A Tutorial on Kernel Methods for Categorization

Frank Jäkel, Bernhard Schölkopf, Felix A. Wichmann

December 29, 2007

This is a preprint that may differ from the final version of the manuscript that is published in Journal of Mathematical Psychology 51(6):343-358.

Abstract

The abilities to learn and to categorize are fundamental for cognitive systems, be it animals or machines, and therefore have attracted attention from engineers and psychologists alike. Modern machine learning methods and psychological models of categorization are remarkably similar, partly because these two fields share a common history in artificial neural networks and reinforcement learning. However, machine learning is now an independent and mature field that has moved beyond psychologically or neurally inspired algorithms towards providing foundations for a theory of learning that is rooted in statistics and functional analysis. Much of this research is potentially interesting for psychological theories of learning and categorization but also hardly accessible for psychologists. Here, we provide a tutorial introduction to a popular class of machine learning tools, called kernel methods. These methods are closely related to perceptrons, radial-basis-function neural networks and exemplar theories of categorization. Recent theoretical advances in machine learning are closely tied to the idea that the similarity of patterns can be encapsulated in a positive definite kernel. Such a positive definite kernel can define a reproducing kernel Hilbert space which allows one to use powerful tools from functional analysis for the analysis of learning algorithms. We give basic explanations of some key concepts—the so-called kernel trick, the representer theorem and regularization—which may open up the possibility that insights from machine learning can feed back into psychology.

Machine learning is occupied with inventing computer algorithms that are capable of learning. For example, a machine that is equipped with a digital camera is shown instances of handwritten digits. Imagine an application where postal codes on letters have to be recognized so that the letters can be sorted automatically. The machine is shown many instances of each digit and has to learn to classify new instances based on the experience with the old ones. The prospect of not having to program a machine explicitly but rather having a machine learn from examples has attracted engineers to study learning since the early days of artificial intelligence. In their quest for intelligent machines early research was inspired by neural mechanisms and ideas from reinforcement learning. However, for practical applications researchers in machine learning also need to take technical constraints (like scalability, robustness and speed) into account. Furthermore, a good understanding of what the algorithm does, perhaps even with performance guarantees, would be very desirable if the algorithm was to be used in practice. Usually these constraints require techniques and insights from statistics, optimization and complexity theory that make the algorithm implausible as a psychological model of learning. Nevertheless, some of the methods used in machine learning are still strikingly similar to models that are discussed in psychology. Many of the ideas about learning that can be found in the machine learning literature are certainly based on the same intuitions that psychologists have.

Kernel methods, in particular, can be linked to neural network models and exemplar theories of categorization. Psychologically speaking, a kernel can often be thought of as a measure for stimulus similarity. In a category learning task it seems natural to assume that the transfer from old to new stimuli will depend on their similarity. In fact, this idea can be found throughout machine learning and psychology. As categorization is an important cognitive ability it has received a lot of attention from machine learning and psychology. It is also in categorization models that the similarity between machine learning methods and psychological models becomes most obvious.

This paper is a tutorial on kernel methods for categorization. These methods try to tackle the same problems that human category learners face when they try to learn a new category. Hence, we think that the mathematical tools that are used in machine learning show a great potential to be also useful for psychological theorizing. Even if most of the solutions that machine learning offers turned out to be psychologically implausible, psychologists should still find it interesting to see how a related field deals with similar problems—Especially as machine learning methods are increasingly used for the analysis of neural and behavioral data. At the very least, this paper provides an introduction to these new tools for data analysis. We find, however, that some of the methods in machine learning are closely related to categorization models that have been suggested in psychology. Briefly, some kernels in machine learning are akin to a class of similarity measures considered in psychology. This class of similarity measures is based on Shepard's *universal law of generalization* and has been used extensively in exemplar models of categorization (Nosofsky, 1986; Kruschke, 1992). Kernel methods are also like exemplar models in other respects: They usually store all the exemplars they have encountered during the course of learning and they can be implemented in a neural network. In two related papers we have used some of the results presented here to clarify the relationship between similarity and generalization in categorization models and to resolve some conceptual problems with popular models of similarity (Jäkel et al., 2007a,b). These other two papers focus on the psychological aspects of kernels, whereas in this paper we concentrate more on the mathematical aspects.

There are several useful introductions to kernel methods in the machine learning literature but none of them is addressing psychologists directly—hence this paper. Most of the technical material we present is based on two recent books on kernel methods (Christianini and Shawe-Taylor, 2000; Schölkopf and Smola, 2002) and standard results in linear algebra (Strang, 1988, e.g.). We will assume that the reader has had some previous exposure to linear algebra, for example in the context of artificial neural networks or psychometrics. However, in order to make the paper accessible to a larger audience we included reminders of relevant results throughout the text.

1 Inner products

So what is a kernel? Kernels can be regarded as a non-linear generalization of inner products. We will take a little detour before explaining kernels and discuss the relationship between inner products, perceptrons and prototypes. This will set the stage on which kernels appear naturally to solve non-linear classification problems.

1.1 Perceptrons

The perceptron can be considered the most basic of all pattern recognition algorithms. It was conceived as a simple model for learning in the brain (Rosenblatt, 1958). A pattern x, in the form of n real numbers x_1 to x_n, is fed into a neuron. The inputs are weighted by the synaptic strengths w of the n connections to the neuron. There are n real numbers w_1 to w_n that represent the synaptic strength of each input. The neuron integrates all its weighted inputs by summing them up:

$$\langle w, x \rangle = \sum_{i=1}^{n} w_i x_i. \tag{1}$$

If the excitation of the neuron is greater than a threshold value θ the neuron fires. The excitation is a linear combination of the inputs. For this reason the perceptron is also referred to as a linear classifier. Mathematically speaking, the neuron calculates the standard inner product, denoted with brackets $\langle \cdot, \cdot \rangle$, of the vector x with the vector w, both of which are elements in a n-dimensional vector space. Inner products are also called dot products or scalar products in linear algebra.

A vector space with an inner product $\langle \cdot, \cdot \rangle$ is a very rich representation and has a natural measure of length and angle that conforms to intuitions about Euclidean space. The length or norm $\|\cdot\|$ of any vector w can naturally be defined with the help of the inner product as:

$$\|w\|^2 = \sum_{i=1}^{n} w_i^2 = \langle w, w \rangle . \tag{2}$$

By using Pythagoras' theorem one can find that this is in agreement with Euclidean intuitions. All the familiar properties of Euclidean space can be expressed in terms of the standard inner product. The distance $d(\cdot, \cdot)$ between two points x and y in the space can then be defined as the length of their difference vector

$$d(x, y) = \|x - y\| = \sqrt{\langle x - y, x - y \rangle}. \qquad (3)$$

This distance can be used to define a metric on the space. Moreover, the angle α between two vectors v and w can be expressed as

$$\cos\alpha = \frac{\langle v, w \rangle}{\|v\| \, \|w\|}. \qquad (4)$$

In particular, two vectors are perpendicular whenever their inner product is zero.

Geometrically speaking, the weight vector together with the threshold can be interpreted as the normal vector of a hyperplane. Checking whether the inner product is bigger than the threshold is equivalent to checking which side of the hyperplane a pattern vector x falls on. In this way a simple classification can be implemented by separating the vector space into the two parts on both sides of the hyperplane. This is illustrated in Figure 1. The figure shows a two-dimensional space and each point in the space defines a possible pattern. There are two classes of patterns (circles and crosses) and several instances of each class are shown. A vector w pointing away from the origin is depicted together with its hyperplane $\langle w, x \rangle = 0$, that is the set of all points x that are perpendicular to w. If the inner product between w and x is greater than zero the two vectors form an angle that is less than 90 degrees, hence w and x lie on the same side of the hyperplane. It is possible to shift the hyperplane along the vector w by changing the threshold parameter θ. In this example we have chosen w and θ such that the hyperplane that they define can correctly separate the circles from the crosses. In general, the learning problem for the perceptron is to find a vector w and a threshold θ that separates two classes of patterns as well as possible. It is a very common view to see learning as adapting weights in a neural network. There is a long list of learning algorithms that try to accomplish this task of which the perceptron learning algorithm is just one.

1.2 Prototypes

Take the psychologically rather than neurally motivated example of a prototype learner (Posner and Keele, 1968; Reed, 1972). The learning machine is given a set of patterns A that are known to belong to one class and a set of patterns B that are known to belong to another class. The prototype learner is usually understood as trying to extract the central tendency of the two classes. Hence, to separate A from B the arithmetic means of all examples in A and B are calculated: $\overline{a} = \frac{1}{|A|} \sum_{a \epsilon A} a$ and $\overline{b} = \frac{1}{|B|} \sum_{b \epsilon B} a$. A new pattern x is classified as belonging to class A if it is closer to $\overline{a}$ (the mean of A) than to $\overline{b}$ (the mean of B). We can take "closer" to mean Euclidean distance in the vector space in which the patterns are given. The Euclidean distance between two points x and y is given by the square root of the inner product of the difference vector with

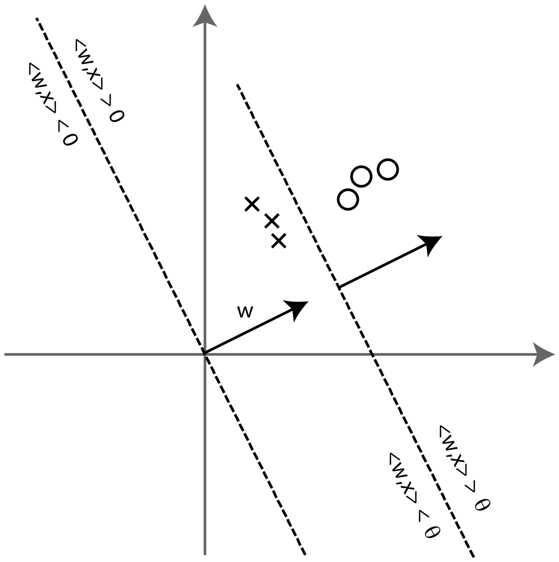

Figure 1: Each perceptron defines a hyperplane in a vector space. The weight vector w is the normal vector of this hyperplane and the threshold θ defines the offset from the origin. On one side of the hyperplane (the side that w points to) the inner product of all points with w is greater than θ. On the other side the inner product is smaller than θ. In this example we have chosen w and θ so they can separate the circles from the crosses.

