A Brief Review Of Support Vector Machines and a Proposal For A New Kernel Via Localization Heuristics

Malik Balogoun

A Thesis
in
The Department
of
Mathematics and Statistics

Presented in Partial Fulfillment of the Requirements
for the Degree of‘ Master of Arts (Mathematics) at
Concordia University
Montreal, Quebec, Canada

August, 2019

CONCORDIA UNIVERSITY
School of Graduate Studies

This is to certify that the thesis prepared

By: Malik Balogoun

Entitled: A Brief Review Of Support Vector Machines and a Proposal

For A New Kernel Via Localization Heuristics

and submitted in partial fulfillment of the requirements for the degree of

Master of Arts (Mathematics)

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final Examining Committee:

Cody Hyndman	Chair
Wei Sun	Examiner
Debaraj Sen	Examiner
Arusharka Sen	Supervisor

Approved by Cody Hyndman
Chair of Department

August 2019, Andre Roy
Dean of Faculty

ABSTRACT

A Brief Review Of Support Vector Machines and a Proposal For A New Kernel Via Localization Heuristics

Malik Balogoun, MA

Concordia University, 2019

This thesis deals with a particular problem of binary classification case in the framework of support vector machine. The case displays observations from two classes, and uniformly distributed on a space so that linear separation by a hyperplane is only possible in tiny cubes (or rectangles) of that space. The general approach to classification in the input space is then extended with the design of a new ad hoc kernel that is expected to perform better in the feature space than the most common kernels found in the literature. Theoretical discussions to support the validity, the convergence to Bayes classifier of the new designed kernel and its application to simulated dataset will be our core contribution to one a way we can approach a classification problem.

In order to make our way to this goal and grasp the necessary mathematical tools and concepts in support vector machine, a literature review is provided with some applications in the first four sections of this document. The last and fifth section brings an answer the question that motivates this research.

ACKNOWLEDGEMENTS

To my supervisor, Dr. Sen Arusharka, I couldn't have done this without you. Thank you for your help, your guidance and your support along the way.

Contents

List of Figures

1 Introduction

In machine learning, the Support Vector Machine (SVM) is a type of supervised learning that uses algorithms to either classify or apply regression analysis on a dataset. In this thesis, we will only focus on the SVM as a tool for classification. First we provide related concepts and second, we provide properties of Bayes classifier of the new proposed kernel. Various types of kernels are considered which is plausible in this contest. Formally, the method solves a binary discrimination problem which takes a training sample, and constructs the function f which estimates the decision boundary by allowing an optimal margin. The SVM's algorithm finds the best estimation of the function f learned on a sample training dataset such that the misclassification error rate on another test dataset is minimum. The SVM learning algorithm applies on quantitative observations as well as qualitative ones.

The SVMs theory was originally established in the works of Vapnik, and Chervonenkis in 1963 before it became more popular in the 1990's with many applications in real world problems. The boundary induced by the algorithm can be either linear or non-linear depending on the *generally unknown* distribution of the learned sample data. In each case, some mathematical concepts will be explained to justify the chosen algorithm. The convergence of the studied SVM's algorithms towards the Bayes Classifier will be assessed as well as the convergence of an ad hoc learning algorithm method towards the Bayes classifier.

2 Support Vector Machine as a linear classifier

2.1 The separating hyperplane

In a p - *dimensional space*, a *affine* hyperplane is $(p-1)$ - dimensional subspace which is described by the set of the form:

$$\{x = (x_1, ..., x_p) \in \mathbb{R}^p : \beta_0 + \sum_{j=1}^{p} \beta_j x_j = \beta_0 + \beta^T x = 0\}, \tag{1}$$

with $\beta = (\beta_1, ..., \beta_p)$ the normal vector to the hyperplane.
The hyperplane divides the space in two half-spaces defined by the sets of the form:

$$\{(x_1, ..., x_p) \in \mathbb{R}^p : \beta_0 + \sum_{j=1}^{p} \beta_j x_j < 0\} \tag{2}$$

and,

$$\{(x_1, ..., x_p) \in \mathbb{R}^p : \beta_0 + \sum_{j=1}^{p} \beta_j x_j > 0\} \tag{3}$$

hence a suitable geometric representation to perform a linear separation. Notice that in $\mathbb{R}^2$, the hyperplane gives the equation of a line, and that of a plane in $\mathbb{R}^3$. For example, let's consider a sample data points $(x_1, ..., x_n)$ of size n made of two classes. If we construct such a hyperplane so that all the observations of one class fall in the set of relation (2) and the remainder in the set of relation (3), the separation is achieved. By convention, the observations are labelled $y_i = -1$ in set (2) and $y_i = 1$ in set (3). Therefore, any new observation $x_i, i = n+1$, will be classify according to the hyperplane boundary as -1 or 1 and we can write:

$$y_i(\beta_0 + \sum_{j=1}^{p} \beta_j x_j) > 0 \tag{4}$$

However, the existence of such a hyperplane is not unique and it always possible for example to find $\varepsilon > 0$ such that $\{(x_1, ..., x_p) \in \mathbb{R}^p : \beta_0 + \varepsilon +$

$\sum_{j=1}^{p} \beta_j x_j = 0\}$ is another a separating hyperplane.

2.2 Maximal margin classifier

The concept of Maximal margin classifier is now introduced to help us to find the best hyperplane among the infinity number of choices. This specific separating hyperplane will be chosen in a way to maximize the smallest distance between any data point and itself. Let's digress here, and discuss the linear algebra theory that gives us the *shortest* distance of a given data point to a hyperplane in the space.

2.2.1 Distance of a point to a hyperplane

For the simplicity of the illustration, let's consider a hyperplane in $\mathbb{R}^3$, *i.e. a plane*. If P is a point on the hyperplane L and we want to find the distance of any point Q (*not on* L) to L, given that O is the origin of the vector space and β a normal vector to L at P.

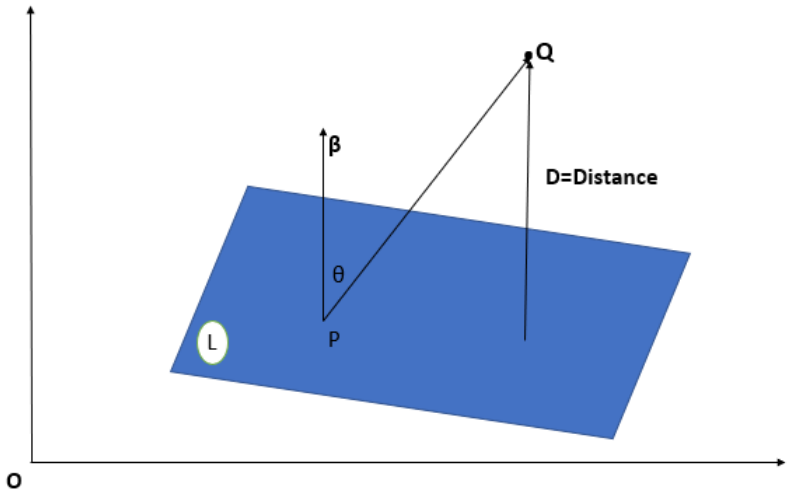

Figure 1: Distance of a given data point Q the a hyperplane

The signed (*"signed" because* Q *can be on either side of* L) distance D of Q to the hyperplane L is derived as follows:

if $\overrightarrow{OQ} = x = (x_1, ..., x_p)$, is a given point of the space not the hyperplane L, and $\overrightarrow{OP} = x_0 = (x_{10}, ..., x_{p0})$, the point on the hyperplane L closest to x, we have $(x - x_0)$ is a multiple of β, so that $x - x_0 = c\beta$, hence $c = \beta^T(x - x_0)/||\beta||^2$, therefore $||x - x_0|| = |c|||\beta||$,

Hence the signed distance of any point x to the hyperplane L is:

$$D = \frac{(\beta^T x - \beta_0)}{\|\beta\|} \tag{5}$$

2.2.2 Finding the optimal separating plane

We can now compute the distance of any point to the hyperplane. However, for the choice of the hyperplane to be optimal, we must find a margin $M > 0$ being the maximal possible such that the closest points on each side of the hyperplane are at the signed distance M from it. In others words, we must solve the following optimization problem:

$$\max_{\beta, \beta_0, \|\beta\|} M$$

$$\text{subject to } \frac{y_i(x_i^T\beta + \beta_0)}{\|\beta\|} \geq M \Rightarrow y_i(x_i^T\beta + \beta_0) > M\|\beta\|, \quad i = 1, ..., N$$

with $x_i = (x_{1i}, ..., x_{pi})$ and $y_i = \{-1, 1\}$
The data points x_i in the space $\mathbb{R}^p$ such that:

$$\frac{y_i(x_i^T\beta + \beta_0)}{\|\beta\|} = M,$$

i.e. the closest points on each side of the hyperplane are called **Support Vectors**, and the lines which support them are the **boundaries**.
To simplify the above problem, we can arbitrarily set $\|\beta\| = \frac{1}{M}$, since any

scaled vector of the β is also a normal vector to the hyperplane, and using the norm of that scaled vector *i.e. any positive real number(here, we choose* $\frac{1}{M}$*)* doesn't change the optimization problem.
The constraint becomes $y_i(x_i^T\beta + \beta_0) \geq 1$ and, if we label x_{i-} and x_{i+} the supports vectors respectively below and above the classifier, we can write that $x_{i-}^T\beta + \beta_0 = -1$ and $x_{i+}^T\beta + \beta_0 = 1$; moreover there exist $r > 0$ such that $x_{i+} = x_{i-} + r\beta$.
Note that maximizing $M = \|x_{i+} - x_{i-}\| = \|r\beta\|$ is equivalent to maximizing $M^2 = r$ or minimizing $\frac{1}{r}$.

So:

$$\begin{aligned} 1 &= x_{i+}^T\beta + \beta_0 \\ &= [(x_{i-} + r\beta)\beta] + \beta_0 \\ &= r\|\beta\|^2 + x_{i-}^T\beta + \beta_0 \\ &= r\|\beta\|^2 - 1 \end{aligned}$$

therefore,

$$r = \frac{2}{\|\beta\|^2} \Rightarrow \frac{1}{r} = \frac{1}{2}\|\beta\|^2$$

Finally, to find the maximal margin classifier, we have to solve the following convex optimization problem:

$$\min_{\beta,\beta_0} \frac{1}{2}\|\beta\|^2, \text{ subject to } y_i(x_i^T\beta + \beta_0) \geq 1 \tag{6}$$

The Primal Lagrange function L_P is:

$$L_P = \frac{1}{2}\|\beta\|^2 - \sum_{i=1}^{N} \alpha_i[y_i(x_i^T\beta + \beta_0) - \|s_i\| - 1] \tag{7}$$

with α_i being the Lagrange multiplier for a given data point x_i.

Then the successive derivatives with respect to β and β_0 give :

$\frac{\partial L_P}{\partial \beta} = 0 \Rightarrow \sum_{i=1}^{N} \alpha_i y_i x_i = \beta$
$\frac{\partial L_P}{\partial \|s_i\|} = 0$
$\frac{\partial L_P}{\partial \beta_0} = 0 \Rightarrow \sum_{i=1}^{N} \alpha_i y_i = 0$

By replacing them in (7) and knowing that $\beta^T = \sum_{i=1}^{N} \alpha_i y_i x_i^T$, we obtain a new expression for the Lagrangian, L_D for the Dual that we will maximize by having with an appropriate software solve it for the values of $\alpha_i' s$.

$$\begin{aligned} L_D &= \frac{1}{2}\beta^T\beta - \left(\sum_{i=1}^{N} \alpha_i y_i x_i^T\right)\beta + \sum_{i=1}^{N} \alpha_i \\ &= \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{k=1}^{N} \alpha_i \alpha_k y_i y_k x_i^T x_k \\ &\text{subject to } \alpha_i \geq 0 \text{ and } \sum_{i=1}^{N} \alpha_i y_i = 0 \end{aligned}$$

The Karush-Kuhn-Tucker suggests that:

$$\alpha_i[y_i(x_i^T\beta + \beta_0) - 1] = 0 \qquad \text{for all } i \tag{8}$$

Therefore, with $\alpha_i \geqslant 0$:
i) if $y_i(x_i^T\beta + \beta_0) - 1 > 0$ for the data points correctly classified and not on the boundary, then we must have $\alpha_i = 0$
ii) if $\alpha_i > 0$, we must have $y_i(x_i^T\beta + \beta_0) - 1 = 0$, thus only the points on the boundaries, the support vectors, correspond to $\alpha_i > 0$. A given point x_i

along with its value y_i, we solve for β_0 from (8) as follows:

$$\beta_0 = \frac{1}{y_i} - x_i^T \beta \text{ with } \sum_{i=1}^{N} \alpha_i y_i x_i = \beta$$

Finally, an estimation (*based on the training data points*) of the maximal margin classifier is completely define, and the class $y^* \in \{-1, 1\}$ of any new test data point x^* will determined as:

$$Class(x^*) = sign[(x^*)^T \beta + \beta_0]$$

2.3 Support Vector classifier (SVC)

The maximal margin classifier can encounters two main problems:
- the non-robustness, *i.e.* in case the margin is not wide enough, the addition of new data points could lead to a new maximal margin classifier as new support vectors will be defined.

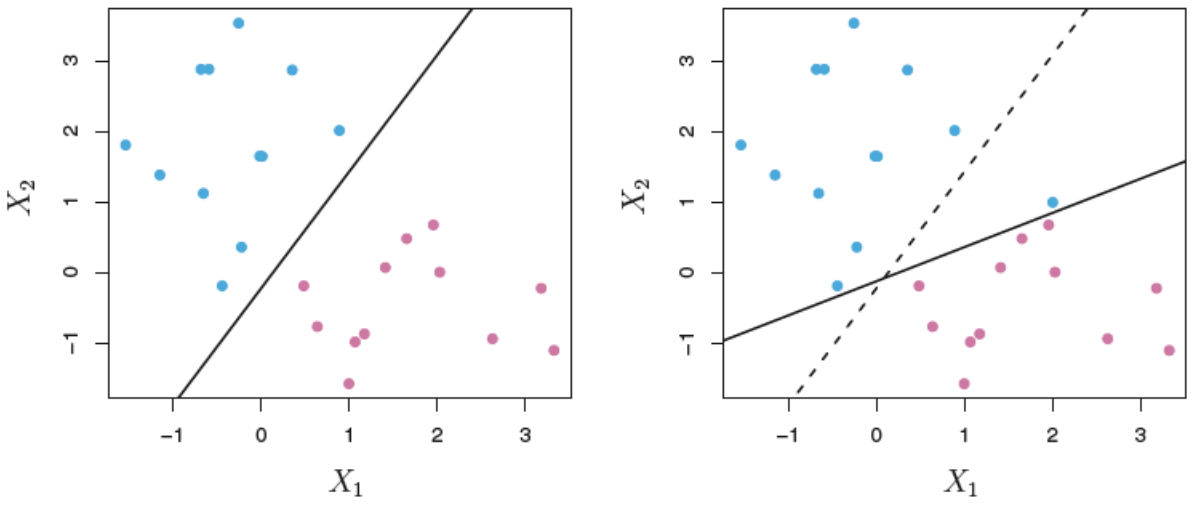

Figure 2: Non robustness of the maximal margin classifier to entry of a new data point *([2], chapter 9, pp:345)*

- the non separable data, *i.e.* no linear hyperplane can be found to separate the data in two distinct categories. However a maximal margin classifier can still be obtain by allowing a few observations to be misclassified.

