Support Vector Machines and Kernels

Carlo Tomasi

November 1, 2021

As we saw in the previous note, a Support Vector Machine (SVM) is a linear classifier with decision rule

$$\hat{y} = h(\mathbf{x}) = \text{sign}(\mathbf{w}^{*T}\mathbf{x} + b^*) \; . \tag{1}$$

The parameters $\mathbf{w}^*, b^*$ of this rule are the solution to the problem

$$(\mathbf{w}^*, b^*) = \text{ERM}_T(\mathbf{w}, b) = \arg\min_{(\mathbf{w},b)} L_T(\mathbf{w}, b) \tag{2}$$

where the empirical risk L_T is

$$L_T(\mathbf{w}, b) \stackrel{\text{def}}{=} \frac{1}{2}\|\mathbf{w}\|^2 + \frac{C_0}{N}\sum_{n=1}^{N} \max\{0, 1 - y_n(\mathbf{w}^T\mathbf{x}_n + b)\} \; . \tag{3}$$

The vector $\mathbf{w}^*$ that determines the orientation of the optimal decision hyperplane enjoys a rather surprising property: It is a linear combination of the data points $\mathbf{x}_n$ in the training set T. This property is encapsulated by the so-called *representer theorem*, which is formulated and proven next, and will be shown to have far-reaching consequences.

1 The Representer Theorem

The gist of the representer theorem is that there exist coefficients $\beta_1, \ldots, \beta_N$ such that

$$\mathbf{w}^* = \sum_{n=1}^{N} \beta_n \mathbf{x}_n \; .$$

This is a rather striking result, since the vectors $\mathbf{x}_1, \ldots, \mathbf{x}_N$ do *not* generally span all of $\mathbb{R}^{d+1}$, especially if you consider that it is entirely possible that there are fewer data points than dimensions, $N \ll d$. For instance, if the data points $\mathbf{x}_n$ are color images, they can have millions of pixels, with each pixel color represented by three numbers, so d is in the many millions in that case. The number N of training images, on the other hand, is often much smaller.

While the representer theorem may seem to be just a theoretical curiosity, we will see soon that it has profound practical consequences, especially when combined with the fact that a large fraction of the β_n turn out to be zero. Let us first formulate and prove the result itself.

The representer theorem holds for a broader class of SVMs than we have seen so far, namely, it holds for all so-called *kernel SVMs*, which we will look at later on. In order to be able to use this theorem in this broader setting, we first abstract the structure of the SVM optimization problem in equations 2, 3 into more general notation, and prove the theorem for this more general formulation.

General Formulation of the Representer Theorem Rewrite the training risk in equation 3 as follows:

$$\begin{aligned} L_T(\mathbf{w}, b) &= \frac{1}{2}\|\mathbf{w}\|^2 + \frac{C_0}{N}\sum_{n=1}^{N}\max\{0, 1 - y_n(\mathbf{w}^T\mathbf{x}_n + b)\} \\ &= R(\|\mathbf{w}\|) + S\left(\mathbf{w}^T\mathbf{x}_1 + b,\ \ldots,\ \mathbf{w}^T\mathbf{x}_N + b\right) . \end{aligned}$$

In the specific case, the last expression can be obtained from one before it by the following definitions:

$$\begin{aligned} R(a) &= \frac{1}{2}a^2 \\ S(a_1, \ldots, a_N) &= \frac{C_0}{N}\sum_{n=1}^{N}\max\{0, 1 - y_n a_n\} . \end{aligned}$$

Please check that this is indeed the case.

However, we will prove the representer theorem not only for these specific definitions, but also in the more general case where the functions R and S satisfy the following weaker properties:

- R is any strictly increasing function from $\mathbb{R}_+$ to $\mathbb{R}$.
- S is any function from $\mathbb{R}^N$ to $\mathbb{R}$.

Proving the representer theorem in this more general formulation will become useful later on.

Since the representer theorem is a statement about $\mathbf{w}$, the value of the bias term b is irrelevant. We remove it from consideration by decomposing the minimization problem in equation 2 into two steps:[1] First minimize with respect to $\mathbf{w}$ and then with respect to b:

$$(\mathbf{w}^*, b^*) = \arg\min_b \min_{\mathbf{w}} L_T(\mathbf{w}, b) .$$

In this formulation, the minimization with respect to $\mathbf{w}$ considers b to be fixed, and we can restrict our attention to the subproblem

$$\mathbf{w}^*(b) = \arg\min_{\mathbf{w}} L_T(\mathbf{w}, b)$$

where b is an arbitrary constant.

To summarize what we are about to prove, the representer theorem states that if a function has the form

$$L(\mathbf{w}, b) = R(\|\mathbf{w}\|) + S\left(\mathbf{w}^T\mathbf{x}_1 + b,\ \ldots,\ \mathbf{w}^T\mathbf{x}_N + b\right) \tag{4}$$

where R and S satisfy the general properties listed above and b is a fixed real number, the value $\mathbf{w}^*$ of $\mathbf{w}$ that minimizes $L(\mathbf{w})$ is a linear combination of the data points $\mathbf{x}_1, \ldots, \mathbf{x}_N$:

$$\mathbf{w}^* = \sum_{n=1}^{N}\beta_n\mathbf{x}_n .$$

[1] This decomposition is just a way to *think* about optimization, and not how optimization is actually done in practice.

