Université Jean Monnet

Advanced Machine Learning

Assignment Report

Kernel methods

Author:
Rohil Gupta
Mehdi Zarria

Supervisor:
Amaury Habrard

November 4, 2018

Contents

List of Figures

1 Introduction

Kernel methods owe their name to the use of kernel functions, which enable them to operate in a high-dimensional, implicit feature space without ever computing the coordinates of the data in that space, but rather by simply computing the inner products between the images of all pairs of data in the feature space. This operation is often computationally cheaper than the explicit computation of the coordinates. This approach is called the "kernel trick". Kernel functions have been introduced for sequence data, graphs, text, images, as well as vectors.

2 Different Applications of Kernel Methods

During the assignment we were asked to implement different kernel techniques on different machine learning algorithms from the scratch and compare the results with the standard implementations available in python.

2.1 Kernel Principal Component Analysis

2.1.1 Overview of the method

Principal component analysis (PCA)[2] is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables (entities each of which takes on various numerical values) into a set of values of linearly uncorrelated variables called principal components. Kernel PCA is the nonlinear form of PCA, which better exploits the complicated spatial structure of high-dimensional features by using different kernels by projecting linearly inseparable data onto a higher dimensional space where it becomes linearly separable.

2.1.2 Basic PCA

PCA is mathematically defined as an orthogonal linear transformation that transforms the data to a new coordinate system such that the greatest variance by some projection of the data comes to lie on the first coordinate (called the first principal component), the second greatest variance on the second coordinate, and so on. The figure 1 gives the pesudo code implementation of the Pinciple Component Analysis.

Algorithm 1 Principal Component Analysis

procedure PCA
  Compute dot product matrix: $\mathbf{X}^T\mathbf{X} = \sum_{i=1}^{N}(\mathbf{x}_i - \boldsymbol{\mu})^T(\mathbf{x}_i - \boldsymbol{\mu})$
  Eigenanalysis: $\mathbf{X}^T\mathbf{X} = \mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^T$
  Compute eigenvectors: $\mathbf{U} = \mathbf{X}\mathbf{V}\boldsymbol{\Lambda}^{-\frac{1}{2}}$
  Keep specific number of first components: $\mathbf{U_d} = [\mathbf{u}_1, \ldots, \mathbf{u}_d]$
  Compute d features: $\mathbf{Y} = \mathbf{U_d}^T\mathbf{X}$

Figure 1: Pseudo Code for Principal Component Analysis

2.1.3 Kernel PCA

The basic idea to deal with linearly inseparable data is to project it onto a higher dimensional space where it becomes linearly separable. Let us call this nonlinear mapping function ϕ so that the mapping of a sample x can be written as x$\rightarrow \phi$(x), which is called "kernel function."

Now, the term "kernel" describes a function that calculates the dot product of the samples x under ϕ. The pesudo code for the K-PCA implementation is given in figure 2

$$\kappa(x_i, x_j) = \phi(x_i)\phi(x_j)^T \tag{1}$$

$G_{ij} \leftarrow K(x^{(i)}, x^{(j)})$ for $i, j \in \{1, \ldots, n\}$
$(u_1, \ldots, u_n, \lambda_1, \ldots, \lambda_n) \leftarrow$ spectral decomposition $G = \sum_{l=1}^{n} \lambda_l u_l u_l^H$
procedure FINDDATAPRINCIPALCOMPONENTS
  $\Psi_B(\Phi(x^{(i)}))_j \leftarrow \sqrt{\lambda_j} u_{j_i}$ for $i \in \{1, \ldots, n\}, j \in \{1, \ldots, k^*\}$
  return $\Psi_B(\Phi(x^{(1)})), \ldots, \Psi_B(\Phi(x^{(n)}))$
end procedure
function FINDPRINCIPALCOMPONENTS(x)
  $\Psi_B(\Phi(x))_j \leftarrow \frac{1}{\sqrt{\lambda_j}} u_j^\top \begin{pmatrix} K(x,x^{(1)}) \\ \vdots \\ K(x,x^{(n)}) \end{pmatrix}$
  return $\Psi_B(\Phi(x))$
end function

Figure 2: Pseudo Code for Kernel PCA

2.1.4 Experiment and Results

To test the usefulness of the kernel trick on K-PCA, we used the Swiss Roll data-set to generate with 1200 data-points. The figure of the data-set is given in figure 3

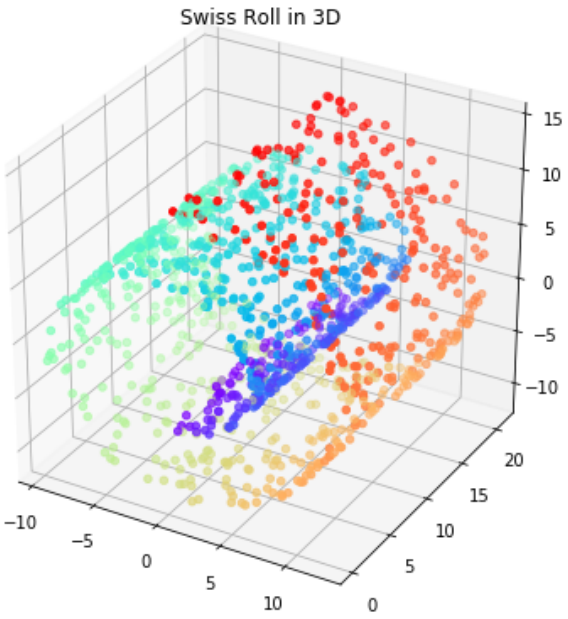


Figure 3: Swiss Roll Dataset

It is clear from the dataset that the data is linearly inseparable and it was seen with the normal implementation of PCA with linear kernel that it did not perform as desired. It is shown in figure 4 and 5, showing respectively the two components and one component after applying linear PCA over the swiss roll dataset.

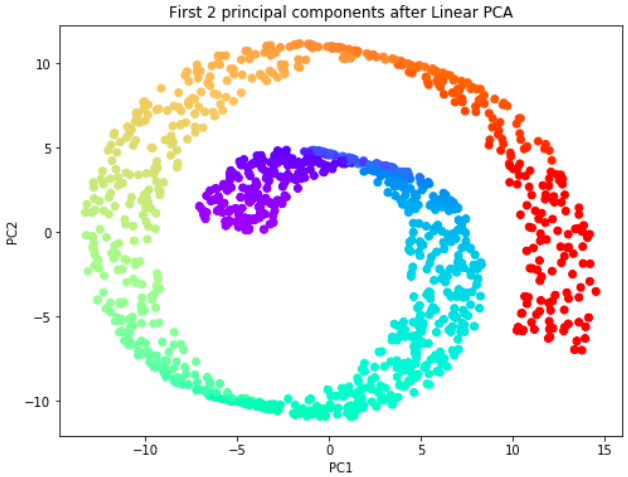


Figure 4: First 2 Components after Linear PCA

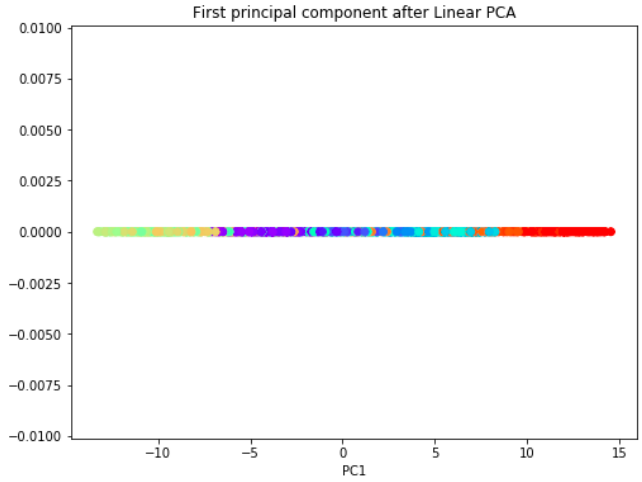
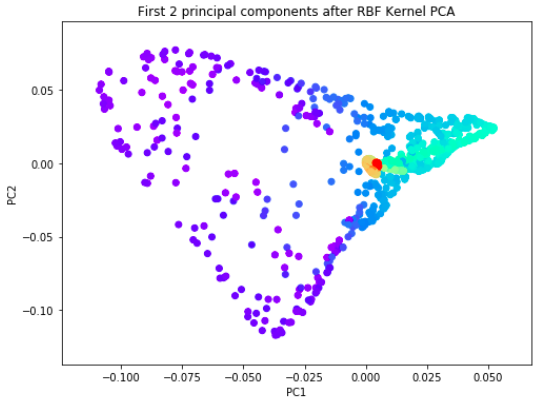


Figure 5: First Principal Component after Linear PCA

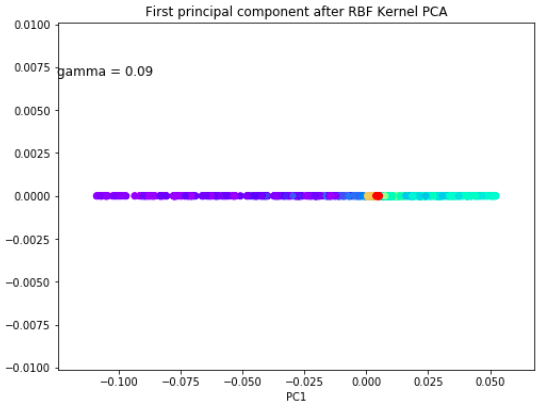
Since, the Linear PCA was not able to perform as desired, it was seen a viable option to apply Kernel methods to project the dataset to higher dimensional space to get the linearly separable porjection of data-points and project them back to normal dimension space and get the linearly spearable dataset.
The implementation of different methods are given below:

1. **Radial basis function (RBF) Kernel:**
   The RBF Kernel was found to be able to linearly separate the Swiss Roll dataset for the gamma values leading to 0.09, this can be seen the figures in figure 6, which shows the first two components in figure 6a and first component in figure 6b of RBF Kernel on PCA.




```
The running time for RBF Kernel with 0.09 gamma is 1.294734001159668 seconds
```

(a) First 2 Components of RBF Kernel



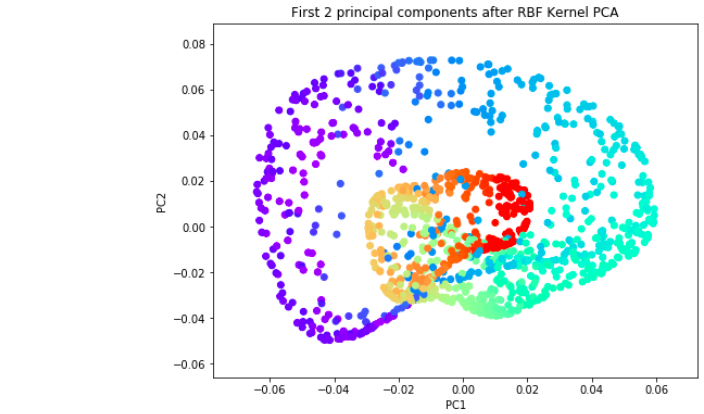

```
The running time for RBF Kernel with 0.09 gamma is 1.3861966133117676 seconds
```

(b) First Component of RBF Kernel

Figure 6: RBF Kernel with gamma value equal to 0.09

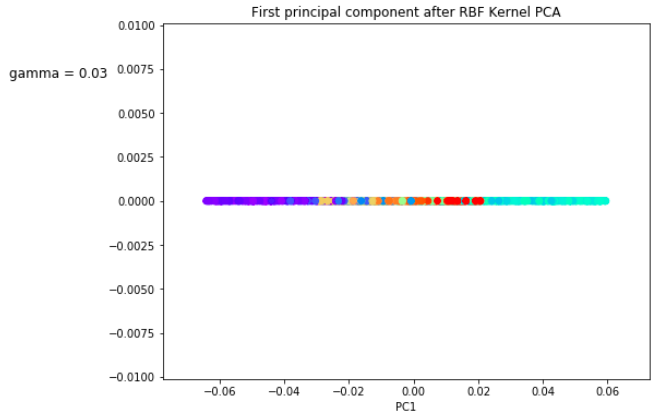
It was also found that if we increase the value of gamma to 0.01, the capability of the RBF Kernel to create a linearly separable projection was lost. This shows that as the value of gamma value decreases the

ability to linearly separate the data-points diminishes. This can be seen the figure 7 clearly with gamma value equal to 0.03. Further results can be seen in the Jupyter Notebook.




```
The running time for RBF Kernel with 0.03 gamma is 1.2614893913269043 seconds
```

(a) First 2 Components of RBF Kernel




```
The running time for RBF Kernel with 0.03 gamma is 1.3717920780181885 seconds
```

(b) First Component of RBF Kernel

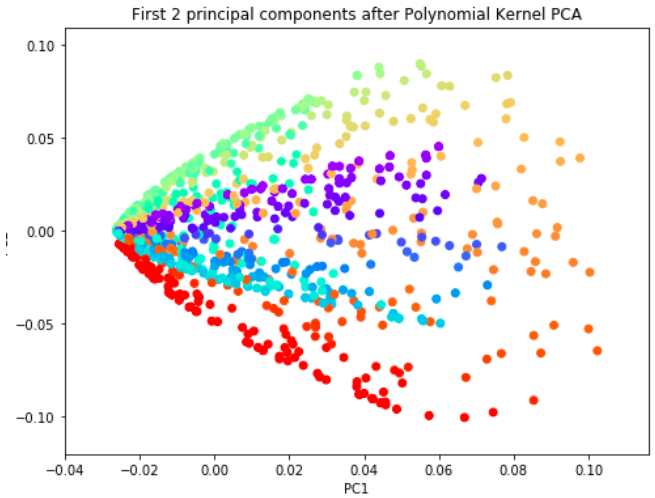
Figure 7: RBF Kernel with gamma value equal to 0.03

2. **Polynomial Kernel:**
   The polynomial kernel performed reasonably well with different polynomial constant value. The Polynomial Kernel trick was found to be also able to separate the Swiss Roll Dataset for different polynomial

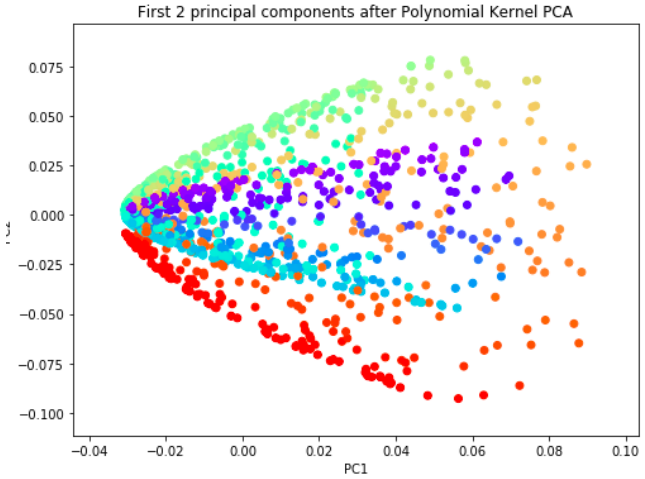
constant value.

It can be seen in the figure 8 the implementation of the Polynomial Kernel on Swiss Roll Dataset.The figure 8a shows the Swiss Roll transformed into the representation in which the data-points which were once lineraly inseparable can now be separated with the polynomial constant value equal to 0.01. As the value of polynomial constant increase the ability of the Polynomial Kernel to separate the point decreases. It can be clearly seen in figure 8b and 8c.

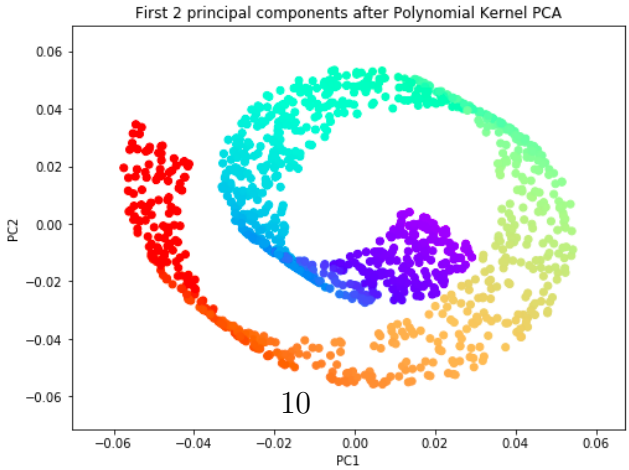
It can be seen in figure 8c that for a very large value of polynomial constant, in this case 10000, polynomial kernel behaves like Linear Kernel PCA as shown in figure 4.



(a) Polynomial constant value equal to 0.01



(b) Polynomial constant value equal to 0.1

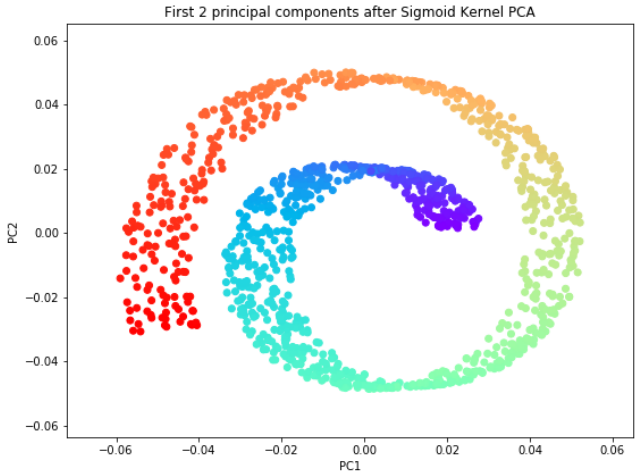


(c) Polynomial constant value equal to 10000

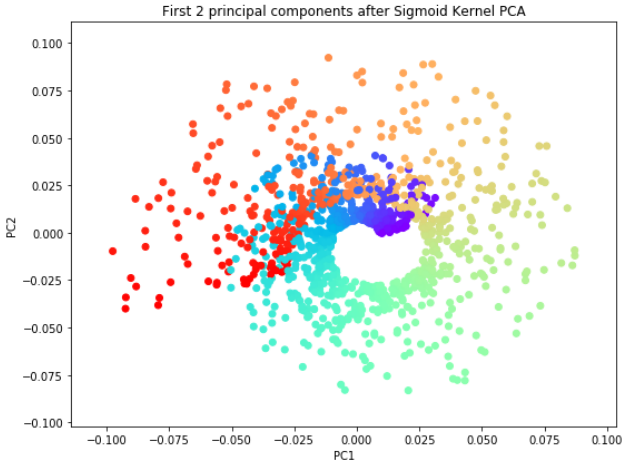
Figure 8: First 2 components of Polynomial Kernel PCA with different values of polynomial constant

3. **Sigmoid Kernel:**
   The Sigmoid Kernel performed the best, being able to transform the Swis Roll Dataset to a linearly separable projection where an classifier can easily learn the data-points.