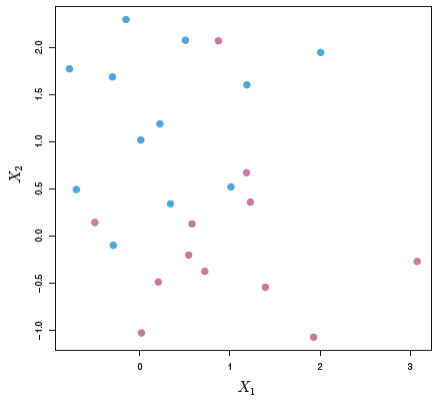

Figure 3: A case where the classes of the data points are not linearly separable *[2, chapter 9, pp:344]*

These two shortcomings are overcome with the support vector classifier as a generalization of the maximal margin classifier.

2.3.1 Construction of a SVC

In the both above cases *i.e. non-robustness and or overlapping classes* , let's define for a given data point x_i a slack variable ξ_i which denotes:

i) the distance by which a training data point x_i violates the margin distance M without being misclassified. The maximal margin classifier constructed with a few margin violations of such $x_i's$ fixes the non-robustness as it allows a wider margin which guarantees a better classification for a test data point.

ii) in the situation of overlapping classes, the slack variables $\xi_i's$ are used directly on the training data as no hyperplane separator is possible.
So we can rewrite the constraint of the classical optimization problem as follow:

$$y_i(x_i^T\beta + \beta_0) \geq M - \xi_i, \qquad i = 1, ..., N \tag{9}$$

However, although the previous relation is established naturally by construction, it does lead to a convex optimization problem.
Therefore, it is more convenient to replace it with its equivalent:

$$y_i(x_i^T\beta + \beta_0) \geq M(1 - \xi_i), \qquad i = 1, ..., N \tag{10}$$

where ξ_i is now expressed as the proportion of M by which x_i violates the margin.
So $\xi_i = 0$ occurs when x_i is correctly classified, and $\xi_i = 1$ occurs when x_i violated the margin by a distance M and is location on the separating hyperplane. $\xi_1 > 1$ is a clear misclassification Thus, if we want a maximum of K training data points to be misclassified while finding the support vector classifier, we should set a budget $\sum_{i=1}^{N} \xi_i \leqslant K$. Notice here that if $\sum_{i=1}^{N} \xi_i = K$, we might have no point misclassified but rather several points only violating the margin.

2.3.2 Mathematical definition of a SVC

Let's C be the the cost that we pay for every single data point x_i that violates the margin or that is misclassified, *i.e.* $\xi_i > 0$. Then, the optimization problem whose solution gives the SVC, is expressed as follow:

$$\begin{aligned} &\min_{\beta,\beta_0} \frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^{N} \xi_i \\ &\text{subject to } \xi_i \geq 0, \;\; y_i(x_i^T\beta + \beta_0) \geq 1 - \xi_i, \qquad i = 1, ..., N \end{aligned} \tag{11}$$

If we want the cost $C = \infty$, for the optimization problem will remain valid only if we have $\xi_i = 0$ *i.e. all the x_i are correctly classified and none of them has violated the margin*, hence the Support Vector Classifier becomes exactly the maximal margin classifier.

From (10), the primal Lagrangian L_P is derived and it follows that:

$$L_P = \frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\alpha_i[y_i(x_i^T\beta + \beta_0) - (1-\xi_i)] - \sum_{i=1}^{N}\mu_i\xi_i \qquad (12)$$

The $\alpha_i' s$ and $\mu_i' s$ are the Lagrange multipliers.

By differentiating successively with respect to β, β_0 and ξ_i we obtain:

$$\begin{aligned}
&\frac{\partial L_P}{\partial \beta} = 0 \Rightarrow \beta = \sum_{i=1}^{N}\alpha_i y_i x_i \\
&\frac{\partial L_P}{\partial \beta_0} = 0 \Rightarrow 0 = \sum_{i=1}^{N}\alpha_i y_i \\
&\frac{\partial L_P}{\partial \xi_i} = 0 \Rightarrow \sum_{i=1}^{N}\alpha_i = CN - \sum_{i=1}^{N}\mu_i \Rightarrow \alpha_i = C - \mu_i \quad \forall i
\end{aligned} \qquad (13)$$

Re-expressing (11) with the results in (12) give the Dual function L_D as follows:

$$\begin{aligned}
L_D = inf(L_P) &= \frac{1}{2}\beta^T\beta - \left(\sum_{i=1}^{N}\alpha_i y_i x_i^T\right)\beta + \underbrace{\sum_{i=1}^{N} C\xi_i + (C-\mu_i)(1-\xi_i) - \mu_i\xi_i}_{=\sum_{i=1}^{N} C-\mu_i = \sum_{i=1}^{N}\alpha_i} \\
&= \sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{k=1}^{N}\alpha_i\alpha_k y_i y_k x_i^T x_k \\
&\left(\text{ by noticing that } \beta^T = \sum_{i=1}^{N}\alpha_i y_i x_i^T \right)
\end{aligned} \qquad (14)$$

L_D is a concave function, as it is the point-wise infimum of L_P which can be viewed as a family of affine functions. So,with an appropriate quadratic programming software, we maximize the Dual obtained in (13) subject to the constraints only in terms of $\alpha_i' s$ *i.e.* : $0 \leqslant \alpha_i \leqslant C$ (*since* $\alpha_i = C - \mu_i$ *and*

$\mu_i \geqslant 0$), and $\sum_{i=1}^{N} \alpha_i y_i$.

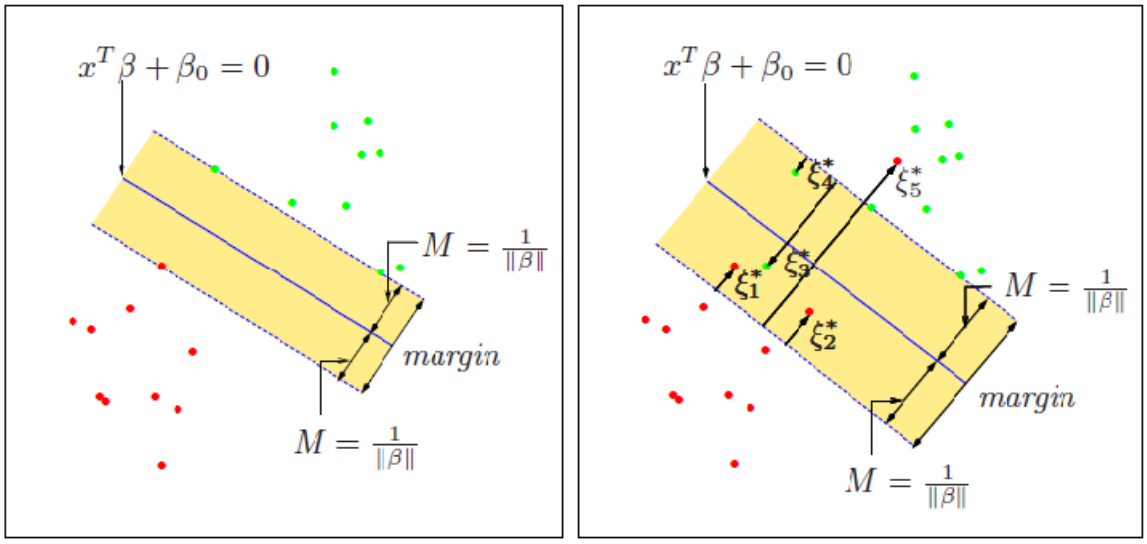

Figure 4: On the left is shown a maximal margin classifier in the linearly separable case, on the right, the maximal margin classifier has a soft margin to guarantee robustness *[1, chapter 12, pp:418]*

i) For $\alpha_i = C$, we identify the data points x_i that violate the margin by a distance $\xi_i > 0$
ii) For $\alpha_i = 0$, we can identifies the data points correctly classified and not on the boundaries. They correspond to $\xi_i = 0$ as well
iii) For $0 < \alpha_i < C$, we identify the data points on the boundary which are at a distance M from the support vector classifier.

In fact, these conclusions can be drawn simply by considering a Karush-Kuhn-Tucker condition which states that:

$$\alpha_i[y_i(x_i^T\beta + \beta_0) - (1 - \xi_i)] = 0 \tag{15}$$

Finally, any support vectors (*data points on the boundaries or those which violates the margin*) x_i can be used to deduce β_0 from (15), and given $\beta = \sum_{i=1}^{N} \alpha_i y_i x_i$, an estimation of the **Soft-Margin** separating hyperplane is ex-

pressed as:

$$\widehat{f}(x) = x^T\beta + \beta_0$$

Consequently, a new test data point x^* whose class $y^* \in \{-1, 1\}$ is classify as:

$$Class(x^*) = sign[(x^*)^T\beta + \beta_0]$$

3 Support Vector Machine as a non-linear classifier

So far, the algorithm for the optimal linear classifier is only suitable for simplistic data classification. In practice, many data points are just impossible to separate with a hyperplane, even if we allow a soft-margin with a budget for misclassification as we discuss in the previous section.

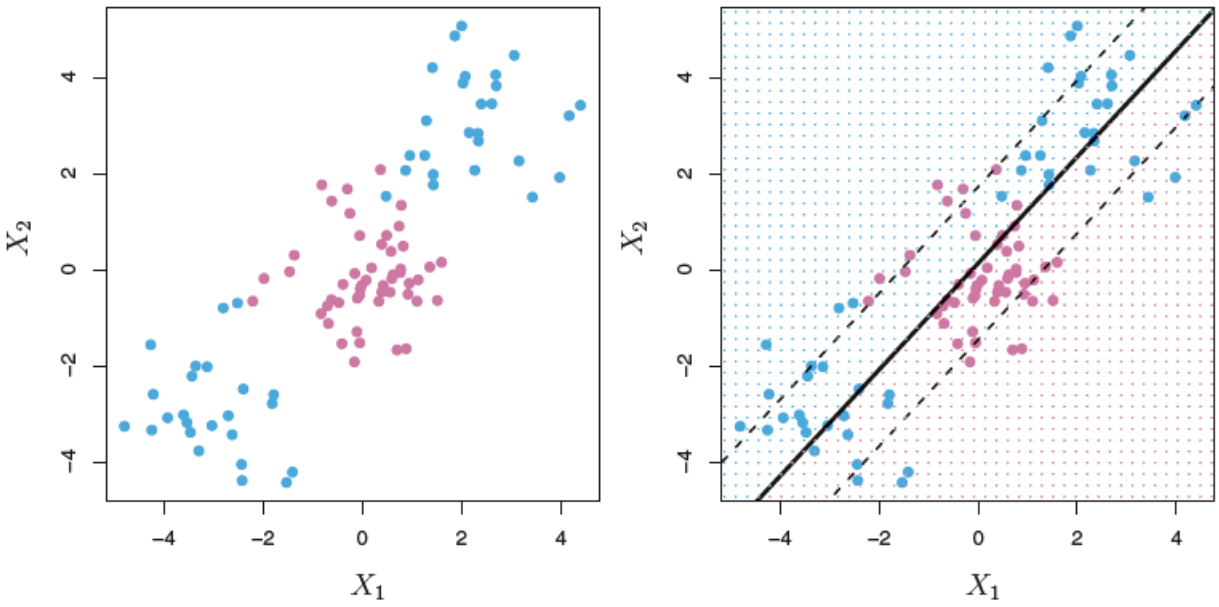

Figure 5: Non adaptiveness of a linear classifier in some cases *[2, chapter 9, pp:349]*

Therefore, a non-linear classifier is required, and the goal is to find a rule, *generally expressed by a (number of) function(s)* learned from the training data, that we will be as efficient as possible to determine the class of a new test data point.

3.1 Vapnik Chervonenkis (VC) dimension

The function space *or the set of functions* that will express the rule for classification is characterized by its capacity *i.e. complexity, expressive power, or flexibility* depending on how the class of the training data points are mixed and overlapped.
The cardinality of the largest training data points that can be classified by the set of functions is defined as the **VC dimension**.
Given a classification model *i.e. a choice of a set of function space* with a VC dimension D, a well known result in statistical learning theory stipulate that:

$$Pr\left\{test\ \ error \leqslant training\ \ error + \sqrt{\frac{1}{N}[D(ln(\frac{2N}{D}) + 1) - ln(\frac{\eta}{4})]}\right\} = 1 - \eta \tag{16}$$

$\forall\ 0 \leqslant \eta \leqslant 1$ and with the condition that $D \ll N$ such that over-fitting is avoided. Otherwise the classifier function space will be too flexible and the test error will consequently become very high.
Overall, the goal is to find a rule that helps to classify. In the case of the Support Vector Classifier, we came up with a function:

$$\widehat{f}(x) = x^T\beta + \beta_0 = \sum_{i=1}^{N} \alpha_i y_i \underbrace{x^T x_i}_{\langle x, x_i \rangle} + \beta_0 \tag{17}$$

We must then find a similar approach with the function space that defines such a classifying rule.

3.2 Classical approach with Kernels

In the Kernel approach, each data point x_i in the in the *p-dimensional space* is transformed into $h(x_i)$ in a enlarged *M-dimensional space* such that:

$$h(x_i) = (h_1(x_i), h_2(x_i), ..., h_M(x_i))$$

with $M > p$ and the basis functions $h_1, h_2, ...$and h_M suitably chosen so that we ensure linear class separation in the new feature space induced by the transformations. By similarity with (17), we could therefore define a Support Vector Classifier $\widehat{f}$ based on the training data as follows:

$$\widehat{f}(x) = h(x)^T\beta + \beta_0 = \sum_{i=1}^{N} \alpha_i y_i \underbrace{h(x)^T h(x_i)}_{\langle h(x), h(x_i)\rangle} + \beta_0 \tag{18}$$

The function $K(x, x_i) = \langle h(x), h(x_i)\rangle \ \forall i$ is called the **Kernel**, and has the particularity to be defined every time that the training data are linearly separable with soft-margin in the new feature space. Moreover, the kernel is required to be positive semi-definite.

The approach with kernels displays a computational advantage. In fact, the Lagrange Dual whose solution gives the classifier function $\widehat{f}$ can be defined by computing with a **fixed number of the kernel** $K(x_i, x_j) = \langle h(x_i), h(x_j)\rangle$ **computations** computations *i.e.* $\binom{N}{2}$ with $i, j = 1, ..., N$, **but not on the dimension size** M of the new feature space. This advantage is very appreciated when $M \to \infty$ (very high). Depending on the mixture of the classes on the training data points, a specific kernel is designed to achieve to lowest training and test errors.

Some commonly used Kernels are:

i) the d^{th} degree polynomial Kernel: $K(x_i, x_j) = (1+\langle x_i, x_j\rangle)^d$ which basically uses polynomial functions to find a support vector classifier in the enlarged feature space. Taking $d = 2$, and for two data points $x_i = (x_{i1}, x_{i2})$, and $x_j = (x_{j1}, x_{j2})$ we have:

$$
\begin{aligned}
K(x_i, x_j) &= (1 + \langle x_i, x_j \rangle)^2 \\
&= (1 + x_{i1}x_{j1} + x_{i2}x_{j2})^2 \\
&= 1 + (x_{i1}x_{j1})^2 + 2x_{i1}x_{j1} + (x_{i2}x_{j2})^2 + 2x_{i2}x_{j2} + 2x_{i1}x_{j1}x_{i2}x_{j2} \\
&= \langle h(x_i), h(x_j) \rangle
\end{aligned}
$$

We can then clearly recognize that $M = 6$ and the set of functions $h_1, ..., h_6$ such that:

$$
\begin{aligned}
h(x_k) = \Big(& h_1(x_k) = 1, h_2(x_k) = (x_{k1})^2, h_3(x_k) = \sqrt{2}x_{k1}, \\
& h_4(x_k) = (x_{k2})^2, h_5(x_k) = \sqrt{2}x_{k2}, h_6(x_k) = \sqrt{2}x_{k1}x_{k2} \Big), \qquad k = i, j
\end{aligned}
$$

ii) The radial basis function **(RBF)** Kernel:
$K(x_i, x_j) = exp(-\gamma \|x_i - x_j\|^2)$ with $\gamma = \dfrac{1}{2\sigma^2} > 0$, and σ a free parameter.
Replacing this kernel expression in relation (18) above, we see clearly that only the data points that are very close in distance to the new test observation x will essentially determine its class, as the exponential function *with the minus sign* systematically guarantees very low weight to sensibly far data points in the summation calculation in (18).
The classifier function using the radial basis has therefore a very local behaviour although it doesn't have an explicit graphical function. This has the advantage to deal with complicated cases where no function space could be found due to the clustered, mixed, or overlapped characteristics of the data points classes. Since very enlarged new feature space always achieves separability when the transformation set of functions are correctly defined, the RBF kernel can then be viewed as a method that leads to infinite dimensional for new feature space.