itself: $\langle x-y, x-y\rangle$ (Eq. 3). Therefore, a Euclidean prototype classifier decides that a new stimulus x belongs to class A whenever

$$\begin{aligned}
\left\langle x-\bar{b}, x-\bar{b}\right\rangle &> \langle x-\bar{a}, x-\bar{a}\rangle \\
\langle x, x\rangle - 2\left\langle \bar{b}, x\right\rangle + \left\langle \bar{b}, \bar{b}\right\rangle &> \langle x, x\rangle - 2\langle \bar{a}, x\rangle + \langle \bar{a}, \bar{a}\rangle \\
2\langle \bar{a}, x\rangle - 2\left\langle \bar{b}, x\right\rangle &> \langle \bar{a}, \bar{a}\rangle - \left\langle \bar{b}, \bar{b}\right\rangle \\
\left\langle \bar{a}-\bar{b}, x\right\rangle &> \frac{1}{2}\left(\langle \bar{a}, \bar{a}\rangle - \left\langle \bar{b}, \bar{b}\right\rangle\right) \\
\left\langle \bar{a}-\bar{b}, x\right\rangle &> \theta. \qquad (5)
\end{aligned}$$

Remember that the definition of the inner product $\langle\cdot,\cdot\rangle$ involves a sum and therefore it is linear in both arguments: For all x, y and z it holds that $\langle x, y+z\rangle = \langle x, y\rangle + \langle x, z\rangle$ and also that $\langle x+y, z\rangle = \langle x, z\rangle + \langle y, z\rangle$. We have used this property extensively in the above derivation. From the last line of (5) it can be seen that the prototype classifier defines a hyperplane in the input space just as the perceptron in Eq. (1) does. The weight vector w is given by the difference of the means $\bar{a}-\bar{b}$ and the threshold θ by the right hand side $\frac{1}{2}\left(\langle \bar{a}, \bar{a}\rangle - \left\langle \bar{b}, \bar{b}\right\rangle\right)$. However, θ is really just a bias parameter that determines which of the two category responses is preferred and might be chosen differently. The crucial fact is that for the prototype classifier we take an inner product with the difference vector of the means.

1.3 Positive definite matrices

The inner product defined in Eq. (1) is called the standard inner product because it naturally arises in the context of Euclidean spaces. In general, an inner product $\langle\cdot,\cdot\rangle$ has to fulfill three formal properties that ensure that the norm, distance and angle will behave as in Euclidean space. First, it has to be symmetric: For all real-valued vectors w and v it holds that $\langle w, v\rangle = \langle v, w\rangle$. This reflects the fact that the (absolute) angle between two vectors does not depend on whether it is measured from w to v or from v to w. Second, an inner product has to be linear in its arguments, that is for a real number a and three vectors u, v and w it holds that $\langle au, v\rangle = a\langle u, v\rangle$ and $\langle u+w, v\rangle = \langle u, v\rangle + \langle w, v\rangle$. Because of the symmetry an inner product is linear in both arguments. Third, an inner product has to be positive definite. By positive it is meant that $\langle w, w\rangle \geq 0$ for all w. Definiteness refers to $\langle w, w\rangle = 0$ if and only if $w = 0$. Positive definiteness is a natural requirement for a length measure. Remember that the inner product of a vector with itself $\langle w, w\rangle$ defines the square of the length of the vector $\|w\|^2$ and the squared length always has to be positive and is only zero for the zero vector. It is easily verified that the standard inner product (1) fulfills all three axioms.

The standard inner product is by no means the only interesting inner product. A generalization that will be very important in the following is given by

$$\langle w, v\rangle_K = \sum_{i=1}^{n}\sum_{j=1}^{n} w_i v_j k_{ij}. \qquad (6)$$

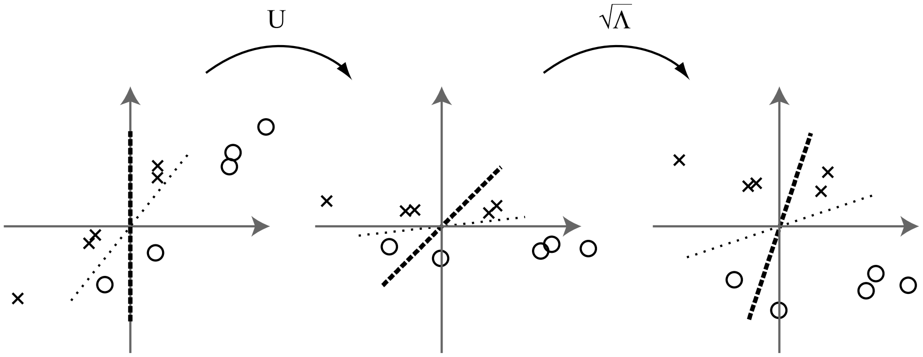

Figure 2: Whether a prototype classifier can separate two classes depends also on the inner product that is chosen. The left panel shows two classes (circles and crosses) with highly correlated dimensions—in this case the standard inner product is not appropriate. The thick dashed line depicts the decision bound for a prototype classifier when the standard inner product is used. The dotted line depicts a decision bound with a different inner product. This inner product corresponds to the standard inner product in the space depicted on the right that can be obtained by rotating and rescaling the original space.

Taking K to be a matrix with entries k_{ij} and ^{T} to denote the transpose of a matrix (rows and columns exchanged) the definition can be written more elegantly as a matrix multiplication

$$\langle w, v\rangle_K = w^T K v. \tag{7}$$

If K is the identity matrix the standard inner product (1) is recovered. In order for this definition to result in an inner product the three axioms have to be fulfilled. Symmetry depends on the symmetry of K. If $k_{ij} = k_{ji}$ for all i and j then the definition in (6) will be symmetric. As a consequence of the linearity of the sum it is immediately clear that (6) is always linear. It remains to demand positive definiteness. The inner product defined in Eq. (6) is positive definite if the matrix K is positive definite, that is for all vectors w the quadratic form that defines the squared length of the vector $\|w\|^2$ is positive,

$$w^T K w \geq 0, \tag{8}$$

and zero if and only if w is zero. It is a standard result in linear algebra that symmetric positive definite matrices can be decomposed into principal components. Principal component analysis (PCA) is used frequently in the analysis of psychological data, for example covariance matrices are positive definite and they are often subjected to PCA. There is a rotation matrix U and a diagonal matrix Λ that contains only positive eigenvalues such that $K = U^T \Lambda U$. With this result the inner product (7) can be rewritten as

$$\langle w, v\rangle_K \quad = \quad w^T K v$$

$$
\begin{aligned}
&= w^T \left(U^T \Lambda U\right) v \\
&= w^T \left(\sqrt{\Lambda} U\right)^T \left(\sqrt{\Lambda} U\right) v \\
&= \left(\sqrt{\Lambda} U w\right)^T \left(\sqrt{\Lambda} U v\right) \\
&= \left\langle \sqrt{\Lambda} U w, \sqrt{\Lambda} U v \right\rangle .
\end{aligned}
$$

Remember that for any two matrices A and B that can be multiplied $(AB)^T = B^T A^T$. By using U the vectors v and w are rotated such that they coincide with the principal components. After that they are rescaled using the diagonal matrix $\sqrt{\Lambda}$. In this coordinate system the inner product $\langle w, v \rangle_K$ amounts to a standard inner product. In order for K to implement an inner product all eigenvalues have to be positive. Otherwise there could be vectors with a squared length smaller than zero—clearly in contradiction with Euclidean intuitions. Note also that if some eigenvalues were zero then there would be vectors, other than the zero vector, with a length zero. All this illustrates the close connections between the standard inner product, Euclidean space and positive definite matrices. Positive definite matrices can define an inner product. If the coordinate axes are rotated and rescaled appropriately this inner product becomes a standard inner product and therefore can induce a norm, a metric and angles that behave like the familiar Euclidean ones.

1.4 Prototypes and orthogonality

As an example consider the following classification problem. The left panel of Figure 2 shows two categories drawn from two Gaussian distributions. Each point is a stimulus that is described by two dimensions. The dimensions are highly correlated for both stimulus classes. On a first glance a prototype learner will not find a good decision bound to separate the two classes. The decision bound that results from the standard prototype classifier is depicted as the thick dashed line. As the decision bound is orthogonal to the shortest connection between the two means it cannot pay due respect to the correlations in the classes. On a first glance one could think that the problem is that the prototype classifier cannot deal with correlation and therefore such a problem cannot be solved by a prototype classifier. However, the problem does not actually lie in the prototype classifier as such, it lies in the inner product that is used to define orthogonality. A prototype learner that does not fail for even the simplest category structures should take the covariance of the stimulus dimensions into account (Reed, 1972; Fried and Holyoak, 1984; Ashby and Gott, 1988). If we take the inner product $\langle \cdot, \cdot \rangle_K$ to be given by a positive definite matrix K that is the inverse of the covariance matrix of the classes we get the *right* definition of orthogonality. The resulting decision bound is depicted as a dotted line. K is the inverse of a positive definite matrix (covariance matrices are always positive

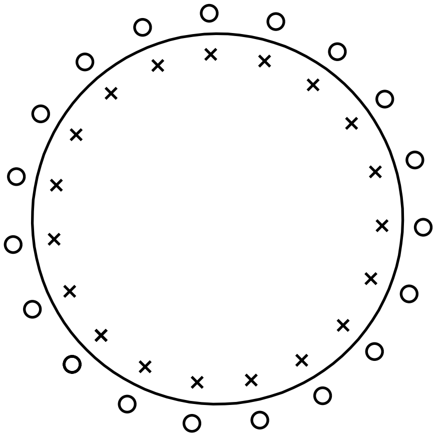

Figure 3: The crosses and the circles cannot be separated by a linear perceptron in the plane.

definite) and is therefore also a positive definite matrix. Hence, it corresponds to the standard inner product after rotating and scaling the space with the matrices U and $\sqrt{\Lambda}$. The middle panel in Figure 2 shows the rotated space and the right panel shows the space after scaling.

