Support Vector Machines (H02D3a)

Exercise Sessions report

Marimuthu Ananthavelu (r0652832), MAI, KU Leuven.

2016-2017.

Table of Contents

1. Classification

1.1 Two Gaussians

Geometric construction of a classifier line looks complicated given the data points are spread across each other. The classifying line should ideally be a decision boundary between the two classes namely 'versicolor' and 'virginica. In our case, the decision boundary cannot be a straight line considering the nature of the data points which all are pertaining to two different classes. When the decision boundary separates these 2 classes with clearly indicating all the data points pertaining to 'versicolor' on one side and all the data points pertaining to 'virginica' on the other side, then we can consider that the classifier is optimal/valid.

1.2 The Support Vector Machines

1.1.1. When adding the new data points to the vector space the following happens;

- Each additional data point gets added to the dataset
- Decision boundary adjusts itself with respect to the modified dataset and accordingly the number of iterations gets modified with respect the inclusion of data point and re-optimization.
- When the new data point is added to the extreme left or right sides of the decision boundary, the existing decision boundary changes its position drastically (making a larger difference in the positional points in the vector space) considering the new data point.

1.1.2. When adding an outlier, the following happens;

- With the same given parameters of 'C' and 'kernel sigma', the decision boundary changes with respect to the new data point and the new support vector is getting added to the side opposite of the class in which the outlier is present.

1.1.3. C is the parameter for the soft margin cost function, which controls the influence of each individual support vector.

Changing the C value to different values the following are the observations;

- The increase in C parameter, makes the classifier to be more accurate as possible by penalising the misclassification heavily.
- The decrease in C parameter, makes the classifier to be leaving some of the data points under the wrong class which makes the classifier less robust.
- For example, with C parameter of '1', the classifier is as below;

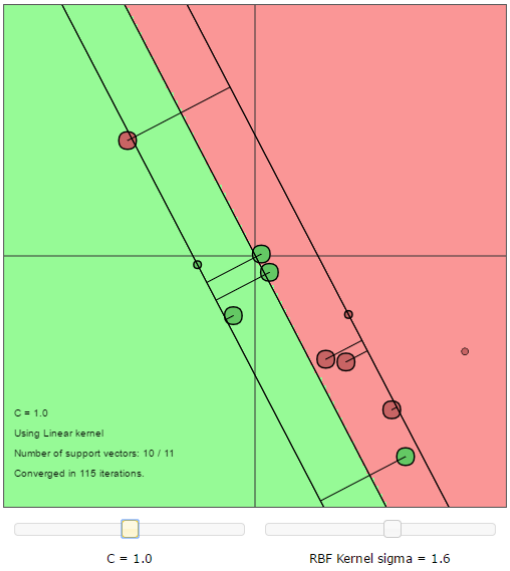


Fig.1.Classifier with C=1

With C parameter of '25', the classifier is as below;

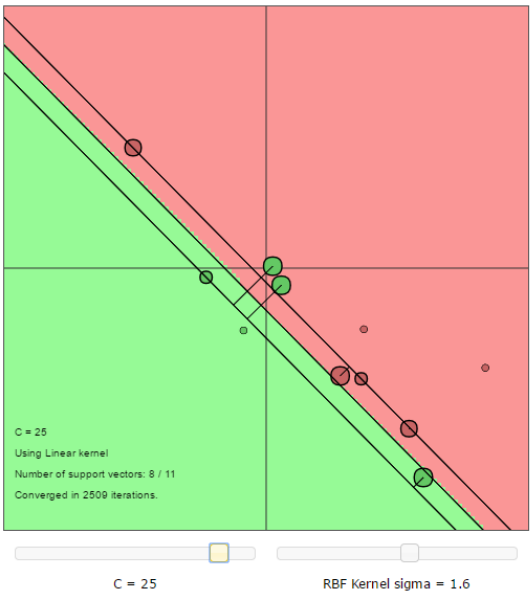


Fig.2.Classifier with C=25

- In Fig-1, with C=1, the classifier is not robust and classifies one or some of the data points on the wrong side of the decision boundary.
- In Fig-2, with C=25, the classifier attempts to be more robust and creates a very closely possible decision boundary especially giving importance to the data points which all are outliers.

