**Kernel Tricks**

**Shashank Bharat**

*College of Business, Student of Master of Science in Business Analytics, California State University, East Bay*

*Abstract* – Kernel tricks are a class of algorithms used for pattern analysis. They are a well-established tool that analyze the relationship between output of a function and the corresponding input data. Using a kernel function, we can study and find general types of relations like classifications, correlations, principal components, rankings and clusters. In machine learning, the kernel tricks are broadly used in the Support Vector Machines to bridge linearity and non-linearity. This paper talks about various Kernel tricks that are used in most common in order to reduce the amount of computation required by the Support Vector Machines. This paper also discusses the application of Kernel tricks in areas other than Support Vector Machines.

Keywords – Kernel, Support Vector Machines, Hyperplane, Kernel Functions, Polynomial Kernel, Radial Kernel, Cross-Validation, Support Vector Classifier, Regression, Deep Learning, Text Recognition

# Introduction

## The Kernels

In order to make the data linearly separable, we need linearly separable input vectors so that we can train the linear algorithms on it. Kernels do this by automatically linearly separating the input data. We need a mapping (ⱷ) to remap the input space into a Feature space F, which is defined as:

This Feature space F is the transformation into a higher order feature vector, called a feature map

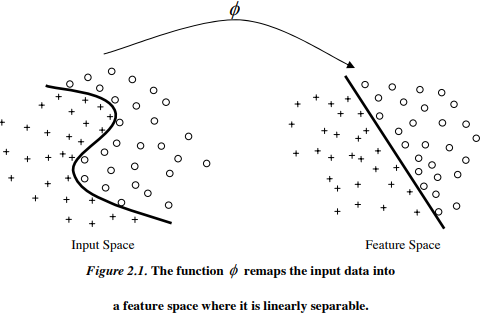


Image Source: [1]

A kernel is a function k that for all satisfies

A kernel is a mapping from is given by where

A kernel is a function that calculates how identical two vectors are. As an example, consider a second-dimensional input space and a feature mapping

For this feature mapping, we compute the feature function following way-

= {, }

=

## Some Basics of Support Vector Machines

The Support Vector Machine (SVM) is a supervised machine learning model that is largely used for classifications. It gains an understanding of how to segregate distinct groups by establishing decision boundaries.

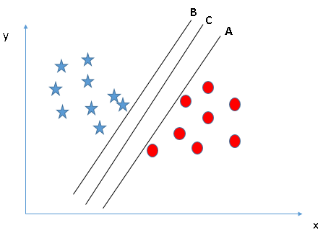
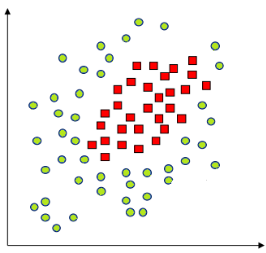


Image Source: [2]

Any subspace in 3 or more dimensions is called a hyperplane. In the figure above, we observed that there are two classes of observations: the blue stars and the red circles points. There is an umpteen number of ways to segregate these two classes as shown in the graph above. However, we want to discover the finest hyperplane that could maximize the margin between these two distinct classes. This means that we want the distance between the hyperplane and its nearest data points on each side to be the largest. It sounds straightforward in the example above. However, not all data or classes are easily linearly separable. In fact, almost all the data are randomly distributed in the real world, which makes it hard enough to segregate different classes linearly.





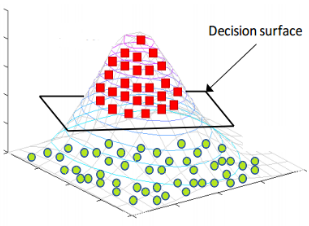


Image Source: [3]

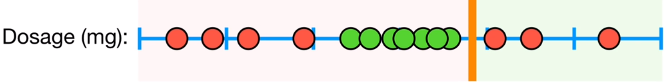
As we can see in the above figure, if we find a technique to map the data from 2-dimensional space to 3-dimensional space, we can then find a decision surface that distinctly divides between dissimilar classes. However, when there are a high number of dimensions, computations within that space become very expensive. This is the time when kernel trick jumps in. It permits us to operate in the native feature space without computing the coordinates of the data in a higher-dimensional space.