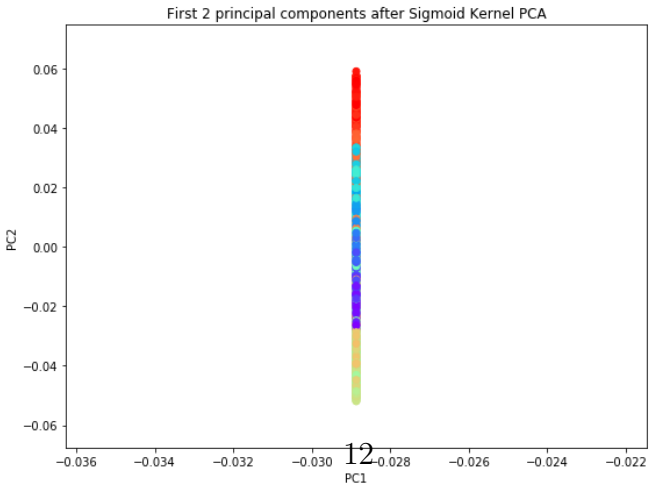
   The figure 9 shows the different projections obtained after varying the values of sigmoid constant and the alpha values. It can be seen from figures 9a, 9b and 9c that as the value of value of Sigmoid Constant value increases and the value of alpha decreases to a very small value, we obtain a projection which is linearly separable, as shown in figure 9c. Thus, confirming the fact that the small value of Sigmoid alpha and large value of Sigmoid constant is favourable for the it's implementation.



(a) Sigmoid constant value equal to 1 and alpha 0.0001



(b) Sigmoid constant value equal to 1 and alpha 0.01



(c) Sigmoid constant value equal to 10 and alpha 0.0001

Figure 9: First 2 components of Sigmoid Kernel PCA with different values of sigmoid constant and alpha.

2.1.5 Conclusion

- The Kernel method for PCA is good way to transform the linearly inseparable data to a projection where a linearly separable transformation is obtained and this dataset can be fed to a classifier for modelling.
- It was found that Sigmoid Kernel and RBF Kernel produce very good results to obtain good transformation.
- The obtained results were compared with the standard implemetation of Kernel PCA available in sklearn library and it was to be consistent with it.
- It was also found out that the polynomial kernel is slow and takes more time to obtain the transformation compared to other two kernel methods.

2.2 Kernel K-Means

2.2.1 Overview of the method

The k-mean algorithm is a machine learning approach used to detect groups in a given data set, it is generally used for unsupervised tasks. The K-means algorithm is an iterative method leading to split the data set into a given number of groups that share certain characteristics.

2.2.2 Basic K-Means

For a K-mean algorithm we need a data set which is not labeled and the number K (the number of cluster to which the data set will be split), the main question of this algorithm is how to choose the number K. first of all, we will explain briefly how the K mean works, and then how we can choose the number K. As we told, the k-means algorithm aims to split the data set into K groups, and for each group we calculate a centroid to represent it. As we told, the K-mean algorithm is an iterative approach aiming to split the data set into K groups. To train this algorithms we follow the pseudo code given bellow:

The algorithm is separated into two major parts: Data assignment and centroid update, for each iteration we calculate the distance between the

Algorithm 1 Basic K-Means

1: *Input*: Dataset, K (number of clusters)
2: *Output*: A set of K clusters
3: **procedure**
4: *Choose arbitrarily K object from the data set*
5: *loop*:
6: *For each example in dataset:*
7: $example[cluster] \leftarrow \underset{c_i \in C}{argmin} dist(example, c_i)^2$
8: *recalculate the new centroids using the mean equation*

example and the different centroids representing the clusters, then we assign the example to the cluster corresponding to the minimum distance $\underset{c_i \in C}{argmin} dist(example, c_i)^2$. For the distance, it can be the Euclidean distance or any other type of distances, it depends on the data to cluster, for example, to cluster the text data we can use the edit distance.
After the assignment of each example to the corresponding cluster, we proceed to update the centroids representing the cluster, in order to take account the new examples. In general we use barycenter of the data in the cluster

$$c_i = \frac{\sum_{x_i \in c_i} m_i * x_i}{\sum m_i} \qquad \text{where } m_i \text{ is the weight of } x_i \tag{2}$$

In the general cases the dataset is iid, the weight of all the examples is the same and equal to 1. In that case (2) is simplified to

$$c_i = \frac{1}{\mid c_i \mid} \sum_{x_i \in c_i} x_i \tag{3}$$

2.2.3 Kernel K-Means

The kernel approach for the K-mean [1] algorithm is used to classify data sets that cannot be classified using balls. For example, we can imagine a data set of circular shape as it shown in figure 10, it is clear that we have two cluster, 1: the central data in green, 2: the data in the circumference. By using a simple K-mean approach, the two calculated centroids will superimpose, and

that will lead to miss-classification of the examples.
For the previous reason, we use Kernel approach, to take our data to high dimension space, and run the classical K-mean in the new space. In this paragraph we will show theoretically how to use the kernel methods in the K-mean algorithm.
In the previous paragraph, we used the $\underset{c_i \in C}{argmin} dist(example, c_i)^2$ to assign the example x_i to a given cluster. The objective is to find a kernalized written to this formula.

$$\begin{aligned} dist(x_i, c_i)^2 &= (x_i - c_i)^\top (x_i - c_i) \\ &=< x_i, x_i > -2 < x_i, c_i > + < c_i, c_i > \\ &= K(x_i, x_i) - 2K(x_i, c_i) + K(c_i, c_i) \end{aligned} \tag{4}$$

For the sake of simplicity and reducing the time complexity, we can split the previous equation into three parts:
$1 :$ dist1 $= K(x_i, x_i)$ the kernel function applied to the given example
$2 :$ dist2 $= K(x_i, c_i)$, the minimum values obtained by applied the kernel function to the example and the different centroids
$3 :$ dist3 $= K(c_i, c_i)$, the kernel function applied to the centroids
The pseudo code used in this case will be the following:

Algorithm 2 Kernel K-Means

Input: Dataset, k (number of clusters), the kernel K
Output: A set of K clusters
procedure
    Choose arbitrarily k object from the dataset
loop:
    $dist3 \leftarrow [K(c_i, centroids)$for each c_i in centroids$]$
    For each example in dataset:
        $dist1 \leftarrow K(example, example)$
        $dist2 \leftarrow [K(eample, c_i)$for each c_i in centroids$]$
        $example[cluster] \leftarrow \underset{c_i \in C}{argmin}(dist1 - 2dist2 + dist3)$
    recalculate the new centroids using the mean equation

2.2.4 Experiment and Results

To test the usefulness of the kernel tricks on K-Mean algorithm, we used the circle dataset, with 400 examples as it shown in figure 10

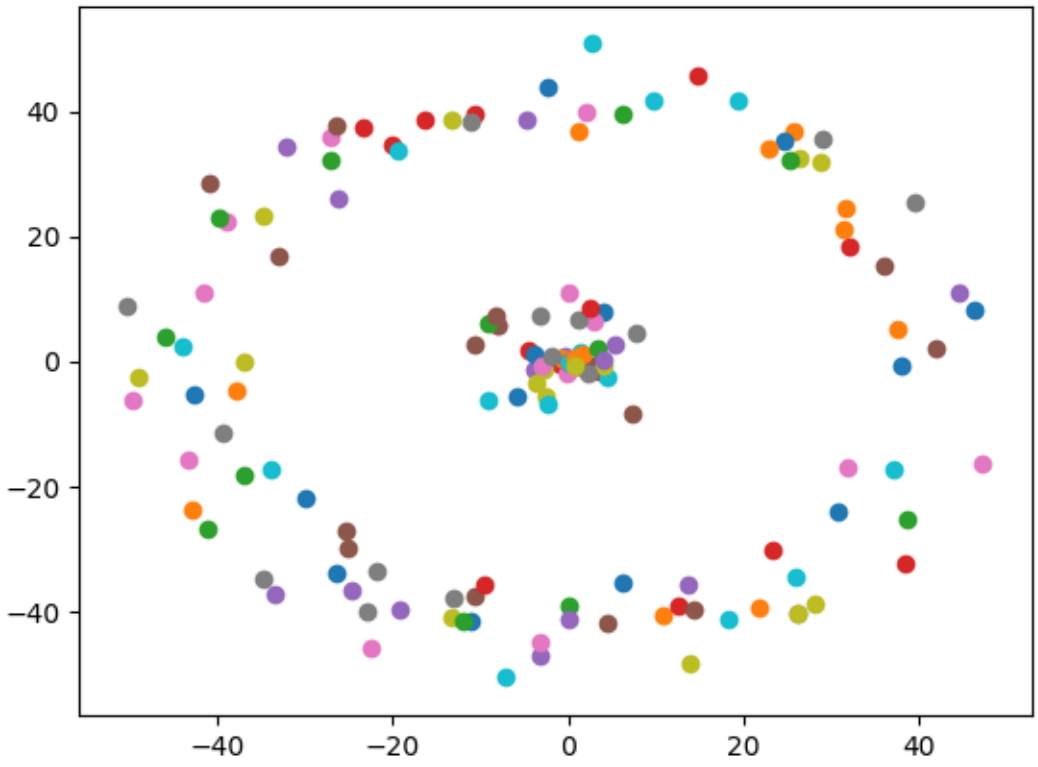


Figure 10: Circle Dataset

It is clear that this dataset has two cluster, for K-Mean algorithm we can test also the number of cluster, but in this project we focused on the use of kernel tricks, for this reason we will escape the part of finding the optimal number of clusters.

