Kernel Methods in Medical Imaging

G. Charpiat, M. Hofmann, B. Schölkopf

ABSTRACT We introduce machine learning techniques, more specifically kernel methods, and show how they can be used for medical imaging. After a tutorial presentation of machine learning concepts and tools, including Support Vector Machine (SVM), kernel ridge regression and kernel PCA, we present an application of these tools to the prediction of Computed Tomography (CT) images based on Magnetic Resonance (MR) images.

1 Introduction

Machine learning has shown dramatic progress over the last decades, creating tools like the well-known Support Vector Machine (SVM), which have been intensively applied to many different fields and have proved their efficiency. Learning tools have often changed the whole perspective of the issues they have been applied to. For example, in computer vision, the detection of objects in images, and the automatic classification of images into categories (landscape, car, etc.) rely now most often on intensive patch-based learning, whereas it was previously commonly thought that a complete image segmentation would be required. The results of this machine learning approach are often surprisingly good, showing that under certain conditions, many tasks are much easier to solve by incorporating prior knowledge retrieved from a set of examples.

In medical imaging, approaches are often *example*-based, in the sense that the aim often consists in the automatization of a task already performed by hand by medical people on a few examples, such as segmentation, registration, detection (of tumors, of organs) or classification. As medical imaging deals with images, there is also much inspiration to get from what has already been achieved in computer vision, in object detection [9] as well in shape priors [5].

We start here with a tutorial on machine learning techniques. We present basic concepts, and then focus on kernel methods. We introduce standard tools like kernel ridge regression, SVM and kernel PCA. Then we apply some of these tools to the case of medical image prediction, when the Magnetic Resonance scan of a patient is known and we would like to guess what the corresponding Computed Tomography scan would look like.

2 Machine learning with kernels

This section describes the central ideas of kernel methods in a nutshell by providing an overview of the basic concepts. We first state mathematically the problems of classification and regression. Then we introduce the concept of *kernel* and explain the *kernel trick* which leads to kernel PCA as well as kernel ridge regression. The last concept introduced is the *support vector* (SV), which is the basis of the SVM. We have tried to keep this tutorial as basic as possible and refer to [11] for further details.

2.1 Basics

Classification and Regression
Suppose we are given a set of m objects $(x_i)_{1\leqslant i\leqslant m} \in \mathcal{X}^m$ with labels $(y_i)_{1\leqslant i\leqslant m} \in \mathcal{Y}^m$. If the number of possible labels is finite and small, then we can be interested in *classification*, i.e. in finding the label to assign to a new object based on the given examples. Otherwise, if the labels are values in a vector space, we can be interested in *regression*, i.e. in extrapolating previously observed values to any new object. Thus classification and regression tasks can be embedded in a similar framework, one aiming to predict discrete labels and the other one to continuous values.

The objects $(x_i)_{1\leqslant i\leqslant m}$ are often named *patterns*, or *cases*, *inputs*, *instances*, or *observations*. The $(y_i)_{1\leqslant i\leqslant m}$ are called *labels*, or *targets*, *outputs* or sometimes also *observations*. The set of all correspondences $(x_i, y_i)_{1\leqslant i\leqslant m}$ given as examples is called the *training set*, whereas we name *test set* the set of new objects for which we would like to guess the label by extracting knowledge from the examples in the training set.

In both cases, classification or regression, we aim to generalize the correspondences (x_i, y_i) to a function f defined on the set $\mathcal{X}$ of all possible objects and with values in the set $\mathcal{Y}$ of all possible labels. The label predicted for a new test object x would then be $f(x)$. Here we have no particular assumption on the spaces $\mathcal{X}$ and $\mathcal{Y}$ except that $\mathcal{Y}$ should be a vector space if we are interested in regression (in order to extrapolate continuously between any two values). But we have a strong intuitive assumption on f: it should generalize as well as possible the given examples, i.e. if x is *close* to an already observed input x_i, its output $f(x)$ should be *close* to the already observed output y_i. The whole difficulty consists in defining precisely what we mean by "close" in the spaces $\mathcal{X}$ and $\mathcal{Y}$. More precisely, we need to quantify the similarity of inputs in $\mathcal{X}$ and the cost of assigning wrong outputs in $\mathcal{Y}$.

Loss function
Generally, expressing a distance or similarity measure in $\mathcal{Y}$ is easy. In the case of regression, the Euclidean distance in $\mathcal{Y}$ is often a simple, convenient choice. However we can consider other functions than distances, provided they express the cost of assigning a wrong label. We call the *loss function*

the sum of the costs (or losses) of all mistakes made when we consider a particular possible solution f and apply it to all known examples. For instance we can choose:

$$L(f, (x_i, y_i)_{1 \leqslant i \leqslant m}) = \sum_{i=1}^{m} \|f(x_i) - y_i\|_y$$

Duality between features and similarity measures
On the other hand, expressing a similarity measure in $\mathcal{X}$ is much more difficult and lies at the core of machine learning. Either the space $\mathcal{X}$ has been carefully chosen so that the representation of the observed objects x_i are meaningful, in the sense that their "natural" distance in $\mathcal{X}$ (say the Euclidean distance if $\mathcal{X}$ is a vector space) is meaningful, in which case learning will be easy; either $\mathcal{X}$ is non-trivial and we need to choose a set of N sensible *features* (seen as a function Φ from $\mathcal{X}$ to $\mathcal{H} = \mathbb{R}^N$), so that if we compute these *features* $\Phi(x_i)$ for each x_i, we can consider a more natural distance in the *feature space* $\mathcal{H}$. From a certain point of view, choosing a sensible *feature map* Φ or choosing a sensible distance in $\mathcal{X}$ (or in the feature space $\mathcal{H}$) are equivalent problems, and hence equivalently hard in the general case.