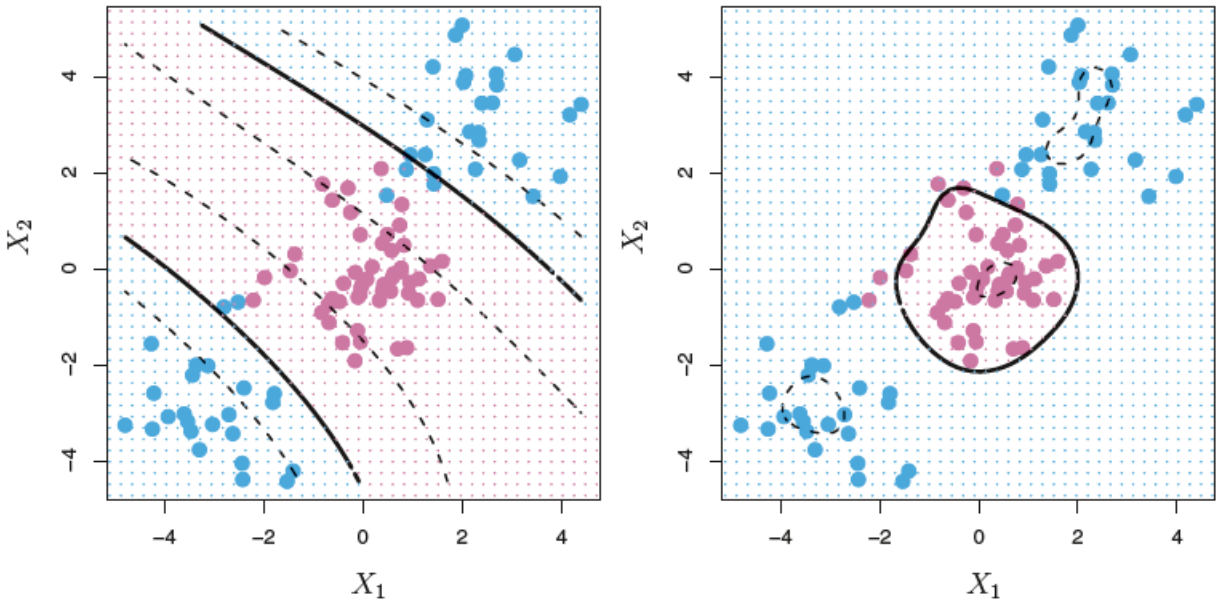

Figure 6: On the left,the use of the polynomial kernel performs poorly in separating the class. However, the graph on the right shows how the use of the RBF kernel is well adapted for the purpose *([2], chapter 9, pp:353)*

3.3 Classification with probabilistic kernel Bayes rule

Let $D_N = \{(\underline{\mathbf{x}}_i, y_i), i = 1, ..., N\}$ be the training dataset of N observations with $\underline{x}_i$ being a vector (*or observation*)in a n-dimensional vector space, and $y_i \in \{-1, 1\}$. We define $\widehat{y}_i$ as the predicted class for the observation i by any classification rule φ. The empirical risk on a test data set *as an estimate of the error rate* defined as $R(\varphi) = \frac{1}{K}\sum_1^K \mathbb{I}_{(\widehat{y}_i \neq y_i)}$ quantify the proportion of mistake made the classification rule on the test data containing K new observations.

we'll now prove that the Bayes classifier defined as:

$$\widehat{y}_i = argmax_{y_i \in \{-1,1\}} \mathbb{P}[Y = y_i | X = \underline{\mathbf{x}}_i] \tag{19}$$

is the a minimizer for both the test error rate and the expected error rate.

3.3.1 Optimality of Bayes classifier and lower boundary feature of Bayes error rate

For a given observation $\underline{x_i}$ in a test dataset, the test error rate is:

$$\mathbb{P}[\widehat{y_i} \neq y_i | \underline{X} = \underline{\mathbf{x}_i}] = 1 - \mathbb{P}[\widehat{y_i} = y_i | \underline{X} = \underline{\mathbf{x}_i}]$$

We can clearly see that only the Bayes classifier defined in (19) minimizes the latter quantity to its smallest value. In this case, it's assume that the distribution of X is known but not specified.We can then conclude that for any classifier suitable to classify the test data set generated form the distribution of X, the optimality of Bayes classifier is verified. Consequently, in practice:

$$\begin{aligned} \text{if } \mathbb{P}[\widehat{y_i} = 1 | \underline{X} = \underline{\mathbf{x}_i}] \geq \mathbb{P}[\widehat{y_i} = -1 | \underline{X} = \underline{\mathbf{x}_i}] \text{ or equivalently,} \\ \text{if } \mathbb{P}[\widehat{y_i} = 1 | \underline{X} = \underline{\mathbf{x}_i}] \geq 0.5 \\ \text{Then } \widehat{y_i} = 1 \end{aligned}$$

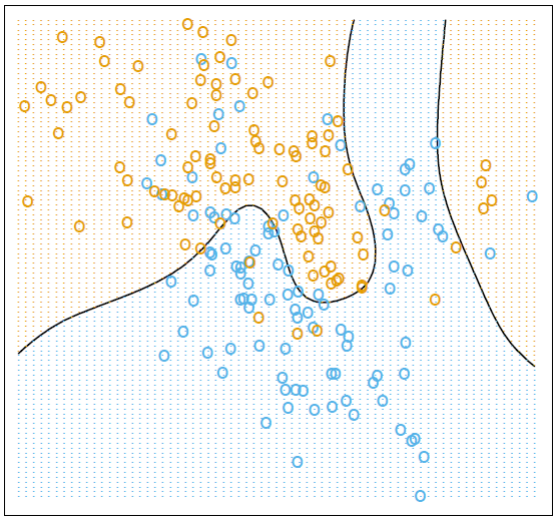

Figure 7: Exemple of Bayes optimal classifier along: the black line in the picture. The "+1" data points are represented in brown whereas the "-1" ones are represented in blue*[1, chapter 2, pp:21]*

On the other hand, the Bayes error rate does not take into consideration one observation to be tested, but rather all the observations generated by the

distribution of $\underline{X}$. Its expression is then given by:

$$\begin{aligned}\mathbb{P}[\widehat{y}_i \neq y_i] &= \mathbb{E}\left[\mathbb{P}[\widehat{y}_i \neq y_i|\underline{X}]\right] \\ &= \mathbb{E}\left[1 - \max_{y\in\{-1,1\}} \mathbb{P}[\widehat{y}_i \neq y|\underline{X}]\right] \\ &= 1 - \mathbb{E}\left[\max_{y\in\{-1,1\}} \mathbb{P}[\widehat{y}_i \neq y|\underline{X}]\right]\end{aligned}$$

Since the optimal Bayes classifier makes the Bayes error irreducible for any distribution of $\underline{X}$, the Bayes error is then viewed as the lowest boundary of the expected error rate of any classifier.

3.3.2 Dealing with unavailability of Bayes classifier, and convergence of other approaches

As discussed above, the Bayes classifier displays two main characteristics:
i) It requires the knowledge of the conditional distribution given $\underline{X}$ (true parameters of the distribution must be known)
ii)It functions as an unattainable gold standard (*lowest boundary of the expected error rate*) to which other classification methods are compared to judge their efficiency.
To overcome these two problematic features, the two following approaches are developed:
a) **The Quadratic Discriminant Analysis**: assumption of a normal distribution with overlapping classes (the distributions have at least different means).
Recall that we are interested in:

$$\begin{aligned}\mathbb{P}[y_i = y|\underline{X} = \underline{\mathbf{x}}_i] &= \frac{f(\underline{x}|y)\mathbb{P}[Y = y]}{\mathbb{P}[\underline{X} = \underline{\mathbf{x}}_i]} \text{ by Bayes rule and with } y \in \{-1, 1\} \\ &\propto f(\underline{x}|y)\mathbb{P}[Y = y] \text{ , since } \mathbb{P}[\underline{X} = \underline{\mathbf{x}}_i] \text{ doesn't depend on } y.\end{aligned}$$

It's assumed that $f(\underline{x}|y) = \dfrac{exp\Big[-\dfrac{1}{2}(\underline{x}_i - \mu_y)^T\Sigma_y^{-1}(\underline{x}_i - \mu_y)\Big]}{\sqrt{(2\Pi)^n|\Sigma_y|}}$, which is the multivariate normal distribution of $\underline{\mathbf{x}}$ given $y \in \{-1, 1\}$ in the n-dimensional space. μ_y and Σ_y are respectively the mean and the covariance matrix within the class y.
Let $\Pi_y = \mathbb{P}[Y = y]$.Then:

$$\mathbb{P}[y_i = y|\underline{X} = \underline{\mathbf{x}}_i] \propto f(\underline{x}|y)\Pi_y$$

Thus,

$$\begin{aligned}
\widehat{y_i} &= \underset{y\in\{-1,1\}}{arg\max}\ f(\underline{x}|y)\Pi_y \\
&= \underset{y\in\{-1,1\}}{arg\max}\ ln[f(\underline{x}|y)\Pi_y]\ \text{, since } ln \text{ is an increasing function} \\
&= \underset{y\in\{-1,1\}}{arg\max}\ -ln\left(\sqrt{(2\Pi)^n|\Sigma_y|}\right) - \frac{1}{2}(\underline{x}_i - \mu_y)^T\Sigma_y^{-1}(\underline{x}_i - \mu_y) + ln(\Pi_y)
\end{aligned}$$

$$\underset{y\in\{-1,1\}}{arg\max}\ ln(\Pi_y) - \frac{1}{2}(\underline{x}_i - \mu_y)^T\Sigma_y^{-1}(\underline{x}_i - \mu_y) - \frac{1}{2}ln(|\Sigma_y|) \tag{20}$$

$\left(\text{since } -\dfrac{n}{2}ln(2\Pi)\perp y\right)$
By setting:

$$a_y = -\frac{1}{2}\Sigma_y^{-1}, \quad b_y = \mu_y^T\Sigma_y^{-1} \quad \text{, and} \quad c_y = ln(\Pi_y) - \frac{1}{2}ln(|\Sigma_y|) - \frac{1}{2}\mu_y^T\Sigma_y^{-1}\mu_y$$

Relation (20) it rewritten as follows:

$$\widehat{y_i} = \underset{y\in\{-1,1\}}{arg\max}\ \underline{x}_i^T a_y \underline{x}_i + b_y\underline{x}_i + c_y \tag{21}$$

Defining $F_y(\underline{x}_i) := \underline{x}_i^T a_y \underline{x}_i + b_y\underline{x}_i + c_y$ leads to the conclusion that the decision boundary (*the Quadradic Discriminant classifier*) is made of the

observations $\underline{x}_i$ such that $F_{(-1)}(\underline{x}_i) = F_{(+1)}(\underline{x}_i)$

Remark:

- Let $\mathcal{S}_y = \left\{ \underline{x}_i : y_i = y \right\}$ with $y \in \{-1, 1\}$.

In this case where the multivariate normal distribution $\mathcal{N}(\mu_y, \Sigma_y)$ is assumed, if its parameters and Π_y are not known, they can be estimated by:

$\widehat{\Pi}_y = \dfrac{card(\mathcal{S}_y)}{N}$, N being the total number observations for the two classes +1 and -1

$\widehat{\mu}_y = \dfrac{1}{card(\mathcal{S}_y)} \sum_{i \in \mathcal{S}_y} \underline{x}_i,$

and the estimated covariance matrix,

$$\widehat{\Sigma}_y = \frac{1}{card(\mathcal{S}_y - 1)} \sum_{i \in \mathcal{S}_y} \left(\underline{x}_i - \widehat{\mu}_y \right) \left(\underline{x}_i - \widehat{\mu}_y \right)^T$$

b) **The K-Nearest Neighbour (KNN)**: no assumption about the distribution of the $\underline{x}_i' s$, and no overlapping classes.

This probabilistic approach determines the class of a new observation $\underline{x}_0$ by choosing the class that predominates among its K-nearest observations in the training set.

Mathematically, it mimics the Bayes classifier rule as follows:

$$if \;\; \widehat{\mathbb{P}}[y_0 = y | \underline{X} = \underline{x}_0] = \frac{1}{K} \sum_{i \in \mathcal{N}_0} \mathbb{I}_{y_i = y} \geq 0.5 \;\; then \;\; \widehat{y}_0 = y$$

$$with \;\; y \in \{-1, 1\} \;\; and \;\; \mathcal{N}_0 = \{ \underline{x}_i \;\; : \;\; 1 \leq i \leq K \}$$

Remark:

The class of a test observation $\underline{x}_0$ is determined by choosing the class of its closest observations *(which are given higher weights by construction of the exponential function).*

- Notice also that, in the implementation of the RBF kernel, the smaller the

tuning parameter γ is, the greater is the number of observations $\underline{x}_i's$ whose classes substantially influence the determination of the class for $\underline{x}_0$.

4 Application

4.1 The k-fold cross-validation algorithm implementation

In implementing the various SVM algorithms for this application section, the k-fold cross validation will be applied to determine the best parameters of the model that insure the best Bias-variance trade-off for the classifier boundary. The k-fold cross-validation involves splitting the dataset into k subsets. For each $i = 1, ..., k$ the following steps are repeated:
i) The model is trained on all training data except those in from the i^{th} subset.
ii) Using the latter trained model, predictions are performed for each observation from the i^{th} subset.
iii) Prediction errors (out-of-sample errors) for each observation of the i^{th} subset are recorded.
Then, the set of all out-of-sample errors (from all k subsets) are jointly considered to calculate a goodness-of-fit statistics e.g. the out-of-sample MSE obtained by taking the sample average of all squared out-of-sample prediction errors. The model retained is the one yielding the best out-of-sample goodness-of-fit.

4.2 SVM on observations with overlapping classes

Linear Kernel

we will first implement the classical SVM with linearly separable case along with a soft margin.
Let's generate $N = 100 = N_1 + N_2$ observations from two bivariate normal distributions, N_1 $i.i.d$ observations from $\mathcal{N}(\mu_1, \Sigma_1)$, and N_2 $i.i.d$ observa-

tions from $\mathcal{N}(\mu_2, \Sigma_2)$ with N_1 and N_2 unknown, $\mu_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$, $\mu_2 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$, and $\Sigma_1 = \Sigma_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$.

