Lecture notes on Support Vector Machines

These notes contain a gentle introduction to *Support Vector Machines* for the problem of binary classification. The notes present the key concepts that the development of SVMs brought to machine learning. First, we will cover the notion of *margin* when presenting the most simple case, namely, the case of linearly separable data. Then, we will generalize this to *soft margin SVM* which in essence is a form of regularized SVM that balances the size of the margin with the presence of a small number of "mistakes". And finally, we will introduce the so-called *kernel trick* which achieves non-linear classification using the linear machinery of the SVM.

A brief history on SVMs. Support Vector Machines were originally invented by Vapnik and Chervonenkis in the 70s but it was not until 20 years later that through the introduction of the *kernel trick* they were popularized. It was a revolution and completely displaced neural networks which was the method of choice at the time for a good 20 years. The benefits of SVMs with respect to neural networks were many: they were very much rooted in rigorous statistical and optimization theory, they did not suffer from local minima, and they were robust especially in the presence of limited data. Of course, this changed with the advent of deep learning and the availability of large volumes of data and computational resources.

Linearly separable data: hard margins

A *support vector machine* or SVM for short is a method for finding a linear hyperplane that separates examples from different classes. In the following picture, you can see some data on two dimensions, and each data point is labelled with a class (circles or squares). On the left, there are several separating lines (hyperplanes in general) which achieve this goal. On the right, you see the hyperplane that the SVM would find, which is the one that *maximizes the margin*, namely, the one that is at most distance from any point in the data. This notion of *maximum margin* is at the core of the definition of SVM, and it is based on learning theory associating the size of the margin to the generalization capability of the resulting hyperplane. For those of you interested, this is based on the VC-dimension (Blumer et al. 1989; Vapnik 2000).

So, in essence, an SVM is an optimization method that, given a (binary) labelled training set, it will produce the separating hyperplane with maximum margin. Contrast this with e.g. the Perceptron, which produces *any* separating hyperplane. Let us start introducing notation and main ideas.

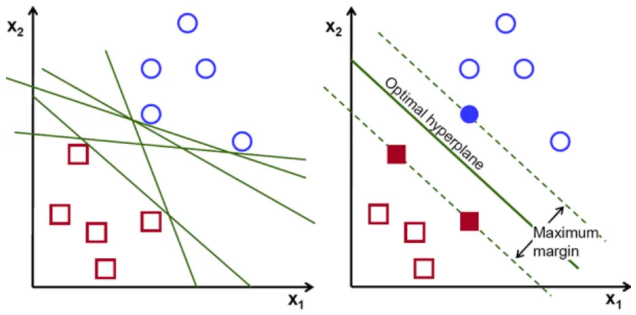


Figure 1: Picture taken from here

As usual, the input dataset is given by a finite set of example-label pairs: $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$ where each $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \{-1, +1\}$.

We assume for the moment that the input data are *linearly separable* and thus there exists at least one separating hyperplane given by a vector of coefficients $\mathbf{w} \in \mathbb{R}^d$ and a bias $b \in \mathbb{R}$ such that, for every $i = 1, \ldots, n$:

$$\begin{cases} \mathbf{w}^T\mathbf{x}_i + b > 0 & \text{if } y_i = +1 \\ \mathbf{w}^T\mathbf{x}_i + b < 0 & \text{if } y_i = -1 \end{cases}$$

These two conditions can be summarized with the following condition:

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) > 0.$$

Actually, the quantity $y_i(\mathbf{w}^T\mathbf{x}_i + b)$ is a very important notion, it not only tells us whether there was a mistake on $\mathbf{x}_i$ (if $y_i(\mathbf{w}^T\mathbf{x}_i + b) < 0$) but its magnitude tells us how far apart it is from the separating hyperplane, and thus, how *confident* we may be in its predicted classification. This quantity is known as the *functional margin* of $\mathbf{x}_i$.