Proof of the Representer Theorem Let $\mathcal{X}$ be the space in $\mathbb{R}^d$ spanned by the vectors $\mathbf{x}_1, \ldots, \mathbf{x}_N$, and let $\mathcal{X}^\perp$ be the orthogonal complement of $\mathcal{X}$. That is, $\mathcal{X}^\perp$ is the set of all vectors in $\mathbb{R}^d$ that are orthogonal to all the vectors in $\mathcal{X}$. Then *any* vector can be written as a linear combination of vectors that jointly span $\mathcal{X}$ and $\mathcal{X}^\perp$, because these vectors then span all of $\mathbb{R}^d$. So we can write the optimal solution as

$$\mathbf{w}^* = \sum_{n=1}^{N} \beta_n \mathbf{x}_n + \mathbf{u} \quad \text{for some} \quad \mathbf{u} \in \mathcal{X}^\perp$$

without loss of generality.[2] Proving the representer theorem amounts to showing that $\mathbf{u} = 0$.

By contradiction, suppose that $\mathbf{u} \neq 0$, and consider the vector

$$\mathbf{w} = \mathbf{w}^* - \mathbf{u} = \sum_{n=1}^{N} \beta_n \mathbf{x}_n \ ,$$

which is the component of $\mathbf{w}^*$ in $\mathcal{X}$. The vectors $\mathbf{w}$ and $\mathbf{u}$ are orthogonal to each other because $\mathbf{w} \in \mathcal{X}$ and $\mathbf{u} \in \mathcal{X}^\perp$. Therefore, from Pythagoras's theorem, we have

$$\|\mathbf{w}^*\|^2 = \|\mathbf{w}\|^2 + \|\mathbf{u}\|^2 \ ,$$

and the assumption that $\mathbf{u}$ is nonzero then leads to the inequality

$$\|\mathbf{w}\| < \|\mathbf{w}^*\| \ .$$

Since the function R is strictly increasing, this in turn implies that

$$R(\|\mathbf{w}\|) < R(\|\mathbf{w}^*\|) \ . \tag{5}$$

In addition, since $\mathbf{u}$ is orthogonal to every $\mathbf{x}_n$, we can write the following for every n:

$$\mathbf{w}^T \mathbf{x}_n + b = (\mathbf{w}^* - \mathbf{u})^T \mathbf{x}_n + b = (\mathbf{w}^*)^T \mathbf{x}_n - \mathbf{u}^T \mathbf{x}_n + b = (\mathbf{w}^*)^T \mathbf{x}_n + b \ .$$

Therefore,

$$S\left(\mathbf{w}^T \mathbf{x}_1 + b, \ \ldots, \ \mathbf{w}^T \mathbf{x}_N + b\right) = S\left((\mathbf{w}^*)^T \mathbf{x}_1 + b, \ \ldots, \ (\mathbf{w}^*)^T \mathbf{x}_N + b\right) \ .$$

Combining this equation and the inequality 5 into the definition 4 of $L(\mathbf{w})$ yields

$$L(\mathbf{w}, b) < L(\mathbf{w}^*, b) \ .$$

This contradicts the assumption that $\mathbf{w}^*$ is optimal, and therefore $\mathbf{u}$ must be zero.

This completes the proof.

Of course, since this result was derived for *any* value of the bias term b, it also holds when $b = b^*$, the optimal value.

[2]Since the vectors $\mathbf{x}_1, \ldots, \mathbf{x}_N$ are generally not linearly independent, the coefficients $\beta_1, \ldots, \beta_N$ are generally not unique, but this fact does not matter for this proof.

Support Vectors A practically advantageous consequence of the representer theorem is *sparsity*. To define and understand this concept, suppose that the SVM classifies a training sample $(\mathbf{x}_n, y_n)$ correctly and with a margin of at least equal to the reference margin μ^*. Then, the definition of sample margin

$$\mu_{(\mathbf{w},b^*)}(\mathbf{x}, y) \;=\; \mu^* y \left[(\mathbf{w}^*)^T \mathbf{x} + b^*\right]$$

implies that

$$y_n \left[(\mathbf{w}^*)^T \mathbf{x}_n + b^*\right] > 1$$

and therefore that sample incurs zero loss:

$$\ell_{(\mathbf{w}^*,b^*)}(\mathbf{x}_n, y_n) \;=\; \rho(1 - y_n[(\mathbf{w}^*)^T \mathbf{x}_n + b^*]) \;=\; \max\{0, 1 - y_n[(\mathbf{w}^*)^T \mathbf{x}_n + b^*]\} = 0 \; .$$

In other words, as long as a sample is classified correctly and with sufficient margin (greater than μ^*) it has no influence on the solution: The parameters $\mathbf{w}^*, b^*$ of the optimal classifier's decision hyperplane depend only on the samples that are either classified incorrectly or classified correctly but with a margin smaller than or equal to μ^*. These "problem samples" are called the *support vectors*: They "support" the hyperplane in that they pin it down regardless of where the other samples are. Once the hyperplane has been found, the set of support vectors, which are typically sparse in the training set, are the only vectors that matter.