1.5 Non-linear classification problems

A linear classifier like the perceptron is a very attractive method for classification because it builds on strong geometric intuitions and the extremely well-developed mathematics of linear algebra. However, there are problems that a linear classifier cannot solve—at least not directly. As several psychological theories of categorization are based on linear classifiers this issue has also attracted some attention in the psychological literature (Medin and Schwanenflugel, 1981; Smith et al., 1997). One example of a problem that cannot be solved with a linear classifier can be seen in Figure 3. For a long time, the most popular approach to solve non-linear problems like this one was to use a multi-layer perceptron. Multi-layer perceptrons are known to be able to approximate any function (Hornik et al., 1989) and can be trained efficiently by using the back-propagation algorithm (Rumelhart et al., 1986). The approach that will be presented here is fundamentally different. The strategy is to use a non-linear function to map the input patterns into a space where the problem can be solved by a linear classifier. The following toy-example illustrates this approach.

Figure 3 shows examples from two classes (crosses and circles) that cannot be separated by a hyperplane in the input space (i.e. a straight line in two dimensions). Instead of trying to classify the examples in the input space that is given by the values x_1 and x_2 the data are transformed in a non-linear way. Linear classification of the data is then attempted in the transformed space.

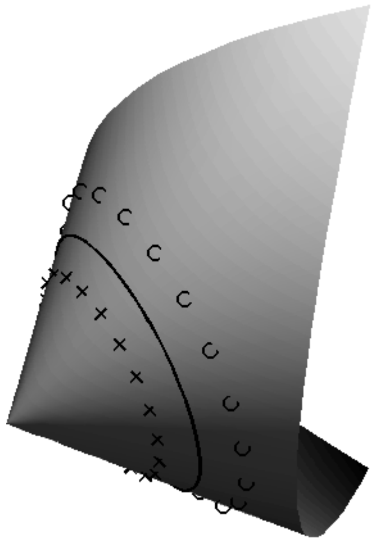

Figure 4: The crosses and circles from Figure 3 can be mapped to a three-dimensional space in which they can be separated by a linear perceptron.

In machine learning such a non-linear transform is called a feature map and the resulting space a feature space. The term "feature" is already heavily overloaded in psychology. Therefore we will use the more neutral terms linearization function and linearization space instead. The term linearization space was used in an early paper on kernel methods (Aizerman et al., 1964). For example, consider the following linearization function $\Phi : \mathbb{R}^2 \mapsto \mathbb{R}^3$

$$\Phi(x) = \begin{pmatrix} \phi_1(x) \\ \phi_2(x) \\ \phi_3(x) \end{pmatrix} = \begin{pmatrix} x_1^2 \\ \sqrt{2}x_1 x_2 \\ x_2^2 \end{pmatrix}. \tag{9}$$

This transformation maps the example patterns to a three-dimensional space that is depicted in Figure 4. The examples live on a two-dimensional manifold of this three-dimensional space. In this space the two classes become linearly separable, that is it is possible to find a two-dimensional plane such that the circles fall on one side and the crosses on the other side. This shows that with an appropriate non-linear transformation of the input a simple linear classifier can solve the problem. The linearization approach is akin to transforming the data in data-analysis before fitting a linear model. In the current example each hyperplane in the linearization space defines a quadratic equation in the input space. Hence, it is possible to deal with quadratic (i.e. non-linear) functions by only using linear methods. In general, the strategy is to *preprocess* the data with the help of a function Φ such that a linear perceptron model is likely to be applicable. Formally this can be expressed as

$$\langle w, \Phi(x) \rangle = \sum_{i=1}^{n} w_i \phi_i(x), \tag{10}$$

where n is now the dimension of the linearization space and w is a weight vector in the linearization space. It is clear that there is a wide variety of non-linear functions that can be used to preprocess the input. In fact, this approach was very popular in the early days of machine learning (Nilsson, 1965). The problem is of course that the function Φ has to be chosen before learning can proceed. In our toy example we have only shown how one can use linear methods to deal with quadratic functions but usually one will not know in advance whether it is possible to separate the data with a quadratic function. However, if Φ is chosen to be sufficiently flexible, for example instead of a quadratic function with only three coefficients one could choose a high order polynomial with many coefficients, then it may be possible to approximate even very complicated decision functions. This comes at the cost of increasing the dimensionality of the linearization space and the number of free parameters. Therefore early machine learning research has tried to avoid this. After Minsky and Papert (1967) could show that some interesting predicates cannot be separated by any finite dimensional perceptron, not even by choosing a polynomial of a very high degree, the promises of flexible, linear perceptrons seemed rather dim for many practical applications. Only later did researchers in machine learning discover that by using kernels they could easily build infinite dimensional perceptrons that seem powerful enough for all practical purposes.

2 Kernels

The next section will introduce the *kernel trick* that makes it possible to work with high dimensional (even infinite dimensional) and flexible linearization spaces.

2.1 The kernel trick

There is an interesting observation about the linearization function that was used in the foregoing example. The standard inner product between two input vectors in the linearization space can be calculated without having to explicitly map the data into the linearization space. For two points x and y in $\mathbb{R}^2$

$$\langle \Phi(x), \Phi(y) \rangle = x_1^2 y_1^2 + 2x_1x_2y_1y_2 + x_2^2 y_2^2 = \langle x, y \rangle^2 .$$

A more general result can be proved. For an n dimensional input space a class of popular and flexible linearization functions is given by all monomials of degree d. A monomial of degree d takes the product of d components of an input vector x. E.g., for $n = 5$ the following are monomials of degree $d = 3$: x_1^3, $x_1x_2x_5$ and $x_2^2x_4$. The possible number of monomials is given by choosing d out of n with replacement. The order does not matter because of the commutativity of the product. However, for simplicity let us consider a linearization function that takes all n^d possible ordered monomials. Thus, $x_1x_2x_3$ is a dimension in the new space but $x_2x_3x_1$ would be another dimension. For the linearization function $\Phi' : \mathbb{R}^n \mapsto \mathbb{R}^{n^d}$ that computes all ordered monomials it holds that

$$\begin{aligned}
\langle \Phi'(x), \Phi'(y) \rangle &= \sum_{i_1=1}^{n} \sum_{i_2=1}^{n} ... \sum_{i_d=1}^{n} x_{i_1} x_{i_2} ... x_{i_d} y_{i_1} y_{i_2} ... y_{i_d} \\
&= \sum_{i_1=1}^{n} x_{i_1} y_{i_1} \sum_{i_2=1}^{n} x_{i_2} y_{i_2} ... \sum_{i_d=1}^{n} x_{i_d} y_{i_d} \\
&= \left(\sum_{i=1}^{n} x_i y_i \right)^d = \langle x, y \rangle^d .
\end{aligned}$$

Calculating the inner product in the linearization space is the same as taking the inner product in the original space and taking it to the power of d. Computationally, this is an extremely attractive result. Remember that a high number of dimensions is needed to make the linearization space sufficiently flexible to be useful. If calculated naively the computational effort of the inner product in the linearization space scales with its dimensions. However, this result shows that, in the case of a monomial linearization function, it is not necessary to explicitly map the vectors x and y to the n^d dimensional linearization space to calculate the dot product of the two vectors in this space. It is enough to calculate the standard inner product in input space and take it to the power of d.

The function $k(x, y) := \langle \Phi(x), \Phi(y) \rangle = \langle x, y \rangle^d$ is our first example of a kernel, the so-called polynomial kernel. Intuitively, kernels can provide a way to efficiently calculate inner products in higher dimensional linearization spaces. They also provide a convenient non-linear generalization of inner products. With the help of a kernel, it is easy to build non-linear variants of simple linear algorithms that are based on inner products. This is called the *kernel trick* in the machine learning literature.

Take as an example the prototype classifier, again. Instead of taking the mean in input space, like in Eq. (5), one can construct a prototype classifier in the linearization space. We will take the threshold θ to be a free parameter that we can tune to account for biases. For the left hand side we now want to take the mean in the linearization space, that is the mean after we applied the mapping Φ. To decide whether x belongs to class A we also map x to the linearization space and check that

$$\begin{aligned}
\left\langle \frac{1}{|A|} \sum_{a \epsilon A} \Phi(a) - \frac{1}{|B|} \sum_{b \epsilon B} \Phi(b), \Phi(x) \right\rangle &> \theta \\
\frac{1}{|A|} \sum_{a \epsilon A} \langle \Phi(a), \Phi(x) \rangle - \frac{1}{|B|} \sum_{b \epsilon B} \langle \Phi(b), \Phi(x) \rangle &> \theta \\
\frac{1}{|A|} \sum_{a \epsilon A} k(a, x) - \frac{1}{|B|} \sum_{b \epsilon B} k(b, x) &> \theta, \qquad (11)
\end{aligned}$$

where as before A and B are sets of patterns from two classes and the linearity of the inner product and the sum were used.

