An overview of Reproducing Kernel Hilbert and Banach Spaces and the Kernel trick.

Alberto Acevedo

December 18, 2020

Contents

1 Intoduction

In an introductory machine learning course one typically encounters optimization problems of the following sort.

$$\min_{f\in V}\sum_{j\in N}\mathcal{L}(f(x_j),y_i)+\mu\|f\|_V^\lambda.$$

Here, V is some vector space we assume our minimizer to coincide in, $\mathcal{L}$ is the chosen loss function, f is the optimizer given the data $\{x_i,y_i\}_{i\in N}$, and N is the size of the data. The item $\mu\|f\|_V^\lambda$ is a penalty term where $\|.\|_V$ is the norm of the respective vector space while λ and μ function as penalty parameters. It is customary to let $\lambda=1$ or 2. When $\mu=0$ we find ourselves in the unpenalized model fitting case. In one of the simplest cases, $\mathcal{L}$ may be taken to be the RSS, $\mu=0$ and $V=\mathbb{R}^d$ where $d=dim(x_i)$, i.e. d is the dimension of the predictor vectors with the exclusion of the bias. The optimization problem in this this case is simply

$$\min_{\beta\in\mathbb{R}^d}\sum_{j\in N}\|y_j-\sum_{i=0}^{d}\beta_i x_{ji}\|_2.$$

Let us say the minimizer for the above is β^{OLS}, then this vector will minimize the error above and if the source of the data is indeed linear in nature then this model will serve in making good predictions. If we wan to be more stringent we might add a penalization like the ones mentioned above, two examples of this sort of regression are LASSO and Ridge Regressions. i.e.

$$\min_{\beta\in\mathbb{R}^d}\sum_{j\in N}\|y_j-\sum_{i=0}^{d}\beta_i x_{ji}\|_2+\mu\|\beta\|_1.$$

for the LASSO case and

$$\min_{\beta\in\mathbb{R}^d}\sum_{j\in N}\|y_j-\sum_{i=0}^{d}\beta_i x_{ji}\|_2+\mu\|\beta\|_2.$$

for the Ridge regression case. This is indeed the case were we have restricted ourselves to a linear model and all that is left to do is fit the model to the data by finding the optimal vector in $\mathbb{R}^d$. These types of models fall under the category of *parametric* models. Indeed the effectiveness of our optimization will be limited by the size and structure of the space V we are optimizing over. Let us minimize the RSS problem above over an infinite dimensional space as an example. Assuming that $V=\mathcal{H}$, some infinite dimensional *Hilbert* space, the optimization problem may be recast as follows.

$$\min_{f\in\mathcal{H}}\sum_{j\in N}\mathcal{L}_{RSS}(f(x_j),y_i).$$

This is the archetypical non-parametric machine learning optimization problem. In this case we are making no assumptions about the structure of the predictor function, i.e. f, except for the fact that we are constraining it to live in a particular *Hilbert* space $\mathcal{H}$. If the *Hilbert* space is picked adequately we may find a better fitting model that outperforms the simple linear model.

There is an issue with non-parametric learning however. In general it is not as parsimonious as we would like. To appreciate this consider some set of data $\{x_i, y_i\}_{i\in N}$ which comes from some natural process exhibiting a linear relationship. i.e.

$$y_i = \sum_j \hat{\beta}_j x_{ij} + \epsilon.$$

Where ϵ is a small parameter of noise and the $\hat{\beta}_j$ are taken to be the ground truth parameters. If we use the usual RSS scheme we can attain a robust approximation in a parsimonious manner since we need only execute the following computation, due to the Gauss-Markov theorem, given our data.

$$\hat{\beta} = (X^T X)^{-1} X^T y.$$

Where X is the design matrix and y is the out put vector of our data. In the non-parametric case let us say that we use some *Hilbert* space of polynomials not knowing before hand that the data follows an underlying linear ground truth. For visual simplicity let us reduce dimensionality to the one predictor case. Regardless of predictor space dimension we would have to explore a vast infinite dimensional *Hilbert* space ultimately yielding a model of the form

$$f(x) = \sum_i c_i p_i(x)$$

where the c_i scale the importance of the respective polynomial p_i corresponding to some *Hilbert* space. Of course we would expect the linear term in this *Hilbert* space to carry the largest weight and additional non-linear terms to deviate our model from the ground truth. Since is not optimal to deduce an infinite quantity of parameters c_i, not parsimonious, and we are therefore left with the task of finding a truncation point which in general is not a trivial procedure. In this short paper we present two types of vector spaces that mitigate the usage of non-parametric techniques, yielding parsimoniousness in both cases. These spaces are denominated as Reproducing Kernel Hilbert Spaces, RKHS,and the more general Reproducing Kernel Banach Spaces, RKBS. We will first talk about the utility of RKHS spaces, contrast such *Hilbert* spaces with regular *Hilbert* spaces and then present some examples. We will then briefly touch on RKBS, introducing the via a contrast to RKHSs.

