ADVANCED MACHINE LEARNING
KERNEL METHODS

Duy Anh Philippe Pham *

November 14, 2020

Thanks to kernalisation it is possible to project data in a larger dimensional space in order to take advantage of linear tools.
The goal of this report is to present kernelization methods by making a comparison with primitive methods for PCA and Kmeans. Then we will use kernel trick to solve the one class SVDD algorithm and to do online learning. You will find at this address the implementation of the algorithms with their commentary : `https://github.com/Daphilippe/Kernel-methods`

1 PCA

1.1 Problem Statement

By using the Principal Component Analysis method a.k.a. PCA, we have a better representation of data. Usually PCA is used to project data from an n-dimensional space to a lower dimensional space. Here we will show that kernelisation also allows to change the data representation space.
We take advantage of this first part to make a benchmark allowing to compare the native method of the kernalised method. Our goal is to identify the influence of hyperparameters on the kernelisation.

1.2 Theory

1.2.1 PCA

To implement PCA, we first need to group the data in a matrix table. We re-center the data and then compute the covariance matrix.
From this matrix we can make a singular value decomposition using svd or an eigens values search. We check that the eigens values are well classified in decreasing order and we proceed to the reduction of dimension keeping only the features with the most important contributions. The inertia rate, a.k.a. variance, is known thanks to eigenvalues.
Note: The Singular Value Decomposition and the eigenvalue decomposition are switchable. This has no impact in the final result because the SVD is a generalization to non-square matrices of the eigenvalue decomposition. We can replace one by the other in this case because we multiply the transpose of a matrix by itself which gives a square matrix.

*dap.pham@cpe.fr

1.2.2 Kernel PCA

In order to make the PCA kernel it is enough to use a projection kernel allowing to project in another space. This step is done before the first step of the PCA.

1.3 Results

This subpart is the control group allowing to see the differences with the kernalised method. Particular care is taken in the implementation of the different tests in order to have the most general results possible.

1.3.1 PCA

For a different number of data, we have this representation in Figure 1.

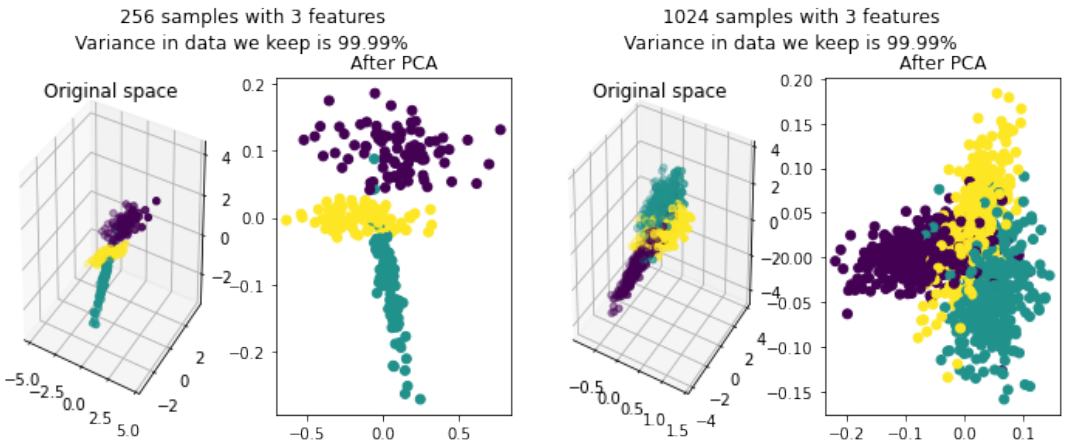

Figure 1: Comparison between 2 datasets processed by PCA

We notice that the variance is very important here and not representative of a general case because the data are distributed very closely around the projection plane as we can see in the 3D representation "Original Space".
We proceed to a benchmark control which will allow us to compare with the kernalised method. We will compare the relative speeds of executions and of the variance.

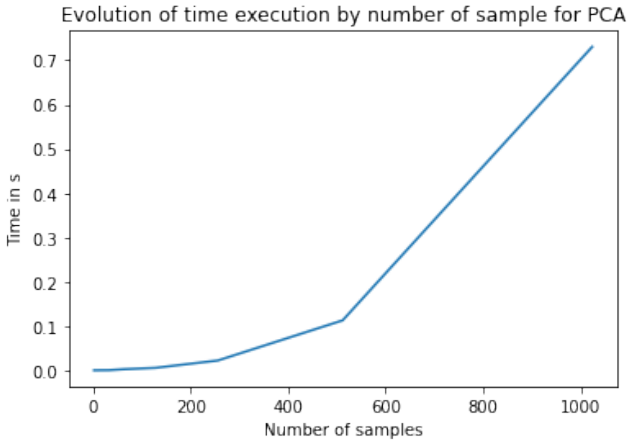

Figure 2: Evolution of time execution by number of sample for Kernel PCA

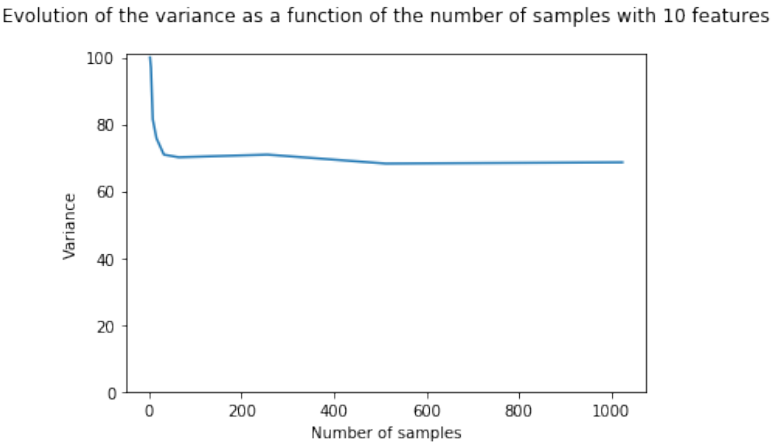

Figure 3: Evolution of the variance as a function of the number of samples

1.3.2 Kernel PCA

We test different kernel on the same dataset: cosine, polynomial and gaussian.