The geometry of our separating hyperplane is useful to derive the statement of the optimization problem:

The *distance* of a point $\mathbf{x}_i$ to the separating hyperplane given by parameters $(\mathbf{w}, b)$, is given by $\frac{|\mathbf{w}^T\mathbf{x}_i + b|}{\|\mathbf{w}\|}$, and it also known as the *geometric margin* of $\mathbf{x}_i$. Since scaling $(\mathbf{w}, b)$ still results in the same separating hyperplane, we choose the representation for our separating hyperplane that leaves at distance 1 the closest data point, which is called a *support vector*. So, the closest points to our separating hyperplane are those that lie on the lines $\mathbf{w}^T\mathbf{x} + b = \pm 1$, and thus a *support vector* is a point $\mathbf{x}_i$ such that $\mathbf{w}^T\mathbf{x}_i + b = +1$ or $\mathbf{w}^T\mathbf{x}_i + b = -1$. Note that for all non support vectors we have that $|\mathbf{w}^T\mathbf{x}_i + b| > 1$. The *margin* of this hyperplane is $\frac{2}{\|\mathbf{w}\|}$.

Summarizing, the important points in the geometry of the problem are:

- a separating hyperplane is given by parameters $\mathbf{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$

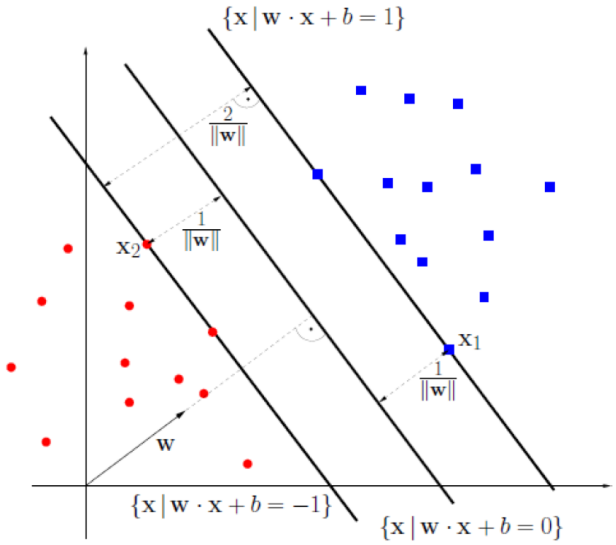


Figure 2: Geometry of the separating hyperplane scaled to be at distance 1 from support vectors

- support vectors $\mathbf{x}_{sv}$ are the closest points to the separating hyperplane: $\mathbf{w}^T\mathbf{x}_{sv} + b = \pm 1$
- the separating hyperplane should separate all data correctly: $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$, for all $i = 1, \ldots, n$ This is called functional margin
- the margin of the separating hyperplane is given by $\frac{2}{\|\mathbf{w}\|}$

With the scaling restriction on our separating plane, the problem of finding the maximum-margin separating hyperplane for linearly separable data becomes a classic quadratic problem in constrained optimization:

$$\begin{aligned} \min_{\mathbf{w},b} \quad & \frac{1}{2}\|\mathbf{w}\|^2 \\ \text{subject to:} \quad & y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \quad \forall i = 1, \ldots, n \end{aligned}$$

This is known as the *primal problem* of hard-SVM, and it can be solved using classical theory of constrained optimization, we will not include its full resolution here but this can be seen in many tutorials and other sources. A good one is, for examaple, this very nice lecture on SVMs from MIT. We will mainly hint to how its *dual problem* (lagrangian) is built and what the interpretation on the lagrangian multipliers is.