The first practical consequence of the existence of a relatively small set of support vectors is that they provide some intuition as to which training samples are difficult for the classifier to classify correctly, as they are either misclassified or classified with a small margin. Since support vectors are generally near the decision boundary, the designer of a classification system may look at pairs of support vectors that are near each other but have different (true) labels. These are examples of data points that are hard for the classifier to distinguish from each other, and they may suggest ways to improve the classifier. For instance, the designer may just provide more data points that are similar to those, in the hope to pin down the decision boundary more precisely. Or she may figure out what makes the two data points in a pair different from each other, and perhaps design a way to process the data to make those differences more salient, so that the training set becomes closer to being linearly separable.

A second advantage deriving from the representer theorem is deeper and more far-reaching from a conceptual standpoint, and is discussed in the next sections of this note. In preparation, we now show that *the representer theorem allows rewriting both the decision rule and the optimization target for SVMs in terms of inner products of data points (points in X)*. Specifically, by the representer theorem, the vector $\mathbf{w}^*$ in the unique solution of the SVM training problem can be written as follows:

$$\mathbf{w}^* = \sum_{n=1}^{N} \beta_n \mathbf{x}_n \; . \tag{6}$$

Replacing this expression for $\mathbf{w}^*$ into the decision rule 1 yields

$$\hat{y} = h(\mathbf{x}) = \text{sign}\left(\sum_{n=1}^{N} \beta_n \mathbf{x}_n^T \mathbf{x} + b^*\right) \tag{7}$$

and doing the same with the risk function in equation 3 yields

$$L_T(\mathbf{w}, b) \;=\; \frac{1}{2} \sum_{m,n=1}^{N} \beta_m \beta_n \mathbf{x}_m^T \mathbf{x}_n + \frac{C_0}{N} \sum_{n=1}^{N} \rho\left(1 - y_n \left(\sum_{m=1}^{N} \beta_m \mathbf{x}_m^T \mathbf{x}_n + b\right)\right) \tag{8}$$

where ρ is the hinge function.

As a consequence, *both training and testing involve the data samples* $\mathbf{x}_n$ *(or, for points outside the training set,* $\mathbf{x}$*) exclusively through their inner products* $\mathbf{x}_m^T\mathbf{x}_n$ *(or* $\mathbf{x}_n^T\mathbf{x}$*).* We now explore why this fact is so important.

2 Augmented Features

We saw in an earlier note that whether the samples in a binary classification problem are linearly separable depends on the representation of the data space X. For instance, consider a problem in which $X = \mathbb{R}^2$ and the decision boundary is the parabola $x_2 = x_1^2$, with the samples with positive label in the convex region delimited by the parabola. Then, the classification rule is

$$h(\mathbf{x}) = \begin{cases} 1 & \text{if } x_2 \geq x_1^2 \\ -1 & \text{otherwise.} \end{cases}$$

These samples are of course not linearly separable. However, one can transform the data by replacing every point $\mathbf{x} = (x_1, x_2)$ with a new point $\mathbf{z} = (z_1, z_2) = (x_1^2, x_2)$, and now the decision boundary becomes $z_2 = z_1$, which is a straight line. The new set of samples is linearly separable.

An alternative to *changing* the representation, as we did above, is to *augment* it instead. For instance, we could have defined $\mathbf{z} = (z_1, z_2, z_3) = (x_1, x_2, x_1^2)$. Here, we *added* the feature x_1^2 as a new component into the vector, rather than replacing x_1 with it. The new samples $(\mathbf{z}, y)$ are linearly separable: There is a line that separates the projections $((z_2, z_3), y)$ and therefore a plane (spanned by the line and a vector along the z_1 axis) that separates the samples $(\mathbf{z}, y)$ themselves.

Augmentation seems wasteful: The new representation is redundant, and computations become more complex. In addition, augmentation increases the dimensionality of X, and we are now courting the curse of dimensionality. In other words, we increased both the computational complexity (how many operations to perform) and the sample complexity (how many training samples are needed for training given a performance target) of the problem. Why think of feature augmentation, then? The reason is that, when faced with a new classification problem, we do not know the decision boundary, so it is generally hard to guess a transformation that will make the problem linearly separable if it is not. Instead, we can add *many* transformations of the variables, in the hope that a combination of some of those might make the problem linearly separable.