Using RBF kernel:

1.1.4. Switching over to RBF kernel gives the decision boundary which is not a straight line but closed over the different cluster(s) of data points. This clearly indicates that the RBF can be used for non-linearly separable data points. The classifier space looks as below;

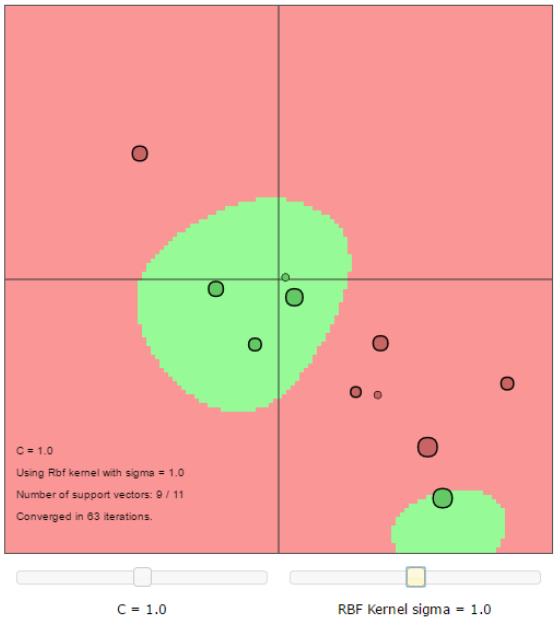


Fig.3.Classifier using BF Kernel

1.1.5. Sigma is a parameter of the Gaussian kernel and defines the steepness of the rise around the landmark. The following changes happen with respect to the change in 'sigma' value.

- Increase in 'sigma' makes the decision boundary with stronger smoothing i.e. high bias with low variance
- Decrease in 'sigma' makes the decision boundary with overfitting for the chosen data points

Changing the both the hyperparameters provides the following the observations for a given kernel;

- The C Parameter implies the penalties on the misclassified data and thus higher C value provides better classified data with smooth boundaries.
- High C parameter value and low sigma value provides an optimal decision boundary

1.1.6. Role of the Given Kernel, Regularization parameter, and the kernel parameter:

- Linear Kernel, is limited to classifying the data points which all are linearly separable using the straight line. They can't be used for complex classification applications where as the linearly non-separable data are presented.
- C Parameter in both the kernels, gives an indication about smoothing of the decision boundary. The higher the value, the more penalty it awards for the misclassifying data points. The less they are, the kernel will leave some of the outliers and place the data points on the wrong side of the decision boundary.
- Sigma is applicable to the 'RBF' kernel function where the higher value puts the data points on the wrong side of the decision boundary whereas the small values make the decision boundary which overfits.
- In case of Linear kernel, an optimum C Parameter is required.
- In case of RBF kernel, an optimum C and Sigma parameter are required.
- Essentially *both* the kernel and regularization parameters (C) control capacity, and one can get a diagonal trough in the Cross-validation error as a function of the hyper-parameters because their effects are correlated and different combinations of kernel parameter and regularization provide similarly good models.

1.1.7. Role of Support Vectors:

Fig.4.Support vectors visualization

- In the above figure, the support vectors are the points which lie along the supporting hyperplanes (the hyperplanes parallel to the dividing hyperplane at the edges of the margin). Regardless of the number of dimensions or size of data set, the number of support vectors could be as little as 2.
- When the margin becomes very wide, there are lots of support vectors.
- The number of support vectors depend upon the how much C parameter is allowed for the given distribution of the data. If large value of C parameter is allowed, then we will have a large number of support vectors and If small value of C parameter is allowed, we will have very few support vectors.
- Accuracy of our classification depends on the trade-off between a high-complexity model which may over-fit the data and a large-margin which will incorrectly classify some of the training data in the interest of better generalization. The number of support vectors can range from very few to every single data point which will completely over-fit your data. This tradeoff is controlled via C parameter and through the choice of kernel and kernel parameters.
- The support vector machine searches for the closest points which it calls the "support vectors. Support Vector Machines are an optimization problem. They are attempting to find a hyperplane that divides the two classes with the largest margin. The support vectors are the points which fall within this margin. For an example, in the following classification problem, there are 2 support vectors (one on negative side and another on positive side)