In the beginning we tried to classify the dataset without using any kernel, the result we got is shown in the figure bellow, we can remark that the K-

Mean algorithm converge to superposed centroids, that is because our dataset is circular (one center). For this reason, we thought that this dataset will illustrate the usefulness of the kernels

After that we used the RBF kernel defined by the following equation

$$K_\sigma(x, x') = \exp - \frac{\| x - x' \|^2}{2\sigma^2} \tag{5}$$

To tune this kernel we used several values of sigma in order to obtain the one leading to the better classification.

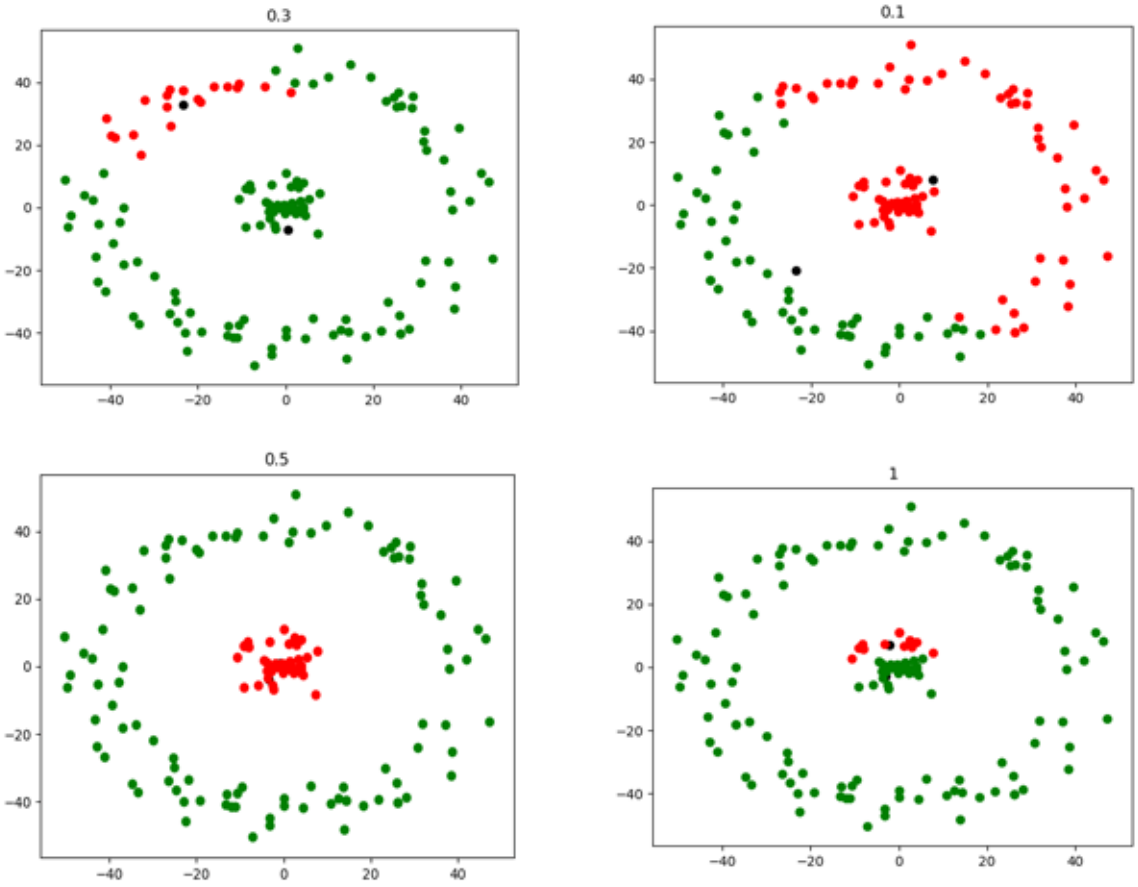


Figure 11: RBF Kernel

From the previous figure we can notice that the best value of sigma in

our case is 0.5. It leads to the perfect separation between the two cluster and with an accuracy of 100%.

For the second experiment, we tried the polynomial kernel which is defined by the following equation:

$$K_{c,d}(x, x') = (x^\top x' + c)^d \tag{6}$$

For this kernel we tried to search for the polynomial degree that leads to a good separation, and after that we tuned the penalty term which the constant c. Bellow the picture representing the different results in function of the degree, by taking c = 0.

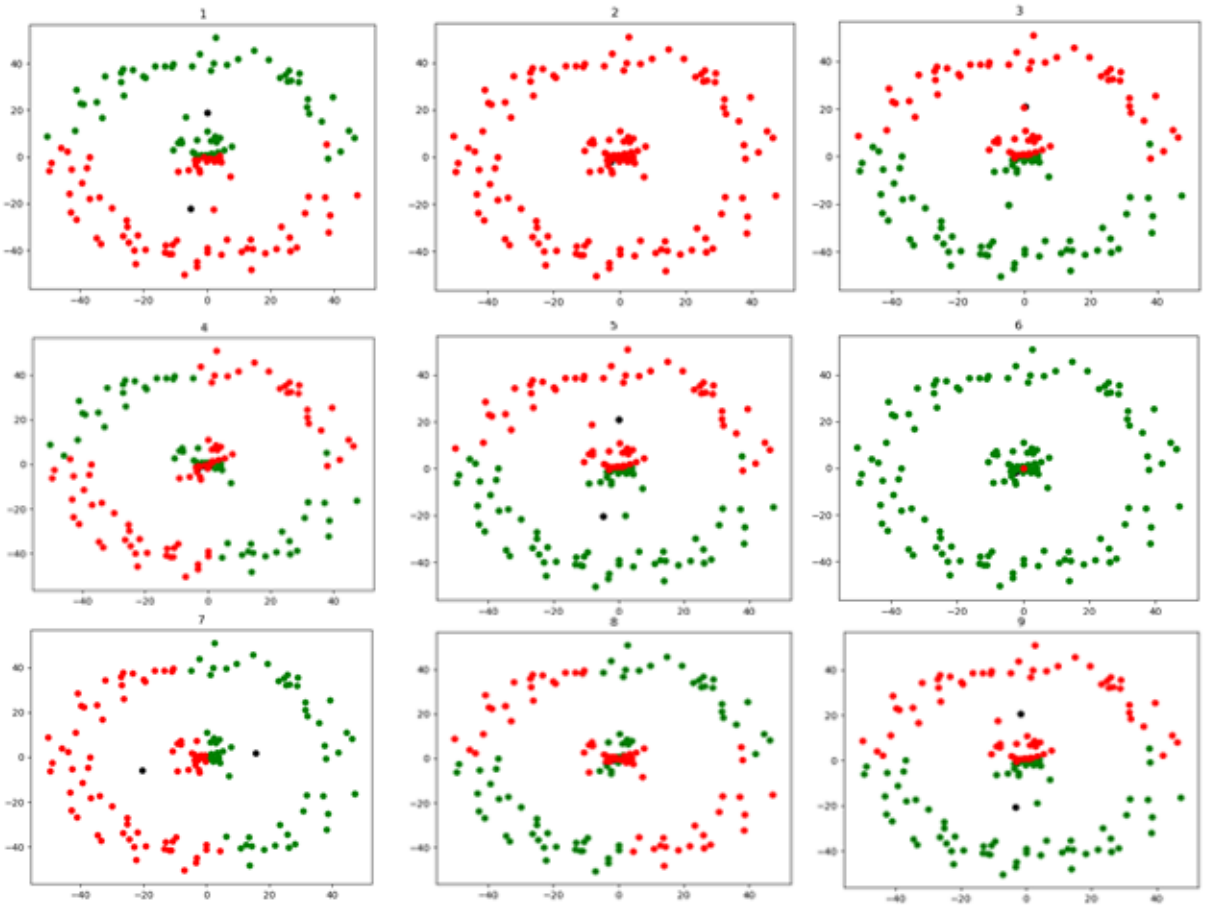


Figure 12: Effect of polynomial degree on the classification

The thing that we can notice, is that the parity of the degree has a role on the classification, that is to say, for the odd degree, we always have a linear separator on the projected space, instead of the even degree the classification take an elliptic shape. For the second part we will use degree 2, because is the smallest even degree given an elliptic shape, and we will try to tune the parameter c(penalty term), in order to force the classifier to better classify our dataset.

