Austin SCHWINN
Jérémie BLANCHARD

KERNEL METHODS - REPORT

Advanced Machine Learning – Practical Session

GitHub: https://github.com/amschwinn/adv_machine_learning_lab

UJM Saint-Etienne – Master 2 Machine Learning and Data Mining – 2018/2018

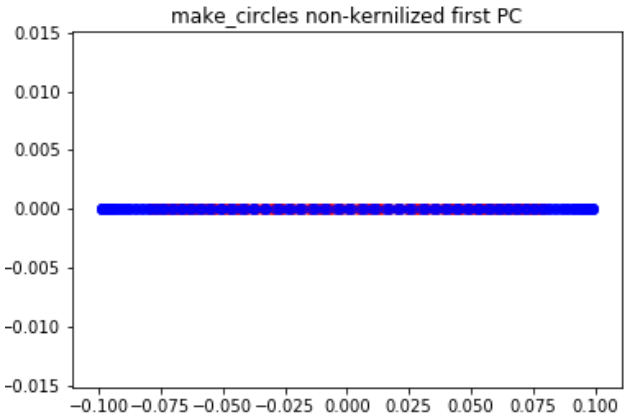
Table of Contents

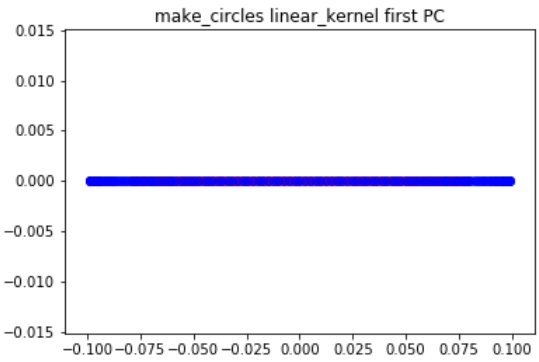
Kernel-PCA

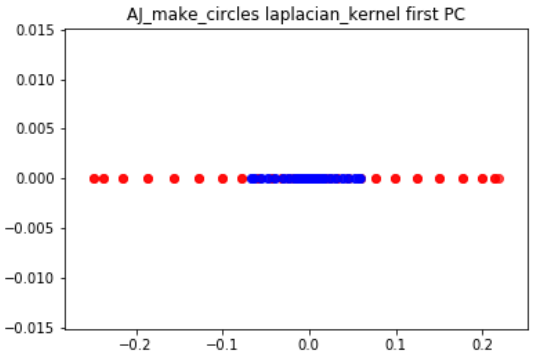
We chose to implement PCA and kernel-PCA in Python, then we launch different tests with different kernels to visualize the differences between them, and see if kernelizing data improve really the result.

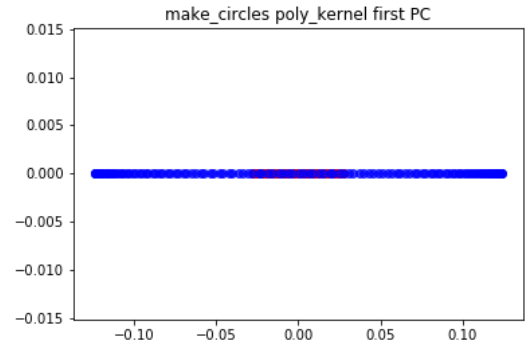
At first, we have done our tests by running the PCA and k-PCA on our data and only keeping the first principal component, let's see how our data behave with this configuration.

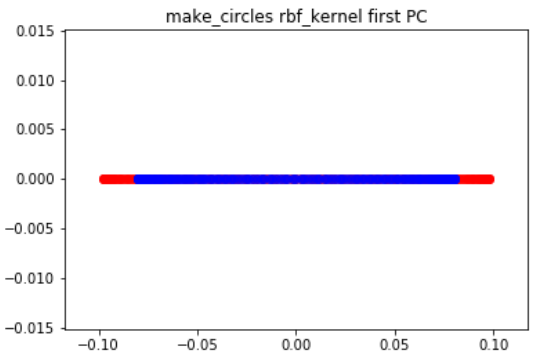
First principal component | Circle shape





