Support Vector Machines with Convex Combination of Kernels

Farnoosh Rahimi

A Thesis
In
The Department
of
Mathematics and Statistics

Presented in Partial Fulfilment of the Requirements
for the degree of Master of Science (Mathematics) at
Concordia University
Montreal, Quebec, Canada

August , 2018

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: Farnoosh Rahimi

Entitled: Support Vector Machines with Convex Combination of Kernels

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

Master of Science (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:

_____ Chair
Dr. Arusharka Sen

_____ Examiner
Dr. Debaraj Sen

_____ Examiner
Dr. Lisa Kakinami

_____ Thesis Supervisor
Dr. Arusharka Sen

Approved by _____
Dr. Arusharka Sen, Graduate Program Director

_____
Dr. Andre G Roy, Dean, Faculty of Arts and Science

Date 28-August-2018

Abstract

Support Vector Machines with Convex Combination of Kernels

Farnoosh Rahimi

Support Vector Machine (SVMs) are renowned for their excellent performance in solving data-mining problems such as classification, regression and feature selection (Blei, Ng, & Jordan, 2003). In the field of statistical classification, SVMs classify data points into different groups based on finding the hyperplane that maximizes the margin between the two classes. SVMs can also use kernel functions to map the data into a higher dimensional space in case a hyperplane cannot be used to do the separation linearly. Using specific kernels allows us to model a particular feature space, and a suitable kernel can improve the SVMs' performance to classify data more accurately. We present a method to combine existing kernels in order to produce a new kernel which improves the accuracy of the classification and reduce the process time. We will discuss the theoretical and computational issues on SVMs. We are going to implement our method on a simulated data-set to see how it works, and then we will apply it to some large real-world data-sets.

Acknowledgements

I thank all who in one way or another contributed in the completion of this thesis. First, I would like to thank my research supervisor, Prof. Arusharka Sen, for the invaluable guidance, advice, and encouragement he has provided throughout this research, and my time as his student. Also, I must express my gratitude to Mahdi, my husband, for his love, understanding, and continuing support to complete this research work.

Furthermore, I am so grateful to my parents for their love, caring and sacrifices for educating and preparing me for my future.

Finally, I would like to thank the faculty members and my friends who helped me through my study at Concordia University.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Support Vector Machines are supervised learning methods that analyze data for classification and regression. SVM was first suggested in the 1990s by Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik, based on Statistical Learning Theory and have become well-known algorithm. SVMs become very powerful in areas such as classification of images, face detection, text and hypertext categorization, hand writing recognition, bioinformatics, and so forth (Drucker, Wu, & Vapnik, 1999).

The primitive SVM was suggested in 1963 by Vapnik, who introduced a way to construct a hyperplane that can split the training data-set according to their class labels in a n-dimensional input space. The SVM supervised algorithm trains a model based on maximum-margin hyperplane; this hyperplane has the largest

distance from the nearest training data points of either class. These points are called support vectors which directly determine the optimal classifier.

Later, in 1992, non-linear classification in SVM was proposed by Boser, Guyon, and Vapnik on the basis of kernel functions. This algorithm is very similar to the original one, except that every dot product is replaced by a non-linear kernel function which allows the model to fit the maximum-margin classifier in some feature space.

In this paper we concentrate on SVMs for two-class classification; our classes are $y = +1$ and $y = -1$. Support vector classification algorithm always looks for optimal decision surfaces. We construct the separation surface based on Maximal Margin Classifier for linear separable cases, Soft Margin Classifier for the case such that hyperplane can mostly separate the classes, and Kernel Functions for non-linear separable cases.

1.2 The Hard Margin Classifier

Consider the case which can be correctly separated by a hyperplane, then among all possible infinite number of hyperplanes this can separate two classes, we are looking for one which has the farthest distance from the training data points of each class. The nearest data points to the specific hyperplane are called Support Vectors and the maximum distance between them and the hyperplane is the maximal margin (M).

Suppose that we have a set of training data $D = \{x_i, y_i\}\,,\ i = 1, 2, .., m$, where x_i is a m- dimensional real vector and y_i denotes the class labels corresponding to x_i. y_i is either $+1, -1$. A decision hyperplane can be defined as $D(x) = w^T x + b$, where m-dimensional normal vector w is perpendicular to the decision hyperplane and b is the intercept of the hyperplane (bias). Binary SVMs are classifiers which categorize training data into two classes.

Here is the SVM classification function:

$$F(x) = w^T x + b \tag{1.1}$$

The decision rule is based on the sign of this function as follows:

$$\begin{cases} w^T x_i + b > 0 & if \quad y_i = 1 \\ w^T x_i + b < 0 & if \quad y_i = -1 \end{cases} \tag{1.2}$$

Or these condition can be written as:

$$y_i(w^T x_i + b) > 0 \tag{1.3}$$

Since our training data are supposed to be completely separable, none of the points are on the $w^t x_i + b = 0$, so instead of inequality (1.3) we can always write $w^t x_i + b > a$, or $w^t x_i + b < -a$, or equivalently, $w^t x_i + b > 1$, or $w^t x_i + b < -1$ for $i = 1, 2, .., m$.

In order to define optimal separating hyperplane, we need to maximize the distance to the closest point from either classes.

The linear classifier is $D(x_i) = y_i(w^T x_i + b) \geq 1$. The hyperplane with the maximum margin is our ideal optimum hyperplane which is the solution to the optimization problem.

The Euclidean distance from a training datum to the decision hyperplane can be defined as follows:

- Let d_n is the distance from an arbitrary point x, in class with label $+1$, to the decision hyperplane which is parallel to w, so $d_n = kw$
- x_0 is the projection of point x onto the separating hyperplane, $x_0 = x - d_n$, and it is on the decision hyperplane which implies:
  $w^T x_0 + b = 0 \ \ or \ \ w^T(x - d_n) + b = 0 \ \ or \ \ w^T(x - kw) + b = 0$ which defines k as $\frac{w^T x + b}{w^T w}$.
- On the other hand; length of d_n is
  $||d_n|| = \sqrt{d_n^T d_n} = \sqrt{k^2 w^T w} = \sqrt{(\frac{w^T x + b}{w^T w})^2 w^T w} = \frac{|w^T x + b|}{||w||} = \frac{|D(x)|}{||w||}$

denote $y_i \in \{+1, -1\}$ is the class of x_i, then the distance becomes:

$$||d_n|| = \frac{y_i D(x_i)}{||W||}$$

we can say that all the data must satisfy:

$$\frac{y_i D(x_i)}{||W||} \geq M \tag{1.4}$$

for $i = 1, 2, ..., m$, and M is the margin.

Thus, the margin becomes:

$$\frac{1}{||W||} \tag{1.5}$$

To maximize M we need to minimize the Euclidean norm of w, Thus, the SVM becomes a constrained optimization problem as follows (Abe, 2010):

$$\begin{cases} minimize\ \ Q(w) = \frac{1}{2}||w||^2 \qquad w.r.t \quad w, b \\ Subject\ \ to: \quad y_i(w^T x_i + b) \geq 1 \qquad for\ \ every\ \ (x_i, y_i) \end{cases} \tag{1.6}$$

1.2.1 Solving the Optimization Problem

As we already mentioned, the only data points which satisfy the strict equality are support vectors.

In the constrained problem (1.6); $Q(w)$ is a convex function of w and the constrains are linear in w.

To solve this optimization problem we use the method of Lagrange multipliers (Yu & Kim, 2012), as it satisfies the Karush- Kuhn-Tucker (KKT) conditions (Abe, 2005):

$$\begin{cases} Q(w,b,\alpha) = \frac{1}{2}w^T w - \sum_{i=1}^{m} \alpha_i (y_i(w^T x_i + b) - 1) \\ 1. \quad \frac{d}{dw} Q(w,b,\alpha) = 0 \\ 2. \quad \frac{d}{db} Q(w,b,\alpha) = 0 \\ 3. \quad \alpha_i (y_i(w^T x_i + b) - 1) = 0 \qquad i = 1,2,..,m \\ 4. \quad \alpha_i \geq 0 \end{cases} \tag{1.7}$$

from (1.7)- third condition the following must be satisfied:

$$\begin{cases} \alpha_i = 0, or \\ \alpha_i \neq 0, and \quad y_i(w^T x_i + b) = 1) \end{cases} \tag{1.8}$$

the data points such that $\alpha_i \neq 0$ are called support vectors.

from (1.7)- first condition and second condition we can respectively reach to:

$$w = \sum_{i+1}^{m} \alpha_i y_i x_i, \qquad \sum_{i=1}^{m} \alpha_i y_i = 0 \tag{1.9}$$

Thus, we can replace (1.9) in $Q(w,b,\alpha)$, and we obtain the following dual problem:

$$\begin{cases} Q(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j x_i^T x_j = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} (\sum_{i=1}^{m} \alpha_i y_i x_i)^T (\sum_{i=1}^{m} \alpha_i y_i x_i) \geq 0 \\ Subject\ to: \quad \sum_{i=1}^{m} \alpha_i y_i = 0 \qquad \alpha_i \geq 0 \qquad for\ i = 1,2,..,m \end{cases} \tag{1.10}$$

As we mentioned earlier, the primal optimization problem deals with a convex

function and linear constraints, however, it is possible to construct a dual problem such that it has the same optimal value as the primal one.

Maximizing (1.10) under its constrain is a concave quadratic programming problem, so if a solution exists ($i.e$, the classification problem is linearly separable), the global optimal solution α_i exists (Abe, 2005).

The constraint of the $Q(\alpha)$ shows that the data that are associated with non-negative α_i, are support vectors for any classes. In other words, the datum such that its corresponding α_i is zero will not affect the optimum Weight vector w due to (1.9).

Considering our Decision Function:

$$D(x) = \sum_{i \in supportvectors} \alpha_i y_i x_i^T x + b \tag{1.11}$$

if we can compute the non-negative α_i^* and the corresponding support vectors, we can find b^* as follows:

$$b^* = 1 - w^* x_i \qquad\qquad where\ x_i\ is\ a\ support\ vector.$$

Then datum x will classify into:

$$\begin{cases} Class\ \ with\ \ label + 1: & if\ \ D(x) > 0 \\ Class\ \ with\ \ label - 1: & if\ \ D(x) < 0 \\ On\ \ the\ \ boundary: & if\ \ D(x) = 0 \end{cases} \tag{1.12}$$

1.3 The Soft Margin SVM

The discussion so far was surrounding linearly separable training data, however, when the data are not linearly separable there is no solution to the optimization problem $Q(w, b, \alpha)$ with $M > 0$ and the hard margin SVM is unsolvable (Abe, 2010). To deal with such cases, Soft Margin SVM is applicable to an inseparable cases while still maximizing the margin. In this case, a classifier could be used to mostly separate the cases using Soft Margin Classifier.

The method introduces the non-negative slack variables $\xi_i \geq 0$ such that feasible solutions always exist; it means that we allow some observations to violate the margin or even hyperplane in order to get a better result in classifying the remaining observation.

Following is the optimization problem for the soft margin:

$$\begin{cases} Minimize: & Q(w, b, \xi) = \frac{1}{2}||w||^2 + C\sum_{i=1}^{m} \xi_i \\ Subject\ \ to: & y_i(w^T x_i + b) \geq 1 - \xi_i \qquad i = 1, 2, .., m,\ \ \xi = (\xi_1, \xi_2, .., \xi_m)^T \end{cases} \tag{1.13}$$

Where C is non-negative tuning parameter which determines the trade-off between the margin size and the amount of error in classification, and $\xi_i's$ are slack variables that let training data to be on a wrong side of margin or decision boundary.

According to the constraint of (1.13) by maximizing the margin, the amount of mis-classification error will be minimized (Abe, 2005).

1.3.1 Solving the Optimization Problem

Similar to the case of hard-margin SVM, the primal form of optimization problem can be transformed to the following dual form by introducing the non-negative Lagrange multipliers α_i and β_i:

$$\begin{cases} Q(w,b,\xi,\alpha,\beta) = \frac{1}{2}||w||^2 + C\sum_{i=1}^{m}\xi_i - \sum_{i=1}^{m}\alpha_i(y_i(w^T x_i + b) - 1 + \xi_i) - \sum_{i=1}^{m}\beta_i\xi_i \\ \alpha = (\alpha_1, .., \alpha_m)^T, \qquad \beta = (\beta_1, .., \beta_m)^T \end{cases} \tag{1.14}$$

It satisfies the KKT conditions: (Abe, 2005):

$$\begin{cases} 1. \quad \frac{dQ(w,b,\xi,\alpha,\beta)}{dw} = 0, \\ 2. \quad \frac{dQ(w,b,\xi,\alpha,\beta)}{db} = 0, \\ 3. \quad \frac{dQ(w,b,\xi,\alpha,\beta)}{d\xi} = 0. \\ 4. \quad \alpha_i(y_i(w^T x_i + b) - 1 + \xi_i) = 0 \qquad for\ \ i = 1, .., m, \\ 5. \quad \beta_i\xi_i = 0 \qquad for\ \ i = 1, .., m, \\ 6. \quad \alpha_i \geq 0, \quad \beta_i \geq 0, \quad \xi_i \geq 0 \qquad for\ \ i = 1, .., m, \end{cases} \tag{1.15}$$

By reducing (1.15)1-3 conditions to the following, respectively:

$$w = \sum_{i=1}^{m} \alpha_i y_i x_i, \qquad \sum_{i=1}^{m} \alpha_i y_i = 0, \qquad \alpha_i + \beta_i = C, \quad for \quad i = 1, 2, .., m. \tag{1.16}$$

The dual form becomes;

$$\begin{cases} Maximize : Q(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j x_i^T x_j \\ Subject\ \ to : \quad \sum_{i=1}^{m} y_i \alpha_i = 0 \qquad C \geq \alpha_i \geq 0 \quad for\ \ i = 1, , 2, .., m. \end{cases} \tag{1.17}$$

from (1.15) and as it was shown in Figure 1.1, we have the following cases for α_i:

- $\alpha_i = 0$. Then $\xi_i = 0$, thus datum x_i is correctly classified.
- $\alpha_i = C$. Then $y_i(w^T x_i + b) - 1 + \xi_i = 0$ and $\xi_i \geq 0$, thus datum x_i is support vector. If $0 \leq \xi_i \leq 1$ it's correctly classified, and if $\xi_i \geq 1$ it's miss-classified.
- $0 < \alpha_i < C$. Then $y_i(w^T x_i + b) - 1 + \xi_i = 0$ and $\xi_i = 0$, so x_i is support vector.

The decision function we had for Hard-Margin SVM is the same as the one we have for Soft-Margin SVM as follows:

$$D(x) = \sum_{i \in SupportVectors} \alpha_i y_i x_i^T x + b \tag{1.18}$$

The datum x will be classified into:

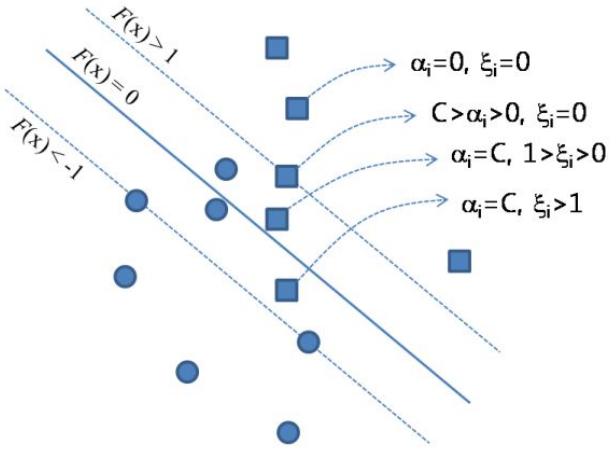


FIGURE 1.1: Graphical relationships among $\alpha_i, \xi_i, and\ C$
(Yu & Kim, 2012)

$$\begin{cases} Class\ with\ label + 1: & if\ \ D(x) > 0 \\ Class\ with\ label - 1: & if\ \ D(x) < 0 \\ On\ the\ boundary: & if\ \ D(x) = 0 \end{cases} \tag{1.19}$$

1.4 Kernel Methods and SVMs

The support vector classifier may not be always the best choice for the classification, in most cases we confront a case with data non-linearly separable. Thus, to enhance linear separability, Kernels are used to non-linearly map the original input space into a high dimensional space called the feature space (Abe, 2005). Then, SVMs are used to find the hyperplane of maximal margin in the new feature space.

the dot product of two m-dimensional vectors a,b is defined by:

$$< a, b >= \sum_{i=1}^{r} a_i b_i$$

Thus, the dot product of two observations x_i ,x_i' is given by:

$$< x_i, x_i' >= \sum_{j=1}^{p} x_{ij} x_{ij}' \tag{1.20}$$

Considering $g(x) = (g(x_1), g(x_2), ..., g(x_l))^T$ as a non-linearly function that transfers m-dimensional input data into the l-dimensional feature space, we can define the linear decision function in the feature space as follows:

$$D(x) = w^T g(x) + b \tag{1.21}$$

where w is an l-dim vector, and b is bias term.

Computing the inner product in high-dimension feature space will be quite complex, thus using the Kernel approaches will avoid this issue. Kernel functions can be used as a replacement of an inner product in feature space with a kernel function H as follows:

$$H(x, x') =< g(x), g(x') > \tag{1.22}$$

According to Mercer's theorem (Abe, 2005), a kernel function H is a valid kernel if it satisfies following conditions;

for all natural number m and real number h_i:

$$\sum_{i,j=1}^{m} h_i h_j H(x_i, x_j) \geq 0$$

Satisfying Mercer's condition means the kernel matrix is positive definite, which means the optimization problem is a concave quadratic programming problem and has an unique global optimum solution (Cristianini & Shawe-Taylor, 2000).

The advantage of using Kernel Trick is to create a non-linear decision function without using an explicit mapping to an othere space (Abe, 2005).
Using the Kernel function, the corresponding Lagrange dual problem in the feature space is defined as follows:

$$\begin{cases} maximize: & Q(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j H(x_i, x_j) \\ Subject\ to: & \sum_{i=1}^{m} y_i \alpha_i = 0, \qquad 0 \leq \alpha_i \leq C \quad for\ i = 1, .., m. \end{cases} \tag{1.23}$$

where $H(x_i, x_j)$ is positive definite kernel which satisfies Mercer's theorem and is used instead of $g(x)$ to map input training data to feature space.

The decision function is defined by (Abe, 2005):

$$D(x) = \sum_{i \in SupportVectors} \alpha_i y_i H(x_i, x) + b$$

where

$$b = y_j - \sum_{i \in supportvectors} \alpha_i y_i H(x_i, x_j)$$

The datum x will classify into:

$$\begin{cases} Class\ \ with\ \ label + 1: & if\ \ D(x) > 0 \\ Class\ \ with\ \ label - 1: & if\ \ D(x) < 0 \\ On\ \ the\ \ boundary: & if\ \ D(x) = 0 \end{cases} \tag{1.24}$$

1.4.1 Specific Kernels

The followings are popular used kernel functions in Support Vector Machines (Abe, 2005):

Linear Kernels

In the case that the training data can be linearly separable in the input space and there is no necessity to map the input space to feature space, we use Linear Kernels:

$$H(x, x') = x^T x' \tag{1.25}$$

Radial Basis Function Kernels(RBF)

Another popular kernels is the RBF Kernels which is defined as:

$$H(x, x') = exp(-\gamma||x - x'||^2) \qquad where\ \ \gamma\ \ is\ \ positive\ \ constant. \tag{1.26}$$

Polynomial Kernels

The Polynomial Kernel with degree d is as follows:

$$H(x, x') = (x^T x' + 1)^d \qquad where\ d\ is\ a\ natural\ number. \tag{1.27}$$

1.4.2 Visualization of Kernels in bi-variate sample

For illustration of the procedure of training a non-linear SVM function, we are going to generate a sample from a bi-variate normal distribution to visualize them in two-dimensional space.

The steps of simulating our data from bi-variate normal distribution are as follows:

In order to compute the correlated x, y, we need to generate z_1, z_2 firstly.

Let z_1, z_2 be two independent random variables which have standard normal distribution with mean=0 and variance=1.

Assume we have the arbitrary parameters of $\mu_x, \sigma_x, \mu_y, \sigma_y,\ and\ r$ related to x, y respectively.

we are given:

$$\begin{cases} x = \sigma_x[\sqrt{1-r^2}z_1 + rz_2] + \mu_x, \\ \\ y = \sigma_y z_2 + \mu_y \end{cases} \tag{1.28}$$

we can prove that $x \sim N(\mu_x, \sigma_x^2)$, and $y \sim N(\mu_y, \sigma_y^2)$ as follows:

$$x = \sigma_x[\sqrt{1-r^2}N(0,1) + rN(0,1)] + \mu_x$$

$$= [N(0, (1 - r^2)\sigma_x^2) + N(0, r^2\sigma_x^2) + \mu_x$$

$$= [N(0, \sigma_x^2)] + \mu_x = N(0, \sigma_x^2)$$

and $y = \sigma_y N(0, 1) + \mu_y = N(\mu_y, \sigma_y^2)$.

Consequently, to generate bi-variate Normal random variables with $x \sim N(\mu_x, \sigma_x^2)$, $y \sim N(\mu_y, \sigma_y^2)$, with correlation r, we need to generate two uncorrelated, Standard Normal variables z_1, z_2 and use (1.28).

Here we use software R to simulate x, y. We put the following arbitrary values for $\mu_x, \mu_y, \sigma_x, \sigma_y, r$:

$$\mu_x = 1, \sigma_x = 2$$
$$\mu_y = 1, \sigma_y = 4, r = -0.6.$$

As we can see in the R-Codes in Appendix A.1 , we randomly pick 200 random numbers z_1, z_2 to generate bi-variate observations (x_i, y_i) for $i = 1, 2, .., 200$.
We also generate a label vector y_s such that it will randomly assign each observation to each of the classes $+1, -1$ based on the random criteria we put into R-Codes. Finally, we combine the training data points with vector y_s to produce our final data points to fit our corresponding three common Kernel functions we already introduced.

Figure 1.2 is the scatter plot of generated data-set, where Class-1(response variable= 0) has the solid circle symbol •, and Class-2 (response variable= 1) has the triangle symbol ▲.

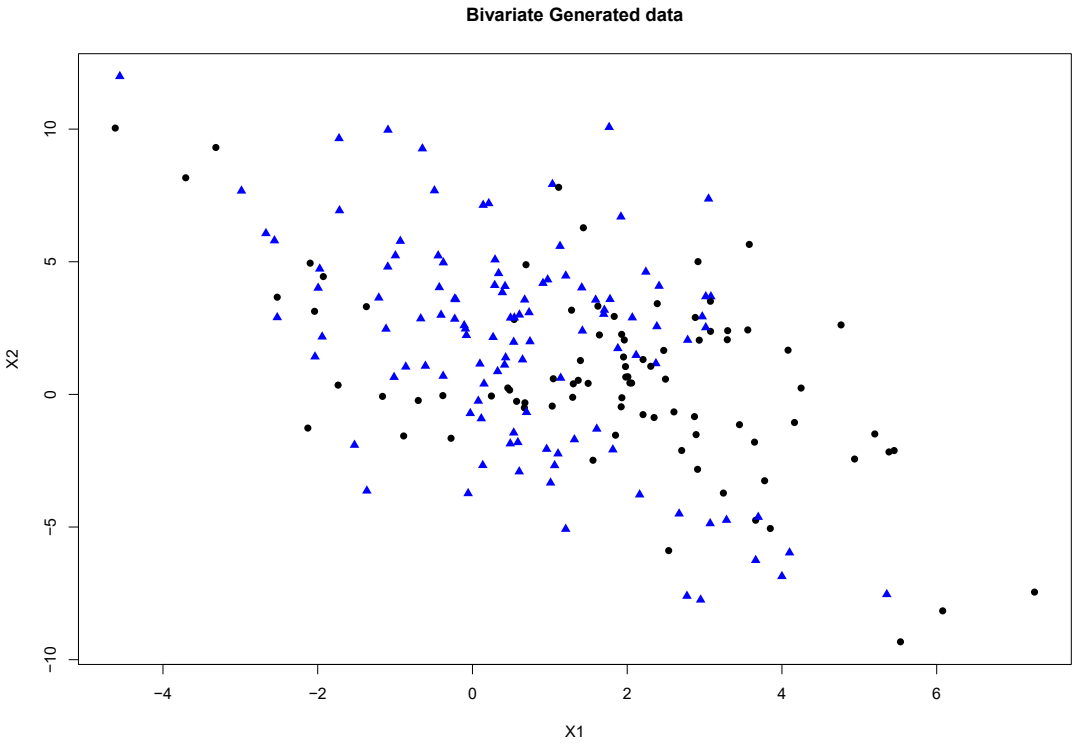


FIGURE 1.2: Scatter Plot, Generate Bi-variate data-set

The frequency table shows the percentage contribution of each class:

	Class1- Circle	Class2- Triangle
Frequency	87	113
Percentage	43.5	56.5

TABLE 1.1: The frequency table of Generated Bi-variate data-set

We split the data-set in four, we take three parts of it for training our model, and the last part for evaluating the model. We execute each of the trained SVMs models on our unseen test-set.

The plots and results are as follows:

	CER	FPR	FNR	ACR	AUC	# SV
Linear Kernel	0.3	0.27	0.358	0.7	0.755	128
RBF Kernel	0.2	0.162	0.263	0.8	0.856	107
Polynomial Kernel	0.34	0.35	0	0.66	0.703	139
Logistic Regression	0.26	0.235	0.312	0.74	0.76	NA

TABLE 1.2: Generated Bi-variate data-set, basic kernels Average-Results

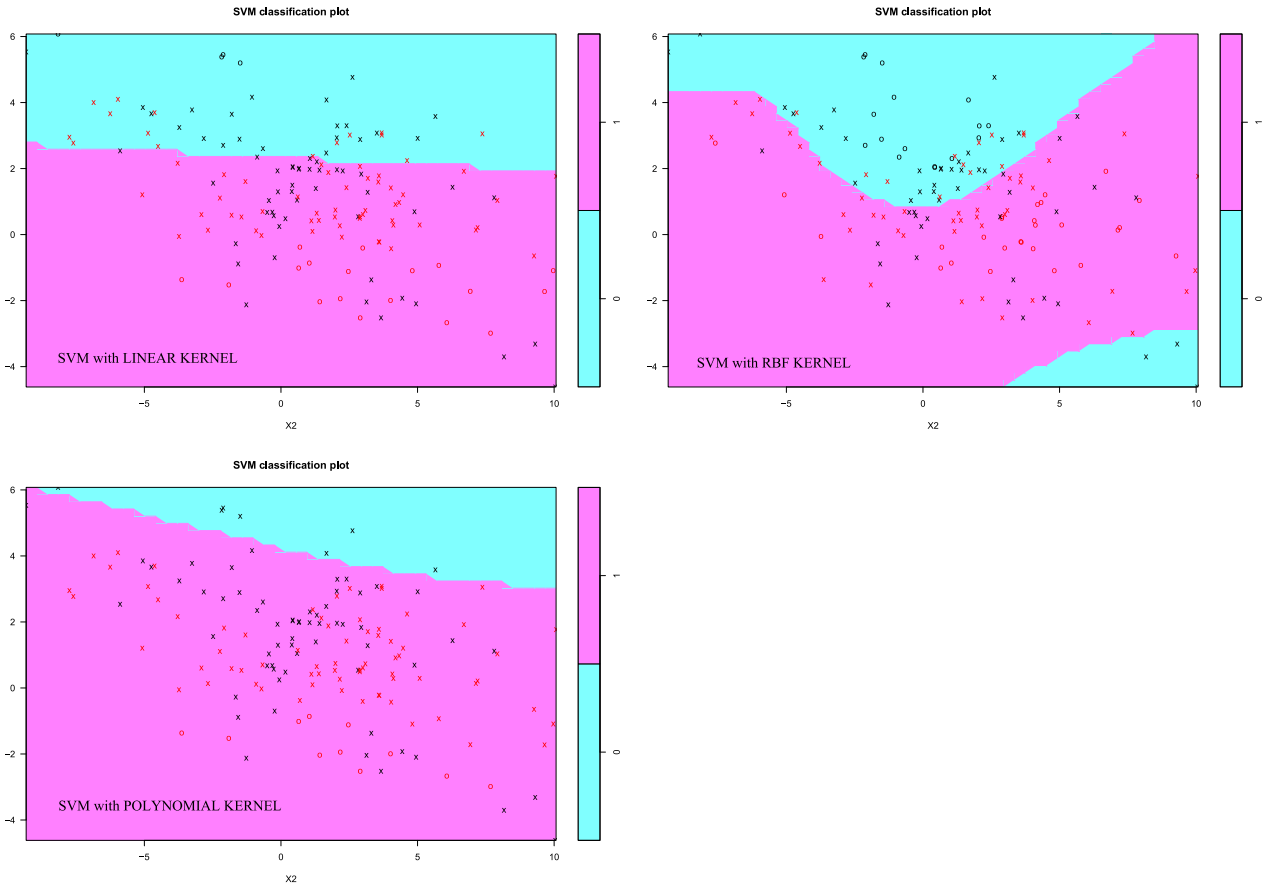


FIGURE 1.3: SVM with basic kernels, Generated Bi-variate data-set

Based on Table 1.2, we can conclude that the SVM with RBF kernel has the best performance with 80% accuracy rate in classification, however, the SVM with Polynomial kernel has the worst performance with 66% accuracy rate and the largest number of support vectors.

We also take into account the Logistic Regression classifier as our reference classification method.

The R codes implementation we already used, provided the training function with default values for our SVMs' parameters, such as $C_{Linear} = 1, C_{RBF} = 1, \gamma = \frac{1}{dim} = 0.5$, $d = 3$. On the other hand, our goal is to optimize the performance of each model by tuning their parameters.

To reach this goal, we are going to utilize a grid search over the specified parameter ranges. Results are given in Table 1.3:

	CER	FPR	FNR	ACR	AUC	#SV
Linear Kernel(C=10)	0.299	0.27	0.35	0.71	0.759	127
RBF Kernel(gamma=0.1, C=100)	0.2	0.16	0.26	0.8	0.864	196
Polynomial Kernel(d=2)	0.38	0.35	0.5	0.62	0.65	124
Logistic Regression	0.26	0.235	0.312	0.74	0.76	NA

TABLE 1.3: Generated Bi-variate data-set, TUNED-basic kernels Results

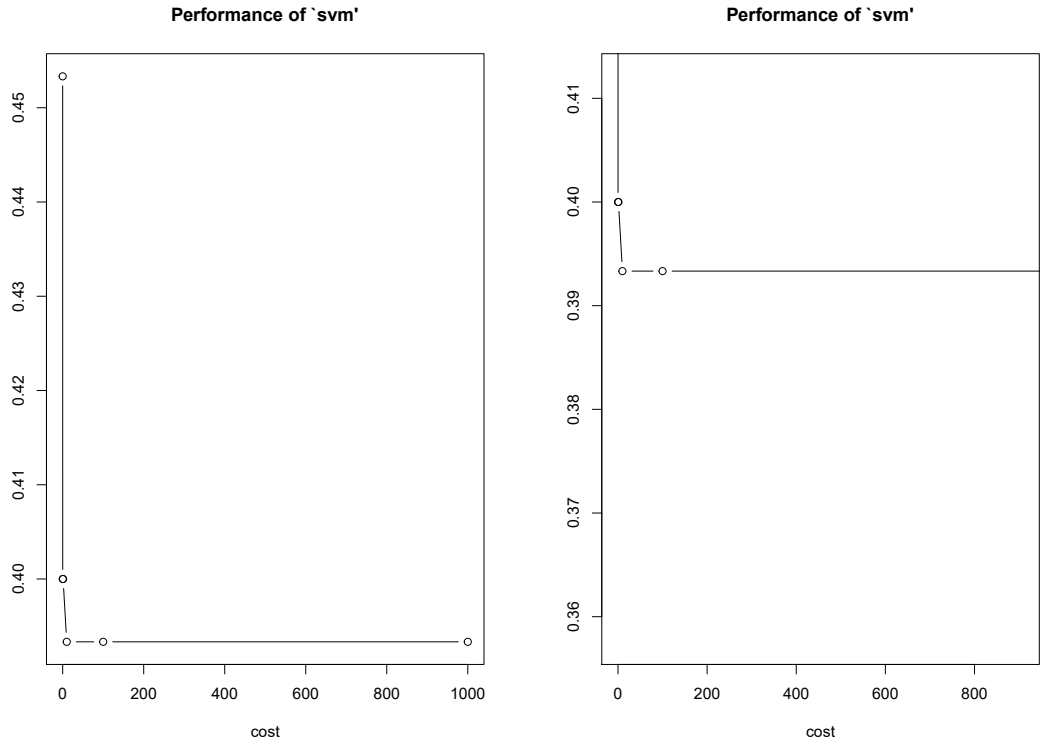


FIGURE 1.4: Tuning the parameter Cost, Linear Kernel, Generated Bi-variate data-set

Based on this experiment, we can say that in the SVM model with Linear kernel, parameter $C = 10$ yields the best accuracy rate of 71%.

Similarly, for the SVM model with RBF kernel, the best model occurs at $C = 100, \gamma = 0.1$ which increases the accuracy rate by 1%. On the contrary, for the model with Polynomial kernel, the model with $d = 2$, yields a miss-classification error of 39.3%, and declines the accuracy rate by 6%. Thus, Tuned RBF kernel has the best performance based on classification level.