For instance, we could add all monomials of the original variables up to some degree k. If $k = 3$ and $\mathbf{x} = (x_1, x_2)$, we have

$$d' = \binom{d+k}{d} = \binom{2+3}{2} = 10$$

monomials of degree up to three in two variables:

$$1\,,\ \ x_1\,,\ \ x_2\,,\ \ x_1^2\,,\ \ x_1x_2\,,\ \ x_2^2\,,\ \ x_1^3\,,\ \ x_1^2x_2\,,\ \ x_1x_2^2\,,\ \ x_2^3\,. \tag{9}$$

From Taylor's theorem, we know that if the degree k is large enough, we can approximate *any* hypersurface to any degree of accuracy. Therefore, it is more than a *hope* that augmentation will lead to linear separability if carried far enough: It's a theorem!

But what about sample complexity and computational complexity? Defining sample complexity quantitatively would require a somewhat lengthy detour into the relevant theory. That detour is left to Appendix A, which is optional reading. The bottom line is rather striking: *If* the data space

$X \subset \mathbb{R}^d$ for an SVM classifier is bounded, that is, if there exists a sufficiently large d-dimensional hypersphere that contains all of X, then *the sample complexity of SVMs does not depend on the dimensionality d of X*. Thus, as long as the data space for an SVM is bounded, we can augment features to our heart's desire without incurring the curse of dimensionality! This result does *not* hold for linear-regression classifiers.

On the other hand, as the augmented data vectors $\mathbf{z}_n$ get longer and longer it takes more and more time to process the data, both during training and during inference. That is, the computational complexity increases. Fortunately, the representer theorem comes to the rescue through the notion of *kernels*. Well beyond giving us inexpensive computation, kernels also enhance the expressive power of SVMs in major ways, transforming them from linear classifiers to classifiers with potentially arbitrarily complex and nonlinear decision boundaries. These topics are discussed next.

3 Kernels

The idea of feature augmentation can be formulated abstractly by saying that instead of using the original feature vector $\mathbf{x}$ as a data point, we use some higher-dimensional feature vector $\mathbf{z} = \boldsymbol{\varphi}(\mathbf{x})$ obtained by some transformation of $\mathbf{x}$. Thus, we can see $\boldsymbol{\varphi}$ as a feature mapping from $\mathbb{R}^d$ to $\mathbb{R}^{d'}$, where $d' > d$, and often $d' \gg d$. If we use these new features, equations 7 and 8 can be written as follows

$$\hat{y} \;=\; h(\mathbf{x}) \;=\; \text{sign}\left(\sum_{n=1}^{N} \beta_n K(\mathbf{x}_n, \mathbf{x}) + b^*\right) \tag{10}$$

$$L_T(\mathbf{w}, b) \;=\; \frac{1}{2}\sum_{m,n=1}^{N} \beta_m \beta_n K(\mathbf{x}_m, \mathbf{x}_n) + \frac{C_0}{N}\sum_{n=1}^{N} \rho\left(1 - y_n\left(\sum_{m=1}^{N} \beta_m K(\mathbf{x}_m, \mathbf{x}_n) + b\right)\right) \tag{11}$$

where

$$K(\mathbf{x}, \boldsymbol{\xi}) \stackrel{\text{def}}{=} \boldsymbol{\varphi}(\mathbf{x})^T \boldsymbol{\varphi}(\boldsymbol{\xi})\ .$$

For given $\mathbf{x}$ and $\boldsymbol{\xi}$ this quantity is a single number, regardless of the dimensionality d' of the codomain of $\boldsymbol{\varphi}$. Importantly, all the training and inference algorithms need to know is $K(\mathbf{x}_m, \mathbf{x}_n)$ and $K(\mathbf{x}_n, \mathbf{x})$, because the individual (high-dimensional!) terms $\boldsymbol{\varphi}(\mathbf{x}_n)$ and $\boldsymbol{\varphi}(\mathbf{x})$ do not appear anywhere in isolation.

The first, relatively minor way for exploiting this observation is that if we have a mapping $\boldsymbol{\varphi}$ in mind and somehow we find an inexpensive way to compute the inner product $K(\mathbf{x}_m, \mathbf{x}_n)$, then we can save computation. For instance, it is not clear how to do this with the monomials listed in expression 9. However, let us replace these by the following:

$$\boldsymbol{\varphi}(\mathbf{x}) = \boldsymbol{\varphi}(x_1, x_2) = (1\,,\ \sqrt{3}x_1\,,\ \sqrt{3}x_2\,,\ \sqrt{3}x_1^2\,,\ \sqrt{6}x_1x_2\,,\ \sqrt{3}x_2^2\,,\ x_1^3\,,\ \sqrt{3}x_1^2x_2\,,\ \sqrt{3}x_1x_2^2\,,\ x_2^3)^T\ .$$