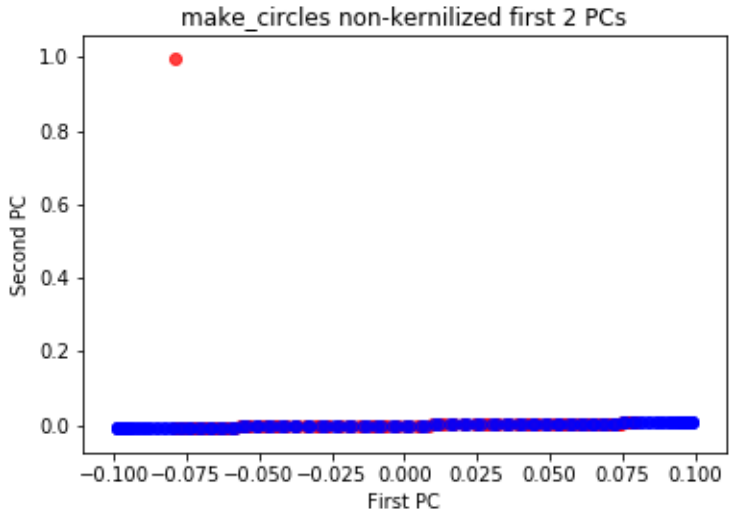


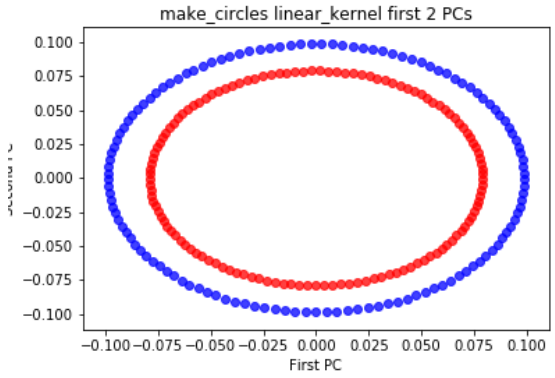


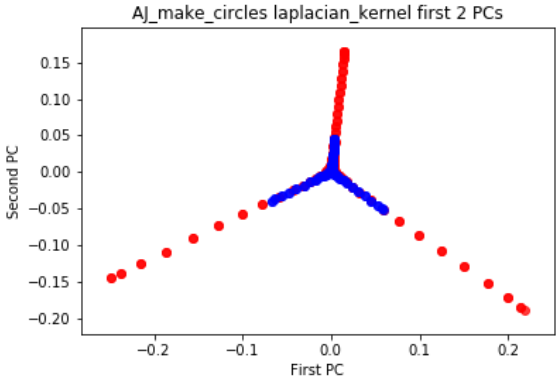


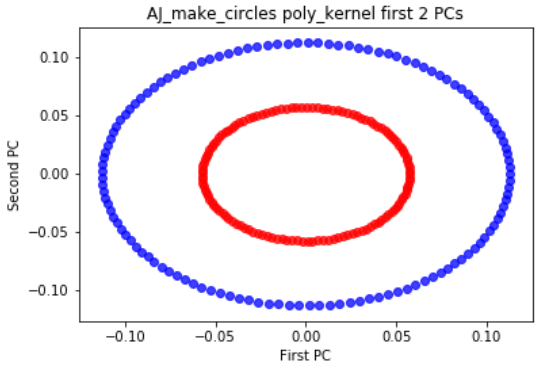
By keeping the first principal component, we can see that we are not able to split the data into two different clusters, even with kernels.

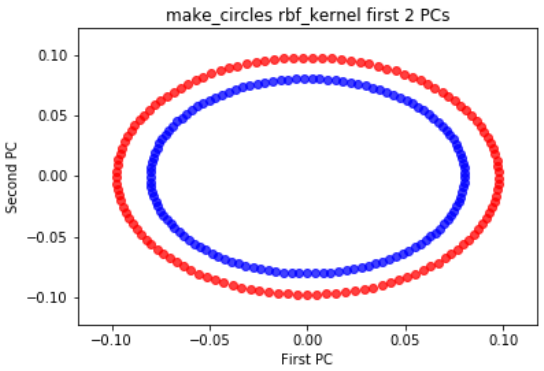
Two first principal components | Circle shape





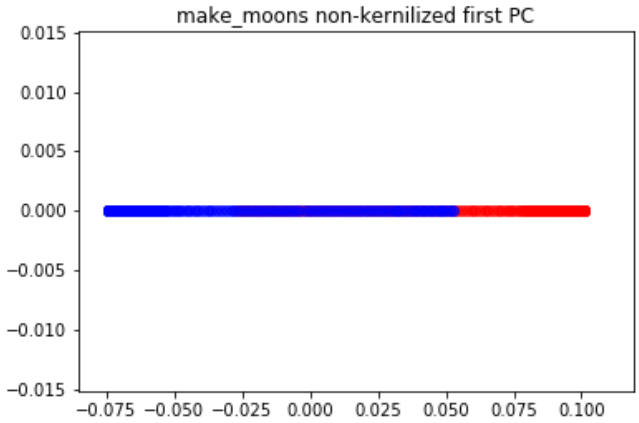


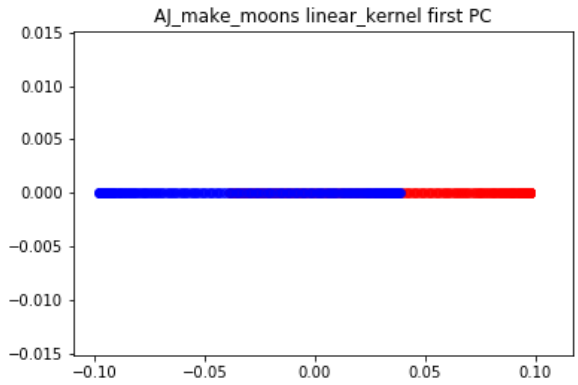


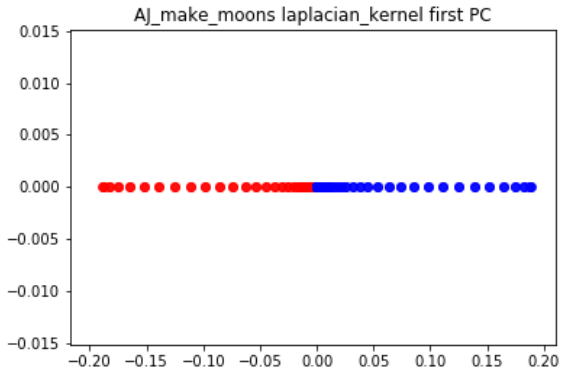


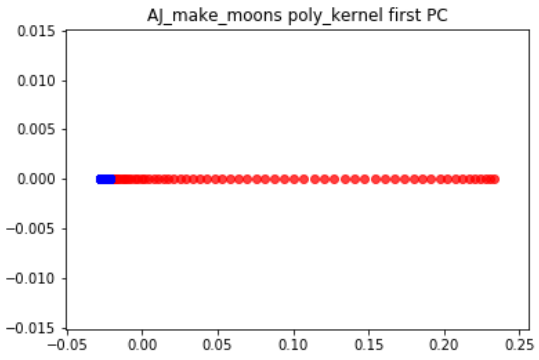
Now by using variables in a 2-dimensionals space we can clearly see the effects of kernels. Indeed, without kernel our data are not differentiable except for some outliers. With kernels we have a big improvement and every kernels give a different shape and a different efficiency.