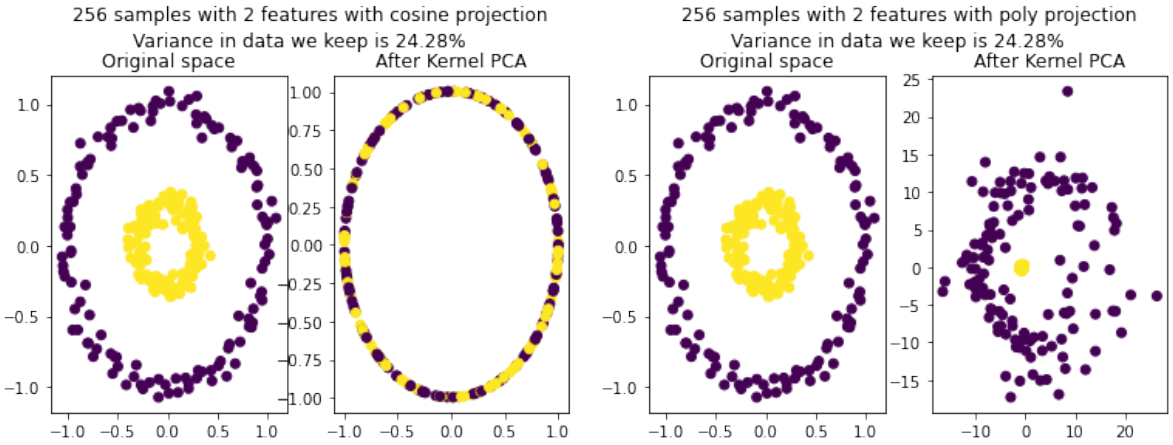

Figure 4: Comparison of Cosine and Polynomial kernel

Cosine and polynomial kernel We focus on the Gaussian projection until the end of this part. This projection is defined by this equation:

$$\forall X \in Dataset \text{ and } x \in X, f(x) = e^{-\gamma(x-x_0)^2} \text{ with } x_0 = mean(X)$$

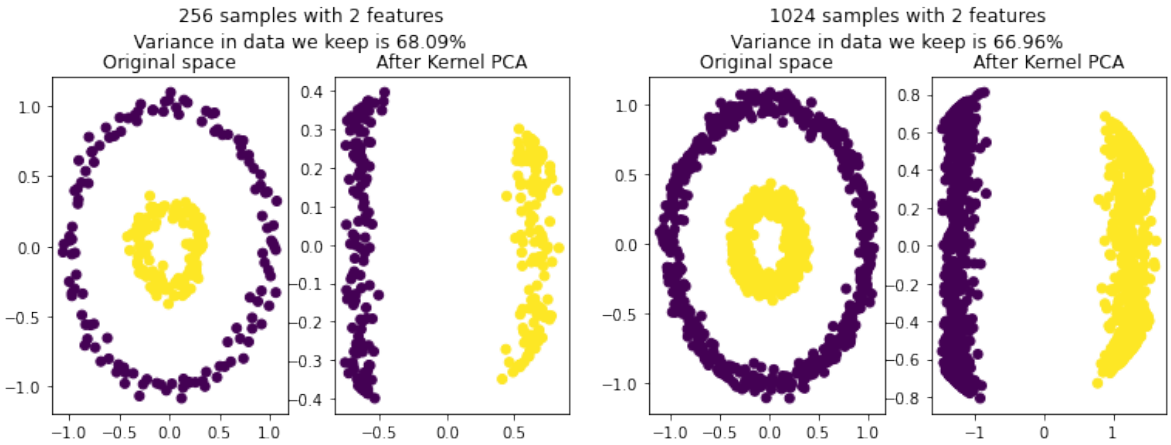

Figure 5: Comparison between 2 datasets processed by PCA

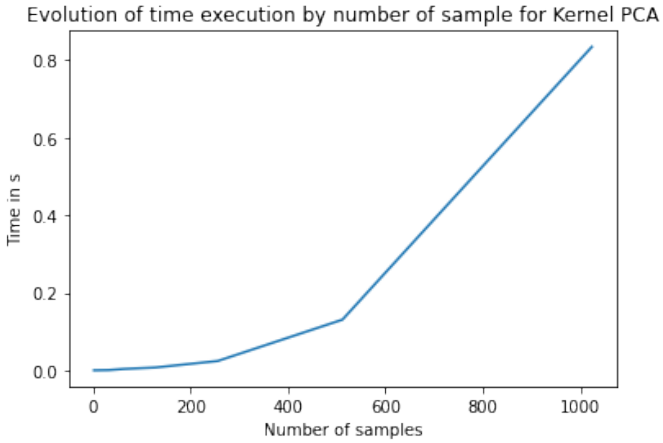

Figure 6: Evolution of time execution by number of sample for Kernel PCA

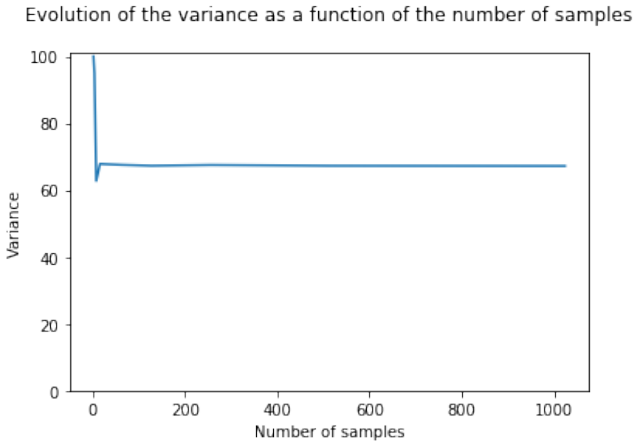

Figure 7: Evolution of the variance as a function of the number of samples

Gaussian kernel For different gamma values we have different projections (Figure 8)