Optimization problem over functions
The problem of classification or regression can be written as an optimization problem over all possible functions f: find the best function f from $\mathcal{X}$ to $\mathcal{Y}$ such that it minimizes

$$L(f, (x_i, y_i)_{1 \leqslant i \leqslant m}) + R(f)$$

where $R(f)$ is a regularizer constraining f to be *smooth* in some way with respect to the similarity measure chosen in $\mathcal{X}$. Note that we could also have restricted f to be a member of a small function space $\mathcal{F}$. There are very nice theoretical results concerning the function space in the kernel case (see for example section 2.3 about ridge regression).

2.2 Kernels

This section aims to define kernels and to explain all facets of the concept. It is a preliminary step to the following sections dedicated to kernel algorithms themselves.

A *kernel* is any symmetric similarity measure on $\mathcal{X}$

$$\begin{array}{rcl} k : \mathcal{X} \times \mathcal{X} & \rightarrow & \mathbb{R} \\ (x, x') & \mapsto & k(x, x'), \end{array}$$

that is, a symmetric function that, given two inputs x and x', returns a real number characterizing their similarity (cf. [10, 1, 3, 4, 7]).

Kernels as inner products in the feature space
In the general case, either $\mathcal{X}$ is not a vector space, or the natural Euclidean inner product in $\mathcal{X}$ is not particularly relevant as a similarity measure. Most often, a set of possibly-meaningful *features* is available, and we can consequently use the *feature map*

$$\begin{aligned} \Phi : \mathcal{X} &\rightarrow \mathcal{H} \\ x &\mapsto \mathbf{x} := \Phi(x). \end{aligned}$$

Φ will typically be a nonlinear map with values in a vector space. It could for example compute products of components of the input x. We have used a bold face $\mathbf{x}$ to denote the vectorial representation of x in the feature space $\mathcal{H}$. We will follow this convention throughout the chapter.

We can use the non-linear embedding of the data into the linear space $\mathcal{H}$ via Φ to define a similarity measure from the dot product in $\mathcal{H}$,

$$k(x, x') := \langle \mathbf{x}, \mathbf{x}' \rangle_{\mathcal{H}} = \langle \Phi(x), \Phi(x') \rangle_{\mathcal{H}} . \tag{1.1}$$

The freedom to choose the mapping Φ will enable us to design a large variety of similarity measures and learning algorithms. The transformation of x_i into $\Phi(x_i) = \mathbf{x}_i$ can be seen as a change of the inputs, i.e. as a new model of the initial problem. However, we will see later that, in some cases, we won't need to do this transformation explicitly, which is very convenient if the number of features considered (or the dimension of $\mathcal{H}$) is high.

Geometrical interpretation and kernel trick
Through the definition of k, we can provide geometric interpretation of the input data:

$$\|\mathbf{x}\|_{\mathcal{H}} = \|\Phi(x)\|_{\mathcal{H}} = \sqrt{\langle \Phi(x), \Phi(x) \rangle_{\mathcal{H}}} = \sqrt{k(x, x)}$$

is the *length* (or *norm*) of $\mathbf{x}$ in the feature space. Similarly, $k(x, x')$ computes the cosine of the angle between the vectors $\mathbf{x}$ and $\mathbf{x}'$, provided they are normalized to length 1. Likewise, the distance between two vectors is computed as the length of the difference vector:

$$\|\mathbf{x} - \mathbf{x}'\|^2_{\mathcal{H}} = \|\mathbf{x}\|^2 + \|\mathbf{x}'\|^2 - 2 \langle \Phi(x), \Phi(x') \rangle = k(x, x) + k(x', x') - 2k(x, x').$$

The interesting point is that we could consider any such similarity measure k and forget about the associated Φ: we would still be able to compute lengths, distances and angles with the only knowledge of k thanks to these formulas. This framework allows us to deal with the patterns geometrically through a understated non-linear embedding, and thus lets us study learning algorithms using linear algebra and analytic geometry. This is known as the *kernel trick*: any algorithm dedicated to Euclidean geometry involving only distances, lengths and angles can be *kernelized* by replacing all

occurrences of these geometric quantities by their expressions as a function of k. Next section is dedicated to such *kernelizations*.

Examples of kernels
Let us introduce the most-commonly used kernels. They are namely: the polynomial kernel

$$k(x, x') = \langle x, x' \rangle^d ,$$

and the Gaussian

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

for suitable choices of d and σ. Let us focus on the Gaussian case: the similarity measure $k(x, x')$ between x and x' is always positive, and is maximal when $x = x'$. All points $\mathbf{x}$ have the same unit norm (since $k(x, x) = 1\ \forall x$) and consequently the images of all points x in the associated feature space $\mathcal{H}$ lie on the unit sphere.

Reproducing kernels as feature maps
One could wonder what is the feature map Φ which was used to build the Gaussian kernel. In fact kernel theory goes far beyond the way we introduced kernels. Let us consider any symmetric function k, not necessarily related to a feature map. Let us suppose also that k, seen as an operator, is positive definite, that is to say that for any L^2 function $\alpha : \mathcal{X} \to \mathbb{R}$:

$$\int_{\mathcal{X}\times\mathcal{X}} \alpha(x) k(x, x') \alpha(x')\, dx\, dx' \ \geqslant 0.$$

Note that to be able to integrate over $x \in \mathcal{X}$, we need a measure on $\mathcal{X}$. This measure is often thought of as a *probability measure* over $\mathcal{X}$, giving more weight to objects that are more likely to appear.

Then we can define *from this kernel* k an associated feature map by:

$$\begin{array}{rcl} \Phi : \mathcal{X} & \to & \mathcal{F}(\mathcal{X}) \\ x & \mapsto & \mathbf{x} := k(x, \cdot). \end{array} \tag{1.2}$$

This image of any input x by Φ is the function

$$\begin{array}{rcl} k(x, \cdot) : \ \mathcal{X} & \to & \mathbb{R} \\ x' & \mapsto & k(x, x'). \end{array} \tag{1.3}$$

Φ has now values in the space $\mathcal{F}(\mathcal{X})$ of functions over $\mathcal{X}$ instead of having values in just a finite dimensioned vector space like $\mathbb{R}^N$.