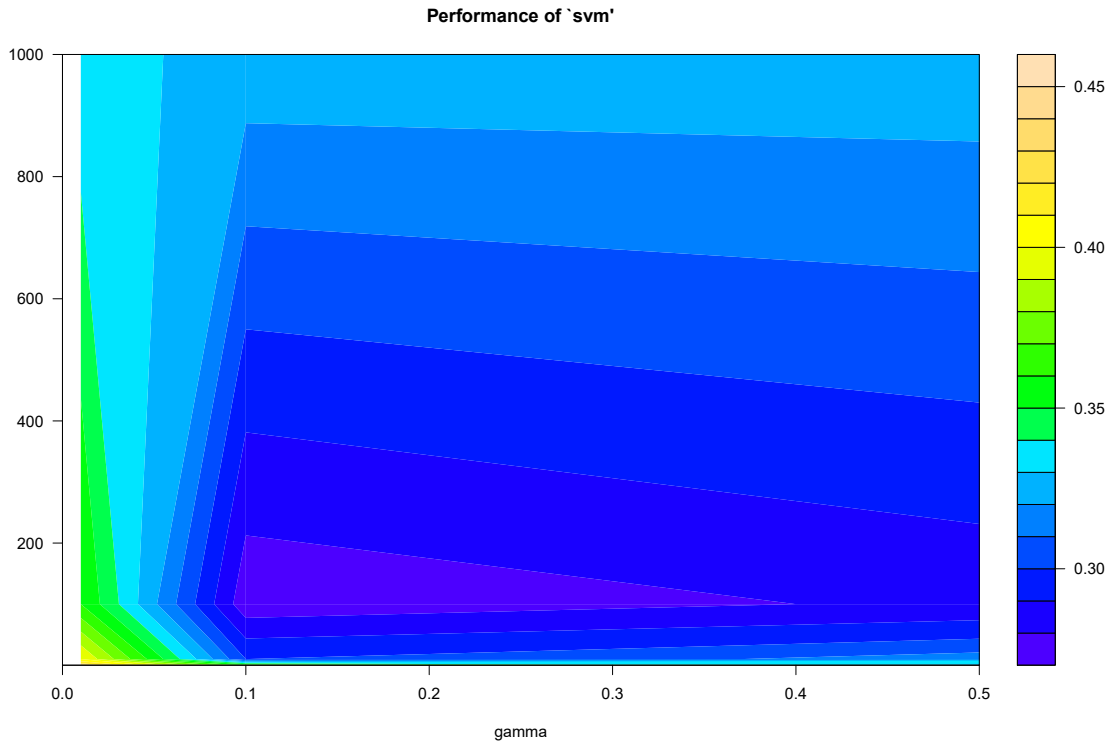


FIGURE 1.5: Tuning the parameter Cost,γ, RBF Kernel, Generated Bi-variate data-set

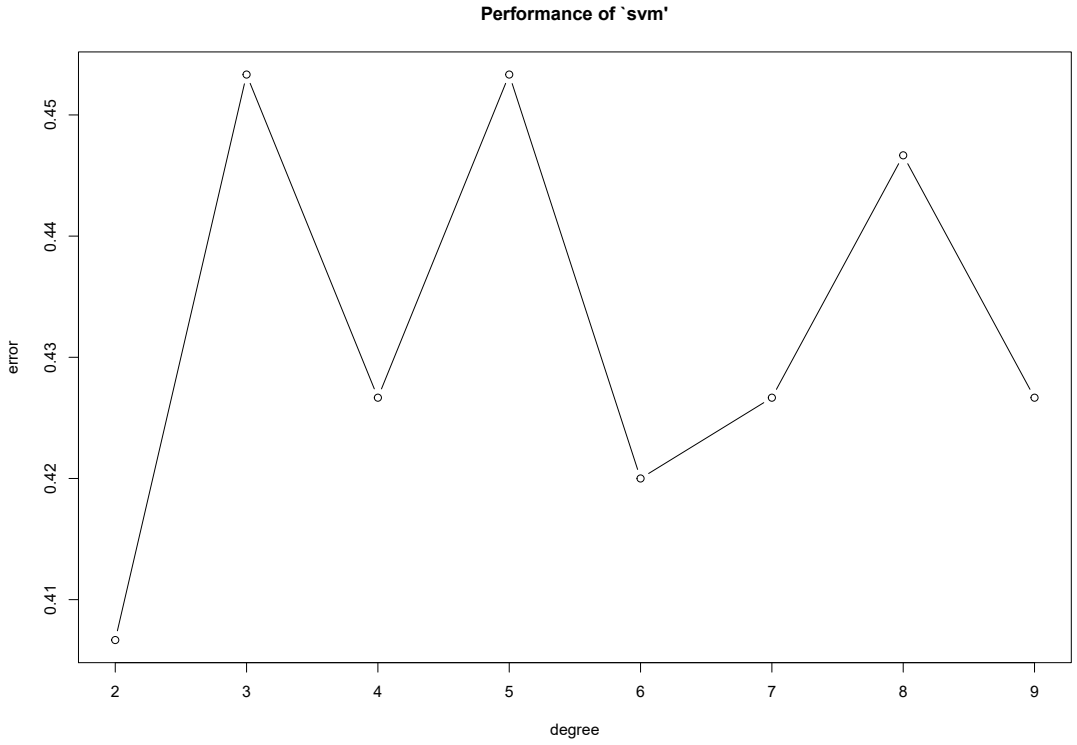


FIGURE 1.6: Tuning the parameter degree, Polynomial Kernel, Generated Bi-variate data-set

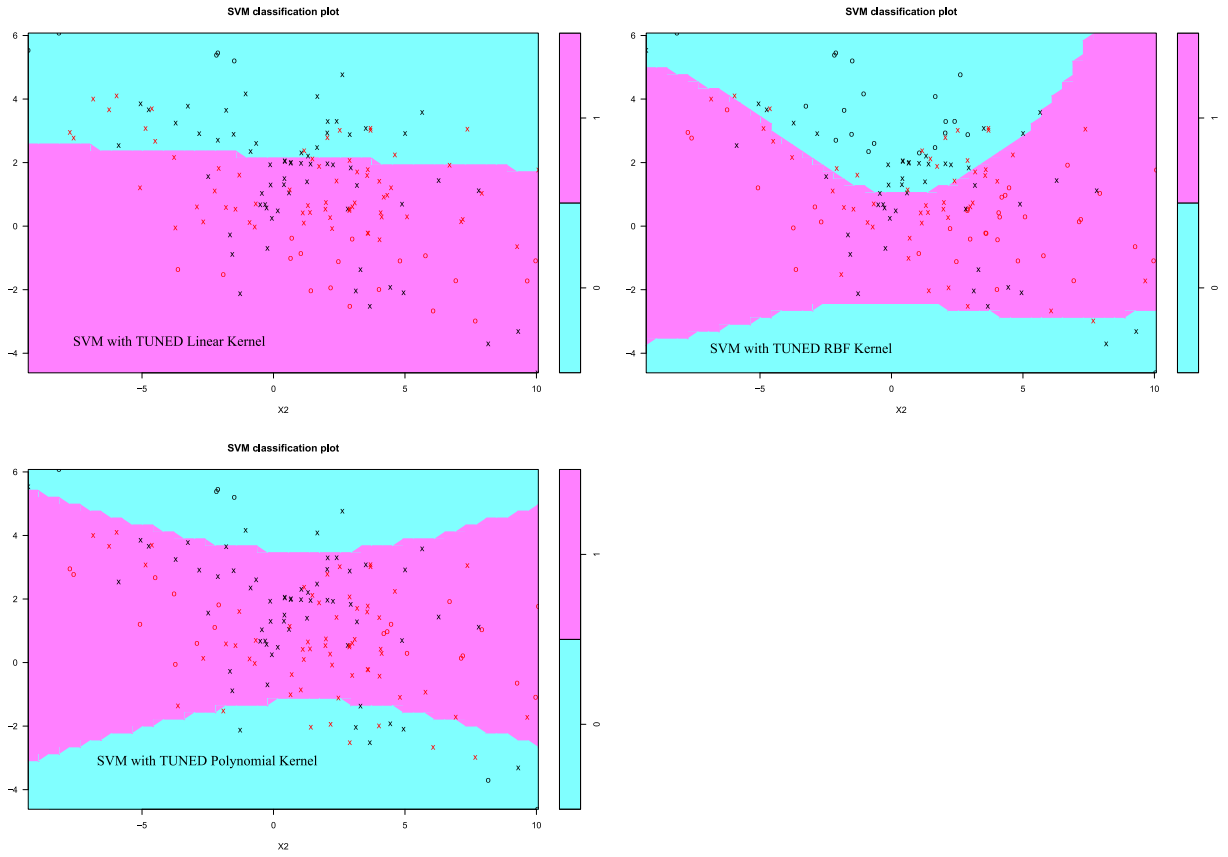


FIGURE 1.7: SVM with TUNED-basic kernels, Generate Bi-variate data-set

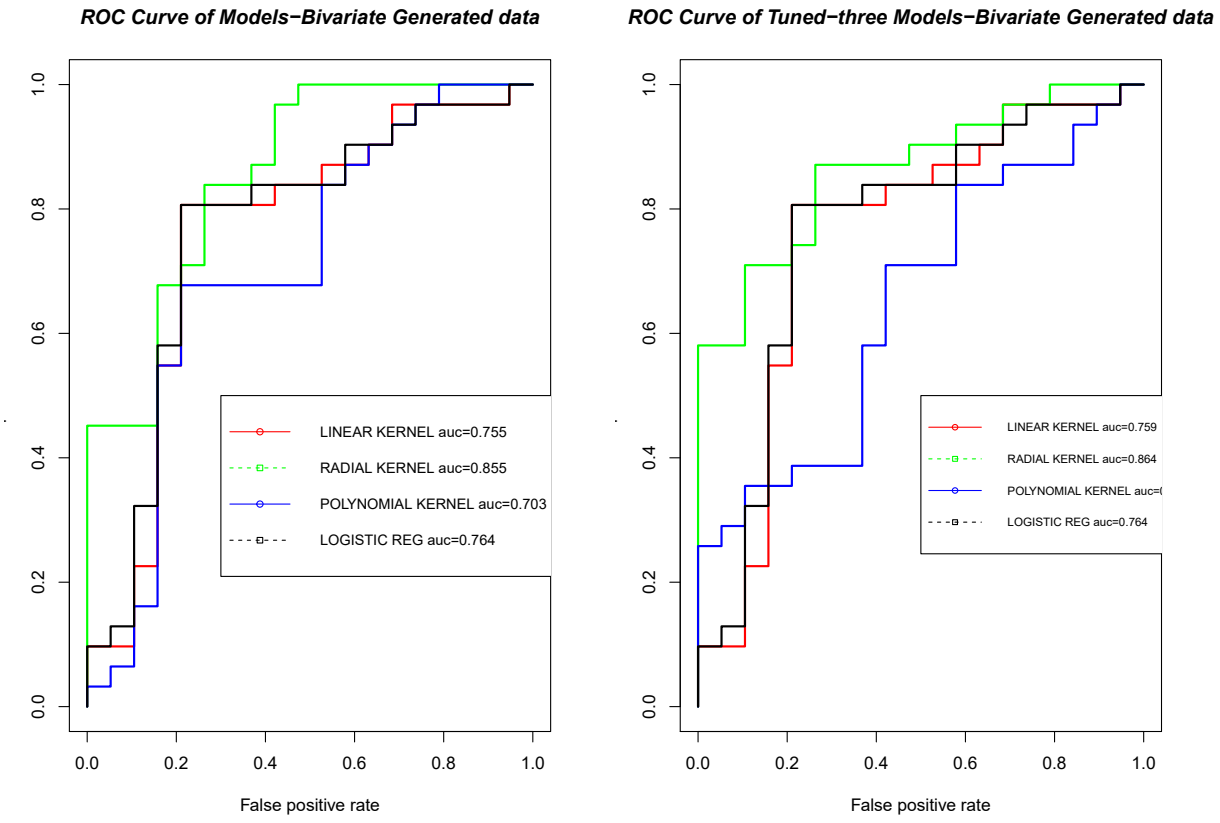


FIGURE 1.8: ROC curve, Basic Kernels vs Logistic Reg, Tuned Basic Kernels vs Logistic Reg, Generated Bi-variate data-set

1.5 Support Vector Regression

SVR works as a regression methods based on all the main principles that characterize the SVM classification except a few minor differences (Smola & Schölkopf, 2004). SVR is a non-parametric technique such that the output model does not depend on the dimensionality of the input space (Drucker, Burges, Kaufman, Smola, & Vapnik, 1997), and it only depends on the kernel functions.

In SVR we don't consider the errors, as long as they are less than certain value of ξ and will not accept any deviation larger than this (Smola & Schölkopf, 2004). The idea of the SVR is to determine a function that can approximate the future value accurately (Wu, Ho, & Lee, 2004).

Suppose we have a training set (x_i, y_i) for $i = 1, 2, .., m$ are continuous variables. The SVR estimating function is as follows;

$$f(x) =< w, x > +b \qquad\qquad w \in x,\ \ b\ \ is\ \ bias\ \ term. \tag{1.29}$$

In ξ-SVregression, our aim is to find the $f(x)$ based on the value w, b such that it can correctly predict the target values of response variable and also accept a small error in fitting the training data-set based on ξ (Vastrad et al., 2013).

Under Least Square Regression, the ξ-insensitive loss function proposed by Vapnik is as follows (Basak, Pal, & Patranabis, 2007):

$$L_\xi(y, f(x)) = \begin{cases} 0 & if\ \ |y - f(x)| \leq \xi \\ |y - f(x)| - \xi & otherwise \end{cases} \tag{1.30}$$

Where L_ξ is called Linear ξ-insensitive loss and $\xi \geq 0$.

SVM regression performs linear regression in high dimensional feature space and tries to minimize $||w||^2$.

Introducing the non-negative slack variables ξ_i, ξ_i^* for $i = 1, 2, 3, .., m$ will help us to measure the deviation of training samples outside of ξ-insensitive zone. Now we can make the optimization problem as follows: (Basak et al., 2007)

$$min_{w,b,\xi_i,\xi_i^*} \quad \frac{1}{2}||w||^2 + C\sum_{i=1}^{m}(\xi_i + \xi_i^*) \tag{1.31}$$

$$\text{Subject to: } \begin{cases} f(x_i) - y_i \leq \xi + \xi_i \\ y_i - f(x_i) \leq \xi + \xi_i^* \\ \xi_i, \xi_i^* \geq 0 \qquad i = 1, 2, .., m, \;\; f(x_i) =< w, x_i > +b \end{cases}$$

1.5.1 Solving the Optimization Problem

Similar to SVMs in sections 1.2.1, 1.3.1, we can transform this primal form, equation(1.31), of optimization problem using non-negative Lagrange Multipliers as follows:

$$L = \frac{1}{2}||w||^2 + C\sum_{i=1}^{m}(\xi_i + \xi_i^*) - \sum_{i=1}^{m}\alpha_i(\xi + \xi_i - y_i + f(x_i)) - \sum_{i=1}^{m}\alpha_i^*(\xi + \xi_i^* + y_i - f(x_i)) - \sum_{i=1}^{m}(\eta_i\xi_i + \eta_i^*\xi_i^*) \tag{1.32}$$

$\alpha_i, \alpha_i^*, \eta_i, \eta_i^* \geq 0$

The partial derivatives of L with respect to $(w, b, \xi_i, \xi_i^*, \eta_i, \eta_i^*)$ have been described as follows:

$$\begin{cases} \frac{dL}{db} = \sum_{i=1}^{m}(\alpha_i^* - \alpha_i) = 0 \\ \frac{dL}{dw} = w - \sum_{i=1}^{m}(\alpha_i^* - \alpha_i)x_i = 0 \\ \frac{dL}{d\xi_i^*} = C - \alpha_i^* - \eta_i^* = 0 \\ \frac{dL}{d\xi_i} = C - \alpha_i - \eta_i = 0 \\ \frac{dL}{d\eta_i} = \sum_{i=1}^{m}\xi_i \leq 0 \\ \frac{dL}{d\eta_i^*} = \sum_{i=1}^{m}\xi_i^* \leq 0 \end{cases}$$

So the dual optimization problem becomes:

$$\text{Maximize} \begin{cases} -\frac{1}{2}\sum_{i,j=1}^{m}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*) < x_i, x_j > -\xi\sum_{i=1}^{m}(\alpha_i + \alpha_i^*) + \sum_{i=1}^{m} y_i(\alpha_i - \alpha_i^*) \\ Subject\ \ to: \sum_{i=1}^{m}(\alpha_i - \alpha_i^*) = 0 \qquad\qquad \alpha_i, \alpha_i^* \in [0, C] \end{cases}$$

From the second condition: $w = \sum_{i=1}^{m}(\alpha_i - \alpha_i^*)\alpha_i$, thus: $f(x) = \sum_{i=1}^{m}(\alpha_i - \alpha_i^*) < x_i, x > +b$

According to KKT conditions, only for $|f(x_i) - y_i| \geq \xi$ the Lagrange multipliers are non-zero. The standard SVR is as follows:

$$f(x) = \sum_{i=1}^{m}(\alpha_i^* - \alpha_i)K(x_i, x) + b \tag{1.33}$$

where $K(x_i, x) = \phi_i(x_i)\phi(x)$.

1.6 Principle Component Analysis (PCA)

1.6.1 Background

Another technique which is a very useful application in fields such as compression and classification would be PCA. It is also very powerful in face detection, image compression, and pattern recognition (Perez & Nussbaum, 2003).

PCA reduces the dimensionality of a data-set, finds a low dimensional feature space that can express the direction of the maximum variation in data, and keeps most of the sample's information.

The sample data-sets can be reconstructed with the new variables called Principle Components, which are linearly uncorrelated to each other. After we find principle components, we choose the ones can represent most of the information of the sample data as feature vectors (Perez & Nussbaum, 2003). The first principle component is the component which contains the greatest amount of the variance in the data.

1.6.2 Application

PCA transforms the origin data into a new coordinate system which has dimensions align with directions of maximal variation of data.

Assuming we have a data matrix with its variance- covariance matrix $\sum$, we can calculate the eigen-values and eigen-vectors corresponding to data covariance matrix. Eigen-values will help us determining the number of orthogonal components

we're using in our PCA. On the other hand, eigen-vectors will assist us discovering the relationship between principle components and original features we had in the data-set. Thus, eigen-values and eigen-vectors do the transformation of original feature space into principle components (Perez & Nussbaum, 2003).

In the third chapter, we will deal with large data-sets with many variables. Thus, we will apply PCA to reduce the number of interrelated variables to discover important features in our data-sets, and then will apply SVM algorithm on the new feature space.

1.7 Receiver Operating Characteristics Curve (ROC)

The ROC plot is a measure to evaluate the performance of classifiers.

Based on Confusion Matrix which is a two by two table, we can define the two basic measurements of 1-Specificity and Sensitivity which can help us with performance, visualization and selection of classifiers. An example of a confusion matrix for a binary classifier is as shown in figure (1.9) (Fawcett, 2006):

FIGURE 1.9: Confusion Matrix

Observed/Predicted	Class1(Positive)	Class2(Negative)
Class1(Positive)	a	b
Class2(Negative)	c	d

Assume we have two possible predicted class:$1, 2$ and a, b, c, d as follows:

a: is the number of correct predictions that the data point is in class(1).

b: is the number of incorrect predictions that the data point is in class(2).
c: is the number of incorrect predictions that the data point is in class(1).
d: is the number of correct predictions that the data point is in class(2).

From the confusion matrix we can define the following five basic terms:

True Positive Rate(TPR), or Sensitivity :$TPR = \frac{a}{a+b}$

The proportion of Class(1)-Positive cases which were correctly classified.

True Negative Rate(TNR), or Specificity :$TNR = \frac{d}{c+d}$

The proportion of Class(2)-Negative cases which were correctly classified.

False Positive Rate(FPR), or 1-Specificity :$FPR = \frac{c}{c+d}$

The proportion of Class(1)-Positive cases which were incorrectly classified as class2-Negative.

False Negative Rate(FNR):$FNR = \frac{b}{a+b}$

The proportion of Class(2)-Negative cases which were incorrectly classified as class(1)-Positive.

Accuracy Rate(ACR): $ACR = \frac{a+d}{a+b+c+d}$

The proportion of the total number of predictions that were correctly classified.

In ROC curve the Sensitivity is plotted in function of 1-Specificity for different data points. If the classification has been done perfectly, the ROC curve for sure passed through the upper left corner of the plot. So overally, if the ROC curve is closer to upper left corner of the plot, the accuracy rate of the test is higher and the ROC curve has a better performance (Fawcett, 2006).

1.7.1 Area Under The ROC Curve(AUC)

By using the ROC graph, we can always use the AUC measurement.

AUC is an area under the curve calculated in the ROC graph, such that AUC with score 1.0 is for the classifier with the perfect performance level, AUC with score 0.5 is for the classifier with the random performance level, and there is no AUC with score less than 0.5 (Fawcett, 2006).

Chapter 2

Customizing Kernel Functions in SVM

2.1 Convex Combination of Kernel Functions

Choosing an appropriate Kernel function is the most important step in SVMs classification methods (Abe, 2005).

In chapter one, in the simulated example, we only employed the single kernel classifiers, but a single kernel SVM may not be sufficient to solve complex real-world data-sets and may require to combine more than one kernel to develop a classifier with good outcomes. Thus, in this chapter we are going to propose SVMs with convex combination of kernels with non-negative weights. We wish to find the best combination of these base kernels considering that they individually satisfy Mercer's conditions.

addition to base kernels we introduced in chapter 1, we will discuss the following new feature mapping function in this chapter:

$$K(x, x') = \beta_1 K_{linear}(x, x') + \beta_2 K_{RBF}(x, x') + (1 - \beta_1 - \beta_2) K_{Polynomial}(x, x') \quad (2.1)$$

where $\beta_1, \beta_2, (1 - \beta_1 - \beta_2)$ are non-negative coefficients, and $0 \leq \beta_1, \beta_2 \leq 1$. K_{Linear} , K_{RBF}, and $K_{Polynomial}$ are valid kernels, thus, by Mercer's Theorem, $K(x, x')$ is also a valid kernel function.

2.1.1 Implementation in R (Simulated example)

Assuming we have a base kernel $K(x, x')$ and a set of 500 random numbers, $x = (x_1, x_2, .., x_{500})$, Gram Matrix of K at x is a $m * m$ matrix with: (Herbrich, 2016)

$$G_{ij} = K(x_i, x_j) \quad (2.2)$$

According to Mercer's Theorem, Gram Matrix G, is Positive Semi-Definite matrix and has its corresponding eigenvalue decomposition as follows: (Herbrich, 2016) $G = U\Lambda U'$ where $U = (u'_1, u'_2, .., u'_p)$ is eigen-vector matrix with $UU' = I$ and

$$\Lambda = \begin{bmatrix} \lambda 1 & 0 & 0 & ... & 0 \\ 0 & \lambda 2 & 0 & ... & 0 \\ .. & .. & \lambda 3 & ... & 0 \\ 0 & 0 & ... & .. & \lambda m \end{bmatrix} \qquad m \leq p, \qquad \lambda_1 \geq \lambda_2, ... \geq \lambda_m \geq 0$$

It's also possible to express Gram matrix elements in terms of the mapping feature $\phi(x_i) = \sqrt{\Lambda_i} u_i$:

$$G_{ij} =< \phi(x_i), \phi(x_j) >= (\sqrt{\lambda} u_i)'(\sqrt{\lambda} u_j) = u_i' \Lambda u_j = k_{ij} = \sum_{m=1}^{500} \lambda_m v_{im} v_{jm} \qquad (2.3)$$

Thus, the Gram matrix can be shown as follows;

$$G = \begin{bmatrix} < \phi(\vec{x_1}), \phi(\vec{x_1}) > & ... & ... & ... & < \phi(\vec{x_1}), \phi(\vec{x_n}) > \\ ... & < (\phi\vec{x_2}), \phi(\vec{x_2}) > & ... & ... & .. \\ & & & & \\ ... & ... & ... & ... & < \phi(\vec{x_n}), \phi(\vec{x_n}) > \end{bmatrix}$$

Following simulated experiments are the first ninth feature plots of different kernels vs our 500 data points:

Figure 2.1 shows the feature mapping vs data points in linear kernel SVM, and it seems like it has only one $\phi(x) = -x$ and the rest do not follow any $\phi(x)$.

Figure 2.2 images the feature mapping vs data points in RBF kernel SVM, apparently, it follows the $Sin(x)$, $Cos(x)$ functions patterns.

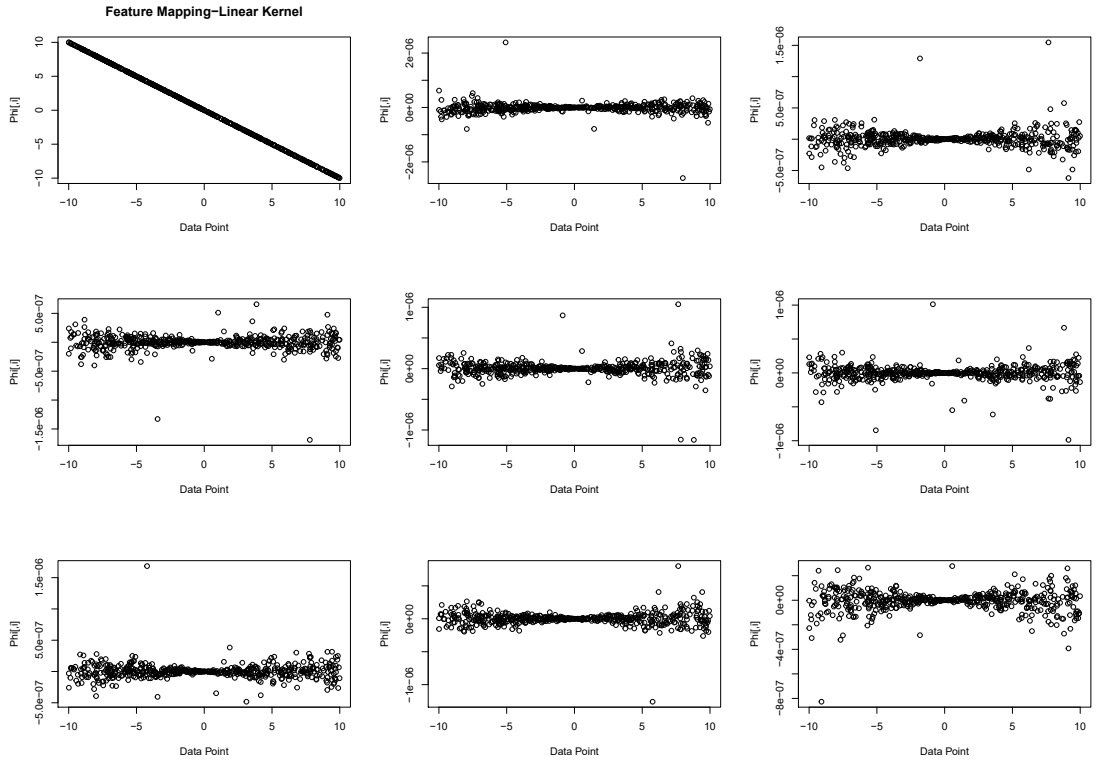


FIGURE 2.1: Feature mapping correspond to Linear Kernel($C = 1$) - Simulated Example

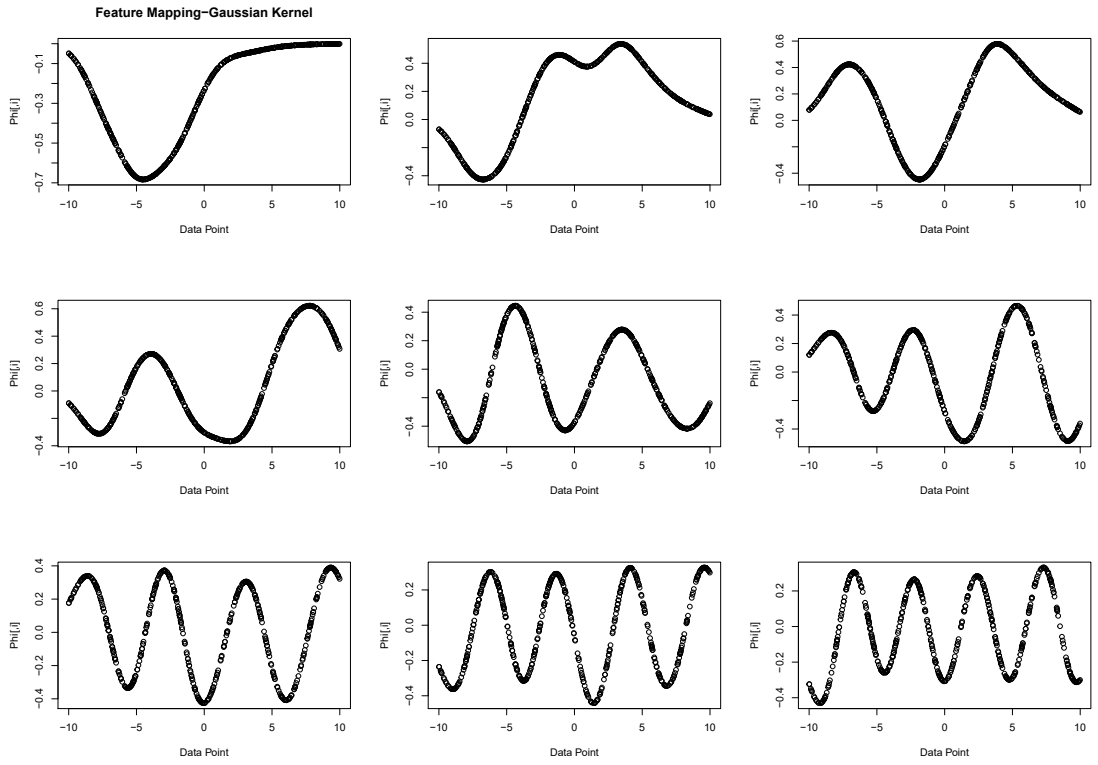


FIGURE 2.2: Feature mapping correspond to RBF Kernel ($C = 1, \sigma = 0.5$)- Simulated Example

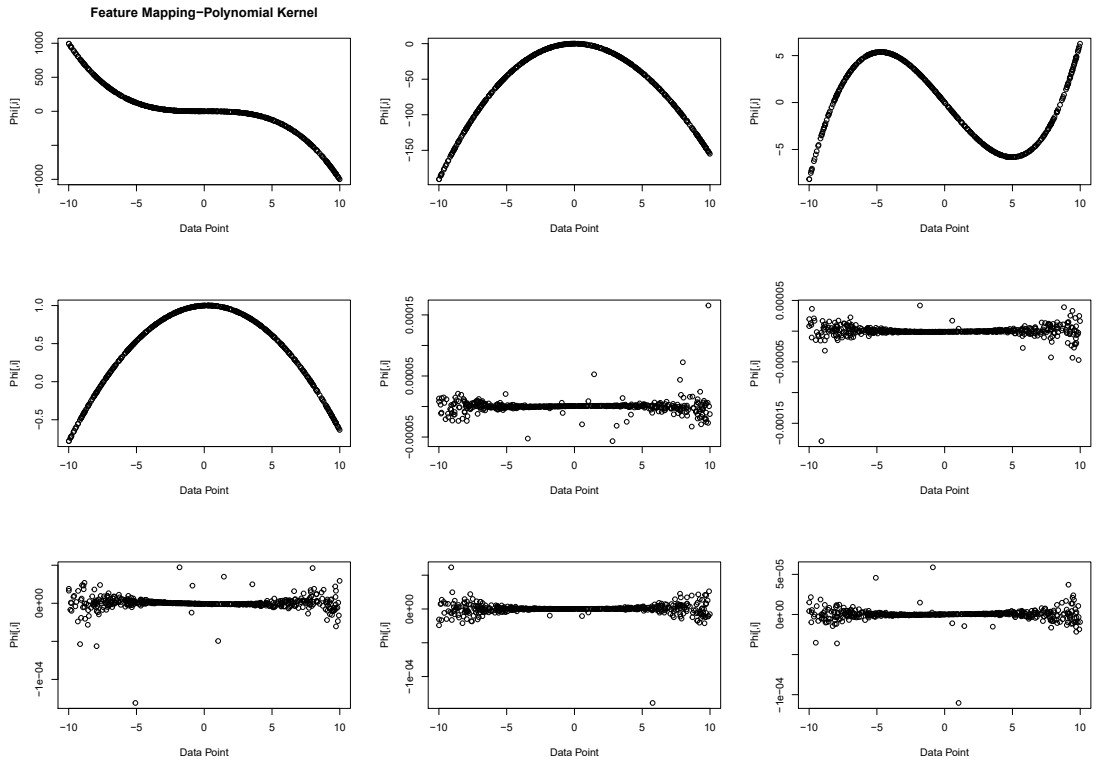


FIGURE 2.3: Feature mapping correspond to Polynomial Kernel($d = 3$) - Simulated Example

Figure 2.3 illustrates the feature mapping vs data points in Polynomial kernel SVM, as we can see it only has four features $(\phi_1(x), \phi_2(x), \phi_3(x), \phi_4(x))$.

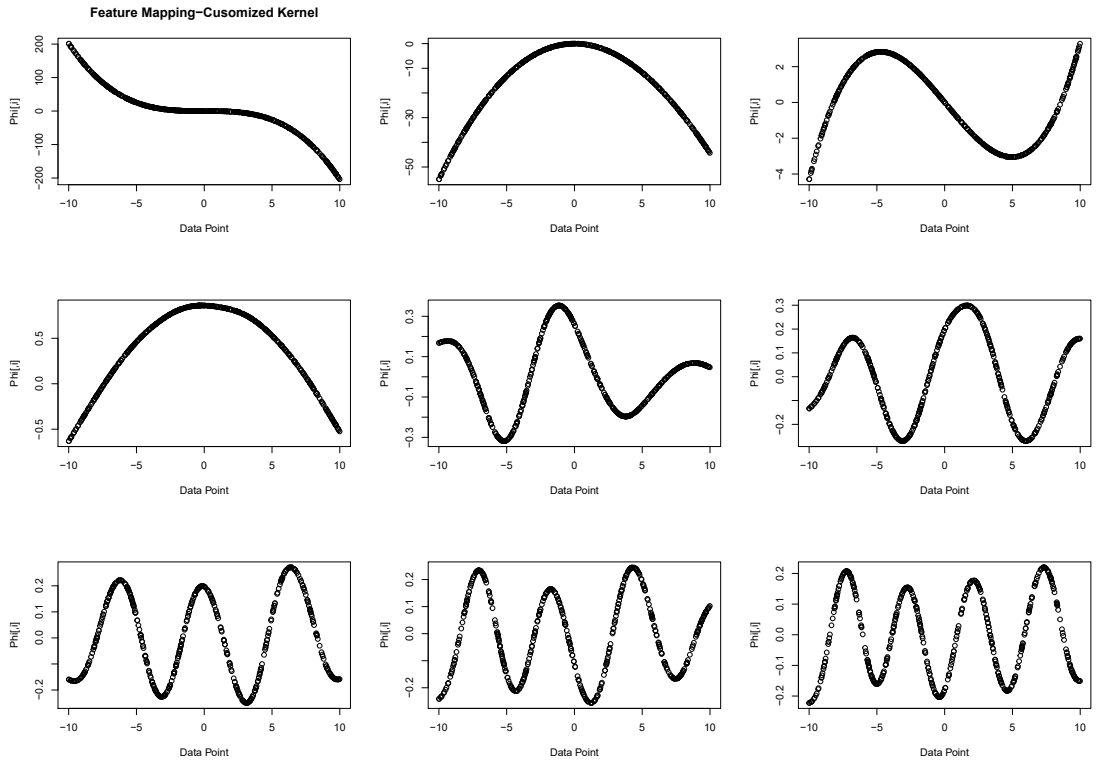


FIGURE 2.4: Feature mapping correspond to Customized Kernel($0.33 * LinearKernel + 0.33 * RBFKernel + 0.33 * PolynomialKernel$ - Simulated Example

Figure 2.4 denotes the feature mapping vs data points in customized kernels SVM. The first plot shows the linearity and non-linearity, the next three plots are similar to polynomial kernel and the rest behave like RBF kernel SVM.