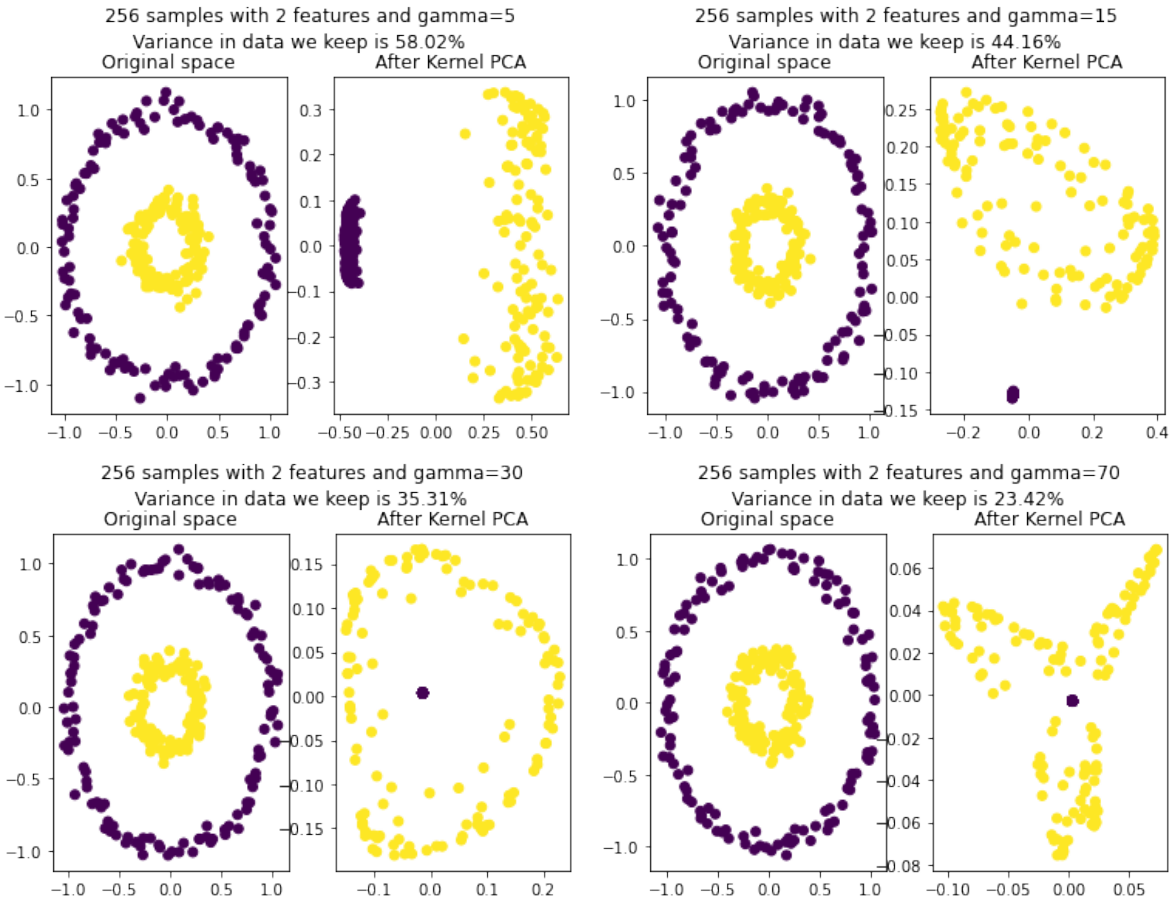

Figure 8: Comparison between different gamma values

The gamma variable is considered a hyperparameter. It is a parameter that is specific to the method of kernelisation and the projector chosen here.

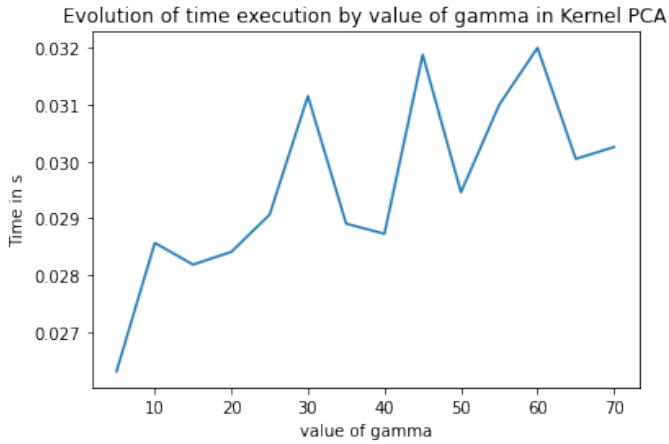

Figure 9: Comparison between different gamma values

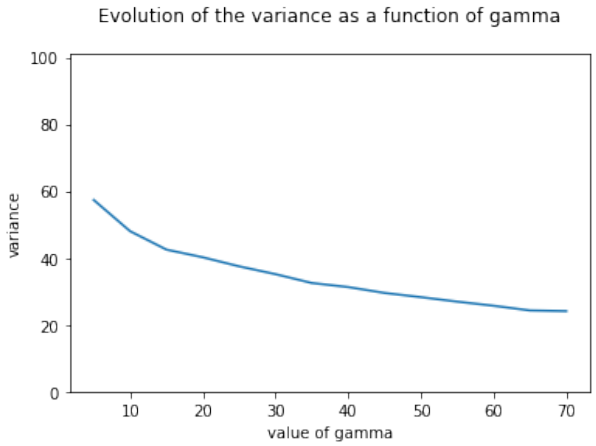

Figure 10: Evolution of the variance as a function of gamma value

1.4 Discuss

In a qualitative way we have data representations that are more explicit or otherwise easy to interpret. (Figure 1 and Figure 5). The execution time (Figure 2 and Figure 6) as well as the variance (Figure 3 and Figure 7) are of the same order of magnitude. It is not a discriminating characteristic and it is an expected result since the algorithmic nucleus between the PCA and the PCA kernel is the same. Note that the less data you have to process, the easier it is to find an optimal projection, i.e. maximizing the variance. This is why with a small number of examples we have a variance close to 1, i.e. minimal loss of information.
It is interesting to point out that in a space of 12 features, keeping in mind that 2 main components we have a loss of only about 1 third of the information in an asymptotic way. It would be interesting to push the investigations on even larger dimensions.(Figure 3)
It can be seen that there is a slight increase in computation time for high gamma values, but this is not a decisive criteria in view of the orders of magnitude. The contribution can therefore be considered negligible compared to the contribution of the number of data (Figure 6 and Figure 9). On the other hand, the variance is significantly impacted depending on the gamma value. We can assume a logarithmic decay or a convergence towards a value for a fixed dataset. (Here around 30, Figure 10)

We conclude in this first part that the influence of kernelisation in terms of computation is relatively negligible compared to the algorithm without kernelisation. Therefore we dispense with a benchmark on the nucleus of the algorithm because it depends above all on its implementation.
On the other hand one will keep the Gaussian kernel because it offers the greatest diversity of projection as a function of gamma. This gamma parameter should be chosen carefully because it has a direct influence on the quality of the projection, in the case of PCA the variance is directly impacted. Its contribution in the execution time is negligible.

2 Kmeans

2.1 Problem Statement

Kmeans is a segmentation algorithm based on the calculation of barycenters and the association of an identical label to the barycenter closest to the considered point. By repeating this process we improve the prediction and we have a convergence that is guaranteed. The most important

part of this process is the calculation of the distance between the barycenter and the point under consideration. However, there are problems where the topology does not allow a good segmentation of the data and one wonders if a kernelisation solution can solve this problem.

2.2 Theory

2.2.1 Kmeans

To come back in more detail on the Kmeans algorithm, k barycentre associated with a label must initially be randomly assigned. Then a first calculation of the closest neighbors is made with respect to the Euclidean distance between the barycenter and the example under consideration. The label associated with the barycenter with the smallest distance is associated with the barycenter. We note that the Euclidean distance is the most relevant here. At the end of the label attributions the barycenters are recalculated for each group. The process is repeated as many times as necessary to ensure convergence. In our study only the labeling is important, one could also return the position of the barycentres as an indication.

2.2.2 Kernel Kmeans

To make the kmeans kernel it is enough to project the data beforehand with a projection kernel and then reuse the previous kmeans algorithm. Thanks to the kernel trick we don't need to reproject the result in the original space because in this case we are only interested in labels.