The lagrangian introduces one dual variable α_i for each constraint $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$ and thus for each training point $\mathbf{x}_i$. The lagrangian becomes:

$$L(\mathbf{w}, b, \{\alpha_i\}_{i=1}^n) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{n} \alpha_i(y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1)$$

This is a convex, quadratic function and therefore has a unique minimum. Necessary conditions for the solution are given by:

- $\nabla_{\mathbf{w}} L(\mathbf{w}, b, \{\alpha_i\}_{i=1}^n) = \mathbf{w} - \sum_i \alpha_i y_i \mathbf{x}_i = 0$, and so: $\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$
- $\nabla_b L(\mathbf{w}, b, \{\alpha_i\}_{i=1}^n) = \sum_i \alpha_i y_i = 0$

Substituting this back into the lagrangian we obtain the *dual problem* of hard-SVM, which is:

$$\max_{\{\alpha_i\}_{i=1}^n} \quad \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$
$$\text{subject to:} \quad \alpha_i \geq 0 \quad \forall i = 1, \ldots, n \quad \text{and} \quad \sum_i \alpha_i y_i = 0$$

Notice that this optimization problem only depends on the α_i; it can be solved using standard quadratic programming methods. At the maximum, the *complementary slackness conditions* imply that, for all i:

- if $\alpha_i^\star > 0$, then $y_i(\mathbf{w}^\star \mathbf{x}_i + b^\star) = 1$ (i.e., $\mathbf{x}_i$ is a *support vector*)
- if $y_i(\mathbf{w}^\star \mathbf{x}_i + b^\star) > 1$, then $\alpha_i^\star = 0$

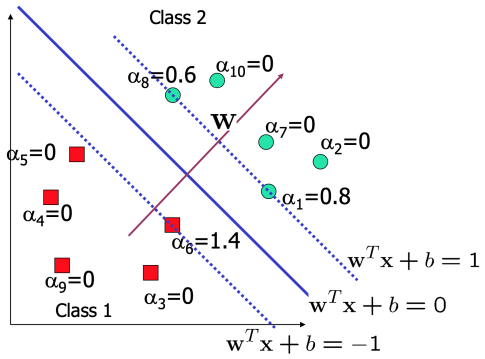


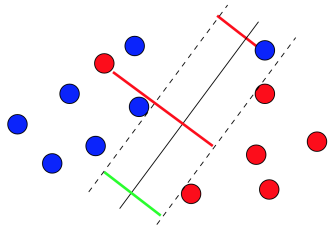
Figure 3: At the optimum, $\alpha_i > 0$ if and only if $\mathbf{x}_i$ is a *support vector*

Once we have found the optimal $\alpha_i^\star$, we can recover the original variables as in:

- $\mathbf{w}^\star = \sum_i \alpha_i^\star y_i \mathbf{x}_i$, and
- $b^\star = y_i - \mathbf{w}^\star \mathbf{x}_i$ for any *support vector* $\mathbf{x}_i$

Non-linearly separable data: soft margins

What happens when we have data that is not linearly separable?



In this case, we can no longer guarantee that there is a solution for which $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$ and so we must alter our formulation to contemplate *errors* in the classification. In essence, soft-margin SVM is a regularized version of the hard-margin SVM that allows to trade-off between large margins and few errors.

The *error* of a point $(\mathbf{x}_i, y_i)$ is given by its *functional margin*:

- if $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$, then the *error* on this example is 0
- if $y_i(\mathbf{w}^T\mathbf{x}_i + b) < 1$, then the *error* is $1 - y_i(\mathbf{w}^T\mathbf{x}_i + b)$

This error can be written in one single line as $l_i = \max(0, 1 - y_i(\mathbf{w}^T\mathbf{x}_i + b))$, which is known as the hinge loss.

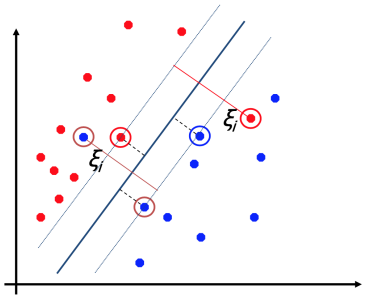


Figure 4: To deal with *errors* we introduce new *slack variables* ξ_i for each $\mathbf{x}_i$, which results in what is known as *soft margin*.