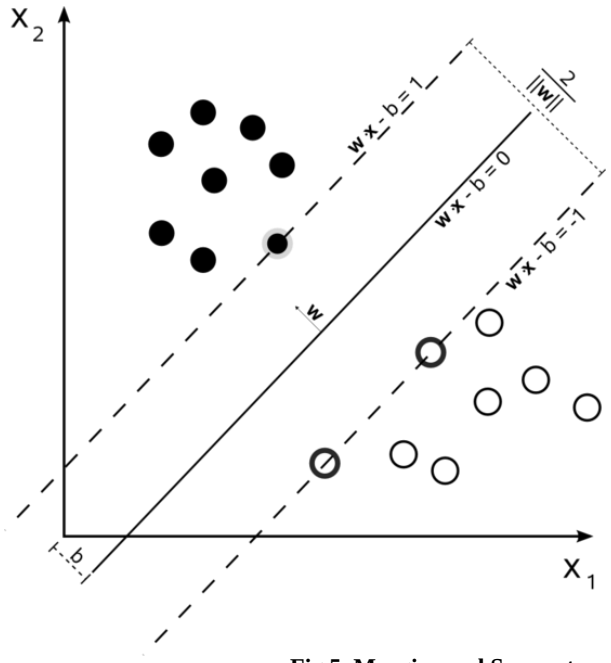


Fig.5. Margins and Support vectors visualization

1.1.8. In the support-vectors learning algorithm the complexity of the construction does not depend on the dimensionality of the feature space, but on the number of support vectors.

- Support Vectors basically are the observations needed to define the fitted model. The more complex the model is (the more specific and overtrained), the more data points will be needed to specify the model Very few support vectors usually define a general model (with more misclassifications), while many support vectors specify a very specific model (possibly overfit). For a linear vector machine, support vectors are the observations lying within the 3margin or miss- classified. A point becomes a support vector when he he plays a role in defining the model/classification boundaries.
- Addition of more data points increases the number of support vectors normally as the decision boundary changes and accordingly, the data points on close to the margin changes.
- A Particular data point becomes support vector when it falls under or situated on the margins in the classification space.

1.3 Using LS-SVMlab

1. The performance on the given test set for iris dataset using the linear (kernel) function is as below:

#misclass = 11, error rate = 55.00%

When changing the kernel to polynomial with varying degree of polynomials, the misclassifications and the error rates are as below;

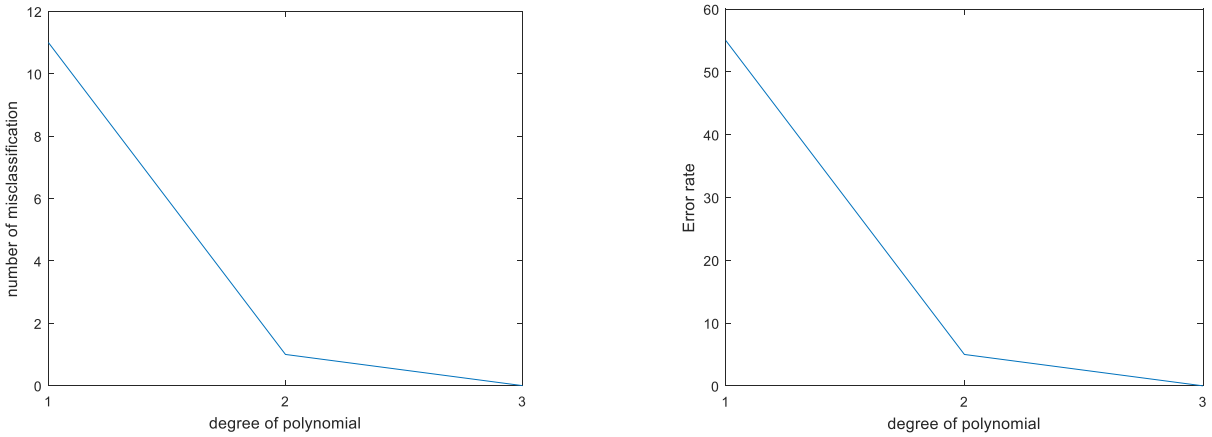


Fig.6. Polynomial kernel – Misclassification and Error rate

The observations when increasing the degree of polynomial have similar impacts as the value of Sigma decreases. The increased polynomial makes the classifier to separate the non-linearly separable data points using the curvature/quadratic based decision boundary.