2.3 Results

2.3.1 Kmeans

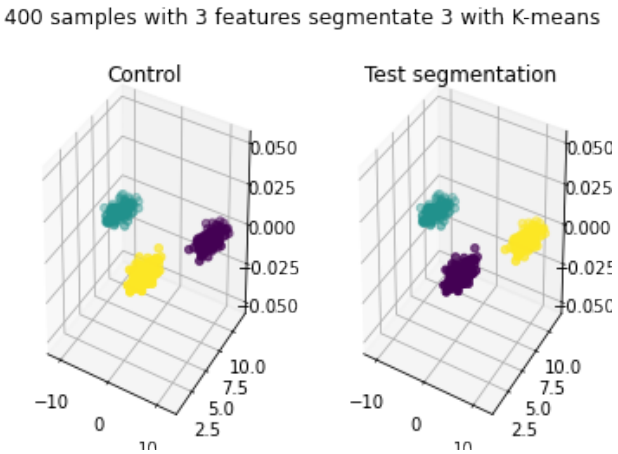

Figure 11: Comparison between 2 datasets processed by Kmeans

We generate data labeled as a control group. It is important to note that we never use its labels as a support for the determination of our solution. These labels are simply there to get an idea of the performance of the algorithm we have developed.

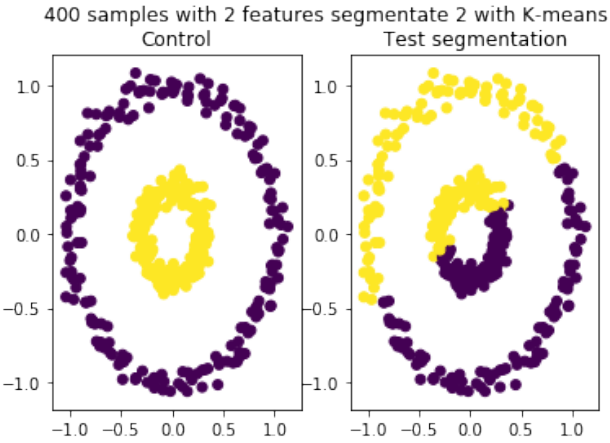

Figure 12: Linearly non-separable dataset

2.3.2 Kernel Kmeans

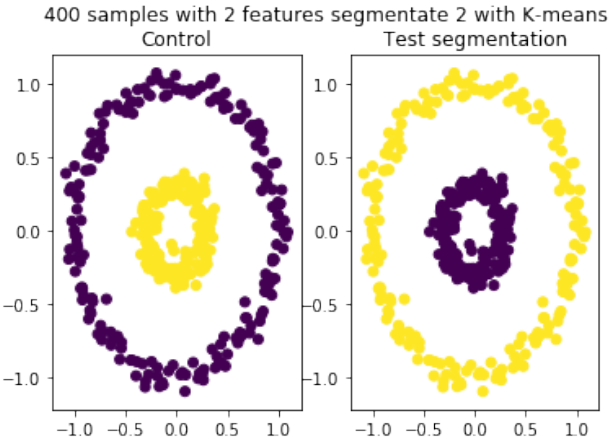

Figure 13: Comparison between 2 datasets processed by Kmeans

In figure 13 the segmentation is identical, except for an arbitrary label value.

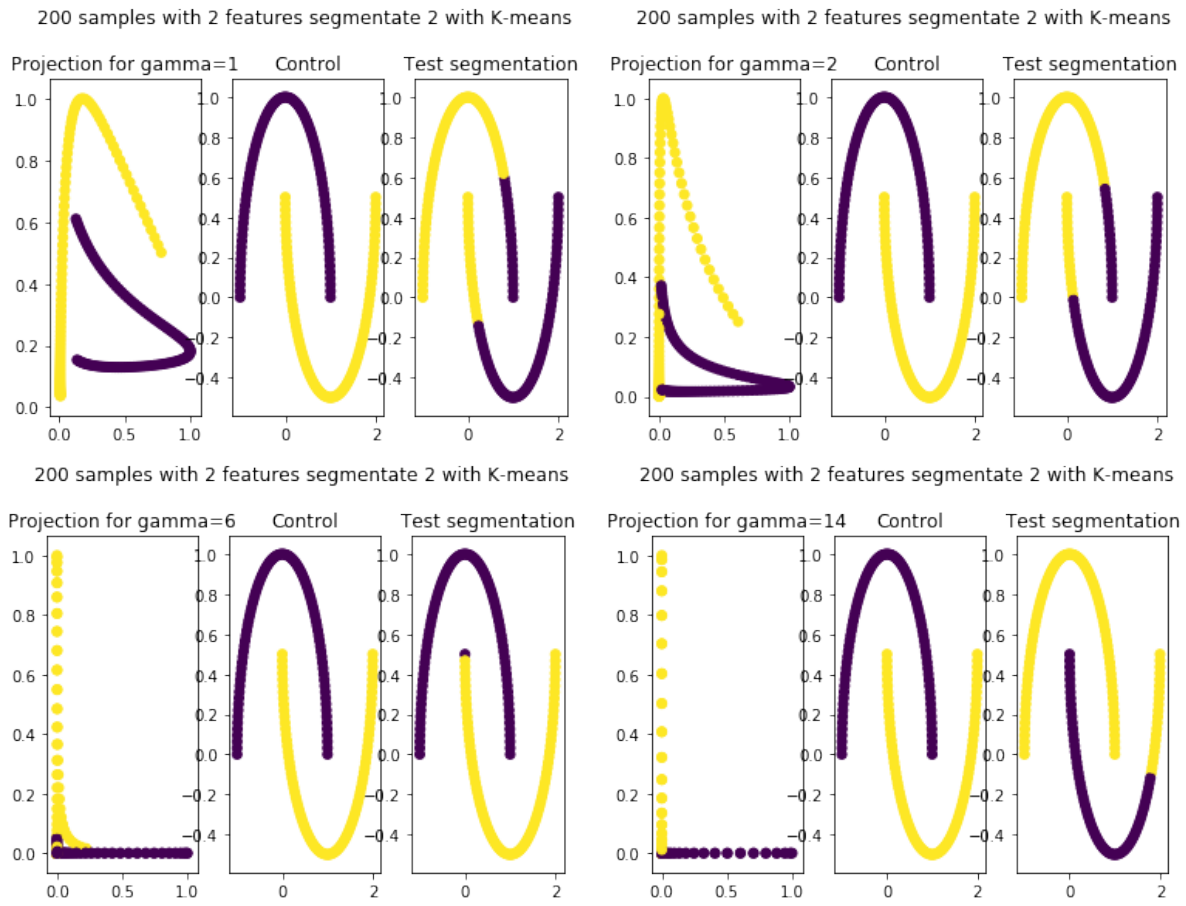

Figure 14: Comparison between different gamma values

Using the labeling of the control group, we can create the equivalent of a loss function $L_{0/1}$. This function returns the percentage of disagreement between the control label and the created label. This allows us to make a more relevant comparison. We can therefore understand why gamma can be considered as a hyperparameter on which we are going to operate in order to get the best possible result.