2 Reproducing Kernel Hilbert Spaces.

2.1 A definition

Let us first formally define an RKHS.

Definition 1. *RKHS: Let X be an arbitrary set and $\mathcal{H}$ a Hilbert space of real-valued functions on X. The evaluation functional over the Hilbert space of functions $\mathcal{H}$ is a linear functional that evaluates each function at a point x,*

$$L_x : f \to f(x) \forall f \in \mathcal{H}.$$

We say that $\mathcal{H}$ is a reproducing kernel Hilbert space if, for all $x \in X$, L_x is continuous at any $f \in \mathcal{H}$.

This definition is just the standard definition given in textbooks and sites such as "wikipedia". The requirement of continuity in this definition is the key feature since it allows us to make use of the *Rieze* representation theorem. i.e. for any functional l in the dual space of some *Hilbert* space $\mathcal{H}$ there exist a unique $y \in \mathcal{H}$ such that for all $x \in \mathcal{H}$ we have

$$l(x) = \langle y, x \rangle.$$

Using the the notation from the definition above, and the Daume paper[2] I use as part of my resources, this means that there is a unique function $K_x \in \mathcal{H}$ such that $\forall f \in \mathcal{H}$,

$$L_x(f) = f(x) = \langle f, K_x \rangle_{\mathcal{H}}$$

.

This is called the *Reproducing Property.*

2.2 The Reproducing Kernel.

The fact that every evaluation functional has the *Reproducing Property* means that if we start of with a functional L_y and evaluate the term term $K_x \in \mathcal{H}$ at y we have

$$L_y(K_x) = K_x(y) = \langle K_x, K_y \rangle_{\mathcal{H}} := K(x, y).$$

This is the *Reproducing Kernel*, it is denominated as such due to the fact that the *Reproducing Property* was executed twice in order to attain it. Before we discuss its useful features let us formally define it below.

Definition 2. *Reproducing Kernel: :Let X be an arbitrary set, and $\mathcal{H}$ a Hilbert space of all functions $f : X \to \mathbb{R}$. It, for all $x \in X$, the linear evaluation functional $L_x : \mathcal{H} \to \mathbb{R}$ is continuous at every $f \in \mathcal{H}$, we can construct the Reproducing Kernel, which is a bivariate function $K : X \times X \to \mathbb{R}$ defined by*

$$K(x, y) = \langle K_x, K_y \rangle_{\mathcal{H}}$$

and the Hilbert space $\mathcal{H}$ is called a Reproducing Kernel Hilbert Space (RKHS).

Indeed an RKHS is simply a Hilbert space with additional structure making it a subset of the vector spaces denominated Hilbert spaces, this is elucidated in figure 1.

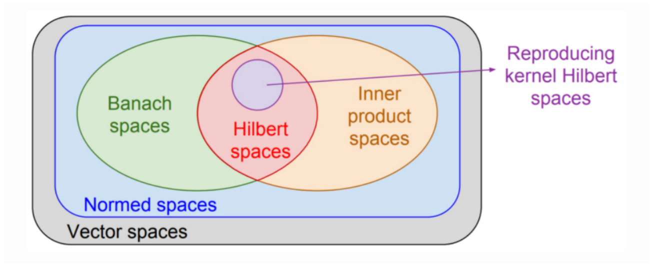


Figure 1: A diagram of the relationships between various vector spaces including RKHS. The image is obtained from the helpful online document on RKHS that may be found at the following site https://ngilshie.github.io/jekyll/update/2018/02/01/RKHS.html

2.3 The Kernel trick.

To make use of these RKHS we must have a linking map between the corresponding *Hilbert* space $\mathcal{H}$ and and the attribute space X. Such a map is usually called a *feature* map, we label such maps ϕ.

$$\phi : X \to \mathcal{H}.$$

Such mappings may be accomplished by exploiting the *Reproducing Property*. i.e. we let $\phi(x) = K_x$ for all $x \in X$. To elucidate on the practicality of these RKHS let us turn to an example. A prime application ground for RKHS may be found in the SVM machine learning schemes. Consider the following data, seen in figure 2, which we are to classify.