Thus, we can say that by combining three kernels together, we are basically, giving more essence of Linear, RBF and Polynomial features to the feature space. Based on the data-set distributions, this method can construct more accurate classification.

The other major challenges that are encounter in SVMs algorithm are selecting the best coefficient for each of the base kernels and the optimal values of the SVMs parameters.

Our method to select the best weights in this combination is as follows:

Due to the constraint we had in equation (2.1), $\sum_{i=1}^{3} \beta_i = 1$, we need to take into account that $0 \leq \beta_1 \leq 1$, $\beta_2 \leq 1 - \beta_1$, and $\beta_3 = 1 - \beta_1 - \beta_2$.

So by considering these three terms, we generate randomly a vector of 400 random numbers between $[0, 1]$ with the sequence of 0.01 as our β_1.

Similarlly, we generate our vector of β_2 except that it needs to be less than $1 - \beta_1$.

Afterwards, among these two vectors, we select those combinations of β_1 and β_2 which are unique.

Next, we construct the vector of β_3 based on the selected β_1, β_2.

Using this method we generate all random combinations of β_1, β_2, and β_3.

To measure the performance of the proposed kernel, it is mandatory step to tune its parameters to increase the success rate of the classification. In the next chapter we will go through the experimental setup to apply all we mentioned in chapter one and two on the real-world data-sets.

Chapter 3

Experimental Setup

3.1 Background

This section evaluates the SVMs (with base Kernels, and Customized Kernels) on real-world data-sets. The results of SVMs with customized kernels will be compared to results of SVMs with base kernels, and also we consider the comparison of our method with Logistic Regression as the other classification method.

Furthermore, we are going to apply PCA on our large- scale data-sets as a dimensionality reduction technique. Once we finish performing these methods on our data, we will execute our SVMs algorithm on them to discover the differences.

3.2 Implementation in R

In this section we work on three real- world data-sets which have been taken from the UCI Repository of Machine Learning at

(ftp.ics.uci.edu://pub/machine-learning-databases

http://www.ics.uci.edu/mlearn/MLRepository.html) :

■ One of the data we are going to use in this section is called SpamBase, which is about Spam/ non- Spam e-mail automatic detection. The last column of this data-set denotes whether the e-mail's type was considered Spam or not. There are also other 57 attributes in this data-set which indicate the frequencies of a specific word or a character was appearing in the e-mail that consequently, have some relation with the type of the e-mail (Dheeru & Karra Taniskidou, 2017).

Here are the definitions of the attributes: (Dheeru & Karra Taniskidou, 2017)

▲ 48 continuous real [0,100] attributes of type word-freq-WORD = percentage of words in the e-mail that match WORD.
▲ 6 continuous real [0,100] attributes of type char-freq-CHAR]= percentage of characters in the e-mail that match CHAR.
▲ 1 continuous real [1,...] attribute of type capital-run-length-average =average length of uninterrupted sequences of capital letters.
▲ 1 continuous integer [1,...] attribute of type capital-run-length-longest = length of longest uninterrupted sequence of capital letters.
▲ 1 continuous integer [1,...] attribute of type capital-run-length-total . = sum of length of uninterrupted sequences of capital letters = total number of capital letters in the e-mail.
▲ 1 nominal 0,1 class attribute of type spam = denotes whether the e-mail was considered spam (1) or not (0).

■ The other data- set we are going to take under consideration is Pima-Indian-Diabetes data-set. The goal of this data-set is to predict the existence of diabetes based on given the nine diagnostic measures among patients.

The last column of this data-set denotes whether the patient is diagnosed with

diabetes or not. In this data all the patients are female over 21 years old of Pima Indian heritage (Dheeru & Karra Taniskidou, 2017). There are 9 attributes and 768 observations in this data-sets as follows:

▲ Pregnancies: Number of times pregnant
▲ Glucose: Plasma glucose concentration a 2 hours in an oral glucose tolerance test
▲ BloodPressure: Diastolic blood pressure (mm Hg)
▲ SkinThickness: Triceps skin fold thickness (mm)
▲ Insulin: 2-Hour serum insulin (mu U/ml)
▲ BMI: Body mass index (weight in kg/(height in m)^2)
▲ DiabetesPedigreeFunction: Diabetes pedigree function
▲ Age: Age (years)
▲ Outcome: Class variable (0 or 1)

■ And the last data-set we are going to take into account is the Wisconsin-Breast-Cancer data-sets. The objective of this data-set is to predict the diagnosis of breast cancer after analyzing the pre-recorded data. The attributes definition is as follows:

▲ radius (mean of distances from center to points on the perimeter)
▲ texture (standard deviation of gray-scale values)
▲ perimeter
▲ area
▲ smoothness (local variation in radius lengths)
▲ compactness (perimeter^2 / area - 1.0)
▲ concavity (severity of concave portions of the contour)
▲ concave points (number of concave portions of the contour)
▲ symmetry
▲ fractal dimension ("coastline approximation" - 1)
▲ Diagnosis (M = malignant, B = benign)

Data-set	Attributes	Instances
SpamBase	58	4601
Pima-Indian-Diabetes	9	768
Wisconsin-Breast-Cancer	11	699

TABLE 3.1: The data-sets were used to express the performance of our method

3.3 Applications of SVM- SpamBase data-set

As we already mentioned above, SpamBase data-set denotes whether the e-mails are spam or not. From the following table we can see the proportion of each class in this data-set:

	Class1- Spam	Class0- NonSpam
Frequency	1813	2788
Percentage	39.4	60.59

TABLE 3.2: The frequency table of SpamBase data-set

One of the most powerful software which is widely used for implementation of SVMs is R, and also some of the related packages for SVMs in R are "*kernlab*" and "*e*1071". These packages are capable of fitting different SVMs on data-sets and get the results of them. The related R codes are attached to appendix (A.2).

In this experiment in order to evaluate our classifier, we use ten-fold Cross-Validation. By using this technique we randomly partition our original sample into 10 sub-samples. Out of these 10 sub-samples, we take 9 of them as our training-set, and the other one, as a test-set. We repeat it for 10 times to use every sub-sample as our validation test-set once, then the ten results will be averaged to produce our estimation. The cross-validation procedure can prevent the over-fitting problem.

There are couple of basic measures such as, Classification Error rate(CER), False Positive Rate(FPR), False Negative Rate(FNR), Accuracy Rate(ACR), and Area Under ROC Curve (AUC), which help us evaluate the quality of prediction on our test set.

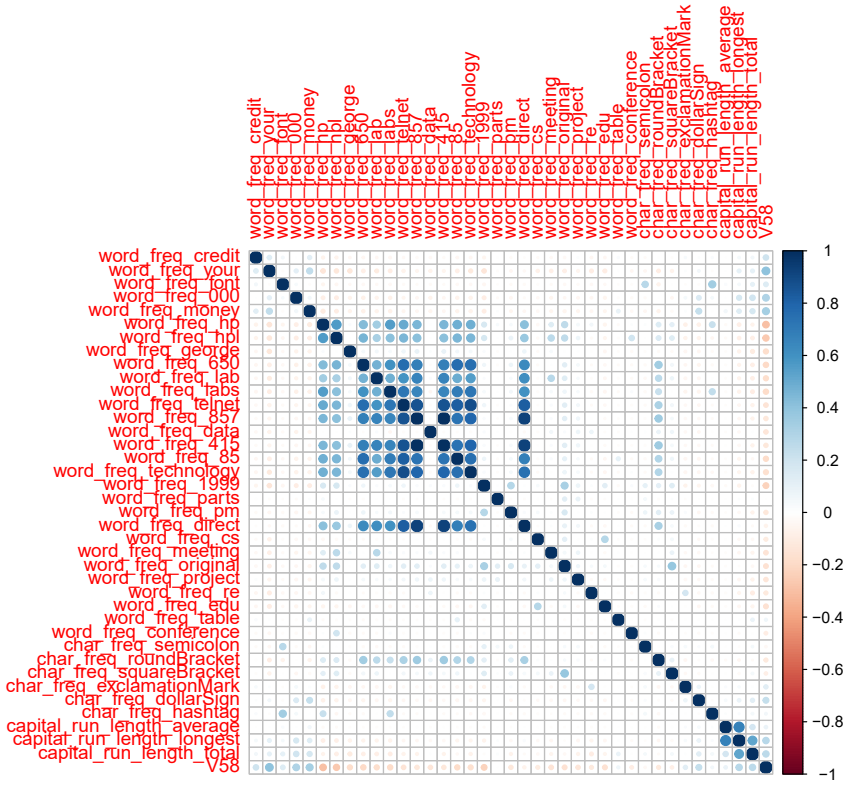


FIGURE 3.1: Correlation between features- Sample SpamBase data-set

Applying all these measures on our SpamBase Data-set is too time and resource consuming, therefore we decide to take a sample of 1000 data-points to execute our study. The following tables show the proportion of Spam(non- Spam) E-mails in the sample, and also give a brief summary of these measures after applying basic Kernels on our SpamBase data-set:

	Class1- Spam	Class0- NonSpam
Frequency	394	606
Percentage	39.4	60.6

TABLE 3.3: The frequency table of SpamBase SAMPLE-data-set

	CER	FPR	FNR	ACR	AUC	# SV
Linear Kernel	0.098	0.134	0.058	0.91	0.951	162
RBF Kernel	0.10	0.105	0.10	0.89	0.956	509
Polynomial Kernel	0.28	0.108	0.31	0.71	0.915	560
Logistic Regression	0.1	0.157	0.05	0.9	0.945	NA

TABLE 3.4: SpamBase Sample data-set, basic kernels Average-Results

We fit models with Linear kernel, RBF kernel, Polynomial kernels with their default parameters' values which is $C_{linear} = 1$, $C_{RBF} = 1$, $\gamma = 1/dim[dataset]$, and $d = 3$.

Selection of the kernel parameters is a principal step in the area of Support Vector Machine. In order to get an optimized SVMs with classical Kernels, we tune their parameters 10-fold cross-validation with trying the different range of parameters' value and compare the performance. The goal is to identify the best value for each parameter, so that the classifier can accurately predict test data-set (Hsu, Chang, Lin, et al., 2003).

First, in Linear Kernel function ($H(x, x') = x^T x'$), we only consider the optimization of the parameter C(cost), this C_{linear} will be chosen from the range of {0.01, 0.1, 1, 10, 100, 1000}. The larger C gives lower-bias and higher- variance model.

Next, in RBF Kernel function ($H(x, x') = exp(-\gamma||x - x'||^2)$), addition to parameter C_{RBF}, which is chosen from the same range of C_{linear}, we need to tune the parameter γ, which is a positive parameter that can define the radius (Abe, 2005). Intuitively, if γ is small, then the variance is large which implies that the support vector has wide- spread influence and leads to low-bias models, and vice-versa for

the large γ.

We are going to select γ from the range $\{0.01, 0.1, 1, 10, 100, 1000\}$. Now, after we chose the range of C_{RBF}, γ, we use "Grid- Search" on C, γ using 10-fold cross-validation to select the best performance among various pairs of (C, γ).

Finally, in Polynomial Kernel function ($H(x, x') = (x^T x' + 1)^d$), in order to choose an appropriate parameter value, we will select d (the degree of the polynomial) from the range of $\{2, 3, 4, 5, 6, 7, 8, 9\}$.

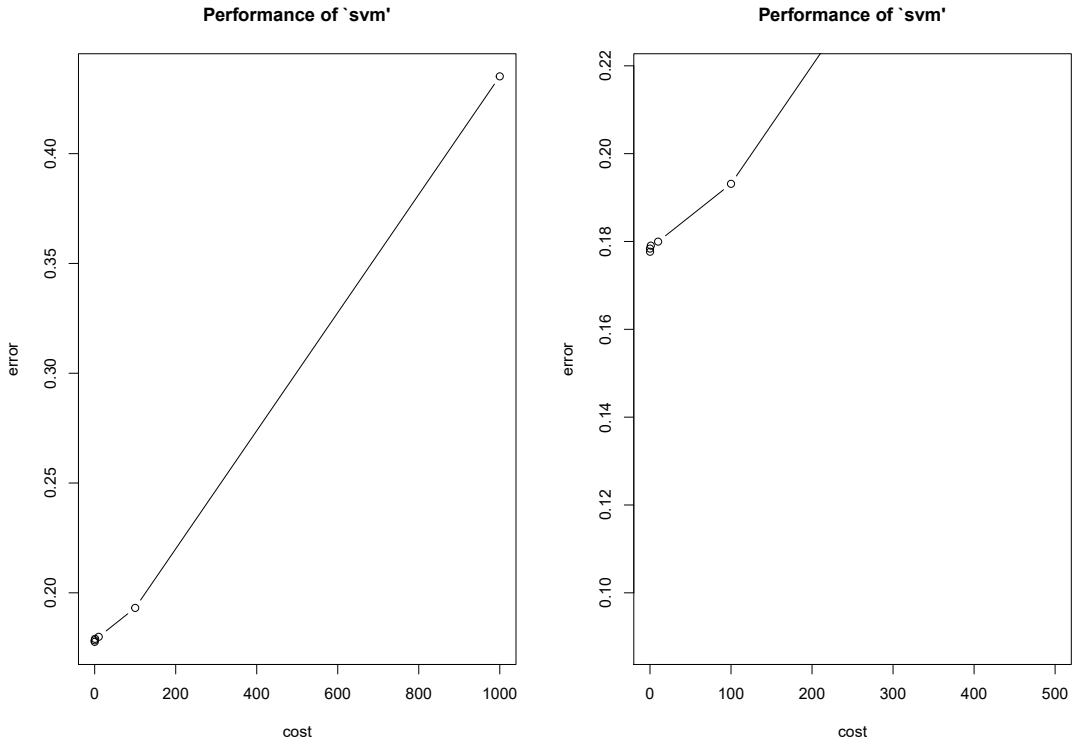


FIGURE 3.2: Tuning the parameter Cost, Linear Kernel, Sample SpamBase

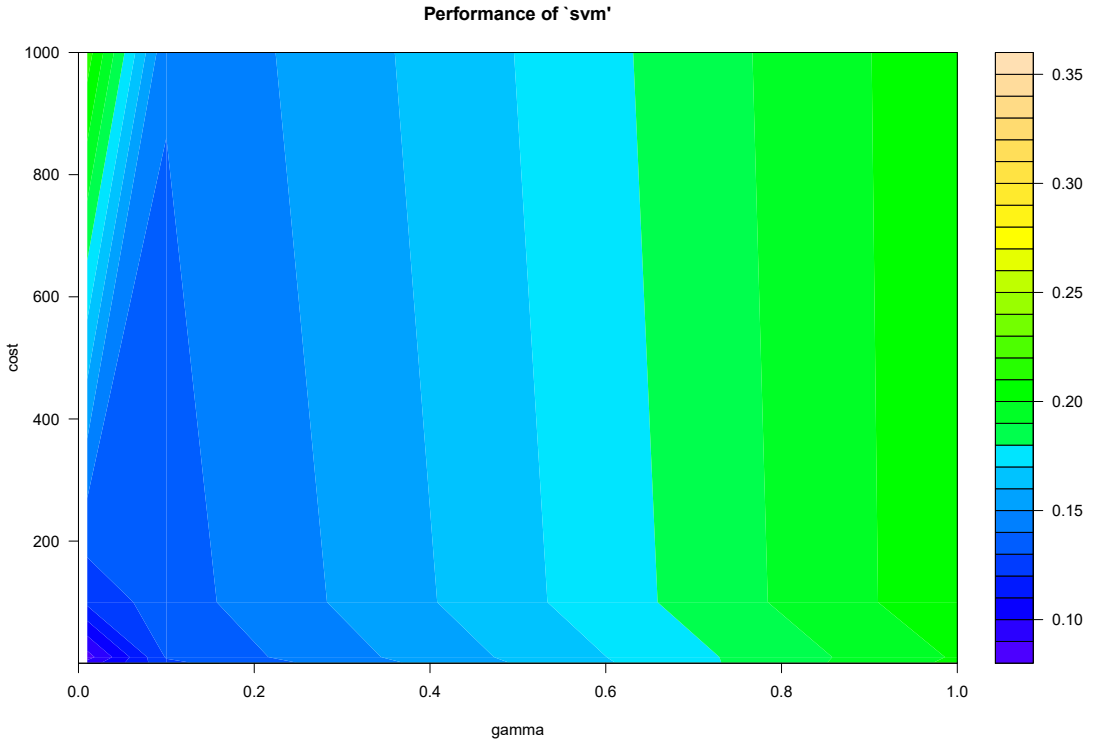


FIGURE 3.3: Tuning the parameter Cost,γ, RBF Kernel, Sample SpamBase

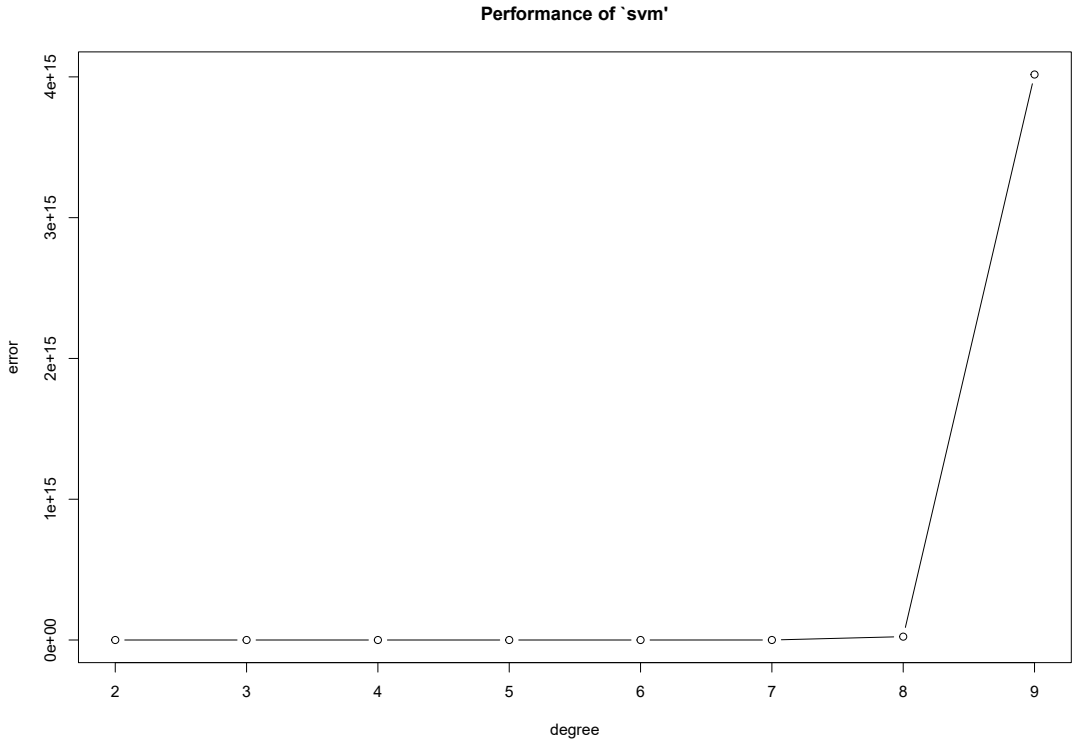


FIGURE 3.4: Tuning the parameter degree, Polynomial Kernel, Sample SpamBase

The results of tuning parameters in classical kernels on SpamBase data-set are conducted in table 3.5.

It shows the slightly improvements on the errors rate, accuracy rate, and area under ROC plot. Best performance assigned to SVM with RBF kernels ($\gamma = 0.01, C_{RBF} = 10$) with 90% accuracy rate, and 17% improvements in FNR.

	CER	FPR	FNR	ACR	AUC	#SV
Linear Kernel(C=0.1)	0.096	0.133	0.068	0.90	0.952	201
RBF Kernel(gamma=0.01, C=10)	0.1	0.123	0.083	0.9	0.964	223
Polynomial Kernel(d=2)	0.226	0.093	0.263	0.77	0.917	446
Logistic Regression	0.1	0.157	0.05	0.9	0.945	NA

TABLE 3.5: SpamBase Sample data-set, TUNED-basic kernels Results

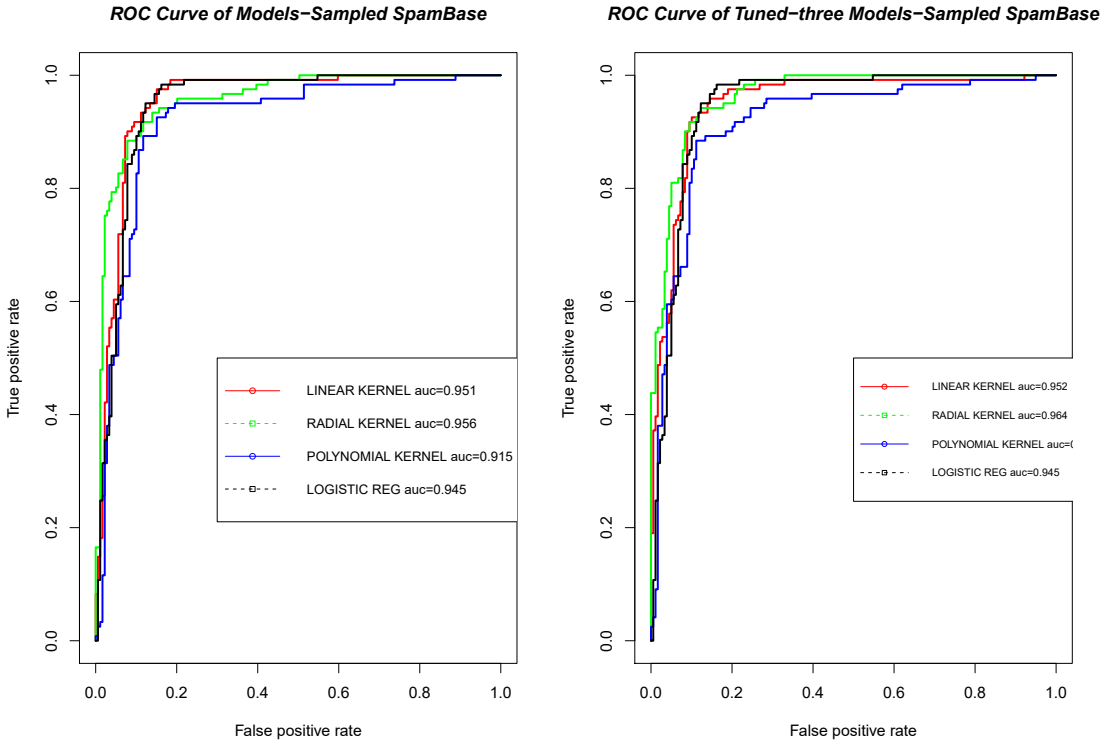


FIGURE 3.5: ROC curve, Basic Kernels vs Logistic Reg, Tuned Basic Kernels vs Logistic Reg, Sample SpamBase

As we already mentioned in Chapter 1, one of the common technique to reduce the dimensionality of data-set is Principal Components Analysis (PCA). Although

PC1	PC2	PC3	PC4	PC5	PC6	PC7
0.275368766	-0.642958582	0.094378031	-0.074509982	-0.029040853	-0.116031537	-0.048563547
PC8	PC9	PC10	PC11	PC12	PC13	PC14
0.077837107	-0.024104329	0.071553836	0.101779209	-0.054194468	-0.031444830	-0.026170811
PC15	PC16	PC17	PC18	PC19	PC20	PC21
0.039685942	0.008086267	0.020747465	0.037911564	0.031605140	0.009877624	-0.030447840
PC22	PC23	PC24	PC25	PC26	PC27	PC28
-0.036746918	-0.013489834	-0.006692263	-0.015245681	-0.004653620	0.030091821	0.050851432
PC29	PC30	PC31	PC32	PC33	PC34	PC35
0.008372082	-0.061496150	-0.017183532	-0.018691508	0.016876327	0.038468740	-0.034032199
PC36	PC37	PC38	PC39	PC40	PC41	V58
-0.002422441	0.035203717	-0.034045235	-0.042713355	0.069131365	-0.020197758	1.000000000

FIGURE 3.6: Correlation between first 41 PCs with response variable- Sample SpamBase data-set

SVM classification is very accurate, but it is very time consuming if you are dealing with a large scale data-set (Sundaram, 2009).

Here after we applied PCA on our Centered, Scaled data-set, we select principle components which capture at least 90% of the total variance. In the above figure we can see the correlation between Principle Components and the response variable. PC1 has the greatest relationship with response variable, while PC36 has the lowest impact on dependent variable.

```
Importance of components:
                          PC1     PC2     PC3     PC4    PC5     PC6     PC7     PC8     PC9
Standard deviation     2.7285 1.84046 1.41475 1.32670 1.2900 1.24247 1.21660 1.20209 1.17633
Proportion of Variance 0.1306 0.05943 0.03511 0.03088 0.0292 0.02708 0.02597 0.02535 0.02428
Cumulative Proportion  0.1306 0.19003 0.22515 0.25603 0.2852 0.31231 0.33827 0.36363 0.38790
                          PC10    PC11    PC12    PC13    PC14    PC15    PC16    PC17    PC18
Standard deviation     1.16325 1.12807 1.11263 1.09535 1.06848 1.05614 1.04900 1.03286 1.02287
Proportion of Variance 0.02374 0.02233 0.02172 0.02105 0.02003 0.01957 0.01931 0.01872 0.01836
Cumulative Proportion  0.41164 0.43397 0.45568 0.47673 0.49676 0.51633 0.53564 0.55435 0.57271
                          PC19    PC20    PC21    PC22    PC23    PC24    PC25    PC26    PC27
Standard deviation     1.0130 1.00694 1.00133 0.99464 0.98683 0.97778 0.96114 0.95728 0.94457
Proportion of Variance 0.0180 0.01779 0.01759 0.01736 0.01708 0.01677 0.01621 0.01608 0.01565
Cumulative Proportion  0.5907 0.60850 0.62609 0.64345 0.66053 0.67730 0.69351 0.70959 0.72524
                          PC28    PC29    PC30    PC31    PC32    PC33    PC34    PC35   PC36
Standard deviation     0.93599 0.92548 0.92433 0.89726 0.87553 0.86510 0.86456 0.84169 0.8307
Proportion of Variance 0.01537 0.01503 0.01499 0.01412 0.01345 0.01313 0.01311 0.01243 0.0121
Cumulative Proportion  0.74061 0.75564 0.77063 0.78475 0.79820 0.81133 0.82444 0.83687 0.8490
                          PC37    PC38   PC39    PC40   PC41    PC42    PC43    PC44    PC45
Standard deviation     0.82439 0.82264 0.8062 0.78903 0.7847 0.77144 0.73918 0.73104 0.72169
Proportion of Variance 0.01192 0.01187 0.0114 0.01092 0.0108 0.01044 0.00959 0.00938 0.00914
Cumulative Proportion  0.86090 0.87277 0.8842 0.89510 0.9059 0.91634 0.92593 0.93530 0.94444
                          PC46    PC47    PC48    PC49    PC50    PC51   PC52    PC53    PC54
Standard deviation     0.71726 0.67999 0.66484 0.58887 0.57667 0.55790 0.4652 0.43812 0.41591
Proportion of Variance 0.00903 0.00811 0.00775 0.00608 0.00583 0.00546 0.0038 0.00337 0.00303
Cumulative Proportion  0.95347 0.96158 0.96933 0.97542 0.98125 0.98671 0.9905 0.99387 0.99691
                          PC55   PC56    PC57
Standard deviation     0.30448 0.2830 0.05845
Proportion of Variance 0.00163 0.0014 0.00006
Cumulative Proportion  0.99854 0.9999 1.00000
```

FIGURE 3.7: Importance of Principle Components- SpamBase Sample data-set

The above summary method denotes the importance of principle components in our data-set. As we can see, Principle Components are in such way that the higher significance factors are placed first, and the subsequent factors are afterwards. The first Principle Component can explain 13% of the variance of the data, the second Principle Component has an importance of 6% and so on (Figure 3.8, 3.9).

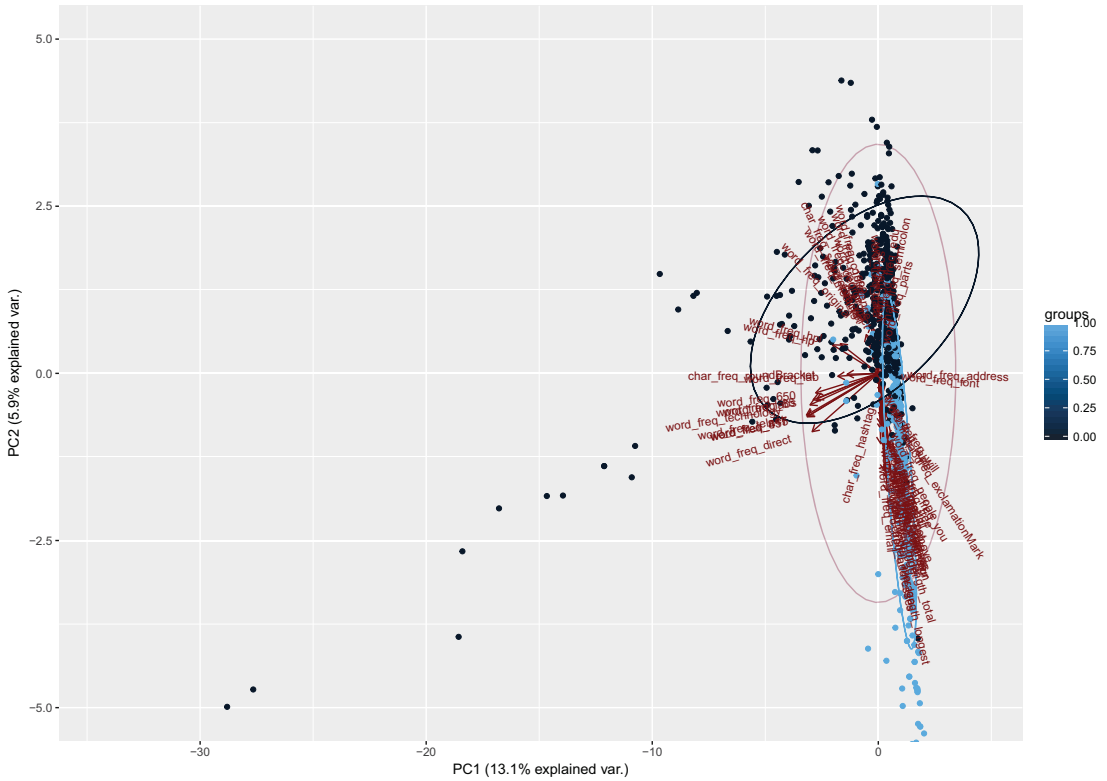


FIGURE 3.8: PC1 vs PC2 - SpamBase Sample data-set

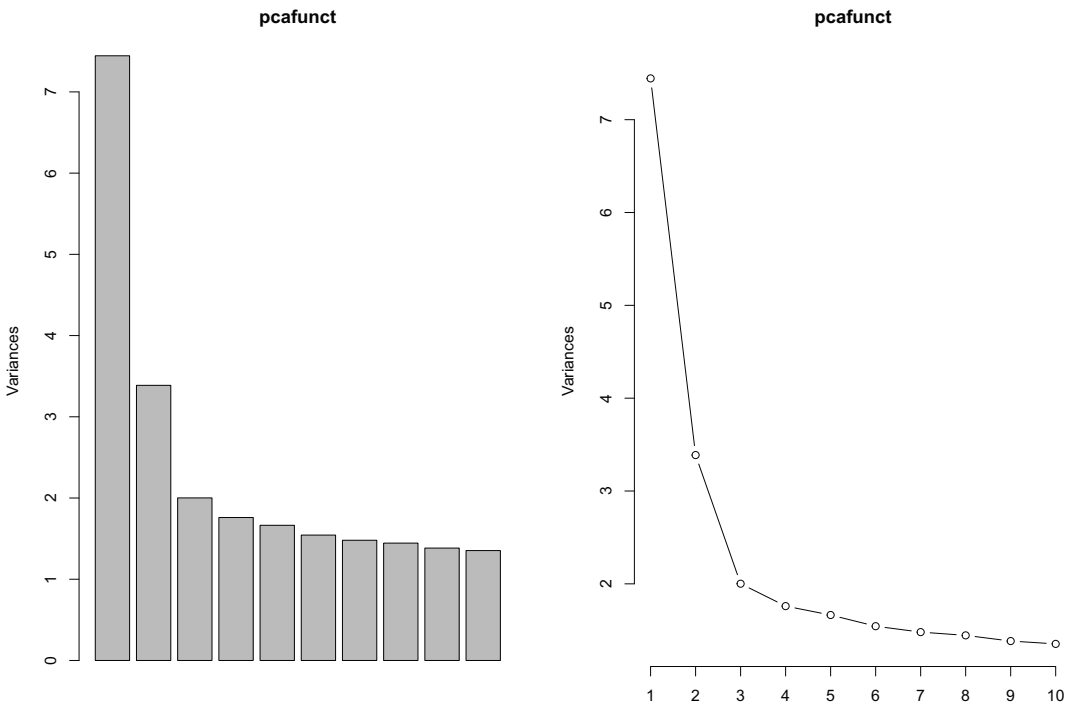


FIGURE 3.9: Top Ten PCS- SpamBase Sample data-set

Table 3.6 shows that the proposed model with 41 Principle Components achieved approximately same results as our data-set with 57 variables had in terms of detection errors and accuracy rate.