Let's check if classes are linearly separable in 100 randomly chosen observations:

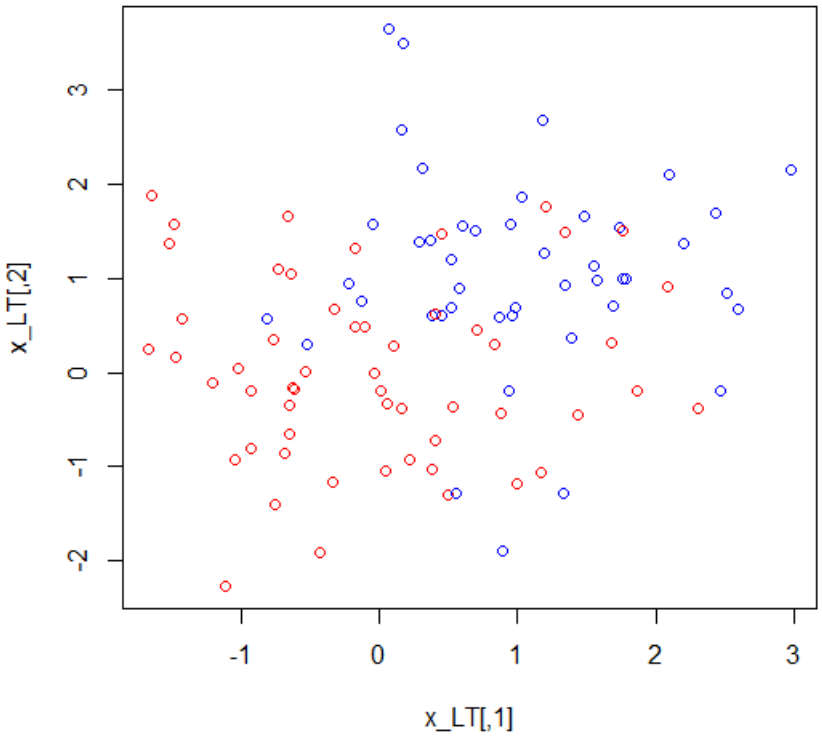

Figure 8: Plot of training dataset for overlapping classes

The plot shows that the two groups are NOT linearly separable (except with soft margin). First, we implement the linear kernel with soft margin. For $k = 10$, the cross validation algorithm gives that the *cost* $= 0.1$ resulted in the lowest cross validation error rate, thus the best model.

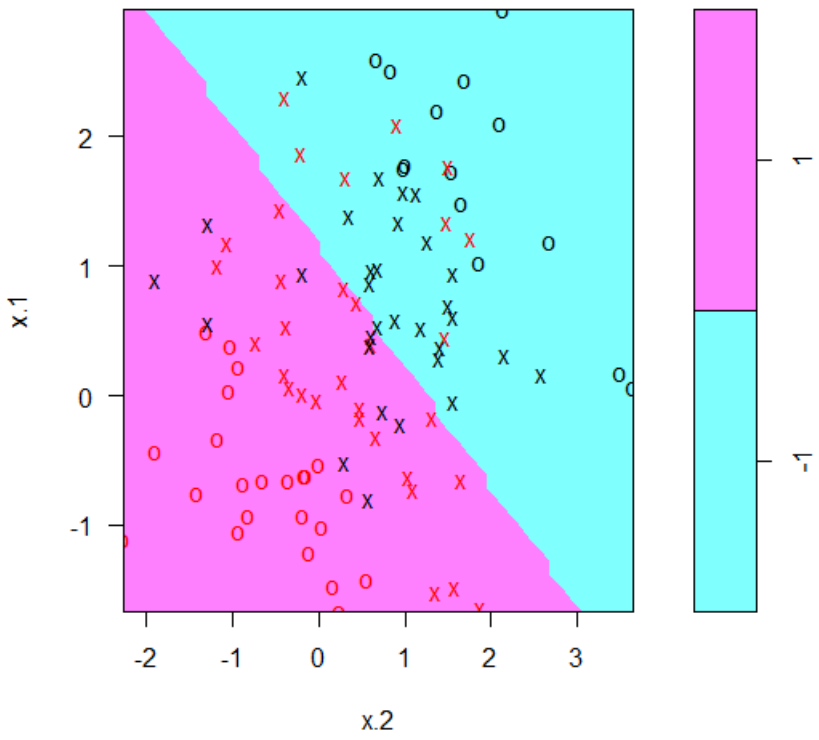

Figure 9: SVM classification plot for the linear kernel best model: The support vectors are marked with a cross

The best model displays 63 support vectors, 32 for the class "-1" and 31 for the class "1". Let's now use the best model function to predict on a test data generated exactly as the training dataset, i.e. with $N_1 + N_2 = N$, N_1 and N_2 are unknown and are generated from the same bivariate normal distributions as described previously.

	TRUE	
PREDICT	-1	1
-1	47	6
1	8	39

Figure 10: Performance of the best linear kernel model on the test dataset

Among 100 test observations, 86 are classified correctly, thus an error rate of 14%.

Polynomial Kernel

Let's now learn the same training dataset with the polynomial kernel and check its performance on the same test dataset. The 10-fold cross validation gives that $d = 1$ as resulted in the lowest cross validation error rate, thus the best model.

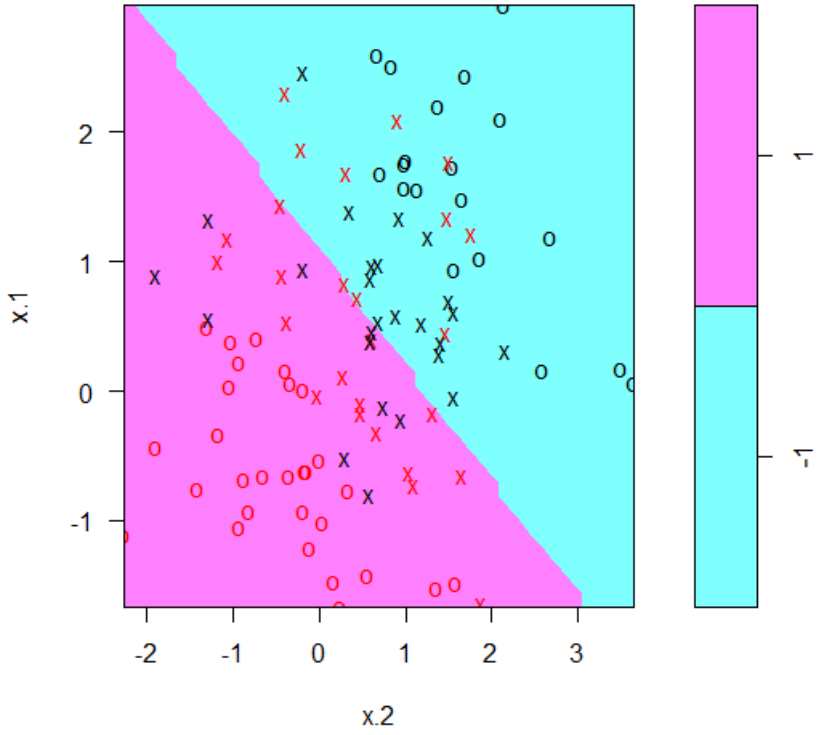

Figure 11: SVM classification plot for the polynomial kernel best model: The support vectors are marked with a cross

The best model displays 52 support vectors, 26 for the class "-1" and 26 for the class "1". Let's now use the best model function to predict on a test dataset.

	TRUE	
PREDICT	-1	1
-1	47	6
1	8	39

Figure 12: Performance of the best Polynomial kernel model on the test dataset

Among the 100 observations, 86 are classified correctly, thus an error rate of 14% , similar to the Linear kernel case. When a larger training and test datasets ($N = 200$) are considered, the error rate in the best model for the linear kernel is 21.5%, whereas it amounts up to 24% for the polynomial kernel.

See table below:

Linear kernel

	TRUE	
PREDICT	-1	1
-1	72	18
1	25	85

Polynomial kernel

	TRUE	
PREDICT	-1	1
-1	68	19
1	29	84

Figure 13: Performance of the best linear and polynomial kernel models on the test dataset with $N = 200$ (larger dataset)

We noticed that by increasing to $N = 200$, the error rate for both the linear and the polynomial kernels are higher although they have been given a larger training sample. So as N goes to infinity, neither of them converges to the Bayes classifier. Since cross validation has already chosen the optimal parameter for the best model, any attempt to increase these would lead to over-fitting, low error rate in the training sample, but NOT convergence to Bayes classifier as the resulting model would perform poorly on the test dataset.

Radial Basis Function kernel

Let's now learn the same training dataset with the Radial Basis Function kernel and check its performance on the same test dataset. The 10-fold cross validation gives that $gamma = 0.5$ as resulted in the lowest cross validation error rate, thus the best model.

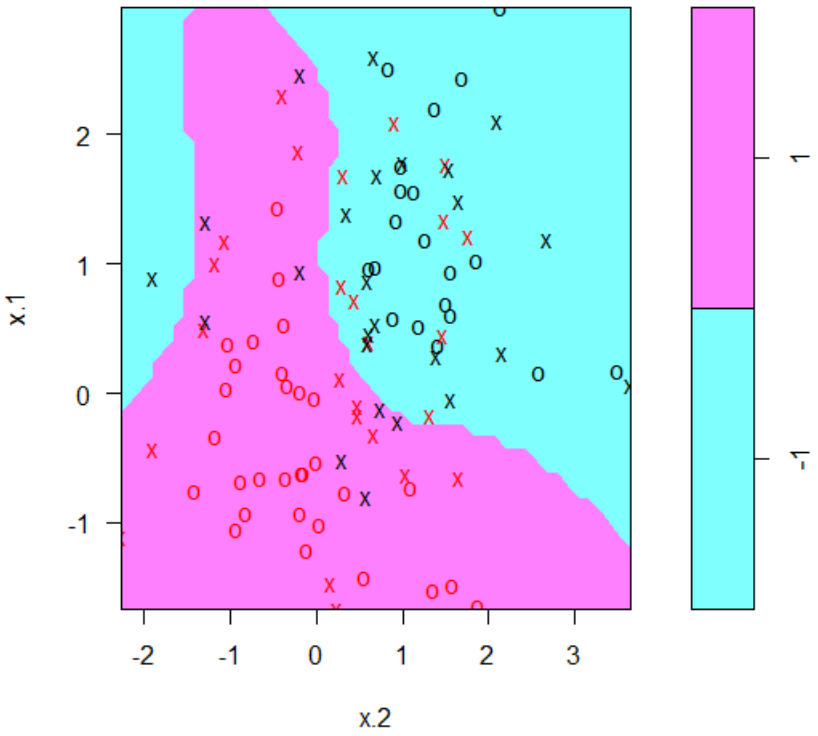

Figure 14: SVM classification plot for the Radial Basis Function kernel best model: The support vectors are marked with a cross

The best model displays 50 support vectors, 25 for the class "-1" and 25 for the class "1". Let's now use the best model function to predict on the test dataset.

	TRUE	
PREDICT	-1	1
-1	38	7
1	17	38

Figure 15: Performance of the best RBF kernel model on the test dataset

Among the 100 observations, 76 are classified correctly, thus an error rate of 24%. This error rate is higher than what we got from the Linear and the polynomial kernels. This poorer performance shows that the radial kernel seems not to be adaptive to classify more efficiently the observations from two overlapping classes.

4.3 SVM on observations with mixed and non-overlapping classes

Let's consider a training set of size $N = N_1 + N_2 + N_3 = 100$ where N_1, N_2, and N_3 are unknown and $i.i.d$ observations respectively generated from the bivariate normal distributions $\mathcal{N}(\mu_1, \Sigma_1)$, $\mathcal{N}(\mu_2, \Sigma_2)$ and $\mathcal{N}(\mu_3, \Sigma_3)$,
$\mu_1 = \begin{pmatrix} -2 \\ -2 \end{pmatrix}$, $\mu_2 = \begin{pmatrix} 2 \\ 2 \end{pmatrix}$, $\mu_3 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$, with $\Sigma_1 = \Sigma_2 = \Sigma_3 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$.
N_1 and N_3 have the same group, and constitute together one sample. We obtain the following plot:

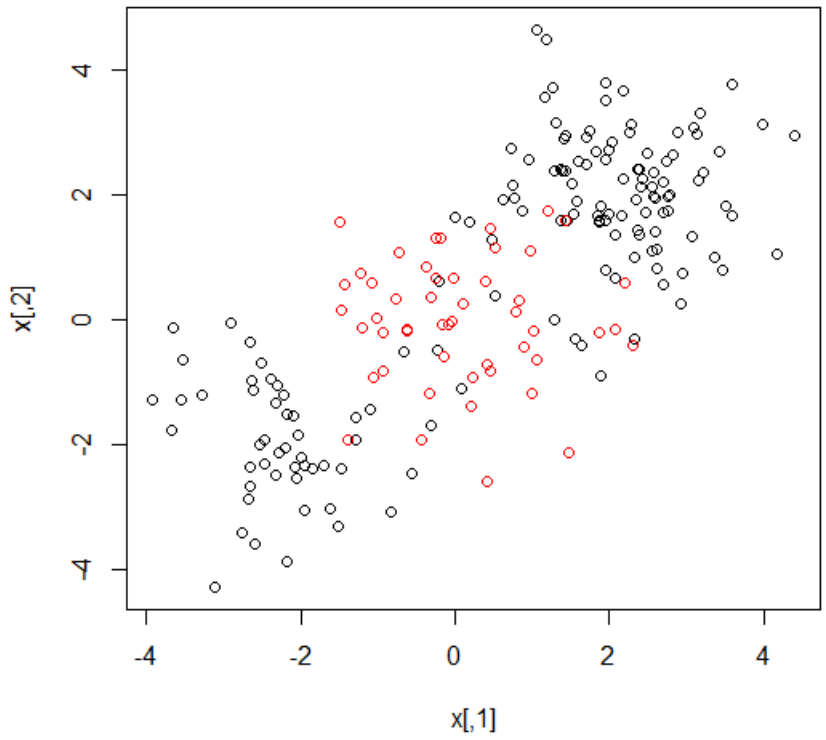

Figure 16: Plot of training dataset for mixed and non-overlapping classes

The plot shows clearly that the two groups are NOT linearly separable, and neither the linear nor the polynomial kernel are suitable to implement the SVM algorithm. The implementation of the SVM with RBF kernel and the choice of the 10-fold cross validation algorithm gives that $gamma = 2$ resulted in the lowest cross validation error rate, thus the best model.

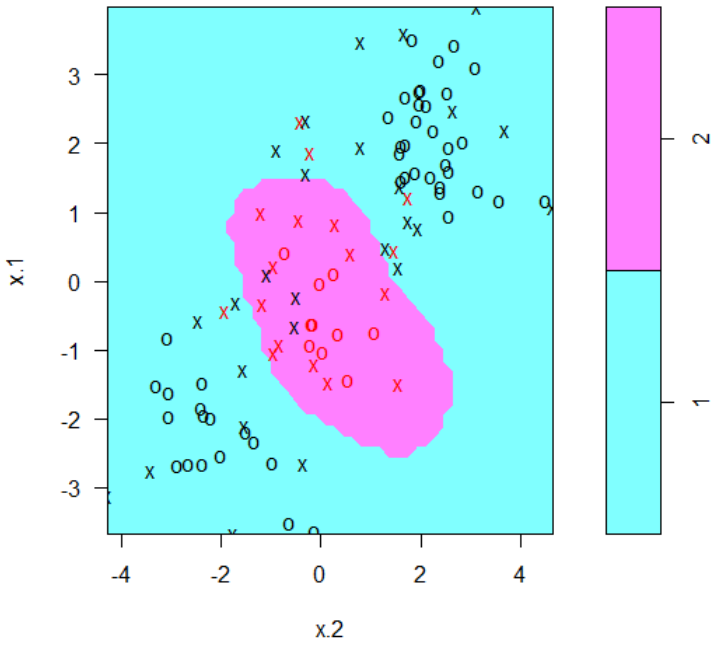

Figure 17: SVM classification plot for the RBF kernel best model: The support vectors are marked with a cross

The best model displays 47 support vectors, 17 for the class "-1" and 26 for the class "1". Let's now use the best model function to predict on a test data generated exactly as the training dataset, i.e. with $N_1 + N_2 = N_3 = N = 100$, N_1, N_2 and N_3 are unknown and are generated from the same bivariate normal distributions as described previously.