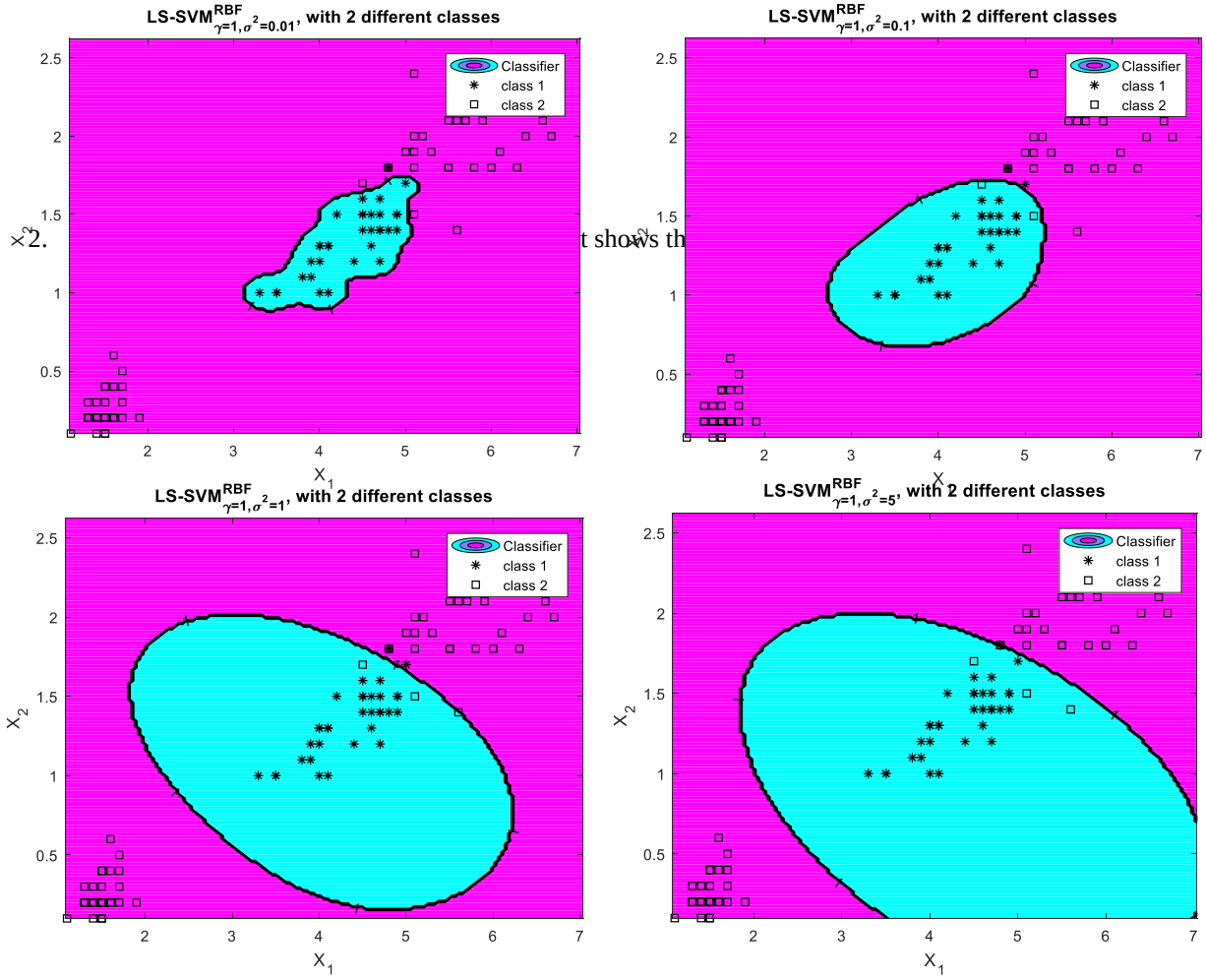


Fig.7. Polynomial kernel effects of degree of increase

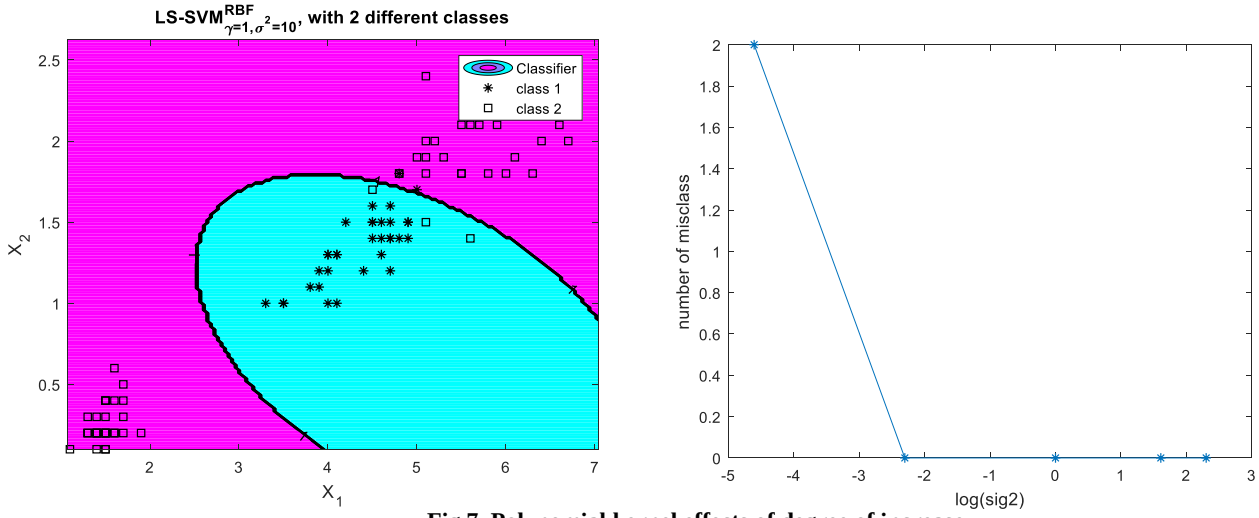


Fig.7. Polynomial kernel effects of degree of increase

3. With fixed Sigma2 value and varying gamma values, the number of misclassifications are as below;

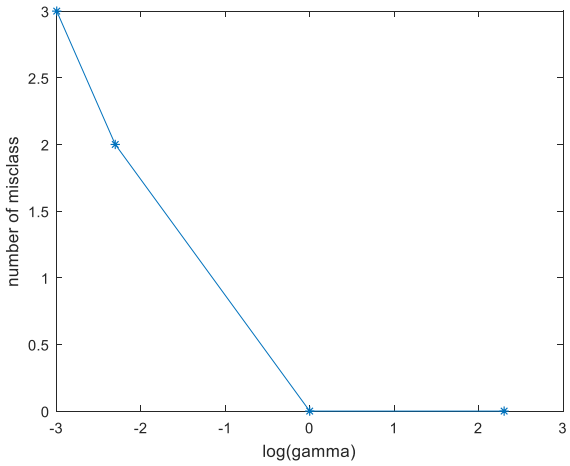


Fig.8. Effect of gamma values

- A Good range of gamma values starts from ~0.05. Below this value, the contour plot is finding a matrix with the same value everywhere and doesn't have any level change, so no contour could be drawn.