The input space could be a 16×16 matrix of pixel values, that is a 256 dimensional space. The linearization space could be all monomials of degree 10. Mapping the inputs to the linearization space and calculating the mean there would be prohibitive as the mapping of each input to this space takes a large number of dimensions. Despite the high number of dimensions it is possible to use a prototype classifier in the linearization space by taking advantage of the kernel trick. By using the kernel trick it is not necessary to calculate the mean in the linearization space but a prototype classifier can still be used by using Eq. (11).

In psychology, monomial linearization functions have been used before, for example, in the configural-cue model by Gluck and Bower (1988). They used a linear perceptron with monomial features for modeling the categorization data of Shepard et al. (1961). They also noted that increasing the number of input dimensions with monomials increases the capabilities of a perceptron similar to increasing the number of hidden units in a multi-layer perceptron—however, in an inelegant way because the number of monomials increases drastically with the number of features. Therefore, the number of weights that have to be adapted also increases drastically. The above result shows that it is possible to calculate the response of a perceptron, for example when the weights are adapted by using the prototype rule (but also using other learning rules), without actually having to compute all monomials. In fact, neither the monomials nor the weights in the linearization space have to be computed to evaluate the prototype-perceptron in Eq. (11).

2.2 Reproducing kernel Hilbert space

Every linearization function Φ defines a kernel function via

$$k(x, y) = \langle \Phi(x), \Phi(y) \rangle \, . \tag{12}$$

It is always possible to define a kernel by choosing a linearization function Φ and an inner product. The function $k(\cdot, \cdot)$ can be evaluated by explicitly mapping patterns to the linearization space and calculating the inner product in the linearization space. However, as the example of the polynomial kernel has shown, sometimes it is not necessary to actually compute Φ. It is natural to ask under what circumstances does a function $k(\cdot, \cdot)$ implement an inner product in a linearization space and what does the corresponding linearization space and linearization function look like. As it turns out there is a well-developed branch of mathematics that deals with these questions: Functional analysis. In short the answer is that if $k(\cdot, \cdot)$ is a symmetric and positive definite kernel then k implements an inner product in a linearization space. Constructing a linearization space and an inner product for a positive definite kernel is the purpose of this section.

First, the introduction of some notation is required. For a set of patterns x_1 to x_N and a function $k(\cdot, \cdot)$ of two arguments the kernel matrix is the matrix that collects all pairwise applications of k to the patterns. Let us denote this $N \times N$ matrix with K and denote the entry in the i^{th} row and j^{th} column with k_{ij} then

$$K \text{ with } k_{ij} = k(x_i, x_j)$$

is called the kernel matrix or Gram matrix for the patterns $x_1, ..., x_N$. A real and symmetric function $k(\cdot, \cdot)$, that is a function with the property $k(x, y) = k(y, x)$, is called a *positive definite* kernel if for all choices of N points the corresponding kernel matrix K is *positive semi-definite*, that is for all N-dimensional vectors w

$$w^T K w \geq 0. \tag{13}$$

Note that for a matrix to be positive semi-definite we do not require that equality only holds for $w = 0$ (as opposed to the definition of a positive definite matrix, see Eq. 8). As K is only positive semi-definite it can have eigenvalues that are zero and does not have to be full rank. This definition of a positive definite kernel seems confusing because for a kernel to be positive definite we require the corresponding kernel matrices to be positive semi-definite. However, the definition we give is the usual definition used in machine learning and therefore we will use it, too (Schölkopf and Smola, 2002).

With the definition of a positive definite kernel in mind, it is possible to construct a vector space, an inner product, and a linearization function such that the kernel condition (12) is fulfilled. In the following, these three steps are demonstrated in a purely formal way. After that, the formal steps are illustrated by an example, using the Gaussian kernel.

2.2.1 Step 1: Constructing a vector space

The vector space will be a space of functions constructed from the kernel. Let $k(\cdot, x)$ denote a function that is taken to be a function of its first argument with a fixed second argument. The vector space is then defined as all functions of the form

$$f(x) = \sum_{i=1}^{N} w_i k(x, x_i). \tag{14}$$

Each function in the space is a linear combination of kernel functions $k(\cdot, x_i)$ and can be expressed by some set of N patterns $x_1, ..., x_N$ with real coefficients $w_1, ..., w_N$. It is important to realize that these N patterns could be different for different functions. All functions are linear combinations of kernel functions given by k and because they are linear combinations they define a vector space—functions can be added and multiplied with scalars. When functions are added potentially all the kernel functions of the two added functions need to be included in the expansion of the summed function but the sum will still be in the vector space.

The expansion of f given in Eq. (14) might not be unique. There is no requirement in the definition of f that the kernel functions need to be linearly independent. If they are not independent then the same function can be expressed in different ways. The function space is the span of the generating system of functions. If there is an infinite number of potential independent kernel functions then the vector space is infinite dimensional, even though each function f can be expressed by a finite sum.

2.2.2 Step 2: Constructing an inner product

Next we will equip this vector space with an inner product. A possibly infinite dimensional vector space with an inner product is called a *pre-Hilbert space*. If the limit points of all Cauchy sequences are included in the space, the space is completed and turned into *Hilbert space* proper. Completeness is, for example, important for defining unique projections. We will ignore these technicalities here (Schölkopf and Smola, 2002, but see) and simply note that Hilbert spaces can be thought of as the possibly infinite dimensional generalization of Euclidean spaces. Take a function f with an expansion given by Eq. (14) and let $g(x) = \sum_{i=1}^{M} v_i k(x, y_i)$ be another function from this space then we can define the inner product between the two functions f and g as

$$\langle f, g\rangle_{\mathcal{H}} = \sum_{i=1}^{N}\sum_{j=1}^{M} w_i v_j k(x_i, y_j). \tag{15}$$

In order to distinguish the inner product in Hilbert space from the normal inner product in Euclidean space we have added the little index $\mathcal{H}$. We have to show that this definition is indeed an inner product. First we have to show that it is well-defined. The particular expansions of f and g that are used in the definition might not be unique, as mentioned above. Fortunately, the definition (15) does not depend on the particular expansions of f and g that are used to calculate the inner product. To see this, let $f(x) = \sum_{i=1}^{N'} w'_i k(x, x'_i)$ and $g(x) = \sum_{i=1}^{M'} v'_i k(x, y'_i)$ be two new expansions of f and g that are different from the ones used in the definition of the inner product (15). They will, however, result in the same inner product because

$$\begin{aligned}\sum_{i=1}^{N}\sum_{j=1}^{M} w_i v_j k(x_i, y_j) &= \sum_{i=1}^{N} w_i g(x_i) \\ &= \sum_{i=1}^{N}\sum_{j=1}^{M'} w_i v'_j k(x_i, y'_j) \\ &= \sum_{j=1}^{M'} v'_j f(y'_j) \\ &= \sum_{i=1}^{N'}\sum_{j=1}^{M'} w'_i v'_j k(x'_i, y'_j).\end{aligned} \tag{16}$$

Therefore, (15) is indeed well-defined. To show that it is an inner product it also has to be symmetric, linear in its arguments and positive definite. As k is symmetric in both arguments the above definition is also symmetric. It is obviously linear because of the linearity of the sum. Positive definiteness means that $\langle f, f\rangle_{\mathcal{H}} \geq 0$ where equality only holds for $f = 0$. Note that $\langle f, f\rangle_{\mathcal{H}} = w^T K w$ by definition. As the defining property of a positive definite kernel is that the kernel matrix K is always positive semi-definite (Eq. 13), it is immediately clear that $\langle f, f\rangle_{\mathcal{H}} \geq 0$. Definiteness is a bit more tricky but it can be proved that for all positive definite kernels definiteness of (15) holds (Schölkopf and Smola, 2002). Hence, all positive definite kernels can define an inner product in the above way. This may also justify calling these kernels positive definite.

2.2.3 Step 3: Constructing a linearization function

Each kernel function $k(\cdot, x)$ with a fixed x is trivially contained in the vector space. It is simply an expansion with only one kernel function and a weight of one. Therefore, the inner product (15) of this function with a function f that has N coefficients w_i and kernel functions $k(\cdot, x_i)$ is

$$\langle k(\cdot, x), f\rangle_{\mathcal{H}} = \sum_{i=1}^{N} w_i k(x, x_i) = f(x), \tag{17}$$

by the definition of the function space (Eq. 14). This is a remarkable fact: The inner product with the function $k(\cdot, x)$ evaluates the function f at point x. Therefore $k(\cdot, x)$ is also called the representer of evaluation. Another remarkable property directly follows from the definition of the inner product (15)

$$\langle k(\cdot, x), k(\cdot, y)\rangle_{\mathcal{H}} = k(x, y) \tag{18}$$

because each of the two kernel functions has a simple expansion with just one summand and a coefficient of one. Due to these two properties the linear space of functions as given in Eq. (14) with the above dot product $\langle \cdot, \cdot\rangle_{\mathcal{H}}$ is called a *reproducing kernel Hilbert space* (RKHS) in functional analysis (if it is completed).

Now, a linearization function can be defined in the following way $\Phi(x) := k(\cdot, x)$. Because of the reproducing property the kernel condition $k(x, y) = \langle \Phi(x), \Phi(y)\rangle_{\mathcal{H}}$ holds for this linearization function. The linearization space is a space of functions over the x. The linearization function that was constructed maps each point x in the input space to a function $k(\cdot, x)$ in the linearization space.

Remember what is accomplished by this. Starting from a positive definite kernel a vector space, an inner product and a linearization function were constructed such that the kernel condition (12) holds. If the kernel is easy to calculate then by means of the kernel it is possible to calculate inner products in the linearization space without actually mapping the points x and y into it. Understanding this trick opens up a box of new non-linear tools for data analysis. Any method where the linearization space only occurs in inner products of the form (12) can benefit. In fact, there is now a long list of familiar linear methods that have been kernelized. This list includes kernel principal component analysis (Schölkopf et al., 1998) and many others (Schölkopf and Smola, 2002).