	CER	FPR	FNR	ACR	AUC	# SV
Linear Kernel	0.1	0.14	0.064	0.90	0.944	179
RBF Kernel	0.11	0.12	0.105	0.89	0.954	516
Polynomial Kernel	0.2	0.14	0.33	0.69	0.96	569
Logistic Regression	0.1	0.15	0.064	0.89	0.943	NA

TABLE 3.6: SpamBase Sample data-set, Based on 41 PCAs-basic kernels Results

The following table demonstrates the results of applying Tuned SVMs with 41 PCs on our data-sets, in which you see they are very close to the results of sample SpamBase data-set.

	CER	FPR	FNR	ACR	AUC	#SV
Linear Kernel(C=0.01)	0.17	0.084	0.189	0.856	0.95	549
RBF Kernel(gamma=0.01, C=10)	0.103	0.12	0.098	0.89	0.961	420
Polynomial Kernel(d=2)	0.273	0.124	0.305	0.73	0.89	531
Logistic Regression	0.1	0.15	0.064	0.89	0.943	NA

TABLE 3.7: SpamBase Sample data-set, Based on 41 PCAs-TUNED basic kernels Results

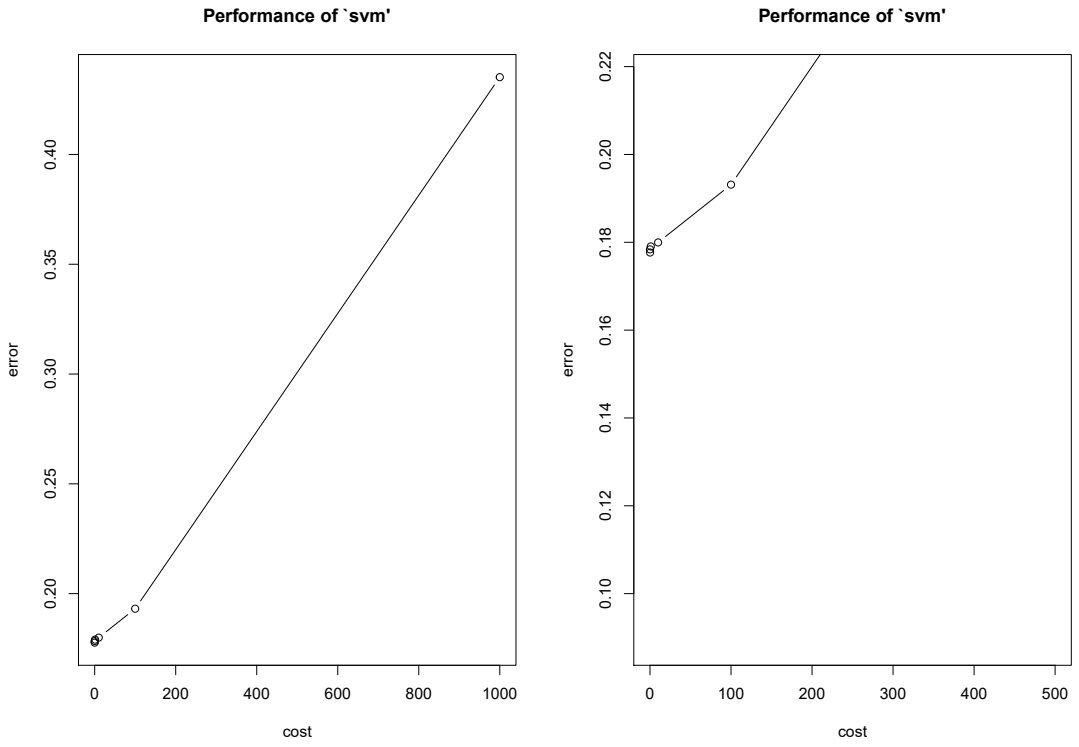


FIGURE 3.10: Tuning the parameter Cost, Linear Kernel, Sample SpamBase

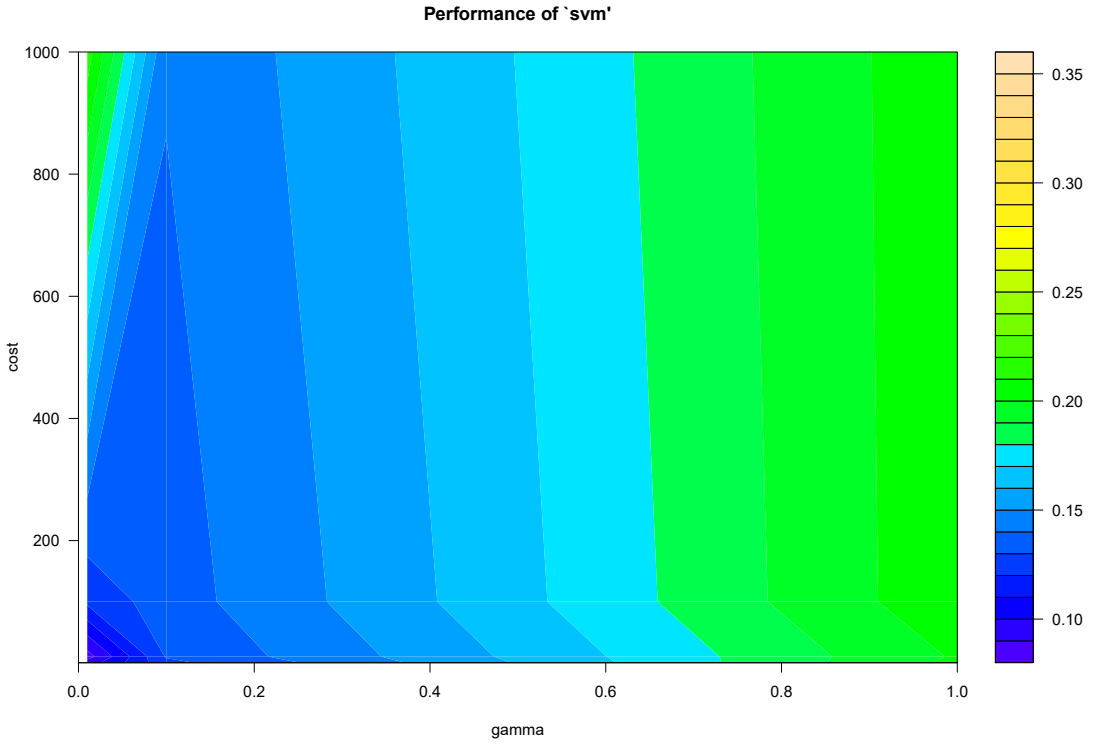


FIGURE 3.11: Tuning the parameter Cost,γ, RBF Kernel, Sample SpamBase

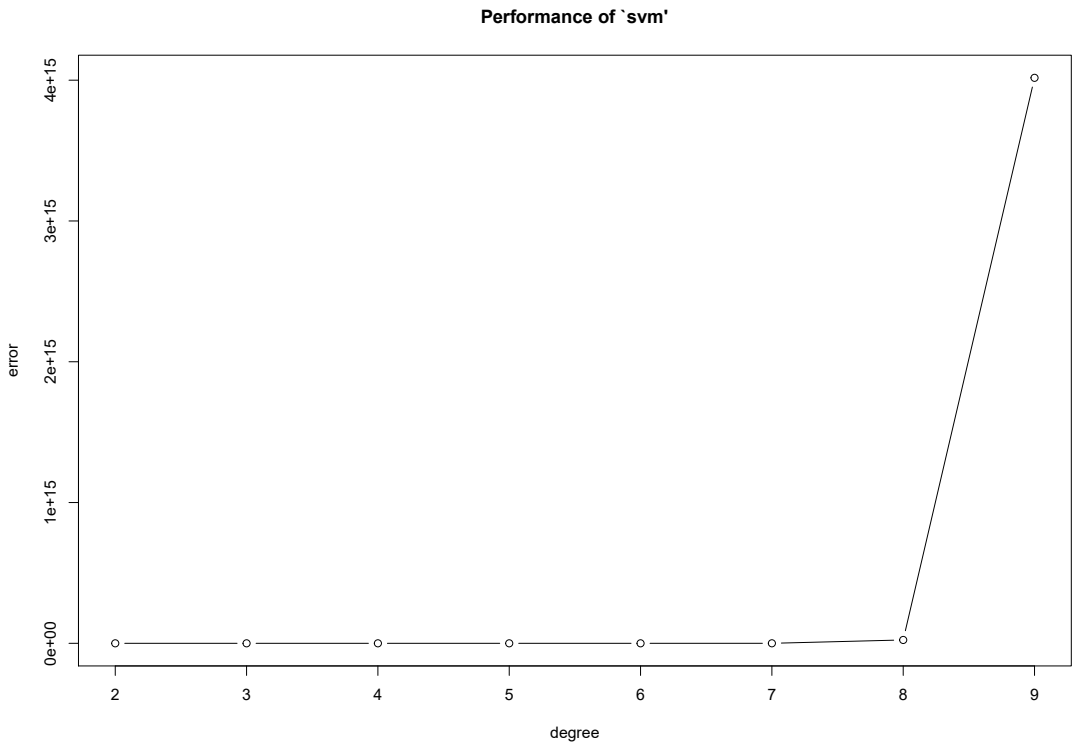


FIGURE 3.12: Tuning the parameter degree, Polynomial Kernel, Sample Spam-Base

Thus, combining PCA and SVM has great performance in our data classification algorithm, and it saved a lot of the detection time by reducing the dimensionality of the data-set.

More over, after going in depth of experiment on our data-set, we reached to the point that by taking just first 20 PCs, we lose on the accuracy rate only slightly, and still stand on a great accuracy rate and performance.

Here is the table of results for implementing only 20 PCs in our data-set, followed by the tuned model results.

	CER	FPR	FNR	ACR	AUC	# SV
Linear Kernel	0.1	0.11	0.06	0.90	0.954	182
RBF Kernel	0.096	0.081	0.10	0.90	0.949	523
Polynomial Kernel	0.25	0.08	0.31	0.74	0.901	537
Logistic Regression	0.096	0.11	0.08	0.89	0.943	NA

TABLE 3.8: SpamBase Sample data-set, Based on 20 PCAs-basic kernels Results

	CER	FPR	FNR	ACR	AUC	#SV
Linear Kernel(C=0.01)	0.15	0.06	0.18	0.85	0.95	600
RBF Kernel(gamma=0.1, C=1)	0.096	0.103	0.092	0.9	0.948	516
Polynomial Kernel(d=2)	0.26	0.1	0.315	0.72	0.88	528
Logistic Regression	0.096	0.11	0.08	0.89	0.949	NA

TABLE 3.9: SpamBase Sample data-set, Based 20 PCAs-TUNED basic kernels Results

3.3.1 Implementation of SVMs with New-Kernel Function

Our new combined kernel is a convex combination of classical kernels for which the weights of the kernels are computed randomly, and executed in our SVM model to derive the results in order to verify whether the new kernel algorithm is more

capable of classifying the data or not.

We will do the comparison between the outcome of this model with the results of tuned models from the previous experiments which were done above.

First, we need to take a look to the randomly selected β_1,β_2, and $\beta 3$.

	beta1	beta2	beta3
1	0.29	0.33	0.38
2	0.41	0.41	0.18
3	0.89	0.04	0.07
4	0.04	0.75	0.21
5	0.55	0.45	0
6	0.46	0.49	0.05
7	0.45	0.1	0.45
8	0.1	0.58	0.32
9	0.04	0.06	0.9
10	0.33	0.23	0.44
11	0.66	0.1	0.24
12	0.71	0.18	0.11
13	0.6	0.18	0.22
14	0.29	0.11	0.6
15	0.14	0.08	0.78
16	0.02	0.12	0.86
17	0.48	0.04	0.48
18	0.76	0.05	0.19
19	0.21	0.16	0.63
20	0.32	0.22	0.46
21	0.14	0.69	0.17
22	0.41	0.43	0.16
23	0.41	0.5	0.09
24	0.37	0.36	0.27
25	0.15	0.5	0.35
26	0.14	0.85	0.01
27	0.23	0.73	0.04
28	0.26	0.46	0.28
29	0.04	0.4	0.56
30	0.44	0.34	0.22
31	0.12	0.24	0.64
32	0.2	0.65	0.15
33	0.12	0.2	0.68
34	0.9	0.08	0.02
35	0.09	0.19	0.72
36	0.27	0.64	0.09
37	0.45	0.32	0.23
38	0.76	0.22	0.02
39	0	0.58	0.42
40	0.22	0.34	0.44
41	0.38	0.59	0.03
42	0.24	0.44	0.32

	beta1	beta2	beta3
43	0.1	0.66	0.24
44	0.43	0.56	0.01
45	0.17	0.62	0.21
46	0.13	0.54	0.33
47	0.18	0.06	0.76
48	0.09	0.55	0.36
49	0.47	0.51	0.02
50	0.51	0.27	0.22
51	0.33	0.67	0
52	0.61	0.25	0.14
53	0.41	0.56	0.03
54	0.14	0.47	0.39
55	0.3	0.57	0.13
56	0.06	0.85	0.09
57	0.72	0.11	0.17
58	0.4	0.08	0.52
59	0.31	0.49	0.2
60	0.37	0.3	0.33
61	0.15	0.8	0.05
62	0.09	0.39	0.52
63	0.14	0.46	0.4
64	0.62	0.34	0.04
65	0.67	0.22	0.11
66	0.79	0.03	0.18
67	0.44	0.29	0.27
68	0.41	0.47	0.12
69	0.01	0.09	0.9
70	0.18	0.66	0.16
71	0.23	0.43	0.34
72	0.07	0.64	0.29
73	0.5	0.5	0
74	0.39	0.58	0.03
75	0.24	0.04	0.72
76	0.11	0.67	0.22
77	0.39	0.61	0
78	0.44	0.17	0.39
79	0.22	0.03	0.75
80	0.5	0	0.5
81	0.35	0.55	0.1
82	0.37	0.38	0.25
83	0.35	0.51	0.14
84	0.53	0.14	0.33

	beta1	beta2	beta3
85	0.41	0	0.59
86	0.26	0.09	0.65
87	0.67	0.14	0.19
88	0.53	0.28	0.19
89	0.71	0.15	0.14
90	0.48	0.41	0.11
91	0.26	0.44	0.3
92	0.32	0.18	0.5
93	0.62	0.17	0.21
94	0.47	0.18	0.35
95	0.41	0.48	0.11
96	0.57	0.19	0.24
97	0.4	0.2	0.4
98	0.36	0.4	0.24
99	0.29	0.71	0
100	0.17	0.34	0.49
101	0.21	0.14	0.65
102	0.04	0.93	0.03
103	0.7	0.01	0.29
104	0.41	0.01	0.58
105	0.92	0.07	0.01
106	0.73	0.16	0.11
107	0.69	0.18	0.13
108	0.05	0.26	0.69
109	0.39	0.39	0.22
110	0.48	0.11	0.41
111	0.7	0.3	0
112	0.54	0.09	0.37
113	0.05	0.51	0.44
114	0.26	0.17	0.57
115	0.4	0.52	0.08
116	0.19	0.57	0.24
117	0.15	0.51	0.34
118	0.81	0.01	0.18
119	0.55	0.05	0.4
120	0.17	0.17	0.66
121	0.22	0.77	0.01
122	0.59	0.19	0.22
123	0.27	0.15	0.58
124	0.53	0.29	0.18
125	0.79	0.21	0
126	0.16	0.38	0.46
127	0.47	0.37	0.16
128	0.33	0.01	0.66
129	0.02	0.51	0.47
130	0	0.95	0.05
131	0.07	0.12	0.81
132	0.74	0.02	0.24
133	0.07	0.41	0.52
134	0.65	0.06	0.29
135	0.13	0.03	0.84
136	0.4	0.55	0.05
137	0.22	0.35	0.43
138	0.05	0.22	0.73
139	0.39	0.54	0.07
140	0.22	0.02	0.76
141	0.05	0.38	0.57
142	0.67	0.25	0.08
143	0.1	0.59	0.31
144	0.1	0.31	0.59
145	0.1	0.56	0.34
146	0.79	0.02	0.19
147	0	0.39	0.61
148	0.48	0.3	0.22
149	0.15	0.45	0.4
150	0	0.05	0.95
151	0.39	0.35	0.26
152	0.72	0.12	0.16
153	0.05	0.08	0.87
154	0.24	0.07	0.69
155	0.35	0.19	0.46
156	0.1	0.3	0.6
157	0.03	0.31	0.66
158	0.03	0.62	0.35
159	0.92	0	0.08
160	0.62	0.09	0.29
161	0.31	0.54	0.15
162	0.65	0.15	0.2
163	0.41	0.27	0.32
164	0.12	0.26	0.62
165	0.53	0.27	0.2
166	0.22	0.42	0.36
167	0.49	0.32	0.19
168	0.37	0.56	0.07
169	0.39	0.04	0.57
170	0.23	0.01	0.76

	beta1	beta2	beta3
171	0.62	0.12	0.26
172	0.13	0.51	0.36
173	0.62	0.06	0.32
174	0.42	0.09	0.49
175	0.14	0.14	0.72
176	0.19	0.77	0.04
177	0.36	0.59	0.05
178	0.01	0.01	0.98
179	0.48	0.27	0.25
180	0.34	0.48	0.18
181	0.45	0.37	0.18
182	0.53	0.19	0.28
183	0.21	0.08	0.71
184	0.31	0.42	0.27
185	0.59	0.17	0.24
186	0.37	0.31	0.32
187	0.54	0.39	0.07
188	0.18	0.79	0.03
189	0.28	0.06	0.66
190	0.09	0.2	0.71
191	0.21	0.45	0.34
192	0.29	0.24	0.47
193	0.1	0.81	0.09
194	0.24	0.28	0.48
195	0.27	0.61	0.12

FIGURE 3.13: Randomly Selected Coefficients of Basic Kernels- $\beta_1, \beta_2, \beta_3$

So the first five SVM models with the New-Kernels for Sample SpamBase data-set are combination of first row $\beta_1, \beta_2, \beta_3$ and basic kernels as follows:

$$\begin{cases} K_1(x, x') = 0.29 * K_{Linear} + 0.33 * K_{RBF} + 0.38 * K_{POLY} \\ K_2(x, x') = 0.41 * K_{Linear} + 0.41 * K_{RBF} + 0.18 * K_{POLY} \\ K_3(x, x') = 0.89 * K_{Linear} + 0.04 * K_{RBF} + 0.07 * K_{POLY} \\ K_4(x, x') = 0.04 * K_{Linear} + 0.75 * K_{RBF} + 0.21 * K_{POLY} \\ K_5(x, x') = 0.55 * K_{Linear} + 0.45 * K_{RBF} + 0 * K_{POLY} \end{cases}$$

Where the parameters in New-Kernels are fixed as follows:

$C_{Linear} = 1, C_{RBF} = 1, \gamma = 1/[dim = 57]$, and degree=3, and only the corresponding $\beta_1, \beta_2, \beta_3$ are changing.

Following is the average results of applying 195 models with 10-fold cross validation, along with random coefficients in our convex linear combination on the sample SpamBase data-set.

	CER	FPR	FNR	ACCR	AUC
1	0.2	0.274074	0.139394	0.8	0.802253
2	0.196667	0.268657	0.138554	0.803333	0.801791
3	0.176667	0.253623	0.111111	0.823333	0.817951
4	0.2	0.270677	0.143713	0.8	0.800545
5	0.086667	0.079646	0.090909	0.913333	0.961079
6	0.166667	0.240876	0.104294	0.833333	0.825107
7	0.203333	0.276119	0.144578	0.796667	0.80193
8	0.2	0.270677	0.143713	0.8	0.800822
9	0.213333	0.297872	0.138365	0.786667	0.793296
10	0.203333	0.276119	0.144578	0.796667	0.801838
11	0.2	0.274074	0.139394	0.8	0.801745
12	0.19	0.268116	0.123457	0.81	0.812041
13	0.2	0.274074	0.139394	0.8	0.802068
14	0.203333	0.282609	0.135802	0.796667	0.800499
15	0.2	0.274074	0.139394	0.8	0.799021
16	0.21	0.292857	0.1375	0.79	0.79385
17	0.203333	0.276119	0.144578	0.796667	0.801884
18	0.196667	0.272059	0.134146	0.803333	0.801653
19	0.203333	0.282609	0.135802	0.796667	0.800406
20	0.203333	0.276119	0.144578	0.796667	0.801791
21	0.206667	0.278195	0.149701	0.793333	0.800683
22	0.196667	0.268657	0.138554	0.803333	0.802115
23	0.183333	0.257353	0.121951	0.816667	0.816012
24	0.2	0.270677	0.143713	0.8	0.800914
25	0.2	0.270677	0.143713	0.8	0.801468
26	0.163333	0.23913	0.098765	0.836667	0.839374
27	0.173333	0.251799	0.10559	0.826667	0.822014
28	0.196667	0.265152	0.142857	0.803333	0.800545
29	0.21	0.286765	0.146341	0.79	0.800729
30	0.196667	0.268657	0.138554	0.803333	0.801514
31	0.203333	0.282609	0.135802	0.796667	0.800268
32	0.203333	0.272727	0.14881	0.796667	0.801791
33	0.206667	0.284672	0.141104	0.793333	0.800037
34	0.183333	0.257353	0.121951	0.816667	0.835588
35	0.206667	0.284672	0.141104	0.793333	0.799621
36	0.18	0.251852	0.121212	0.82	0.81555
37	0.196667	0.268657	0.138554	0.803333	0.801422
38	0.18	0.251852	0.121212	0.82	0.836142
39	0.2	0.270677	0.143713	0.8	0.801514
40	0.203333	0.276119	0.144578	0.796667	0.801653
41	0.173333	0.248175	0.110429	0.826667	0.825246
42	0.2	0.270677	0.143713	0.8	0.800822
43	0.196667	0.265152	0.142857	0.803333	0.800499
44	0.16	0.22963	0.10303	0.84	0.839282
45	0.196667	0.265152	0.142857	0.803333	0.800683
46	0.2	0.270677	0.143713	0.8	0.800914
47	0.203333	0.279412	0.140244	0.796667	0.799252
48	0.2	0.270677	0.143713	0.8	0.801699
49	0.173333	0.244444	0.115152	0.826667	0.834388
50	0.2	0.274074	0.139394	0.8	0.801745
51	0.086667	0.094017	0.081967	0.913333	0.962464
52	0.19	0.264706	0.128049	0.81	0.804654
53	0.173333	0.248175	0.110429	0.826667	0.825985
54	0.196667	0.268657	0.138554	0.803333	0.802253
55	0.186667	0.262774	0.122699	0.813333	0.807747
56	0.176667	0.25	0.115854	0.823333	0.81495
57	0.196667	0.272059	0.134146	0.803333	0.801745
58	0.203333	0.276119	0.144578	0.796667	0.801561
59	0.2	0.270677	0.143713	0.8	0.80133
60	0.2	0.274074	0.139394	0.8	0.800776
61	0.176667	0.253623	0.111111	0.823333	0.819151
62	0.203333	0.276119	0.144578	0.796667	0.80096
63	0.2	0.274074	0.139394	0.8	0.802161
64	0.166667	0.240876	0.104294	0.833333	0.826677
65	0.19	0.268116	0.123457	0.81	0.811302
66	0.196667	0.272059	0.134146	0.803333	0.801745
67	0.196667	0.268657	0.138554	0.803333	0.801007
68	0.19	0.264706	0.128049	0.81	0.810102
69	0.213333	0.297872	0.138365	0.786667	0.793065
70	0.206667	0.278195	0.149701	0.793333	0.800914
71	0.2	0.270677	0.143713	0.8	0.80133
72	0.196667	0.265152	0.142857	0.803333	0.800683
73	0.086667	0.079646	0.090909	0.913333	0.961448
74	0.173333	0.248175	0.110429	0.826667	0.825338
75	0.206667	0.284672	0.141104	0.793333	0.799575
76	0.196667	0.265152	0.142857	0.803333	0.800683
77	0.09	0.094828	0.086957	0.91	0.962094
78	0.2	0.274074	0.139394	0.8	0.802299
79	0.203333	0.279412	0.140244	0.796667	0.799437
80	0.203333	0.276119	0.144578	0.796667	0.80193
81	0.18	0.255474	0.116564	0.82	0.812272
82	0.196667	0.265152	0.142857	0.803333	0.801145
83	0.19	0.264706	0.128049	0.81	0.8041
84	0.2	0.274074	0.139394	0.8	0.801284
85	0.203333	0.282609	0.135802	0.796667	0.800591
86	0.203333	0.282609	0.135802	0.796667	0.800406
87	0.196667	0.272059	0.134146	0.803333	0.802022
88	0.2	0.274074	0.139394	0.8	0.802022

	CER	FPR	FNR	ACCR	AUC
89	0.19	0.268116	0.123457	0.81	0.804331
90	0.19	0.268116	0.123457	0.81	0.809363
91	0.2	0.270677	0.143713	0.8	0.800729
92	0.203333	0.276119	0.144578	0.796667	0.801561
93	0.2	0.274074	0.139394	0.8	0.802022
94	0.2	0.274074	0.139394	0.8	0.801699
95	0.19	0.268116	0.123457	0.81	0.809086
96	0.196667	0.268657	0.138554	0.803333	0.801561
97	0.2	0.274074	0.139394	0.8	0.802576
98	0.196667	0.265152	0.142857	0.803333	0.801191
99	0.09	0.101695	0.082418	0.91	0.962325
100	0.203333	0.276119	0.144578	0.796667	0.801468
101	0.206667	0.284672	0.141104	0.793333	0.800406
102	0.176667	0.25	0.115854	0.823333	0.821875
103	0.196667	0.268657	0.138554	0.803333	0.80133
104	0.206667	0.284672	0.141104	0.793333	0.800868
105	0.173333	0.244444	0.115152	0.826667	0.833418
106	0.19	0.268116	0.123457	0.81	0.812641
107	0.19	0.264706	0.128049	0.81	0.808209
108	0.206667	0.284672	0.141104	0.793333	0.799898
109	0.196667	0.265152	0.142857	0.803333	0.801376
110	0.2	0.274074	0.139394	0.8	0.802299
111	0.09	0.080357	0.095745	0.91	0.960386
112	0.2	0.274074	0.139394	0.8	0.802438
113	0.203333	0.276119	0.144578	0.796667	0.80133
114	0.21	0.286765	0.146341	0.79	0.801099
115	0.183333	0.26087	0.117284	0.816667	0.812641
116	0.196667	0.265152	0.142857	0.803333	0.800914
117	0.2	0.270677	0.143713	0.8	0.801376
118	0.196667	0.272059	0.134146	0.803333	0.801653
119	0.2	0.274074	0.139394	0.8	0.802253
120	0.206667	0.284672	0.141104	0.793333	0.800314
121	0.16	0.233577	0.09816	0.84	0.842052
122	0.2	0.274074	0.139394	0.8	0.802068
123	0.21	0.286765	0.146341	0.79	0.800591
124	0.2	0.274074	0.139394	0.8	0.801838
125	0.09	0.072727	0.1	0.91	0.959786
126	0.203333	0.276119	0.144578	0.796667	0.801422
127	0.196667	0.268657	0.138554	0.803333	0.802161
128	0.206667	0.284672	0.141104	0.793333	0.800637
129	0.203333	0.276119	0.144578	0.796667	0.801145
130	0.18	0.258993	0.111801	0.82	0.813657
131	0.206667	0.284672	0.141104	0.793333	0.797775
132	0.2	0.274074	0.139394	0.8	0.801791
133	0.203333	0.276119	0.144578	0.796667	0.80096
134	0.196667	0.268657	0.138554	0.803333	0.801284
135	0.21	0.292857	0.1375	0.79	0.795466
136	0.173333	0.251799	0.10559	0.826667	0.824369
137	0.203333	0.276119	0.144578	0.796667	0.801653
138	0.203333	0.279412	0.140244	0.796667	0.799621
139	0.176667	0.253623	0.111111	0.823333	0.815504
140	0.2	0.274074	0.139394	0.8	0.799437
141	0.21	0.286765	0.146341	0.79	0.800637
142	0.183333	0.26087	0.117284	0.816667	0.814257
143	0.2	0.270677	0.143713	0.8	0.800729
144	0.21	0.286765	0.146341	0.79	0.800222
145	0.2	0.270677	0.143713	0.8	0.80133
146	0.196667	0.272059	0.134146	0.803333	0.801699
147	0.21	0.286765	0.146341	0.79	0.799714
148	0.2	0.274074	0.139394	0.8	0.801653
149	0.2	0.274074	0.139394	0.8	0.802207
150	0.213333	0.297872	0.138365	0.786667	0.792419
151	0.2	0.270677	0.143713	0.8	0.801053
152	0.196667	0.272059	0.134146	0.803333	0.801745
153	0.21	0.292857	0.1375	0.79	0.793435
154	0.206667	0.284672	0.141104	0.793333	0.800083
155	0.203333	0.276119	0.144578	0.796667	0.80193
156	0.21	0.286765	0.146341	0.79	0.799991
157	0.206667	0.284672	0.141104	0.793333	0.800314
158	0.2	0.270677	0.143713	0.8	0.801191
159	0.176667	0.25	0.115854	0.823333	0.814581
160	0.196667	0.268657	0.138554	0.803333	0.801376
161	0.203333	0.272727	0.14881	0.796667	0.802299
162	0.2	0.274074	0.139394	0.8	0.802022
163	0.2	0.274074	0.139394	0.8	0.800822
164	0.203333	0.282609	0.135802	0.796667	0.800083
165	0.2	0.274074	0.139394	0.8	0.802068
166	0.196667	0.268657	0.138554	0.803333	0.801838
167	0.196667	0.268657	0.138554	0.803333	0.802068
168	0.176667	0.253623	0.111111	0.823333	0.815365
169	0.21	0.286765	0.146341	0.79	0.801007
170	0.2	0.274074	0.139394	0.8	0.799391
171	0.196667	0.268657	0.138554	0.803333	0.801653
172	0.2	0.270677	0.143713	0.8	0.801561
173	0.2	0.274074	0.139394	0.8	0.801422
174	0.203333	0.276119	0.144578	0.796667	0.801653
175	0.206667	0.284672	0.141104	0.793333	0.799437
176	0.173333	0.251799	0.10559	0.826667	0.821044
177	0.176667	0.253623	0.111111	0.823333	0.823999
178	0.213333	0.297872	0.138365	0.786667	0.792096
179	0.196667	0.268657	0.138554	0.803333	0.801191
180	0.2	0.270677	0.143713	0.8	0.801653
181	0.196667	0.268657	0.138554	0.803333	0.80193
182	0.196667	0.268657	0.138554	0.803333	0.801237

	CER	FPR	FNR	ACCR	AUC
183	0.206667	0.284672	0.141104	0.793333	0.799668
184	0.196667	0.265152	0.142857	0.803333	0.800776
185	0.196667	0.268657	0.138554	0.803333	0.801561
186	0.2	0.274074	0.139394	0.8	0.800591
187	0.173333	0.248175	0.110429	0.826667	0.816427
188	0.176667	0.25	0.115854	0.823333	0.823168
189	0.206667	0.284672	0.141104	0.793333	0.800545
190	0.206667	0.284672	0.141104	0.793333	0.799529
191	0.2	0.270677	0.143713	0.8	0.801376
192	0.203333	0.276119	0.144578	0.796667	0.801561
193	0.176667	0.25	0.115854	0.823333	0.815319
194	0.203333	0.276119	0.144578	0.796667	0.801376
195	0.19	0.264706	0.128049	0.81	0.809548

FIGURE 3.14: Average CER, FPR, FNR, ACCR, AUC- Sample SpamBase, Customized NEW-Kernel

Based on the Figure 3.11, we can plot four graphs according to β_1, β_2 and four types of measures:

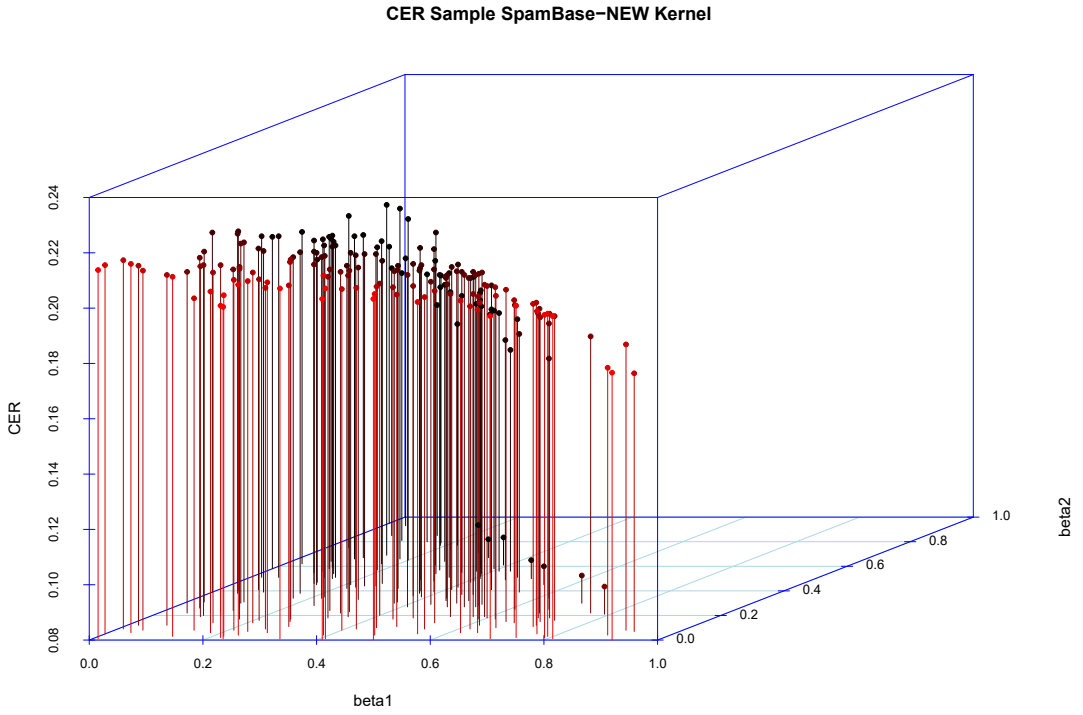


FIGURE 3.15: Average Classification Error rate, 10fold CV, Vs, β_1, β_2, Sample SpamBase