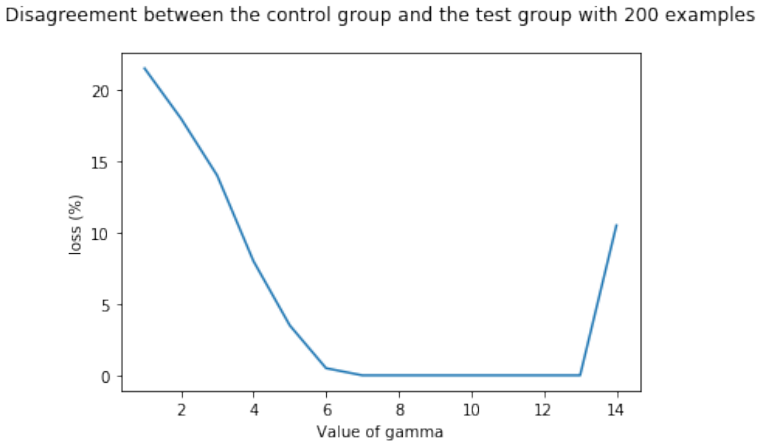

Figure 15: Comparison between different gamma values

2.4 Discuss

We can see that the kmeans algorithm is efficient in segmenting problems linearly. (Figure 11). However, in non-linear cases as in Figure 12, the algorithm is not efficient. We can see that

by kernalising we have a more than satisfying result. By looking in more detail we look for borderline cases where it is more difficult to separate the data. In order to have a satisfying result we test different gamma values.
In Figure 14 we can see that from gamma = 7 we have a good result. One might think that it is enough to put a high gamma value to have the best separability but this is not the case. We can see an increase of errors beyond gamma=13 in this example.
It is interesting in figure 14 to observe that the resolution in the projection space is simpler for the algorithm. We use the labeling from this projection in order to export it in the original space. It is assumed de facto that the order of the examples has not been modified.

Kernelisation in the case of Kmeans allows solving non-linearly separable problems. One must take special care in choosing the right projection kernel. In the case of a Gaussian projection, the gamma value can be considered as a hyperparameter on which one will act in order to have the best possible result. By associating a cost function to it we can make a gradient descent in order to have the best result. However, this supposes that we already have labeled data beforehand, which is absurd.
This semantic analogy is for information, it would be interesting to be able to define a cost function independent of a control group but with the memory of previous tests. It is a path to be explored.

3 One class SVM and Maximum Enclosing Ball

3.1 Problem Statement

One Class Support vector Machine (OCSVM) is a classification algorithm using a single labeled class to determine its zone of influence in a data space. This kind of algorithm allows in particular the detection of anomalies. We implement this algorithm to highlight the role of the kernelisation.

3.2 Theory

An optimization problem can be solved in two ways: by the primal form or the dual form. In the case of the OCSVM this is written

in the primal form: $\begin{cases} min_{R\in\mathbb{R},c\in\mathbb{R}^d}(R^2) \\ \text{with } \forall i \in [\![1,n]\!], \|x_i - c\|^2 \leq R^2 \end{cases}$

in the dual form: $\begin{cases} min_{\alpha\in\mathbb{R}^2}(\alpha^T G\alpha - \sum_{i=1}^n \alpha_i \|x_i\|^2) \\ \text{with } \sum_{i=1}^n \alpha_i = 1 \text{ and } \forall i \in [\![1,n]\!], 0 \leq \alpha_i \leq C \end{cases}$

To find a solution of its two forms, we have to do a gradient descent. In the dual form a constrained or unconstrained resolution is put forward using the constant C, whereas in the primal form this constraint is implicit.

3.3 Results

We chose to solve the dual problem in quadratic form.

Algorithm 1 OCSVM

Input: X, kernel, kernel_parameters ▷ In this case we chose rbf kernel
Output: R ▷ Output: Decision
procedure OCSVM DUAL OPTIMIZATION PROBLEM(X, kernel, kernel_parameters):
$G \leftarrow kernel(X,\ kernel\_parameters)$ ▷ kernelisation
$alpha \leftarrow \begin{cases} min_{\alpha \in \mathbb{R}^2}(\alpha^T G \alpha - \sum_{i=1}^{n} \alpha_i \|x_i\|^2) \\ \text{with } \sum_{i=1}^{n} \alpha_i = 1 \text{ and } \forall i \in [\![1, n]\!], 0 \leq \alpha_i \leq C \end{cases}$ ▷ corresponds to SVDD.fit()
$R \leftarrow G.alpha$ ▷ corresponds to SVDD.decision_function()

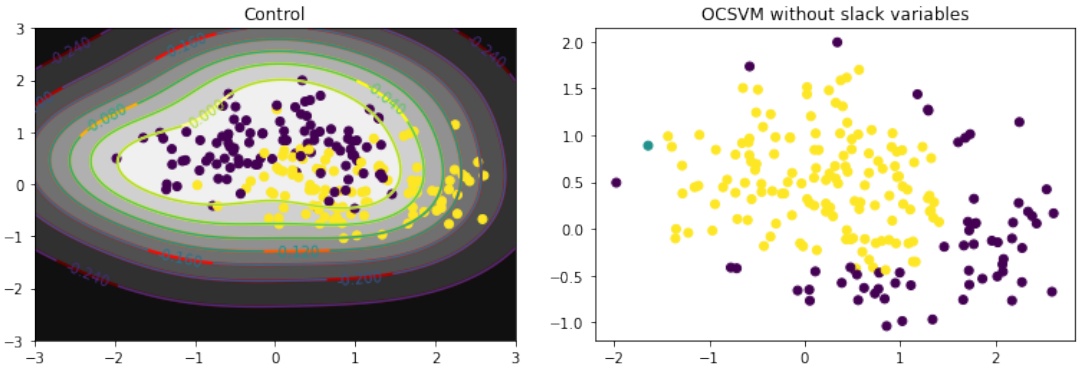

Figure 16: OCSVM without slack variables

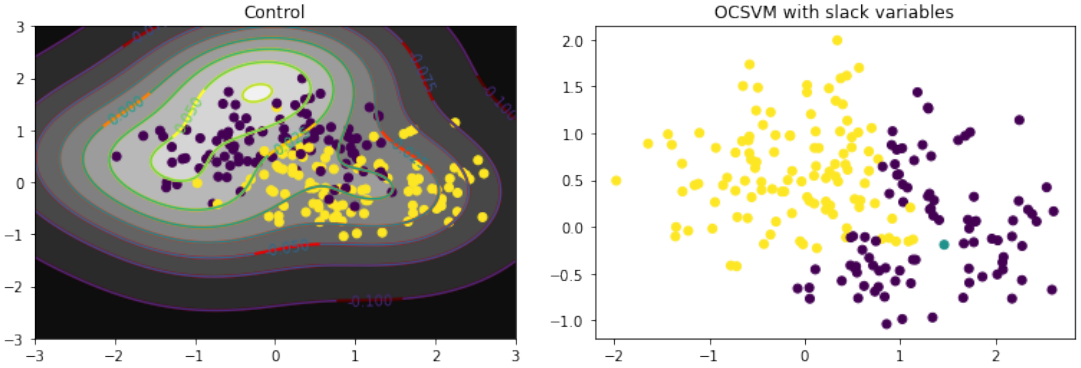

Figure 17: OCSVM with slack variables

The support point used to define the boundary is highlighted in green.