These monomials are scaled versions of the original ones, and therefore span the same space. With this new definition, however, we can write

$$\begin{aligned}(\mathbf{x}^T\boldsymbol{\xi} + 1)^3 &= (x_1\xi_1 + x_2\xi_2 + 1)^3 \\ &= 1 + 3x_1\xi_1 + 3x_2\xi_2 + 3x_1^2\xi_1^2 + 6x_1x_2\xi_1\xi_2 + 3x_2^2\xi_2^2 + x_1^3\xi_1^3 \\ &\quad +3x_1^2x_2\xi_1^2\xi_2 + 3x_1x_2^2\xi_1\xi_2^2 + x_2^3\xi_2^3 \\ &= \boldsymbol{\varphi}(\mathbf{x})^T\boldsymbol{\varphi}(\boldsymbol{\xi}) = K(\mathbf{x}, \boldsymbol{\xi})\ .\end{aligned}$$

Thus, instead of computing the ten products and nine sums necessary to compute the inner product of $\boldsymbol{\varphi}(\mathbf{x})$ ad $\boldsymbol{\varphi}(\boldsymbol{\xi})$, we just compute the four products and two sums needed to compute $(\mathbf{x}^T\boldsymbol{\xi}+1)^3$. This is of course a modest saving. However, as both the degree k and the number d of variables increase, the savings become exponentially large.

On the other hand, there is a much neater and much more useful idea for exploiting the observations made earlier: We can just come up with some *kernel* $K(\mathbf{x},\boldsymbol{\xi})$, without knowing what transformation $\boldsymbol{\varphi}$ generates it! Of course, we cannot just use *any* function K of two vectors $\mathbf{x},\boldsymbol{\xi}\in\mathbb{R}^d$, because K needs to be indistinguishable from an inner product. For instance, inner products are symmetric:

$$\mathbf{x}^T\boldsymbol{\xi}=\boldsymbol{\xi}^T\mathbf{x}$$

and satisfy the Cauchy-Schwartz inequality:

$$(\mathbf{x}^T\boldsymbol{\xi})^2\leq\|\mathbf{x}\|^2\ \|\boldsymbol{\xi}\|^2\ ,$$

and therefore a valid kernel $K(\mathbf{x},\boldsymbol{\xi})$ must satisfy

$$K(\mathbf{x},\boldsymbol{\xi})=K(\boldsymbol{\xi},\mathbf{x})\quad\text{and}\quad K^2(\mathbf{x},\boldsymbol{\xi})\leq K(\mathbf{x},\mathbf{x})\ K(\boldsymbol{\xi},\boldsymbol{\xi})\ .$$

However, these conditions are not enough to guarantee that $K(\mathbf{x},\boldsymbol{\xi})$ is the inner product in some vector space.

Fortunately, there is a characterization of all valid kernels in the literature [4]. If we have a finite set of vectors $\mathbf{x}_n\in\mathbb{R}^d$ (as we do with a training set), then a symmetric function $K(\mathbf{x},\boldsymbol{\xi})\ :\ \mathbb{R}^d\times\mathbb{R}^d\to\mathbb{R}$ is a kernel function on that set if and only if the matrix with entry i,j equal to $K(\mathbf{x}_i,\mathbf{x}_j)$ is positive semi-definite.

There is also an analogous result for infinite sets. This is important, because we would like to know that a given $K(\mathbf{x},\boldsymbol{\xi})$ is a kernel without first having to choose a training set. In the continuous case, a continuous, symmetric function $K(\mathbf{x},\boldsymbol{\xi})$ is a kernel function, that is, it can be written as

$$K(\mathbf{x},\boldsymbol{\xi})=\boldsymbol{\varphi}(\mathbf{x})^T\boldsymbol{\varphi}(\boldsymbol{\xi})$$

for some mapping $\boldsymbol{\varphi}$ from $\mathbb{R}^d$ to $\mathbb{R}^{d'}$ if and only if for every function $f\ :\ \mathbb{R}^d\to\mathbb{R}$ for which the integral

$$\int_{\mathbb{R}^d}f(\mathbf{x})\,d\mathbf{x}\quad\text{is finite,}$$

the following condition holds:

$$\int_{\mathbb{R}^d\times\mathbb{R}^d}K(\mathbf{x},\boldsymbol{\xi})\,f(\mathbf{x})\,f(\boldsymbol{\xi})\,d\mathbf{x}\,d\boldsymbol{\xi}\ \geq\ 0\ .$$

This condition is called *Mercer's condition*, and you should be able to recognize it as an immediate extension of the notion of a positive semi-definite matrix to the continuous case. There is a whole theory that lets one check algorithmically whether this condition holds, and the theory is based on eigenfunctions, the functional equivalent of eigenvectors [4]. As a very important example, the *Gaussian kernel*

$$K(\mathbf{x},\boldsymbol{\xi})=e^{-\frac{\|\mathbf{x}-\boldsymbol{\xi}\|^2}{\sigma^2}}\tag{12}$$

can be shown to satisfy Mercer's condition, and is therefore a kernel. SVMs that use a Gaussian kernel are called *Radial Basis Function (RBF) SVMs*. Many ways to build other kernels can be found in the literature [1, 2].