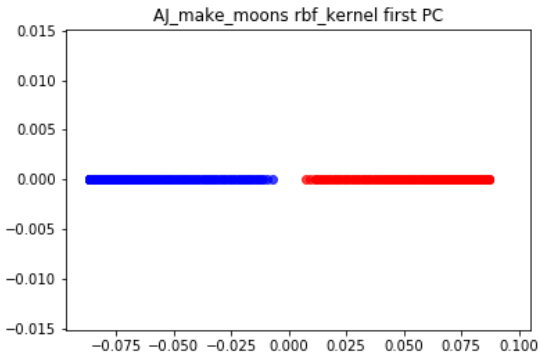
First principal component | Moon shape





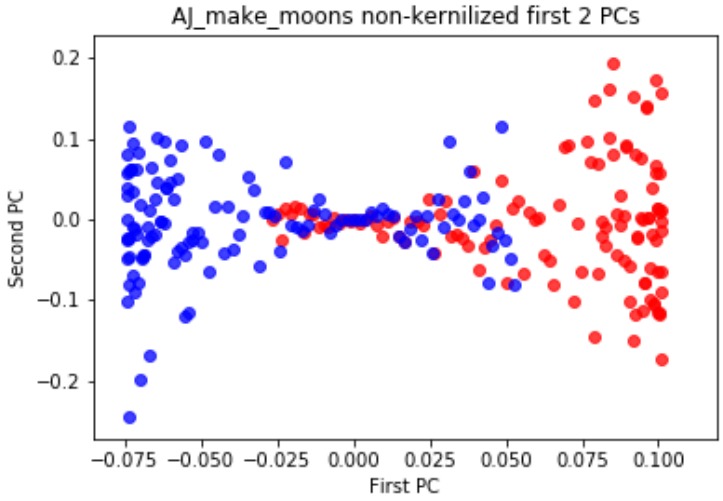


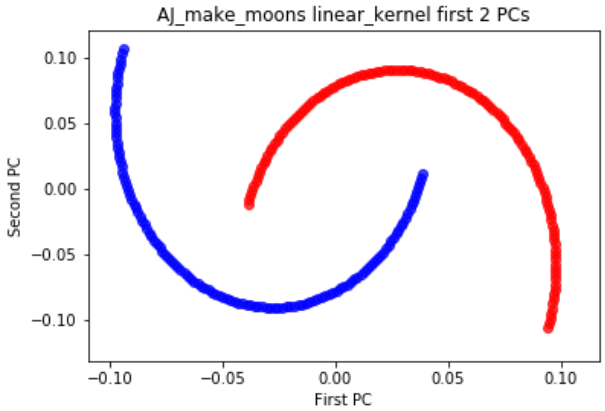


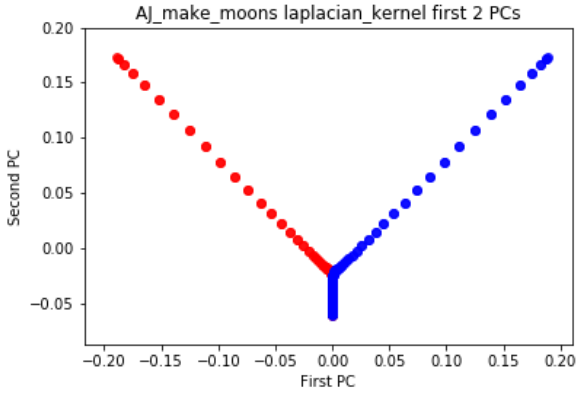


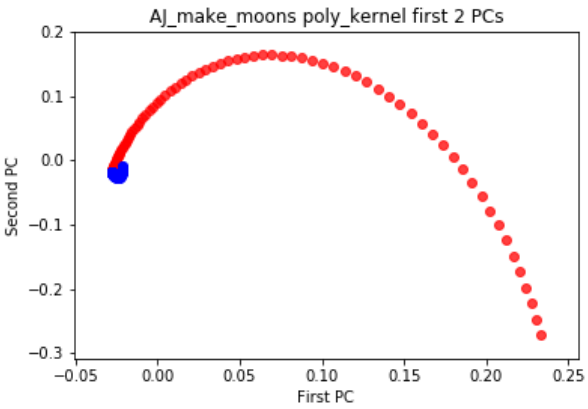
The differences between the moon shape, and the circle shape that we saw before, is that with the moon shape even by keeping only the First principal component we are able, with a specific kernel, to differentiate our classes. Here, the RBF kernel works really well on this dataset compared to another one.

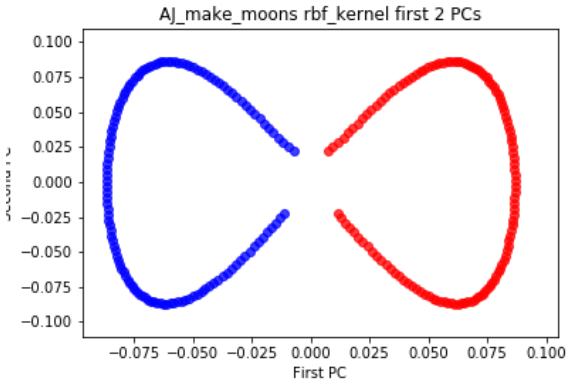
Two first principal components | Moon shape





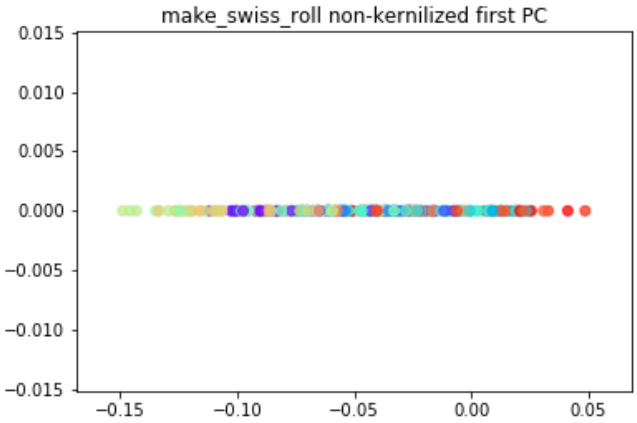


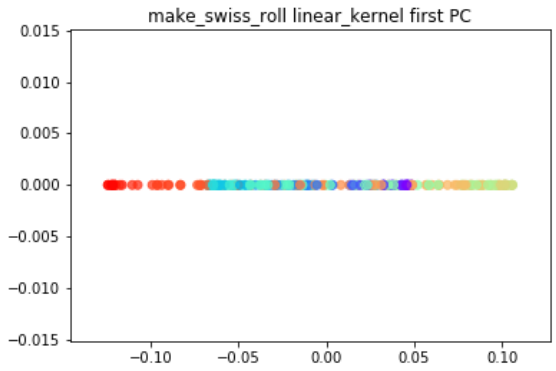


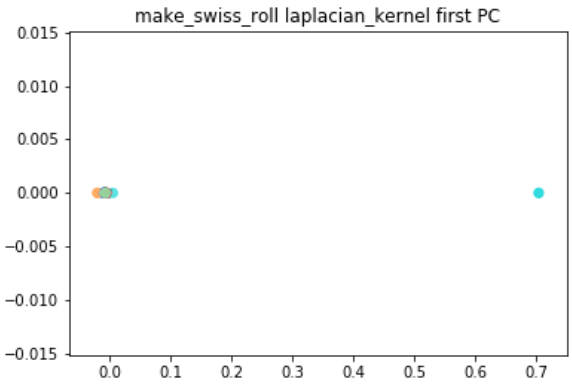


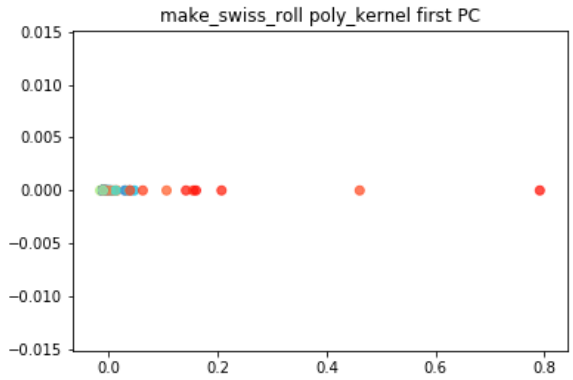
Here thanks to the kernels it is possible to differentiate our data in some cases. We can see that some kernels are better than other to classify these data. The RBF and linear one are doing a good job to separate our data.