2.3 Gaussian kernel example

The Gaussian kernel has frequently been used in psychology to model the similarity between two mental representations x and y (Nosofsky, 1986, 1990; Ashby and Maddox, 1993). It is defined as

$$k(x, y) = \exp^{-\|x-y\|^2} . \tag{19}$$

A standard result in functional analysis is that the Gaussian kernel is a positive definite function and that any kernel matrix K resulting from the Gaussian kernel is always full rank (Schoenberg, 1938; Schölkopf and Smola, 2002). We will not prove these two facts but we will take them for granted in what follows. The fact that the kernel matrices are always full rank and therefore positive definite (and not just semi-definite) is important and should be kept in mind.

2.3.1 Step 1: Constructing a vector space

As a vector space we take all functions that can be expressed as a linear combination of Gaussian kernel functions (see Eq. 14). One example of such a function is shown in Figure 5. The Gaussian functions are depicted with dotted lines. Their height is scaled with the weight w_i that each function receives in the sum. Summing the Gaussian functions results in the solid function. It is easy to imagine that by changing the weights and adding more Gaussian functions very different functions can be implemented or at least approximated. While each function is a finite sum, the space includes all functions that can be expressed in this way with infinitely many different choices for Gaussian functions. In fact, there are uncountably many choices because each point on the axis is a potential candidate for a Gaussian functions centered on this point. For this reason this vector space does not have a finite dimensional basis. It is not possible to describe all functions that are spanned by the Gaussian functions with a finite number of basis functions. We have an example of an infinite dimensional space—that however in many respects is similar to the ordinary finite dimensional vector spaces that are the subject of linear algebra. For example, this infinite dimensional space can also be equipped with an inner product.

2.3.2 Step 2: Constructing an inner product

Eq. (15) defines an inner product that can be used. With the inner product on the function space it is possible to define a norm and a metric on the space in the same way as it is done in Euclidean spaces. One can even calculate angles between functions. Let us, for illustration, calculate the angle between two Gaussian functions. This is done in the same way as in Euclidean spaces (see

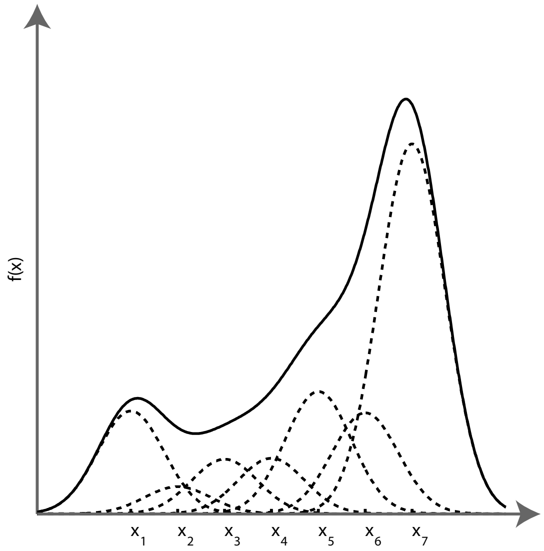

Figure 5: An example of a function from the linear function space defined in Eq. (14). Each function in the space is a linear combination of generating functions that are given by the kernel: $f(x) = \sum_i^N w_i k(x, x_i)$. Here we have depicted seven Gaussian basis functions as dotted lines. Their height is proportional to their weight w_i. The sum of these is shown with the solid line. All functions that can be expressed as such a linear combination of kernel functions are in the function space.

Eq. 4). First note that each Gaussian function has a trivial expansion in the function space. It is simply itself with a weight of one. The norm of a Gaussian function is one because

$$\|k(\cdot, x)\|^2 = \langle k(\cdot, x), k(\cdot, x)\rangle = k(x, x) = \exp^{-\|x-x\|^2} = 1 \tag{20}$$

which is only using the definition of the inner product (15) and the Gaussian kernel. Therefore, the cosine of the angle between two Gaussian functions is directly given by the inner product. As two Gaussian functions each have an expansion with only one weight that is set to one their inner product is $\langle k(\cdot, x), k(\cdot, x)\rangle = k(x, y)$, which is of course the reproducing property (18). Hence, $k(x, y)$ can be interpreted as the cosine of the angle between two Gaussian functions centered on x and y, respectively. This has interesting consequences. For two non-identical points x and y it holds that $1 > k(x, y) > 0$. Therefore, the angle between two Gaussian functions lies between 0 and 90 degrees. The further two Gaussians are apart the greater is their angle. Functions that are far apart are almost orthogonal. This makes sense because they span different parts of the function space. But as no two Gaussians are completely orthogonal it also means that the Gaussian functions do certainly not form an orthogonal basis of the function space. We have noted before that for the Gaussian kernel the kernel matrices (that collect all pairwise inner products of the Gaussians) are always full rank, hence any number of Gaussian functions are always linearly independent and therefore form a basis for the subspace that they span.

2.3.3 Step 3: Constructing a linearization function

The linearization function Φ maps points from the space in which x is defined to a space of functions over all possible x. In the example of the Gaussian kernel this means that we map a point x to the function $k(\cdot, x)$, a Gaussian function centered on x. In a psychological setting imagine x to be the representation of a stimulus in some perceptual space. Imagine further that k is interpreted as a similarity measure. The further two stimuli are apart the less similar they are and this relationship is captured in the Gaussian kernel. Mapping a stimulus x to the function $k(\cdot, x)$ means replacing a stimulus with its similarity to all other stimuli. Representation in this RKHS is literally representation of similarities (Edelman, 1998). In a companion paper we discuss some consequences of this observation in more detail (Jäkel et al., 2007b). Here, it suffices to say that calculating similarity by a Gaussian function is the same as taking an inner product in the corresponding RKHS that was constructed above.

2.4 Prototypes and exemplars

Let us stay with the example of the Gaussian kernel (19) for a while. Keep in mind that in a psychological setting the Gaussian kernel $k(x, y)$ is interpreted as the similarity between two stimuli x and y. To see the potential of the RKHS view of the Gaussian kernel for psychological theorizing, imagine we construct a prototype classifier in RKHS according to inequality (11). Let us assume that the bias parameter θ is set to zero. In this case, for a prototype classifier in the linearization space we decide that x belongs to class A and not to class B if $\Phi(x)$ is closer to the mean of the stimuli in A than in B. This can be done by checking that

$$\frac{1}{|A|}\sum_{a \epsilon A} k(a,x) > \frac{1}{|B|}\sum_{b \epsilon B} k(b,x), \tag{21}$$

that is the mean similarity of x to all exemplars of class A is bigger than the mean similarity to all exemplars of class B. The left side of the inequality (if appropriately normalized) can be interpreted as a kernel-density estimate for the class density of A (Ashby and Alfonso-Reese, 1995). It can also be interpreted as an estimate for the degree that a new x belongs to A. In any case, this is the most basic exemplar model of categorization, but it was derived from a prototype classifier.

2.5 Infinite dimensional perceptrons

Remember that the reason why we introduced the kernel trick was that a flexible preprocessing is needed so that a linear classifier can solve many non-linear categorization problems. In the previous section we examined the prototype classifier in an infinite dimensional function space, in this section we examine linear perceptrons in such a space. It will turn out that a linear classifier in an infinite-dimensional space can separate all possible stimuli in two classes.

Formally, a perceptron with a preprocessor Φ is given by an inner product of the pattern x mapped to a linearization space and a vector w in the same space: $\langle w, \Phi(x)\rangle$. Let us choose the infinite dimensional linearization space to be the RKHS that is associated with a suitable kernel, for example the Gaussian kernel. The perceptron in this RKHS is then defined by the inner product between two functions. The pattern x is preprocessed by the function $\Phi(x) := k(\cdot, x)$ which maps x to the function that describes its similarity to all other stimuli. The role of the weight vector w in the classical perceptron (10) is taken by the coefficients for a function in the RKHS. Let us denote this function with f. As before—see equation (17)—we denote the coefficients that f takes in the expansion given by the kernel functions with w. With this notation an infinite dimensional perceptron can be written as:

$$\langle f, \Phi(x)\rangle_{\mathcal{H}} = \langle f, k(\cdot, x)\rangle_{\mathcal{H}} = \sum_{i=1}^{N} w_i k(x, x_i) = f(x). \tag{22}$$

As $k(\cdot, x)$ is the representer of evaluation (17), the inner product of f with the pattern x mapped to the linearization space is just f evaluated at x. If $f(x)$ is greater than a threshold the pattern x is categorized as one class otherwise as the other class.

The learning problem for an infinite-dimensional perceptron is then to find a suitable function f in the RKHS that can categorize the patterns under consideration correctly. It may seem that this is a difficult problem because we have to find a function in an infinite dimensional space rather than a weight vector in a finite dimensional space as for the perceptron. However, there is a simple solution for f. Say, a subject wants to learn to discriminate between two different categories of stimuli. The subject is given a training set of N exemplars x_1, ..., x_N. Each stimulus has a class label that we denote with y_1, ..., y_N. The class label y_i for pattern x_i can be either $+1$ or -1, depending on which category x_i belongs to. We will treat the categorization problem like a regression problem. The aim of the category learner is to find a function f defined on the perceptual space such that $f(x)$ is $+1$ whenever x belongs to one class and -1 when x is in the other class. Instead of searching for the best function in the whole RKHS we will only consider a subspace of all functions in the RKHS and show that in this subspace there is a function that can solve the regression problem perfectly. The subspace we consider is all linear combinations of the kernel functions on the exemplars:

$$f(x) = \sum_{i=1}^{N} w_i k(x, x_i). \tag{23}$$

The output of this function can be thought of as calculating a weighted similarity to all exemplars. This function is a linear combination of kernel functions. As all kernel functions are in the RKHS their linear combinations are also in the RKHS. Therefore, for a fixed set of N exemplars their linear combination spans a subspace of the RKHS that is at most N-dimensional. We refer to this subspace as the span of the exemplars. The span of the exemplars seems to be a only a small subset of all the functions in the infinite dimensional RKHS but it contains a function that solves the regression problem perfectly.