Please go through the references [3], [4], [5] to get more knowledge on the basics of kernels and Support vector machines.

## A Medicine Dosage Example



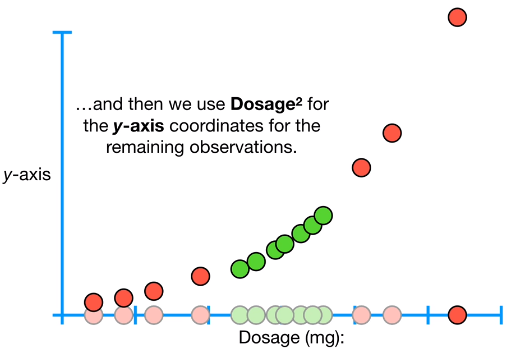
The above example represents the medicine dosage given to patients. Red dots represent that patients were not cured while green dots represent that patients were cured. In other words, the drug only works when given appropriately (representing the green dots).



In this case, no matter where we put the classifier, we’ll make a lot of misclassifications.

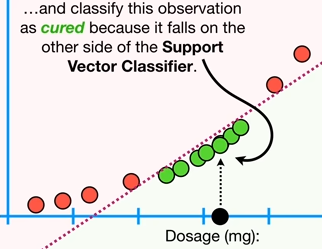
# The polynomial kernel

So, we put a Dosage on the x-axis, and the y-axis would be the square of each dosage dot. E.g. if the x-axis has a value of 0.5, the corresponding y-axis value would be (0.52=0.25). We do it for all values on the x-axis, hence, we get the following graph. Since each observation has now both x and y coordinates, the data are now 2-dimensional.

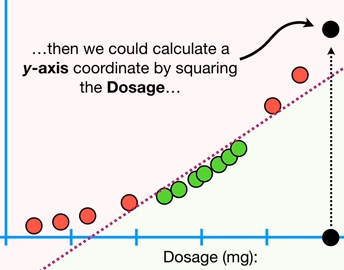


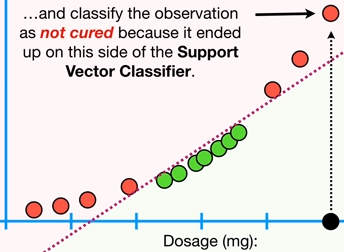
Now, we can draw a support vector classifier that separates the people who were cured (green dots) from the people who were not cured (red dots). Therefore, the support vector classifier can be used to classify the new observations. The black dots in the following figures show the value of the dosage on the x-coordinate and its corresponding value of dosage2 on the y-coordinate. Since the y-coordinate of the black dot falls on the lower side of the support vector classifier, we classify this observation as “cured”.

## 



In the following figures, since the value of the y-coordinate is on the upper side of the support vector classifier, we classify this observation as “not cured”.





Now, going back to the 1-dimensional data, why we created the y-axis coordinate as dosage2, and why not dosage3 or dosage4? How do we decide on how to transform the data? In order to make calculations possible, Support Vector Machines use **Kernel Functions** to systematically detect Support Vector Classifiers in higher dimensions.

In the above example, we used the **Polynomial Kernel** which has a parameter, d, which stands for the degree of the polynomial. When d = 1, the Polynomial Kernel computes the relationships between each pair of observations in 1-Dimension. These relationships are then used to find the support vector classifier. When d=2, we get a 2nd dimension based on Dosages2 and the Polynomial Kernel computes the 2-Dimensional relationships between each pair of the observations which are again used to find a support vector classifier. As the value of d increases to 3 and more, we get more dimensions to find a support vector classifier. We can find the optimal value for d using the **Cross-Validation** [6]. Cross-validation is a process in which we train the model using the subset of the data set and then evaluate using the corresponding subset of the data set. In summary, the Polynomial Kernel systematically increases the dimensions by the degree of the polynomial, and the relationships between each pair of observations are used to find a Support Vector Classifier. At last, we can find good value for d using cross-validation.