The magic comes from Moore-Aronszajn theorem [2] which states that it is *always* possible, for any symmetric positive definite function k, to build a *reproducing kernel Hilbert space* (RKHS) $\mathcal{H} \subset \mathcal{F}(\mathcal{X})$ so that

$$\forall x, x' \in \mathcal{X}, \qquad k(x, x') = \langle k(x, \cdot), k(x', \cdot) \rangle_{\mathcal{H}} = \langle \Phi(x), \Phi(x') \rangle_{\mathcal{H}} . \tag{1.4}$$

Because of such a property, symmetric positive definite kernels are also called *reproducing kernels*. This theorem highlights the duality between *reproducing kernels* k and *feature maps* Φ: choosing the feature space or choosing the kernel is equivalent, since one determines the other.

The Gaussian case (details)
We can make explicit the inner product on $\mathcal{H}$ in the Gaussian case. The associated norm is

$$\|f\|^2_{\mathcal{H}} = \int_{\mathcal{X}} \sum_{n=0}^{\infty} \frac{\sigma^{2n}}{n!\ 2^n} \left(\frac{d^n}{dx^n} f\right)^2(x)\ dx = \sum_{n=0}^{\infty} \frac{\sigma^{2n}}{n!\ 2^n} \left\| \frac{d^n}{dx^n} f \right\|^2_{L^2(\mathcal{X})}$$

which penalizes all fast variations of f at all derivative orders. We refer to [6] for a more general mathematical study of radial basis functions. Intuitively, consider the operator $P = e^{-\frac{\sigma^2}{2}\frac{d^2}{dx^2}} := \sum_n \frac{(-\sigma^2/2)^n}{n!} \frac{d^{2n}}{dx^{2n}}$. In the Fourier domain, it writes $e^{\sigma^2 w^2/2}$, whereas $k(x,\cdot)$ becomes $\sigma e^{-\sigma^2 w^2/2} e^{-iwx}$. Thus $\frac{1}{\sigma} P(k(x,\cdot)) = \delta_x(\cdot)$ is a Dirac peak in the space $\mathcal{D}(\mathcal{X})$ of distributions over $\mathcal{X}$. The inner product $\langle f, g\rangle_{\mathcal{H}} := \left\langle \frac{1}{\sigma} P(f),\ g \right\rangle_{\mathcal{D}(\mathcal{X})}$ on $\mathcal{H}$ will therefore satisfy:

$$\langle k(x,\cdot), f\rangle_{\mathcal{H}} := \left\langle \frac{1}{\sigma} P\left(k(x,\cdot)\right),\ f \right\rangle_{\mathcal{D}(\mathcal{X})} = \langle \delta_x, f\rangle_{\mathcal{D}(\mathcal{X})} = f(x)$$

hence, for the particular case $f = k(x',\cdot)$,

$$\langle k(x,\cdot),\ k(x',\cdot)\rangle_{\mathcal{H}} = k(x,x').$$

The overfitting problem
The kernel k should be chosen carefully, since it is the core of the generalization process: if the neighborhood induced by k is too small (for instance if k is a Gaussian with a tiny standard deviation σ), then we will overfit the given examples without being able to generalize to new points (which would be found very dissimilar to all examples). On the contrary, if the neighborhood is too large (for instance if k is a Gaussian with a standard deviation so huge that all examples are considered as very similar), then it is not possible to distinguish any clusters or classes.

Kernels as regularizers
We introduced initially kernels as similarity measures on the space $\mathcal{X}$ of inputs. But with the reproducing kernel framework, the choice of a kernel implies a structure on the space of functions from $\mathcal{X}$ to $\mathbb{R}$, in particular it defines a norm on this space. Consequently choosing a kernel is the same as choosing a *regularizer* on the function space.

Let us go back to the initial problem, and, for the sake of simplicity, let us consider the case where the output space $\mathcal{Y}$ is included in $\mathbb{R}$. We expressed the classification or regression problem as the search for the best

function f from $\mathcal{X}$ to $\mathcal{Y}$ that minimizes a loss plus a regularizer on f. We have found here a natural way to choose the regularizer according to the kernel, i.e. $R(f) = \|f\|^2_{\mathcal{H}}$.

A whole class of problems involving this norm can be shown [11] to have solutions in the span of functions $k(x_i, \cdot)$, i.e. solutions f have the form $f(x) = \sum_i \alpha_i k(x_i, x)$. Interestingly, this allows the reduction of the search space (the function space) to a linear, finite-dimensioned subspace, while involving non-linear quantities (the kernels $k(x, x')$ or the features $\Phi(x)$).

2.3 Kernelization of existing linear algorithms

We now have all the concepts required to transform existing algorithms dealing linearly with data into kernel methods. We consider standard, simple algorithms such as PCA and linear regression and build out of them more efficient tools which take advantage of the prior knowledge provided by the definition of a kernel and of their ability to deal linearly with non-linear quantities.

