Reproducing Kernel Hilbert Spaces

Siim Erik Pugal
214704YAFB

December 2022

Introduction

A reproducing kernel Hilbert space (RKHS) is a mathematical concept that provides a way to represent and analyze functions in a high-dimensional space. It is a special type of Hilbert space, which is a vector space equipped with an inner product, that has a specific property called the reproducing property.

In a RKHS, the inner product of two functions can be computed by evaluating the functions at a single point, rather than integrating over the entire domain. This allows us to define and manipulate functions in a RKHS using techniques from linear algebra, such as projection and interpolation.

One of the main applications of RKHSs is in kernel methods, a class of non-parametric machine learning algorithms that can be used for tasks such as regression and classification. In kernel methods, the set of possible models is defined as a RKHS, and the goal is to find the function in this space that best fits the observed data. The key idea behind kernel methods is to use a kernel function to implicitly map the input data into a high-dimensional space, where it becomes easier to find a good model. The kernel function defines the inner product in the RKHS, and the choice of kernel can have a significant impact on the performance of the algorithm.

RKHSs are useful because they provide a way to represent and analyze functions in a high-dimensional space, even when the dimensionality of the input data is very large. They also allow us to use powerful tools from linear algebra to analyze and manipulate functions, which can be very useful in machine learning and other fields.

RKHS in modelization problems

In modelization problems, we aim to find the best function from a set of possible models, denoted as $\mathcal{M}$, that best fits a set of observations $(x_i, y_i)_i$ where x_i and y_i are elements of sets X and Y, respectively. We want the values of y_i to be similar to the output of the function $f(x_i)$, and for values of x that have not been observed, we want the output of the function $f(x)$ to be plausible. To determine the "best-fit" function, we need to define a measure of optimality, which is mathematically expressed as minimizing the following function:

$$\min_{f \in \mathcal{M}} L(\{x_i\}_i, \{y_i\}_i, f) + R(f)$$

The loss function L measures the difference between the predicted value $f(x_i)$ and the actual observation y_i. In regression problems, where $Y = \mathbb{R}^n$, this is often done using the sum of squared differences $|y_i - f(x_i)|^2_{\mathbb{R}^n}$. In classification problems, where $Y = 0, 1$, common loss functions include the hinge loss and the logistic loss.

The regularization function R measures the variations between the predictions $f(x_i)$ and $f(x_j)$. It helps prevent overfitting by promoting more homogenous predictions for points that are "close" together. Examples of regularization functions include the total variation norm, which is related to harmonic functions.

Optimization problems are often formulated in a convex setting to ensure both tractability and uniqueness of the solution. However, neural networks, which have a large number of parameters, require different optimization techniques such as stochastic gradient descent to find the optimal solution.

The choice of loss and regularization functions depends on the characteristics of the observations and the set of models $\mathcal{M}$. Models can be classified as parametrical, such as neural networks, or non-parametrical, such as kernel methods. In the theory of reproducing kernel Hilbert spaces, the set of models $\mathcal{M}$ is a Hilbert space of functions on X, equipped with a topology stronger than the pointwise topology.

Shortcomings

There are a few shortcomings of Reproducing Kernel Hilbert Spaces to consider:

- Computational complexity: RKHS methods can be computationally intensive, especially for large-scale problems. This can limit their practical use in certain situations.
- Kernel choice: Choosing the appropriate kernel function is important for the effectiveness of RKHS methods. There is often a trade-off between smoothness and fit to the data, and selecting the right kernel can be challenging.
- Limited expressiveness: RKHS methods are limited in their expressiveness, as they only consider linear combinations of the kernel functions. This can make them less effective for certain types of problems.
- Sensitivity to hyperparameters: RKHS methods can be sensitive to hyperparameters, such as the regularization parameter, and selecting the appropriate values can be challenging.
- Limited to Euclidean spaces: RKHS methods are typically only defined for Euclidean spaces, which can limit their applicability to certain types of data.

Overall, while RKHS methods can be effective for certain types of problems, they may not be the best choice in all situations. It is important to carefully consider the strengths and limitations of RKHS methods when deciding whether to use them for a particular problem.