The Polynomial Kernel that we used in the above example is of the form- **(a . b + r)d**, a and b refers to two different observations we want to calculate the high dimensional relationship for in the dataset, r determines the coefficient of the polynomial, d sets the degree of the polynomial. In our example, we assume r = ½, and d = 2. Our Polynomial Kernel now has the form:

(a.b + ½)2 = (a.b + ½) (a.b + ½) = ab + a2b2 + 1/4.

This equation is equal to this dot product: (a, a2, ½). (b, b2, ½). This dot product gives us the high-dimensional coordinates for the data. The parameters (a and b) are x-axis coordinates, parameters (a2 and b2) are y-axis coordinates and parameters (1/2,1/2) are z-axis coordinates. As the z-axis coordinates are constant, we can ignore them. Coefficient r, and degree d, are determined using the cross-validation. Thus, we have x and y-axis coordinates for the data in a higher dimension. If we want to know the high dimensional relationships between two observations, let’s say Dosage 9 and 14, then:

**(a.b + r) d** =(9.14 + ½)2 = (126 + 1/2)2 = 126.52 = 16002.25

16002.25 is one of the 2-dimensional relationships that we need to solve for the Support Vector Classifier, even though we didn’t transform the data to 2-dimensions.

# DEGREE-K POLYNOMIAL KERNEL

Let’s take two feature vectors. The corresponding Order-2 polynomial will look as below. A new coefficient is added to the term . Same feature mapping is done for where it changes from 2-dimensional form to a 5-dimensional form.

Order-2 polynomial: ]

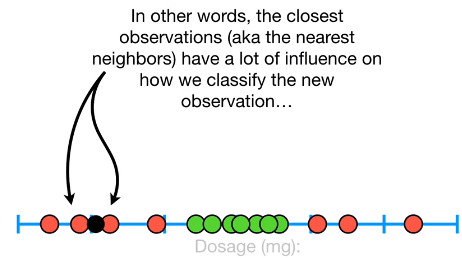
Kernel function

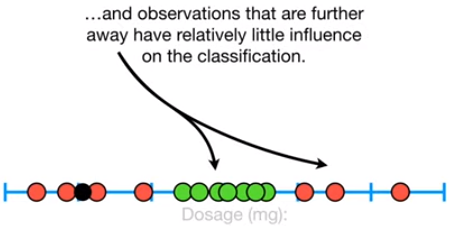
As can be seen above, the dot product of the two feature vectors is computationally very easy to compute. Hence, what matters is the dot product of the original feature vectors, and we do not need to find the feature mapping explicitly which is in 5-dimensions. This method is called the Kernel Trick and the same is shown above by the Kernel Function. The coefficient doesn’t matter as it is just a scalar number. Similarly, Degree-K Polynomial Kernel can be written as:

Please go through the [7], [8], [9] reference links in order to read more about *“The Polynomial Kernel”*

# the radial kernel

Another very commonly used Kernel is the **Radial Kernel**, also known as Radial Basis Function (RBF) Kernel or the **Gaussian Kernel**. The Radial Kernel behaves like a Weighted Nearest Neighbor model. In simple terms, the nearest neighbors have a lot of influence on how we classify the new observation than the neighbors that are far away relatively. SVM using the Gaussian Kernel works similar as the K nearest neighbor classification. The nearest is the shortest distance that is calculated between the support vectors and the new data points while in kNN its between training all instances and the new data. SVM learns which support vectors to choose and the final number is usually much smaller in comparison to size of the training set.





So, in the figure above, the Radial Kernel uses the classification of the two nearest neighbors shown for the new observation. The Radial Kernel is of the form:

. exp

is a symmetric function such that for any n number of data points, the matrix ,] is positive semi-definite.