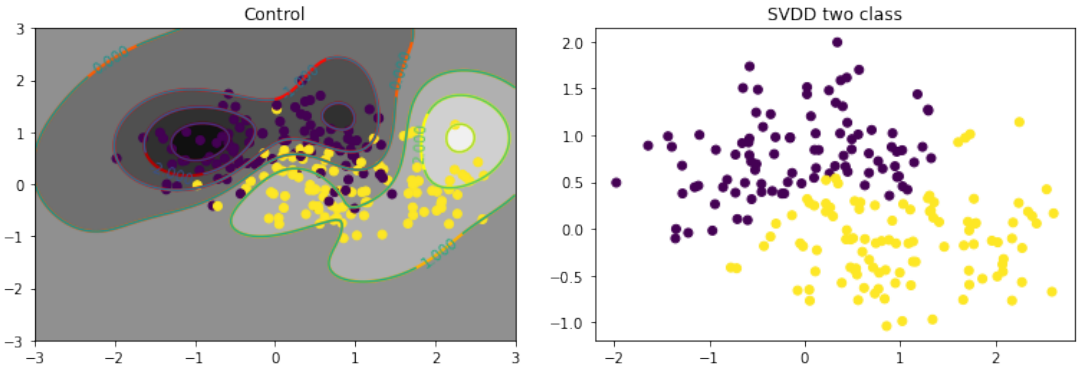

Figure 18: Two Class Support Vector Machine

3.4 Discuss

As can be seen in Figure 16, the image on the left is used to delineate the boundary of the data through the boundary points. If we use variable slacks, we will relax the constraints, i.e. we will allow that points belonging to our data set are not in the boundary, which is observed in the Figure 17. It is interesting to note that we can increase or decrease this boundary by adding a threshold when calculating R for the resolution of the dual form. Depending on the application, it can be useful to overestimate or underestimate the boundary. It can also be seen that the higher the constraint, the less the effect of kernelization is visible. Indeed, we tend towards a circle which is the most economical solution with a strong constraint.
However, this OCSVM method has limitations. Indeed, considering only one label, it is possible to make mistakes when using it in real case. As shown in figure 16 at left and figure at 17 left with a set of 2 labels. The yellow label is present within the borders of the purple labels. It is perfectly normal that the OCSVM algorithm is not able to refine the boundary because it is provided with only one class.
With slack variables we lose precision but you have less recall because the boundary is narrower. One solution is therefore to make a SVM in order to discriminate between the two labels. As can be seen in Figure 18, there is a better demarcation of the boundary between the two labels, and this increases precision while having little recall. Our result would gain in quality with a better parameterization of the parameters. It would be interesting to calculate the accuracy and recall rate according to the different parameters.

Kernelization allows here to separate a non-linearly separable data set with a linear separation tool. To do this, as in the previous cases, we project the data in a higher dimension and we make the separation in this universe. The difficulty in this case lies in the fact that we need to bring the solution back into the space of the initial data in order to display the boundary, compute R.
This costs in computing time but in the case of a direct use of the algorithm for example for the detection of anomalies, we do not need this return, we just need to evaluate the data to be processed and the algorithm returns the label 1 if it is in the boundary and -1 otherwise.

4 Online Learning and SVM

4.1 Problem Statement

Regardless of the performance of the previous algorithms, these need an initial data set to work. From an algorithmic point of view it is not possible to feed them samples by samples. This prevents on the one hand a fast deployment, the time to retrieve a large set of data, and on the other hand the algorithm is less scalable in the future. That's why we are studying online learning algorithms allowing a training example for example or by dataset. They thus offer greater flexibility at first glance.

4.2 Online Passive-Aggressive Algorithms

In this section we study the Online Passive-Aggressive Algorithm. It is important to note that this algorithm is a linear binary classification algorithm. By using a kernelization kernel as seen previously we can solve problems that seem non-linearly separable in higher dimension.

Algorithm 2 OPAA

Input: X, Y, updater, parameter_updater ▷ X: dataset or sample of data, Y: Labeling
Output: w ▷ linear separator ▷ Output: Label
procedure OPAA(X, Y, updater, parameter_updater):
  $w \leftarrow (0,...,0)$ ▷ Initialize
  for $x_t \in X$ and $y_t \in Y$ **do**
    $\widehat{y_t} \leftarrow sign(wx_t)$ ▷ Prediction
    if $\widehat{y_t} != y_t$ **then** ▷ Correct the disagreement
      $loss_t \leftarrow max(0, 1 - y_t(wx_t))$ ▷ Cost function
      $\tau_t \leftarrow updater(loss_t, x_t, parameter\_updater)$ ▷ Deviation
      $w \leftarrow w + \tau.y_t.x_t$ ▷ Correction

We took a set of data and separated it into two sets, one training set of 200 samples, the other test set of 100 samples. This was to be used to study the variance bias of our algorithm. These data are noisy in order to test the influence of the different updaters. Figure 19 is our reference. We can train our algorithm by sample as well as by grouping examples.

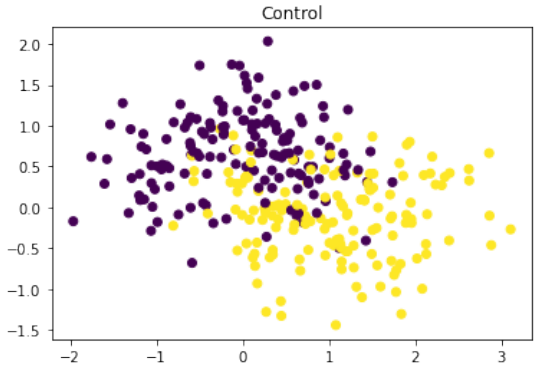

Figure 19: Control

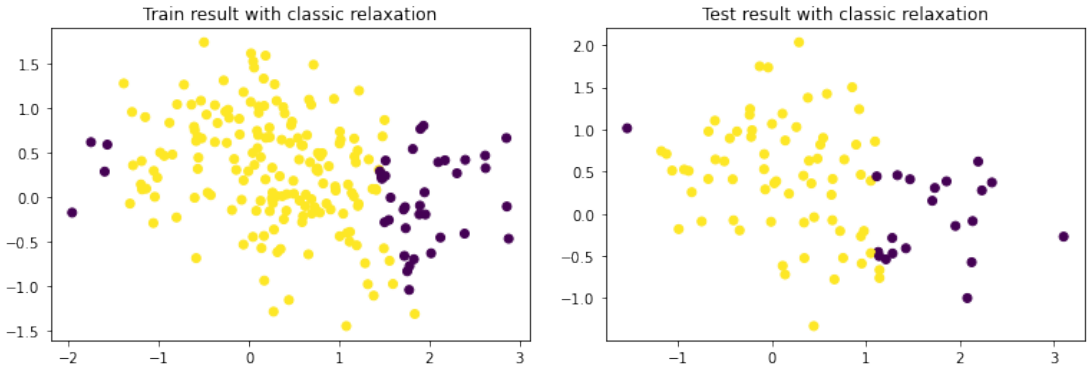

Figure 20: Train and Test results

Regardless of the highly biased results, there is little variance between the training phase and the test phase (Figure 20). We use its different updaters:

- The classic update : $\tau = \frac{loss}{\|x_i\|^2}$
- a first relaxation: $\tau = min(C, \frac{loss}{\|x_i\|^2})$

- a second relaxation: $\tau = \frac{loss}{\|x_i\|^2 + \frac{1}{2C}}$

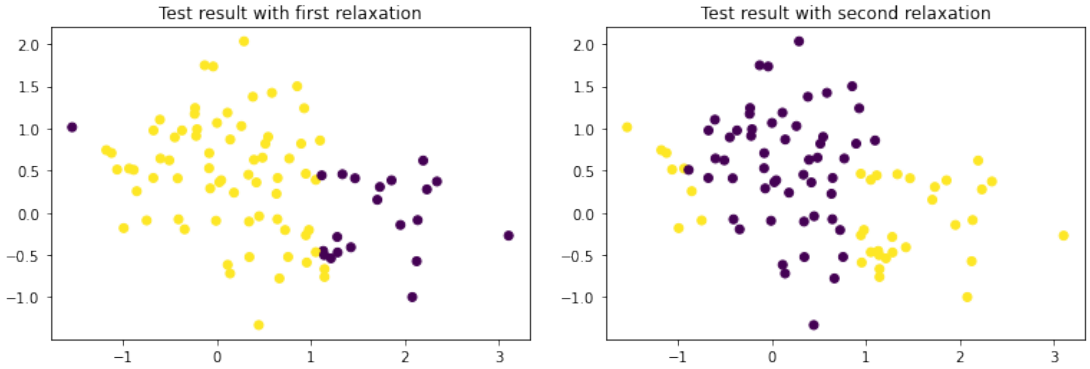

Figure 21: Comparison with first and second relaxation

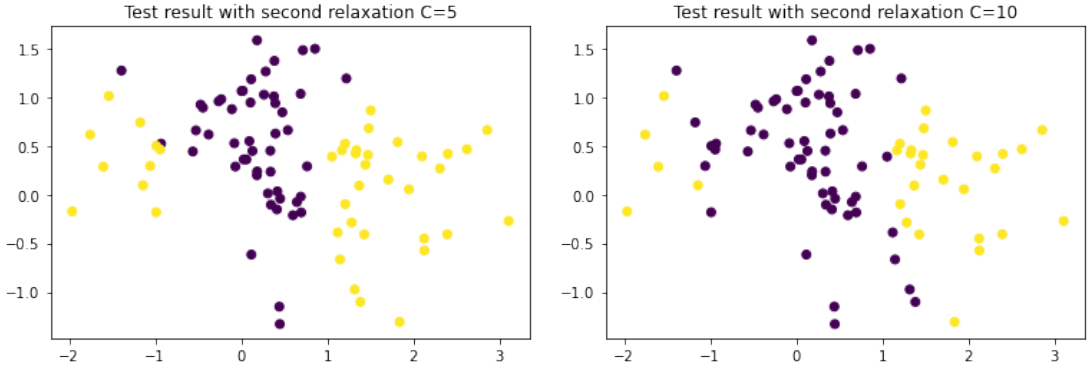

Figure 22: Comparison with different values of C for second relaxation

It is important to note that the value of C does not have the same influence depending on the updater. We can see that when C tends towards infinity we tend towards the classical updater. For low values of C, in the first one we have a threshold that allows the algorithm to be more reactive to change. And in the second we have a finer adjustment of this sensitivity as can be seen in figure 22.

This algorithm allows rapid deployment without the need for a large initial dataset to train it. However with a bad updater, it can be too sensitive, and have stability problems. On the other hand, by the fact that it can be fed sample by sample, it would be more prone to data poisoning because the control of each independent sample is more difficult than an overview of the dataset.

4.3 Incremental SVM

Online learning algorithm for the SVM allows in theory to classify data in two categories by a training sample by sample. We have the choice between two approaches. The SMO approach (algorithm 3) and the LASVM (algorithm 4) approach which are very similar.

Algorithm 3 SMO

Input: x_k, y_k, kernel, kernel_parameters

procedure SMO(x_k, y_k): ▷ for each sample

$\alpha \leftarrow 0$ ▷ Step 1

$g \leftarrow y_k - predict(x_k)$ ▷ compute the gradient

if $\exists(i,j)/\{\alpha_i < B_i \ and \ \alpha_j > A_j \ and \ g_i - g_j > \tau\}$ **then** ▷ Step 2