The resulting optimization problem becomes:

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

where C is a hyper-parameter that trades off large margin and error (regularizer).

Now, we can introduce lagrange multipliers α_i like in the case of hard-margin, and additional γ_i to deal with the $\xi_i \geq 0$ inequalities, and form the lagrangian:

$$L(\mathbf{w}, b, \xi_i, \alpha_i, \gamma_i) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i - \sum_i \alpha_i\{y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1 + \xi_i\} - \sum_i \gamma_i\xi_i$$

We again look at what happens at locations where the gradients vanish:

- $\nabla_{\mathbf{w}} L(\mathbf{w}, b, \xi, \alpha, \gamma) = 0$ implies $\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$
- $\nabla_b L(\mathbf{w}, b, \xi, \alpha, \gamma) = \sum_i \alpha_i y_i = 0$
- $\nabla_{\xi_i} L(\mathbf{w}, b, \xi, \alpha, \gamma) = 0$ implies $C - \alpha_i - \gamma_i = 0$

Substituting these equalities back into the lagrangian, we obtain the dual formulation for the optimization problem:

$$\begin{aligned} \max_{\{\alpha_i\}_{i=1}^n} \quad & \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \\ \text{subject to:} \quad & 0 \leq \alpha_i \leq C \quad \forall i = 1, \ldots, n \quad \text{and} \quad \sum_i \alpha_i y_i = 0 \end{aligned}$$

Note that the only difference w.r.t. the formulation of hard-margin is that we have an upper bound on the value that α_i takes. Note that we recover hard-margin SVM from this last formulation by setting $C = \infty$. Again, with standard software we can solve this quadratic program and obtain the optimal values for α_i. We can again recover the parameters for the separating hyperplane in the same manner as before:

- $\mathbf{w}^\star = \sum_i \alpha_i^\star y_i \mathbf{x}_i$, and
- $b^\star = y_i - \mathbf{w}^\star \mathbf{x}_i$ for any *support vector* $\mathbf{x}_i$ s.t. $0 < \alpha_i < C$.

In this case, the *complementary slackness conditions* imply that, for all i:

- $\alpha_i = 0$ implies $y_i(\mathbf{w}^T \mathbf{x}_i + b) > 1$ and so $\xi_i = 0$ ($\mathbf{x}_i$ is not a *support vector*)
- $0 < \alpha_i < C$ implies $y_i(\mathbf{w}^T \mathbf{x}_i + b) = 1$ and $\xi_i = 0$ ($\mathbf{x}_i$ is a *non-bound support vector*)
- $\alpha_i = C$ implies $y_i(\mathbf{w}^T \mathbf{x}_i + b) < 1$ and $\xi_i > 0$ ($\mathbf{x}_i$ is a *bound support vector*)

Non-linear classification: the *kernel trick*

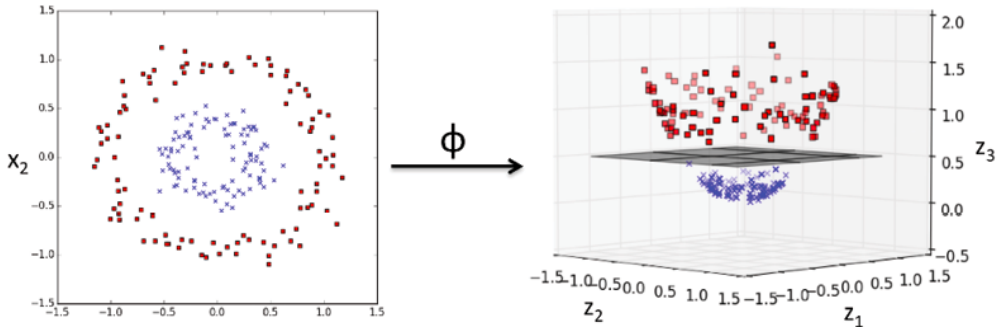


Figure 5: Non-linear feature map allowing for linear classification in augmented feature space.