The Radial Kernel finds support vector classifiers in infinite dimensions. Let’s talk about how the Radial Kernel determines how much influence each observation in the training dataset has on classifying new observations. For simplification, Radial Kernel can be expressed as . Just like with the Polynomial Kernel, a and b refer to two different Dosage measurements. The difference between the measurements is then squared, giving us the squared distance between the two observations. γ (gamma), which is determined by the cross-validation, scales the squared distance, and thus, it scales the influence. For example, if we consider two nearest observations like a = 2.5 and b = 4 and take γ = 1 in the Radial Kernel expression, we get the value 0.11. For γ = 2, we get 0.01. Now, if we determine how much influence the two observations have when they are relatively far from each other. Let’s take a = 2.5 and b = 16, for γ = 1 we get the value ℮–182.25 which is very close to zero. Thus, the further two observations are from each other, the less influence they have on each other. The amount of influence one observation has on another is calculated as a function of the squared distance between the two. Just like the Polynomial Kernel, when we insert values into the Radial Kernel, we get a high-dimensional relationship. Here 0.11, ℮–182.25, etc. are all the higher dimensional relationships between the two observations. Since the Radial Kernel finds the support vector classifiers in infinite dimensions, it is hard to visualize its mapping.

Please go through the [10], [11], [12] reference links in order to read more about *“The Radial Kernel”.*

# KERNEL PERCEPTRON ALGORITHM

We have n feature vectors given by

:

Initialize = 0, two loops

t = 1.....T

i = 1….n

If the agreement, <= 0, we misclassify the data point. So, we update the as,

We update if any only if we misclassified a data point by the amount of . But finding can be computationally very hard.

where(Kernel Function). So, the explicit feature map of has been replaced by a simple kernel function and it’s output is a real number as it is easy to compute the dot product. Also, when we update the ,

, we get kernel perceptron algorithm(updated),

Initialize , , two loops,

t = 1.....T

i = 1….n

,

By implementing this kind of kernelized perceptron, we can incorporate whole bunch of non-linearity into the algorithm and thus not limiting ourselves to linear classifiers. This is how the kernel tricks make our task easier.

Please go through [13] to know more about Kernelized Perceptron.

# Kernel composition rules

By using the composition rules, we can create very powerful and better kernels.

1. is a kernel function.

, .

Thus, is a kernel function.

1. and .

Then Ḱ = .

= and

Thus, Ḱ is a kernel function.

1. If and are kernels, then

is a kernel.

1. If and are kernels, then

is a kernel. Thus,

is a kernel too.

So, the officially kernel can be defined as:

A kernel → is a symmetric function such that for any n data points, , the nn matrix is a positive semi-definite.

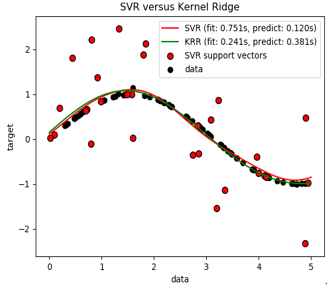
Please go through [14] to know more about Kernel Composition rules and properties.

# other models implementing kernel tricks

There are many other algorithms that operate with kernels. Those include the kernel perceptron, Gaussian processes, ridge regression, principal components analysis (PCA), canonical correlation analysis, linear adaptive filters, spectral clustering, and many others. Any linear model can be turned into a non-linear model by applying the "kernel trick", that is, by replacing the features (predictors) with a **kernel function**.