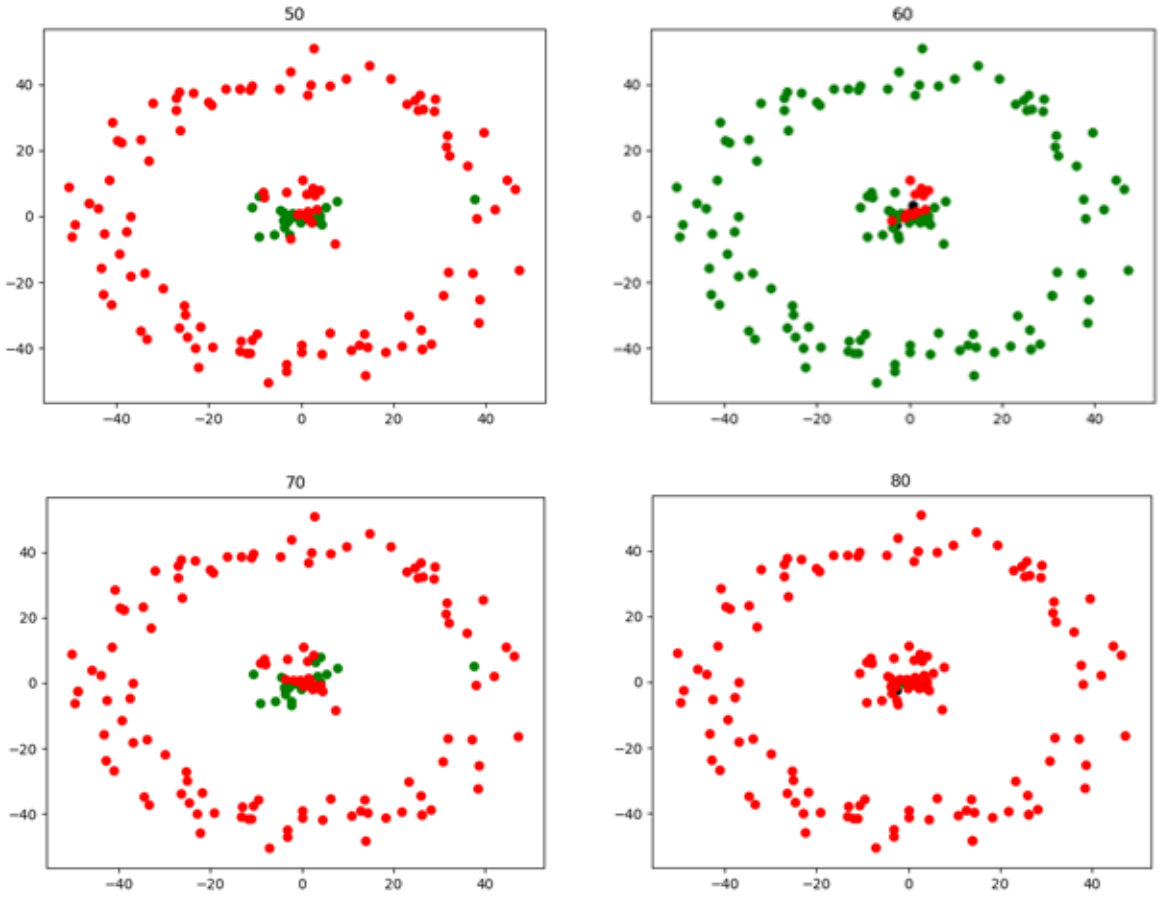


Figure 13: Effect of constant on the classification 1

As we can notice the values 50, 60 and 70 give a good representation, even if the accuracy is not promising, for this reason we tried to deep more in these values and take range from 60 to 70, the figure 16 illustrate that the

best value of the parameter c we could find is 65.

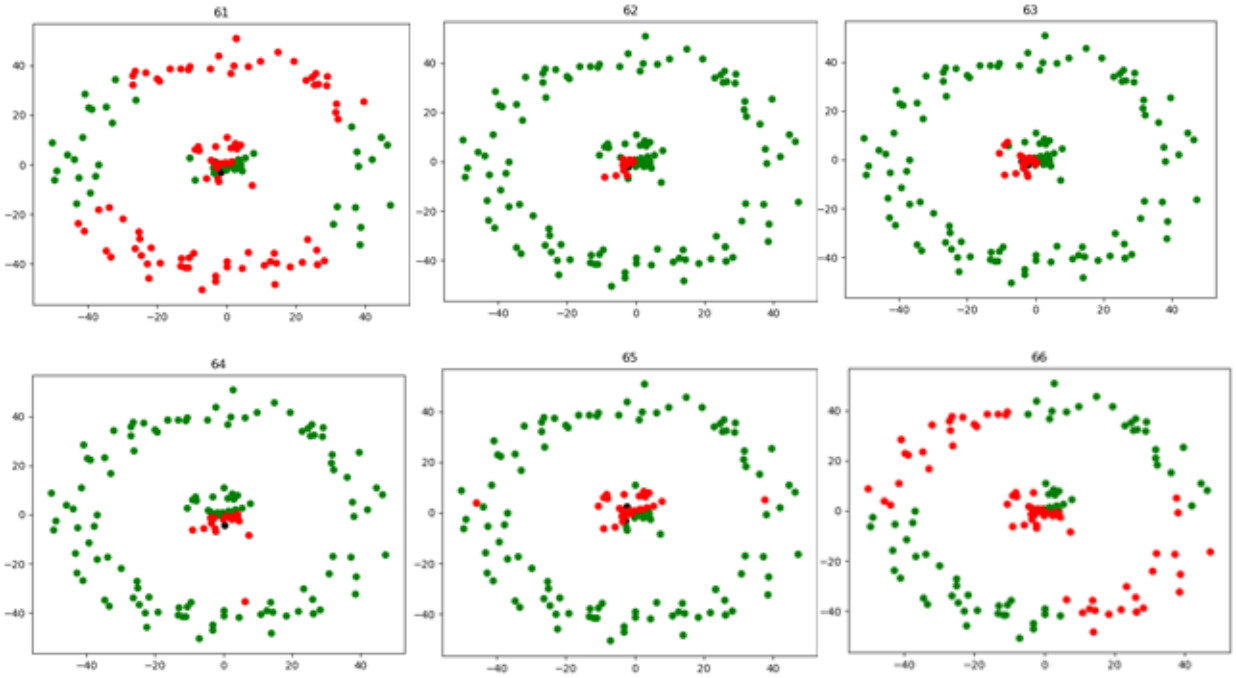


Figure 14: Effect of constant on the classification 2

Finally, we used the sigmoid kernel defined by the following equation

$$K_{\alpha,c}(x, x') = \tanh\left(\alpha x^\top x' + c\right) \tag{7}$$

Using the same procedure as the polynomial kernel, we fixed at the beginning the constant c to the value 0, and we tried several values of α. In the figure bellow the different results we got through this experience.

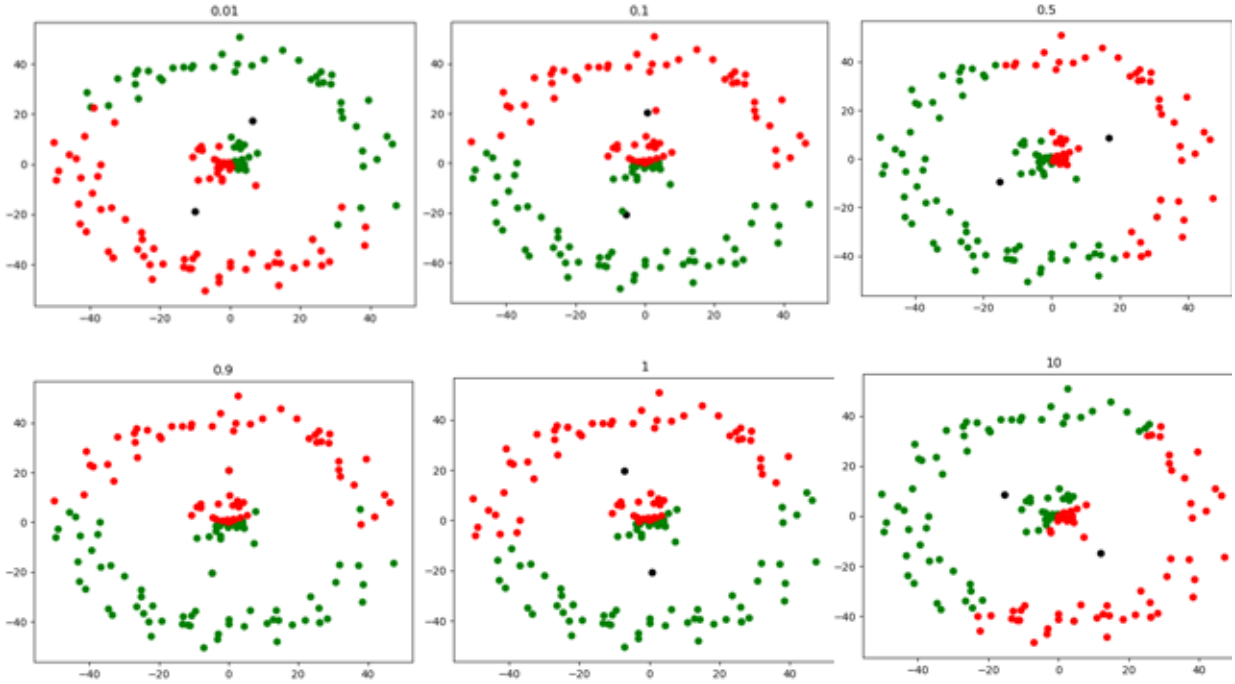


Figure 15: Effect of sigma on the classification

Even with using several values of sigmas we notice that the classification is far from given the best classification. For this purpose, we proceed by testing the effect of the constant on the classification, note that the constant c refers to the penalty term. For this experience we used as sigma parameter the value $\frac{1}{|dataset|}$