Appearances of RKHS in mathematics

Some examples where RKHS are applied are:

- Hilbertian functional analysis and Sobolev spaces are a special case of kernel theory.
- Kernel operators, integral kernels, Green functions of partial differential equations all derive from the seminal work of Mercer and often use Bochner's representation theorem (1929).
- Gaussian processes are related through the Gram-covariance matrix to kernel theory. Stochastic processes with Brownian movement are thus close-by as well.
- Non-linear statistics such as kernel spectral theory or kernel principal component analysis (KPCA) are just a reformulation of classical linear statistics to a non-linear setting.

Hilbert Space and Kernel

Let X be a set deprived of any structure. A **kernel** on X is a function of $X \times X$ in $\mathbb{R}$.

Definition: Positive definite kernel in $\mathbb{R}$

A kernel $K : X \times X \to \mathbb{R}$ is called positive definite if it verifies:

- the symmetry property: $\forall x, y \in X,\ K(x, y) = K(y, x)$
- the positivity condition:

$$\forall (a_i)_{i=1..N} \in \mathbb{R}^N, (x_i)_{i=1..N} \in X^N,\ \sum_{i=1}^{N} \sum_{j=1}^{N} a_i a_j K(x_i, x_j) \geq 0$$

Such a kernel is a proto-scalar product over the set X: that is symmetrical and positive definite but not necessarily bilinear.

The Gram Matrix

Another important property that the RKHS makes use of is something called the Gram matrix. The Gram matrix is often used in the context of kernel methods because it allows us to encode the inner products of a set of vectors using a kernel function, which can be computationally more efficient than computing the inner products directly. It is also used to analyze the properties of the kernel function and the set of vectors.

For a set of vectors $v_1, v_2, ..., v_n$, the Gram matrix is an $n \times n$ matrix defined as:

$$K = \begin{bmatrix} k(v_1, v_1) & k(v_1, v_2) & ... & k(v_1, v_n) \\ k(v_2, v_1) & k(v_2, v_2) & ... & k(v_2, v_n) \\ ... & ... & ... & ... \\ k(v_n, v_1) & k(v_n, v_2) & ... & k(v_n, v_n) \end{bmatrix}$$

where $k(\cdot, \cdot)$ is a kernel function. The entries of the Gram matrix are the inner products of the vectors in the original set, computed using the kernel function.

Properties

Here are some properties of a kernel that are worth noting:

1. $k(x, x) \geq 0$. (Think about the Gram matrix of $n = 1$)
2. $k(u, v) \leq \sqrt{k(u, u)k(v, v)}$. (This is the Cauchy-Schwarz inequality.)

To see why the second property holds, we consider the case when $n = 2$:

Let $a = \begin{bmatrix} k(v, v) \\ -k(u, v) \end{bmatrix}$. The Gram matrix $K = \begin{bmatrix} k(u, u) & k(u, v) \\ k(v, u) & k(v, v) \end{bmatrix} \geq 0 \Leftrightarrow a'Ka \geq 0$

$\Leftrightarrow [k(v, v)k(u, u) - k(u, v)^2]k(v, v) \geq 0.$

Since we know that $k(v, v) \geq 0$, we have that $k(v, v)k(u, u) \geq k(u, v)^2$.

Examples:

- The scalar product $(\cdot,\cdot)_{\mathbb{R}^n}$ of $\mathbb{R}^n$ verifies the positivity condition which writes as:

$$\forall (a_i) \in \mathbb{R}^N,\ (x_i) \in X^N,$$

$$\sum_{i=1}^{N}\sum_{j=1}^{N} a_i a_j K(x_i,x_j) := \sum_{i=1}^{N}\sum_{j=1}^{N} a_i a_j (x_i,x_j)_{\mathbb{R}^n} = \left\| \sum_{i=1}^{N} a_i x_i \right\|_{\mathbb{R}^n}^2 \geq 0$$