## Kernel Ridge Regression and Classification

The Kernel Ridge Regression integrates Ridge Regression and Classification using a kernel trick. The Kernel Ridge learns the model which is identical to that learned by Support Vector Regression. The main difference between Kernel Ridge Regression and Support Vector Regression is that they both use different loss functions. Typically, fitting Kernel Ridge is faster than Support Vector Regression for medium-sized training data sets. This is because they use different loss functions. KRR uses ‘squared error loss’ while SVR uses ‘epsilon-insensitive’. Fitting a KRR can done in a closed form. The learned model is non-sparse and therefore, slower than SVR, which learns a sparse model for epsilon > 0 at prediction time. Prediction time refers to the time taken to predict the target values. However, for large-sized training datasets, Support Vector Regression performs better. About prediction time, Kernel Ridge performs no better than Support Vector Regression for all sizes of training datasets. The below figure does the comparison of the time for fitting and prediction by Kernel Ridge and Support Vector Regression for different sizes of training datasets.



As we can see above, SVR bandwidth and complexity selected and prediction done in 0.120 s. KRR bandwidth and complexity selected and prediction done in 0.381 s.

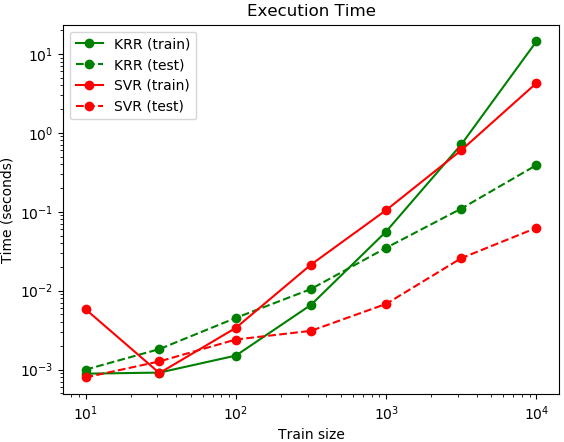


Image Source: [15]

Please go through the [16], [17] reference links to get more knowledge on the topic above.

## Kernel Methods for Vector Output

Previously, the Kernel Methods have been used for scalar outputs by supervised learning problems. Now there is an increasing scope for methods that deal with multiple outputs, e.g. Multitask Learning. Problems can be Multiclass or a Multilabel for which the vector-valued function describes how the kernels translates to the setting of multiple outputs. The problem can be any like learning an unknown relationship function f between an input space and output space . For example, in sensor networks, missing signals from various signals can be predicted by utilizing their correlation with observed signals obtained from other sensors. Another example can be of geostatistics, where prediction of heavy pollutant metals, which is very expensive, can be done using oversampled and inexpensive variables as a proxy. Kernels that capture the relationship between these kinds of problem allow them to borrow strength from each other this way. The algorithm of this type includes multi-task learning (also called multi-output learning or vector-valued learning), transfer learning, and co-kriging. Multi-label classification can be clarified as mapping inputs to the coding vectors with length equal to the number of classes.

Please go through the [18], [19] references for more knowledge on *“Kernel Methods for Vector Output”*

## Kernel Methods for Deep Learning

In machine learning, recent work has shown the circumstances that favors deep architectures, like deep neural nets, over shallow architectures like support vector machines. Deep architectures learn the complicated mappings by transforming their inputs through multiple layers of non-linear processing. For example, in Multilayer Kernel Machines (MKMs), which are deep kernel-based architectures, deep learning is achieved through iterative applications of the kernel PCA. In MKMs, the features at one layer from kernel PCA are inputs to the next layer. While any non-linear kernel is fit to be used for MKMs layer wise PCA, arc-cosine kernels are one of the natural choices that mimic the computations in the neural nets. Support Vector Machines, unlike Deep Networks, are trained by solving a simple problem in quadratic programming. For multiclass data sets, generally, the Deep Networks perform better as compared to Support Vector Machines.