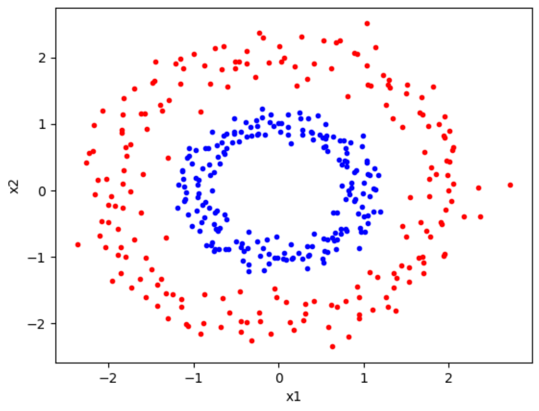


Figure 2: I have undertaken the usual 2 concentric circles data problem.This data was generated by adapting the code provided in https://ngilshie.github.io/jekyll/update/2018/02/01/RKHS.html.

It is clear that a one dimensional hyperplane here will not be capable of dividing the blue data points from the red data points regardless of the level non-linearity that is embraced. To tackle such problems one may map the data to the so called feature space. To choose a proper feature map we may first ask ourselves what property the red points have that make them distinguishable from the blue points other than their labels. One may quickly deduce that the red points are indeed further away from the origin of than the blue points. We may therefore include the euclidean distance as a component of the feature map. i.e.

$$\phi(x) = [x_1, x_2, -x_1^2 - x_2^2]^T.$$

I have chosen the negative of the euclidean distance in order that the blue dots find themselves yielding larger third components of the feature map than the red points. We in a sense lift our analysis to a higher dimension in order to make hyperplane fitting techniques efficient. The data in the lifted space now looks as seen in figure 3. We fit a classification plane to the data using SVM, it is a success!!

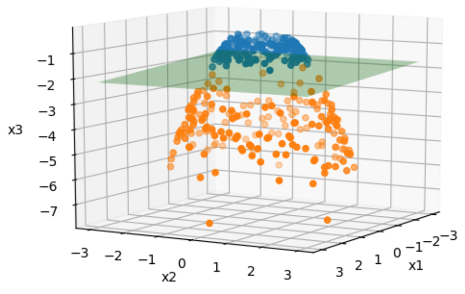


Figure 3: Classification using SVM in feature space. We execute SVM using sklearn in python by adapting the code provided in https://ngilshie.github.io/jekyll/update/2018/02/01/RKHS.html.

Now, to fully appreciate how the RKHS is being used here let us take a look at the SVM optimization problem. Its dual optimization problem for fitting a hyperplane to the data $\{x_i, y_i\}_{i=1}^N$ is the following.

$$\max_{\bar{\alpha}}\{\sum_i \alpha_i - \frac{1}{2}\sum_i\sum_j y_i y_j \alpha_i \alpha_j \langle x_i, x_j\rangle_X\}$$

subject to the constraint s

$$\alpha_i \geq 0 \forall i \in \{1, ..., N\}$$

and

$$\sum_i \alpha_i y_i = 0.$$

The new optimization problem after being mapped to feature space will resemble the optimization problem above with the inner products in the maximizing term being swapped for the corresponding *Reproducing Kernel* terms. i.e. instead of computing $\langle x_i, x_j\rangle$ we now compute

$$K(x_i, x_j) = \langle \phi(x_i), \phi(x_j)\rangle_{\mathcal{H}}.$$

Where $\mathcal{H}$ is the *Hilbert* space associated with the feature space. Notice that for the case of the concentric circles the feature map that we chose yields the kernel

$$\langle \phi(x), \phi(y) \rangle_{\mathbb{R}^3} = \langle x, y \rangle_{\mathbb{R}^2} + \|x\|_2^2 \|y\|_2^2 = K(x, y).$$

This does not seem like much at first glance but we now have a means to implicitly compute the inner product terms without having to compute the feature vectors themselves. It is a fact that if we see it fit to assume the existence of some kernel, underlying the classification of the data, there will always be some *Reproducing Hilbert* space which pertains to such a kernel. This is indeed why these kernel methods are so powerful. This fact is a result of the $Moore - Aronszajn$ theorem which says that every positive semi-definite kernel is a *Reproducing Kernel* for some corresponding *Reproducing Kernel Hilbert* space. In this lower dimensional setting the rewards are not fully appreciable so it is a good idea to do a more complicated example.