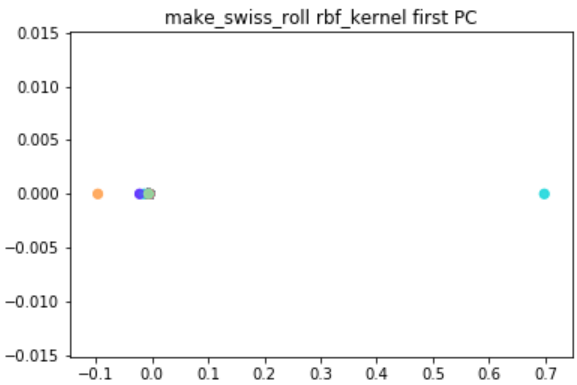
First principal component | Swiss role shape





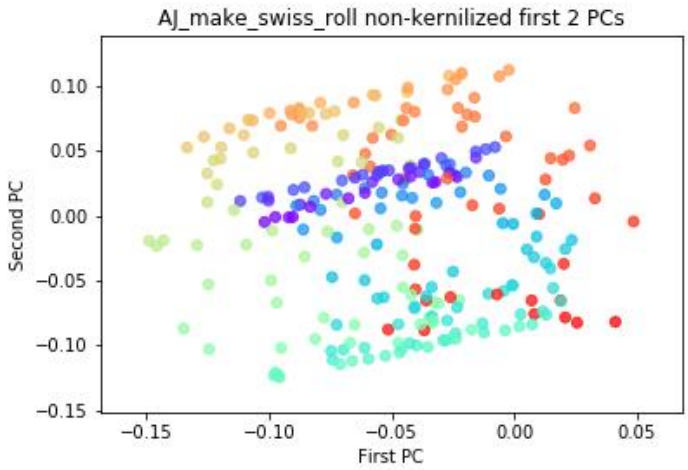


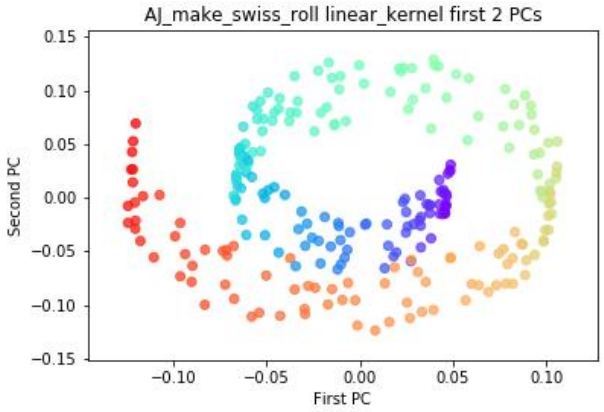


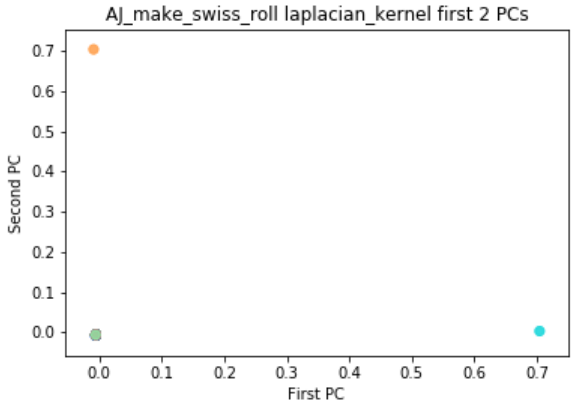


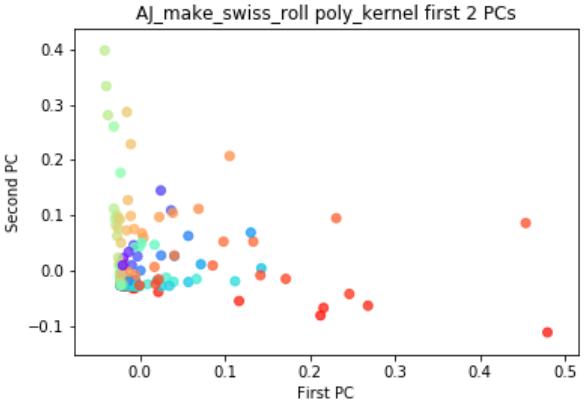
The kernels add some light improvements to the data classification, but it's still not possible to differentiate our dataset.

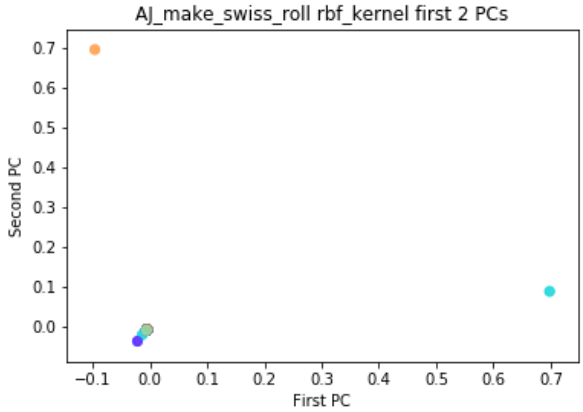
Two first principal components | Swiss role shape











This Swiss role shape dataset is a perfect example that some kernels are better than other to classify specific data. Indeed, here the linear kernel is way more efficient than the others kernels to classify this swiss role data shape.

Efficiency

The goal of this experiment was also to run some efficiency test to see if the kernels have an impact on the computing time.