Kernels, Support Vectors, and Decision Boundaries The kernel idea implies that rather than being restricted to the inner product $K(\mathbf{x}, \boldsymbol{\xi}) = \mathbf{x}^T \boldsymbol{\xi}$ when working with SVMs, we can replace $K(\cdot, \cdot)$ by *any* function $\mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ that satisfies Mercer's condition. Why is this useful? The main reason is that instead of finding a decision hyperplane, an SVM that uses kernels can find decision hyper-surfaces that are potentially arbitrarily complex and nonlinear.

The decision rule for an SVM using kernel K is (equation 10)

$$h(\mathbf{x}) \;=\; \text{sign}\left(\sum_{n=1}^{N} \beta_n K(\mathbf{x}_n, \mathbf{x}) + b^*\right)$$

and the decision hyper-surface has therefore equation

$$\sum_{n=1}^{N} \beta_n K(\mathbf{x}_n, \mathbf{x}) + b^* = 0 \; .$$

When K is the inner product, this hyper-surface is a hyperplane. For different choices of K, on the other hand, the hyper-surface can have a very different shape. Consider for instance the Gaussian kernel in equation 12. The equation of the decision hyper-surface then reads

$$\sum_{n=1}^{N} \beta_n e^{-\frac{\|\mathbf{x}-\mathbf{x}_n\|^2}{\sigma^2}} + b = 0 \; . \tag{13}$$

One can prove that in that case the values of β_n for the support vectors with true positive label ($y_n = 1$) are positive, and those for the support vectors with a true negative label ($y_n = -1$) are negative [1, 2].

This fact leads to an interesting geometric interpretation of equation 13: Place a positive Gaussian, scaled by β_n and with parameter σ, at every support vector $\mathbf{x}_n$ with positive true label, and place a negative Gaussian, scaled by β_n and with parameter σ, at every support vector $\mathbf{x}_n$ with negative true label. The decision hyper-surface is the point-by-point sum of all these functions. Thus, the positive support vectors are "pulling up" the score function, and the negative support vectors are "pulling it down." The decision hyper-surface in the original space is the set of hyper-surfaces where this sum equals $-b$. Figure 1 illustrates.

Thus, with enough support vectors, an RBF SVM can follow arbitrarily complex boundaries, and the boundary weaves its way around the Gaussians between positive and negative support vectors.

Training Kernel SVMs While the decision boundary for a kernel SVM is the solution of an equation that is nonlinear in the data points $\mathbf{x}_n$, the equation is still affine in the coefficients β_n that appear in the risk function L_T in equation 11. In addition, the risk function is convex for the same reasons that the original risk function for linear SVMs is convex, namely, that L_T is the sum of two convex functions in the coefficients β_n: The regularization term is quadratic and therefore convex. The average hinge loss is the composition of an affine function and a convex function (the hinge function), and is therefore convex as well.

Because of this, we can again use gradient descent in one of its versions to minimize the risk L_T and train the SVM. However, the number of unknowns β_n is now N, the number of training samples, rather than $d+1$, where d is the dimensionality of the data space. Since typically $N \gg d$, the stochastic version of gradient descent is even more attractive here. As for the linear case, even in the kernel case other formulations are possible and lead to different optimization methods.

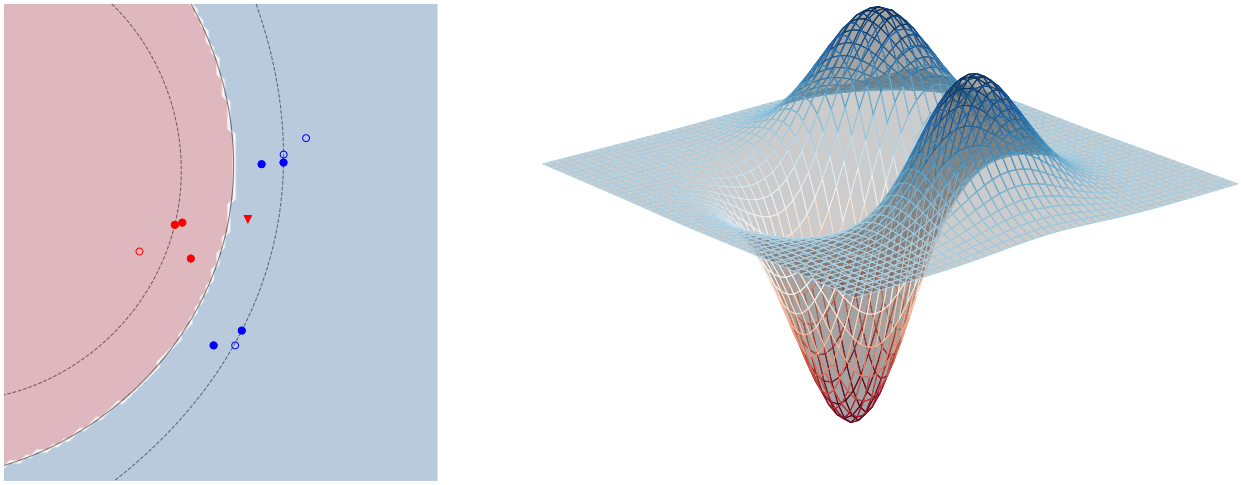

Figure 1: The geometry of an RBF SVM for a toy two-dimensional classification problem. *Left:* The SVM classifies any data point $\mathbf{x}$ in the red region as negative, and any data point in the blue region as positive. The solid line between the two regions is the decision boundary. (A hypersurface in two dimensions is a curve.) The two dashed curves delimit the reference margin. The red symbols are negative samples from the training set and the blue symbols are positive samples. Hollow circles are samples that are classified correctly and are outside the reference margin. Filled circles are classified correctly but are within the reference margin. The red triangle is a misclassified negative training sample. Thus, the eight filled symbols (circles or triangle) are support vectors. In bigger problems, there are typically many fewer support vectors than other vectors.