2.4 More computational examples

Consider the erratic data presented in figure 4. Unlike the example in the previous subsection there is no clear evidence of what the proper feature map to use in this case should be. We may therefore go straight to guessing what a good kernel might be. If we make the guess that the kernel is of the linear type then the SVM scheme yields the classification seen in figure 5 with an accuracy of about 54%. While using a radial basis function kernel(RBF) yields a classification accuracy of 86% These later results are seen in figure 5.

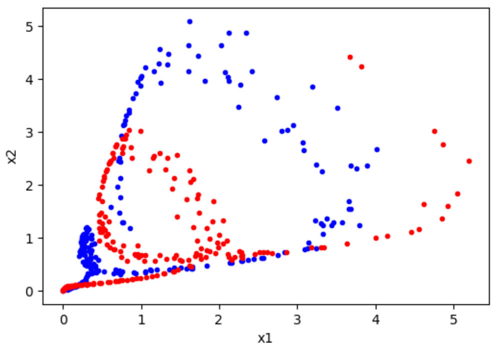
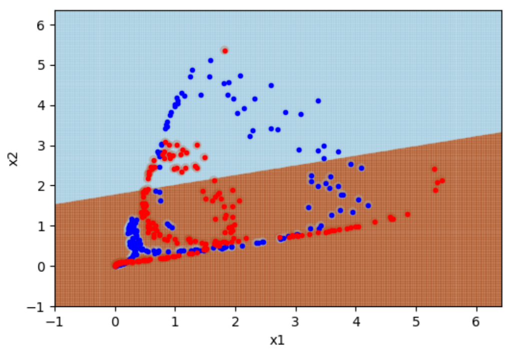


Figure 4: Here we generate some erratic data using simple functions with python. The details are included in my python file. Note that it is hopeless to find some linear hyperplane that robustly classifies data emerging from this mechanism.

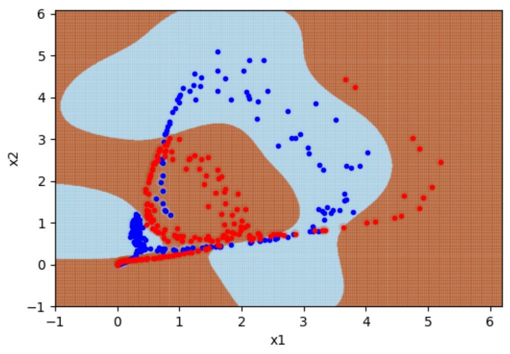
In the case were we use the so called radial basis function kernel, we are using an object of the following form

$$K(x, y) = e^{-\frac{\|x-y\|^2}{2\sigma^2}}.$$

We will stick to a value of $\sigma = 1$ consistently. Indeed due to the $Moor - Aronszajn$ theorem this to is a *Reproducing Kernel*. It turns out that the feature space, *Hilbert* space, associated with such a kernel is infinite dimensional. This means that the corresponding map



(a) Linear Kernel, classification accuracy 54 percent.



(b) RBF Kernel, classification accuracy 86 percent.

Figure 5: A comparison of employing linear kernels vs RBF kernels.

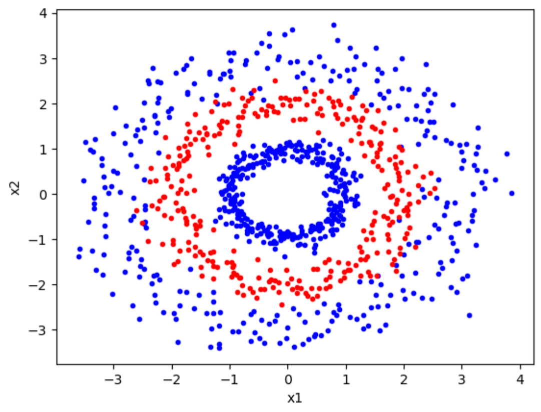
takes the data as input and outputs an infinite dimensional vector. However, thanks to the fact that the associated feature space is a *Reproducing Kernel Hilbert Space* we need only compute the kernel above at the different data points. This allows us to tackle highly non-linear classifications problems with very little linear cost. This is what the kernel "trick" is all about and why it is so popular, it is computationally cheap and robust. Finding the correct kernel is another story best saved for another paper.