$\lambda \leftarrow min \frac{g_i - g_j}{K_{ii} + K_{jj} - 2K_{ij}}$ ▷ Step 3

$\alpha_i \leftarrow \alpha_i + \lambda$

$\alpha_j \leftarrow \alpha_j - \lambda$

$\forall s \in \{1, ..., n\}, g_s \leftarrow g_s - \lambda(K_{is} - K_{js})$

Step 4

Return Step 2

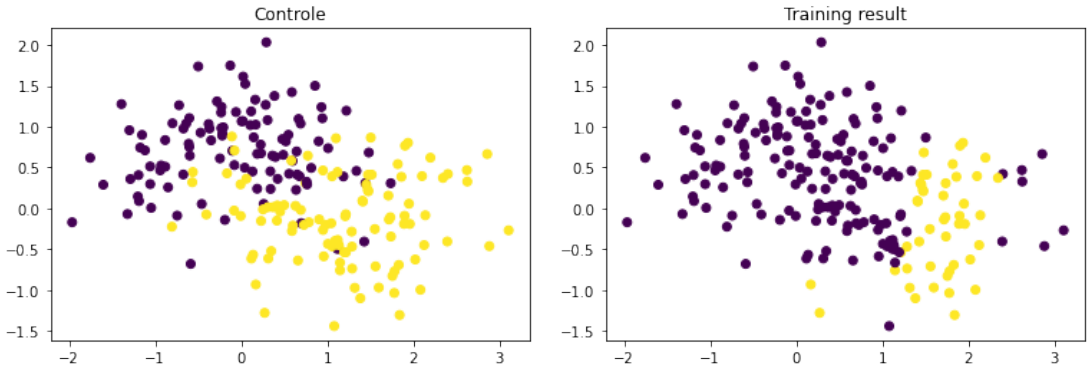

Figure 23: Training time

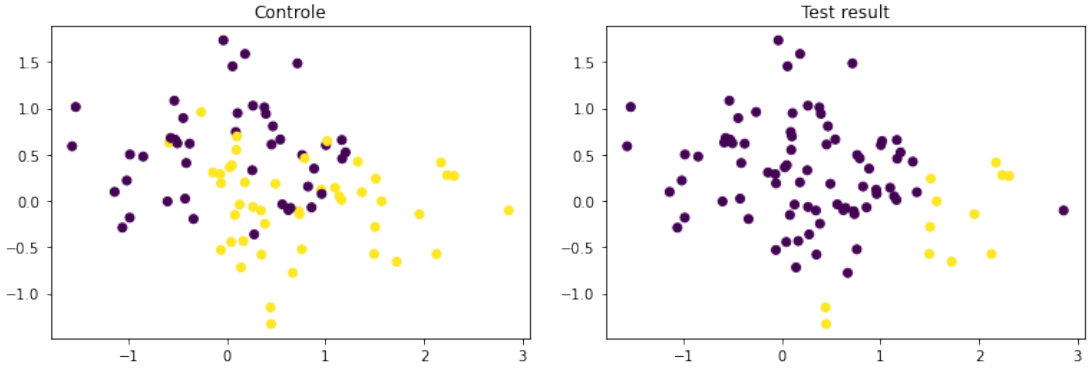

Figure 24: Testing time

A solution that we have considered in order to have a better stability at the start of the learning phases and to allow the algorithm to be fed initially by a whole dataset, and not sample by sample. We should deepen the investigations in this direction and allow a weighting of the learning. In the sense that during an injection of a set of data that the weight of this learning is more preponderant than sample by sample. Indeed, it is assumed that the injection of a dataset imports more significant and less biased information potentially than a single example. Moreover, we note that in the LASVM algorithm we check that the example we are currently processing is not already in the list of examples on which we have learned, this avoids modifying the parameters already updated in case of repetition.

Algorithm 4 LASVM

Input: x_k, y_k, τ, kernel, kernel_parameters

procedure LASVM(x_k, y_k, τ): ▷ for each sample

if $x_k \notin S_x$ **then**

$\alpha_k \leftarrow 0$ ▷ Step 1: Initialization

$g_k \leftarrow y_k - predict(x_k)$ ▷ compute the gradient

$S_{x_k} \leftarrow x_k$ *and* $S_{y_k} \leftarrow y_k$ ▷ Support of learning

if $y_k = 1$ **then** ▷ Step 2: Online Iterations ▷ Step 2.1: Process

if $\alpha_s > A_s$ **then**

$i \leftarrow k$

$j \leftarrow argmin_{s \in S_x}(g_s)$

else

if $\alpha_s < B_s$ **then**

$i \leftarrow argmax_{s \in S_x}(g_s)$

$j \leftarrow k$

if $\exists (i,j) / \{\alpha_i < B_i$ *and* $\alpha_j > A_j$ *and* $g_i - g_j > \tau\}$ **then**

$\lambda \leftarrow min \frac{g_i - g_j}{K_{ii} + K_{jj} - 2K_{ij}}$

$\alpha_i \leftarrow \alpha_i + \lambda$

$\alpha_j \leftarrow \alpha_j - \lambda$

$\forall s \in \{1, ..., n\}, g_s \leftarrow g_s - \lambda(K_{is} - K_{js})$

$i \leftarrow argmax_{s \in S_x}(g_s)$ ▷ Step 2.2: Reprocess

$j \leftarrow argmin_{s \in S_x}(g_s)$

if $\exists (i,j) / \{\alpha_i < B_i$ *and* $\alpha_j > A_j$ *and* $g_i - g_j > \tau\}$ **then**

$\lambda \leftarrow min \frac{g_i - g_j}{K_{ii} + K_{jj} - 2K_{ij}}$

$\alpha_i \leftarrow \alpha_i + \lambda$

$\alpha_j \leftarrow \alpha_j - \lambda$

$\forall s \in \{1, ..., n\}, g_s \leftarrow g_s - \lambda(K_{is} - K_{js})$

$i \leftarrow argmax_{s \in S_x}(g_s)$

$j \leftarrow argmin_{s \in S_x}(g_s)$

for $s \in S$ **do**

if $y_s = -1$ *and* $g_s \geq g_i$ **then**

$S.remove(s)$

if $y_s = 1$ *and* $g_s \leq g_j$ **then**

$S.remove(s)$

$\delta \leftarrow g_i - g_j$

while $\delta \leq \tau$ **do** ▷ Step 3: Finishing

Go to Step 2.2

5 Conclusion

Kernelization allows to export non-linearly separable problems to a higher dimension where it is easier to separate them linearly. It is therefore important to choose your projection kernel carefully in order to have a better representation of the data in this new space. This allows data visualization for example with the PCA agorithm. The computation time for this projection is generally negligible compared to the execution of a classification algorithm, in most cases this operation is done only before the execution of the algorithm for example for Kmeans.
It was legitimately asked whether this method was also functional for online marchine learnign algorithms. In spite of our non-significant results due to badly fixed hyperparameters, it can be observed that it can solve non-linearly separable problems at first glance. We then notice that the kerneling trick can appear as well in the training phase as in the prediction phase and remain totally invisible to the end user.

In a second step, we were asked to set up online learning algorithms and to study them. These algorithms are fundamentally different from the previous ones because they require flexibility to adapt to the new data they have while being stable, i.e. not being too sensitive to the diversity of the data and continuing to learn correctly. Apart from implementation, it is this second aspect that is the most difficult in our opinion.

6 References

7 Table of contents

Contents

1 PCA **1**
1.1 Problem Statement . . . 1
1.2 Theory . . . 1
1.2.1 PCA . . . 1
1.2.2 Kernel PCA . . . 2
1.3 Results . . . 2
1.3.1 PCA . . . 2
1.3.2 Kernel PCA . . . 3
1.4 Discuss . . . 6

2 Kmeans **6**
2.1 Problem Statement . . . 6
2.2 Theory . . . 7
2.2.1 Kmeans . . . 7
2.2.2 Kernel Kmeans . . . 7
2.3 Results . . . 7
2.3.1 Kmeans . . . 7
2.3.2 Kernel Kmeans . . . 8
2.4 Discuss . . . 9

3 One class SVM and Maximum Enclosing Ball **10**
3.1 Problem Statement . . . 10
3.2 Theory . . . 10
3.3 Results . . . 10
3.4 Discuss . . . 12

4 Online Learning and SVM **12**
4.1 Problem Statement . . . 12
4.2 Online Passive-Aggressive Algorithms . . . 12
4.3 Incremental SVM . . . 14

5 Conclusion **17**

6 References **17**

7 Table of contents **18**