The formulation of hard-margin and soft-margin SVM leads to a very good classifier for linearly separable or almost linearly separable data. The resulting classifier has good

generalization capability, especially if the margin achieved by the resulting classifier is large. However, SVM is inherently a linear classifier and its application to real data is limited due to this fact.

The *kernel trick* allows SVMs (and other linear models like logistic regression or Perceptron) to work in high-dimensional, non-linear feature spaces however using its linear classification machinery.

The *trick* part is that we may achieve this *for free*, at no extra preprocessing cost. This is done through the use of *kernel functions*, which we explain next.

But before we proceed, it is important to note that both in the hard-margin and soft-margin case, the resulting classifier is constructed by linearly combining input training data points: $\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$. On a new test point $\mathbf{x}$ the prediction according to this classifier is given by the following function:

$$\hat{y} = \text{sign}(\sum_{j=1}^{d} w_j x_j + b) = \text{sign}\left(\mathbf{w}^T \mathbf{x} + b\right) = \text{sign}\left(\sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i^T \mathbf{x} + b\right) = \text{sign}(\sum_{s \in S} \alpha_s y_s \mathbf{x}_s^T \mathbf{x} + b)$$

where S stands for the indices of the support vector set.

In order to make new predictions, we can store and use the solution weight vector $\mathbf{w}$, or we could just remember support vectors since they are the ones that will contribute non-zero terms to $\sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i \mathbf{x}'$.

Depending on d (number of features of $\mathbf{x}$) and the number of support vectors, it could be more convenient to just store the support vectors together with their non-zero α_i and their label y_i to make future predictions. If the solution is such that there are few support vectors, especially if the examples are very high-dimensional, it could be much better to work with the dual representation of the separating hyperplane given by the dual variables $\{\alpha_i\}_{i=1}^{n}$.

But the really interesting part of the above equation is that, when we work in *dual space* the only thing what we need to compute in order to make predictions on new examples $\mathbf{x}$ is the inner product between support vectors and the new test example: $\mathbf{x}_s^T \mathbf{x}$.

Now, suppose that there is some complex embedding of input examples $\mathbf{x} \in \mathbb{R}^d$ into a high-dimensional space $\Phi(\mathbf{x}) \in \mathbb{R}^M$, which we call *feature map* $\Phi : \mathbb{R}^d \to \mathbb{R}^M$, with $M \gg d$. Given such a *feature map*, we define its associated *kernel function* k_Φ:

$$k_\Phi(\mathbf{u}, \mathbf{v}) = \Phi(\mathbf{u})^T \Phi(\mathbf{u})$$

for any $\mathbf{u}, \mathbf{v} \in \mathbb{R}^d$. The inner product computed by the kernel operates in the expanded feature space, however, for some feature maps, *it is possible to compute the associated kernel in time that is independent from M*. In such cases, we can have SVM operate in the high-dimensional rich space without incurring in the extra cost of building the feature map explicitly, since it only needs *inner products*.

This goes for making predictions but also for the optimization problem in its dual formulation. For example, all we need to do in the more general case of soft-margin SVM is to solve it using an appropriate kernel function $k(\cdot, \cdot)$ instead of the inner product:

$$\begin{aligned} \max_{\{\alpha_i\}_{i=1}^n} \quad & \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \, k(\mathbf{x}_i, \mathbf{x}_j) \\ \text{subject to:} \quad & 0 \leq \alpha_i \leq C \quad \forall i = 1, \ldots, n \quad \text{and} \quad \sum_i \alpha_i y_i = 0 \end{aligned}$$

So, given a feature map Φ with its kernel function k_Φ, in order to make predictions, all we need to do is use the formula:

$$\hat{y} = \text{sign}(\sum_{s \in S} \alpha_s y_s k_\Phi(\mathbf{x}_s, \mathbf{x}) + b)$$