Please visit the [20], [21] references to get more information on “Kernel Methods for Deep Learning”

# Important applications of kernel tricks

## Kernel Tricks for Hand Writing Text Recognition

It is the ability of the machine to extract and interpret handwritten input from various sources such as touch screens, paper documents, and photographs. The image of the text can be sensed using different techniques like scanning or word recognition. It has numerous applications like reading aid for the blind, bank cheques, and transformation of any hand-written document into structural text format. Generally, this process uses the kernel methods used by Multilayer Neural Network or the Support Vector Machines. Normally, a six-stage process is followed as shown in the figure below:

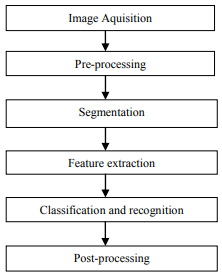


Image Source: [22]

Please visit the [22], [23] references to get more information on *“Kernel Tricks for Hand-Writing Text Recognition”*

## Kernel Tricks for 3D Reconstruction

3D Reconstruction is the process of capturing the appearance, structure, and outline of actual objects. For example, a 3D human face can be reconstructed using the technique of kernel-based deep neural networks. Generally, the 3D Reconstruction consists of the following steps: Image acquisition, Camera calibration, Feature Extraction (Kernel-based algorithms are used here). Magnetic Response Image Reconstruction often uses Kernel Principal Component Analysis where a linear analysis is carried out on a high dimension non-linear feature space. In return, 3D Reconstruction has its application in many other important fields like Medicine, Robotic Mapping, Gaming, Earth Observation, Archaeology, Augmented Reality, and Reverse Engineering.

For more detailed information on how kernel tricks are used in 3D Reconstruction, please go through the [24], [25] references.

# Advantages and limitations of kernel tricks

## **Advantages**

1. We usually get guaranteed precise performance when applying the kernel tricks to a model.
2. Usually, a small set of examples are required to train the model as compared with other techniques.
3. It may work on non-linear separable data without increasing the complexity of the model.
4. The kernel tricks allow the running of several different linear algorithms in a very high dimensional feature space.

## **Limitations**

1. Sometimes, the kernel tricks are unable to linearly separate the data while working in a high dimensional feature space.
2. The selection of an appropriate kernel function is not easy. Hence, we may not end up with an optimal solution.
3. Sometimes, it may be difficult to understand and interpret the complexity of the final model.

# Conclusion

Although some examples we discussed to show the data being transformed from a low dimension to a relatively high dimension, Kernel functions only calculate the relationship between every pair of points as if they are in the higher dimensions; they don’t actually do the transformation. This trick, calculating the high-dimensional relationships without transforming the data to the higher dimension, is called the **Kernel trick.** In this paper, we have explained two Kernel Tricks used in most common.

The Kernel Trick reduces the amount of computation required for the Support Vector Machines by avoiding the actual transformation of the data from low to high dimensions. When we have two categories, but no linear classifier that separates them in a proper way, Support Vector Machines work by moving the data into a relatively high dimensional space. They find a relatively high dimensional Support Vector Classifier that can effectively classify the observations.

There are many other kernels like Linear kernel, Exponential kernel, Laplacian kernel, Hyperbolic or the Sigmoid kernel etc. Please go through [26], [27] to know more about them.

##### References

1. Dan George Bucatanschi

Kernel Methods for Image Processing

<http://personal.denison.edu/~kretchmar/pubs/FaceKernel.pdf>

1. Analytics Vidhya

Understanding Support Vector Machines(SVM) algorithm from examples(along with code)

<https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/>

1. Grace Zhang

What is Kernel Trick? Why is it important?

<https://medium.com/@zxr.nju/what-is-the-kernel-trick-why-is-it-important-98a98db0961d>

1. DataFlair Team

Kernel Functions- Introduction to SVM Kernel & Examples

<https://data-flair.training/blogs/svm-kernel-functions/>

1. Geoff Gordon

Support Vector Machines and Kernel Methods

<https://www.cs.cmu.edu/~ggordon/SVMs/new-svms-and-kernels.pdf>

1. Prashant Gupta

Cross-Validation in Machine Learning

<https://towardsdatascience.com/cross-validation-in-machine-learning-72924a69872f>