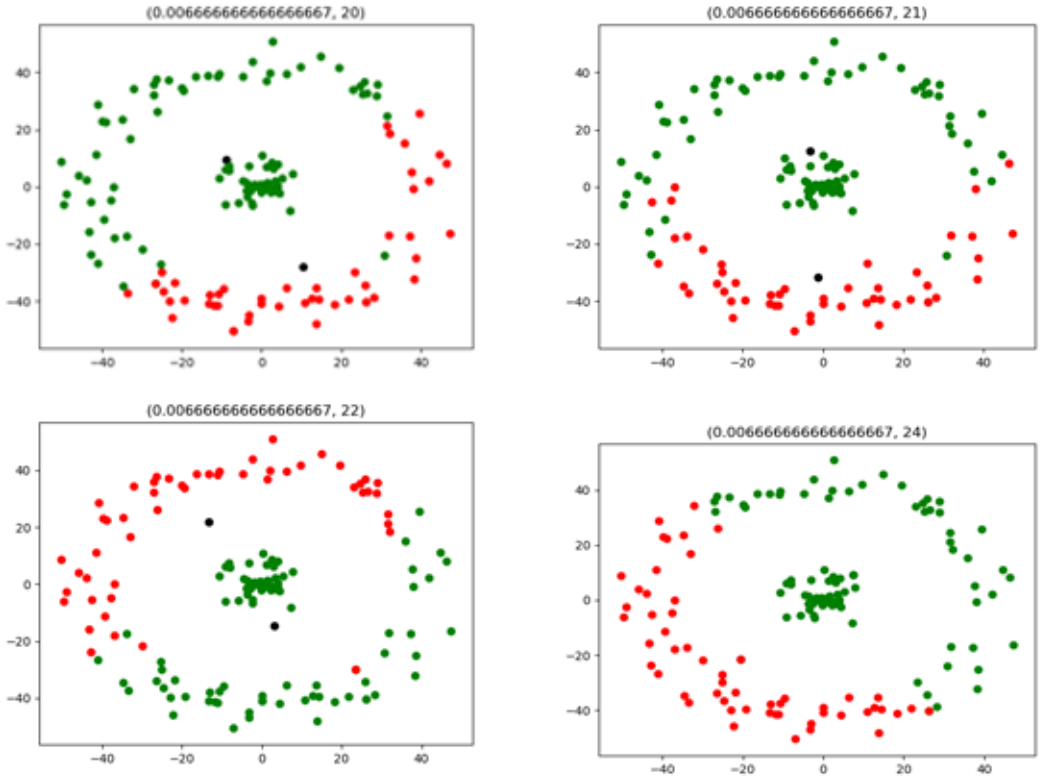


Figure 16: Effect of the constant on the classification

The result above was the best one we could get using the sigmoid kernel.

2.2.5 Conclusion

The K-Mean algorithm is a good approach to classify non labeled data, and the use of kernel can help it to classify more different data distributions by taking theme to high dimensional space.
The complexity of kernelized K-Mean is n^3, that is to say, for huge detasets it will takes long time to train, and in that case it could be more suitable to use the kernelized SVM.
Last thing about K-Mean is that the choice of the initial centroids has an effect on the final result, for this reason a person should think about the choice of these centroids instead of a random choice in order to lead to an optimal solution.

2.3 Kernel Logistic Regression(LR)

2.3.1 Overview on the method

The logistic regression method leads to calculate the probability of an outcome. Generally the LR algorithm can be used to predict k classes of a given example, but practically, we used for binary classification. In this part we will concentrate on a binomial distribution using the Logistic Regression method.

2.3.2 Basic LR

[3] The Logistic Regression uses a linear model to predict probability of an example, in that case we benefit from:

- The calculated value $\in$ [0,1]
- The simplicity of the linear regression algorithm

We recall that a the logitic function is defined by

$$P(Y|X = x) = \frac{1}{1 + \exp^{-(x^\top \beta + \beta_0)}} \tag{8}$$

is the probability to get the value 1 given the example x. As we considerate our problem is a binomial distribution, we can say that the objective is to maximize the likelihood of our model. The likelihood for a binomial distribution is defined as:

$$l(\beta, Y) = \prod_{x_i \in X} p_{x_i}^{y_i} (1 - p_{x_i})^{1-y_i} \tag{9}$$

As the probability function is exponentially defined, we can introduce the logarithmic function into the likelihood, this new notion is named the logit likelihood, and defined as following

$$\begin{aligned} L(\beta, Y) &= \ln l(\beta, Y) \\ &= \ln \prod_{x_i \in X} p_{x_i}^{y_i} (1 - p_{x_i})^{1-y_i} \\ &= \sum_{x_i \in X} y_i \ln (p_i) + (1 - y_i) \ln (1 - p_i) \end{aligned} \tag{10}$$

The usefulness of this procedure is to simplify the calculation of the derivative function. From now on we will use the logit function for all the theoretical calculus.
The likelihood calculated in equation 10 does not have a canonical form to be maximized, for this reason we should use approximated methods, it turns that the logit function is a concave, so it has a global maximum, we can use the Newton method to calculate this optimal solution.
For this reason we use the SGD formula, we will go deeper in its proof in the Kernel part.

$$\begin{aligned} \beta_{n+1} &= \beta_n - \alpha \frac{\partial L}{\partial \beta} \\ &= \beta_n - \frac{\alpha}{|X|} x^\top (y - \hat{y}) \end{aligned} \tag{11}$$

The algorithm used for a Basic LR is the following:

Algorithm 3 Basic LR

1: *Input*: Dataset, α
2: *Output*: Classifier h
3: **procedure**
4: *initialize the vector* β
5: *loop till convergence*:
6: $z \leftarrow x^\top \beta$
7: $h \leftarrow sigmoid(z)$
8: $\beta \leftarrow \beta - \alpha \frac{1}{|X|} x^\top (y - h)$

2.3.3 Kernel LR

[5] In this section we will go deeper in mathematical proofs of different equations of the LR algorithm, and at the end we will show how we can kernelize the LR approach.
Note that the probability of a point x, can be calculated using the sigmoid function.

$$P(Y|X = x) = sigmoid_{\beta,\beta_0}(x^\top \beta + \beta_0) \tag{12}$$

and the derivative of sigmoid function is

$$\frac{\partial sigmoid}{\partial \beta} = sigmoid(1 - sigmoid) \tag{13}$$

Using the equations 11, 12 and 13, we can calculate the derivative of the logit function.

$$\begin{aligned}\frac{\partial L(\beta, Y)}{\partial \beta_j} &= \frac{\partial}{\partial \beta_j} \sum_{x_i \in X} y_i \ln(p_i) + (1 - y_i) \ln(1 - p_i) \\ &= \sum_{x_i \in X} y_i \frac{\partial \ln(p_i)}{\partial \beta_j} + (1 - y_i) \frac{\partial \ln(1 - p_i)}{\partial \beta_j}\end{aligned} \tag{14}$$

Using the following equation $\frac{\partial p_i}{\partial \beta_j} = x_{ij} p_i (1 - p_i)$ in 14, we get

$$\begin{aligned}(14) &= \sum_{x_i \in X} y_i x_{ij}(1 - p_i) - x_{ij}(1 - y_i) p_i \\ &= \sum_{x_i \in X} x_{ij}(y_i - p_i) \\ &= X_j^\top (Y - \hat{Y})\end{aligned} \tag{15}$$

Where X_j is the jth column of the data matrix and $\hat{Y}$ is the calculated probabilities. At this point we explained the theoretical study of the basic Logistic Regression algorithm, and how to optimize the likelihood using the SGD method.
Now we will show how we can kernelize the LR algorithm. From the equation 15 the optimal solution to maximize the likelihood is $\sum_{x_i \in X} x_{ij}(y_i - p_i)$ for every β_j, by using this value in the probability formula, we obtain

$$\begin{aligned}P(Y|X = x) &= (1 + \exp(-\sum_{x_i \in X} \sum_{x_i \in X} x_i^\top x_j (y_i - p_i)))^{-1} \\ &= (1 + \exp(-\sum_{x_i \in X} \sum_{x_i \in X} K(x_i, x_j)(y_i - p_i)))^{-1}\end{aligned} \tag{16}$$

Now we are able to write a kernelized version of the logistic regression, as all the other formulas mentioned before are written in function of the probability formula.
The pseudo code describing the Kernel LR is the following:

Algorithm 4 Kernel LR

Input: Dataset, α , Kernel type
Output: Classifier h
procedure
    initialize the vector β
    initialize the Kernel matrix$K$
loop till convergence:
    $z \leftarrow K\beta$
    $h \leftarrow sigmoid(z)$
    $\beta \leftarrow \beta - \alpha \frac{1}{|X|} K^{\top}(y - h)$

For the Kernel functions we used the same defined in the previous parts.

2.3.4 Experiment and results

Using our implementation of the simple and the Kernel logistic regression, we lead some experiments to verify the usefulness of the kernelized LR on dataset representing an ambiguous data distribution. in the first part we used the circular dataset.