A very simple hyperplanar classifier
To show the spirit of *kernelization*, let us first describe a very simple learning algorithm for binary classification. The label space $\mathcal{Y}$ contains only two elements, $+1$ and -1, and the training set consists of labeled examples of the two classes. The basic idea is to assign any previously unseen pattern x to the class with closest mean. Let us work directly in the feature space $\mathcal{H}$ and deal with $\mathbf{x} = \Phi(x)$ instead of x since the metric which makes sense is the one in the feature space. In $\mathcal{H}$, the means of the two classes are:

$$\mathbf{c}_+ = \frac{1}{m_+} \sum_{\{i|y_i=+1\}} \mathbf{x}_i \quad \text{and} \quad \mathbf{c}_- = \frac{1}{m_-} \sum_{\{i|y_i=-1\}} \mathbf{x}_i, \tag{1.5}$$

where m_+ and m_- are the number of examples with positive and negative labels, respectively. Half way between $\mathbf{c}_+$ and $\mathbf{c}_-$ lies the point $\mathbf{c} := (\mathbf{c}_+ + \mathbf{c}_-)/2$. We compute the class of $\mathbf{x}$, based on the angle between the vector $\mathbf{x} - \mathbf{c}$ and the vector $\mathbf{w} := \mathbf{c}_+ - \mathbf{c}_-$ (see figure 1):

$$\begin{aligned} y &= \operatorname{sgn} \langle (\mathbf{x} - \mathbf{c}), \mathbf{w} \rangle_{\mathcal{H}} = \operatorname{sgn} \langle (\mathbf{x} - (\mathbf{c}_+ + \mathbf{c}_-)/2), (\mathbf{c}_+ - \mathbf{c}_-) \rangle_{\mathcal{H}} \\ &= \operatorname{sgn} \left(\langle \mathbf{x}, \mathbf{c}_+ \rangle_{\mathcal{H}} - \langle \mathbf{x}, \mathbf{c}_- \rangle_{\mathcal{H}} + b \right) \end{aligned} \tag{1.6}$$

$$\text{where we have defined the offset} \qquad b := \frac{1}{2}(\|\mathbf{c}_-\|^2_{\mathcal{H}} - \|\mathbf{c}_+\|^2_{\mathcal{H}}). \tag{1.7}$$

Note that (1.6) induces a decision boundary which has the form of a hyperplane in the feature space. We can now call the kernel trick in order to express all quantities as a function of the kernel, which is the only thing we can easily compute (unless Φ is explicit and simple). But this trick deals only with norms, distances and angles of features points of the form

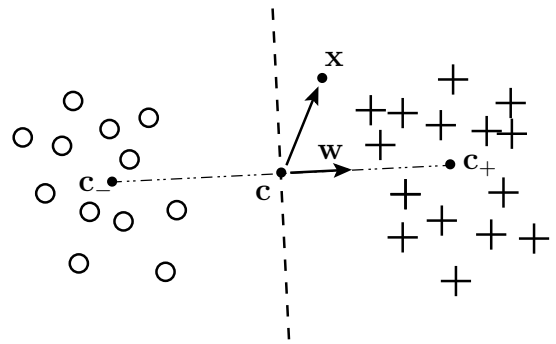

FIGURE 1. A very simple classifier in the feature space: associate to any new point $\mathbf{x}$ the class whose mean c_i is the closest. The decision boundary is an hyperplane.

$\mathbf{x} = \Phi(x)$, for which we already know x. Therefore we need to express the vectors $\mathbf{c}_i$ and $\mathbf{w}$ in terms of $\mathbf{x}_1, \ldots, \mathbf{x}_m$.

To this end, substitute (1.5) into (1.6) to get the *decision function*

$$\begin{aligned} y &= \operatorname{sgn}\Big(\frac{1}{m_+} \sum_{\{i|y_i=+1\}} \langle \mathbf{x}, \mathbf{x}_i \rangle_{\mathcal{H}} - \frac{1}{m_-} \sum_{\{i|y_i=-1\}} \langle \mathbf{x}, \mathbf{x}_i \rangle_{\mathcal{H}} + b \Big) \\ &= \operatorname{sgn}\Big(\frac{1}{m_+} \sum_{\{i|y_i=+1\}} k(x, x_i) - \frac{1}{m_-} \sum_{\{i|y_i=-1\}} k(x, x_i) + b \Big). \quad (1.8) \end{aligned}$$

Similarly, the offset becomes

$$b := \frac{1}{2}\Big(\frac{1}{m_-^2} \sum_{\{(i,j)|y_i=y_j=-1\}} k(x_i, x_j) \; - \; \frac{1}{m_+^2} \sum_{\{(i,j)|y_i=y_j=+1\}} k(x_i, x_j) \Big). \quad (1.9)$$

Surprisingly, it turns out that this rather simple-minded approach contains a well-known statistical classification method as a special case. Assume that the class means have the same distance to the origin (hence $b = 0$, cf. (1.7)), and that k can be viewed as a probability density when one of its arguments is fixed. By this we mean that it is positive and that $\forall x' \in \mathcal{X}$, $\int_{\mathcal{X}} k(x, x')dx = 1$. In this case, (1.8) takes the form of the so-called Bayes classifier separating the two classes, subject to the assumption that the two classes of patterns were generated by sampling from two probability distributions that are correctly estimated by the *Parzen windows* estimators of the two class densities,

$$p_+(x) := \frac{1}{m_+} \sum_{\{i|y_i=+1\}} k(x, x_i) \quad \text{and} \quad p_-(x) := \frac{1}{m_-} \sum_{\{i|y_i=-1\}} k(x, x_i). \quad (1.10)$$

Given some point x, the label is then simply computed by checking which of the two values $p_+(x)$ or $p_-(x)$ is larger, which leads directly to (1.8).

Note that this decision is the best we can do if we have no prior information about the probabilities of the two classes.

The classifier (1.8) is a particular case of a more general family of classifiers, which are of the form of an affine combination of kernels on the input domain, $y = \mathrm{sgn}\,(\sum_{i=1}^{m} \alpha_i k(x, x_i) + b)$. The affine combination corresponds to a separating hyperplane in the feature space. In this sense, the α_i can be considered a *dual representation* of the hyperplane's normal vector [7]. These classifiers are example-based in the sense that the kernels are centered on the training patterns; that is, one of the two arguments of the kernel is always a training pattern. A test point is classified by comparing it to all the training points with a nonzero weight α_i. One of the great benefits that SVM brings in next section is the assignment of a zero weight to most training points and the sensible selection of the ones kept for classification.