Finding a function f of the form (23) that solves the regression problem means finding weights w_1, ..., w_N for the exemplars such that for all j: $f(x_j) = y_j$. We introduce an N-dimensional vector y for the N labels. As we are only interested in the values that the function f takes on the exemplars x_j (with j from 1 to N) we only need to evaluate (23) at these values: $\sum_{i=1}^{N} w_i k(x_j, x_i)$. Let us use a vector w for the weights and using the same notation as before we write K for the matrix that collects all pairwise evaluations of k on the exemplars. Hence, the weights that we seek should solve $y = Kw$. If K has full rank—as the Gaussian kernel for example guarantees—then K is invertible and

$$w = K^{-1} y. \tag{24}$$

There is a unique vector of weights that solves the classification problem perfectly. If K has full rank this is always possible irrespective of the set of exemplars and their category labels[1]. As there is a solution in the span of the exemplars we do not need to work with the infinite dimensional RKHS to find a solution for the categorization problem in the RKHS.

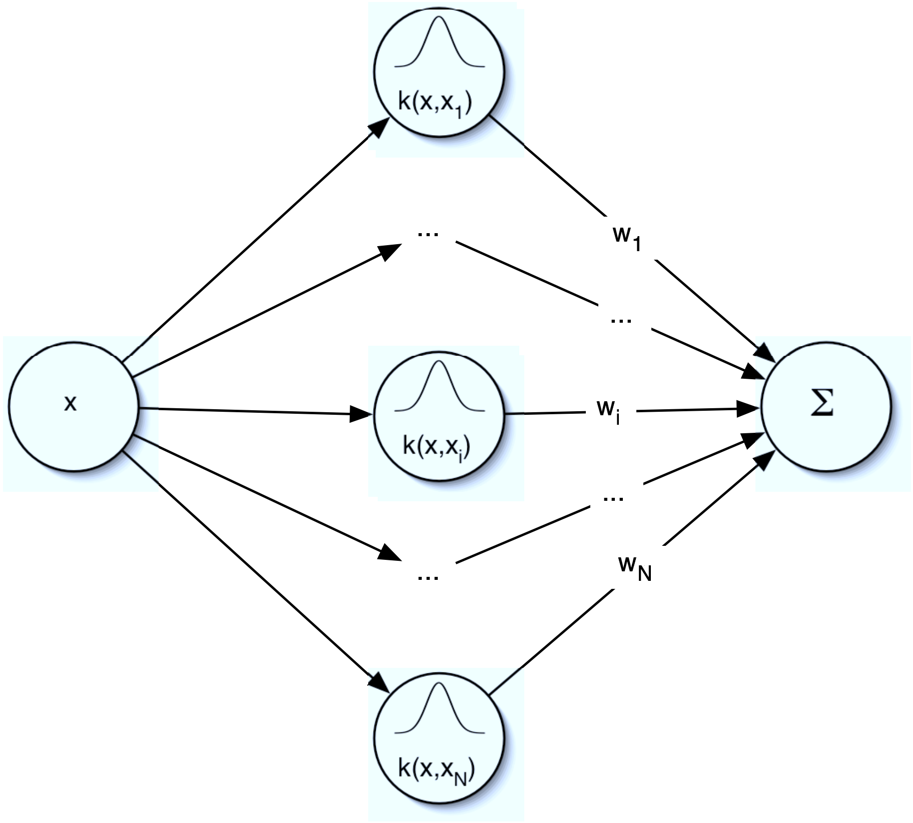

Figure 6: A RBF-network calculates a weighted sum over the responses of N cells with 'tuning curves' given by k.

2.6 Neural networks

The solution to the categorization problem that was discussed in the previous section can be understood as a weighted similarity to the exemplars. Exemplar models like this one can be implemented as a neural network. Imagine a cell that by means of learning has become sensitive to a particular stimulus y—its preferred stimulus. It will still fire if another stimulus x is sufficiently similar to the preferred stimulus. The way that the firing rate of the cell changes with changes in the stimulus is described by its tuning curve that can be modeled by a function like the Gaussian kernel[2]. A simple one layer neural network with several cells that are tuned to certain exemplars, x_1 to x_N, is depicted in Figure 6. Each cell responds to a stimulus x according to its tuning curve, given by k. The activity of all cells is collected by an output neuron that computes a weighted sum of its inputs. The function that this network implements was already given in equation (23).

In the neural network literature a function of the distance between two stimuli is called a radial basis function (RBF) kernel. The Gaussian kernel is the most prominent example for a radial basis function. As neural tuning curves are often found to have a shape like a radial basis function, RBF-networks have repeatedly been advocated as a model for brain function by Poggio and coworkers (Poggio and Girosi, 1989; Poggio, 1990; Poggio and Bizzi, 2004). They have also studied the link to reproducing kernel Hilbert spaces. From a mathematical view-point the problem they are addressing is the learning of an unknown function. Their approach motivates the use of kernels from a function approximation and regularization view.

3 Regularization

By using the exemplar network with as many free parameters as stimuli it is always possible to find weights such that the network can classify all training stimuli perfectly. The price for this flexibility is the danger of overfitting. A network may learn to categorize all training stimuli perfectly but only because it has learned the stimuli by heart. Any regularity in the data is overlooked in this way and therefore the network will not be able to generalize. An example for overfitting is shown in Figure 7. Crosses and circles depict exemplars from two different categories. The two categories are defined by two overlapping Gaussian probability distributions. As we know the distributions that generate the stimuli we can calculate the optimal decision bound which for two Gaussians is a quadratic function (Ashby and Maddox, 1993). The dashed line is this optimal decision bound. The grayscale values depict the function f that was obtained by calculating exemplar weights with a Gaussian kernel as given in

[1] However, note that K might be close to singular in practice.

[2] One important difference between tuning curves being modeled by a Gaussian and psychological similarity that is often also modeled by a Gaussian is of course that similarity is calculated between mental representations whereas tuning curves are usually based on physical measurements.

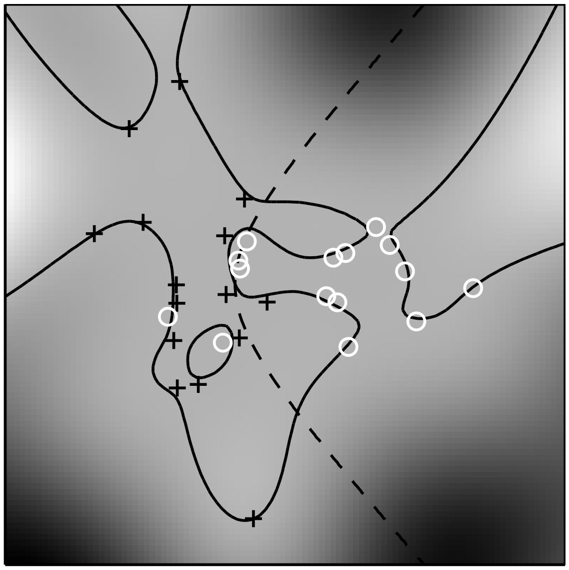

Figure 7: Stimuli from two categories are classified by an exemplar networks as given in equations (23) and (24). The solution is overfitted, that is the training stimuli can be categorized perfectly but generalization to new stimuli will be poor. The optimal decision bound for this category learning problem is shown as a dashed line.

Eq. (24). The solid line is the contour line where $f(x) = 0$. On one side of this contour line the infinite dimensional perceptron would classify stimuli as belonging to one class, on the other side stimuli are classified into the other class. It can be seen that all training stimuli are categorized correctly. The resulting decision bound is obviously not very reasonable, and also very different from the optimal decision bound. Intuitively speaking, the decision bound that the exemplar network calculates is too complicated. The regression function f should not be allowed to vary so wildly and the decision bound should be smoother and less complicated.

One popular strategy to avoid overfitting is regularization (Bishop, 1995; Schölkopf and Smola, 2002; Poggio and Smale, 2003). In regularization there is an additional constraint on the function f that is sought: The function should not only fit the data it should also be smooth and not too "complex". To this end a penalty term is introduced that penalizes functions for being complex. Many modern model selection criteria can be seen as penalizing complexity (Pitt et al., 2002). Instead of only minimizing the error on the training exemplars, which can always be done perfectly, the error plus a penalty term is minimized. The penalty term is also called regularizer.

Let us denote the error that a function f makes on the training exemplars with $c(f)$. Possible examples for such a cost function are the number of misclassifications or the mean square error. If we denote the penalty term by Ω then the function that one seeks to minimize becomes

$$L(f) = c(f) + \Omega(\langle f, f\rangle_{\mathcal{H}}). \tag{25}$$

The penalty term is chosen as a strictly increasing function Ω of $\|f\|^2_{\mathcal{H}} = \langle f, f\rangle_{\mathcal{H}}$, the squared norm of the function in the RKHS that the kernel defines. The learning problem is now understood as trying to find a function f that minimizes L.

3.1 The representer theorem

Ideally, one would like to find the function f that minimizes the regularized error L over all functions in the RKHS. Perhaps surprisingly, the optimal function can be represented as an expansion of the exemplars as given in equation (24). This result is called the *representer theorem*. It shows that for a large class of problems the optimal solution over all functions in the RKHS lies in the span of the exemplars. If the aim of a function learner (be it a brain or a machine) is to minimize a regularized loss then it makes sense to restrict the learning mechanism to an exemplar network of the form (23). The proof that is given by Schölkopf et al. (2001) is short, and it illustrates the power of the RKHS-view of kernels.