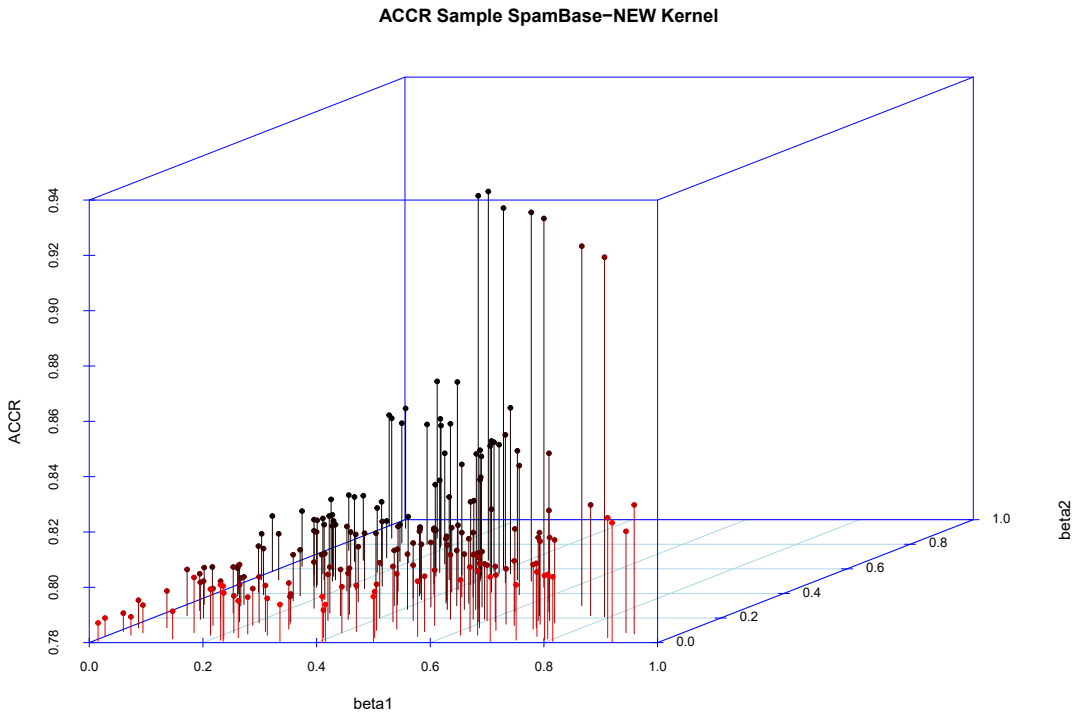


FIGURE 3.16: Average Accuracy rate, 10fold CV, Vs, β_1, β_2, Sample SpamBase

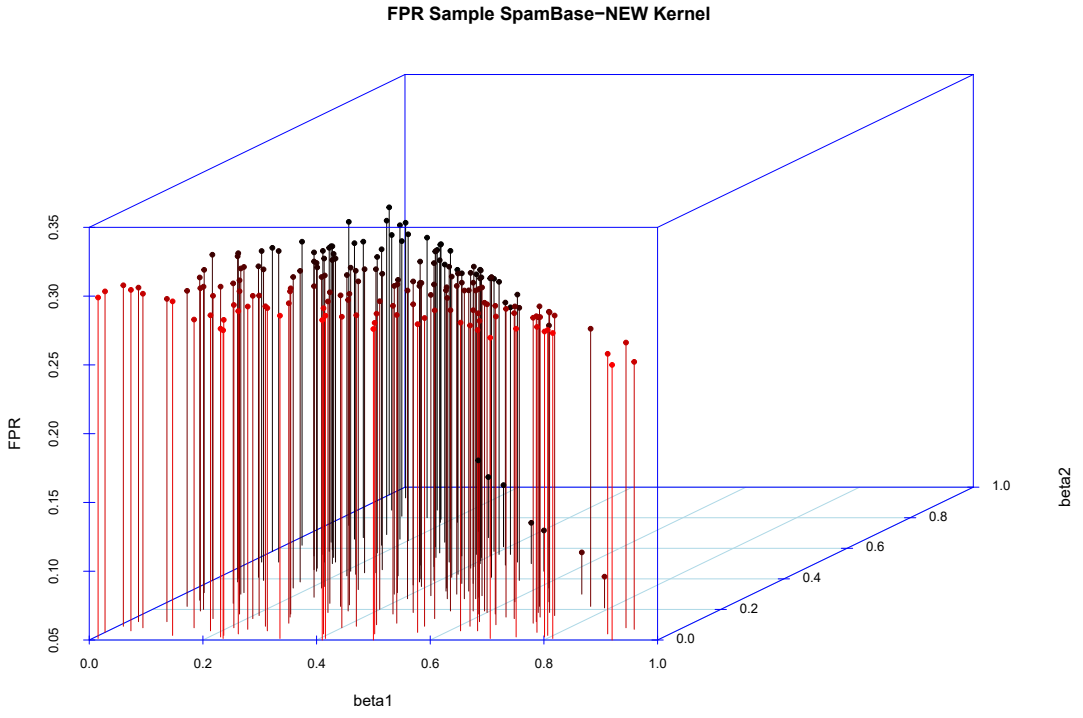


FIGURE 3.17: Average False Positive rate, 10fold CV, Vs, β_1, β_2, Sample SpamBase

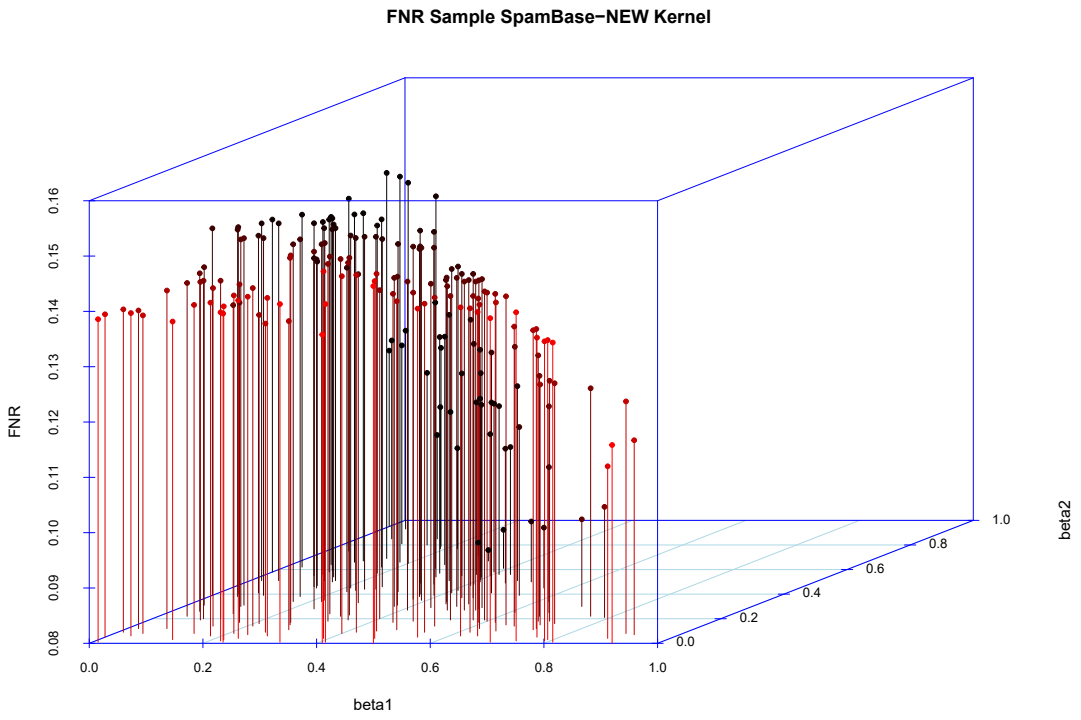


FIGURE 3.18: Average False Negative rate, 10fold CV, Vs, β_1, β_2, Sample SpamBase

Next, a summary of the average results for the top twenty models can be found in the Figure 3.19, 3.20:

ID	CER	FPR	FNR	ACCR	AUC
41	0.173333	0.248175	0.110429	0.826667	0.825246
49	0.173333	0.244444	0.115152	0.826667	0.834388
53	0.173333	0.248175	0.110429	0.826667	0.825985
74	0.173333	0.248175	0.110429	0.826667	0.825338
105	0.173333	0.244444	0.115152	0.826667	0.833418
136	0.173333	0.251799	0.10559	0.826667	0.824369
176	0.173333	0.251799	0.10559	0.826667	0.821044
187	0.173333	0.248175	0.110429	0.826667	0.816427
6	0.166667	0.240876	0.104294	0.833333	0.825107
64	0.166667	0.240876	0.104294	0.833333	0.826677
26	0.163333	0.23913	0.098765	0.836667	0.839374
44	0.16	0.22963	0.10303	0.84	0.839282
121	0.16	0.233577	0.09816	0.84	0.842052
77	0.09	0.094828	0.086957	0.91	0.962094
99	0.09	0.101695	0.082418	0.91	0.962325
111	0.09	0.080357	0.095745	0.91	0.960386
125	0.09	0.072727	0.1	0.91	0.959786
5	0.086667	0.079646	0.090909	0.913333	0.961079
51	0.086667	0.094017	0.081967	0.913333	0.962464
73	0.086667	0.079646	0.090909	0.913333	0.961448

FIGURE 3.19: TOP twenty Average Results, Based on ACCURACY rate, Sample SpamBase

ID	CER	FPR	FNR	ACCR	AUC
5	0.086667	0.079646	0.090909	0.913333	0.961079
51	0.086667	0.094017	0.081967	0.913333	0.962464
73	0.086667	0.079646	0.090909	0.913333	0.961448
77	0.09	0.094828	0.086957	0.91	0.962094
99	0.09	0.101695	0.082418	0.91	0.962325
111	0.09	0.080357	0.095745	0.91	0.960386
125	0.09	0.072727	0.1	0.91	0.959786
44	0.16	0.22963	0.10303	0.84	0.839282
121	0.16	0.233577	0.09816	0.84	0.842052
26	0.163333	0.23913	0.098765	0.836667	0.839374
6	0.166667	0.240876	0.104294	0.833333	0.825107
64	0.166667	0.240876	0.104294	0.833333	0.826677
27	0.173333	0.251799	0.10559	0.826667	0.822014
41	0.173333	0.248175	0.110429	0.826667	0.825246
49	0.173333	0.244444	0.115152	0.826667	0.834388
53	0.173333	0.248175	0.110429	0.826667	0.825985
74	0.173333	0.248175	0.110429	0.826667	0.825338
105	0.173333	0.244444	0.115152	0.826667	0.833418
136	0.173333	0.251799	0.10559	0.826667	0.824369
176	0.173333	0.251799	0.10559	0.826667	0.821044

FIGURE 3.20: TOP twenty Average Results, Based on Classification Error rate, Sample SpamBase

Finally, the best ten convex combinations of basic kernels corresponding to the above Figures are as follows, and clearly, models which have no contributions of Polynomial kernel on them have the best results :

$$\begin{cases} K_1(x,x') = 0.5 * K_{Linear} + 0.5 * K_{RBF} + 0 * K_{POLY} \\ K_2(x,x') = 0.33 * K_{Linear} + 0.67 * K_{RBF} + 0 * K_{POLY} \\ K_3(x,x') = 0.55 * K_{Linear} + 0.45 * K_{RBF} + 0 * K_{POLY} \\ K_4(x,x') = 0.79 * K_{Linear} + 0.21 * K_{RBF} + 0 * K_{POLY} \\ K_5(x,x') = 0.7 * K_{Linear} + 0.3 * K_{RBF} + 0 * K_{POLY} \\ K_6(x,x') = 0.29 * K_{Linear} + 0.71 * K_{RBF} + 0 * K_{POLY} \\ K_7(x,x') = 0.39 * K_{Linear} + 0.61 * K_{RBF} + 0 * K_{POLY} \\ K_8(x,x') = 0.22 * K_{Linear} + 0.77 * K_{RBF} + 0.01 * K_{POLY} \\ K_9(x,x') = 0.43 * K_{Linear} + 0.56 * K_{RBF} + 0.01 * K_{POLY} \\ K_{10}(x,x') = 0.14 * K_{Linear} + 0.85 * K_{RBF} + 0.01 * K_{POLY} \end{cases}$$

From the table (3.10), we can conclude that the model with the NEW-Kernel, compared to the model with TUNED-Polynomial Kernel improves 62% on CER, 15% on FPR, 65% on FNR, and 18% on ACCR. Also, compared to the model with TUNED-Linear Kernel it is 33% worse on FNR, but improves 40% FPR, and 10% CER. Further, comparing the NEW- Kernel performance with Logistic Regression model, it improves 14% in classification error rate and 49% the FPR, and 1% in ACCR, however, it was worse on FNR for 80%. Finally, comparing the accuracy

rate (91.3%) of this SVM with NEW-Kernel with the best classifier in our model which is the RBF kernel model with the highest ACCR(90%) among other tuned models, we can assure that the NEW-Kernel is an ideal kernel to be employed in the future SVMs model estimation.

	CER	FPR	FNR	ACR	AUC	#SV
Linear Kernel(C=0.1)	0.096	0.133	0.068	0.90	0.952	201
RBF Kernel(gamma=0.01, C=10)	0.1	0.123	0.083	0.9	0.964	223
Polynomial Kernel(d=2)	0.226	0.093	0.263	0.77	0.917	446
Logistic Regression	0.1	0.157	0.05	0.9	0.945	NA
Best NEW-Kernel	0.086	0.079	0.0909	0.913	0.961	511

TABLE 3.10: SpamBase Sample data-set, SVM with NEW-Kernel VS TUNED-basic kernels Results

3.3.2 Implementation of PCA on SVMs with New-Kernel Function

In this section, we are trying to combine the advantages of applying PCA and its corresponding NEW-Kernel on our data-set.

According to the results we produced for first 20 PCs which assured us about their ability and accuracy to detect and classify the data-set into two categories, we are going to detect the best convex combination of basic kernels on the data-set obtained from the first 20 PCs.

We repeat the whole process we did for sample SpamBase. The results of average measures for this experiment can be found in figure 3.21-3.25 :

	CER	FPR	FNR	ACCR	AUC
1	0.27	0.336066	0.224719	0.73	0.679025
2	0.236667	0.295082	0.196629	0.763333	0.724549
3	0.19	0.25	0.145349	0.81	0.780415
4	0.243333	0.306452	0.198864	0.756667	0.708204
5	0.096667	0.081818	0.105263	0.903333	0.959509
6	0.18	0.23622	0.138728	0.82	0.795558
7	0.273333	0.341463	0.225989	0.726667	0.683088
8	0.263333	0.327869	0.219101	0.736667	0.678702
9	0.283333	0.35	0.238889	0.716667	0.694353
10	0.273333	0.341463	0.225989	0.726667	0.679348
11	0.26	0.325203	0.214689	0.74	0.694261
12	0.203333	0.265625	0.156977	0.796667	0.76384
13	0.25	0.314516	0.204545	0.75	0.705527
14	0.28	0.34188	0.240437	0.72	0.693799
15	0.29	0.355932	0.247253	0.71	0.696339
16	0.29	0.355932	0.247253	0.71	0.695138
17	0.27	0.336066	0.224719	0.73	0.691999
18	0.24	0.300813	0.19774	0.76	0.72164
19	0.28	0.34188	0.240437	0.72	0.693568
20	0.273333	0.341463	0.225989	0.726667	0.684242
21	0.24	0.300813	0.19774	0.76	0.725795
22	0.233333	0.289256	0.195531	0.766667	0.731982
23	0.203333	0.269231	0.152941	0.796667	0.770534
24	0.26	0.325203	0.214689	0.74	0.68909
25	0.266667	0.333333	0.220339	0.733333	0.677686
26	0.156667	0.206349	0.12069	0.843333	0.843252
27	0.183333	0.238095	0.143678	0.816667	0.802207
28	0.26	0.325203	0.214689	0.74	0.687382
29	0.28	0.344538	0.237569	0.72	0.693661
30	0.25	0.314516	0.204545	0.75	0.703495
31	0.28	0.34188	0.240437	0.72	0.694769
32	0.233333	0.289256	0.195531	0.766667	0.743755
33	0.283333	0.347458	0.241758	0.716667	0.695046
34	0.173333	0.224	0.137143	0.826667	0.81495
35	0.286667	0.352941	0.243094	0.713333	0.695
36	0.2	0.263566	0.152047	0.8	0.771458
37	0.253333	0.32	0.205714	0.746667	0.696893
38	0.17	0.222222	0.132184	0.83	0.814488
39	0.27	0.33871	0.221591	0.73	0.675793
40	0.273333	0.341463	0.225989	0.726667	0.678609
41	0.176667	0.230159	0.137931	0.823333	0.816751
42	0.263333	0.327869	0.219101	0.736667	0.67944
43	0.256667	0.322581	0.210227	0.743333	0.692091
44	0.156667	0.206349	0.12069	0.843333	0.83822
45	0.243333	0.306452	0.198864	0.756667	0.70959
46	0.266667	0.333333	0.220339	0.733333	0.678471
47	0.29	0.355932	0.247253	0.71	0.695508
48	0.266667	0.333333	0.220339	0.733333	0.67667
49	0.163333	0.21875	0.122093	0.836667	0.814811
50	0.25	0.314516	0.204545	0.75	0.70488
51	0.086667	0.086957	0.086486	0.913333	0.957293
52	0.216667	0.270492	0.179775	0.783333	0.74828
53	0.18	0.232	0.142857	0.82	0.816474
54	0.266667	0.333333	0.220339	0.733333	0.677501
55	0.216667	0.277778	0.172414	0.783333	0.755621
56	0.2	0.259843	0.156069	0.8	0.770765
57	0.236667	0.295082	0.196629	0.763333	0.726673
58	0.286667	0.350427	0.245902	0.713333	0.693984
59	0.243333	0.306452	0.198864	0.756667	0.71633
60	0.27	0.33871	0.221591	0.73	0.678979
61	0.183333	0.242188	0.139535	0.816667	0.796205
62	0.286667	0.350427	0.245902	0.713333	0.691814
63	0.266667	0.333333	0.220339	0.733333	0.677963
64	0.18	0.23622	0.138728	0.82	0.798052
65	0.203333	0.265625	0.156977	0.796667	0.764301
66	0.236667	0.295082	0.196629	0.763333	0.723948
67	0.263333	0.327869	0.219101	0.736667	0.689967
68	0.206667	0.267717	0.16185	0.793333	0.761069
69	0.283333	0.35	0.238889	0.716667	0.694123
70	0.233333	0.289256	0.195531	0.766667	0.733783
71	0.27	0.33871	0.221591	0.73	0.678148
72	0.263333	0.327869	0.219101	0.736667	0.68475
73	0.086667	0.079646	0.090909	0.913333	0.958909
74	0.176667	0.230159	0.137931	0.823333	0.816612
75	0.286667	0.352941	0.243094	0.713333	0.695369
76	0.25	0.31746	0.201149	0.75	0.703403
77	0.086667	0.086957	0.086486	0.913333	0.957662
78	0.266667	0.330579	0.223464	0.733333	0.680364
79	0.29	0.355932	0.247253	0.71	0.695554
80	0.28	0.344538	0.237569	0.72	0.693707
81	0.196667	0.261538	0.147059	0.803333	0.768087
82	0.256667	0.322581	0.210227	0.743333	0.69246
83	0.216667	0.270492	0.179775	0.783333	0.75045
84	0.27	0.33871	0.221591	0.73	0.679487
85	0.283333	0.347458	0.241758	0.716667	0.693476
86	0.28	0.34188	0.240437	0.72	0.695138

	CER	FPR	FNR	ACCR	AUC
87	0.236667	0.295082	0.196629	0.763333	0.721917
88	0.236667	0.295082	0.196629	0.763333	0.721825
89	0.22	0.276423	0.180791	0.78	0.747588
90	0.196667	0.261538	0.147059	0.803333	0.764532
91	0.263333	0.327869	0.219101	0.736667	0.68258
92	0.283333	0.347458	0.241758	0.716667	0.692553
93	0.243333	0.306452	0.198864	0.756667	0.710698
94	0.27	0.33871	0.221591	0.73	0.677871
95	0.2	0.263566	0.152047	0.8	0.764532
96	0.263333	0.327869	0.219101	0.736667	0.694215
97	0.263333	0.327869	0.219101	0.736667	0.680318
98	0.256667	0.322581	0.210227	0.743333	0.693615
99	0.086667	0.086957	0.086486	0.913333	0.957385
100	0.28	0.344538	0.237569	0.72	0.690198
101	0.28	0.34188	0.240437	0.72	0.695184
102	0.173333	0.228346	0.132948	0.826667	0.818228
103	0.266667	0.330579	0.223464	0.733333	0.686181
104	0.283333	0.347458	0.241758	0.716667	0.693846
105	0.163333	0.214286	0.126437	0.836667	0.841082
106	0.203333	0.265625	0.156977	0.796667	0.763747
107	0.223333	0.278689	0.185393	0.776667	0.754559
108	0.283333	0.347458	0.241758	0.716667	0.694954
109	0.246667	0.312	0.2	0.753333	0.702941
110	0.266667	0.333333	0.220339	0.733333	0.681657
111	0.09	0.064815	0.104167	0.91	0.957985
112	0.27	0.336066	0.224719	0.73	0.680271
113	0.276667	0.344262	0.230337	0.723333	0.677917
114	0.283333	0.347458	0.241758	0.716667	0.693199
115	0.196667	0.257813	0.151163	0.803333	0.775013
116	0.256667	0.322581	0.210227	0.743333	0.69246
117	0.27	0.33871	0.221591	0.73	0.677871
118	0.24	0.300813	0.19774	0.76	0.723902
119	0.263333	0.327869	0.219101	0.736667	0.68198
120	0.28	0.34188	0.240437	0.72	0.695369
121	0.156667	0.206349	0.12069	0.843333	0.837989
122	0.25	0.314516	0.204545	0.75	0.705527
123	0.283333	0.347458	0.241758	0.716667	0.693476
124	0.236667	0.295082	0.196629	0.763333	0.72478
125	0.096667	0.074074	0.109375	0.903333	0.957108
126	0.273333	0.341463	0.225989	0.726667	0.682672
127	0.233333	0.289256	0.195531	0.766667	0.732121
128	0.28	0.34188	0.240437	0.72	0.695554
129	0.276667	0.344262	0.230337	0.723333	0.685027
130	0.186667	0.244094	0.144509	0.813333	0.797498
131	0.29	0.355932	0.247253	0.71	0.696477
132	0.26	0.325203	0.214689	0.74	0.6944
133	0.286667	0.350427	0.245902	0.713333	0.69126
134	0.266667	0.330579	0.223464	0.733333	0.686135
135	0.29	0.355932	0.247253	0.71	0.695785
136	0.18	0.23622	0.138728	0.82	0.795789
137	0.27	0.33871	0.221591	0.73	0.678148
138	0.29	0.355932	0.247253	0.71	0.694815
139	0.186667	0.244094	0.144509	0.813333	0.783323
140	0.29	0.355932	0.247253	0.71	0.695831
141	0.28	0.344538	0.237569	0.72	0.693061
142	0.196667	0.257813	0.151163	0.803333	0.773951
143	0.263333	0.327869	0.219101	0.736667	0.67981
144	0.283333	0.347458	0.241758	0.716667	0.693615
145	0.27	0.33871	0.221591	0.73	0.677824
146	0.24	0.300813	0.19774	0.76	0.721363
147	0.283333	0.347458	0.241758	0.716667	0.694123
148	0.25	0.314516	0.204545	0.75	0.704418
149	0.266667	0.333333	0.220339	0.733333	0.677963
150	0.283333	0.35	0.238889	0.716667	0.692922
151	0.26	0.325203	0.214689	0.74	0.690198
152	0.226667	0.280992	0.189944	0.773333	0.731151
153	0.29	0.355932	0.247253	0.71	0.694584
154	0.283333	0.347458	0.241758	0.716667	0.695508
155	0.273333	0.341463	0.225989	0.726667	0.684427
156	0.283333	0.347458	0.241758	0.716667	0.693661
157	0.28	0.34188	0.240437	0.72	0.695184
158	0.266667	0.333333	0.220339	0.733333	0.676901
159	0.196667	0.257813	0.151163	0.803333	0.773858
160	0.266667	0.330579	0.223464	0.733333	0.686043
161	0.23	0.286885	0.191011	0.77	0.742509
162	0.243333	0.306452	0.198864	0.756667	0.717669

	CER	FPR	FNR	ACCR	AUC
163	0.263333	0.327869	0.219101	0.736667	0.680133
164	0.28	0.34188	0.240437	0.72	0.693661
165	0.24	0.300813	0.19774	0.76	0.718039
166	0.266667	0.333333	0.220339	0.733333	0.67764
167	0.24	0.300813	0.19774	0.76	0.721778
168	0.186667	0.244094	0.144509	0.813333	0.785078
169	0.283333	0.347458	0.241758	0.716667	0.69403
170	0.29	0.355932	0.247253	0.71	0.695923
171	0.266667	0.330579	0.223464	0.733333	0.691537
172	0.266667	0.333333	0.220339	0.733333	0.677178
173	0.27	0.336066	0.224719	0.73	0.680502
174	0.273333	0.338843	0.22905	0.726667	0.692507
175	0.286667	0.352941	0.243094	0.713333	0.695138
176	0.183333	0.238095	0.143678	0.816667	0.803592
177	0.18	0.23622	0.138728	0.82	0.796066
178	0.28	0.349593	0.231638	0.72	0.69186
179	0.263333	0.327869	0.219101	0.736667	0.692876
180	0.24	0.300813	0.19774	0.76	0.723902
181	0.236667	0.295082	0.196629	0.763333	0.724687
182	0.266667	0.330579	0.223464	0.733333	0.688628
183	0.286667	0.352941	0.243094	0.713333	0.695046
184	0.26	0.325203	0.214689	0.74	0.688674
185	0.263333	0.327869	0.219101	0.736667	0.693938
186	0.263333	0.327869	0.219101	0.736667	0.679856
187	0.193333	0.255814	0.146199	0.806667	0.780507
188	0.176667	0.230159	0.137931	0.823333	0.816843
189	0.28	0.34188	0.240437	0.72	0.695646
190	0.286667	0.352941	0.243094	0.713333	0.695046
191	0.193333	0.259542	0.142012	0.806667	0.789556
192	0.206667	0.274809	0.153846	0.793333	0.777737
193	0.16	0.208	0.125714	0.84	0.825846
194	0.203333	0.272727	0.14881	0.796667	0.777275
195	0.166667	0.211382	0.135593	0.833333	0.809548

FIGURE 3.21: Average CER, FPR, FNR, ACCR, AUC- 20PCs,SpamBase, Customized NEW-Kernel

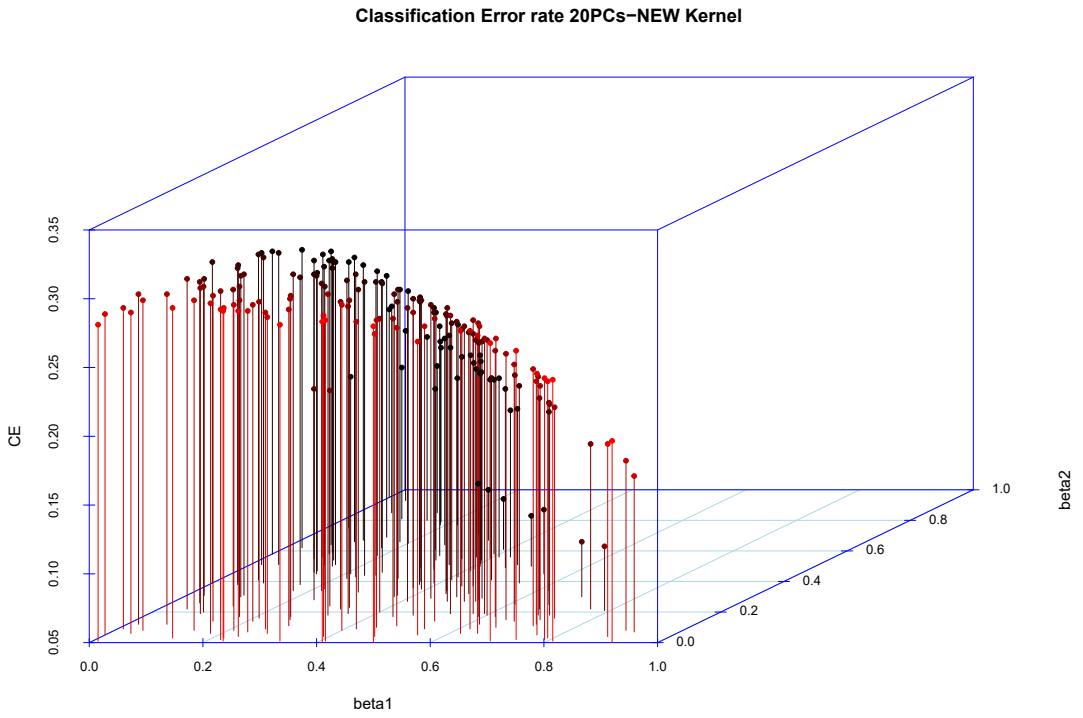


FIGURE 3.22: Average Classification Error rate, 10fold CV, Vs, β_1, β_2, 20PCs SpamBase

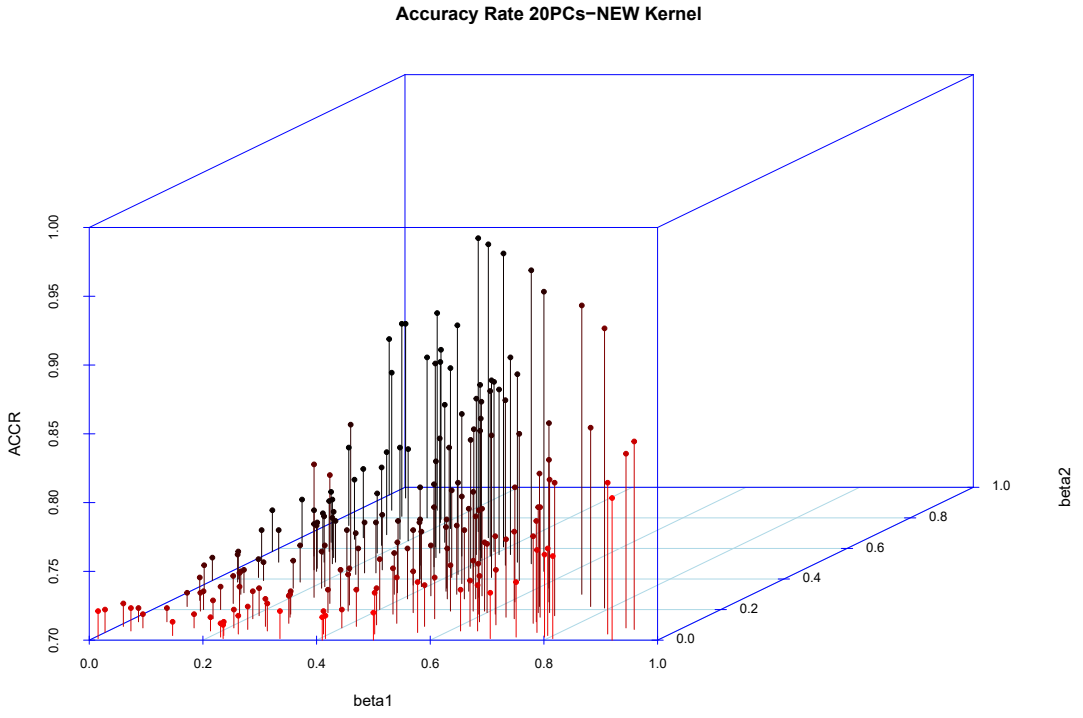


FIGURE 3.23: Average Accuracy rate, 10fold CV, Vs, β_1, β_2, 20PCs SpamBase

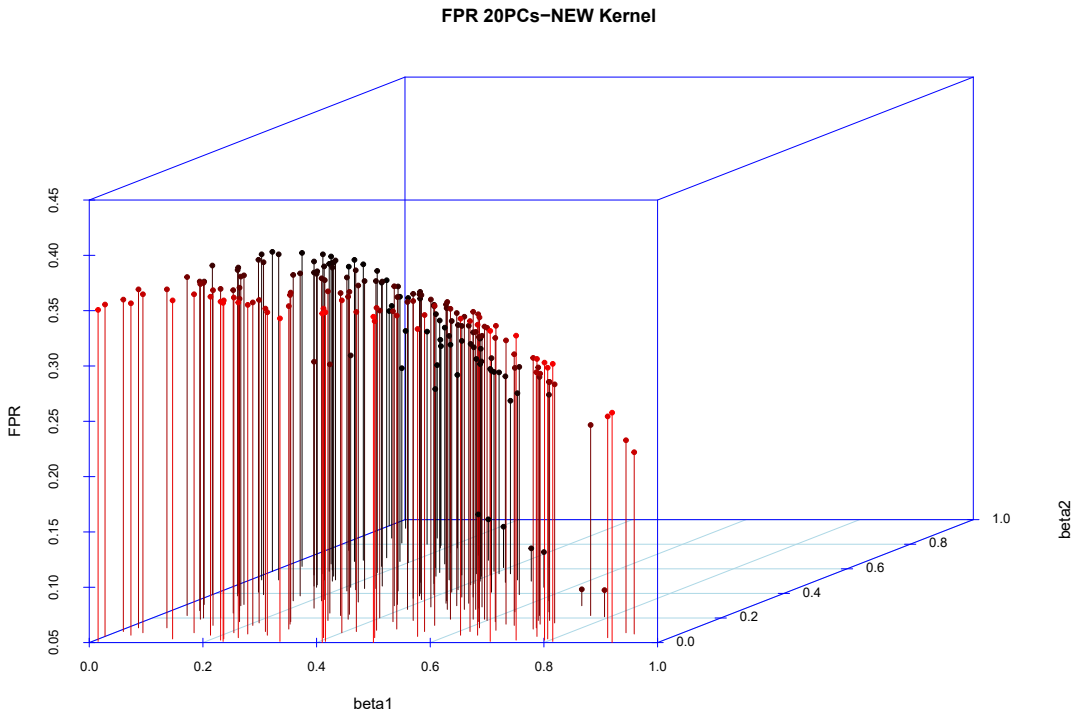


FIGURE 3.24: Average False Positive rate, 10fold CV, Vs, β_1, β_2, 20PCs Spam-Base