- The Gaussian kernel over $\mathbb{R}^n$, $K(x,y) = \exp(-\frac{\|x-y\|_{\mathbb{R}^n}^2}{2\sigma^2})$, is positive definite.
- The Laplacian kernel over $\mathbb{R}^n$, $K(x,y) = \exp(-\lambda\|x-y\|_{\mathbb{R}^n})$, is positive definite.
- The Cauchy kernel over $\mathbb{R}^n$, $K(x,y) = \frac{1}{1+\frac{\|x-y\|_{\mathbb{R}^n}^2}{\sigma^2}}$, is positive definite.
- The minimum kernel over $[0,1]$, $K(x,y) = \min(x,y)$, is positive definite.
- The maximum kernel over $[0,1]$, $K(x,y) = \max(x,y)$, is not positive definite.
- The kernel over $]-1,1[$ defined by $K(x,y) = \frac{1}{1-xy}$, is positive definite.

Let H be a Hilbert space, suppose there exists a function $\Phi : X \to H$. Then the kernel on X defined by $K(x,y) := (\Phi(x),\Phi(y))_H$ is positive definite. If we can embed X in H, we get positive definite kernels!

More on optimization and modelization

Let's consider the modelization problem again, where the set of models is chosen to be H_K:

$$\min_{f\in H_K} L(\{x_i\}_i,\{y_i\}_i,f) + R(f)$$

There is a theorem called the **Regularization Theorem** that states that, under certain conditions, the optimal model in this case only depends on the $(K_{x_i}(\cdot))_{i\leq n}$ through the following relationship:

$$\exists (\alpha_i)_{i\leq n} \in \mathbb{R}^n \quad \overline{f}(\cdot) = \sum_{i=1}^{n} \alpha_i K_{x_i}(\cdot)$$

This theorem is useful because it shows that, when we penalize the norm $|f|_{H_K}$, there is no "additional information" in points that have not been seen. In fact, the smaller the RKHS norm, the smoother the function.

Since this theorem holds for a finite number of observations, we can work in a finite-dimensional space $(K_{x_i}(\cdot))_{i\leq n}$, where the scalar product is easy to compute using the kernel trick. This allows us to solve tractable problems such as Kernel Ridge Regression (KRR), a non-linear least squares method:

$$\min_{f\in H_K} \sum_{i=1}^{n} |y_i - f(x_i)|^2 + \lambda \|f\|_{H_K}^2$$

KRR is useful for non-linear approximation, but it is not the most famous kernel method. Support Vector Machines (SVMs) have played a significant role in the success of kernel theory.

Implementation of Kernel Ridge Regression (KRR)

In this section, we demonstrate an implementation of Kernel Ridge Regression (KRR) that illustrates the impact of regularization. We provide a matplotlib plot to show the effect of the regularization parameter λ.

Note that this solution is "non-parametrical" in the sense that we are not looking for a specific form of the function, such as $\overline{f}(\cdot) = A \cdot \cos(\omega t + \varphi)$. Instead, we simply want to find a function f that approximates the observed data points $y_i \simeq f(x_i)$ and can be generalized to other points close to the observed x_i.

To do this, we will use a Gaussian kernel with a variance of 5 times the distance between points on the x-axis. This will lead to an $H_\infty([0,1])$ approximation of the curve, where H_∞ is the Sobolev space of ∞ order (a completion of $(D)([0,1])$ and the Schwartz space of functions).

Declaration of initial variables

In this problem, we will approximate a cloud of points that roughly follows a sinusoidal curve, with some randomly generated points near the curve. To illustrate this example, we show a possible solution with the help of Python. The points will be randomly drawn from a normal (Gaussian) distribution. We will call this noise function $R \propto Norm(\mu, \sigma)$, where the mean is $\mu = 0$ and the standard deviation is $\sigma = 2$. The vector of scattered points is saved in $g(\vec{t}) = a_0 \sin\left(2\pi f_0 \vec{t}\right)$.