	TRUE	
PREDICT	-1	1
-1	74	7
1	3	16

Figure 18: Performance of the best linear kernel model on the test dataset

Among 100 test observations, 90 are classified correctly, thus an error rate of 10%.

5 Discussion on two ad hoc learning classification algorithms based on the linear kernel and RBF kernel

5.1 Ad hoc learning classification algorithm based solely on linear kernel

In this part, we'll first prove that it can be found a universal consistent kernel generated as the sum of inner product kernels from each split made on the training dataset. Secondly, we will empirically verify it convergence towards the Bayes classifier with simulated data as the area of splits are getting smaller (towards 0).

5.1.1 Constuction of the Ad hoc kernel $\mathbb{K}$:

Let's split the training dataset in K splits for linear SVM implementation in each of them and let's consider the following classifier for any test observation x:

$$\widehat{f}(x) = \sum_{j=1}^{K} h_j \left(\beta_{0j} + \sum_{i=1}^{N_j} \alpha_{ij} y_{ij} \underbrace{x^T x_i}_{\langle x, x_i \rangle} \right) \mathbb{I}_{x,x' \in \mathcal{S}_j} \tag{22}$$

with $\mathcal{S}_j$ being the set of all the data observations in the j^{th} split and N_j, the number of the training observations in the j^{th} split.
We now have to justify that the following kernel is well defined:

$$\mathbb{K} = \sum_{j=1}^{K} h_j \langle x, x^{'} \rangle \mathbb{I}_{x,x' \in \mathcal{S}_j}$$

with h_j the weight of $\mathcal{S}_j$ the set of training observations in the j^{th} split, and $\sum_{j=1}^{K} h_j = 1$.
The verification will involve to check if the constructed kernel is **positive definite, and universally consistent**.

- **Positive definiteness**:

Let's X_j be a $N_j \times p$ matrix for each split j. We need to check if the $p \times p$

$X_j^T X_j$ is positive definite. Let u be an arbitrary $p \times 1$ matrix.

$u^T(X_j^T X_j)u = (X_j u)^T X_j u$, setting $X_j u = (v_1, .., v_{N_j})$ which is a matrix $N_j \times 1$

we obtain that:

$$u^T(X_j^T X_j)u = (X_j u)^T X_j u = \sum_{i=1}^{N_j} v_i^2 > 0$$

Thus the positive definiteness is proved.

Then the resulting linear combination kernel $\mathbb{K} = \sum_{j=1}^{K} h_j \langle x, x' \rangle \mathbb{I}_{x,x' \in \mathcal{S}_j}$ is also **positive definite**, i.e. $\mathbb{K} \geq 0$ *[8, chapter 13. pp 408].*

5.1.2 Creation of a dataset to check for convergence to Bayes Classifier

Description of dataset :(*where the two classes have their observations mixed* ***without*** *any pattern*)

Let's generate $N_4 = 250$ observations from a bivariate uniform distribution $\mathcal{U}\Big[(0, 10), (0, 10)\Big]$.

For any observation $X = (x_1, x_2)$, $Y = 1 \quad$ if $\quad \sin(x_1^{x_2}) > 0$,

else $Y = -1$

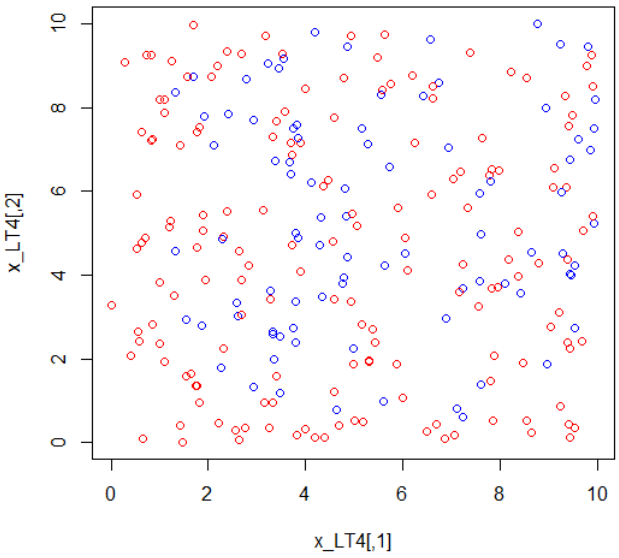

Figure 19: Plot of the two classes from the uniform distribution

We first implement the Radial Basis Function kernel (RBF) with soft margin. For $k = 10$, the cross validation algorithm gives that the *cost* $= 0.001$ resulted in the lowest cross validation error rate, thus the best model.

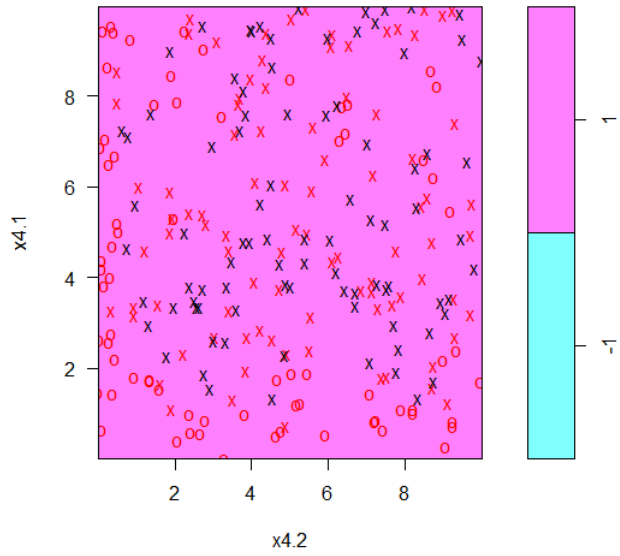

Figure 20: SVM classification plot for the RBF kernel best model : The support vectors are marked with a cross

Let's now use the best model function to predict on a test data generated exactly as the training dataset, i.e. with $N_4 = 250$ observations generated

from the same bivariate uniform distribution as described previously.

	TRUE	
PREDICT	-1	1
-1	78	67
1	44	61

Figure 21: Performance of the best linear kernel model on the test dataset

Among 250 test observations, 111 are misclassified thus an error rate of 44.4%.

We then implement the linear kernel with soft margin. For $k = 10$, the cross validation algorithm gives that the $cost = 0.001$ resulted in the lowest cross validation error rate, thus the best model.

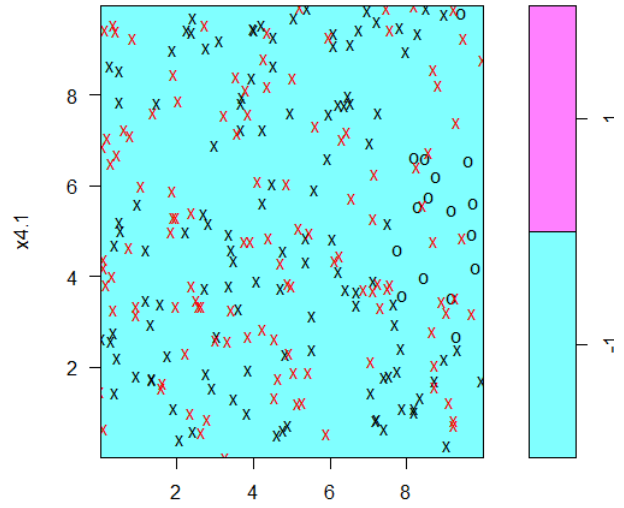

Figure 22: SVM classification plot for the linear kernel best model : The support vectors are marked with a cross

Let's now use the best model function to predict on a test data generated exactly as the training dataset, i.e. with $N_4 = 250$ observations generated from the same bivariate uniform distribution as described previously.

	TRUE	
PREDICT	-1	1
-1	122	128
1	0	0

Figure 23: Performance of the best linear kernel model on the test dataset

Among 250 test observations, 128 are misclassified thus an error rate of 51.2% higher than the RBF error rate.

5.1.3 Splitting the dataset in subsets and checking for its convergence to Bayes classifier

Let's split the the training dataset into 16 (subsets) sub-quadrants of equal area (2.5×2.5). For the uniformly distributed dataset for the two classes, we will implement linear SVM in each sub-quadrant **of same weight (area)** h_j**,** $j = 1, ..., 16$ to check whether their overall error rate is lower than that of its test dataset(generated as the training dataset).
We obtain the following results in each subsets SS1,SS2,...,SS16:

SS1

	TRUE	
PREDICT	-1	1
-1	8	6
1	0	0

SS2

	TRUE	
PREDICT	-1	1
-1	1	4
1	9	6

SS3

	TRUE	
PREDICT	-1	1
-1	2	1
1	3	7

SS4

	TRUE	
PREDICT	-1	1
-1	6	5
1	0	1

SS5

	TRUE	
PREDICT	-1	1
-1	4	15
1	0	0

SS6

	TRUE	
PREDICT	-1	1
-1	0	0
1	8	6

SS7

	TRUE	
PREDICT	-1	1
-1	7	8
1	0	0

SS8

	TRUE	
PREDICT	-1	1
-1	3	8
1	0	0

SS9

	TRUE	
PREDICT	-1	1
-1	5	2
1	5	4

SS10

	TRUE	
PREDICT	-1	1
-1	6	6
1	4	5

SS11

	TRUE	
PREDICT	-1	1
-1	0	0
1	8	4

SS12

	TRUE	
PREDICT	-1	1
-1	10	7
1	0	0

SS13

	TRUE	
PREDICT	-1	1
-1	10	3
1	0	0

SS14

	TRUE	
PREDICT	-1	1
-1	3	2
1	4	6

SS15

	TRUE	
PREDICT	-1	1
-1	0	0
1	10	12

SS16

	TRUE	
PREDICT	-1	1
-1	0	2
1	6	8

Result:

$$\begin{aligned}&\text{Overall mean error rate} = \\ &= \frac{\text{total number of misclassified observations in each test subset}}{\text{total number of observations in the test dataset}} \\ &= 50.4\%\end{aligned}$$

51.2% and 44.4% are respectively the linear and RBF kernels error rate from the whole test dataset.

Performing the Ad Hod algorithm has led us to a lower error rate when compared to the linear kernel. However this error is higher when compared to the RBF kernel. This is explained by the arbitrary splits operated in the construction of kernel $\mathbb{K}$. This abnormality will be resolved with a refined version of kernel $\mathbb{K}$ discussed in the following subsection, and which will give the lower error rate(than the RBF kernel) we were aiming for in order to get closer to the the Bayes Classifier error rate as expected. **The kernel $\mathbb{K}$ was just a motivating idea.**

5.2 Ad hoc learning classification algorithm based on both linear kernel and RBF kernel

Similarly to the previous, we'll prove here that it can be found a universal consistent kernel generated as the the product of inner product kernel and the Radial Basis Function kernel on the whole training dataset.
The motivation for such a kernel comes from the fact that the splits previously operated led us to $\langle x, x' \rangle \mathbb{I}_{x,x' \in \mathcal{S}_j}$. While the splits arbitrarily causes a loss of data, this inconvenience is corrected by replacing $\mathbb{I}_{x,x' \in \mathcal{S}_j}$ by $exp(-\gamma \|x - x'\|^2)$ for some $\gamma > 0$. In other words, instead of entirely discarding points x' that are too far away from x, we are down-weighting them according to their distance from x. This leads us to the product kernel $\langle x, x' \rangle exp(-\gamma \|x - x'\|^2)$

5.2.1 Constuction of the ad hoc kernel $\mathbb{Q}$:

Let's take the whole training dataset and let's consider the following classifier for any test observation x:

$$\widehat{f}(x) = \sum_{i=1}^{N} \alpha_i y_i \underbrace{\langle x^T, x_i \rangle exp(-\gamma \|x - x_i\|^2)}_{\langle \Phi(x), \Phi(x_i) \rangle} \tag{23}$$

Here, the constructed kernel $\mathbb{Q}$ is **positive definite, and universally consistent**.

- **Positive definiteness**:

Firstly, let's show that the RBF kernel $k'(x, x') = exp(-\gamma \|x - x'\|^2)$.
Since this kernel assumes a map Φ' from the input space $\mathcal{X}$ to a hilbert feature space $\mathcal{H}$ such that: $k'(x, x') = \langle \Phi'(x), \Phi'(x') \rangle$ Where $\Phi'(x)^T = (h_1(x), h_2(x), ...) = (h_1(x), h_2(x), ...h_M(x))$ with $M \to \infty$

Let's X be a $N \times M$ matrix such that:

$$X = \begin{bmatrix} h_1(x_1) & h_2(x_1) & h_3(x_1) & \dots & h_M(x_1) \\ h_1(x_2) & h_2(x_2) & h_3(x_2) & \dots & h_M(x_2) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ h_1(x_N) & h_2(x_N) & h_3(x_N) & \dots & h_M(x_N) \end{bmatrix} = \begin{bmatrix} \Phi'(x_1) \\ \Phi'(x_2) \\ \vdots \\ \Phi'(x_N) \end{bmatrix}$$

We need to check if the $M \times M$ infinite matrix $\mathbb{K}' = X^T X$ is positive definite. Let u be an arbitrary $M \times 1$ matrix.

$u^T(X^T X)u = (Xu)^T Xu$, knowing that $k(x_j, x_k) = \langle \Phi'(x_j), \Phi'(x_k) \rangle = \Phi'(x_j)^T \Phi'(x_k)$ $j, k = 1, 2, ..., N$, there exist α_j, α_k such that:

$$u^T(X^T X)u = (Xu)^T Xu = \sum_{j,k=1}^{N} \alpha_j \alpha_k k(x_i, x_j)$$

Let $k(x_j, x_k) = exp(-\gamma \|x_j - x_k\|^2) = f(\sqrt{2\gamma}\ \mathbb{I}^T(x_j - x_k))$
with $f(t) = exp(-\frac{t^2}{2}) = E[e^{itZ}]$ which is the characteristic function of random variable Z with $\mathcal{N}(0, 1)$ distribution where $\mathbb{I}$ is the unit matrix and i is a complex number.

$$\begin{aligned} u^T(X^T X)u = (Xu)^T Xu &= \sum_{j,k=1}^{N} \alpha_j \alpha_k k(x_j, x_k) \\ &= \sum_{j,k=1}^{N} \alpha_j \alpha_k E[e^{i\sqrt{2\gamma}\ \mathbb{I}^T(x_j - x_k)Z}] \\ &= E\left[\sum_{j,k=1}^{N} \alpha_j e^{i\sqrt{2\gamma}\ \mathbb{I}^T x_j} \alpha_k e^{-i\sqrt{2\gamma}\ \mathbb{I}^T x_k} \right] \\ &= E\left[\left| \sum_{j=1}^{N} \alpha_j e^{i\sqrt{2\gamma}\ \mathbb{I}^T x_j} \right|^2 \right] > 0 \end{aligned}$$

Thus the positive definiteness of the RBF kernel $\mathbb{K}'$ is proved.