Principal component analysis

Suppose we are given a set of unlabeled points, or a set of points of the same class. In the case of a vector space, we could perform a principal component analysis (PCA) to extract the main axes of the cloud of points. These main axes can then be used as a low-dimensional coordinate system expressing most of the information contained in the initial vector coordinates.

PCA in feature space leads to an algorithm called *kernel PCA* [12]. By solving an eigenvalue problem, the algorithm computes nonlinear feature extraction functions

$$f_n(x) = \sum_{i=1}^{m} \alpha_i^n k(x_i, x), \tag{1.11}$$

where, up to a normalizing constant, the α_i^n are the components of the nth eigenvector of the kernel matrix $K_{ij} := (k(x_i, x_j))$.

In a nutshell, this can be understood as follows. To perform PCA in $\mathcal{H}$, we need to find eigenvectors $\mathbf{v}$ and eigenvalues λ of the so-called *covariance matrix* $\mathbf{C}$ in the feature space, where

$$\mathbf{C} := \frac{1}{m} \sum_{i=1}^{m} \Phi(x_i)\Phi(x_i)^\top. \tag{1.12}$$

Here, $\Phi(x_i)^\top$ denotes the transpose of $\Phi(x_i)$. When $\mathcal{H}$ is very high dimensional, the computational costs of doing this directly are prohibitive. Fortunately, one can show that all solutions to

$$\mathbf{C}\mathbf{v} = \lambda \mathbf{v} \tag{1.13}$$

with $\lambda \neq 0$ must lie in the span of Φ-images of the training data. Thus, we may expand the solution $\mathbf{v}$ as

$$\mathbf{v} = \sum_{i=1}^{m} \alpha_i \Phi(x_i), \tag{1.14}$$

thereby reducing the problem to that of finding the α_i. It turns out that this leads to a dual eigenvalue problem for the expansion coefficients,

$$K\boldsymbol{\alpha} = m\lambda\boldsymbol{\alpha}, \tag{1.15}$$

where $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_m)^\top$.

To extract nonlinear features from a test point x, we compute the dot product between $\Phi(x)$ and the nth normalized eigenvector in feature space,

$$\langle \mathbf{v}^n, \Phi(x) \rangle = \sum_{i=1}^{m} \alpha_i^n k(x_i, x). \tag{1.16}$$

Usually, this will be computationally far less expensive than taking the dot product in the feature space explicitly.

Kernel ridge regression and the representer theorem
Let us now consider the case of regression: we know the values $y_i \in \mathbb{R}$ of a function at m given points $(x_i)_{1 \leqslant i \leqslant m}$ and we would like to interpolate it to any new point $x \in \mathcal{X}$. The notion of regression requires the one of regularization, so we choose a kernel k and use the associated norm $\|\cdot\|_{\mathcal{H}}$. The problem can be expressed mathematically as the search for the best function $f : \mathcal{X} \to \mathbb{R}$ which minimizes a weighted sum of the prediction errors $(f(x_i) - y_i)^2$ at known points and the regularity cost $\|f\|_{\mathcal{H}}$:

$$\inf_{f:\mathcal{X}\to\mathbb{R}} \left\{ \sum_{i=1}^{m} (f(x_i) - y_i)^2 \; + \; \lambda \|f\|_{\mathcal{H}}^2 \right\} \tag{1.17}$$

Representer Theorem *The solution f of (1.17) in the RKHS belongs to the span of functions $k(x_i, \cdot)$ and thus admits a representation of the form*

$$f(x) = \sum_{j=1}^{m} \alpha_j k(x_j, x). \tag{1.18}$$

More details can be found in ([11], p. 89). Using (1.18) and (1.4), the problem (1.17) becomes:

$$\inf_{\boldsymbol{\alpha}\in\mathbb{R}^m} \left\{ \sum_{i=1}^{m} \Big(\sum_j \alpha_j k(x_j, x_i) - y_i \Big)^2 + \lambda \sum_{i,j} \alpha_i \alpha_j k(x_i, x_j) \right\}. \tag{1.19}$$

By computing the derivative with respect to $\boldsymbol{\alpha}$, denoting by K the $m \times m$ matrix $\big(k(x_i, x_j)\big)_{i,j}$, and by Y the vector $(y_i)_{1 \leqslant i \leqslant m}$ we obtain:

$$2K(K\boldsymbol{\alpha} - Y) + 2\lambda K\boldsymbol{\alpha} = 0$$

which leads, since K is positive definite, to the linear system:

$$(K + \lambda \,\mathrm{Id})\, \boldsymbol{\alpha} = Y. \tag{1.20}$$

where Id is the identity matrix.

2.4 Support vectors

The kernelized examples in the previous section are able to deal linearly with the non-linear priors on the data (i.e., the kernel, which induces a feature space and a metric therein) and are consequently able to deal with far more general tasks than usual linear classification or regression. However the computation of the label to assign to a new test point involves its distances to all training points, and consequently these algorithms are naturally slow if the training set is big. Instead of using tricks to reduce the training set size or to avoid the computation of all distances for each new point, one can wonder whether there would exist another, similar approach, which would naturally and directly lead to a huge compression of the training data, keeping only a few meaningful training points to predict the labels of new test points. Such an approach does exist. We present here the fundaments of support vector classification.