We will start with $N = 200$ points and set $\sigma = \frac{5.}{N}$ and $\vec{t} = \left[0., 0.005, 0.01, ..., 0.995\right]$. The main parameter we will change is λ, which we will initially set to $\lambda_0 = 0.02$ and later also $2 \cdot 10^{-10}$ for comparison.

```
N = 200
sigma_kernel= 5./N
t = np.arange(0.0, 1.0, 1./N)
mu, sigma_noise = 0, 2  # mean and standard deviation for noise
                        # signal generation R_noise =
                        # np.random.normal(mu, sigma_noise, t.size)
a0      = 5
f0      = 2
lmbda0  = 2E-2
g = a0*np.sin(2*np.pi*f0*t)+R_noise # basically a set of randomly
                                    # scattered points that some-
                                    # what follow a sinus wave
```

Solution

To get a better view of the independent matrix components and how we arrive at our final values I have dedicated this section here.

$$K = \exp\left(-\frac{d(x,y)}{\sigma^2}\right), \quad d(x,y) = \begin{bmatrix} a_{11} & a_{12} & ... & a_{1N} \\ a_{21} & a_{22} & ... & a_{2N} \\ ... & ... & ... & ... \\ a_{N1} & a_{N2} & ... & a_{NN} \end{bmatrix}$$

For our matrix values we shall construct pairwise distances between observations in N-dimensional space. This gives us the benefit of working with a symmetric matrix, meaning $a_{11} = a_{12} = ... = a_{NN} = a_1$, as well as $a_{12} = a_{21} = ... = a_2$ and $a_{13} = a_{31} = ... = a_3$ and so on.

$$K = \begin{bmatrix} \exp\left(-\frac{a_1}{\sigma^2}\right) & \exp\left(-\frac{a_2}{\sigma^2}\right) & \exp\left(-\frac{a_3}{\sigma^2}\right) & ... & 0. \\ \exp\left(-\frac{a_2}{\sigma^2}\right) & \exp\left(-\frac{a_1}{\sigma^2}\right) & \exp\left(-\frac{a_2}{\sigma^2}\right) & ... & 0. \\ \exp\left(-\frac{a_3}{\sigma^2}\right) & \exp\left(-\frac{a_2}{\sigma^2}\right) & \exp\left(-\frac{a_1}{\sigma^2}\right) & ... & 0. \\ ... & ... & ... & ... & ... \\ 0. & ... & \exp\left(-\frac{a_3}{\sigma^2}\right) & \exp\left(-\frac{a_2}{\sigma^2}\right) & \exp\left(-\frac{a_1}{\sigma^2}\right) \end{bmatrix}$$

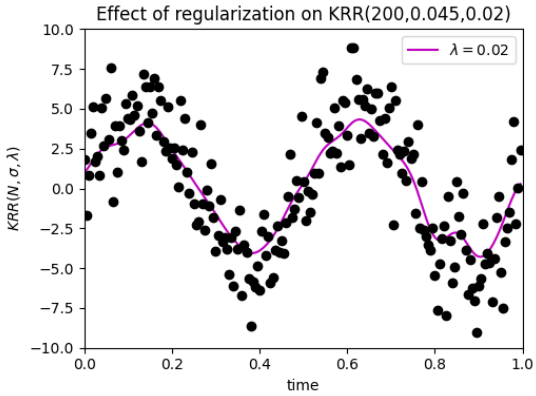
$$\boxed{S_{approx} = K \cdot \mathbf{u}}$$

$$(K + \lambda_0 \cdot N \cdot I)\mathbf{u} = g, \quad \mathbf{u} = (K + \lambda_0 \cdot N \cdot I)^{-1} g.$$

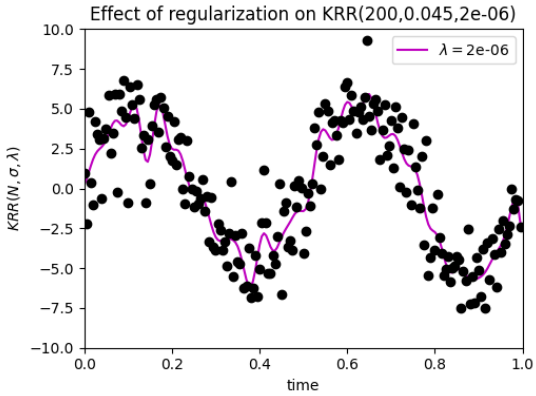
In Python, all of this is written like so:

```
from scipy.spatial.distance import pdist, squareform
pairwise_sq_dists = squareform(pdist(np.reshape(t,[-1,1]),
                    'sqeuclidean')) # a list of 200 list with
                                    # 200 elements inside one list
K = np.exp(-pairwise_sq_dists / sigma_kernel**2) # a sparce matrix
u = K + lmbda0*t.size*np.eye(t.size)
Sapprox = np.dot(K,np.linalg.solve(u, g)) # solution
```

So our final solution, that follows the could of points is stored in S_{approx}, the result of which can be easily plotted.



