the main (apparent) drawback of SVMs, namely, that the complexity of classifying some input vector is $O(N)$, the size of the training set. This results in two contradictory forces. On one hand we want large training set to improve the quality of the classifier, but that will increase the complexity of classification. We now proceed to demonstrate how this can be overcome, justifying why SVMs belongs to the so called *sparse kernel machines*.

Appendix E of [1] shows that each lagrange multipliers a_n, $n = 1, \cdots, N$, must satisfy $a_n, \geq 0$, $g(\mathbf{x}_n) \geq 0$, and $a_n g(\mathbf{x}_n) = 0$ where $g(\mathbf{x}_n)$ denotes the constraint function. These are the *Karish-Kuhn-Tucker*(KKT) properties, and for the SVM takes the form

$$\begin{aligned} a_n &\geq 0 \\ t_n y(\mathbf{x}_n) - 1 &\geq 0 \\ a_n\{t_n y(\mathbf{x}_n) - 1\} &= 0 \end{aligned}$$

In particular, from the last property, either $a_n = 0$ or $t_n y(\mathbf{x}_n) = 1$. Those points for which the former is true, i.e. $a_n = 0$, will not appear in the sum of Eq. (11). All other points are called the *support vectors*. Because these vectors satisfy the latter condition, i.e. $t_n y(\mathbf{x}_n) = 1$, they must lie in the margin as shown by Eq. (5). We see then that once the model is trained, all but a few vectors can be discarded resulting in an efficient evaluation of the model, i.e.,

$$y(\mathbf{x}) = \sum_{m \in \mathcal{S}} a_n t_n k(\mathbf{x}, \mathbf{x}_n) + b \tag{12}$$

where $\mathcal{S}$ denotes the set of indexes of the support vectors.

1.3 Evaluation of b

We can determine the value of the threshold parameter b using the fact that any support vector $\mathbf{x}_n$ satisfies $t_n y(\mathbf{x}_n) = 1$. Using Eq.(11) we obtain

$$t_n \left(\sum_{m \in \mathcal{S}} a_m t_m k(\mathbf{x}_n, \mathbf{x}_m) + b \right) = 1,$$

where $\mathcal{S}$ denotes the set of indexes of the support vectors.

Although we can solve this equation for b using an arbitrarily chosen support vector $\mathbf{x}_n$, a numerically more stable solution is obtained by first multiplying through by t_n, making use of $t_n^2 = 1$, and then averaging these equations over all support vectors and solving for b to give

$$b = \frac{1}{N_{\mathcal{S}}} \sum_{n \in \mathcal{S}} \left(t_n - \sum_{m \in \mathcal{S}} a_m t_m k(\mathbf{x}_n, \mathbf{x}_m) \right)$$

1.4 Multiclass SVMs

SVMs are fundamentally a two-class classifiers. Several attempts has been made so far to extend it to multi-class classification but it remains an open problem. Despite its limitations, in practice the method *one-versus-the-rest* is the most commonly used. This method constructs K separate SVMs in which the k-th model $y_k(\mathbf{x})$ is trained using the data from class $\mathcal{C}_k$ as the positive examples and the data from the remaining $K - 1$ classes as the negative examples. See [1], section 7.1.3 for further details.

2 Kernels

We have already seen that increasing M, the dimension of the feature space, cannot hinder the generalization error. However, increasing M has an important impact in the computational cost, in particular, the $O(M)$ cost of performing the inner products required for computing the kernel $k(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle$ of input vectors $\mathbf{x}$ and $\mathbf{z}$. As an example, consider the following mapping into feature space from 2 to 4 dimensions:

$$\phi \ : \ (x_1, x_2) \rightarrow (x_1^2, x_1 x_2, x_2 x_1, x_2^2) \tag{13}$$

resulting in 4 components for ϕ.

In many practical cases, it is not uncommon to consider transformation into feature spaces of much larger dimensions (millions, even infinite as will be exemplified below). We show now how the *kernels* can help reducing this computation to the original $O(D)$ of the inner product in x-space. For that, let's express explicitelly the inner product for our example:

$$\begin{aligned} \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle &= \langle (x_1^2, x_1x_2, x_2x_1, x_2^2), (z_1^2, z_1z_2, z_2z_1, z_2^2) \rangle & (14) \\ &= (x_1z_1)^2 + 2x_1z_1x_2z_2 + (x_2z_2)^2 & (15) \end{aligned}$$

It is not hard to show that the same result can be obtained by computing

$$\langle \mathbf{x}, \mathbf{z} \rangle^2 \tag{16}$$

Left as a quick exercise for the reader. The important point is that the computation of the inner product directly over the feature space takes M operation (e.g., 4 operations in the above example), whereas using Eq. (16) it takes D operations to compute the inner product between $\mathbf{x}$ and $\mathbf{z}$ (e.g., 2 operations in the above example) plus 1 operation to compute the squared, which is $O(D)$.