Right: A 3D view of the decision score $s(\mathbf{x}) = \sum_{n=1}^{N} \beta_n e^{-\frac{\|\mathbf{x}-\mathbf{x}_n\|^2}{\sigma^2}} + b$ for the same SVM. Negative values are in red, positive ones in blue. Each positive support vector (blue filled circles in the diagram on the left) contributes a positive Gaussian centered on it. However, only two blue bells are visible on the right because any two Gaussians centered at nearby points add up to a single bell. The four negative Gaussians centered at the four negative support vectors (red filled symbols on the left) are close enough to each other to merge into a single negative bell. The zero-crossing of the surface on the right (that is, the curve with equation $s(\mathbf{x}) = 0$) is the decision boundary, corresponding to the solid black curve on the left. The training samples that are classified correctly and are outside the reference margin (hollow circles in the diagram on the left) do not contribute to the score function $s(\mathbf{x})$.

References

[1] N. Cristianini and J. Shawe-Taylor. *An introduction to support vector machines.* Cambridge University Press, Cambridge, UK, 2000.

[2] B. Schölkopf and A. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization and Beyond.* MIT Press, Cambridge, MA, 2002.

[3] S. Shalev-Shwartz and S. Ben-David. *Understanding Machine Learning: From Theory to Algorithms.* Cambridge University Press, Cambridge, UK, 2014.

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

Appendix

A Sample Complexity

This Section conveys the gist of what would take several lectures to work out in detail, so there will be no proofs. The goal of this Appendix is to explain what "sample complexity" means for a machine learning problem. If you are interested, you can find a good treatment of these topics in a recent book [3].

Intuitively, the more training samples we have, the better a given training algorithm should be able to do in terms of generalization, because a large training set is a more faithful and detailed proxy for all the data we might encounter in the future. Sample complexity measures how fast the size N of a training set needs to grow as we make more and more exacting demands on the ability of the classifier to generalize. Formalizing this notion requires some care, as we now see.

Recall that we assume that all data, both that in the training set and that which we encounter when we deploy the algorithm, comes from a given but unknown probability distribution $p(\mathbf{x}, y)$, which we called the *model* of the problem. At training time, we are given a training set T of size N, and the training algorithm finds the classifier $\hat{h}$ in the hypothesis space $\mathcal{H}$ that minimizes the empirical risk $L_T(h)$:

$$\hat{h} = ERM_T(\mathcal{H}) \in \arg\min_{h \in \mathcal{H}} L_T(h) \ .$$

Ideally, the empirical risk achieved by $\hat{h}$ on T is the smallest one on $\mathcal{H}$, so that

$$L_T(\hat{h}) = L_T(\mathcal{H}) = \min_{h \in \mathcal{H}} L_T(h) \ .$$

When deployed, that classifier will achieve a *statistical* risk

$$L_p(\hat{h}) = \mathbb{E}_p[\ell(y, \hat{h}(\mathbf{x}))]$$

that is generally higher than the empirical risk $L_T(\hat{h})$, because $\hat{h}$ is likely to overfit T to some degree.

More importantly, the statistical risk $L_p(\hat{h})$ achieved by training our classifier on T will also be higher than the smallest achievable *statistical* risk in $\mathcal{H}$ and under the given model $p(\mathbf{x}, y)$,

$$L_p(\mathcal{H}) = \min_{h \in \mathcal{H}} L_p(h) \quad \text{where} \quad L_p(h) = \mathbb{E}_p[\ell(y, h(\mathbf{x}))] \ .$$

The statistical risk $L_p(\hat{h})$ is higher than $L_p(\mathcal{H})$ because $\hat{h}$ was trained on T, and T is a poor proxy for $p(\mathbf{x}, y)$.

> **Notation:** There are at least three risks here, so let us keep them straight. To summarize: The classifier $\hat{h}$ is found by minimizing the empirical risk on T, resulting in a value $L_T(\hat{h})$ for the empirical risk. This classifier achieves a statistical risk $L_p(\hat{h})$ when deployed, and this statistical risk is typically greater than both the empirical risk $L_T(\hat{h})$ on T and the smallest achievable statistical risk $L_p(\mathcal{H})$ for the given model p and hypothesis space $\mathcal{H}$, because T represents p poorly.