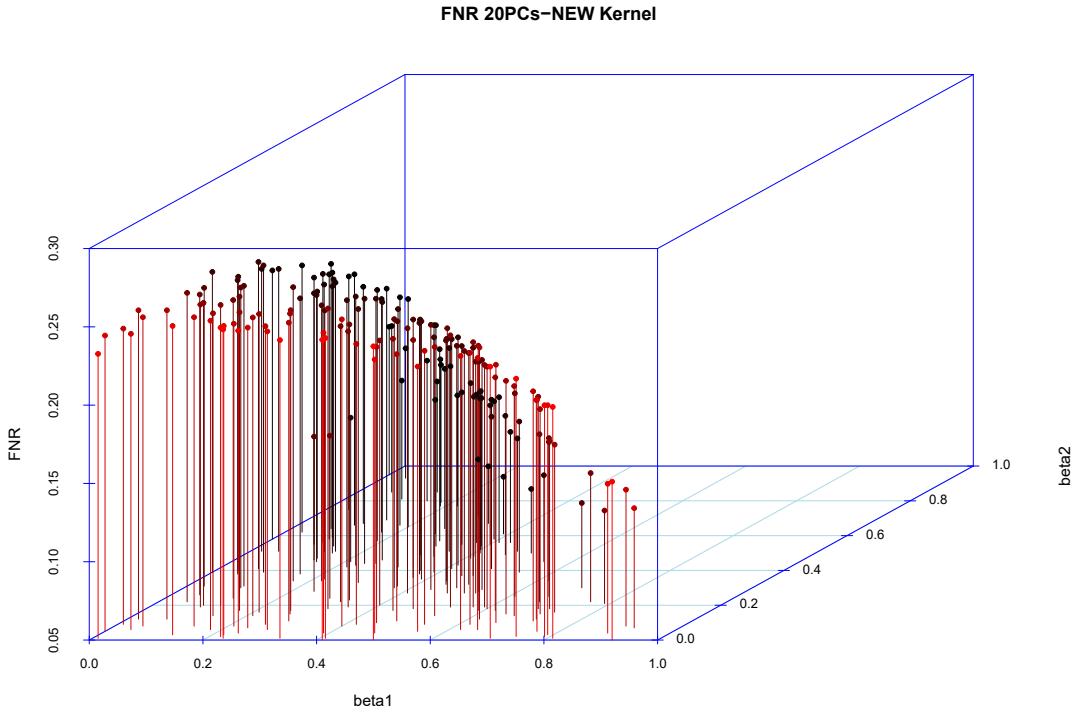


FIGURE 3.25: Average False Negative rate, 10fold CV, Vs, β_1, β_2, 20PCs Spam-Base

Here are the tables corresponding to top twenty models with the lowest Classification Errors and the highest Accuracy rates:

ID	CER	FPR	FNR	ACCR	AUC
177	0.18	0.23622	0.138728	0.82	0.796066
41	0.176667	0.230159	0.137931	0.823333	0.816751
74	0.176667	0.230159	0.137931	0.823333	0.816612
188	0.176667	0.230159	0.137931	0.823333	0.816843
34	0.173333	0.224	0.137143	0.826667	0.81495
102	0.173333	0.228346	0.132948	0.826667	0.818228
38	0.17	0.222222	0.132184	0.83	0.814488
195	0.166667	0.211382	0.135593	0.833333	0.809548
49	0.163333	0.21875	0.122093	0.836667	0.814811
105	0.163333	0.214286	0.126437	0.836667	0.841082
193	0.16	0.208	0.125714	0.84	0.825846
26	0.156667	0.206349	0.12069	0.843333	0.843252
44	0.156667	0.206349	0.12069	0.843333	0.83822
121	0.156667	0.206349	0.12069	0.843333	0.837989
5	0.096667	0.081818	0.105263	0.903333	0.959509
125	0.096667	0.074074	0.109375	0.903333	0.957108
111	0.09	0.064815	0.104167	0.91	0.957985
51	0.086667	0.086957	0.086486	0.913333	0.957293
73	0.086667	0.079646	0.090909	0.913333	0.958909
77	0.086667	0.086957	0.086486	0.913333	0.957662
99	0.086667	0.086957	0.086486	0.913333	0.957385

FIGURE 3.26: TOP twenty Average Results, Based on ACCURACY rate, 20 PCS-Sample SpamBase

ID	CER	FPR	FNR	ACCR	AUC
51	0.086667	0.086957	0.086486	0.913333	0.957293
73	0.086667	0.079646	0.090909	0.913333	0.958909
77	0.086667	0.086957	0.086486	0.913333	0.957662
99	0.086667	0.086957	0.086486	0.913333	0.957385
111	0.09	0.064815	0.104167	0.91	0.957985
5	0.096667	0.081818	0.105263	0.903333	0.959509
125	0.096667	0.074074	0.109375	0.903333	0.957108
26	0.156667	0.206349	0.12069	0.843333	0.843252
44	0.156667	0.206349	0.12069	0.843333	0.83822
121	0.156667	0.206349	0.12069	0.843333	0.837989
193	0.16	0.208	0.125714	0.84	0.825846
49	0.163333	0.21875	0.122093	0.836667	0.814811
105	0.163333	0.214286	0.126437	0.836667	0.841082
195	0.166667	0.211382	0.135593	0.833333	0.809548
38	0.17	0.222222	0.132184	0.83	0.814488
34	0.173333	0.224	0.137143	0.826667	0.81495
102	0.173333	0.228346	0.132948	0.826667	0.818228
41	0.176667	0.230159	0.137931	0.823333	0.816751
74	0.176667	0.230159	0.137931	0.823333	0.816612
188	0.176667	0.230159	0.137931	0.823333	0.816843

FIGURE 3.27: TOP twenty Average Results, Based on Classification Error rate, 20 PCs-Sample SpamBase

Thus, top ten convex combinations of basic kernels based on Figure 3.26, 3.27 are defined as follows:

$$
\begin{cases}
K_1(x,x') = 0.29 * K_{Linear} + 0.71 * K_{RBF} + 0 * K_{POLY} \\
K_2(x,x') = 0.39 * K_{Linear} + 0.61 * K_{RBF} + 0 * K_{POLY} \\
K_3(x,x') = 0.5 * K_{Linear} + 0.5 * K_{RBF} + 0 * K_{POLY} \\
K_4(x,x') = 0.33 * K_{Linear} + 0.67 * K_{RBF} + 0 * K_{POLY} \\
K_5(x,x') = 0.7 * K_{Linear} + 0.3 * K_{RBF} + 0 * K_{POLY} \\
K_6(x,x') = 0.79 * K_{Linear} + 0.21 * K_{RBF} + 0 * K_{POLY} \\
K_7(x,x') = 0.55 * K_{Linear} + 0.45 * K_{RBF} + 0 * K_{POLY} \\
K_8(x,x') = 0.22 * K_{Linear} + 0.77 * K_{RBF} + 0.01 * K_{POLY} \\
K_9(x,x') = 0.43 * K_{Linear} + 0.56 * K_{RBF} + 0.01 * K_{POLY} \\
K_{10}(x,x') = 0.14 * K_{Linear} + 0.85 * K_{RBF} + 0.01 * K_{POLY}
\end{cases}
$$

.

	CER	FPR	FNR	ACR	AUC	#SV
Linear Kernel(C=0.01)	0.15	0.06	0.18	0.85	0.95	600
RBF Kernel(gamma=0.1, C=1)	0.096	0.103	0.092	0.9	0.948	516
Polynomial Kernel(d=2)	0.26	0.1	0.315	0.72	0.88	528
Logistic Regression	0.096	0.11	0.08	0.89	0.949	NA
Best NEW-Kernel	0.086	0.086	0.086	0.913	0.9571	507

TABLE 3.11: 20 PCs-Sample SpamBase data-set, SVM with NEW-Kernel VS TUNED-basic kernels Results

Obviously, the NEW-Kernel performed a great job with the best accuracy rate among other models. In the following table we express the improvements we achieved from this model in the measures:

	Linear Kernel	RBF Kernel	Polynomial Kernel	Logistic regression
CER	42	10	67	10
FPR	-43	16	14	21
FNR	52	6	72	-7
ACR	7	1.5	26.8	2.5
AUC	.73	.94	8	0.84

TABLE 3.12: 20 PCs-Sample SpamBase data-set, Improvement percentage compared to The NEW-Kernel

The rest of this chapter will be assigned to the two more real-world data-sets that we already introduced in 3.2.

3.4 Applications of SVM on two more real-world data-sets

In this section we evaluate the SVMs with basic Kernels, and the NEW-kernel on two more real-world data-set. The SVMs models were compared with Logistics Regression model.

To determine the optimal parameters in each model, 10-fold cross-validation was performed on our data-sets. The experiment on Pima-Indian-Diabetes data-set shows no improvements after applying the SVM with NEW-Kernel, however, the experiment on Breast-Cancer-Wisconsin data-set has 50% reduction in number of

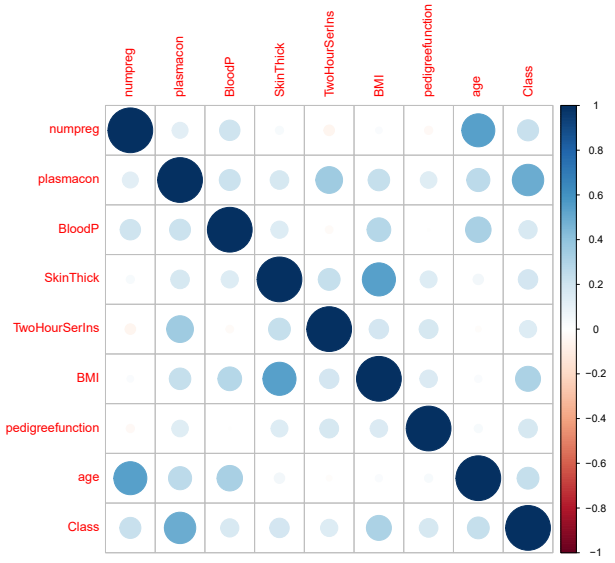


FIGURE 3.28: Correlation between features- Pima Indian Diabetes data-set

support vectors, and provide more accurate classification prediction after using the SVM with NEW-Kernel.

3.4.1 Pima Indian Diabetes data-set

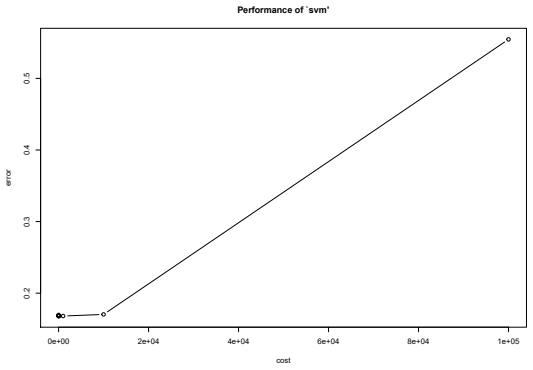
This data-set has missing data for some of the features that can lead to malfunction of the SVMs algorithm. We calculate the median values for the specific features and substitute that value wherever we have a missing datum.

	Class1- Diabetic	Class0- Non-diabetic
Frequency	268	500
Percentage	34.8	65.2

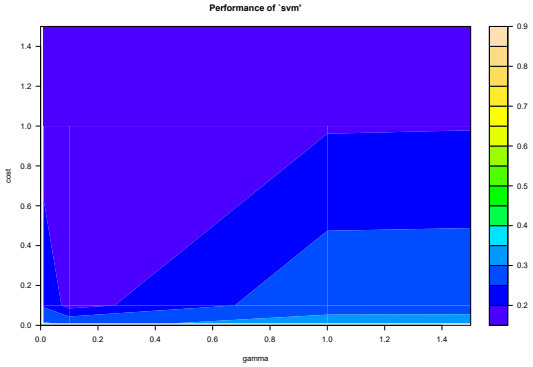
TABLE 3.13: The frequency table of Pima-Indian-Diabetes data-set

	CER	FPR	FNR	ACR	AUC	# SV
Linear Kernel	0.23	0.26	0.22	0.76	0.84	298
RBF Kernel	0.20	0.25	0.21	0.77	0.841	415
Polynomial Kernel	0.28	0.23	0.29	0.71	0.77	469
Logistic Regression	0.21	0.23	0.20	0.78	0.85	NA

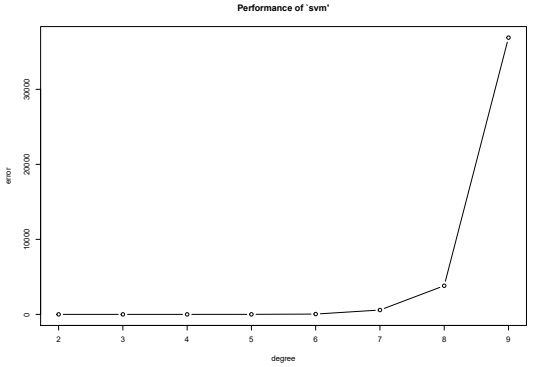
TABLE 3.14: Pima-Indian-Diabetes Sample data-set, basic kernels Average-Results



(a) Tuning the parameter Cost, Linear Kernel, Pima-Indian-Diabetes



(b) Tuning the parameter Cost,γ, RBF Kernel, Pima-Indian-Diabetes



(c) Tuning the parameter degree, Polynomial Kernel, Pima-Indian-Diabetes

FIGURE 3.29: Tuning the parameter $C_{Linear}, C_{RBF}, \gamma, d$, Pima-Indian-Diabetes

	CER	FPR	FNR	ACR	AUC	#SV
Linear Kernel(C=0.1)	0.21	0.25	0.22	0.77	0.85	308
RBF Kernel(gamma=0.1, C=1)	0.24	0.27	0.2	0.77	0.84	332
Polynomial Kernel(d=2)	0.33	0.39	0.33	0.66	0.7	376
Logistic Regression	0.21	0.23	0.20	0.78	0.85	NA

TABLE 3.15: Pima-Indian-Diabetes, TUNED-basic kernels Results

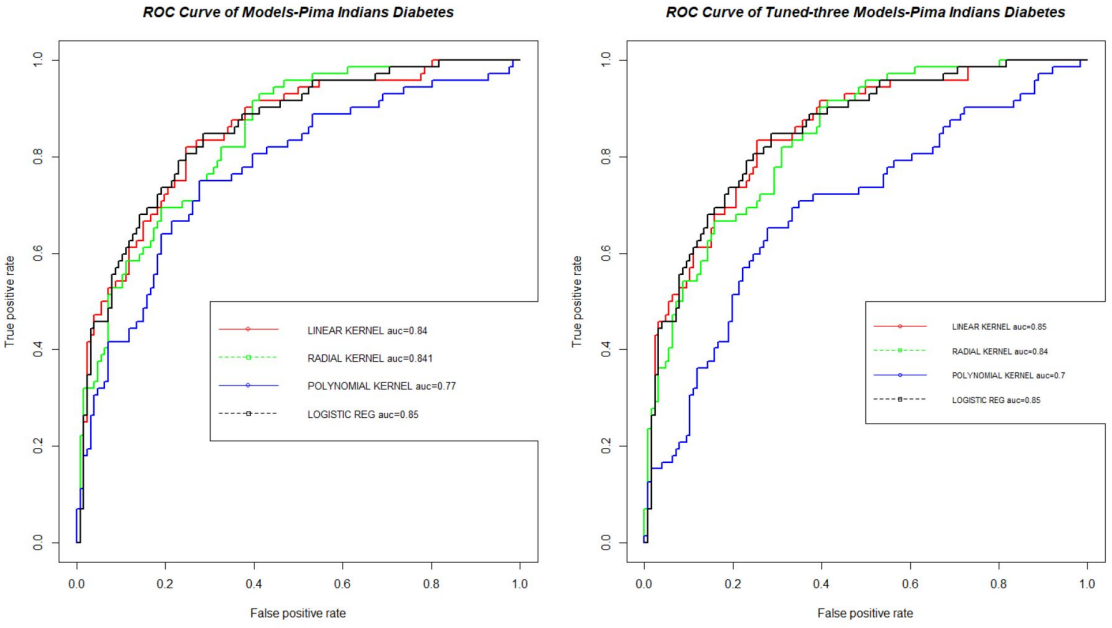
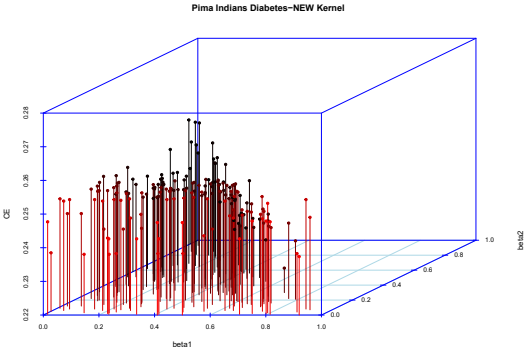
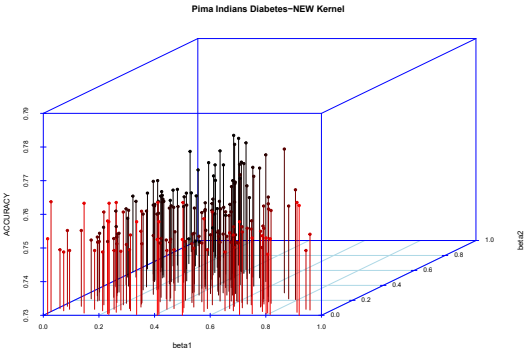


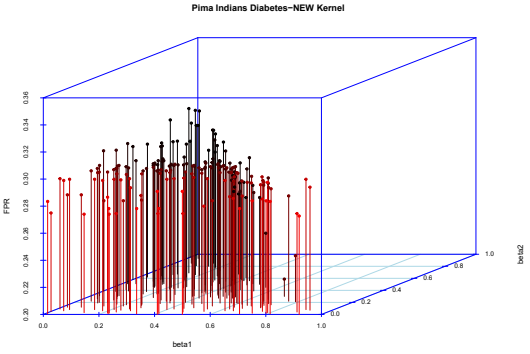
FIGURE 3.30: ROC curve, Basic Kernels vs Logistic Reg, Tuned Basic Kernels vs Logistic Reg, Pima-Indian-Diabetes



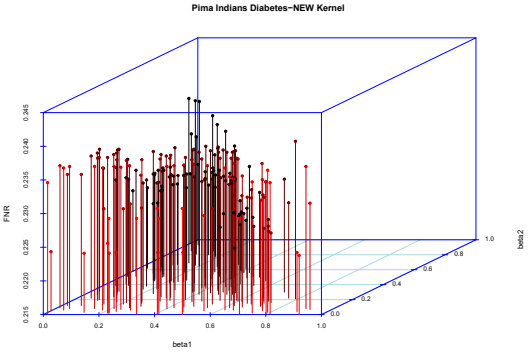
(a) Average Classification Error rate, 10fold CV, Vs, β_1, β_2, Pima-Indian-Diabetes



(b) Average Accuracy rate, 10fold CV, Vs, β_1, β_2, Pima-Indian-Diabetes



(c) Average False Positive rate, 10fold CV, Vs, β_1, β_2, Pima-Indian-Diabetes



(d) Average False Negative rate, 10fold CV, Vs, β_1, β_2, Pima-Indian-Diabetes

FIGURE 3.31: Average measure rates, 10fold CV, Vs, β_1, β_2, Pima-Indian-Diabetes

ID	CE	FPR	FNR	ACR	AUC
27	0.237374	0.280702	0.219858	0.762626	0.78362
49	0.237374	0.272727	0.223776	0.762626	0.78362
53	0.237374	0.272727	0.223776	0.762626	0.784832
73	0.237374	0.264151	0.227586	0.762626	0.847222
74	0.237374	0.272727	0.223776	0.762626	0.785273
75	0.237374	0.272727	0.223776	0.762626	0.768298
79	0.237374	0.272727	0.223776	0.762626	0.768519
90	0.237374	0.272727	0.223776	0.762626	0.77899
125	0.237374	0.234043	0.238411	0.762626	0.848876
135	0.237374	0.272727	0.223776	0.762626	0.767857
136	0.237374	0.272727	0.223776	0.762626	0.78362
150	0.237374	0.272727	0.223776	0.762626	0.76918
159	0.237374	0.272727	0.223776	0.762626	0.779431
169	0.237374	0.272727	0.223776	0.762626	0.76907
187	0.237374	0.272727	0.223776	0.762626	0.782077
5	0.232323	0.24	0.22973	0.767677	0.846892
51	0.232323	0.259259	0.222222	0.767677	0.843254
77	0.232323	0.259259	0.222222	0.767677	0.843474
99	0.232323	0.259259	0.222222	0.767677	0.842593
177	0.232323	0.267857	0.21831	0.767677	0.783069
111	0.227273	0.212766	0.231788	0.772727	0.848655

(a) TOP twenty Average Results, Based on ACCURACY rate, Pima-Indian-Diabetes

ID	CER	FPR	FNR	ACR	AUC
111	0.227273	0.212766	0.231788	0.772727	0.848655
5	0.232323	0.24	0.22973	0.767677	0.846892
51	0.232323	0.259259	0.222222	0.767677	0.843254
77	0.232323	0.259259	0.222222	0.767677	0.843474
99	0.232323	0.259259	0.222222	0.767677	0.842593
177	0.232323	0.267857	0.21831	0.767677	0.783069
3	0.237374	0.272727	0.223776	0.762626	0.779321
17	0.237374	0.272727	0.223776	0.762626	0.76918
27	0.237374	0.280702	0.219858	0.762626	0.78362
49	0.237374	0.272727	0.223776	0.762626	0.78362
53	0.237374	0.272727	0.223776	0.762626	0.784832
73	0.237374	0.264151	0.227586	0.762626	0.847222
74	0.237374	0.272727	0.223776	0.762626	0.785273
75	0.237374	0.272727	0.223776	0.762626	0.768298
79	0.237374	0.272727	0.223776	0.762626	0.768519
90	0.237374	0.272727	0.223776	0.762626	0.77899
125	0.237374	0.234043	0.238411	0.762626	0.848876
135	0.237374	0.272727	0.223776	0.762626	0.767857
136	0.237374	0.272727	0.223776	0.762626	0.78362
150	0.237374	0.272727	0.223776	0.762626	0.76918

(b) TOP twenty Average Results, Based on Classification Error rate, Pima-Indian-Diabetes

FIGURE 3.32: Average measure rates, 10fold CV, Vs, β_1, β_2, Pima-Indian-Diabetes

$$
\begin{cases}
K_1(x,x') = 0.7 * K_{Linear} + 0.3 * K_{RBF} + 0 * K_{POLY} \\
K_2(x,x') = 0.55 * K_{Linear} + 0.45 * K_{RBF} + 0 * K_{POLY} \\
K_3(x,x') = 0.39 * K_{Linear} + 0.61 * K_{RBF} + 0 * K_{POLY} \\
K_4(x,x') = 0.33 * K_{Linear} + 0.67 * K_{RBF} + 0 * K_{POLY} \\
K_5(x,x') = 0.29 * K_{Linear} + 0.71 * K_{RBF} + 0 * K_{POLY} \\
K_6(x,x') = 0.36 * K_{Linear} + 0.59 * K_{RBF} + 0.05 * K_{POLY} \\
K_7(x,x') = 0.79 * K_{Linear} + 0.21 * K_{RBF} + 0 * K_{POLY} \\
K_8(x,x') = 0.5 * K_{Linear} + 0.5 * K_{RBF} + 0 * K_{POLY} \\
K_9(x,x') = 0.39 * K_{Linear} + 0.58 * K_{RBF} + 0.03 * K_{POLY} \\
K_{10}(x,x') = 0.41 * K_{Linear} + 0.56 * K_{RBF} + 0.03 * K_{POLY}
\end{cases}
$$

	CER	FPR	FNR	ACR	AUC	#SV
Linear Kernel(C=0.1)	0.21	0.25	0.22	0.77	0.85	308
RBF Kernel(gamma=0.1, C=1)	0.24	0.27	0.2	0.77	0.84	332
Polynomial Kernel(d=2)	0.33	0.39	0.33	0.66	0.7	376
Logistic Regression	0.21	0.23	0.20	0.78	0.85	NA
Best NEW-Kernel	0.22	0.21	0.23	0.77	0.85	415

TABLE 3.16: Pima-Indian-Diabetes data-set, SVM with NEW-Kernel VS TUNED-basic kernels Results

3.4.2 Breast-Cancer-Wisconsin data-set

	Class1- Malignant	Class0- benign
Frequency	241	458
Percentage	34.4	65.6

TABLE 3.17: The frequency table of Breast-Cancer-Wisconsin data-set

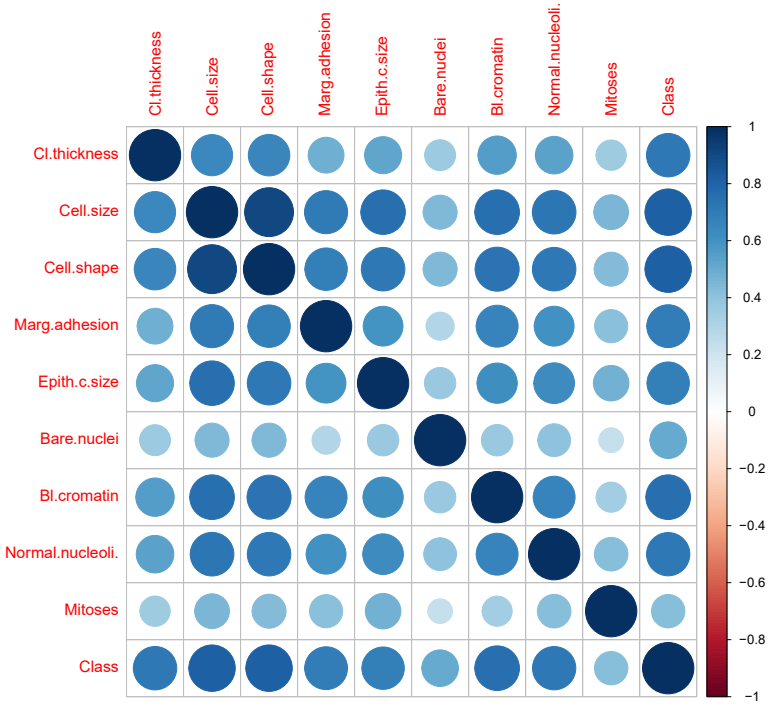
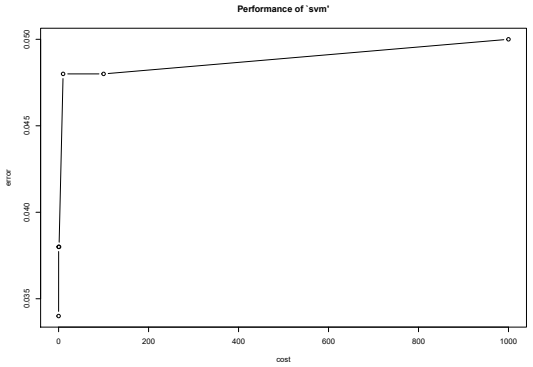


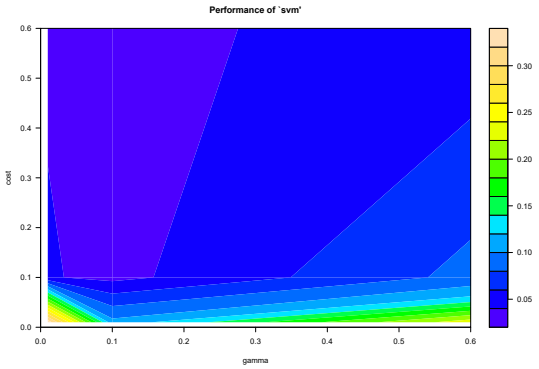
FIGURE 3.33: Correlation between features- Breast-Cancer-Wisconsin data-set

	CER	FPR	FNR	ACR	AUC	# SV
Linear Kernel	0.045	0.06	0.032	0.95	0.994	45
RBF Kernel	0.03	0.064	0.016	0.96	0.996	70
Polynomial Kernel	0.09	0	0.12	0.9	0.99	156
Logistic Regression	0.05	0.06	0.04	0.94	0.98	NA

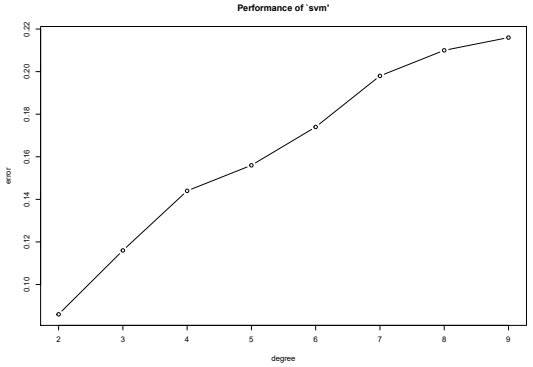
TABLE 3.18: Breast-Cancer-Wisconsin Sample data-set, basic kernels Average-Results



(a) Tuning the parameter Cost, Linear Kernel, Breast-Cancer-Wisconsin



(b) Tuning the parameter Cost,γ, RBF Kernel, Breast-Cancer-Wisconsin



(c) Tuning the parameter degree, Polynomial Kernel, Breast-Cancer-Wisconsin

FIGURE 3.34: Tuning the parameter $C_{Linear}, C_{RBF}, \gamma, d$, Breast-Cancer-Wisconsin

	CER	FPR	FNR	ACR	AUC	#SV
Linear Kernel(C=0.01)	0.04	0.054	0.032	0.96	0.996	107
RBF Kernel(gamma=0.1, C=0.1)	0.05	0.089	0.02	0.95	0.995	132
Polynomial Kernel(d=2)	0.065	0.03	0.08	0.93	0.993	161
Logistic Regression	0.05	0.06	0.04	0.94	0.98	NA

TABLE 3.19: Breast-Cancer-Wisconsin, TUNED-basic kernels Results

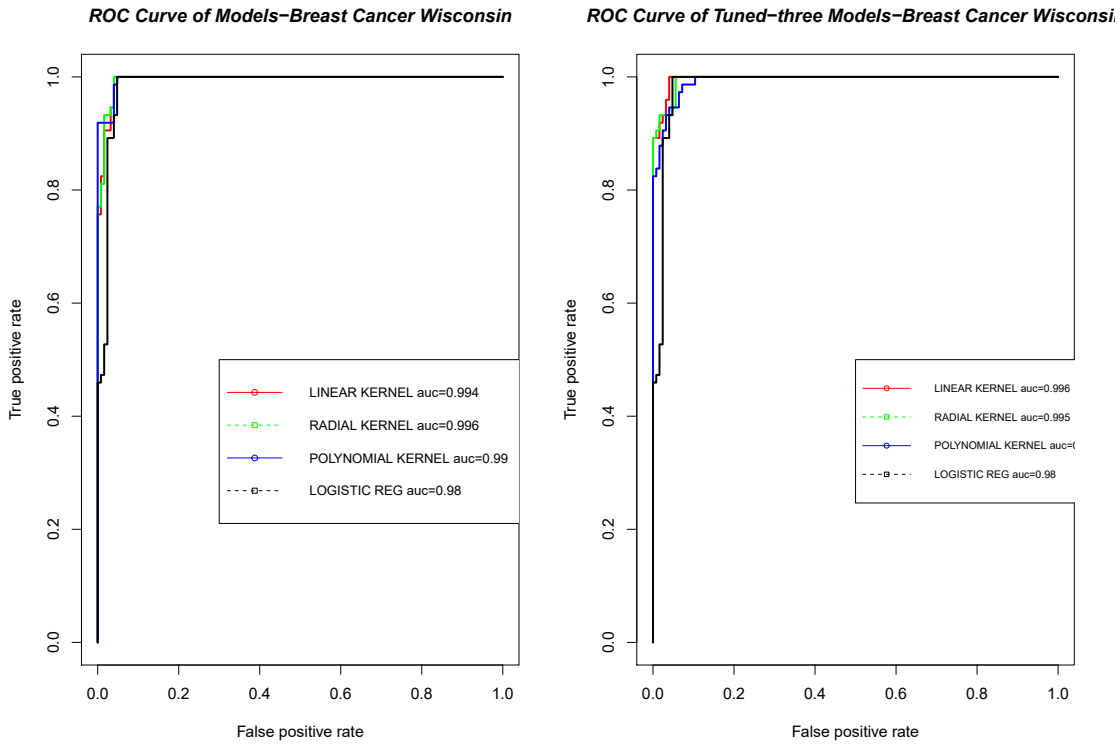
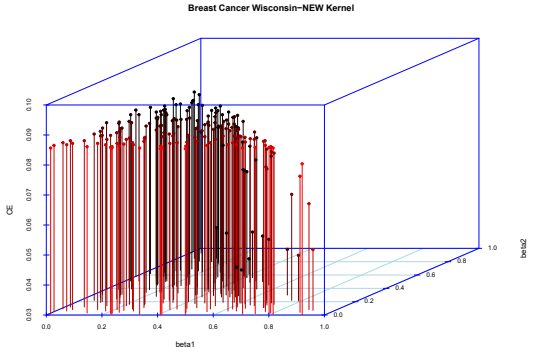
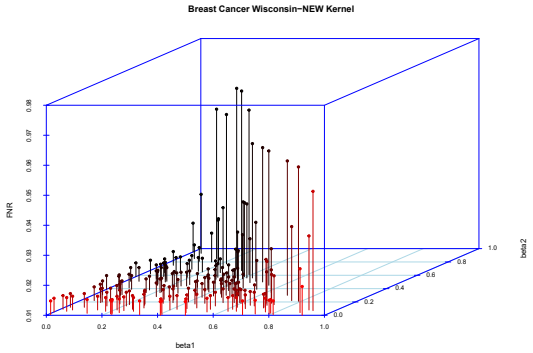