2.5 Returning to the concentric circles problem

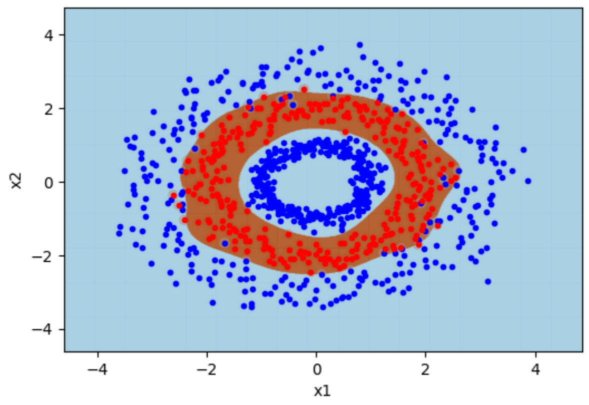
At the beginning of this section we introduced the concentric circle data classification problem. With only two circles this problem is easily solved by using a linear kernel. However, if there are more than two concentric circles this problem indeed needs a higher dimensional feature space in order for us to effectively use SVM. Instead of worrying about what this feature space might look like let us simply make use of the versatilely *RBF* kernel used in the previous subsection and see what happens. i.e.

$$K(x, y) = e^{-\frac{\|x-y\|^2}{2\sigma^2}}.$$

As can be seen in figure 6, this kernel method is indeed very effective, yielding 97% classification accuracy.



(a) Noisy concentric circles data. Generation of data and details may be found in my python files.



(b) Classification using SVM with a RBF kernel. Classification accuracy 97 percent.

Figure 6: Noisy concentric circles binary classification.

3 Reproducing Kernel Banach Spaces.

3.1 A definition

There is a second type of reproducing kernel space that is very practical in nature, namely the *Reproducing Kernel Banach Spaces.* They differentiate themselves from their *Hilbert* space counter parts simply from the fact that the respective kernel need not be generated from an inner product of feature maps but may actually be generated by norms distances of feature maps. This relaxes a rather stringent constraint and opens the door to new possible selections for a *Reproducing Kernel.* I will state the formal definition below but rather than spend too much time on the theoretical details provided in [3] one my goals for this research project became implementing SVM with a *Reproducing Kernel* obtained from a *Banach* space. Here is the definition.

Definition 3. *Reproducing Kernel Banach Space: A RKBS on a set X is a reflexive Banach space $\mathcal{B}$ of functions on X for which the dual of $\mathcal{B}$ is isometric to some Banach space $\mathcal{B}^{\#}$ of functions on X and the point evaluation is continuous on both $\mathcal{B}$ and $\mathcal{B}^{\#}$.*

This definition has been taken directly out of Zhang's and Xu's shared paper [3], and although rather technical, this definition is essentially analogous to that of RKHS but with substitution of the inner products with norms, as already mentioned.

The key difference between working with *Banach* spaces as opposed to *Hilbert* spaces is that once the dimension of the space is fixed there is indeed only one *Hilbert* space since they are isometrically isomorphic to each other. This however is not the case with *Banach* spaces. Indeed, compared to *Hilbert* spaces, *Banach* spaces posses much richer geometric structures which might be potentially useful for learning algorithms. In what follows let us analyse a key example of a *Reproducing Kernel* obtained form a *Banach* space.

Let the $X = \mathbb{R}$ with $\mathbb{I} = [-\frac{1}{2}, \frac{1}{2}]$ and $p \in (1, +\infty)$ with q the conjugate number of p. Now, let

$$\phi(x,t) = e^{-i2\pi xt}.$$

For $f \in L^1(\mathbb{R})$, the Fourier transform is just

$$\hat{f}(t) = \int_{\mathbb{R}} f(x)\phi(x,t)dx.$$

The inverse Fourier transform is of course

$$\hat{f}(t) = \int_{\mathbb{R}} f(x)\phi^*(x,t)dx.$$

Now, following the work of Zhang et al [3] we see that the *Banach* space

$$\mathcal{B} = \{f \in C(\mathbb{R} : supp\hat{f} \subset \mathbb{I}, \hat{f} \in L^p(\mathbb{I})\}$$

is the correct underlying *Banach* space here, with norm

$$\|f\|_{\mathcal{B}} := \|\hat{f}\|_{L^p(\mathbb{I})}$$

.