The smallest achievable statistical risk $L_p(\mathcal{H})$ is generally not equal to zero. For instance, if $\mathcal{H}$ is the set of linear classifiers and the data is not linearly separable, then there is no way to achieve zero risk. However, the closer the classes are to being linearly separable, the smaller $L_p(\mathcal{H})$ will be for that $\mathcal{H}$.

Thus, in general, all we can hope to achieve is some statistical risk $L_p(\hat{h})$ that is equal to $L_p(\mathcal{H}) + \epsilon$ for some $\epsilon > 0$:

$$L_p(\hat{h}) = L_p(\mathcal{H}) + \epsilon \geq 0 \; .$$

We would like ϵ to be small, and the price we must pay to make ϵ smaller is to make the training set T bigger, so that it is a more faithful representation of all the data we have not yet seen.

Loosely speaking, we can view N, the size of T, as a monotonically increasing function of $1/\epsilon$, which we think of as the degree of exactitude we want to achieve: To obtain a large value of $1/\epsilon$ (that is, a small ϵ), we need to make N large enough. Then, we say that a machine learning problem has a low sample complexity if N grows slowly with $1/\epsilon$, that is, as ϵ shrinks.

A key difficulty in formalizing this notion, that is, when trying to speak less loosely, is that it is not possible to have absolute guarantees: The N samples in T are drawn at random out of $p(\mathbf{x}, y)$, and if we are really unlucky we may end up with a particularly bad T (a "statistical fluke"). In that case, the classifier $\hat{h}$ will perform more poorly than if we were given a "more typical" set of N random samples out of p.

Because of this, we cannot demand to be within ϵ from $L_p(\mathcal{H})$ with absolute certainty, but we must settle for a probabilistic guarantee. Specifically, we ask to fall below risk $L_p(\mathcal{H}) + \epsilon$ *with high probability*. More formally, in addition to a number ϵ with $0 < \epsilon$, we also give a positive number δ with $0 < \delta < 1$, and we ask how large N needs to be for it to be unlikely (as measured by the probability δ) that the actual statistical risk $L_p(\hat{h})$ achieved by the classifier $\hat{h}$ is greater than our target $L_p(\mathcal{H}) + \epsilon$. In a formula, we ask for the following inequality to hold:

$$\mathbb{P}[L_p(\hat{h}) \geq L_p(\mathcal{H}) + \epsilon] \leq \delta \; . \tag{14}$$

Note the strict inequalities on δ: This number is assumed to be less than 1, because it bounds a probability, so with $\delta = 1$ the bound would be moot. This number is also strictly positive, because we can never be certain to achieve the desired bound.

We are now ready to define sample complexity: The *sample complexity* of a given machine learning problem is the function $N_{\mathcal{H}}(\epsilon, \delta)$ that specifies the smallest number N of samples that are necessary to satisfy the inequality 14 for the given hypothesis space $\mathcal{H}$ and *regardless of the model* p. That is, $N(\epsilon, \delta)$ must be so large that the bound 14 holds for all p.

As an example, suppose that $\epsilon = 0.1$ and $\delta = 0.01$. Then, if N is large enough for the inequality above to hold, we know that the probability that the loss is more than 0.1 above the minimum

possible loss $L_p(\mathcal{H})$ is less than 1 percent. Stated differently, the loss falls within 0.1 from optimal with 99 percent probability.

The requirement that the bound be independent of the model $p(\mathbf{x}, y)$ is rather strong. As a result, we will typically be able to bound N from below ("it takes at least this many samples") and with an asymptotic bound, rather than giving an exact expression for it.

Sample Complexity for Linear Classifiers and for SVMs Now that we know how to *define* sample complexity, let us see, without proof, what the complexity is for binary linear classifiers in general and for SVMs in particular. We will see that SVMs come with much better guarantees under a weak assumption on X.

For binary linear classifiers, it turns out that N grows at least as fast as the following function:

$$\frac{d + \log(1/\delta)}{\epsilon} .$$

Thus, N must grow linearly with $1/\epsilon$ and with the logarithm of $1/\delta$ for the bound 14 to hold. More importantly, N must grow linearly with the dimension d of the data space X: The more dimensions, the more data we need. This linear dependence is not too bad, given what we know about the curse of dimensionality, and is the main reason why linear classifiers are so successful.

For SVMs, on the other hand, the situation is even better, *if* the data space $X \subset \mathbb{R}^d$ is bounded[3], that is, if there exists a sufficiently large d-dimensional hypersphere that contains all of X. A formula for the actual sample complexity is harder to give in this case, because it depends on the radius of the hypershpere that bounds X, the parameter C used for regularization of the SVM training risk, and the margin between the two classes. However, and most importantly, *the sample complexity of SVMs does not depend on the dimensionality d of X.*

Because of this advantage, when learning SVMs we need not worry about d, and we are effectively free from the curse of dimensionality, as long as the data space X is bounded! The implication for data augmentation is that *we do not need to worry about sample complexity when we add features to the data points* $\mathbf{x}$.

[3]This assumption is useful for SVMs, but would not help for general linear classifiers.