Moon Dataset

Kernel	Seconds
Linear Kernel	0,0139
RBF Kernel	0,0113
Polynomial Kernel	0,0144
Laplacian Kernel	0,0251

Circles Dataset

Kernel	Seconds
Linear Kernel	0,0069
RBF Kernel	0,0285
Polynomial Kernel	0,0139
Laplacian Kernel	0,0217

Swiss Roll Dataset

Kernel	Seconds
Linear Kernel	0,007
RBF Kernel	0,0354
Polynomial Kernel	0,0171
Laplacian Kernel	0,0289

Computation times comparisons

Moon Dataset

Kernel	Variance %
Linear Kernel	100
RBF Kernel	15,0339873
Polynomial Kernel	98,66663
Laplacian Kernel	3,74245794
No kernel	100

Circles Dataset

Kernel	Variance %
Linear Kernel	100
RBF Kernel	13,3932152
Polynomial Kernel	45,6286997
Laplacian Kernel	2,42600815
No kernel	100

Circles Dataset

Kernel	Variance %
Linear Kernel	71,8553113
RBF Kernel	1,56018969
Polynomial Kernel	57,6575035
Laplacian Kernel	1,03815176
No kernel	100

Explained Variance

Thanks to these data we can deduce that it is possible to find a kernel which is fast and able to well separate the data in function of the dataset. Indeed, it seems that some kernels are better than others to separate a given dataset, and it is possible to choose between a high computational speed, a good classification or a mix of both.

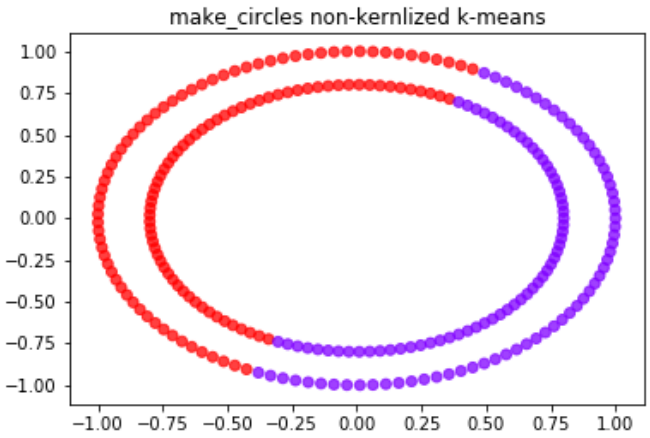
In our example here are the best combinations:

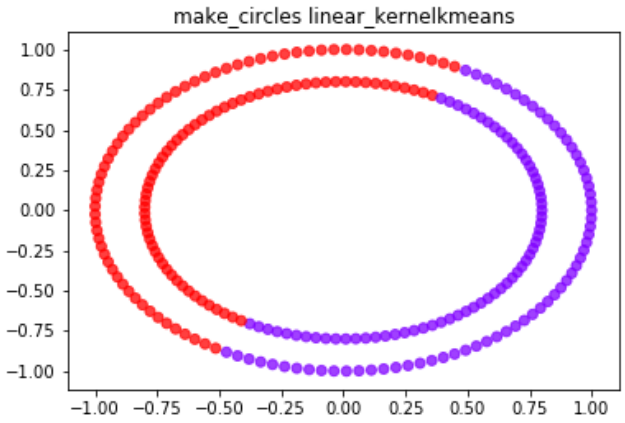
- Moon shape dataset: Linear Kernel – Fast and separate well our data
- Circles shape dataset: Polynomial Kernel – Fast and have a larger margin than the linear kernel
- Swiss Roll dataset: Linear Kernel – Fast and the only one to classify our data without losing too much.

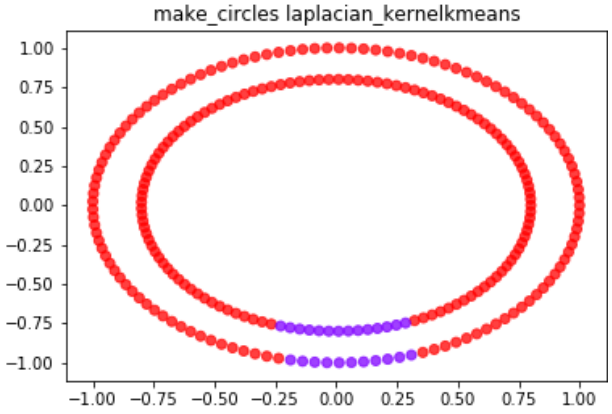
Kernel K-means

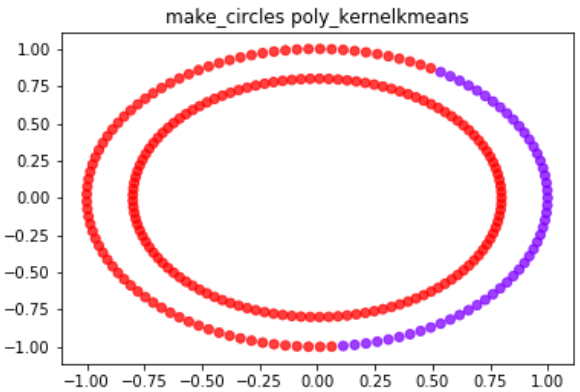
Let's see if we obtain the same kind of results as with PCA by comparing K-Means and Kernel K-Means.