Since we proved earlier that the linear kernel $\mathbb{K}$ is positive definite then $\mathbb{Q} = \mathbb{K}.\mathbb{K}'$ is also positive definite.*[8, chapter 13. pp 408]*

- **Universal Consistency**:

Recall from Section 3.2 that in the kernel approach, each data point x_i in the *p-dimensional space* is transformed via a feature map $h(x_i)$ in a enlarged *higher-dimensional space* such that:

$$h(x_i) = (h_1(x_i), h_2(x_i), ...) \;\; i = 1, ..., N$$

and the expression of the kernel is $k(x_i, x_j) = \langle h(x_i), h(x_j) \rangle$.
Let's assume that $span\{h(x_i) : i \geq 1\}$ forms a sub-algebra of $C(X)$ where $C(X)$ is the subspace of complex-valued continuous functions on X, the original input space for the training observations $x_i's$.
Let k be kernel on X and $h : X \to H$. A function $f : X \to \mathbb{R}$ is induced by k if there exists an element in $w \in H$ such that $f = \langle w, h(.) \rangle$. [10, sec.3, pp 71].
In [8,Definition 2] A continuous kernel $k : K \times K \longrightarrow \mathbb{R}$ is called *universal* if the set of all induced functions is dense in $C(X)$ i.e., for all $g \in C(X)$ and all $\varepsilon > 0$ there exist a function f induced by k with $\|f - g\|_\infty \leqslant \varepsilon$.

Let's prove universality of kernel $\mathbb{Q}$. We know that:

$$\begin{aligned} f(x) = \frac{1}{\sigma^2} x (e^x)^2 = \frac{1}{\sigma^2} x e^{2x} = x \sum_{n=0}^{\infty} \frac{(2x)^n}{n!} \\ = \sum_{n=1}^{\infty} \frac{2^n}{\sigma^2 (n-1)!} x^n \\ = \sum_{n=1}^{\infty} a_n x^n \;\; \text{with} \;\; a_n = \frac{2^n}{\sigma^2 (n-1)!} \end{aligned}$$

We obtain f as a C^∞- function that can be expanded into a Taylor series

in 0, hence by [10, Corollary 10], $f(\langle \sigma x, \sigma y \rangle) = \frac{1}{\sigma^2}\langle \sigma x, \sigma y \rangle \left(e^{\langle \sigma x, \sigma y \rangle} \right)^2$ is an universal kernel.
Consequently, by taking $\gamma = \sigma^2$ in the expression of $\mathbb{Q}(x, y)$, we have:

$$\begin{aligned} \mathbb{Q}(x,y) &= \langle x, y \rangle e^{-\sigma^2 \|x-y\|^2} = \frac{1}{\sigma^2}\langle \sigma x, \sigma y \rangle e^{-\sigma^2(\langle x,x\rangle + \langle y,y\rangle - 2\langle x,y\rangle)} \\ &= \frac{1}{\sigma^2} \frac{\langle \sigma x, \sigma y \rangle (e^{\langle \sigma x, \sigma y \rangle})^2}{e^{\langle \sigma x, \sigma x \rangle} e^{\langle \sigma y, \sigma y \rangle}} \end{aligned}$$

We have already proved above that the numerator is an universal kernel. By [10,Proposition 8], we can conclude that $\mathbb{Q}$ is also an universal kernel.

To prove the consistency, let's states a **few definitions and results** :

- In [3, Lemma 1], if $k : X \times X \to \mathbb{R}$ is a universal kernel on a compact subset X of $\mathbb{R}^p$ and $\Phi : X \to H$ is a feature map on k, then Φ is continuous and

$$d_k(x, x') := \|\Phi(x) - \Phi(x')\|$$

defines a metric on X such that $id : (X, |.|) \to (X, d_k)$ is continuous.

- The above metric can always be used in our context to define a finite covering number defined as:

$$N(X, d_k) := inf\{n \in \mathbb{N} : \exists x_1, x_2, ..., x_n \ \ with \ \ X \subset \cup_{i=1}^{n} B_{d_k}(x_i, \varepsilon)\}$$

for all $\varepsilon > 0$, so (X, d_k) is precompact.
By assuming that the metric space (X, d_k) is also complete (*if every Cauchy sequence of points in X has a limit that is also in X*), we conclude that (X, d_k) is **compact** (*precompact and complete*).

- Finally, let's f_T be the measurable decision function constructed on the training set $T = ((x_1, y_1), ..., (x_N, y_N)) \in (X, Y)^N$ generated from the

distribution P such that $f_T : X \longrightarrow Y$ with $Y = \{-1, 1\}$.
Let's define $\mathcal{R}_P(f_T) = P(\{(x, y) : f_T(x) \neq y\})$ as the risk of f_T.
The **Bayes Risk** is then defined as follows:
$\mathcal{R}_P = inf\{\mathcal{R}_P(f) : f : X \longrightarrow Y\}$ where f is a measurable function. Notice that the Bayes risk is does not depend on the training set T.

The new kernel we have constructed $\mathbb{Q}$ is consistent if it induces a function f_T in the feature space such that : $\mathcal{R}_P(f_T) \longrightarrow \mathcal{R}_P$ as T gets large enough.

- So, with an universal kernel on a compact metric space, the following theorem establishes its almost **consistency** as follows:

THOEREM:[3, Theorem 1] Let $X \subset \mathbb{R}^p$ be compact and $\mathbb{Q} : X \times X \longrightarrow \mathbb{R}$ be a universal kernel. Then for all Borel probability measures P on $X \times Y$ and all $\varepsilon > 0$ there exist a constant $c^* > 0$ and some constant $M > 0$ such that for all $c \geq c^*$ and $N \geq 1$ we have:

$$Pr^*(\{T \in (X \times Y)^N : \mathcal{R}_P(f_T^{\mathbb{Q},c/N}) \leq \mathcal{R}_P + \varepsilon\}) \geq 1 - 2Me^{-(\varepsilon^6/2^{29}M^2)N},$$

where Pr^* is the outer probability of P^N and $M := \frac{64}{\varepsilon} N\left((X, d_k), \frac{\varepsilon}{32\sqrt{c}}\right)$ and $f_T^{\mathbb{Q},c/N}$ is the function in the feature space induced by the kernel $\mathbb{Q}$, depending on N and the constant c .

5.2.2 Expression for the metric d_k on the feature space where $\mathbb{Q}$ is implemented

We defined $q(x, x') = \langle x, x' \rangle exp(-\gamma \|x - x'\|^2)$ as the expression for the Ad hoc kernel $\mathbb{Q}$.
Recall that $\mathbb{Q}$ assumes a map Φ_Q from the input space $\mathcal{X}$ to a hilbert feature space $\mathcal{H}$ such that:

$q(x, x') = \langle \Phi_Q(x), \Phi_Q(x') \rangle$ Where $\Phi_Q(x)^T = (\Phi_1(x), \Phi_2(x), \Phi_3(x)....)$
The metric d_Q can then be expressed as:

$$
\begin{aligned}
(d_Q(x, x'))^2 &= \|\Phi_Q(x) - \Phi_Q(x')\|^2 \\
&= \langle \Phi_Q(x) - \Phi_Q(x'), \Phi_Q(x) - \Phi_Q(x') \rangle \\
&= \langle \Phi_Q(x) - \Phi_Q(x) \rangle - 2\langle \Phi_Q(x) - \Phi_Q(x') \rangle + \langle \Phi_Q(x') - \Phi_Q(x') \rangle \\
&= q(x, x) - 2q(x, x') + q(x', x') \\
&= \langle x, x \rangle - 2\langle x, x' \rangle exp(-\gamma \|x - x'\|^2) + \langle x', x' \rangle \\
&\text{(by using the above expression of } q(x, x')) \\
&= \langle x, x \rangle + \langle x', x' \rangle - 2\langle x, x' \rangle + 2\langle x, x' \rangle - 2\langle x, x' \rangle exp(-\gamma \|x - x'\|^2) \\
&= \|x - x'\|^2 + 2\langle x, x' \rangle (1 - exp(-\gamma \|x - x'\|^2)) \\
&= \|x - x'\|^2 \left(1 + 2\langle x, x' \rangle \frac{1 - exp(-\gamma \|x - x'\|^2)}{\|x - x'\|^2} \right) \\
&\to \|x - x'\|^2 \left(1 - 2\gamma \langle x, x' \rangle \right) \\
&\text{as } x \to x' \text{ i.e. } \gamma \|x - x'\| \to 0 \\
&\text{and since } \frac{1 - e^x}{x} \to -1 \text{ as } x \to 0
\end{aligned}
$$

5.2.3 Implementation of the ad hoc kernel $\mathbb{Q}$:

We use the the same dataset as previously in implementing ad hoc kernel $\mathbb{K}$: we generated $N_4 = 250$ observations from a bivariate uniform distribution $\mathcal{U}\Big[(0, 10), (0, 10)\Big]$.

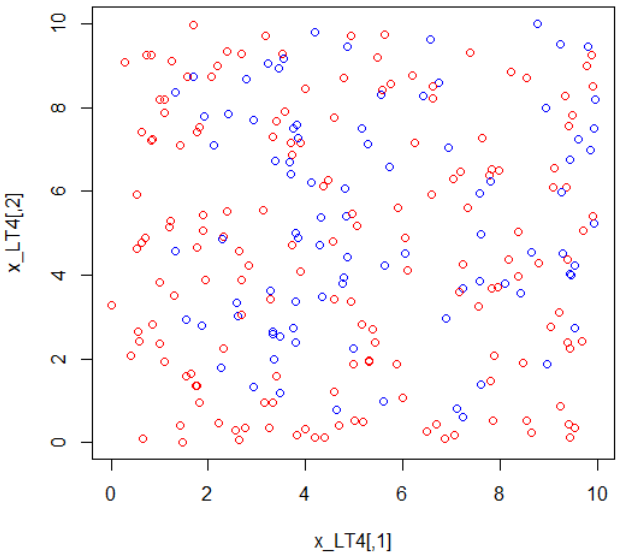

Figure 24: Plot of the two classes from the uniform distribution

The kernel trick allows to replace the inner product by the expression the kernel Q and find the Lagrange multipliers by solving (maximazing the resulting Dual or minimizing its opposite):

$$\min_{\alpha} \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j Q(x_i, x_j) - \sum_{i=1}^{N} \alpha_i$$

$$\text{subject to } \ 0 \leq \alpha_i \leq C \ \text{ where } C \text{ is the cost,}$$

$$\text{and } \ \sum_{i=1}^{N} \alpha_i y_i = 0$$

With the Quadratic Programming (QP) library(quadprog) in R, the following matrix expression is used instead:

$$\min_{\alpha} \alpha^T D - d^T \alpha$$

$$\text{subject to } \ A\alpha \geq \beta_0$$

where $D = yy^T Q$, $d = e$ (all ones vectors), A is the matrix defining the constraints under which we want to minimizing the quadratic function.

Once the α_i are obtained, the prediction of an observation x in the test dataset is given by *sign(f(x))* with:

$$f(x) = \sum_{i=1}^{N_{train}} \alpha_i y_i Q(x_i, x) + \beta_{0,x_i}$$

where by using training dataset, we compute for each obseservation x_i

$$\beta_{0,x_i} = y_i - \sum_{j=1}^{N_{train}} \alpha_j y_j Q(x_i, x_j)$$

Result: **The error rate has given us a percentage of** 42.72% after using cross validation to find the best values for the parameters γ and C. The intuition that motivated the construction of the kernel Q is then well verified: we got a performance is better than that of the linear and the RBF kernel which are respectively 51.2% and 44.4%

6 Conclusion

The geometric polynomial kernel approach ($K(x_i, x_j) = (1 + \langle x_i, x_j \rangle)^d$) is somehow comparable with the probabilistic Quadratic Discriminant Analysis approach as they both lead to a polynomial classifier. However, in the case of normal distribution, while the QDA approach converges to the Bayes classifier (is the Bayes classifier when true parameters of the distributions are used), the geometric approach does not. Rather, the polynomial kernel performs better on distributions which are not Gaussian (*e.g.* with overlapping classes), as the parameter d can be adjusted to lead to a performance closer to the Bayes classifier's performance, but no convergence happens for the polynomial kernel. The choice of d should then be chosen adequately to avoid over-fitting *i.e. avoid a performance better than Bayes classifier.*

Likewise, and similarly to the KNN probabilistic approach and the Gaussian Radial Basis Function kernel geometrical approach are somehow comparable as they both have a local and cluster based separation. However the Gaussian RBF kernel $exp(-\gamma \|x_i - x_j\|^2)$ uses all the training observations by computing the euclidean distance between them and by classifying a defined cluster as one class, and every other observations as the other class. This is the reason explaining the poorer performance of RBF kernel on overlapping classes, as compared to polynomial kernel, or linear kernel (with soft margin). On the other hand, the KNN approach does not converge to the Bayes classifier. In fact the KNN method doesn't encompass all the training when it mimics the Bayes classifier, whereas the RBF kernel learning rate decreases as the training sample size increases.

Finally, we also acknowledge the fact that the splits of the dataset with ad hoc kernel $\mathbb{K}$ and its approximation with ad hoc kernel $\mathbb{Q}$ decrease the error rate, and bring closer to the Bayes classifier error rate, as theoretically proved beforehand with the universal consistency criteria.

References

[1] Hastie T.,Tibshirani R., and Friedman J. (2009). *The elements of statistical learning: Data mining, inference, and prediction.* Stanford, California: Springer Series in Statistics.

[2] Hastie T.,Tibshirani R.,James G., and Witten D. (2013). *An introduction to statistical learning: With application in R.* New york, USA: Springer Texts in Statistics.

[3] Steinwart I., *Support Vector Machines are universally consistent*, *Journal of complexity*, 18, (2002)768-791.

[4] Wikipedia contributors.(2018, June 21). VC dimension. In Wikipedia, The Free Encyclopedia. Retrieved 20:15, August 5, 2018, from https://en.wikipedia.org/w/index.php?title=VC_dimension&oldid=846805558.

[5] SANDIPAN DEY.(2018, April 23). Implementing a Soft-Margin Kernelized Support Vector Machine Binary Classifier with Quadratic Programming in R and Python. Retrieved from https://sandipanweb.wordpress.com/2018/04/23/implementing-a-soft-margin-kernelized-support-vector-machine-binary-classifier-with-quadratic-programming-in-r-and-python/

[6] Godin, F. (2018). *Statistical learning: chapter 5* [Lecture notes].

[7] Godin, F. (2018). *Statistical learning: chapter 11* [Lecture notes].