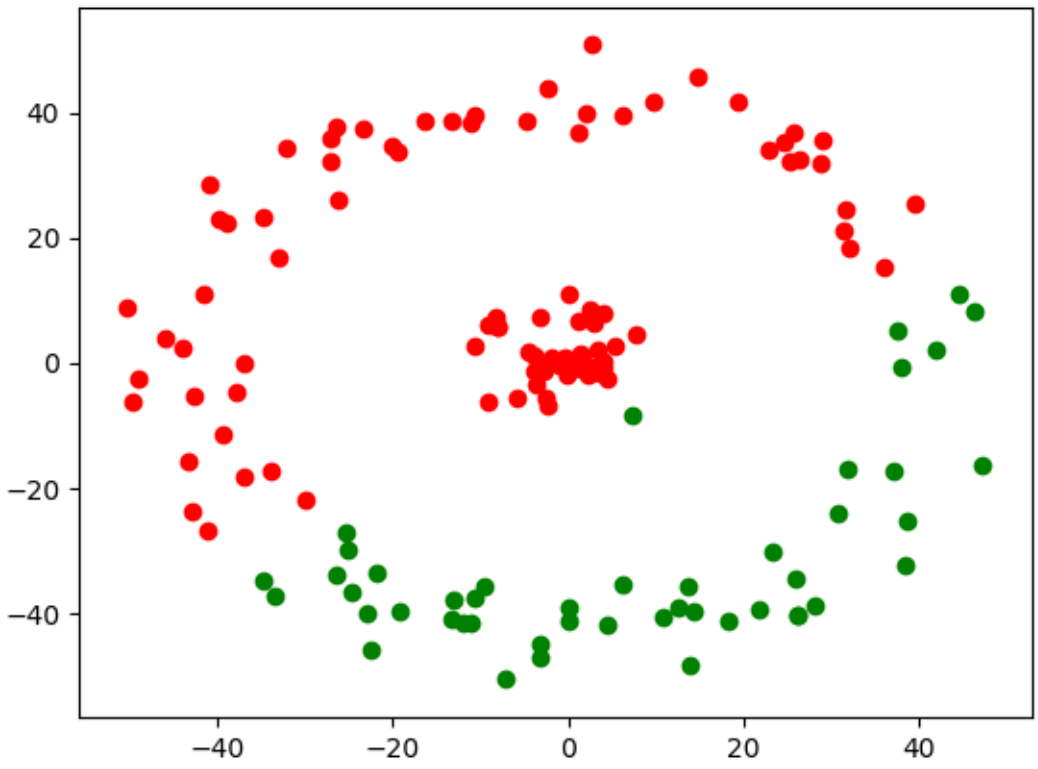


Figure 17: Basic LR classifier on circle dataset

We notice that the basic LR cannot classify our data set, this because of linearity nature of the LR classifier. The idea here is to take the data and project it in a high dimensional space using kernel, then we train a linear separator in the new dimension, we can notice that there is a similarity between the kernel Logistic Regression and the SVM approach. We applied the kernel version of LR on same data and we got the following result:

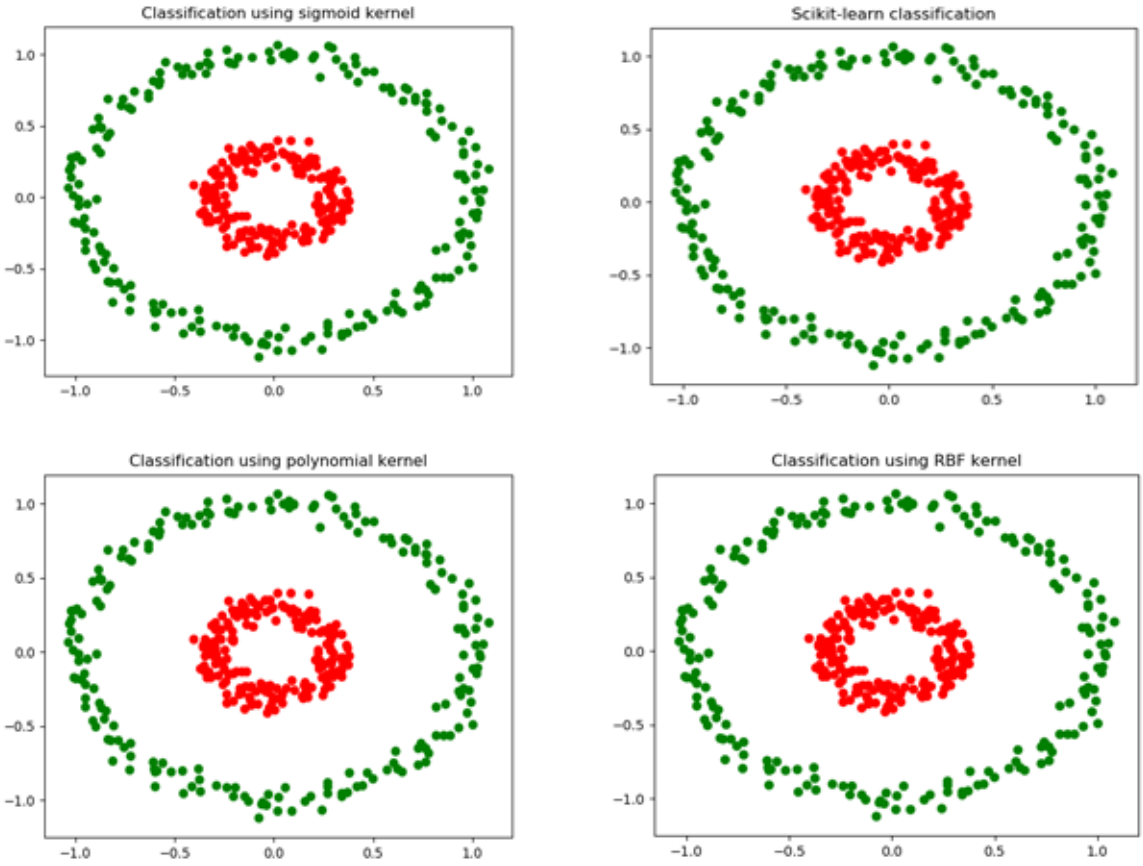


Figure 18: Kernel LR results

The dataset is now perfectly separated,even by using several values on the 3 kernels, this due to the simplicity of our dataset. For more experiments, we used the IRIS dataset. note that the iris dataset has 3 classes (versicolor, virginica, setosa). The pre-built classifier of Scikit-learn library and our implementations.

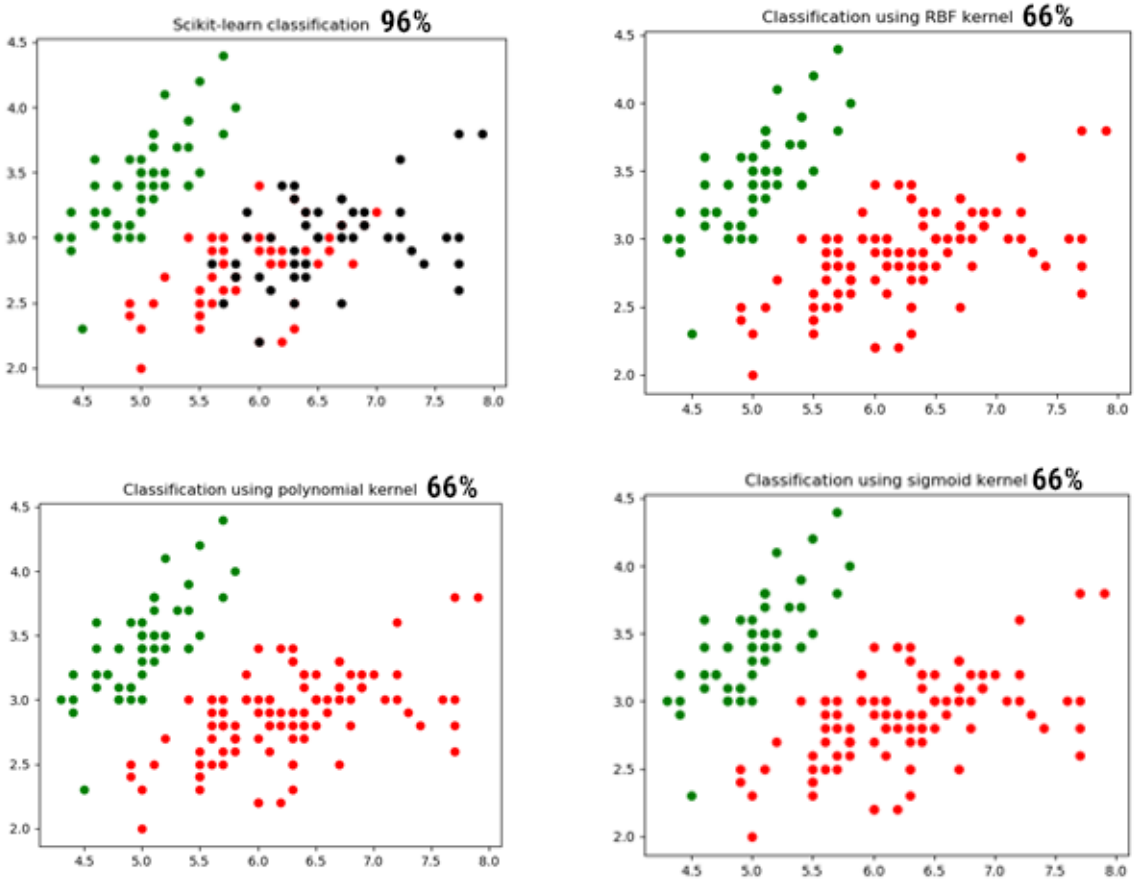


Figure 19: Kernel LR results

For the scikit learn implementation, it is suitable for multi-classification problems, but for our implementation we focused on binary classification, for this reason we notice that we got just two classes instead of 3. But, for this task the KLR still work well as it is able to distinguish between stosa and (versicolar, virginica), because the two last class are more similar.