Furthermore, the *kernel trick* can be used in continuous domains (i.e., infinite dimensions) to compute the kernel efficiently (something that would clearly take infinitely many computations if computed through the inner product in the feature space). An example of a continuous kernel is the Gaussian (a.k.a. *radial basis functions*(RBF)) kernel:

$$k(\mathbf{x}, \mathbf{z}) = exp\left(-\frac{\|\mathbf{x}-\mathbf{z}\|^2}{2\sigma^2}\right).$$

The above kernel makes us wonder whether each function over two vectors (e.g., $\mathbf{x}$ and $\mathbf{z}$ above) is a kernel. Although this is not the case, many functions are. Moreover, there is a simple rule to decide that. For that, let us first introduce the $N \times N$ *gram-matrix* $\mathbf{K}$ whose components $\mathbf{K}_{ij}$ are defined by the inner product, i.e.,

$$\mathbf{K}_{ij} = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle$$

We present here a theorem that specify the condition for a matrix to be a gram (kernel) matrix, only for the discrete case, and leave the case for continuous spaces (called the *Mercer's theorem* to be studied independently by the interested reader (for instance, from C. Campbell online material referenced at the begining of the section).

Theorem 3. *Let $k(\mathbf{x}, \mathbf{y})$ be a real symmetric function on a finite input space, then it is a kernel function if and only if the matrix $\mathbf{K}$ with components $K(\mathbf{x}_i, \mathbf{x}_j)$ is positive semi-definite.*

2.1 Useful properties for constructing new kernels modularly

Given valid kernels $k_1(\mathbf{x}, \mathbf{x}')$ and $k_2(\mathbf{x}, \mathbf{x}')$, the following new kernels will also be valid:

$$\begin{aligned}
k(\mathbf{x}, \mathbf{x}') &= c k_1(\mathbf{x}, \mathbf{x}') \\
k(\mathbf{x}, \mathbf{x}') &= f(\mathbf{x}) k_1(\mathbf{x}, \mathbf{x}') f(\mathbf{x}') \\
k(\mathbf{x}, \mathbf{x}') &= q(k_1(\mathbf{x}, \mathbf{x}')) \\
k(\mathbf{x}, \mathbf{x}') &= exp\left(k_1(\mathbf{x}, \mathbf{x}')\right) \\
k(\mathbf{x}, \mathbf{x}') &= k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}') \\
k(\mathbf{x}, \mathbf{x}') &= k_1(\mathbf{x}, \mathbf{x}') k_2(\mathbf{x}, \mathbf{x}') \\
k(\mathbf{x}, \mathbf{x}') &= \mathbf{x}^T \mathbf{A} \mathbf{x}'
\end{aligned}$$

where $\mathbf{A}$ is a positive semidefinite matrix. You can find more properties at [1].

Since the identity matrix $\mathbf{I}$ is positive semidefinite, $\mathbf{x}^T\mathbf{x}'$ is a valid kernel (7th property). Then, from the 6th property, we have that $(\mathbf{x}^T\mathbf{x}')^2$, the quadratic kernel presented in the example above, is a valid kernel.

2.2 Kernels for particular applications

We give here a simple example and refer the reader to [3] for a more complete list of techniques for constructing kernels for special applications. The example consists of mapping an input text $\mathbf{x}$ into a feature vector $\phi(\mathbf{x})$ consisting on the bag of words vector of $\mathbf{x}$. These vectors can be very large ($\approx$ 30K components). However, the vectors can be very sparse in practice, with many components equal to zero due to lack of appearence of the corresponding word in the text. It is not hard to come up with an efficient algorithm for computing the inner product between two such sparse vectors (exercise: construct such algorithm and analyze its time complexity).

3 Ejercicios

1. Considere el siguiente dataset

x_1	x_2	x_3	y
Entrenamiento			
-1	1	1	-1
1	1	1	1
-1	1	-1	1
-1	-1	1	-1
1	1	-1	1
Testeo			
1	-1	-1	?
-1	-1	-1	?
1	-1	1	?

Hay tres atributos de entrada x_1, x_2, y x_3 que toman los valores 1 y -1. La etiqueta (o clase) está dada en la última columna por y, y también toma los valores + y -.

El objetivo del ejercicio es encontrar la hipotesis final que es inducida por el algoritmo de SVM, y mostrar como ha sido computada. Debe mostrar también cual es la predicción encontrada para los ejemplos de entrenamiento.

Puede tomar la información extra de que los primeros tres ejemplos son vectores soporte, pero los otros no lo son. Con lo que a_4 y a_5 serán ambos cero en la Eq. (10). Esto quiere decir que puede minimizar esta ecuación sobre a_1, a_2 y a_3 usando técnicas de minimización analítica (Note que si alguna de estas cantidades se hace negativas tenemos un problema porque se estaría violando las restricciones). Una ves que haya encontrado un vector solución $\mathbf{w}$, chequeelo mostrando que $t_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1$, para todo n, con igualdad estricta para el caso de los vectores soporte, i.e., los primeros tres ejemplos. Por último, no utilice ningún kernel, sino un simple producto escalar.

References

[1] Christopher M. Bishop. *Pattern Recognition and Machine Learning.* Springer Verlag, 2006.

[2] C. Campbell. An introduction to kernel methods. `www.enm.bris.ac.uk/cig/pubs/2000/svmintro.pdf`.

[3] Thomas Hofmann, Bernhard Schlkopf, and Er J. Smola. A review of kernel methods in machine learning, 2006.

[4] F. Rosenblatt. The perceptron–a perceiving and recognizing automaton. Technical Report 85-460-1, Cornell Aeronautical Laboratory, 1957.

[5] J. Shawe-Taylor, P. L. Barlett, R. C. Williamson, and M. Anthony. Structural risk minimization over data-dependent hierarchies. *IEEE Transactions on Information Theory*, 44:1926–1940, 1998.