1. Ivan Mejia-Guevara and Angel Kuri-Morales

MP-Polynomial Kernel for Training Support Vector Machines

<https://link.springer.com/content/pdf/10.1007%2F978-3-540-76725-1_61.pdf>

1. Kontorovich and Sabato

Introduction to learning and Analysis of Big Data

<https://www.cs.bgu.ac.il/~inabd171/wiki.files/lecture8_handouts.pdf>

1. Wikipedia

Polynomial Kernel

<https://en.wikipedia.org/wiki/Polynomial_kernel>

1. ScienceDirect,

Radial Basis Function

<https://www.sciencedirect.com/topics/computer-science/radial-basis-function>

1. sebastianraschka, Machine Learning FAQ, How do I select SVM kernels?

<https://sebastianraschka.com/faq/docs/select_svm_kernels.html>

1. Hilary, A. Zisserman, SVM dual, kernels and regression

<http://www.robots.ox.ac.uk/~az/lectures/ml/lect3.pdf>

1. Alan Ritter

Kernels and Kernelized Perceptron

<http://aritter.github.io/courses/5523_slides/kernels.pdf>

1. Justin Domke,

Statistical Machine Learning, Kernel Methods and SVMs

<https://people.cs.umass.edu/~domke/courses/sml2011/07kernels.pdf>

1. Scikit Learn

Kernel Ridge Regression

<https://scikit-learn.org/stable/modules/kernel_ridge.html>

1. Gunnar Ratsch, An intoduction to Kernel Methods for Classification, Regression and Structured Data

<https://users.soe.ucsc.edu/~niejiazhong/slides/ratsch.pdf>

1. Myung-Hoe Huh, Department of Statistics, Korea University, Korea, Kernel Trick- Regression and Classification

<https://www.researchgate.net/publication/276128783_Kernel-Trick_Regression_and_Classification>

1. T.Evgeniou, C.A.Micchelli, and M.Pontil. [Learning multiple tasks with kernel methods](http://www.jmlr.org/papers/volume6/evgeniou05a/evgeniou05a.pdf). Journal of MachineLearning Research, 6:615–637, 2005.
2. Kernels for Vector-Valued Functions: a Review Mauricio A. Alvarez, Lorenzo Rosasco, and Neil D. Lawrence

<http://cbcl.mit.edu/publications/ps/MIT-CSAIL-TR-2011-033.pdf>

1. Mikhail Belkin, Siyuan Ma, Soumik Mandal,

To Understand Deep Learning We Need To Understand Deep Learning,

<http://proceedings.mlr.press/v80/belkin18a/belkin18a.pdf>

1. Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, Eric P. Xing, Deep Kernel Learning

<http://proceedings.mlr.press/v51/wilson16.pdf>

1. Parveen Kumar, Nitin Sharma, Arun Rana

Handwritten Character Recognition using Different Kernel based SVM Classifier and MLP Neural Network

<https://pdfs.semanticscholar.org/cb2e/16204c143ec14e343c23e90c441f2447f9fe.pdf>

1. JanainaMourao-Miranda, Kernel Methods for Pattern Recognition

<http://www.mlnl.cs.ucl.ac.uk/pronto/course_slides_2018/KernelMethodsforPatternRecognition_2018.pdf>

1. Johannes F. M. Schmidt, Claudio Santelli, Sebastian Kozerke, MR Image Reconstruction Using Block Matching and Adaptive Kernel Methods

<https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0153736>

1. Klaus Mueller, Roni Yagel, John J. Wheller

Fast Implementations of Algebraic Methods for 3D Reconstruction from Cone-Beam Data

<https://www3.cs.stonybrook.edu/~mueller/papers/impl.pdf>

1. Educba

Kernel Methods

<https://www.educba.com/kernel-methods/>

1. Barbara Engelhardt

Probabalistic Machine Learning, Kernels and Kernel Methods

<https://www.cs.princeton.edu/~bee/courses/scribe/lec_10_09_2013.pdf>