2.3.5 Conclusion

- The Logistic Regression algorithm is efficient approach to classify labeled data.
- Logistic Regression is simple to implement

- It is can be used for multi-classification task
- The Kernel LR is similar to SVM, both of theme train to find a hyperplan to separate the data
- The Kernel LR can be used instead of SVM if the data is not huge, because of its complexity $\mathcal{O}(n^3)$

2.4 Support Vector Data Description

2.4.1 Overview of the method

The Support Vector Data Description (SVDD)[4] is inspired by the Support Vector Classifier. It obtains a spherically shaped boundary around a dataset and analogous to the Support Vector Classifier it can be made flexible by using other kernel functions. The method is made robust against outliers in the training set and is capable of tightening the description by using negative examples.

The main idea of the Support Vector Data Description is to find the smallest enclosing ball around the normally concentrated data which in the domain of Anomaly Detection can be seen as a normal data and the outliers are the fraud points in the data-set. The idea is to get the minimum enclosing ball around the data to distinguish between normal data-points and the fraud data-points. The SVDD method behaves exactly the same as the One-Class Support Vector Machine method with RBF-Kernel.

The underlying problem is to optimize the dual problem with following constraints in the equation.

$$\begin{aligned} \min_{x} \quad & \alpha^\top G\alpha - \alpha^\top diag(G) \\ \text{with} \quad & e^\top \alpha = 1 \\ \text{and} \quad & 0 \leq \alpha_i, i = 1..n \end{aligned}$$

where α is the variable to optimize and G represents the gram matrix obtained.

After the optimization process the α values are used to obtain the center of the enclosing circle and it is also used to obtain the radius of the circle.

2.4.2 Experiment and Results

For the implementation of the algorithm from the scratch the basic definition were written.A function was made to generate the data-set with desired number of samples and the outliers to be introduced to the data-set.
For the experiment the data-set in the figure 20 was made. It is a collection of 300 concentrated sample points and 15 outliers.

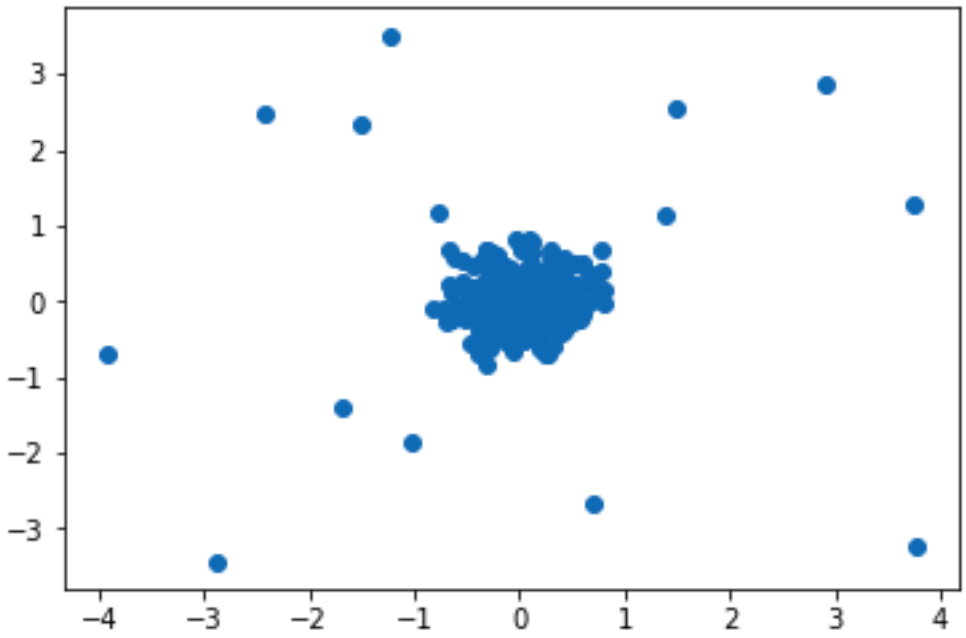
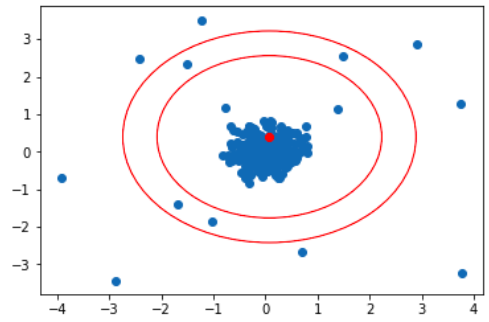


Figure 20: Data-set with 300 sample points and 15 outliers

The python library **cvxpy** python solver was used to solve the optmization problem.

Linear Kernel SVDD:
The Linear Kernel SVDD uses a simple linear kernel to obtain the gram matrix to be fed to the optimization problem solver. The dataset in 20 is fed to the Linear Kernel and the following results are obtained as shown the figure 21.

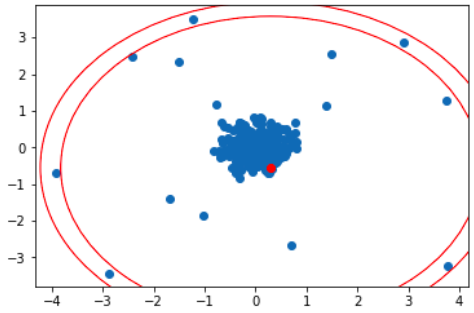
```
Problem Status: optimal
[array([[2.15988657]]), array([[2.81828687]])]
```



```
The number of support vectors are 2
```

(a) SVDD Constant 0.1

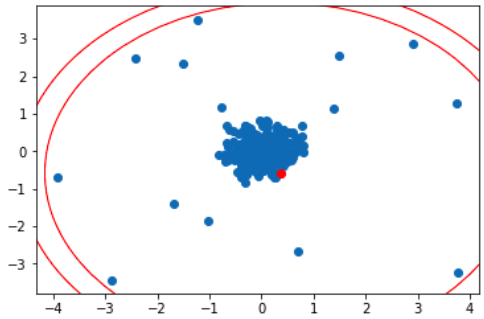
```
Problem Status: optimal
[array([[4.12168698]]), array([[4.52278771]])]
```



```
The number of support vectors are 2
```

(b) SVDD constant 0.3

```
Problem Status: optimal
[array([[4.53186661]]), array([[5.02026474]])]
```



```
The number of support vectors are 2
```

(c) SVDD constant 0.4

Figure 21: Enclosing Ball with different SVDD Constants

It can be seen in figure 21a, 21b and 21c the effect of the different values of the SVDD Constant. As the value of the constant increases, the size of the enclosing ball increases including outlieres also. The slack variable allows the margin to be large and also allow to learn an enclosing ball of large radius. Thus, it is a trade off between the amount of outliers that can be allowed to be considered inside the enclosing ball and the speed of the classification problem.It is because if we allow a very small outliers to be included in the enclosing ball, the process becomes very slow.

The implementation of the **RBF Kernel SVDD** and **Polynomial Kernel SVDD** is detailed in the Jupyter Notebook with the results obtained in it. It performs similar to the Linear Kernel when chosen the right constants for both of them respectively.

2.4.3 Conclusion

The Support Vector Data Description is a powerful algorithm while dealing with the multidimensional outlier detection problem. Instead of estimating a probability density, it obtains a boundary around the data set. By avoiding the estimation of the data density, it can obtain a better data boundary. The data description is inspired by the Support Vector Classifier. The boundary is described by a few training objects, the support vectors. It's use can be widely seen for the Anomaly Detection, which is an important for many industries.

References

[1] I.S. Dhillon, Y. Guan, and B. Kulis. "Kernel k-means: spectral clustering and normalized cuts". In: (2004), p. 556.

[2] Sebastian Rachka. "Kernel tricks and nonlinear dimensionality reduction via RBF kernel PCA". In: URL: https://sebastianraschka.com/Articles/2014_kernel_pca.html.

[3] Germán Rodríguez. "Generalized Linear Models". In: URL: http://data.princeton.edu/wws509/notes/#.

[4] David M.J. Tax and Robert P.W. Duin. "Support Vector Data Description". In: *Machine Learning* 54.1 (Jan. 2004), pp. 45–66. ISSN: 1573-0565. DOI: 10.1023/B:MACH.0000008084.60811.49. URL: https://doi.org/10.1023/B:MACH.0000008084.60811.49.

[5] Ji Zhu and Trevor Hastie. "Kernel Logistic Regression and the Import Vector Machine". In: *Advances in Neural Information Processing Systems 14*. Ed. by T. G. Dietterich, S. Becker, and Z. Ghahramani. MIT Press, 2002, pp. 1081–1088. URL: http://papers.nips.cc/paper/2059-kernel-logistic-regression-and-the-import-vector-machine.pdf.