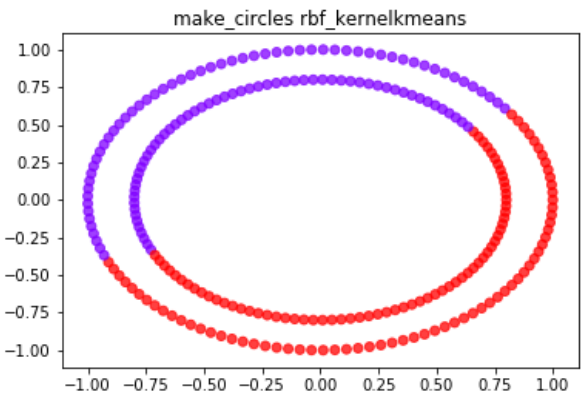
Circles shape dataset



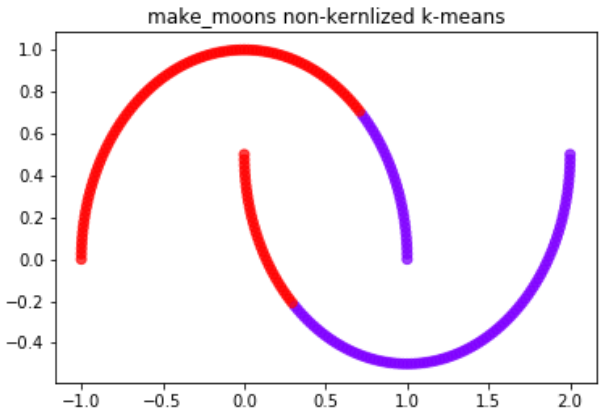


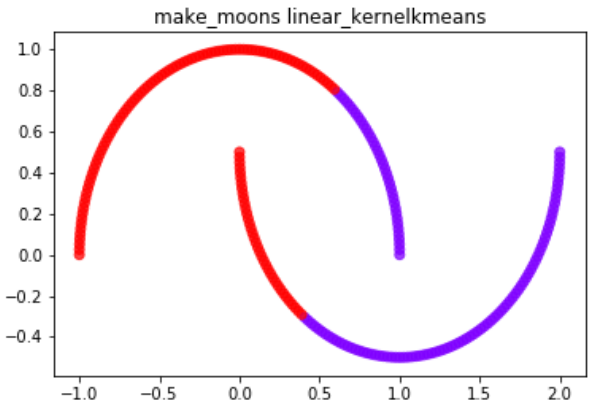


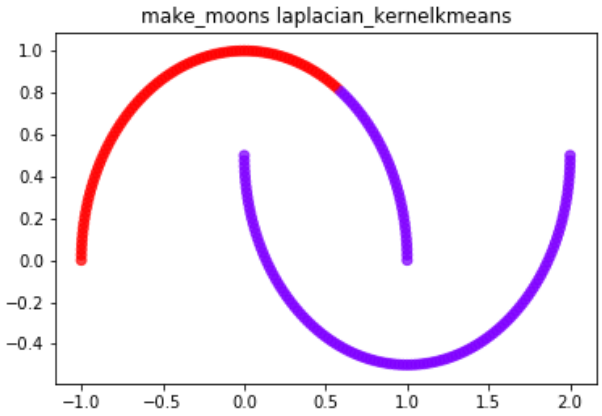


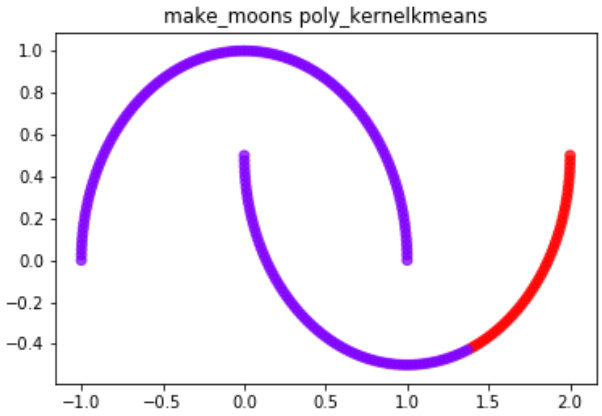


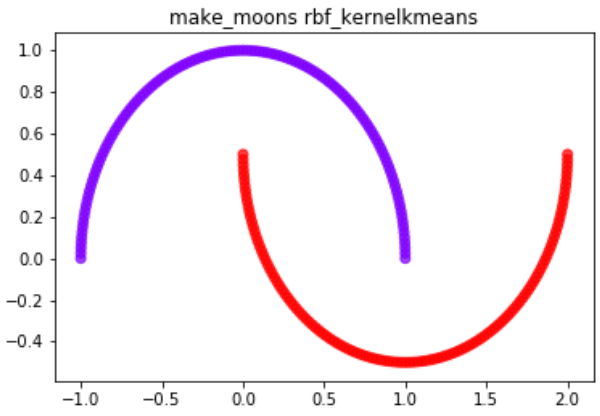
Moon shape dataset



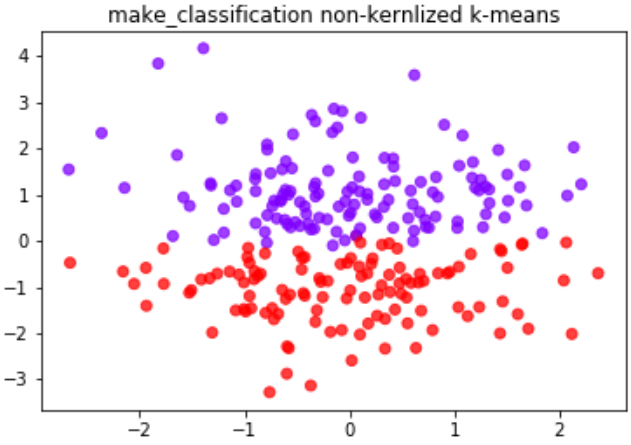


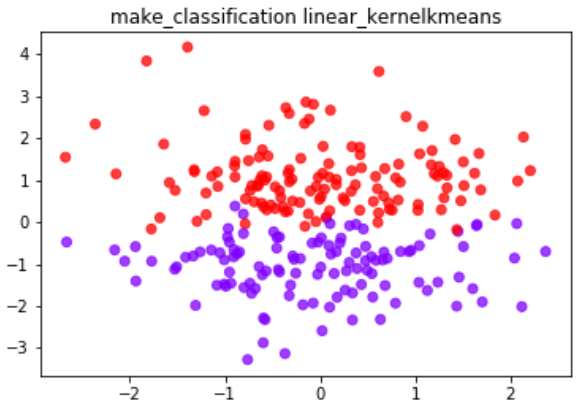


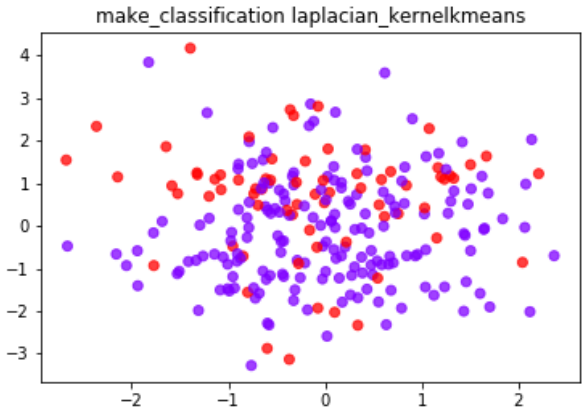


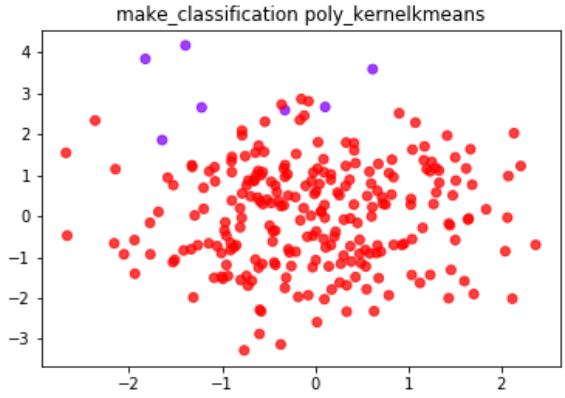


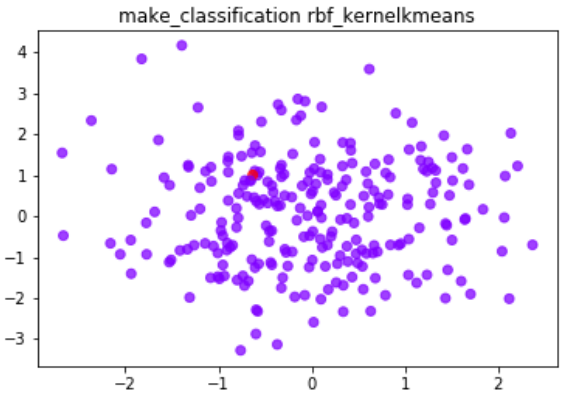
Classification dataset











Efficiency

To determine the impact of kernels we saved the accuracy for every K-Means.

Moon Dataset

Kernel	Accuracy %
Linear Kernel	70,4
RBF Kernel	100
Polynomial Kernel	72,4
Laplacian Kernel	15,2
No kernel	74,4

Circles Dataset

Kernel	Accuracy %
Linear Kernel	50
RBF Kernel	49,2
Polynomial Kernel	70
Laplacian Kernel	48,8
No kernel	50

Classification Dataset

Kernel	Accuracy %
Linear Kernel	40,4
RBF Kernel	48,8
Polynomial Kernel	47,2
Laplacian Kernel	36,4
No kernel	37,6

The fact that kernels are behaving in different ways depending of the dataset shape is also verified with K-Means. The Moon dataset is a good example of it because the RBF Kernel allow us to get a 100% accuracy while the Laplacian one only 15,2%.

This also prove that it is important to run several tests with different kernels before choosing one. Because there are big differences in the results, and we should pick carefully a kernel for a specific dataset.

Logistic Regression

Principles

Logistic Regression is appropriate for binary response variables where the data can be classified into one or two classes. Other type of Logistic regressions can also classify data in more than two classes by adding "gray-scales" between the black and the white classes. But in any case, the response variable is bounded between 0 and 1.

Comparisons

To compare the regular Logistic Regression with the Kernelized logistic regression we used a dataset called "breast-cancer-wisconsin". Let's how our kernels behave with this dataset.