As f is in the RKHS we can split it up into a part $f^{\|}$ that lies in the span of the exemplars (23) and a part $f^{\perp}$ that is orthogonal to it. For the first term $c(f)$ in the regularized loss function $L(f)$ we need to evaluate $f = f^{\|} + f^{\perp}$ only on the exemplars x_1, ..., x_N. Remember that $k(x_i, \cdot)$ is the representer of evaluation for x_i and therefore (see Eq. 17)

$$f(x_i) = f^{\|}(x_i) + f^{\perp}(x_i) = f^{\|}(x_i) + \left\langle f^{\perp}, k(x_i, \cdot)\right\rangle_{\mathcal{H}}.$$

The second term is zero because by definition $f^{\perp}$ is orthogonal to $k(x_i, \cdot)$. Hence, the cost function $c(f)$ is independent of $f^{\perp}$. For the penalty term note that

$$\begin{aligned}\Omega(\langle f, f\rangle_{\mathcal{H}}) &= \Omega\left(\left\langle f^{\|}, f^{\|}\right\rangle_{\mathcal{H}} + 2\left\langle f^{\|}, f^{\perp}\right\rangle_{\mathcal{H}} + \langle f^{\perp}, f^{\perp}\rangle_{\mathcal{H}}\right)\\ &= \Omega\left(\left\langle f^{\|}, f^{\|}\right\rangle_{\mathcal{H}} + \langle f^{\perp}, f^{\perp}\rangle_{\mathcal{H}}\right).\end{aligned}$$

For a minimizer of L the orthogonal part $f^{\perp}$ has to be zero. To see this assume f is a minimizer but $f^{\perp}$ is not zero. As Ω is strictly increasing L can be decreased by choosing $f^{\perp}$ to be zero and hence f was not a minimizer—in contradiction to the assumption. Therefore, the best function in the whole RKHS is given by an expansion of the exemplars (23).

The importance of the representer theorem is that if the regularizer can be cast as a strictly increasing function Ω of $\langle f, f \rangle_{\mathcal{H}}$ then the optimal solution over the whole RKHS is a linear combination of kernel functions centered on the exemplars. Therefore, it is not necessary to work in the infinite dimensional function space to find the best function in it. To find the best function one only has to adjust the exemplar weights. Furthermore, the single best function can often be found analytically or with simple numerical procedures. In fact, kernel methods in machine learning are often derived by starting from a regularized loss and providing a method that can find an optimal solution in the span of the exemplars. All this makes exemplar networks so attractive for machine learning and perhaps this can also provide a theoretical justification for assuming a psychological mechanism that is based on storing exemplars. Exemplar models in psychology simply assume that all exemplars are stored without giving a rational explanation why this should be done. If the objective of a subject could be phrased in terms of a regularized loss of the above form (as it is often done is statistics and machine learning) then, as we know that the optimal solution lies in the span of the exemplars, we would have an argument for using exemplar in the first place.

3.2 Regularization example

To understand the rationale behind regularization better and to appreciate the representer theorem it is helpful to look at a more concrete example.

Let us assume the learning proceeds by trying to find a regression function f that minimizes a certain loss $L(f)$ that has two components: One component $c(f)$ that depends on the error on the training stimuli and a component $\Omega(\langle f, f \rangle)$ that penalizes the function's complexity (see Eq. 25, above). As a measure for the error that the learner makes on the training examples we will take the mean square error: $c(f) := \sum_{j=1}^{N} (f(x_j) - y_j)^2$. The squared loss is not the most reasonable loss function for a categorization problem. However, it has been used in psychology before and might be justified by the Rescorla-Wagner rule (Gluck and Bower, 1988). ALCOVE, a prominent exemplar model, for example, uses a different loss function (Kruschke, 1992). Intuitively, it seems better to minimize misclassifications directly and from a statistical view-point one would want to minimize the negative log likelihood. However, we have chosen to minimize the mean square error because it is conceptually easier. For the penalty term Ω we have chosen the simple case where it is linear with a positive parameter λ. The loss function (25) then becomes:

$$L(f) = \sum_{j=1}^{N} (f(x_j) - y_j)^2 + \lambda \langle f, f \rangle_{\mathcal{H}} \,,$$

where y_j is the value that the function should output on exemplar x_j. The parameter λ can be thought of as controlling the trade-off between a good fit and the penalty.

Above we have—perhaps a bit hand-wavingly—referred to the effect of regularization as penalizing "complexity". We can understand what the regularization in equation (25) does by looking at the form of the penalty term which depends on the squared norm $\langle f, f \rangle_{\mathcal{H}}$ of the function f. Because of the representer theorem the optimal function is of the form (24) and we can rewrite the squared norm of the optimal function as

$$\begin{aligned}
\langle f, f \rangle_{\mathcal{H}} &= \left\langle \sum_{i=1}^{N} w_i k(x_i, \cdot), \sum_{j=1}^{N} w_j k(x_j, \cdot) \right\rangle_{\mathcal{H}} \\
&= \sum_{i=1}^{N} \sum_{j=1}^{N} w_i w_j \left\langle k(x_i, \cdot), k(x_j, \cdot) \right\rangle_{\mathcal{H}} \\
&= \sum_{i=1}^{N} \sum_{j=1}^{N} w_i w_j k(x_i, x_j) \\
&= w^T K w,
\end{aligned}$$

where we have used the linearity of the inner product and the reproducing property. In the regularizer the weights of the function f are multiplied by the similarity of the respective exemplars. In order to make two very similar exemplars output very different function values it is necessary to use very big exemplar weights. As large weights for similar exemplars can only be achieved at a high penalty the regularized function respects the similarity measure better than the non-regularized solution in Figure 7.

As we know by the representer theorem that the optimal function is of the form (24) minimizing $L(f)$ is equivalent to finding exemplar weights w such that

$$\begin{aligned}
L(f) &= \sum_{j=1}^{N} (\sum_{i=1}^{N} w_i k(x_i, x_j) - y_j)^2 + \lambda(w^T K w) \\
&= (Kw - y)^T (Kw - y) + \lambda(w^T K w) \\
&= w^T (KK + K\lambda) w - 2y^T K w + y^T K y
\end{aligned}$$

is minimal. The optimal weights can be found analytically by differentiating this quadratic loss function with respect to w. Setting to zero to find the optimum leads to the following solution for w:

$$\begin{aligned}
2(KK + K\lambda)w - 2Ky &= 0 \\
(KK + K\lambda)w &= Ky \\
(K + \lambda I)w &= y \\
w &= (K + \lambda I)^{-1} y.
\end{aligned}$$

The Hessian of the quadratic function L is given by $(KK + K\lambda)$. If the Hessian is a sum of two positive definite matrices it will be positive definite itself. In this case the solution for w is the unique minimum. A regularized solution for f for the same problem as in Figure 7 is shown in Figure 8. The regularized solution is less complicated and looks more reasonable than the non-regularized solution.

The regularized solution is also known as ridge regression. The non-regularized solution (24) can be recovered from ridge regression by setting the regularization parameter λ to zero. In this case the weights are simply calculated by taking the inverse of K. In the case where λ is large and $(K + \lambda I)$ is dominated by the diagonal matrix λI the weights all have the same absolute value and only vary in their sign. The decision bound in this case is identical to the kernel density estimators given in equation (21). In this extreme the decision bound is only determined by the similarity measure and not by the exemplar weights. Hence, the regularization parameter λ makes it possible to choose a solution that is in-between the two extremes: A weight based solution (24) that will always overfit and a similarity based solution (21) with all exemplar weights set to the same value. By allowing this extra flexibility it is often possible to achieve a better generalization performance than by relying on similarity alone. Of course, the value of the regularization parameter λ has to be chosen wisely. In machine learning this is considered as a model selection problem. One way to find a suitable regularization parameter is by using cross-validation and this is what we have done in Figure 8, too (Schölkopf and Smola, 2002; Pitt et al., 2002). Both, the chosen kernel and the regularization parameter, will determine how well the classifier will generalize to new patterns. It is in the construction of the kernel, however, that engineers can use their insights about a classification problem and their intuitions about the similarity of patterns.

4 Conclusions

We have introduced kernel methods as they are used in machine learning. The most important results here are the kernel trick and its link to reproducing kernel Hilbert spaces. On the way we have hinted to parallels with psychological theories. First, kernel methods can be implemented as a one-layer neural network. Second, the Gaussian kernel can be interpreted as a similarity measure and representation of the stimuli in a RKHS can be seen as representing the stimuli via their similarity to all other stimuli. Third, the most simple exemplar model of categorization is a prototype classifier in this RKHS. Fourth, regularization can be used to avoid overfitting. And fifth, the representer theorem shows that the best regularized function in RKHS can often be found in the span of the exemplars.

In a companion paper (Jäkel et al., 2007b), we describe how the RKHS framework arises naturally from Shepard's universal law of generalization. Shepard's law is closely related to geometric models of similarity and multi-dimensional scaling. Geometric models have been heavily criticized by Tversky and coworkers (Beals et al., 1968; Tversky, 1977; Tversky and Gati, 1982). One of their major criticisms concerns the assumption of the triangle inequality that is

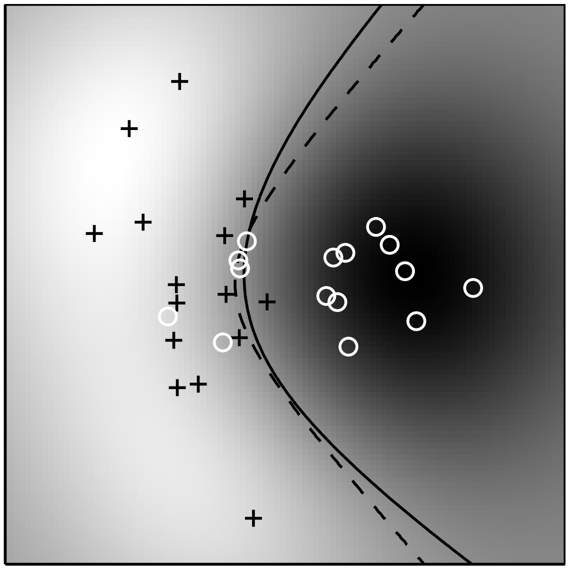

Figure 8: The same categorization problem as in Figure 7 but this time a regularized solution is shown. The regularized decision bound (solid line) is quite close to the optimal decision bound (dashed line).

implicit in all geometric models. However, from the data that are available the triangle inequality is unproblematic as an assumption as long as it is not paired with a second assumption that is known as segmental additivity. The RKHS provides an elegant framework for metric models without segmental additivity.