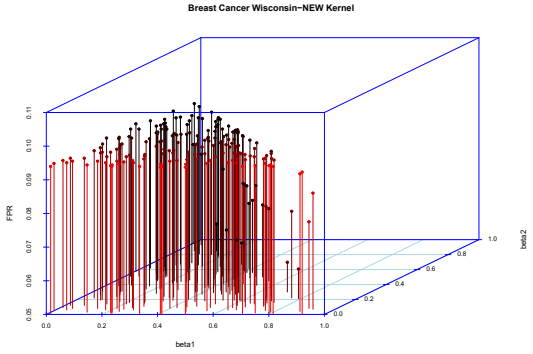
FIGURE 3.35: ROC curve, Basic Kernels vs Logistic Reg, Tuned Basic Kernels vs Logistic Reg, Breast-Cancer-Wisconsin



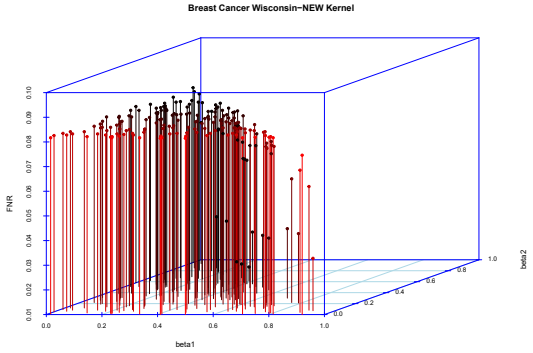
(a) Average Classification Error rate, 10fold CV, Vs, β_1, β_2, Breast-Cancer-Wisconsin



(b) Average Accuracy rate, 10fold CV, Vs, β_1, β_2, Breast-Cancer-Wisconsin



(c) Average False Positive rate, 10fold CV, Vs, β_1, β_2, Breast-Cancer-Wisconsin



(d) Average False Negative rate, 10fold CV, Vs, β_1, β_2, Breast-Cancer-Wisconsin

FIGURE 3.36: Average measure rates, 10fold CV, Vs, β_1, β_2, Breast-Cancer-Wisconsin

ID	CER	FPR	FNR	ACCR	AUC
177	0.075377	0.090909	0.067669	0.924623	0.965217
188	0.075377	0.090909	0.067669	0.924623	0.977926
27	0.070352	0.076923	0.067164	0.929648	0.969454
49	0.070352	0.076923	0.067164	0.929648	0.981494
102	0.070352	0.076923	0.067164	0.929648	0.978372
34	0.065327	0.075758	0.06015	0.934673	0.981048
38	0.065327	0.075758	0.06015	0.934673	0.981717
41	0.065327	0.075758	0.06015	0.934673	0.976923
53	0.065327	0.075758	0.06015	0.934673	0.977258
74	0.065327	0.075758	0.06015	0.934673	0.976923
105	0.050251	0.084507	0.03125	0.949749	0.984727
5	0.045226	0.071429	0.031008	0.954774	0.989632
44	0.045226	0.071429	0.031008	0.954774	0.986399
73	0.045226	0.071429	0.031008	0.954774	0.989409
111	0.045226	0.058824	0.038168	0.954774	0.989186
125	0.045226	0.058824	0.038168	0.954774	0.989075
26	0.040201	0.057971	0.030769	0.959799	0.987737
121	0.040201	0.057971	0.030769	0.959799	0.987179
77	0.035176	0.069444	0.015748	0.964824	0.989632
51	0.030151	0.056338	0.015625	0.969849	0.989521
99	0.030151	0.056338	0.015625	0.969849	0.989632

(a) TOP twenty Average Results, Based on ACCURACY rate, Breast-Cancer-Wisconsin

ID	CER	FPR	FNR	ACCR	AUC
51	0.030151	0.056338	0.015625	0.969849	0.989521
99	0.030151	0.056338	0.015625	0.969849	0.989632
77	0.035176	0.069444	0.015748	0.964824	0.989632
26	0.040201	0.057971	0.030769	0.959799	0.987737
121	0.040201	0.057971	0.030769	0.959799	0.987179
5	0.045226	0.071429	0.031008	0.954774	0.989632
44	0.045226	0.071429	0.031008	0.954774	0.986399
73	0.045226	0.071429	0.031008	0.954774	0.989409
111	0.045226	0.058824	0.038168	0.954774	0.989186
125	0.045226	0.058824	0.038168	0.954774	0.989075
105	0.050251	0.084507	0.03125	0.949749	0.984727
34	0.065327	0.075758	0.06015	0.934673	0.981048
38	0.065327	0.075758	0.06015	0.934673	0.981717
41	0.065327	0.075758	0.06015	0.934673	0.976923
53	0.065327	0.075758	0.06015	0.934673	0.977258
74	0.065327	0.075758	0.06015	0.934673	0.976923
27	0.070352	0.076923	0.067164	0.929648	0.969454
49	0.070352	0.076923	0.067164	0.929648	0.981494
102	0.070352	0.076923	0.067164	0.929648	0.978372
3	0.075377	0.090909	0.067669	0.924623	0.958194

(b) TOP twenty Average Results, Based on Classification Error rate, Breast-Cancer-Wisconsin

FIGURE 3.37: Average measure rates, 10fold CV, Vs, β_1, β_2, Breast-Cancer-Wisconsin

$$
\begin{cases}
K_1(x,x') = 0.29 * K_{Linear} + 0.71 * K_{RBF} + 0 * K_{POLY} \\
K_2(x,x') = 0.33 * K_{Linear} + 0.67 * K_{RBF} + 0 * K_{POLY} \\
K_3(x,x') = 0.39 * K_{Linear} + 0.61 * K_{RBF} + 0 * K_{POLY} \\
K_4(x,x') = 0.22 * K_{Linear} + 0.77 * K_{RBF} + 0.01 * K_{POLY} \\
K_5(x,x') = 0.14 * K_{Linear} + 0.85 * K_{RBF} + 0.01 * K_{POLY} \\
K_6(x,x') = 0.79 * K_{Linear} + 0.21 * K_{RBF} + 0 * K_{POLY} \\
K_7(x,x') = 0.7 * K_{Linear} + 0.3 * K_{RBF} + 0 * K_{POLY} \\
K_8(x,x') = 0.5 * K_{Linear} + 0.5 * K_{RBF} + 0 * K_{POLY} \\
K_9(x,x') = 0.43 * K_{Linear} + 0.56 * K_{RBF} + 0.01 * K_{POLY} \\
K_{10}(x,x') = 0.55 * K_{Linear} + 0.45 * K_{RBF} + 0 * K_{POLY}
\end{cases}
$$

	CER	FPR	FNR	ACR	AUC	#SV
Linear Kernel(C=0.01)	0.04	0.054	0.032	0.96	0.996	107
RBF Kernel(gamma=0.1, C=0.1)	0.05	0.089	0.02	0.95	0.995	132
Polynomial Kernel(d=2)	0.065	0.03	0.08	0.93	0.993	161
Logistic Regression	0.05	0.06	0.04	0.94	0.98	NA
Best NEW-Kernel	0.03	0.05	0.02	0.97	0.996	51

TABLE 3.20: Breast-Cancer-Wisconsin data-set, SVM with NEW-Kernel VS TUNED-basic kernels Results

The results demonstrate some improvements in the classification.

Chapter 4

Conclusion

In this thesis, we compared the performance accuracy of SVM model with basic kernels and the NEW-Kernel function. We have presented a NEW-Kernel function based on a convex combination of classical kernels.

Table 4.1 provides a brief overview of the four SVM implementations.

Average Measure Rates							
UCI-Data-set	Kernel	CER	FPR	FNR	ACR	AUC	$\#SV$
Sample SpamBase	Linear-K	0.096	0.133	0.068	0.90	0.952	201
	RBF-K	0.1	0.123	0.083	0.9	0.964	223
	Polynomial-K	0.226	0.093	0.263	0.77	0.917	446
	Logistic-R	0.1	0.157	0.05	0.9	0.945	NA
	Best NEW-K	0.086	0.079	0.0909	0.913	0.961	511
SpamBase 20PCA	Linear-K	0.15	0.06	0.18	0.85	0.95	600
	RBF-K	0.096	0.103	0.092	0.9	0.948	516
	Polynomial-K	0.26	0.1	0.315	0.72	0.88	528
	Logistic-R	0.096	0.11	0.08	0.89	0.949	NA
	Best NEW-K	0.086	0.086	0.086	0.913	0.9571	507
Pima Indian Diabetes	Linear-K	0.21	0.25	0.22	0.77	0.85	308
	RBF-Kernel	0.24	0.27	0.2	0.77	0.84	332
	Polynomial-K	0.33	0.39	0.33	0.66	0.7	376
	Logistic-R	0.21	0.23	0.20	0.78	0.85	NA
	Best NEW-K	0.22	0.21	0.23	0.77	0.85	415
Breast Cancer Wisconsin	Linear-K	0.04	0.054	0.032	0.96	0.996	107
	RBF-K	0.05	0.089	0.02	0.95	0.995	132
	Polynomial-K	0.065	0.03	0.08	0.93	0.993	161
	Logistic-R	0.05	0.06	0.04	0.94	0.98	NA
	Best NEW-K	0.03	0.05	0.02	0.97	0.996	51

TABLE 4.1: All data-sets- Average Measures Rates, SVM with NEW-Kernel VS TUNED-basic kernels Results

Our method can achieve effective prediction performance for kernel combination problem by increasing the accuracy rate and decreasing the average rate of the measure errors in most data-sets, while there were experiments that had no significant improvements in the measures.

Improving SVM algorithm is very important goal for objective classification, and development of new kernels is one of the ongoing research topics. So defining one appropriate Kernel function can always boost the model performance, and save the detection time dramatically.

After, Chapter 1, which provided an introduction on the concept of Support Vector Machines algorithm and the explanation of the other concepts like PCA, SVR, etc, in Chapter 2, we presented our SVM model with the NEW-Kernel function, and how we manipulated the coefficients of basic kernels. Finally, in Chapter 3, we reported the results of the several experiments on the UCI, Repository of Machine Learning. Specific real-world data-sets demonstrate the advantages of using combined-kernel instead of single kernels on both origin data and PCs data.

Table 4.1 can perfectly express the improvements in each of the data-sets measurements.

For future study, one can consider the convex combination of more basic kernels and KERNEL-Logistic Regression algorithm as a reference model for performance comparison.

References

Abe, S. (2005). *Support vector machines for pattern classification* (Vol. 2). Springer.

Abe, S. (2010). Two-class support vector machines. In *Support vector machines for pattern classification* (pp. 21–112). Springer.

Basak, D., Pal, S., & Patranabis, D. C. (2007). Support vector regression. *Neural Information Processing-Letters and Reviews*, *11*(10), 203–224.

Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent dirichlet allocation. *Journal of machine Learning research*, *3*(Jan), 993–1022.

Cristianini, N., & Shawe-Taylor, J. (2000). *An introduction to support vector machines and other kernel-based learning methods*. Cambridge university press.

Dheeru, D., & Karra Taniskidou, E. (2017). *UCI machine learning repository.* Retrieved from `http://archive.ics.uci.edu/ml`

Drucker, H., Burges, C. J., Kaufman, L., Smola, A. J., & Vapnik, V. (1997). Support vector regression machines. In *Advances in neural information processing systems* (pp. 155–161).

Drucker, H., Wu, D., & Vapnik, V. N. (1999). Support vector machines for spam categorization. *IEEE Transactions on Neural networks*, *10*(5), 1048–1054.

Fawcett, T. (2006). An introduction to roc analysis. *Pattern recognition letters*, *27*(8), 861–874.

Herbrich, R. (2016). *Learning kernel classifiers*. Mit Press.

Hsu, C.-W., Chang, C.-C., Lin, C.-J., et al. (2003). A practical guide to support vector classification.

Perez, M. A., & Nussbaum, M. A. (2003). Principal components analysis as an evaluation and classification tool for lower torso semg data. *Journal of biomechanics*, *36*(8), 1225–1229.

Smola, A. J., & Schölkopf, B. (2004). A tutorial on support vector regression. *Statistics and computing*, *14*(3), 199–222.

Sundaram, N. (2009). Support vector machine approximation using kernel pca. *Univ. California at Berkeley, Berkeley, CA, USA, Tech Rep. UCB/EECS-2009-94.[Online]. Available: http://www. eecs. berkeley. edu/Pubs/TechRpts/2009/EECS-2009-94. pdf*.

Vastrad, C. M., et al. (2013). Study of e-smooth support vector regression and comparison with e-support vector regression and potential support vector machines for prediction for the antitubercular activity of oxazolines and oxazoles derivatives. *arXiv preprint arXiv:1312.2867*.

Wu, C.-H., Ho, J.-M., & Lee, D.-T. (2004). Travel-time prediction with support vector regression. *IEEE transactions on intelligent transportation systems*, *5*(4), 276–281.

Yu, H., & Kim, S. (2012). Svm tutorialclassification, regression and ranking. In

Handbook of natural computing (pp. 479–506). Springer.

Appendix

.1 Bi-variate sample- R codes

```
set.seed(12345)
rbivariate <- function(mean.x = 1, sd.x=2, mean.y=1, sd.y=4, r=-0.6, iter=100) {
 z1 <- rnorm(iter)
 z2 <- rnorm(iter)
 x <- sqrt(1-r^2)*sd.x*z1 + r*sd.x*z2 + mean.x
 y <- sd.y*z2 + mean.y
 return(list(x,y))
}
data <- rbivariate(iter=200)
k1=exp(-abs(data[[1]]/data[[2]]))
z=runif(200, min(k1) , max(k1) )
ys=ifelse(k1>z,1,0)
X1=data[[1]]
X2=data[[2]]
dataanew=cbind(X1,X2)
dataanew<- data.frame(dataanew)
dataa <- data.frame(dataanew, y=as.factor(ys))
Freq= table(dataa$y)
plot(X2 ~ X1,  xlab = "X1",    ylab = "X2",    pch = c(16, 17)[as.numeric(y)],  main = "Bivariate Generated data",
   col = c("black","blue")[as.numeric(y)],    data = dataa)
plot(dataa[,-3],col=(ys+3)/2, pch=19, main="Raw- Generated Data")
N=dim(dataa)[1]
s=sample(1:N, 150)
train=dataa[s,] #make the training sample
test=dataa[-s,] #make the test sample
dim(test)
library(e1071)
library(caret)
set.seed(12345)
svm.modellinear <- svm(y ~., data=train, type='C-classification',
kernel="linear",Scale=TRUE,probability=TRUE,trControl=trainControl(method="cv",number=10, verboseItrer=TRUE,savePredictions =
TRUE))
plotlinear=plot(svm.modellinear,train)
svmmodel.predict<-predict(svm.modellinear,subset(test,select=-y),decision.values=TRUE)
svmmodel.probs<-attr(svmmodel.predict,"decision.values")
svmmodel.class<-predict(svm.modellinear,test,type="class")
```

```
svmmodel.labels<-test$y

#analyzing result

library(SDMTools)

svmmodel.confusion<-confusion.matrix(svmmodel.labels,svmmodel.class)

svmmodel.accuracy<-prop.correct(svmmodel.confusion)

CE=(svmmodel.confusion[1,2]+svmmodel.confusion[2,1])/(svmmodel.confusion[2,2]+svmmodel.confusion[1,2]+svmmodel.confusion[2,
1]+svmmodel.confusion[1,1])

FP=(svmmodel.confusion[2,1])/(svmmodel.confusion[2,1]+svmmodel.confusion[2,2])

FN=(svmmodel.confusion[1,2])/(svmmodel.confusion[1,1]+svmmodel.confusion[1,2])

#roc analysis for test data

svmmodel.prediction<-prediction(-svmmodel.probs,svmmodel.labels)

svmmodel.performanceL<-performance(svmmodel.prediction,"tpr","fpr")

svmmodel.auc<-performance(svmmodel.prediction,"auc")@y.values[[1]]

######################RADIAL TRAINING

svmmodelradial<-svm(y~., data=train, method="C-classification", kernel="radial",cross=10, scale=TRUE,probability=TRUE)

plotradial=plot(svmmodelradial,train)

svmmodel.predict<-predict(svmmodelradial,subset(test,select=-y),decision.values=TRUE)

svmmodel.probs<-attr(svmmodel.predict,"decision.values")

svmmodel.class<-predict(svmmodelradial,test,type="class")

svmmodel.labels<-test$y

#analyzing result

library(SDMTools)

svmmodel.confusion<-confusion.matrix(svmmodel.labels,svmmodel.class)

svmmodel.accuracy<-prop.correct(svmmodel.confusion)

#roc analysis for test data

svmmodel.prediction<-prediction(-svmmodel.probs,svmmodel.labels)

svmmodel.performanceR<-performance(svmmodel.prediction,"tpr","fpr")

svmmodel.auc<-performance(svmmodel.prediction,"auc")@y.values[[1]]

CE=(svmmodel.confusion[1,2]+svmmodel.confusion[2,1])/(svmmodel.confusion[2,2]+svmmodel.confusion[1,2]+svmmodel.confusion[2,
1]+svmmodel.confusion[1,1])

FP=(svmmodel.confusion[2,1])/(svmmodel.confusion[2,1]+svmmodel.confusion[2,2])

FN=(svmmodel.confusion[1,2])/(svmmodel.confusion[1,1]+svmmodel.confusion[1,2])

######################POLYNOMIAL TRAINING

svmmodelpoly<-svm(y~., data=train, method="C-classification", kernel="polynomial",cross=10, scale=TRUE,probability=TRUE)

plotpoly=plot(svmmodelpoly,train)

#predicting the test data
```

```
svmmodel.predict<-predict(svmmodelpoly,subset(test,select=-y),decision.values=TRUE)

svmmodel.probs<-attr(svmmodel.predict,"decision.values")

svmmodel.class<-predict(svmmodelpoly,test,type="class")

svmmodel.labels<-test$y

#analyzing result

svmmodel.confusion<-confusion.matrix(svmmodel.labels,svmmodel.class)

svmmodel.accuracy<-prop.correct(svmmodel.confusion)

#roc analysis for test data

svmmodel.prediction<-prediction(-svmmodel.probs,svmmodel.labels)

svmmodel.performanceP<-performance(svmmodel.prediction,"tpr","fpr")

svmmodel.auc<-performance(svmmodel.prediction,"auc")@y.values[[1]]

CE=(svmmodel.confusion[1,2]+svmmodel.confusion[2,1])/(svmmodel.confusion[2,2]+svmmodel.confusion[1,2]+svmmodel.confusion[2,
1]+svmmodel.confusion[1,1])

FP=(svmmodel.confusion[2,1])/(svmmodel.confusion[2,1]+svmmodel.confusion[2,2])

FN=(svmmodel.confusion[1,2])/(svmmodel.confusion[1,1]+svmmodel.confusion[1,2])

####################LOGISTIC REGRESSION TRAINING

logModel <- glm(y ~.,family=binomial(link='logit'),data=train)

plotlogistic=plot(logModel,train)

results <- predict(logModel,newdata=test[,-3],type='response')

testPredictions <- ifelse(results > 0.5,1,0) # get the right predictions

svmmodel.confusion=table(testPredictions, test[,3])

varImp(logModel)

misClasificError <- mean(testPredictions != test[,3])

print(paste('Accuracy',1-misClasificError))

library(ROCR)

pr <- prediction(results, test[,3])

prf <- performance(pr, measure = "tpr", x.measure = "fpr")

auc <- performance(pr, measure = "auc")

auc <- auc@y.values[[1]]

CE=(svmmodel.confusion[1,2]+svmmodel.confusion[2,1])/(svmmodel.confusion[2,2]+svmmodel.confusion[1,2]+svmmodel.confusion[2,
1]+svmmodel.confusion[1,1])

FP=(svmmodel.confusion[2,1])/(svmmodel.confusion[2,1]+svmmodel.confusion[2,2])

FN=(svmmodel.confusion[1,2])/(svmmodel.confusion[1,1]+svmmodel.confusion[1,2])

#################### TUNED LINEAR

tunelinearr=tune.svm( y~. ,data=train,kernel="linear",cost =10^(-2:3) )

par(mfrow = c(1, 1))
```

```
plot(tunelinearr)
install.packages("zoom")
library(zoom)
zoomplot.zoom(fact=1.1,x=0,y=0)
svm.model1<- svm(y ~ ., data=train, kernel="linear",type='C-classification', cost=10,scale=TRUE,probability=TRUE)
plottunedlinear=plot(svm.model1,train)
svmmodel.predict<-predict(svm.model1,subset(test,select=-y),decision.values=TRUE)
svmmodel.probs<-attr(svmmodel.predict,"decision.values")
svmmodel.class<-predict(svm.model1,test,type="class")
svmmodel.labels<-test$y
library(SDMTools)
svmmodel.confusion<-confusion.matrix(svmmodel.labels,svmmodel.class)
svmmodel.accuracy<-prop.correct(svmmodel.confusion)
#roc analysis for test data
svmmodel.prediction<-prediction(-svmmodel.probs,svmmodel.labels)
svmmodel.performanceLT<-performance(svmmodel.prediction,"tpr","fpr")
svmmodel.auc<-performance(svmmodel.prediction,"auc")@y.values[[1]]
CE=(svmmodel.confusion[1,2]+svmmodel.confusion[2,1])/(svmmodel.confusion[2,2]+svmmodel.confusion[1,2]+svmmodel.confusion[2,
1]+svmmodel.confusion[1,1])
FP=(svmmodel.confusion[2,1])/(svmmodel.confusion[2,1]+svmmodel.confusion[2,2])
FN=(svmmodel.confusion[1,2])/(svmmodel.confusion[1,1]+svmmodel.confusion[1,2])
###################### TUNED RADIAL
par(mfrow = c(1, 1))
tuneradiaal=tune.svm( y~. ,data=train, kernel="radial",cost = 10^(-2:3),gamma =10^(-2:3) )
plot(tuneradiaal,type = c("contour", "perspective"),  theta = 120,color.palette = topo.colors,     nlevels = 20,xlim=c(0,0.5))
svm.model2<- svm(y ~ ., data=train,type='C-classification', kernel="radial", cost=100,gamma=0.1,scale=TRUE,probability=TRUE)
summary(svm.model2)
plottunedradial=plot(svm.model2,train)
svmmodel.predict<-predict(svm.model2,subset(test,select=-y),decision.values=TRUE)
svmmodel.probs<-attr(svmmodel.predict,"decision.values")
svmmodel.class<-predict(svm.model2,test,type="class")
svmmodel.labels<-test$y
library(SDMTools)
svmmodel.confusion<-confusion.matrix(svmmodel.labels,svmmodel.class)
svmmodel.accuracy<-prop.correct(svmmodel.confusion)
```

```
#roc analysis for test data
svmmodel.prediction<-prediction(-svmmodel.probs,svmmodel.labels)
svmmodel.performanceRT<-performance(svmmodel.prediction,"tpr","fpr")
svmmodel.auc<-performance(svmmodel.prediction,"auc")@y.values[[1]]
CE=(svmmodel.confusion[1,2]+svmmodel.confusion[2,1])/(svmmodel.confusion[2,2]+svmmodel.confusion[1,2]+svmmodel.confusion[2,
1]+svmmodel.confusion[1,1])
FP=(svmmodel.confusion[2,1])/(svmmodel.confusion[2,1]+svmmodel.confusion[2,2])
FN=(svmmodel.confusion[1,2])/(svmmodel.confusion[1,1]+svmmodel.confusion[1,2])
###################### TUNED POLYNOMIAL
tunepoly=tune.svm( y~. ,data=train, kernel="polynomial",degree = c(2,3,4,5,6,7,8,9,10))
svm.model<- svm(y ~ ., data=train,type='C-classification', kernel="polynomial",degree=2,scale=TRUE,probability=TRUE)
plottunedpoly=plot(svm.model,train)
svmmodel.predict<-predict(svm.model,subset(test,select=-y),decision.values=TRUE)
svmmodel.probs<-attr(svmmodel.predict,"decision.values")
svmmodel.class<-predict(svm.model,test,type="class")
svmmodel.labels<-test$y
library(SDMTools)
svmmodel.confusion<-confusion.matrix(svmmodel.labels,svmmodel.class)
svmmodel.accuracy<-prop.correct(svmmodel.confusion)
#roc analysis for test data
svmmodel.prediction<-prediction(-svmmodel.probs,svmmodel.labels)
svmmodel.performancePT<-performance(svmmodel.prediction,"tpr","fpr")
svmmodel.auc<-performance(svmmodel.prediction,"auc")@y.values[[1]]
CE=(svmmodel.confusion[1,2]+svmmodel.confusion[2,1])/(svmmodel.confusion[2,2]+svmmodel.confusion[1,2]+svmmodel.confusion[2,
1]+svmmodel.confusion[1,1])
FP=(svmmodel.confusion[2,1])/(svmmodel.confusion[2,1]+svmmodel.confusion[2,2])
FN=(svmmodel.confusion[1,2])/(svmmodel.confusion[1,1]+svmmodel.confusion[1,2])
####################PLOT TRAINING & TUNED TRAINING
par(mfrow = c(1, 2))
plot( svmmodel.performanceL, col="red",lwd=2)
plot(svmmodel.performanceR, add = TRUE, col="green",lwd=2)
plot(svmmodel.performanceP, add = TRUE, col="blue",lwd=2)
plot( prf, add = TRUE,col="black",lwd=2)
legend(0.3, plot_range[2], c("LINEAR KERNEL auc=0.755","RADIAL KERNEL auc=0.855","POLYNOMIAL KERNEL auc=0.703","LOGISTIC REG
auc=0.764"), cex=0.8,
    col=c("red","green","blue","black"), pch=21:22, lty=1:2)
```

```
title(main="ROC Curve of Models-Bivariate Generated data", font.main=4)

plot( svmmodel.performanceLT, col="red",lwd=2)

plot(svmmodel.performanceRT, add = TRUE, col="green",lwd=2)

plot(svmmodel.performancePT, add = TRUE, col="blue",lwd=2)

plot( prf,add=TRUE, col="black",lwd=2)

legend(0.5, plot_range[2], c("LINEAR KERNEL auc=0.759","RADIAL KERNEL auc=0.864","POLYNOMIAL KERNEL auc=0.65","LOGISTIC REG
auc=0.764"), cex=0.7,

    col=c("red","green","blue","black"), pch=21:22, lty=1:2)

title(main="ROC Curve of Tuned-three Models-Bivariate Generated data", font.main=4)
```