Kernel used	Accuracy %
No Kernel	0,978914967
Linear	0,968449404
RBF	0,630934975
Polynomial	0,968480185
Laplacian	0,627425933

Outputs of our Python script

At first, we see that the kernels are not useful to split these data, because the Logistic Regression with the best result is the one with no kernel. It is also important to see that the kernels that works the best with logistic regression are the Linear one and the Polynomial one, because the Logistic regression is a case of Linear Regression.

One Class SVM and Maximum Enclosing Ball

We choose to implement the SVDD Problem under the form of an optimization problem on MatLab because we were able to find a toolbox (SVM-KM) which simplify the optimization task. The implemented version of the SVDD problem is the one with slack variables, in this way our maximum enclosing ball is going to make errors.

$$\begin{cases} \min\limits_{m,c,\xi} & \frac{1}{2}\|c\|^2 - m + C\sum\limits_{i=0}^{n}\xi_i \\ \text{with} & x_i^\top c \geq m + \frac{1}{2}\|x_i\|^2 - \xi_i\,; \quad i = 1,n \\ \text{and} & 0 \leq \xi_i\,; \quad i = 1,n \end{cases}$$

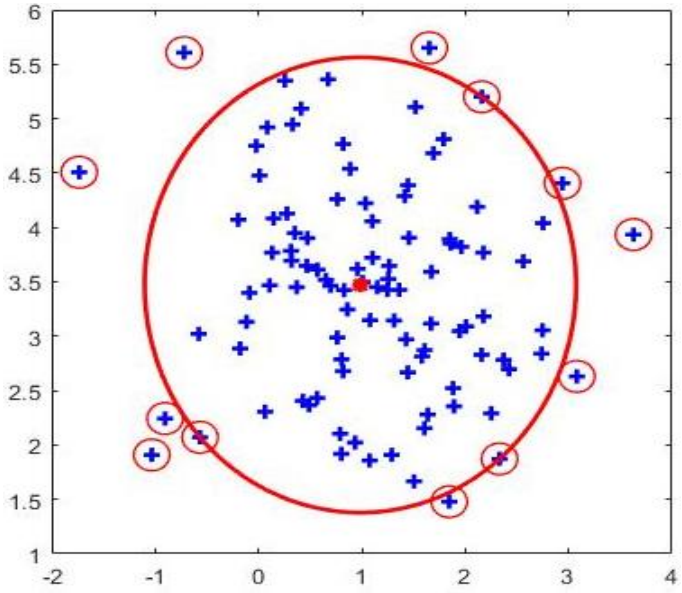
Optimization problem

```
n = 100;
p = 2;
Xi = randn(n,p) + 1.2*ones(n,1)*[1 2.8];

G = Xi*Xi';
nx = diag(G);
e = ones(n,1);
%% SVDD
% min  R^2 + C sum(Xi_i)   s.t.     \| x_i
- c\|^2  < =  R^2 +
xi_i       pour   i=1,n        et 0 <= xi
% Recasted as a QP

 C = 10;
 cvx_begin
    cvx_precision best
    variables m(1) cSVDD(2) xi(n)
    dual variables d dp
    minimize(  .5*cSVDD'*cSVDD - m  + C *
sum(xi) )
    subject to
       d  :  Xi*cSVDD  >= m + .5*nx - xi;
       dp: xi >= 0;
 cvx_end

R = cSVDD'*cSVDD - 2*m;
pos = find(d > eps^.5);
```

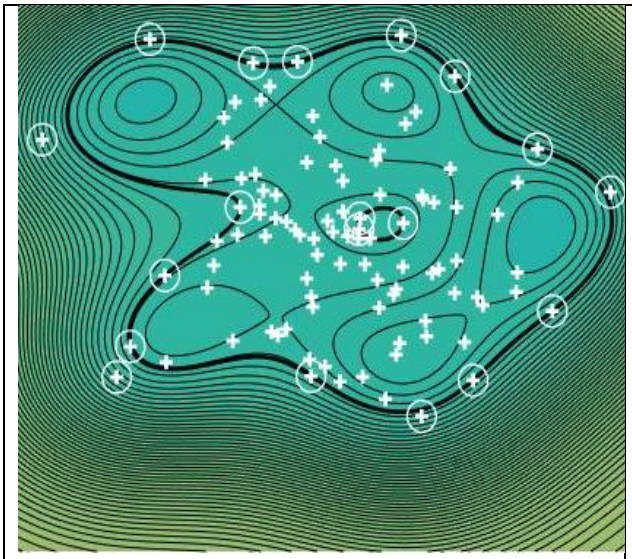
Matlab implementation

To evaluate the performance of this implementation and to compare them with kernels we are going to start with a low C to see if the kernels manage to classify in a better way our data.