Hyperplanar classifier in feature space and margin

We are given a set of points x_i with a binary label $y_i \in \{-1, 1\}$ and we would like to attribute to any new point $x \in \mathcal{X}$ a class label $f(x)$. We consider a kernel k and search for the best hyperplane in the feature space $\mathcal{H}$ which separates the training points $\mathbf{x}_i = \Phi(x_i)$ into two classes, so that f has the form:

$$f(x) = \operatorname{sgn}\left(\langle \mathbf{w}, \mathbf{x}_i \rangle_{\mathcal{H}} + b \right) \tag{1.21}$$

where $\mathbf{w} \in \mathcal{H}$ is a vector normal to the hyperplane and $b \in \mathbb{R}$ is the shift of the hyperplane. Let us rescale the problem by adding the constraint that the closest data point $\mathbf{x}_i$ to the hyperplane satisfies

$$|\langle \mathbf{w}, \mathbf{x}_i \rangle_{\mathcal{H}} + b| = 1. \tag{1.22}$$

Note that the margin, i.e. the distance between the hyperplane and the closest point, is then $1/\|\mathbf{w}\|_{\mathcal{H}}$. We would like the margin to be as large as possible in order to ensure the quality and the robustness of the classification (see figure 2). Therefore we would like to minimize $\|\mathbf{w}\|_{\mathcal{H}}$.

We would like also the predictions $f(x_i)$ on training points to be as good as possible. Since the labels are binary, i.e. $y_i \in \{-1, 1\}$, a correct labelling $f(x_i)$ of the point x_i means $y_i f(x_i) > 0$. Because of constraint (1.22), this is equivalent to:

$$\forall i, \quad y_i \big(\langle \mathbf{w}, \mathbf{x}_i \rangle_{\mathcal{H}} + b \big) \geqslant 1. \tag{1.23}$$

Soft margin

However, in practice, it may happen that the two classes overlap in the feature space and consequently cannot be separated by an hyperplane satisfying (1.23) for all examples i. Outliers may also be present in the training set and it may be better to relax the constraints (1.23) than to overfit the data. Let us denote by ξ_i non-negative slack variables, and relax (1.23) to:

$$\forall i, \quad y_i \big(\langle \mathbf{w}, \mathbf{x}_i \rangle_{\mathcal{H}} + b \big) \geqslant 1 - \xi_i. \tag{1.24}$$

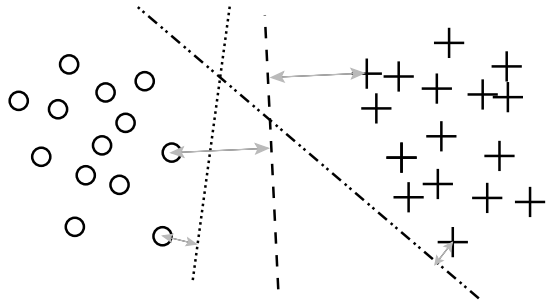

FIGURE 2. Example of a good and two bad hyperplane classifiers for a same training set. The larger the margin is, the better the classifier is likely to perform.

We would prefer the sum of the slacks $\sum_i \xi_i$ to be as small as possible, so we build a *soft margin classifier* by solving

$$\underset{\mathbf{w}\in\mathcal{H},b\in\mathbb{R},\boldsymbol{\xi}\in\mathbb{R}_+^m}{\text{minimize}} \qquad \tau(\mathbf{w},\boldsymbol{\xi}) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{m}\xi_i \tag{1.25}$$

$$\text{subject to} \qquad \forall i, \quad y_i(\langle \mathbf{w}, \mathbf{x}_i\rangle + b) - 1 + \xi_i \geq 0. \tag{1.26}$$

where the constant $C > 0$ determines the trade-off between margin maximization and training error minimization.

Lagrangian approach and dual problem
The *constrained optimization problem* (1.25,1.26) can be solved by introducing *Lagrangian multipliers* $\alpha_i \geqslant 0$ and a *Lagrangian*

$$L(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}) = \tau(\mathbf{w},\boldsymbol{\xi}) - \sum_{i=1}^{m}\alpha_i\big(y_i(\langle \mathbf{x}_i, \mathbf{w}\rangle + b) - 1 + \xi_i\big). \tag{1.27}$$

and by minimizing it with respect to $\mathbf{w}$, b and $\boldsymbol{\xi}$ while maximizing it with respect to $\boldsymbol{\alpha}$. This additional maximization is a practical way to enforce the constraints (1.26). Indeed, for given $\mathbf{w}$, b and $\boldsymbol{\xi}$, if one constraint i was violated in 1.27, then the corresponding $y_i(\langle \mathbf{x}_i, \mathbf{w}\rangle + b) - 1 + \xi_i$ would be negative, and thus maximizing L w.r.t. α_i would lead to infinity. Similarly, for given $\mathbf{w}$, b and $\boldsymbol{\xi}$, the α_i that maximize (1.27) are zero if the corresponding constraints are strictly satisfied (i.e. $y_i(\langle \mathbf{x}_i, \mathbf{w}\rangle + b) - 1 + \xi_i > 0$). This is essentially the Karush-Kuhn-Tucker (KKT) complementarity conditions of optimization theory. Consequently only a few α_i will be non-zero, leading to a sparse representation of the training data.

Maximizing L w.r.t. the primal variables $\mathbf{w}$, b and $\boldsymbol{\xi}$ leads to:

$$\frac{\partial L}{\partial \mathbf{w}} = 0 \;\text{ and }\; \frac{\partial L}{\partial b} = 0 \;\text{ and }\; \forall i,\; \frac{\partial L}{\partial \xi_i} = 0 \text{ or } \left\{ \xi_i = 0 \text{ and } \frac{\partial L}{\partial \xi_i} \geq 0 \right\} \quad (1.28)$$

which are respectively equivalent to

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i \quad (1.29)$$

$$\text{and } \sum_{i=1}^{m} \alpha_i y_i = 0 \;\text{ and }\; \forall i,\; \alpha_i = C \text{ or } \{\xi_i = 0 \text{ and } \alpha_i \leq C\}\,. \quad (1.30)$$

Incorporating (1.29, 1.30) into (1.27) makes $\mathbf{w}$, b and $\boldsymbol{\xi}$ vanish, and together with the kernel trick (1.4) we obtain

$$\underset{\boldsymbol{\alpha} \in \mathbb{R}^m}{\text{maximize}} \qquad W(\boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j k(x_i, x_j) \quad (1.31)$$

$$\text{subject to} \qquad \sum_{i=1}^{m} \alpha_i y_i = 0 \quad \text{and} \quad \forall i,\; 0 \leq \alpha_i \leq C. \quad (1.32)$$

Note that there is an alternative parametrization of SV classifiers where a free parameter ν is used instead of C, with ν asymptotically charaterizing the fraction of points falling into the margin, and the fraction of support vectors.