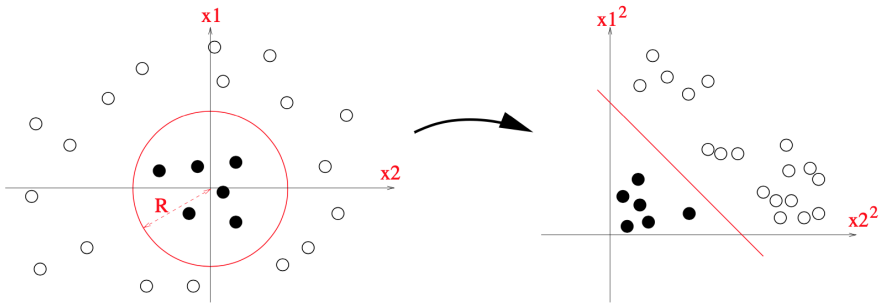
All we have done is to subtitute the inner product between test point and support vector $\mathbf{x}_s^T \mathbf{x}$ by their kernel value $k_\Phi(\mathbf{x}_s, \mathbf{x})$.

It is worth noting that many classical algorithms can be kernelized: Perceptron, logistic regression, SVMs of course, PCA, k-means, nearest neighbors, etc.

In what remains of this document, we are going to cover a few useful and common kernels.

The polynomial kernel

One of the most used kernels, it allows us to operate on the space of polynomials (of a given *degree d*). In the case of $d = 2$, the following figure shows how to extend the dimensionality in order to allow for *linear classification*:



In the quadratic case, the polynomial kernel of degree 2 is defined through the feature map $\Phi(\mathbf{x}) = (x_1^2, \sqrt{2}\, x_1 x_2, x_2^2)^T \in \mathbb{R}^3$ for any example $\mathbf{x} = (x_1, x_2)^T \in \mathbb{R}^2$. So the kernel is:

$$
\begin{aligned}
k_\Phi(\mathbf{u}, \mathbf{v}) &= \Phi(\mathbf{u})^T \Phi(\mathbf{v}) \\
&= (u_1^2, \sqrt{2}\, u_1 u_2, u_2^2)(v_1^2, \sqrt{2}\, v_1 v_2, v_2^2)^T \\
&= u_1^2 v_1^2 + \sqrt{2}\, u_1 u_2 \sqrt{2}\, v_1 v_2 + u_2^2 v_2^2 \\
&= (u_1 v_1 + u_2 v_2)^2 \\
&= (\mathbf{u}^T \mathbf{v})^2
\end{aligned}
$$

More generally, for $\mathbf{u}, \mathbf{v} \in \mathbb{R}^p$, we define the *polynomial kernel* of degree d as:

$$
k_{poly(d)}(\mathbf{u}, \mathbf{v}) \stackrel{\text{def}}{=} (\mathbf{u}^T \mathbf{v} + 1)^d
$$

The associated *feature map* is the space of all monomials of degree up to d.

Exercise: Define the feature map described above and show that the inner product in this expanded feature space is indeed $k_{poly(d)}$.

Other kernels for continuous data

There are several famous *kernels* for continuous data, the most popular one is perhaps the *Gaussian kernel* or *RBF kernel* which has one parameter (the *width* σ):

$$
k_{rbf(\sigma)}(\mathbf{u}, \mathbf{v}) = \exp^{-\frac{\|\mathbf{u}-\mathbf{v}\|^2}{2\sigma^2}}
$$

As you can see, kernels may have *parameters* which should be treated as *hyper-parameters*, so they should be tuned with the usual resampling methods such as *cross-validation* or any suitable alternative.