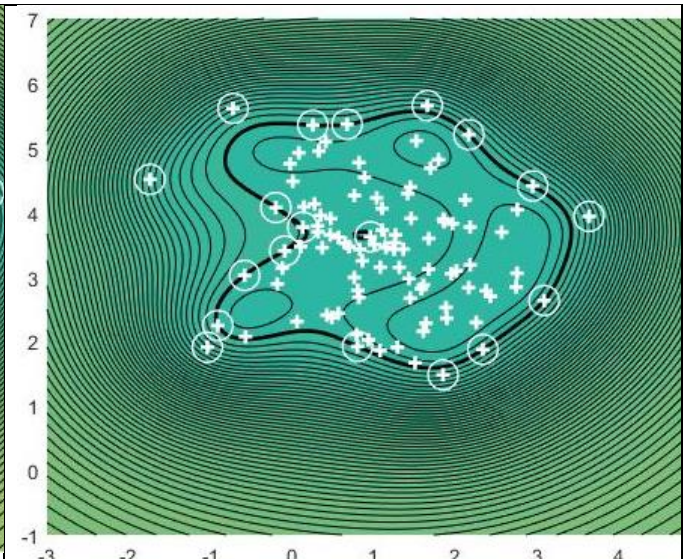
C = 0.1



No kernel



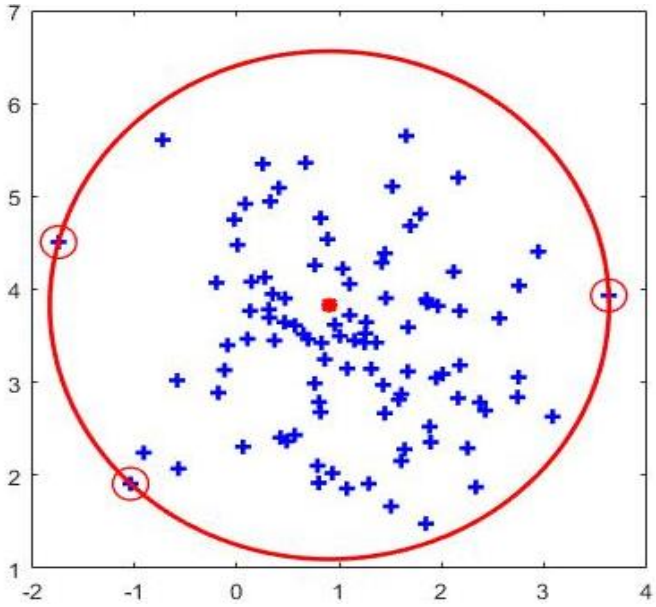
Gaussian Kernel

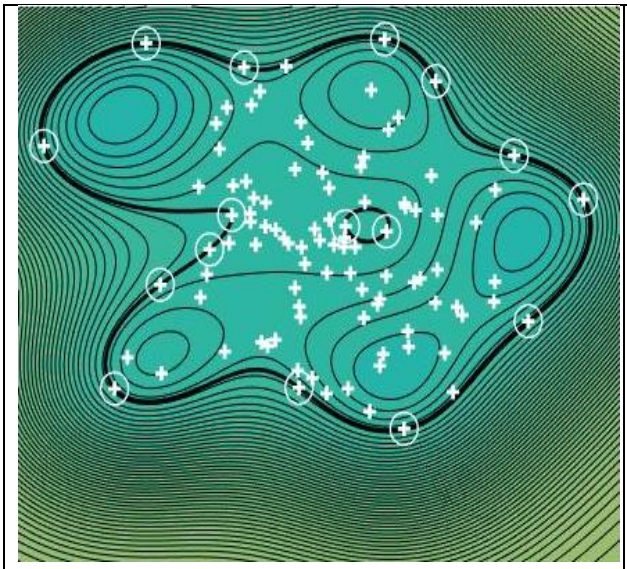


Rational Kernel

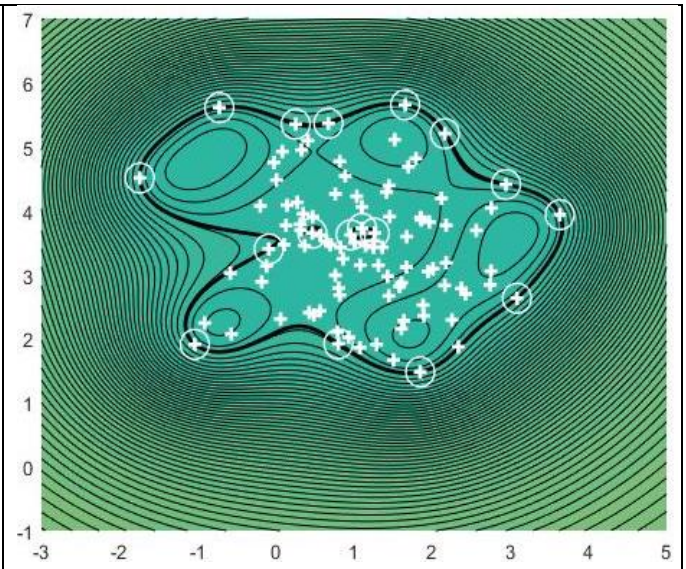
In this case we can see that the gaussian kernel manage better than the rational one to not do any error, but the Rational kernel manage to regroup better the data together than the others.

C = 10





Gaussian Kernel



Rational Kernel

Because of a large C the SVDD does not need any kernel to classify every of his example, and the gaussian does not give any improvement, while the rational one is not doing any mistake and regroup all the data together.

As a conclusion we can say that the Rational kernel should be better that the gaussian or no kernel. Indeed, even if it is doing some mistakes it can regroup the data together and at test time it is going to be more accurate.