.2 SpamBase data-set- R codes

```
#SPAMBASE
svm#############################################################################################################
set.seed(123)
dataa= read.table("C:\\Users\\FARNOOSH\\Desktop\\kernels\\kerneldatasets\\spambase.csv",header=FALSE, sep=",")
colnames(dataa) <-
c("word_freq_make","word_freq_address","word_freq_all","word_freq_3d","word_freq_our","word_freq_over","word_freq_remove",
"word_freq_internet","word_freq_order","word_freq_mail","word_freq_receive","word_freq_will","word_freq_people","word_freq_r
eport","word_freq_addresses","word_freq_free","word_freq_business","word_freq_email","word_freq_you","word_freq_credit","wor
d_freq_your","word_freq_font","word_freq_000","word_freq_money","word_freq_hp","word_freq_hpl","word_freq_george","word_f
req_650","word_freq_lab","word_freq_labs","word_freq_telnet","word_freq_857","word_freq_data","word_freq_415","word_freq_8
5","word_freq_technology","word_freq_1999","word_freq_parts","word_freq_pm","word_freq_direct","word_freq_cs","word_freq_m
eeting","word_freq_original","word_freq_project","word_freq_re","word_freq_edu","word_freq_table","word_freq_conference","char
_freq_semicolon","char_freq_roundBracket","char_freq_squareBracket","char_freq_exclamationMark","char_freq_dollarSign","char_fr
eq_hashtag","capital_run_length_average","capital_run_length_longest","capital_run_length_total", "V58")
Freq= table(dataa$V58)
N=dim(dataa)[1]
s=sample(1:N, 1000)
dataa=as.data.frame(dataa[s,]) #make the new sample of 1000- reduce data set
Freq= table(dataa$V58)
N=dim(dataa)[1]
ss=sample(1:N, 700)
train=dataa[ss,]
test=dataa[-ss,] #make the test sample
dataav58 <- dataa[,58]
dataaData <- dataa[,-58]
estv58 <- dataav58[-s]
###################LINEAR TRAINING SET

library(e1071)
library(caret)
set.seed(123)
svm.model <- svm(V58 ~., data=train, type='C-classification',
kernel="linear",Scale=TRUE,probability=TRUE,trControl=trainControl(method="cv",number=10, verboseItrer=TRUE,savePredictions =
TRUE))
svmmodel.predict<-predict(svm.model,subset(test,select=-V58),decision.values=TRUE)
svmmodel.probs<-attr(svmmodel.predict,"decision.values")
svmmodel.class<-predict(svm.model,test,type="class")
svmmodel.labels<-test$V58
#analyzing result
library(SDMTools)
svmmodel.confusion<-confusion.matrix(svmmodel.labels,svmmodel.class)
svmmodel.accuracy<-prop.correct(svmmodel.confusion)
CE=(svmmodel.confusion[1,2]+svmmodel.confusion[2,1])/(svmmodel.confusion[2,2]+svmmodel.confusion[1,2]+svmmodel.confusion[2,
1]+svmmodel.confusion[1,1])
FP=(svmmodel.confusion[2,1])/(svmmodel.confusion[2,1]+svmmodel.confusion[2,2])
FN=(svmmodel.confusion[1,2])/(svmmodel.confusion[1,1]+svmmodel.confusion[1,2])
#roc analysis for test data
svmmodel.prediction<-prediction(svmmodel.probs,svmmodel.labels)
svmmodel.performanceL<-performance(svmmodel.prediction,"tpr","fpr")
svmmodel.auc<-performance(svmmodel.prediction,"auc")@y.values[[1]]
#####################RADIAL TRAINING
svmmodelradial<-svm(V58~., data=train, method="C-classification",
                    kernel="radial",cross=10, scale=TRUE,probability=TRUE)
svmmodel.predict<-predict(svmmodelradial,subset(test,select=-V58),decision.values=TRUE)
```

```
svmmodel.probs<-attr(svmmodel.predict,"decision.values")
svmmodel.class<-predict(svmmodelradial,test,type="class")
svmmodel.labels<-test$V58
#analyzing result
library(SDMTools)
svmmodel.confusion<-confusion.matrix(svmmodel.labels,svmmodel.class)
svmmodel.accuracy<-prop.correct(svmmodel.confusion)
#roc analysis for test data
svmmodel.prediction<-prediction(svmmodel.probs,svmmodel.labels)
svmmodel.performanceR<-performance(svmmodel.prediction,"tpr","fpr")
svmmodel.auc<-performance(svmmodel.prediction,"auc")@y.values[[1]]
CE=(svmmodel.confusion[1,2]+svmmodel.confusion[2,1])/(svmmodel.confusion[2,2]+svmmodel.confusion[1,2]+svmmodel.confusion[2,
1]+svmmodel.confusion[1,1])
FP=(svmmodel.confusion[2,1])/(svmmodel.confusion[2,1]+svmmodel.confusion[2,2])
FN=(svmmodel.confusion[1,2])/(svmmodel.confusion[1,1]+svmmodel.confusion[1,2])
#####################POLYNOMIAL TRAINING
svmmodelpoly<-svm(V58~., data=train, method="C-classification", kernel="polynomial",cross=10, scale=TRUE,probability=TRUE)
#predicting the test data
svmmodel.predict<-predict(svmmodelpoly,subset(test,select=-V58),decision.values=TRUE)
svmmodel.probs<-attr(svmmodel.predict,"decision.values")
svmmodel.class<-predict(svmmodelpoly,test,type="class")
svmmodel.labels<-test$V58
#analyzing result
svmmodel.confusion<-confusion.matrix(svmmodel.labels,svmmodel.class)
svmmodel.accuracy<-prop.correct(svmmodel.confusion)
svmmodel.accuracy
svmmodel.confusion
#roc analysis for test data
svmmodel.prediction<-prediction(svmmodel.probs,svmmodel.labels)
svmmodel.performanceP<-performance(svmmodel.prediction,"tpr","fpr")
svmmodel.auc<-performance(svmmodel.prediction,"auc")@y.values[[1]]
CE=(svmmodel.confusion[1,2]+svmmodel.confusion[2,1])/(svmmodel.confusion[2,2]+svmmodel.confusion[1,2]+svmmodel.confusion[2,
1]+svmmodel.confusion[1,1])
FP=(svmmodel.confusion[2,1])/(svmmodel.confusion[2,1]+svmmodel.confusion[2,2])
FN=(svmmodel.confusion[1,2])/(svmmodel.confusion[1,1]+svmmodel.confusion[1,2])
####################LOGISTIC REGRESSION TRAINING
# Logistic regression ------------------------------------------------------
logModel <- glm(V58 ~.,family=binomial(link='logit'),data=train)
summary(logModel)
results <- predict(logModel,newdata=test[,-58],type='response')
testPredictions <- ifelse(results > 0.5,1,0) # to find the right predictions
svmmodel.confusion=table(testPredictions, test[,58])
varImp(logModel)
misClasificError <- mean(testPredictions != test[,58])
print(paste('Accuracy',1-misClasificError))

library(ROCR)
pr <- prediction(results, test[,58])
prf <- performance(pr, measure = "tpr", x.measure = "fpr")
auc <- performance(pr, measure = "auc")
auc <- auc@y.values[[1]]
CE=(svmmodel.confusion[1,2]+svmmodel.confusion[2,1])/(svmmodel.confusion[2,2]+svmmodel.confusion[1,2]+svmmodel.confusion[2,
1]+svmmodel.confusion[1,1])
FP=(svmmodel.confusion[2,1])/(svmmodel.confusion[2,1]+svmmodel.confusion[2,2])
```

```
FN=(svmmodel.confusion[1,2])/(svmmodel.confusion[1,1]+svmmodel.confusion[1,2])
##################### TUNED LINEAR
tunelinearr=tune.svm( V58~. ,data=train,kernel="linear",cost =10^(-2:3) )
install.packages("zoom")
library(zoom)
zoomplot.zoom(fact=2.4,x=0,y=0)
svm.model1<- svm(V58 ~ ., data=train, kernel="linear",type='C-classification', cost=0.1,scale=TRUE,probability=TRUE)
svmmodel.predict<-predict(svm.model1,subset(test,select=-V58),decision.values=TRUE)
svmmodel.probs<-attr(svmmodel.predict,"decision.values")
svmmodel.class<-predict(svm.model1,test,type="class")
svmmodel.labels<-test$V58
library(SDMTools)
svmmodel.confusion<-confusion.matrix(svmmodel.labels,svmmodel.class)
svmmodel.accuracy<-prop.correct(svmmodel.confusion)
#roc analysis for test data
svmmodel.prediction<-prediction(svmmodel.probs,svmmodel.labels)
svmmodel.performanceLT<-performance(svmmodel.prediction,"tpr","fpr")
svmmodel.auc<-performance(svmmodel.prediction,"auc")@y.values[[1]]
CE=(svmmodel.confusion[1,2]+svmmodel.confusion[2,1])/(svmmodel.confusion[2,2]+svmmodel.confusion[1,2]+svmmodel.confusion[2,
1]+svmmodel.confusion[1,1])
FP=(svmmodel.confusion[2,1])/(svmmodel.confusion[2,1]+svmmodel.confusion[2,2])
FN=(svmmodel.confusion[1,2])/(svmmodel.confusion[1,1]+svmmodel.confusion[1,2])
##################### TUNED RADIAL
tuneradiaal=tune.svm( V58~. ,data=train, kernel="radial",cost = 10^(-2:3),gamma =10^(-2:3) )
plot(tuneradiaal,type = c("contour", "perspective"), theta = 120,color.palette = topo.colors,
    nlevels = 20,xlim=c(0,1))
svm.model2<- svm(V58 ~ ., data=train,type='C-classification', kernel="radial", cost=10,gamma=0.01)
svmmodel.predict<-predict(svm.model2,subset(test,select=-V58),decision.values=TRUE)
svmmodel.probs<-attr(svmmodel.predict,"decision.values")
svmmodel.class<-predict(svm.model2,test,type="class")
svmmodel.labels<-test$V58
library(SDMTools)
svmmodel.confusion<-confusion.matrix(svmmodel.labels,svmmodel.class)
svmmodel.accuracy<-prop.correct(svmmodel.confusion)
#roc analysis for test data
svmmodel.prediction<-prediction(svmmodel.probs,svmmodel.labels)
svmmodel.performanceRT<-performance(svmmodel.prediction,"tpr","fpr")
svmmodel.auc<-performance(svmmodel.prediction,"auc")@y.values[[1]]
CE=(svmmodel.confusion[1,2]+svmmodel.confusion[2,1])/(svmmodel.confusion[2,2]+svmmodel.confusion[1,2]+svmmodel.confusion[2,
1]+svmmodel.confusion[1,1])
FP=(svmmodel.confusion[2,1])/(svmmodel.confusion[2,1]+svmmodel.confusion[2,2])
FN=(svmmodel.confusion[1,2])/(svmmodel.confusion[1,1]+svmmodel.confusion[1,2])
##################### TUNED POLYNOMIAL
tunepoly=tune.svm( V58~. ,data=train, kernel="polynomial",degree = c(2,3,4,5,6,7,8,9),scale=TRUE,probability=TRUE)
plot(tunepoly)
zoomplot.zoom(fact=3,x=0,y=0)
svm.model<- svm(V58 ~ ., data=train,type='C-classification', kernel="polynomial",degree=2)
svmmodel.predict<-predict(svm.model,subset(test,select=-V58),decision.values=TRUE)
svmmodel.probs<-attr(svmmodel.predict,"decision.values")
svmmodel.class<-predict(svm.model,test,type="class")
svmmodel.labels<-test$V58
library(SDMTools)
svmmodel.confusion<-confusion.matrix(svmmodel.labels,svmmodel.class)
svmmodel.accuracy<-prop.correct(svmmodel.confusion)
```

```
#roc analysis for test data
svmmodel.prediction<-prediction(svmmodel.probs,svmmodel.labels)
svmmodel.performancePT<-performance(svmmodel.prediction,"tpr","fpr")
svmmodel.auc<-performance(svmmodel.prediction,"auc")@y.values[[1]]
CE=(svmmodel.confusion[1,2]+svmmodel.confusion[2,1])/(svmmodel.confusion[2,2]+svmmodel.confusion[1,2]+svmmodel.confusion[2,
1]+svmmodel.confusion[1,1])
FP=(svmmodel.confusion[2,1])/(svmmodel.confusion[2,1]+svmmodel.confusion[2,2])
FN=(svmmodel.confusion[1,2])/(svmmodel.confusion[1,1]+svmmodel.confusion[1,2])
#####################PLOT TRAINING & TUNED TRAINING
par(mfrow = c(1, 1))
plot( svmmodel.performanceL, col="red",lwd=2)
plot(svmmodel.performanceR, add = TRUE, col="green",lwd=2)
plot(svmmodel.performanceP, add = TRUE, col="blue",lwd=2)
plot( prf, add = TRUE,col="black",lwd=2)
legend(0.3, plot_range[2], c("LINEAR KERNEL auc=0.951","RADIAL KERNEL auc=0.956","POLYNOMIAL KERNEL auc=0.915","LOGISTIC REG
auc=0.945"), cex=0.8,
    col=c("red","green","blue","black"), pch=21:22, lty=1:2)
title(main="ROC Curve of Models-Sampled SpamBase", font.main=4)
plot( svmmodel.performanceLT, col="red",lwd=2)
plot(svmmodel.performanceRT, add = TRUE, col="green",lwd=2)
plot(svmmodel.performancePT, add = TRUE, col="blue",lwd=2)
plot( prf,add=TRUE, col="black",lwd=2)
legend(0.5, plot_range[2], c("LINEAR KERNEL auc=0.952","RADIAL KERNEL auc=0.964","POLYNOMIAL KERNEL auc=0.917","LOGISTIC REG
auc=0.945"), cex=0.7,
    col=c("red","green","blue","black"), pch=21:22, lty=1:2)
title(main="ROC Curve of Tuned-three Models-Sampled SpamBase", font.main=4)
plot( svmmodel.performancePCAL, col="red",lwd=2)
plot(svmmodel.performancePCAR, add = TRUE, col="green",lwd=2)
plot(svmmodel.performancePCAP, add = TRUE, col="blue",lwd=2)
plot( prf,add=TRUE, col="black",lwd=2)
legend(0.5, plot_range[2], c("LINEAR KERNEL auc=0.954","RADIAL KERNEL auc=0.949","POLY KERNEL auc=0.901","LOGISTIC REG
auc=0.94"), cex=0.8,
    col=c("red","green","blue","black"), pch=21:22, lty=1:2)
title(main="ROC Curve of PCA20-Sampled SpamBase", font.main=4)
plot( svmmodel.performancePCALT, col="red",lwd=2)
plot(svmmodel.performancePCART, add = TRUE, col="green",lwd=2)
plot(svmmodel.performancePCAPT, add = TRUE, col="blue",lwd=2)
plot( prf,add=TRUE, col="black",lwd=2)
legend(0.5, plot_range[2], c("LINEAR KERNEL auc=0.95","RADIAL KERNEL auc=0.948","POLY KERNEL auc=0.88","LOGISTIC REG
auc=0.94"), cex=0.8, col=c("red","green","blue","black"), pch=21:22, lty=1:2)
title(main="ROC Curve of PCA20-Tuned-three Models-Sampled SpamBase", font.main=4)
print(h)
print(h+coord_cartesian(ylim=c(-0.5,.5)))
##plot pca customized d=3
plot(svmmodel.performanceK1,col="yellow",lwd=2)
plot(svmmodel.performanceK2,add = TRUE,col="red",lwd=2)
plot(svmmodel.performanceK3,add = TRUE,col="green",lwd=2)
plot(svmmodel.performanceK4,add = TRUE,col="blue",lwd=2)
plot(svmmodel.performanceK5,add = TRUE,col="black",lwd=2)
legend(0.5, plot_range[2], c("K1 auc=0.963","K2 auc=0.96","K3 auc=0.81","K4 auc=0.82","K5 auc=0.82"), cex=1,
    col=c("yellow","red","green","blue","black"), pch=21:22, lty=1:2)
title(main="ROC Curve of K1-K5 C=1,d=3,G=1/41 Models SpamBase", font.main=4)
plot(svmmodel.performanceK6,col="yellow",lwd=2)
plot(svmmodel.performanceK7,add = TRUE,col="red",lwd=2)
```

```
plot(svmmodel.performanceK8,add = TRUE,col="green",lwd=2)
plot(svmmodel.performanceK9,add = TRUE,col="blue",lwd=2)
plot(svmmodel.performanceK10,add = TRUE,col="black",lwd=2)
legend(0.5, plot_range[2], c("K6 auc=0.8","K7 auc=0.96","K8 auc=0.81","K9 auc=0.81","K10 auc=0.82"), cex=1,
    col=c("yellow","red","green","blue","black"), pch=21:22, lty=1:2)
title(main="ROC Curve of K6-K10 C=1,d=3,G=1/41 Models SpamBase", font.main=4)
#########################################################################################################################
##plot pca customized d=2
plot(svmmodel.performanceK11,col="yellow",lwd=2)
plot(svmmodel.performanceK22,add = TRUE,col="red",lwd=2)
plot(svmmodel.performanceK33,add = TRUE,col="green",lwd=2)
plot(svmmodel.performanceK44,add = TRUE,col="blue",lwd=2)
plot(svmmodel.performanceK55,add = TRUE,col="black",lwd=2)
legend(0.5, plot_range[2], c("K1 auc=0.96","K2 auc=0.96","K3 auc=0.82","K4 auc=0.86","K5 auc=0.86"), cex=1,
    col=c("yellow","red","green","blue","black"), pch=21:22, lty=1:2)
title(main="ROC Curve of K1-K5 C=1,d=2,G=1/41 Models SpamBase", font.main=4)
plot(svmmodel.performanceK66,col="yellow",lwd=2)
plot(svmmodel.performanceK77,add = TRUE,col="red",lwd=2)
plot(svmmodel.performanceK88,add = TRUE,col="green",lwd=2)
plot(svmmodel.performanceK99,add = TRUE,col="blue",lwd=2)
plot(svmmodel.performanceK100,add = TRUE,col="black",lwd=2)
legend(0.5, plot_range[2], c("K6 auc=0.82","K7 auc=0.963","K8 auc=0.83","K9 auc=0.83","K10 auc=0.85"), cex=1,
    col=c("yellow","red","green","blue","black"), pch=21:22, lty=1:2)
title(main="ROC Curve of K6-K10 C=1,d=2,G=1/41 Models SpamBase", font.main=4)
#########################################################################################################################
######PCA TRAINING
set.seed(123)
mydata= read.table("C:\\Users\\FARNOOSH\\Desktop\\kernels\\kerneldatasets\\spambase.csv",header=FALSE, sep=",")
N=dim(mydata)[1]
s=sample(1:N, 1000)
dataa=as.data.frame(dataa[s,]) #make the new sample of 1000- reduce data set
dim(dataa)
Freq= table(dataa$V58)
mydata.new <- (dataa[, 1:57])
fix(mydata.new)
V58 <- dataa[, 58]
pcafunct <- prcomp(mydata.new, center = TRUE, scale. = TRUE)
summary(pcafunct)
#covariance matrix of PCAs
#diag of covariance matrix is variance of each of principle components= eigenvalue of origin data
diag(var(pcafunct$x))
pcafunct$x
mean(pcafunct$x)
var(pcafunct$x)
eigen(cor(mydata.new))$value
cor(pcafunct$x)
#corr between original variable and PCAs
cor(cbind(mydata.new,data.frame(pcafunct$x)))
head(pcafunct)
print(pcafunct$rotation)
print(pcafunct)
plot(pcafunct)
plot(pcafunct, type = "l")
predict(pcafunct,  newdata=tail(mydata.new, 3))
```

```
#pca slope= pcafunct$x
#y is my new data
y=data.frame(pcafunct$x)
y1=y[,1:2]
library(ggbiplot)
h <- ggbiplot(pcafunct, obs.scale = 1, var.scale = 1,
         groups = V58, ellipse = TRUE,
         circle = TRUE)
print(h)
print(h+coord_cartesian(ylim=c(-5,5)))
x.neww=cbind(y,V58)
dim(x.neww)
x.new<-cbind(y[,1:20],V58)
dim(x.new)
str(x.new)
fix(x.new)
#cor v58 with pcas
cor(x.new)[,42]
N=dim(x.new)[1]
ss=sample(1:N, 700)
train=x.new[ss,]
test=x.new[-ss,]
head(test)
test[,42]
#################PCA LINEAR
library(e1071)
set.seed(123)
svm.model <- svm(V58 ~ ., data=train, type='C-classification', kernel="linear",Scale=TRUE,probability=TRUE)
summary(svm.model)
svmmodel.predict<-predict(svm.model,subset(test,select=-V58),decision.values=TRUE)
svmmodel.probs<-attr(svmmodel.predict,"decision.values")
svmmodel.class<-predict(svm.model,test,type="class")
svmmodel.labels<-test$V58
#analyzing result
library(SDMTools)
svmmodel.confusion<-confusion.matrix(svmmodel.labels,svmmodel.class)
svmmodel.accuracy<-prop.correct(svmmodel.confusion)
#roc analysis for test data
svmmodel.prediction<-prediction(svmmodel.probs,svmmodel.labels)
svmmodel.performancePCAL<-performance(svmmodel.prediction,"tpr","fpr")
svmmodel.auc<-performance(svmmodel.prediction,"auc")@y.values[[1]]
CE=(svmmodel.confusion[1,2]+svmmodel.confusion[2,1])/(svmmodel.confusion[2,2]+svmmodel.confusion[1,2]+svmmodel.confusion[2,
1]+svmmodel.confusion[1,1])
FP=(svmmodel.confusion[2,1])/(svmmodel.confusion[2,1]+svmmodel.confusion[2,2])
FN=(svmmodel.confusion[1,2])/(svmmodel.confusion[1,1]+svmmodel.confusion[1,2])
################################################radial
svmmodelradial<-svm(V58~., data=train, method="C-classification",
            kernel="radial",scale=TRUE, probability=TRUE)
svmmodel.predict<-predict(svmmodelradial,subset(test,select=-V58),decision.values=TRUE)
svmmodel.probs<-attr(svmmodel.predict,"decision.values")
svmmodel.class<-predict(svmmodelradial,test,type="class")
svmmodel.labels<-test$V58
library(SDMTools)
svmmodel.confusion<-confusion.matrix(svmmodel.labels,svmmodel.class)
```

```
svmmodel.confusion
svmmodel.accuracy<-prop.correct(svmmodel.confusion)
#roc analysis for test data
svmmodel.prediction<-prediction(svmmodel.probs,svmmodel.labels)
svmmodel.performancePCAR<-performance(svmmodel.prediction,"tpr","fpr")
svmmodel.auc<-performance(svmmodel.prediction,"auc")@y.values[[1]]
svmmodel.auc
CE=(svmmodel.confusion[1,2]+svmmodel.confusion[2,1])/(svmmodel.confusion[2,2]+svmmodel.confusion[1,2]+svmmodel.confusion[2,
1]+svmmodel.confusion[1,1])
FP=(svmmodel.confusion[2,1])/(svmmodel.confusion[2,1]+svmmodel.confusion[2,2])
FN=(svmmodel.confusion[1,2])/(svmmodel.confusion[1,1]+svmmodel.confusion[1,2])
##################################PCA POLY
svmmodelpoly<-svm(V58~., data=train, method="C-classification", kernel="polynomial",scale=TRUE,probability=TRUE)
#predicting the test data
svmmodel.predict<-predict(svmmodelpoly,subset(test,select=-V58),decision.values=TRUE)
svmmodel.probs<-attr(svmmodel.predict,"decision.values")
svmmodel.class<-predict(svmmodelpoly,test,type="class")
svmmodel.labels<-test$V58

#analyzing result
svmmodel.confusion<-confusion.matrix(svmmodel.labels,svmmodel.class)
svmmodel.accuracy<-prop.correct(svmmodel.confusion)
#roc analysis for test data
svmmodel.prediction<-prediction(svmmodel.probs,svmmodel.labels)
svmmodel.performancePCAP<-performance(svmmodel.prediction,"tpr","fpr")
svmmodel.auc<-performance(svmmodel.prediction,"auc")@y.values[[1]]
CE=(svmmodel.confusion[1,2]+svmmodel.confusion[2,1])/(svmmodel.confusion[2,2]+svmmodel.confusion[1,2]+svmmodel.confusion[2,
1]+svmmodel.confusion[1,1])
FP=(svmmodel.confusion[2,1])/(svmmodel.confusion[2,1]+svmmodel.confusion[2,2])
FN=(svmmodel.confusion[1,2])/(svmmodel.confusion[1,1]+svmmodel.confusion[1,2])
##############################PCA REGRESSION
logModel <- glm(V58 ~.,family=binomial(link='logit'),data=train)
summary(logModel) # get the summary of the model
results <- predict(logModel,newdata=test[,-21],type='response')
testPredictions <- ifelse(results > 0.5,1,0) # get the right predictions
svmmodel.confusion=table(testPredictions, test[,21])
svmmodel.confusion
varImp(logModel)
misClasificError <- mean(testPredictions != test[,42])
print(paste('Accuracy',1-misClasificError))
library(ROCR)
pr <- prediction(results, test[,21])
prf <- performance(pr, measure = "tpr", x.measure = "fpr")
auc <- performance(pr, measure = "auc")
auc <- auc@y.values[[1]]
CE=(svmmodel.confusion[1,2]+svmmodel.confusion[2,1])/(svmmodel.confusion[2,2]+svmmodel.confusion[1,2]+svmmodel.confusion[2,
1]+svmmodel.confusion[1,1])
FP=(svmmodel.confusion[2,1])/(svmmodel.confusion[2,1]+svmmodel.confusion[2,2])
FN=(svmmodel.confusion[1,2])/(svmmodel.confusion[1,1]+svmmodel.confusion[1,2])
############################PCA TUNED LINEAR
tunelinearrpcatuned=tune.svm( V58~.,data=train , kernel="linear",cost = 10^(-2:3))
plot(tunelinearrpcatuned,xlim=c(0,10))
summary(tunelinearrpcatuned)
par(mfrow = c(1, 1))
```

```
zoomplot.zoom(fact=1.1,x=0,y=0)
svm.model<- svm(V58 ~ ., data=train, kernel="linear", method="C-classification", cost=0.01,scale=TRUE,probability=TRUE)
svmmodel.predict<-predict(svm.model,subset(test,select=-V58),decision.values=TRUE)
svmmodel.probs<-attr(svmmodel.predict,"decision.values")
svmmodel.class<-predict(svm.model,test,type="class")
svmmodel.labels<-test$V58
library(SDMTools)
svmmodel.confusion<-confusion.matrix(svmmodel.labels,svmmodel.class)
svmmodel.accuracy<-prop.correct(svmmodel.confusion)
#roc analysis for test data
svmmodel.prediction<-prediction(svmmodel.probs,svmmodel.labels)
svmmodel.performancePCALT<-performance(svmmodel.prediction,"tpr","fpr")
svmmodel.auc<-performance(svmmodel.prediction,"auc")@y.values[[1]]
CE=(svmmodel.confusion[1,2]+svmmodel.confusion[2,1])/(svmmodel.confusion[2,2]+svmmodel.confusion[1,2]+svmmodel.confusion[2,
1]+svmmodel.confusion[1,1])
FP=(svmmodel.confusion[2,1])/(svmmodel.confusion[2,1]+svmmodel.confusion[2,2])
FN=(svmmodel.confusion[1,2])/(svmmodel.confusion[1,1]+svmmodel.confusion[1,2])
###################################PCA TUNED RADIAL
tuneradiaalpcatuned=tune.svm( V58~. ,data=train, kernel="radial",cost =10^(-2:3),gamma =10^(-2:2) )
tuneradiaalpcatuned
plot(tuneradiaalpcatuned,type = c("contour", "perspective"), theta = 120,color.palette = topo.colors,
    nlevels = 20,xlim=c(0,1))
svm.model<- svm(V58 ~ ., data=train, kernel="radial",method="C-classification", cost=1,gamma=0.1)
svmmodel.predict<-predict(svm.model,subset(test,select=-V58),decision.values=TRUE)
svmmodel.probs<-attr(svmmodel.predict,"decision.values")
svmmodel.class<-predict(svm.model,test,type="class")
svmmodel.labels<-test$V58
library(SDMTools)
svmmodel.confusion<-confusion.matrix(svmmodel.labels,svmmodel.class)
svmmodel.accuracy<-prop.correct(svmmodel.confusion)
#roc analysis for test data
svmmodel.prediction<-prediction(svmmodel.probs,svmmodel.labels)
svmmodel.performancePCART<-performance(svmmodel.prediction,"tpr","fpr")
svmmodel.auc<-performance(svmmodel.prediction,"auc")@y.values[[1]]
CE=(svmmodel.confusion[1,2]+svmmodel.confusion[2,1])/(svmmodel.confusion[2,2]+svmmodel.confusion[1,2]+svmmodel.confusion[2,
1]+svmmodel.confusion[1,1])
FP=(svmmodel.confusion[2,1])/(svmmodel.confusion[2,1]+svmmodel.confusion[2,2])
FN=(svmmodel.confusion[1,2])/(svmmodel.confusion[1,1]+svmmodel.confusion[1,2])
#########################################PCA TUNED POLY
tunepolypcatuned=tune.svm( V58~. ,data=train, kernel="polynomial",degree = 2^(1:4))
plot(tunepolypcatuned)
svm.model<- svm(V58 ~ ., data=train, kernel="polynomial",method="C-classification",degree=2,scale=TRUE,probability=TRUE)
svmmodel.predict<-predict(svm.model,subset(test,select=-V58),decision.values=TRUE)
svmmodel.probs<-attr(svmmodel.predict,"decision.values")
svmmodel.class<-predict(svm.model,test,type="class")
svmmodel.labels<-test$V58
library(SDMTools)
svmmodel.confusion<-confusion.matrix(svmmodel.labels,svmmodel.class)
svmmodel.accuracy<-prop.correct(svmmodel.confusion)
#roc analysis for test data
svmmodel.prediction<-prediction(svmmodel.probs,svmmodel.labels)
svmmodel.performancePCAPT<-performance(svmmodel.prediction,"tpr","fpr")
svmmodel.auc<-performance(svmmodel.prediction,"auc")@y.values[[1]]
```

```
CE=(svmmodel.confusion[1,2]+svmmodel.confusion[2,1])/(svmmodel.confusion[2,2]+svmmodel.confusion[1,2]+svmmodel.confusion[2,
1]+svmmodel.confusion[1,1])
FP=(svmmodel.confusion[2,1])/(svmmodel.confusion[2,1]+svmmodel.confusion[2,2])
FN=(svmmodel.confusion[1,2])/(svmmodel.confusion[1,1]+svmmodel.confusion[1,2])

################################################CUSTOMIZED KERNEL#####################################################
set.seed(123)
b1=sample(seq(from = 0, to = 1, by =0.01), size =400, replace =TRUE)
set.seed(4321)
b2=sample(seq(from = 0, to = 1, by =0.01), size = 400, replace = TRUE)
b3= c(1:390)
B=as.matrix(cbind(b1,b2))
B=unique(B)
summed <- rowSums(B[, c(1, 2)])
B=as.matrix(cbind(B,summed,b3))
count3 <- length(which(B[,3] <= 1))
B=head(B[B[,3]<=1,],195)
unique(B[,1])
#b3
for (i in 1:195) {
 B[i,4]=1-B[i,3]
 B
}
newB= B[,-3]
COEF=fix(newB)
library(xtable)
COEF=xtable(COEF)
print.xtable(COEF, type="latex", file="COEF.tex")
write.csv(newB,     "C:\\Users\\FARNOOSH\\Desktop\\COEFB1B2B3.csv")
myfunction <- function(b){K1 <- function(x,y) {newB[b,1]*(sum(x*y) +1)+newB[b,2]*exp(-0.125*sum((x-
y)^2))+newB[b,3]*((sum(x*y)+1)^3)}
class(K1) <- "kernel"
 return(K1)
}
library(kernlab)
iter <- 195
nb_variables <- 5
resultspambase=matrix(unlist(lapply(c(1:iter), function(i) {
 svmmodelz<-ksvm(Class~., data=train, method="C-classification", kernel=myfunction(i),cross=10,scale=TRUE, probability=TRUE)
 #predicting the test data
 svmmodel.predicz<-predict(svmmodelz,subset(test,select=-Class),type="decision")
 svmmodel.claz<-predict(svmmodelz,test)
 svmmodel.labelz<-test$Class
 #analyzing result
 svmmodel.confusion<-confusion.matrix(svmmodel.labelz,svmmodel.claz)
 svmmodel.accuracz<-prop.correct(svmmodel.confusion)
 #roc analysis for test data
 svmmodel.predictioz<-prediction(svmmodel.predicz,svmmodel.labelz)
 svmmodel.performancej<-performance(svmmodel.predictioz,"tpr","fpr")
 svmmodel.auz<-performance(svmmodel.predictioz,"auc")@y.values[[1]]
CE=(svmmodel.confusion[1,2]+svmmodel.confusion[2,1])/(svmmodel.confusion[2,2]+svmmodel.confusion[1,2]+svmmodel.confusion[2,
1]+svmmodel.confusion[1,1])
 FP=(svmmodel.confusion[2,1])/(svmmodel.confusion[2,1]+svmmodel.confusion[2,2])
 FN=(svmmodel.confusion[1,2])/(svmmodel.confusion[1,1]+svmmodel.confusion[1,2])
```

```
 ACC=svmmodel.accuracz
 AUC=svmmodel.auz
 c(CE, FP, FN, ACC, AUC)
})), iter, nb_variables, byrow = T)
resultspambase=resultt
write.csv(resultspambase, "C:\\Users\\FARNOOSH\\Desktop\\resultspambase.csv")
library(xtable)
resultspambase=xtable(resultspambase)
print.xtable(resultspambase, type="latex", file="resultspambase.tex")
resultspambase [,2]
scatterplot3d(newB[,1],newB[,2], resultspambase [,5], pch=20, highlight.3d=TRUE, col.axis="blue",
          col.grid="lightblue",
          type="h", main="ACCR Sample SpamBase-NEW Kernel",xlab="beta1", ylab="beta2",zlab="ACCR")

newB[,3]
library(scatterplot3d)
scatterplot3d(newB[,1],newB[,2], resultspambase [,2], main="3D Scatterplot")
scatterplot3d(newB[,1],newB[,2], resultspambase [,4], pch=20, highlight.3d=TRUE, col.axis="blue",
          col.grid="lightblue",
          type="h")
library(Rcmdr)
a=scatter3d(newB[,1],newB[,2], resultspambase [,1],xlab="beta1", ylab="beta2",zlab="CE")
##################################################correspnde b1 b2 b3 and smallest ce
set.seed(123)
resultspambase = read.table("C:\\Users\\FARNOOSH\\Desktop\\ resultspambase.csv",header=TRUE, sep=",")
orderACCURACY= resultspambase [order(resultsamplespambase$X4),]
write.csv(orderACCURACY, "C:\\Users\\FARNOOSH\\Desktop\\orderACCURACYSPAMBASE.csv")
orderCE= resultspambase [order(resultspambase $X1),]
write.csv(orderCE, "C:\\Users\\FARNOOSH\\Desktop\\orderCESPAMBASE.csv")
which(resultspambase == max(resultspambase $X4), arr.ind = TRUE)

#april2018  kernelmatrix chapter 2 plots
set.seed(123)
data=runif(500, -10,10)
View(data)
ys=ifelse(data>0,1,0)
data1 <- data.frame(data, y=as.factor(ys))
fix(data1)
Freq= table(data1$y)
Freq
library(e1071)
require(kernlab)
dt=as.matrix(data1[,-2])
gaus=rbfdot(sigma=0.5)
kermatrix=kernelMatrix(gaus,dt)
View(kermatrix)
yt <- as.matrix(as.integer(data1[,2]))
yt[yt==2] <- 0
p=kernelPol(gaus, dt, ,yt)
e=kernelMult(gaus, dt, ,yt)
e <- eigen(kermatrix)
eigenvector=as.matrix(e$vectors)
eigenvalue=as.matrix(e$values)
```

```
View(eigenvalue)
a=c(sqrt(e$values[1])*e$vectors[,1])
b=c(sqrt(e$values[2])*e$vectors[,2])
c=c(sqrt(e$values[3])*e$vectors[,3])
d=c(sqrt(e$values[4])*e$vectors[,4])
e=c(sqrt(e$values[5])*e$vectors[,5])
f=c(sqrt(e$values[6])*e$vectors[,6])
g=c(sqrt(e$values[7])*e$vectors[,7])
h=c(sqrt(e$values[8])*e$vectors[,8])
i=c(sqrt(e$values[9])*e$vectors[,9])
par(mfrow = c(3,3 ))
plot(dt,a,xlab="Data Point", ylab="Phi[,i]",main="Feature Mapping-Gaussian
Kernel")
plot(dt,b,xlab="Data Point", ylab="Phi[,i]")
plot(dt,c,xlab="Data Point", ylab="Phi[,i]")
plot(dt,d,xlab="Data Point", ylab="Phi[,i]")
plot(dt,e,xlab="Data Point", ylab="Phi[,i]")
plot(dt,f,xlab="Data Point", ylab="Phi[,i]")
plot(dt,g,xlab="Data Point", ylab="Phi[,i]")
plot(dt,h,xlab="Data Point", ylab="Phi[,i]")
plot(dt,i,xlab="Data Point", ylab="Phi[,i]")
install.packages("xtable")
library(xtable)
ww=xtable(data1)
print.xtable(ww, type="latex", file="ww.tex")
?print.xtable
#
klinear <- function(x,y) {(sum(x*y))}
class(klinear) <- "kernel"
kermatrix=kernelMatrix(klinear,dt)
e <- eigen(kermatrix)
eigenvector=as.matrix(e$vectors)
eigenvalue=as.matrix(e$values)
View(eigenvalue)
a=c(sqrt(e$values[1])*e$vectors[,1])
b=c(sqrt(e$values[2])*e$vectors[,2])
c=c(sqrt(e$values[3])*e$vectors[,3])
d=c(sqrt(e$values[4])*e$vectors[,4])
e=c(sqrt(e$values[5])*e$vectors[,5])
f=c(sqrt(e$values[6])*e$vectors[,6])
g=c(sqrt(e$values[7])*e$vectors[,7])
h=c(sqrt(e$values[8])*e$vectors[,8])
i=c(sqrt(e$values[9])*e$vectors[,9])
par(mfrow = c(3,3 ))
plot(dt,a,xlab="Data Point", ylab="Phi[,i]",main="Feature Mapping-Linear
Kernel")
plot(dt,b,xlab="Data Point", ylab="Phi[,i]")
plot(dt,c,xlab="Data Point", ylab="Phi[,i]")
plot(dt,d,xlab="Data Point", ylab="Phi[,i]")
plot(dt,e,xlab="Data Point", ylab="Phi[,i]")
plot(dt,f,xlab="Data Point", ylab="Phi[,i]")
plot(dt,g,xlab="Data Point", ylab="Phi[,i]")
plot(dt,h,xlab="Data Point", ylab="Phi[,i]")
```

```
plot(dt,i,xlab="Data Point", ylab="Phi[,i]")
kpoly <- function(x,y) {(sum(x*y) +1)^3}
class(k) <- "kernel"
kermatrix=kernelMatrix(kpoly,dt)
e <- eigen(kermatrix)
eigenvector=as.matrix(e$vectors)
eigenvalue=as.matrix(e$values)
View(eigenvalue)
a=c(sqrt(e$values[1])*e$vectors[,1])
b=c(sqrt(e$values[2])*e$vectors[,2])
c=c(sqrt(e$values[3])*e$vectors[,3])
d=c(sqrt(e$values[4])*e$vectors[,4])
e=c(sqrt(e$values[5])*e$vectors[,5])
f=c(sqrt(e$values[6])*e$vectors[,6])
g=c(sqrt(e$values[7])*e$vectors[,7])
h=c(sqrt(e$values[8])*e$vectors[,8])
i=c(sqrt(e$values[9])*e$vectors[,9])
par(mfrow = c(3,3 ))
plot(dt,a,xlab="Data Point", ylab="Phi[,i]",main="Feature Mapping-Polynomial
Kernel")
plot(dt,b,xlab="Data Point", ylab="Phi[,i]")
plot(dt,c,xlab="Data Point", ylab="Phi[,i]")
plot(dt,d,xlab="Data Point", ylab="Phi[,i]")
plot(dt,e,xlab="Data Point", ylab="Phi[,i]")
plot(dt,f,xlab="Data Point", ylab="Phi[,i]")
plot(dt,g,xlab="Data Point", ylab="Phi[,i]")
plot(dt,h,xlab="Data Point", ylab="Phi[,i]")
plot(dt,i,xlab="Data Point", ylab="Phi[,i]")
k <- function(x,y) {0.7*(sum(x*y) +1)+0.33*exp(-0.5*sum((x-
y)^2))+0.33*((0.5*sum(x*y)+1)^3)}
class(k) <- "kernel"
kermatrix=kernelMatrix(k,dt)
e <- eigen(kermatrix)
eigenvector=as.matrix(e$vectors)
eigenvalue=as.matrix(e$values)
View(eigenvalue)
a=c(sqrt(e$values[1])*e$vectors[,1])
b=c(sqrt(e$values[2])*e$vectors[,2])
c=c(sqrt(e$values[3])*e$vectors[,3])
d=c(sqrt(e$values[4])*e$vectors[,4])
e=c(sqrt(e$values[5])*e$vectors[,5])
f=c(sqrt(e$values[6])*e$vectors[,6])
g=c(sqrt(e$values[7])*e$vectors[,7])
h=c(sqrt(e$values[8])*e$vectors[,8])
i=c(sqrt(e$values[9])*e$vectors[,9])

par(mfrow = c(3,3 ))
plot(dt,a,xlab="Data Point", ylab="Phi[,i]",main="Feature Mapping-Cusomized
Kernel")
plot(dt,b,xlab="Data Point", ylab="Phi[,i]")
plot(dt,c,xlab="Data Point", ylab="Phi[,i]")
plot(dt,d,xlab="Data Point", ylab="Phi[,i]")
plot(dt,e,xlab="Data Point", ylab="Phi[,i]")
```

```
plot(dt,f,xlab="Data Point", ylab="Phi[,i]")
plot(dt,g,xlab="Data Point", ylab="Phi[,i]")
plot(dt,h,xlab="Data Point", ylab="Phi[,i]")
plot(dt,i,xlab="Data Point", ylab="Phi[,i]")
```