From this tutorial it should be obvious that exemplar models and kernel methods are based on the same ideas. Their relationship is discussed in greater detail in another forthcoming paper (Jäkel et al., 2007a). Briefly, two very prominent exemplar models, the Generalized Context Model (Nosofsky, 1986) and ALCOVE (Kruschke, 1992), both make use of kernels but in different ways. Only ALCOVE can be mapped directly to a machine learning method and even exhibits some regularization.

We imagine other potential uses for the mathematical tools we have presented. First of all, we hope that this tutorial opens up the recent machine learning literature for more psychologists. Many of the data analytic methods presented in machine learning could be used in psychology—irrespective whether they use the RKHS framework that was the focus here. For example, machine learning methods have been used in psychophysics (Wichmann et al., 2005; Graf et al., 2006) and we believe that many more applications will follow. With regard to Hilbert spaces one can be skeptic whether the infinite dimensional machinery is really necessary for psychological modeling. However, there are many cases where the data that are collected are in terms of functions and therefore naturally described with methods similar to the ones described here (Ramsay and Silverman, 1997). Furthermore, several other authors have recently suggested to use infinite dimensional spaces in the theoretical analysis of behavior (Drösler, 1994; Townsend et al., 2001; Zhang, 2006). In the laboratory, stimuli are almost always defined by a small number of independent variables, and those are the stimuli that we used as examples in this paper. In these examples the approach was to map stimuli from a space with a small number of dimensions to an infinite dimensional space. More realistic stimuli will vary in a plethora of ways and not just along a small number of well-defined dimensions. Infinite dimensional spaces could be attractive for describing such natural stimuli—take for example the features of a face, the shape of a leaf, or the spectrum of a light source. Also the number of channels that humans use to analyze these stimuli might be very large—too large to enumerate them explicitly. The tools we presented in this tutorial might be useful for this enterprise.

References

Aizerman, M. A., Braverman, E. M., Rozonoer, L. I., 1964. Theoretical foundations of the potential function method in pattern recognition learning. Automation and Remote Control 25 (6), 821–837.

Ashby, F. G., Alfonso-Reese, L. A., 1995. Categorization as probability density estimation. Journal of Mathematical Psychology 39, 216–233.

Ashby, F. G., Gott, R. E., 1988. Decision rules in the perception and categorization of multidimensional stimuli. Journal of Experimental Psychology: Learning, Memory and Cognition 14 (1), 33–53.

Ashby, F. G., Maddox, W. T., 1993. Relations between prototype, exemplar, and decision bound models of categorization. Journal of Mathematical Psychology 37, 372–400.

Beals, R., Krantz, D. H., Tversky, A., 1968. Foundations of multidimensional scaling. Psychological Review 75, 127–142.

Bishop, C. M., 1995. Neural Networks for Pattern Recognition. Oxford University Press, Oxford.

Christianini, N., Shawe-Taylor, J., 2000. An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods. Cambridge University Press, Cambridge.

Drösler, J., 1994. Color similarity represented as a metric of color space. In: Fischer, G. H., Laming, D. (Eds.), Contributions to Mathematical Psychology, Psychometrics, and Methodology. Springer, Berlin, pp. 19–37.

Edelman, S., 1998. Representation is representation of similarities. Behavioral and Brain Sciences 21, 449–498.

Fried, L. S., Holyoak, F. J., 1984. Induction of category distributions: A framework for classification learning. Journal of Experimental Psychology: Learning, Memory, and Cognition 10, 234–257.

Gluck, M. A., Bower, G. H., 1988. Evaluating an adaptive network model of human learning. Journal of Memory and Language 27 (2), 166–195.

Graf, A. B. A., Wichmann, F. A., Bülthoff, H. H., Schölkopf, B., 2006. Classification of faces in man and machine. Neural Computation 18, 143–165.

Hornik, K., Stinchcombe, M., White, H., 1989. Multilayer feedforward networks are universal approximators. Neural Networks 2, 359–366.

Jäkel, F., Schölkopf, B., Wichmann, F. A., 2007a. Generalization and similarity in exemplar models of categorization: Insights from machine learning. Submitted.

Jäkel, F., Schölkopf, B., Wichmann, F. A., 2007b. Similarity, kernels and the triangle inequality. Submitted.

Kruschke, J. K., 1992. ALCOVE: an exemplar-based connectionist model of category learning. Psychological Review 99 (1), 22–44.

Medin, D. L., Schwanenflugel, P. J., 1981. Linear separability in classification learning. Journal of Experimental Psychology: Human Learning and Memory 1, 355–368.

Minsky, M., Papert, S., 1967. Linearly unrecognizable patterns. AIM 140, MIT.

Nilsson, N. J., 1965. Learning Machines. McGraw-Hill, New York.

Nosofsky, R. M., 1986. Attention, similarity, and the indentification-categorization relationship. Journal of Experimental Psychology: General 115, 39–57.

Nosofsky, R. M., 1990. Relations between exemplar-similarity and likelihood models of classification. Journal of Mathematical Psychology 34, 393–418.

Pitt, M. A., Myung, I. J., Zhang, S., 2002. Toward a method of selecting among computational models of cognition. Psychological Review 109 (3), 472–491.

Poggio, T., 1990. A theory of how the brain might work. Cold Spring Harbor Symposia on Quantitative Biology LV, 899–910.

Poggio, T., Bizzi, E., 2004. Generalization in vision and motor control. Nature 431 (7010), 768–774.

Poggio, T., Girosi, F., 1989. A theory of networks for approximation and learning. Tech. Rep. A. I. Memo No. 1140, MIT AI LAB and Center for Biological Information Processing Whitaker College, Cambridge, MA.

Poggio, T., Smale, S., 2003. The mathematics of learning: Dealing with data. Notices of the American Mathematical Society 50 (5), 537–544.

Posner, M. I., Keele, S. W., 1968. On the genesis of abstract ideas. Journal of Experimental Psychology 77, 353–363.

Ramsay, J. O., Silverman, B. W., 1997. Functional Data Analysis. Springer, New York.

Reed, S. K., 1972. Pattern recognition and categorization. Cognitive Psychology 3, 382–407.

Rosenblatt, F., 1958. The perceptron: A probabilistic model for information storage and organization in the brain. Psychological Review 65, 386–408.

Rumelhart, D. E., Hinton, G. E., Williams, R. J., 1986. Learning Internal Representations by Error Propagation. MIT Press, Cambridge, MA, Ch. 8, pp. 318–362.

Schoenberg, I. J., 1938. Metric spaces and positive definite functions. Transactions of the American Mathematical Society 44 (3), 522–536.

Schölkopf, B., Herbrich, R., Smola, A. J., 2001. A generalized representer theorem. In: Helmbold, D., Williamson, R. (Eds.), Computational Learning Theory. Vol. 2111. Springer, Berlin, pp. 416–426.

Schölkopf, B., Smola, A., 2002. Learning with Kernels. MIT Press, Cambridge, MA.

Schölkopf, B., Smola, A. J., Müller, K.-R., 1998. Nonlinear component analysis as a kernel eigenvalue problem. Neural Computation 10 (5), 1299–1319.

Shepard, R. N., Hovland, C. I., Jenkins, H. M., 1961. Learning and memorization of classifications. Psychological Monographs 75 (13), 1–42.

Smith, J. D., Murray, M. J., Minda, J. P., 1997. Straight talk about linear separability. Journal of Experimental Psychology: Learning, Memory, and Cognition 23 (3), 659–680.

Strang, G., 1988. Linear Algebra and its Applications, 3rd Edition. Harcourt Brace & Company, Orlando, FL.

Townsend, J. T., Solomon, B., Smith, J. S., 2001. The perfect Gestalt: Infinite dimensional Riemannian face space and other aspects of face perception. In: Wenger, M. J., Townsend, J. T. (Eds.), Computational, Geometric, and Process Perspectives of Facial Cognition: Contexts and Challenges. Lawrence Erlbaum Associates, Mahwah, NJ, pp. 39–82.

Tversky, A., 1977. Features of similarity. Psychological Review 84 (4), 327–352.

Tversky, A., Gati, I., 1982. Similarity, separability, and the triangle inequality. Psychological Review 89 (2), 123–154.

Wichmann, F. A., Graf, A. B. A., Simoncelli, E. P., Bülthoff, H. H., Schölkopf, B., 2005. Machine learning applied to perception: decision-images for classification. In: Saul L. K., Weiss Y., Bottou, L. (Eds.), Advances in Neural Information Processing Systems 17. MIT Press, Cambridge, MA, pp. 1489–1496.

Zhang, J., 2006. Referential duality and representational duality in the scaling of multidimensional and infinite-dimensional stimulus space. In: Colonius, H., Dzhafarov, E. N. (Eds.), Measurement and Representation of Sensations. Lawrence Erlbaum Associates, Mahwah, NJ, Ch. 4, pp. 131–156.