(a) Parameter λ equal to 0.02



(b) Parameter λ equal to $2 \cdot 10^{-6}$

Figures (a) and (b) show how different kernel values can approximate the data cloud. The magenta purple line is formed by the data points stored in the S_{approx} vector. In Figure

(a), λ is set to a relatively large value of 0.02, resulting in a smooth line. In contrast, in Figure (b) where $\lambda = 2 \cdot 10^{-6}$, the function is less smooth and follows the point cloud in a more complex manner, covering a larger area of the points.

Python code

The full Python code is shown here:

```
import numpy as np
import scipy as sp
import matplotlib.pyplot as plt
# all in one
def KRR(N, sig, lmbda):
    #N = 200
    sigma_kernel= sig/N
    t = np.arange(0.0, 1.0, 1./N)
    # mean and standard deviation for noise signal generation
    mu, sigma_noise = 0, 2
    R_noise = np.random.normal(mu, sigma_noise, t.size)
    a0      = 5
    f0      = 2
    g = a0*np.sin(2*np.pi*f0*t)+R_noise
    # A list of N list with N elements inside one list
    pairwise_sq_dists = sp.spatial.distance.squareform(
            sp.spatial.distance.pdist(np.reshape(t,[-1,1]),
            'sqeuclidean'))
    # A sparce matrix
    K = np.exp(-pairwise_sq_dists/sigma_kernel**2)
    Sapprox = np.dot(K,np.linalg.solve(K +
        lmbda*t.size*np.eye(t.size), g))
    fig, ax = plt.subplots()
    plt.subplots_adjust(left=0.25, bottom=0.25)
    # Plotting the results of Sapprox vector
    plt.plot(t, Sapprox,'m-', label ="$\lambda =$%s" % lmbda)
    plt.plot(t, g,'ko')     # plotting the initial noise
    plt.axis([0, 1, -10, 10])
    plt.title('Effect of regularization on KRR(%d,%s,%s)'
                % (N,sigma_kernel, lmbda))
    plt.ylabel('$KRR(N,\sigma,\lambda)$')
    plt.xlabel('time')
    leg = ax.legend(loc ="upper right")
    plt.show()

# Examples:
#KRR(200,9,0.02)
#KRR(200,9,0.000002)
```

Conclusions

In this brief introduction to Reproducing Kernel Hilbert Spaces, we have seen that from a machine learning perspective, it is a non-parametrical modelization technique that is deeply rooted in functional analysis and has applications in many fields. It allows us to transform non-linear problems in low dimensions into linear ones in (countably) infinite dimensions.

The regularization theorem states that the optimal model for a given set of observations only depends on the $(K_{x_i}(\cdot))_{i\leq n}$ through a sum of coefficients. This theorem allows us to work in a finite dimensional space and solve problems such as Kernel Ridge Regression, a non-linear least squares optimization problem, tractably. KRR is useful for non-linear approximation and is commonly used in conjunction with support vector machines. The smaller the norm of the Reproducing Kernel Hilbert Space, the smoother the function. Furthermore, the kernel trick allows us to easily compute the scalar product in RKHS.

RKHS theory is useful for linearizing non-linear problems of low dimension, but it can be computationally intensive for large-scale problems. Researchers are still working on ways to extend this approach to larger problems.

Kernels can be thought of as:

- an embedding $X \to H_K$ that allows for linear operations despite a non-vectorial X,
- a similarity measure between points of X similar to a topographic map,
- a class $\mathcal{M}$ of models that has excellent properties that allow us to compare models using the norm $|\cdot|_{H_K}$.