Actually, we may create new *kernels* by combining existing ones. Suppose that k_1 and k_2 are valid kernels operating on $\mathbb{R}^d$, k_3 is a valid kernel on $\mathbb{R}^M$, and $\Phi : \mathbb{R}^d \to \mathbb{R}^N$ is a feature map, then the following are all valid kernel functions:

- $k_1(\mathbf{u}, \mathbf{v}) + k_2(\mathbf{u}, \mathbf{v})$
- $ak_1(\mathbf{u}, \mathbf{v}) + b$ for any $a, b \geq 0$
- $k_1(\mathbf{u}, \mathbf{v})k_2(\mathbf{u}, \mathbf{v})$
- $\exp^{k_1(\mathbf{u}, \mathbf{v})}$

Exercise: Think of what *feature maps* induce the above new kernel definitions.

We finish this section by stating there there is a large body of theory on *kernels* and their relationship with *Reproducing Kernel Hilbert Spaces* which we will not cover here. The interested reader can refer to (Schölkopf and Smola 2002; Shawe-Taylor and Christianini 2004).

Kernels for structured domains

Another advantage of SVMs or any kernel method is that through the introduction of an appropriate kernel, we may apply the SVM machinery on objects that are defined in complex structured domains, as e.g. the space of *graphs* or *sequences* (Gärtner 2003). The case of *graph kernels* is interesting since it allowed the application of classification in a principled way (through SVMs for example) with just the definition of a suitable kernel to important problems in computational chemistry since a molecule is typically represented as a graph. The introduction to string kernels had a significant impact in computational biology as well, which made it possible to apply machine learning methodology to genomic and proteomic data (Leslie and Kuang 2004).

One famous kernel is the *string kernel* (Lodhi et al. 2002) which defines an inner product over two *strings* as the number of common substrings of a given length l. In general, many of these kernels over discrete data work by counting the number of all common sub-structures up to a given size. So, the implicit feature map is a binary representation (or perhaps counts) of all substructures of a given size.

Final thoughts

SVMs and other kernel methods revolutionazed machine learning in the early 2000 and to this day remain one of the top methods of choice. Next we list their main advantages:

- strong theory basis
- needs to solve quadratic program, so no local minima, unique optimum
- accepts many types of data in with the introduction of an appropriate *kernel*
- provides *sparse* solutions (classification only depends on *support vectors*), thus controlling over-fitting
- explicit complexity control through the C parameter and kernel hyper-parameters
- not so affected by domains with high dimensionality
- excellent practical performance

Their main disadvantages are:

- training requires solving a quadratic problem which may be slow if too much data is available
- choice of best kernel is not clear, and often many experiments need to be done
- performance is often dependent on careful choice of kernel type and parameters

References

Blumer, Anselm, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. 1989. "Learnability and the Vapnik-Chervonenkis Dimension." *J. ACM* 36 (4): 929–65. http://dblp.uni-trier.de/db/journals/jacm/jacm36.html#BlumerEHW89.

Gärtner, Thomas. 2003. "A Survey of Kernels for Structured Data." *SIGKDD Explorations*.

Leslie, Christina, and Rui Kuang. 2004. "Fast String Kernels Using Inexact Matching for Protein Sequences." *Journal of Machine Learning Research* 5: 1435–55. http://www.jmlr.org/papers/volume5/leslie04a/leslie04a.pdf.

Lodhi, Huma, Craig Saunders, John Shawe-Taylor, Nello Cristianini, and Chris Watkins. 2002. "Text Classification Using String Kernels." Edited by Bernhard Schölkopf. *Journal of Machine Learning Research*, no. 2: 419–44. http://jmlr.csail.mit.edu/papers/volume2/lodhi02a/lodhi02a.pdf.

Schölkopf, Bernhard, and Alexander Johannes Smola. 2002. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond.* Adaptive Computation and Machine Learning Series. MIT Press.

Shawe-Taylor, John, and Nello Christianini. 2004. *Kernel Methods for Pattern Analysis*. Cambridge University Press.

Vapnik, Vladimir N. 2000. *The Nature of Statistical Learning Theory*. Springer.