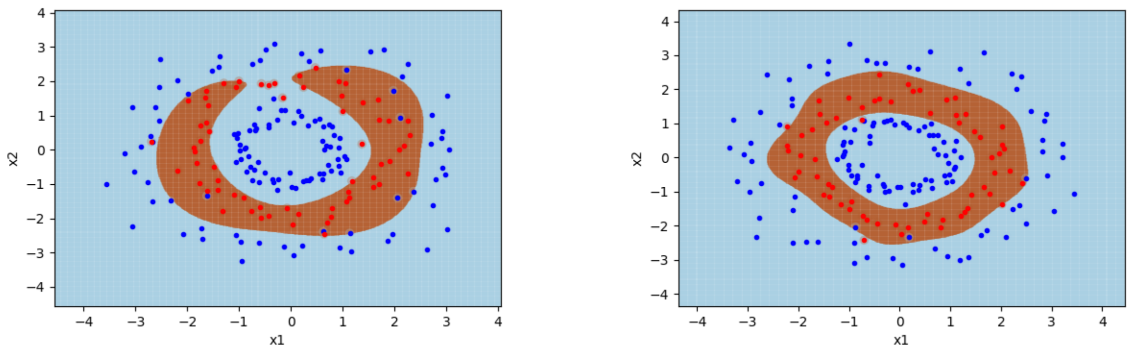
The kernel in this case turns out to be simply

$$K(x,y) = \langle \phi(x), \phi^*(y)\rangle_{L^p(\mathbb{I})} = \int_{\mathbb{I}} e^{-i2\pi xt} e^{-i2\pi yt} dt = sinc(x-y).$$

Here is a nice kernel that we may try to use for classification problems, in what follows we will try to adapt the *sinc* kernel to higher dimensions and implement it.

3.2 Implementing the sinc kernel in place of the RBF kernel.

To finish our excursion, let us take the *RBF* and *sinc* kernels and tackle the concentric circle data classification already encountered multiple times during this paper. Due to some technical difficulties I had to reduce the data size. The reader may find the results in figure 7. Indeed we have refreshed the data for each of the iterations but these slight variations do not affect very much the overall performance. This code was run multiple times and in general the *RBF* kernel outperformed the *sinc* kernel by about the same amount precented in figure 7. The interesting thing to note is that the *sinc* kernel having come from an RKBS is still quite effective. In general, when working with hyperplane fitting or any other machine learning mechanism that employs kernel techniques one should keep in mind that RKBS have viable kernels.



(a) Classification using the *sinc* kernel. Accuracy of about 91 percent.

(b) Classification using the *RBF* kernel. Accuracy of about 97 percent.

Figure 7: Noisy concentric circles binary classification. With two different kernels.

4 Conclusion

We have shown in this short document not only that the kernel trick is effective at lowering computational cost but it is rather powerful at tackling non linear problems of hyperplane fitting via linear means, using SVM for example, by implicitly selecting a feature space where such a linear hyperplane fitting problem is feasible. This implicit selection is done via the selection of some appropriate *Reproducing Kernel*. We also learned that these kernels need not only come from inner products of feature maps but they may also come from norm distances of feature maps. This generalizes the space from which we may select possible kernels to include other richer and widely varying maps afforded by the more general *Banach* spaces. The question of whether it is best to use a RKHS or a RKBS is one that comes to mind but an answer to such a question depends on the problem at hand. The general rule of thumb is to only increase the size of the space you are minimizing over if it does not sacrifice parsimoniousness. Indeed opening up the possibility to RKBS takes us into larger spaces and in some instances this might not be very economical.

References

[1] Unknown *"Reproducing Kernel Hilbert Spaces and Machine Learning"*, (My Data Science Blog, 2018).

[2] Hal Daume III *From Zero to Reproducing Kernel Hilbert Spaces in Twelve Pages or Less*, (http://users.umiacs.umd.edu/~hal/docs/daume04rkhs.pdf), 2004).

[3] Haizhang Zhang, Yuesheng Xu, Jun Zhang. *Reproducing Kernel Banach Spaces for Machine Learning.* , (Journal of Machine Learning Research 10 (2009) 2741-2775).