[8] Schlkopf, B., Smola A. (2001).*Learning with Kernels.*Cambridge, Massachusetts: The MIT Press

[9] Steinwart, On the influence of the kernel on the consistency of support vector machines, Journal of Machine Learning Research 2 (2001) 67-93.

```
FILE 1: ############################ Implementation of Linear, Polynomial
and RBF kernels ##############################

#Support vector machine:
n=200   #intial: n=200


##-I- we will first implement the classical SVM with linearly separable
case along with a soft margin

###-Creation of 2 samples of bivariate normal distribution of size 100
each. The first sample is i.i.d from N(1,sigma)
###-and the second is i.i.d N(0,sigma)
set.seed(1)

x=matrix(rnorm(2*n),ncol=2) # 200 observations
x[1:n/2,]=x[1:n/2,]+1       # 100 first observations with mean "1"
                            # 100 folllowing observations with mean "0"
(standard normal)
y=c(rep(-1,n/2), rep(1,n/2))  # labelling the 100 first observations(that
are shifted) by the same class (say "-1")
                            # and the remaining are labelled another
class ( say "1")

# The first n/2 obsevations with y=-1 are painted in Blue, and the last
100 in Red

###- Let's check if classes are linearly separable in 100 randomly chosen
observations
train=sample(n,n/2) # choose randomly n/2 indexes (observations) among
the n

dat=data.frame(x=x,y=as.factor(y))
dat_LT=dat[train,]
x_LT=x[train,]
y_LT=y[train]

plot(x_LT,col=(3-y_LT))#col=2 means Blue, col=4 means Red


#Comment# The plot shows that the two groups are NOT linearly separable
(except with soft margin)
###-We use the following package to implement svm

install.packages("e1071")

###-and we call this library...
```

```
library(e1071)

###Cross validation using "tune" to select the best model among the cost
values  of our range
set.seed(1)
tune.out=tune(svm,y~.,data=dat_LT,
kernel="linear",ranges=list(cost=(c(0.001, 0.01, 0.1, 1, 5, 10, 100))))
summary(tune.out)

#Comment# cost 0.1 resulted in the lowest cross validation error rate,
thus the best model
bestmodel_L=tune.out$best.model
plot(bestmodel_L, dat_LT)
summary(bestmodel_L)

#Comment# The summary of the best model displays 63 support vectors, 32
for the class "-1" and 31 for the class "1"

### using the best model function to predict on a test data made of the
remaining observations in the dataset
test=-train
dat_Lt=dat[test,]  # Lt= Linear test
ypred=predict(bestmodel_L, dat_Lt)
table(predict=ypred, true=dat_Lt$y)



#Comment# Among n/2=100 observations, 86 are classified correctly, thus
an error rate of 0.14 %

##########################################################################
#########################################################
##########################################################################
#########################################################

###- Let's then try the polynomial kernel and check its performance on
the same dataset
dat_PT=dat_LT # LT= Linear Training, PT=Polynomial Training

tune.out=tune(svm,y~.,data=dat_PT,
kernel="polynomial",ranges=list(cost=c(0.001, 0.01, 0.1, 1, 5, 10, 100),
degree=c(1,2,3,4,5)))
summary(tune.out)
#Comment# cost=5, and degree=1 resulted in the lowest cross validation
error rate, thus the best model

bestmodel_P=tune.out$best.model
plot(bestmodel_P, dat_PT)
summary(bestmodel_P)

#Comment# The summary of the best model displays 52 support vectors, 26
for the class "-1" and 26 for the class "1"

### using the best model function to predict on a test data made of the
remaining observations in the dataset

dat_Pt=dat[test,]  # Pt= Polynomial test
ypred=predict(bestmodel_P, dat_Pt)
table(predict=ypred, true=dat_Pt$y)
```

```
#Comment# Among n/2=100 observations, 86 are classified correctly, thus
an error rate of 0.14 %, similar to the Linear kernel case

##########################################################################
#########################################################
##########################################################################
#########################################################

###- Let's then try the radial kernel and check its performance on the
same dataset
dat_RT=dat_LT # LT= Linear Training, RT=Radial Training

tune.out=tune(svm, y~., data=dat[train,],
kernel="radial",ranges=list(cost=c(0.1, 1, 10, 100, 1000),gamma=c(0.5, 1,
2, 3, 4)))
summary(tune.out)
#Comment# cost=0.1, and gamma=0.1 resulted in the lowest cross validation
error rate, thus the best model

bestmodel_R=tune.out$best.model
plot(bestmodel_R, dat_RT)
summary(bestmodel_R)

#Comment# The summary of the best model displays 50 support vectors, 25
for the class "-1" and 25 for the class "1"

### using the best model function to predict on a test data made of the
remaining observations in the dataset

dat_Rt=dat[test,]  # Rt= Radial test
ypred=predict(bestmodel_R, dat_Rt)
table(predict=ypred, true=dat_Rt$y)

#Comment# Among n/2=100 observations, 76 are classified correctly, thus
an error rate of 0.24 %, higher than what we got from
        #  the Linear and the polynomial kernels. So the radial kernel
seeems not to be adaptive the classifier better the
        # observations from two overlapping classes.


##############################Discuss the case comparing linear and
polynomial when n increases##############################
#We noticed that by increasing to n=400, the error rate for both the
linearr and the polynomial kernels are higher although
#they have been given a larger training sample.So as n goes to infinity,
neither of them converge to the bayes classifier.
# Since cross validation has already chosen the optimal parameter for the
best model, any attempt to change these would lead
# to overfitting, low error rate in the training sample, but NOT
convergence to Bayes classifier as the resulting model
# would perform poorly on the test dataset.
##########################################################################
################################################


##########################################################################
#########
##########################################################################
##############
#Using Radial kernel and playing with gamma and cost
```

```
#Kernel="radial"->Uses radial kernel, we need to set gamma
set.seed(1)


x=matrix(rnorm(n*2),ncol=2) # n observations
x[1:n/2,]=x[1:n/2,]+2       # 100 first observations with mean "2"
x[((n/2)+1):((n/2)+(n/2)/2),]=x[((n/2)+1):((n/2)+(n/2)/2),]-2   # 50
folllowing observations with mean "-2" and the 50 remaining have mean "0"
(standard normal)
y=c(rep(1,(n/2)+(n/2)/2), rep(2,(n/2)/2))  # labelling the 150 first
observations(that are shifted) by the same class (say "-1")
                                           # and the remaining are
labelled another class ( say "1", or "2" for convinience)



###- Let's check if classes are linearly separable in n/2=100 randomly
chosen observations
# train=sample(n,n/2) # choose randomly n/2 indexes (observations) among
the n

dat=data.frame(x=x,y=as.factor(y))
dat_R=dat[train,]  # R= Radial
x_R=x[train,]
y_R=y[train]
plot(x,col=y)


#Comment# The plot shows that the two groups are NOT linearly separable,
and neither the linear nor the polynomial kernel are suitable
        # to implement the svm algorithm

###Cross validation using "tune" to select the best model among the cost
values  of our range
set.seed(1)
tune.out=tune(svm,y~.,data=dat_R, kernel="radial",ranges=list(cost=c(0.1,
1, 10, 100, 1000),gamma=c(0.5, 1, 2, 3, 4)))
summary(tune.out)

#Comment# cost=1 and gamma=2 resulted for the lowest cross validation
error rate, thus the best model
bestmodel_R=tune.out$best.model
plot(bestmodel_R, dat_R)
summary(bestmodel_R)

#Comment# The summary of the best model displays 43 support vectors, 17
for the class "-1" and 26 for the class "1"

### using the best model function to predict on a test data made of the
remaining observations in the dataset
test=-train
dat_Rtest=dat[test,]  # Rtest= Radial test
ypred=predict(bestmodel_R, dat_Rtest)
table(predict=ypred, true=dat_Rtest$y)

#Comment# Among n/2=100 observations, 90 are classified correctly, thus
an error rate of 0.10 %
```

```
FILE 2:########################## Implementing kernel K (linear svm in
each of 16 splits)############################

mysamp=function(n, m, s, lwr, upr, rounding) {
samp = round(rnorm(n, m, s), rounding)
samp[samp < lwr]= lwr
samp[samp > upr] = upr
samp
}

install.packages("e1071")

###-and we call this library...
library(e1071)




set.seed(1)
n4=500

x41= matrix(runif(n4,0,10), ncol=1)

x42= matrix(runif(n4,0,10), ncol=1)


 x4= cbind(x41,x42)

y4=sample(c(-1,1),n4,rep=TRUE)
for (i in 1:n4){
#if (tan(x41[i]+x42[i])/sin(x41[i]^x42[i])>0){y4[i]=1}
if (sin(x41[i]^x42[i])>0){y4[i]=1}
else {y4[i]=-1}
}
y4

###- Let's check if classes
```

```
train=sample(n4,n4/2) # choose randomly n4/2 indexes (observations) among
the n

dat=data.frame(x4=x4,y4=as.factor(y4))
dat_LT4=dat[train,]  # no= no pattern
x_LT4=x4[train,]
y_LT4=y4[train]
plot(x_LT4,col=3-y_LT4)

#########################################################SVM4(Radial)


tune.out=tune(svm,y4~.,data=dat_LT4,
kernel="radial",ranges=list(cost=c(0.001, 0.01, 0.1, 1, 5, 10,
100,1000),gamma=c(0.5, 1, 2, 3, 4)))
summary(tune.out)

#Comment# cost=0.001 and gamma=0.5 resulted for the lowest cross
validation error rate, thus the best model
bestmodel_L4=tune.out$best.model
plot(bestmodel_L4, dat_LT4)
summary(bestmodel_L4)



### using the best model function to predict on a test data made of the
remaining observations in the dataset
test=-train
dat_Lt4=dat[test,]  # Lt4= Linear test 4th sample
ypred=predict(bestmodel_L4, dat_Lt4)
table(predict=ypred, true=dat_Lt4$y4)

#Comment# Among n4/2=250 observations, 111 are misclassified , thus an
error rate of 44.4 %




#########################################################SVM4 (Linear)


tune.out=tune(svm,y4~.,data=dat_LT4,
kernel="linear",ranges=list(cost=(c(0.001, 0.01, 0.1, 1, 5, 10,
100,1000))))
summary(tune.out)

#Comment# cost=0.001 and gamma=0.5 resulted for the lowest cross
validation error rate, thus the best model
bestmodel_L4=tune.out$best.model
plot(bestmodel_L4, dat_LT4)
summary(bestmodel_L4)



### using the best model function to predict on a test data made of the
remaining observations in the dataset
#test=-train
dat_Lt4=dat[test,]  # Lt4= Linear test 4th sample
ypred=predict(bestmodel_L4, dat_Lt4)
```

```
table(predict=ypred, true=dat_Lt4$y4)

#Comment# Among n4/2=250 observations, 128 are misclassified , thus an
error rate of 51.2 %



############################################################(
SS1:)############################################################
set.seed(1)


dat_SS1_LT4=subset(dat_LT4, x4.1<=2.5 & x4.2<=2.5)



#########################################################SVM4_SS1

tune.out=tune(svm,y4~.,data=dat_SS1_LT4,
kernel="linear",ranges=list(cost=(c(0.001, 0.01, 0.1, 1, 5, 10, 100))))
summary(tune.out)

#Comment# cost 5 resulted in the lowest cross validation error rate, thus
the best model
bestmodel_SS1_L4=tune.out$best.model
plot(bestmodel_SS1_L4, dat_SS1_LT4)
summary(bestmodel_SS1_L4)




### using the best model function to predict on a test data made of the
remaining observations in the dataset
dat_SS1_Lt4=subset(dat_Lt4,x4.1<=2.5 & x4.2<=2.5)  # Lt4= Linear test 4rd
sample
ypred=predict(bestmodel_SS1_L4, dat_SS1_Lt4)
table(predict=ypred, true=dat_SS1_Lt4$y4)

#Comment# Among 14 observations, 6 are misclassified, thus an error rate
of 42.857 %


############################################################( SS2
)############################################################
set.seed(1)


dat_SS2_LT4=subset(dat_LT4, x4.1>2.5 & x4.1<=5 & x4.2<=2.5)



#########################################################SVM4_SS2

tune.out=tune(svm,y4~.,data=dat_SS2_LT4,
kernel="linear",ranges=list(cost=(c(0.001, 0.01, 0.1, 1, 5, 10, 100))))
summary(tune.out)

#Comment# cost 0.001 resulted in the lowest cross validation error rate,
thus the best model
bestmodel_SS2_L4=tune.out$best.model
```

```
plot(bestmodel_SS2_L4, dat_SS2_LT4)
summary(bestmodel_SS2_L4)



### using the best model function to predict on a test data made of the
remaining observations in the dataset
dat_SS2_Lt4=subset(dat_Lt4,x4.1>2.5 & x4.1<=5 & x4.2<=2.5)
ypred=predict(bestmodel_SS2_L4, dat_SS2_Lt4)
table(predict=ypred, true=dat_SS2_Lt4$y4)

#Comment# Among 20 observations, 13 are misclassified, thus an error rate
of 65 %



############################################################( SS3:
)############################################################
set.seed(1)


dat_SS3_LT4=subset(dat_LT4, x4.1>5 & x4.1<=7.5 & x4.2<=2.5)



#########################################################SVM4_SS3

tune.out=tune(svm,y4~.,data=dat_SS3_LT4,
kernel="linear",ranges=list(cost=(c(0.001, 0.01, 0.1, 1, 5, 10, 100))))
summary(tune.out)

#Comment# cost=5 resulted in the lowest cross validation error rate, thus
the best model
bestmodel_SS3_L4=tune.out$best.model
plot(bestmodel_SS3_L4, dat_SS3_LT4)
summary(bestmodel_SS3_L4)



### using the best model function to predict on a test data made of the
remaining observations in the dataset
dat_SS3_Lt4=subset(dat_Lt4,x4.1>5 & x4.1<=7.5 & x4.2<=2.5)
ypred=predict(bestmodel_SS3_L4, dat_SS3_Lt4)
table(predict=ypred, true=dat_SS3_Lt4$y4)

#Comment# Among 13 observations, 4 are misclassified, thus an error rate
of 30.769 %



############################################################( SS4:
)############################################################
set.seed(1)


dat_SS4_LT4=subset(dat_LT4, x4.1>7.5 & x4.1<=10 & x4.2<=2.5)
```

```
#####################################################SVM4_SS4)

tune.out=tune(svm,y4~.,data=dat_SS4_LT4,
kernel="linear",ranges=list(cost=(c(0.001, 0.01, 0.1, 1, 5, 10, 100))))
summary(tune.out)

#Comment# cost 0.001 resulted in the lowest cross validation error rate,
thus the best model
bestmodel_SS4_L4=tune.out$best.model
plot(bestmodel_SS4_L4, dat_SS4_LT4)
summary(bestmodel_SS4_L4)



### using the best model function to predict on a test data made of the
remaining observations in the dataset
dat_SS4_Lt4=subset(dat_Lt4,x4.1>7.5 & x4.1<=10 & x4.2<=2.5)
ypred=predict(bestmodel_SS4_L4, dat_SS4_Lt4)
table(predict=ypred, true=dat_SS4_Lt4$y4)

#Comment# Among 12 observations, 5 are misclassified, thus an error rate
of 41.67 %



#############################################################(
SS5:)#############################################################
set.seed(1)


dat_SS5_LT4=subset(dat_LT4, x4.1<=2.5 & x4.2>2.5 & x4.2<=5)



#####################################################SVM4_SS5

tune.out=tune(svm,y4~.,data=dat_SS5_LT4,
kernel="linear",ranges=list(cost=(c(0.001, 0.01, 0.1, 1, 5, 10, 100))))
summary(tune.out)

#Comment# cost 0.001 resulted in the lowest cross validation error rate,
thus the best model
bestmodel_SS5_L4=tune.out$best.model
plot(bestmodel_SS5_L4, dat_SS5_LT4)
summary(bestmodel_SS5_L4)



### using the best model function to predict on a test data made of the
remaining observations in the dataset
dat_SS5_Lt4=subset(dat_Lt4,x4.1<=2.5 & x4.2>2.5 & x4.2<=5)  # Lt4= Linear
test 4rd sample
ypred=predict(bestmodel_SS5_L4, dat_SS5_Lt4)
table(predict=ypred, true=dat_SS5_Lt4$y4)

#Comment# Among 19 observations, 15 are misclassified, thus an error rate
of 78.947 %
```

```
#############################################################( SS6:
)#############################################################
set.seed(1)


dat_SS6_LT4=subset(dat_LT4, x4.1>2.5 & x4.1<=5 & x4.2>2.5 & x4.2<=5)



#########################################################SVM4_SS6

tune.out=tune(svm,y4~.,data=dat_SS6_LT4,
kernel="linear",ranges=list(cost=(c(0.001, 0.01, 0.1, 1, 5, 10, 100))))
summary(tune.out)

#Comment# cost 0.001 resulted in the lowest cross validation error rate,
thus the best model
bestmodel_SS6_L4=tune.out$best.model
plot(bestmodel_SS6_L4, dat_SS6_LT4)
summary(bestmodel_SS6_L4)




### using the best model function to predict on a test data made of the
remaining observations in the dataset
dat_SS6_Lt4=subset(dat_Lt4,x4.1>2.5 & x4.1<=5 & x4.2>2.5 & x4.2<=5)
ypred=predict(bestmodel_SS6_L4, dat_SS6_Lt4)
table(predict=ypred, true=dat_SS6_Lt4$y4)

#Comment# Among 14 observations, 8 are misclassified, thus an error rate
of 57.14 %

#############################################################( SS7:
)#############################################################
set.seed(1)


dat_SS7_LT4=subset(dat_LT4, x4.1>5 & x4.1<=7.5 & x4.2>2.5 & x4.2<=5)



#########################################################SVM4_SS7

tune.out=tune(svm,y4~.,data=dat_SS7_LT4,
kernel="linear",ranges=list(cost=(c(0.001, 0.01, 0.1, 1, 5, 10, 100))))
summary(tune.out)

#Comment# cost=0.001 resulted in the lowest cross validation error rate,
thus the best model
bestmodel_SS7_L4=tune.out$best.model
plot(bestmodel_SS7_L4, dat_SS7_LT4)
summary(bestmodel_SS7_L4)
```

```
### using the best model function to predict on a test data made of the
remaining observations in the dataset
dat_SS7_Lt4=subset(dat_Lt4,x4.1>5 & x4.1<=7.5 & x4.2>2.5 & x4.2<=5)
ypred=predict(bestmodel_SS7_L4, dat_SS7_Lt4)
table(predict=ypred, true=dat_SS7_Lt4$y4)

#Comment# Among 15 observations, 8 are misclassified, thus an error rate
of 53.33 %


############################################################(
SS8:)############################################################
set.seed(1)


dat_SS8_LT4=subset(dat_LT4, x4.1>7.5 & x4.1<=10 & x4.2>2.5 & x4.2<=5)



#########################################################SVM4_SS8

tune.out=tune(svm,y4~.,data=dat_SS8_LT4,
kernel="linear",ranges=list(cost=(c(0.001, 0.01, 0.1, 1, 5, 10, 100))))
summary(tune.out)

#Comment# cost=0.001 resulted in the lowest cross validation error rate,
thus the best model
bestmodel_SS8_L4=tune.out$best.model
plot(bestmodel_SS8_L4, dat_SS8_LT4)
summary(bestmodel_SS8_L4)




### using the best model function to predict on a test data made of the
remaining observations in the dataset
dat_SS8_Lt4=subset(dat_Lt4,x4.1>7.5 & x4.1<=10 & x4.2>2.5 & x4.2<=5)
ypred=predict(bestmodel_SS8_L4, dat_SS8_Lt4)
table(predict=ypred, true=dat_SS8_Lt4$y4)

#Comment# Among 11 observations, 8 are misclassified, thus an error rate
of 72.73 %


############################################################( SS9:
)############################################################
set.seed(1)


dat_SS9_LT4=subset(dat_LT4, x4.1<=2.5 & x4.2>5 & x4.2<=7.5)



#########################################################SVM4_SS9

tune.out=tune(svm,y4~.,data=dat_SS9_LT4,
kernel="linear",ranges=list(cost=(c(0.001, 0.01, 0.1, 1, 5, 10, 100))))
summary(tune.out)
```

```
#Comment# cost 0.001 resulted in the lowest cross validation error rate,
thus the best model
bestmodel_SS9_L4=tune.out$best.model
plot(bestmodel_SS9_L4, dat_SS9_LT4)
summary(bestmodel_SS9_L4)



### using the best model function to predict on a test data made of the
remaining observations in the dataset
dat_SS9_Lt4=subset(dat_Lt4,x4.1<=2.5 & x4.2>5 & x4.2<=7.5)  # Lt4= Linear
test 4rd sample
ypred=predict(bestmodel_SS9_L4, dat_SS9_Lt4)
table(predict=ypred, true=dat_SS9_Lt4$y4)

#Comment# Among 16 observations, 7 are misclassified, thus an error rate
of 43.75 %



############################################################( SS10:
)############################################################
set.seed(1)


dat_SS10_LT4=subset(dat_LT4, x4.1>2.5 & x4.1<=5 & x4.2>5 & x4.2<=7.5)



#########################################################SVM4_SS10

tune.out=tune(svm,y4~.,data=dat_SS10_LT4,
kernel="linear",ranges=list(cost=(c(0.001, 0.01, 0.1, 1, 5, 10, 100))))
summary(tune.out)

#Comment# cost 0.001 resulted in the lowest cross validation error rate,
thus the best model
bestmodel_SS10_L4=tune.out$best.model
plot(bestmodel_SS10_L4, dat_SS10_LT4)
summary(bestmodel_SS10_L4)



### using the best model function to predict on a test data made of the
remaining observations in the dataset
dat_SS10_Lt4=subset(dat_Lt4,x4.1>2.5 & x4.1<=5 & x4.2>5 & x4.2<=7.5)
ypred=predict(bestmodel_SS10_L4, dat_SS10_Lt4)
table(predict=ypred, true=dat_SS10_Lt4$y4)

#Comment# Among 21 observations, 10 are misclassified, thus an error rate
of 47.6 %



############################################################( SS11:
)############################################################
set.seed(1)
```

```
dat_SS11_LT4=subset(dat_LT4, x4.1>5 & x4.1<=7.5 & x4.2>5 & x4.2<=7.5)



##########################################################SVM4_SS11

tune.out=tune(svm,y4~.,data=dat_SS11_LT4,
kernel="linear",ranges=list(cost=(c(0.001, 0.01, 0.1, 1, 5, 10, 100))))
summary(tune.out)

#Comment# cost=0.001 resulted in the lowest cross validation error rate,
thus the best model
bestmodel_SS11_L4=tune.out$best.model
plot(bestmodel_SS11_L4, dat_SS11_LT4)
summary(bestmodel_SS11_L4)



### using the best model function to predict on a test data made of the
remaining observations in the dataset
dat_SS11_Lt4=subset(dat_Lt4,x4.1>5 & x4.1<=7.5 & x4.2>5 & x4.2<=7.5)
ypred=predict(bestmodel_SS11_L4, dat_SS11_Lt4)
table(predict=ypred, true=dat_SS11_Lt4$y4)

#Comment# Among 12 observations, 8 are misclassified, thus an error rate
of 66.67 %



##############################################################( SS12:
)##############################################################
set.seed(1)


dat_SS12_LT4=subset(dat_LT4, x4.1>7.5 & x4.1<=10 & x4.2>5 & x4.2<=7.5)



##########################################################SVM4_SS12

tune.out=tune(svm,y4~.,data=dat_SS12_LT4,
kernel="linear",ranges=list(cost=(c(0.001, 0.01, 0.1, 1, 5, 10, 100))))
summary(tune.out)

#Comment# cost=1 resulted in the lowest cross validation error rate, thus
the best model
bestmodel_SS12_L4=tune.out$best.model
plot(bestmodel_SS12_L4, dat_SS12_LT4)
summary(bestmodel_SS12_L4)



### using the best model function to predict on a test data made of the
remaining observations in the dataset
dat_SS12_Lt4=subset(dat_Lt4,x4.1>7.5 & x4.1<=10 & x4.2>5 & x4.2<=7.5)
ypred=predict(bestmodel_SS12_L4, dat_SS12_Lt4)
table(predict=ypred, true=dat_SS12_Lt4$y4)
```

```
#Comment# Among 17 observations, 7 are misclassified, thus an error rate
of  41.176 %


############################################################( SS13:
)############################################################
set.seed(1)


dat_SS13_LT4=subset(dat_LT4, x4.1<=2.5 & x4.2>7.5 & x4.2<=10)



#########################################################SVM4_SS13

tune.out=tune(svm,y4~.,data=dat_SS13_LT4,
kernel="linear",ranges=list(cost=(c(0.001, 0.01, 0.1, 1, 5, 10, 100))))
summary(tune.out)

#Comment# cost 0.001 resulted in the lowest cross validation error rate,
thus the best model
bestmodel_SS13_L4=tune.out$best.model
plot(bestmodel_SS13_L4, dat_SS13_LT4)
summary(bestmodel_SS13_L4)



### using the best model function to predict on a test data made of the
remaining observations in the dataset
dat_SS13_Lt4=subset(dat_Lt4,x4.1<=2.5 & x4.2>7.5 & x4.2<=10)  # Lt4=
Linear test 4rd sample
ypred=predict(bestmodel_SS13_L4, dat_SS13_Lt4)
table(predict=ypred, true=dat_SS13_Lt4$y4)

#Comment# Among 13 observations, 3 are misclassified, thus an error rate
of 23.0769 %


############################################################( SS14:
)############################################################
set.seed(1)


dat_SS14_LT4=subset(dat_LT4, x4.1>2.5 & x4.1<=5 & x4.2>7.5 & x4.2<=10)



#########################################################SVM4_SS14

tune.out=tune(svm,y4~.,data=dat_SS14_LT4,
kernel="linear",ranges=list(cost=(c(0.001, 0.01, 0.1, 1, 5, 10, 100))))
summary(tune.out)

#Comment# cost 0.001 resulted in the lowest cross validation error rate,
thus the best model
bestmodel_SS14_L4=tune.out$best.model
plot(bestmodel_SS14_L4, dat_SS14_LT4)
summary(bestmodel_SS14_L4)
```

```
### using the best model function to predict on a test data made of the
remaining observations in the dataset
dat_SS14_Lt4=subset(dat_Lt4,x4.1>2.5 & x4.1<=5 & x4.2>7.5 & x4.2<=10)
ypred=predict(bestmodel_SS14_L4, dat_SS14_Lt4)
table(predict=ypred, true=dat_SS14_Lt4$y4)

#Comment# Among 15 observations, 6 are misclassified, thus an error rate
of 40 %



############################################################( SS15:
)############################################################
set.seed(1)


dat_SS15_LT4=subset(dat_LT4, x4.1>5 & x4.1<=7.5 & x4.2>7.5 & x4.2<=10)



#########################################################SVM4_SS15

tune.out=tune(svm,y4~.,data=dat_SS11_LT4,
kernel="linear",ranges=list(cost=(c(0.001, 0.01, 0.1, 1, 5, 10, 100))))
summary(tune.out)

#Comment# cost=0.001 resulted in the lowest cross validation error rate,
thus the best model
bestmodel_SS15_L4=tune.out$best.model
plot(bestmodel_SS15_L4, dat_SS15_LT4)
summary(bestmodel_SS15_L4)




### using the best model function to predict on a test data made of the
remaining observations in the dataset
dat_SS15_Lt4=subset(dat_Lt4,x4.1>5 & x4.1<=7.5 & x4.2>7.5 & x4.2<=10)
ypred=predict(bestmodel_SS15_L4, dat_SS15_Lt4)
table(predict=ypred, true=dat_SS15_Lt4$y4)

#Comment# Among 22 observations, 10 are misclassified, thus an error rate
of 45.45 %


############################################################(
SS16:)############################################################
set.seed(1)


dat_SS16_LT4=subset(dat_LT4, x4.1>7.5 & x4.1<=10 & x4.2>7.5 & x4.2<=10)



#########################################################SVM4_SS16
```

```
tune.out=tune(svm,y4~.,data=dat_SS16_LT4,
kernel="linear",ranges=list(cost=(c(0.001, 0.01, 0.1, 1, 5, 10, 100))))
summary(tune.out)

#Comment# cost=0.001 resulted in the lowest cross validation error rate,
thus the best model
bestmodel_SS16_L4=tune.out$best.model
plot(bestmodel_SS16_L4, dat_SS16_LT4)
summary(bestmodel_SS16_L4)



### using the best model function to predict on a test data made of the
remaining observations in the dataset
dat_SS16_Lt4=subset(dat_Lt4,x4.1>7.5 & x4.1<=10 & x4.2>7.5 & x4.2<=10)
ypred=predict(bestmodel_SS16_L4, dat_SS16_Lt4)
table(predict=ypred, true=dat_SS16_Lt4$y4)

#Comment# Among 16 observations, 8 are misclassified, thus an error rate
of 50 %


######## Over 16 splits error mean is 50.4 %
#################################
```