Support vector machine
Once the quadratic energy (1.31) in $\boldsymbol{\alpha}$ with linear constraints (1.32) has been maximized, equation (1.29) gives us an algorithm of the form (1.21) we were searching for:

$$f(x) = \operatorname{sgn}\,\Big(\sum_i \alpha_i y_i k(x_i, x) + b\Big) \quad (1.33)$$

with $\alpha_i = 0$ for most i. The few data points x_i which have a non-zero coefficient α_i are called *support vectors*. To compute the value of the threshold b, one uses equation (1.30) which states that for any support vector x_i with $\alpha_i < C$, the slack ξ_i is zero and consequently the constraint (1.24) becomes:

$$b = y_i - \sum_j \alpha_j k(x_j, x_i). \quad (1.34)$$

3 Application to Intermodality Image Prediction

As a medical application of the above methods, we look at intermodality image prediction, i.e. the task of predicting an image of a subject (for instance a Computed Tomography (CT) scan), from an image of the same subject in a different modality (here, a Magnetic Resonance (MR) scan), given a training set of corresponding MR-CT pairs from different patients.

3.1 The MR-CT issue

MR-based CT prediction is needed for example for attenuation correction in Positron Emission Tomography (PET). The 511 keV radiation in PET gets attenuated while passing through tissue, and correcting for this effect requires knowledge of the so-called attenuation map, which can be derived from a CT image. In modern scanners, CT images are therefore often acquired alongside the PET image, typically in combined PET/CT scanners. However, the CT scan involves important additional radiation exposure for the patient. Moreover, novel PET/MR scanners are not equipped with a CT scanner, and thus it is desirable to perform the attenuation correction based on the MR, by estimating a "pseudo" CT image from the MR.

MR and CT scanners detect different properties of the matter, and consequently there is no one-to-one correspondence between the intensities in the MR image and the CT intensities. In particular, bone and air both yield no signal in all standard MR sequences, whereas their intensities in CT images are on opposite ends of the scale. For this application, it is therefore crucial to distinguish bone from air, and the MR intensity alone contains no helpful information for this problem.

3.2 Atlas registration vs. patches

Atlas registration is the process of aligning a new image with a template image already segmented into bone, air and tissue regions. This yields a segmentation for the new image. The implicit assumption is that there exists a continuous one-to-one transformation between the new patient and the template, and that this transformation can be easily computed. In the case of medical scans, it turns out that these assumptions are not always satisfied, for instance pockets of gas in the abdominal region are unlikely to occur in the same number and shape for different patients. Even if the assumptions were satisfied, one may be trying to solve a problem more difficult than necessary by searching for a topology-preserving transformation.

Even a rough registration, which does not assume one-to-one correspondence between images, brings useful information since the location of a point in a scan is clearly correlated with the type of tissue which can be found at that point. This correlation is not always decisive enough to determine fully the tissue class, for instance when several scenarios can be thought of at a same place (abdomen or random pocket of gas), or when the registration lacks accuracy.

On the other hand, a patch-based approach would consist in extracting local information from a patch in the MR image centered on the pixel considered, and in classifying this pixel according to similar patches previously observed. This would not require any prior registration, would not assume a one-to-one correspondence between all MR scans and consequently would be able to deal with several possibilities of scenarios for the same location.

It would, in some way, build a prediction by picking parts from different examples. This approach is much more flexible than template registration. However it ignores the important information given by the location.

We proposed in [8] to make simultaneous use of both the local and global information given by patches and registration, respectively. We first estimate a rough registration of the test image to a template image, and call *normalized coordinates* the resulting new positions of pixels.

3.3 Image prediction using kernel methods

The key in working with kernel methods is in designing a kernel, or features, which are adapted to the application. In our case an input will be a pair $x_i = (p_i, c_i)$ of the local patch p_i and its normalized coordinates c_i; we define a similarity measure between inputs by

$$k(x_i, x_j) = \exp\Big(\frac{-\|p_i - p_j\|^2}{2\,\sigma^2_{\text{patch}}}\Big)\ \exp\Big(\frac{-\|c_i - c_j\|^2}{2\,\sigma^2_{\text{pos}}}\Big). \tag{1.35}$$

The parameters σ_{patch} and σ_{pos} involved express the weighting between the different information sources. Their optimal values can be determined by the standard technique of *cross-validation*: to estimate the relevance of any particular choice of $(\sigma_{\text{patch}}, \sigma_{\text{pos}})$, the training set is partitioned into n subsets, and each subset is used for testing the algorithm trained with these parameters on the remaining $n-1$ subsets. The sum of the losses of all subsets is the energy to minimize with respect to the parameters.

For our application, cross-validation typically yields optimal values for σ_{pos} that are far bigger than 1. This implies that registration errors of a few pixels will not affect the accuracy of our algorithm.

In the CT prediction problem, we may be interested in the classification of MR pixels into three classes, namely bone, air and tissue, because in first approximation there is a one-to-one correspondence between these classes and the CT values. We build three binary classifiers with SVM, one for each class against the two others, or more exactly we compute the quantity whose sign is checked in (1.33), and then return the class which achieves the greatest score. We show examples of results in figure 3.