FILE 3: ######################### Implementation of kernel Q (Approximation of kernel K)##############################

```
 library(quadprog)
library(Matrix)
gamma=0.5
new.kernel <- function(x1, x2) {
return(x1%*%x2)*exp(-gamma*(x1-x2)%*%(x1-x2))
 }

svm.fit <- function(X, y, FUN=new.kernel, C=NULL) {
 n.samples <<- nrow(X)
 n.features <<- ncol(X)
 # Gram matrix
   K <- matrix(rep(0, n.samples*n.samples), nrow=n.samples)
 for (i in 1:n.samples){
  for (j in 1:n.samples){
   K[i,j] <- FUN(X[i,], X[j,])
  }
K<<-K
 }
 Dmat <- outer(y,y) * K
 Dmat <- as.matrix(nearPD(Dmat)$mat) # convert Dmat to nearest pd matrix
 dvec <- rep(1, n.samples)
 if (!is.null(C)) { # soft-margin
  Amat <- rbind(y, diag(n.samples), -1*diag(n.samples))
  bvec <- c(0, rep(0, n.samples), rep(-C, n.samples))
 } else {           # hard-margin
  Amat <- rbind(y, diag(n.samples))
  bvec <- c(0, rep(0, n.samples))
 }
Bvec<<-bvec
 res <- solve.QP(Dmat,dvec,t(Amat),bvec=bvec, meq=1)
 a = res$solution # Lagrange multipliers
 # Support vectors have non-zero Lagrange multipliers
 # ...
for(i in 1:length(a)){
if(abs(a[i])<10^-10){a[i]=0}
```

```
}
#sum<<-sum
a<<-a
# return(a)
}
#########################

## Creation of data set
set.seed(1)
n4=500

x41= matrix(runif(n4,0,10), ncol=1)

x42= matrix(runif(n4,0,10), ncol=1)


 x4= cbind(x41,x42)
y4=sample(c(-1,1),n4,rep=TRUE)
for (i in 1:n4){
#if (tan(x41[i]+x42[i])/sin(x41[i]^x42[i])>0){y4[i]=1}
if (sin(x41[i]^x42[i])>0){y4[i]=1}
else {y4[i]=-1}
}
y4

train=sample(n4,n4/2)

 ## Extacting observations for training dataset

x_train=x4[train,]
y_train=y4[train]


## Implementing the Kernel Q (new.kernel)

svm.fit(x_train,y_train,FUN=new.kernel, C=0.1)

test=-train
 ## Extacting observations for test dataset
x_test=x4[test,]
y_test=y4[test]


predict=c(0,rep(length(x_test[,1]))) ## Vector that holds predictions for
test dataset

u=c(0,rep(length(x_train[,1])))

##

for (i in 1:length(x_test[,1])){
for(j in 1:length(a)){
u[j]=a[j]*y_train[j]*x_train[j,]%*%x_test[i,]*exp(-gamma*(x_train[j,]-
x_test[i,])%*%(x_train[j,]-x_test[i,])) + Bvec[j]
}
predict[i]=sum(u)
}

## Computing the error rate
```

```
mean((predict/abs(predict)-y_test)*(predict/abs(predict)-y_test)/4)


######### Kernel Q error mean is 45.2% < Kernel K error mean of 42.72 % <
Radial Basis error mean of 46.4% < Linear SVM error mean of 48.8%
##########################################################
```