3.4 Local learning

If the position is very informative, we can learn locally, i.e. cut the template image into regions and train independently a classifier/regression for each region. For brain images for example, intersubject variability is much smaller than for whole body images. Thus non-rigid registration between subjects is possible with only minor misalignments, and it is reasonable to compare patches only within a localized neighborhood. We use kernel ridge regression in order to take into account the variability of CT values.

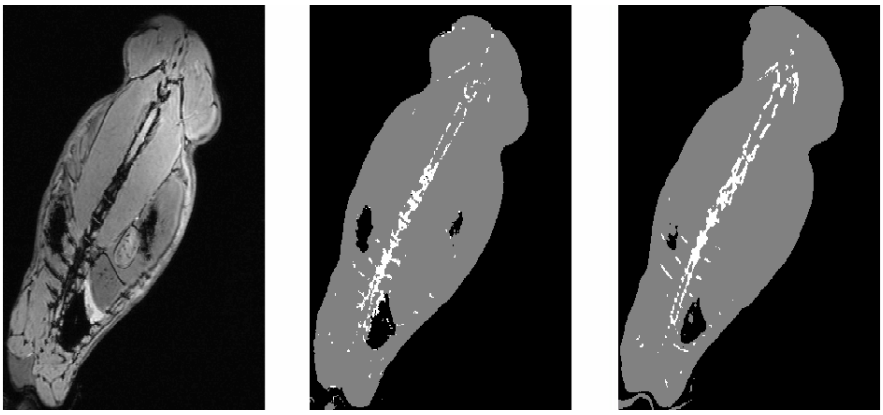

FIGURE 3. Left: MR (T2 MEDIC) of a rabbit. Middle: Three class labels as predicted using SVM. Right: Three class labels obtained by thresholding a CT image of the rabbit. Many differences between b) and c) are due not to false classifications, but to some slight movement between MR and CT scans, which explains the misalignment between the test image a) and the ground truth c).

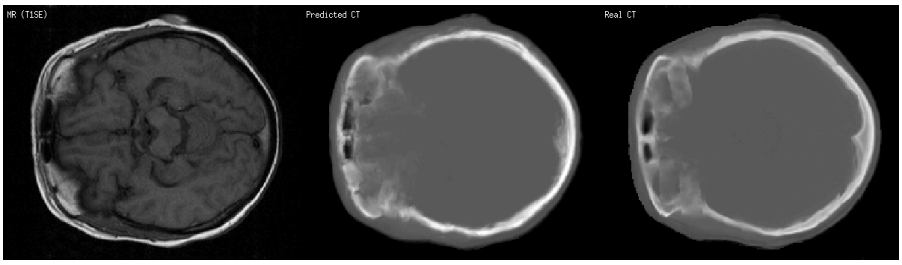

FIGURE 4. New MR image; pseudo-CT predicted by our method; ground truth.

More precisely, from pairs of patches and normalized coordinates, we do not predict the CT value itself but the variation between the CT value and the one in the template at that position. Results are shown in figure 4.

4 Discussion

After a tutorial on kernel methods, we have presented a way to use these machine learning tools to extract information from a set of medical images for MR-based CT prediction, in a framework which makes use of both local and global information. This presents the advantage of requiring neither a precise registration between template and test image, nor a one-to-one correspondence between them. We hope we have woken up the reader's enthusiasm for machine learning in medical imaging, there are still plenty of other ways to use machine learning tools in this field !

Bibliography

[1] M. A. Aizerman, É.. M. Braverman, and L. I. Rozonoér. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25:821–837, 1964.

[2] N. Aronszajn. Theory of reproducing kernels. *Transactions of the American Mathematical Society*, 68:337–404, 1950.

[3] C. Berg, J. P. R. Christensen, and P. Ressel. *Harmonic Analysis on Semigroups*. Springer-Verlag, New York, 1984.

[4] B. E. Boser, I. M. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In D. Haussler, editor, *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, pages 144–152, Pittsburgh, PA, July 1992. ACM Press.

[5] D. Cremers, T. Kohlberger, and C. Schnörr. Shape statistics in kernel space for variational image segmentation. *Pattern Recognition*, 36(9):1929–1943, 2003.

[6] J. Duchon. Spline minimizing rotation-invariant semi-norms in Sobolev spaces. In W. Schempp and K. Zeller, editors, *Constructive theory of functions os several variables, Lecture Notes in Mathematics, 571*. Springer-Verlag, Berlin, 1977.

[7] I. Guyon, B. Boser, and V. Vapnik. Automatic capacity tuning of very large VC-dimension classifiers. In S. J. Hanson, J. D. Cowan, and C. L. Giles, editors, *Advances in Neural Information Processing Systems*, volume 5, pages 147–155. Morgan Kaufmann, San Mateo, CA, 1993.

[8] M. Hofmann, F. Steinke, V. Scheel, G. Charpiat, M. Brady, B. Schölkopf, and B. Pichler. MR-based PET attenuation correction – method and validation. In *IEEE Medical Imaging Conference*, 2007.

[9] F. Jurie and C. Schmid. Scale-invariant shape features for recognition of object categories. In *International Conference on Computer Vision & Pattern Recognition*, volume II, pages 90–96, 2004.

[10] J. Mercer. Functions of positive and negative type and their connection with the theory of integral equations. *Philosophical Transactions of the Royal Society, London*, A 209:415–446, 1909.

[11] B. Schölkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond (Adaptive Computation and Machine Learning)*. The MIT Press, December 2001.

[12] B. Schölkopf, A. J. Smola, and K.-R. Müller. Kernel principal component analysis. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, pages 327–352. MIT Press, Cambridge, MA, 1999. Short version appeared in *Neural Computation* 10:1299–1319, 1998.