- When the gamma value is increased further (500 in our case for the given data set and the parameter settings), the decision boundary becomes 2 instead of original one. Now the misclassified data points are also increased (from 0 to 1). Thus, the maximum parameter value in this case is preferred to be less than 500. This is visible as below;

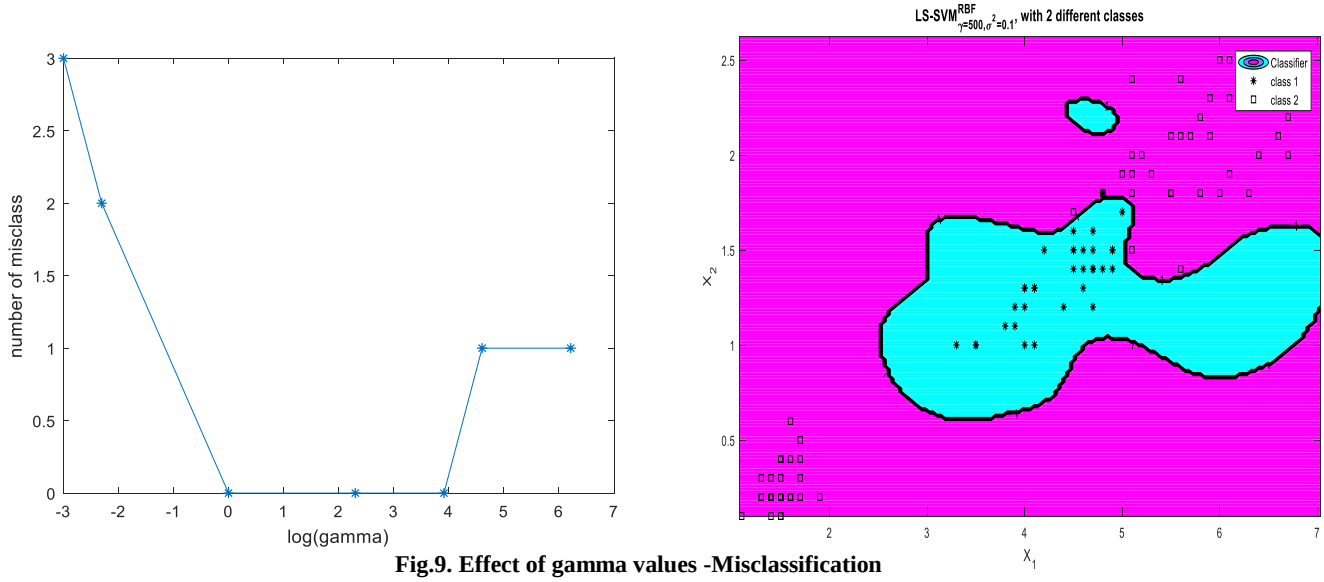


Fig.9. Effect of gamma values -Misclassification

1.1.9. The Choice of hyper parameters:

Performance in terms of misclassification on the validation set:

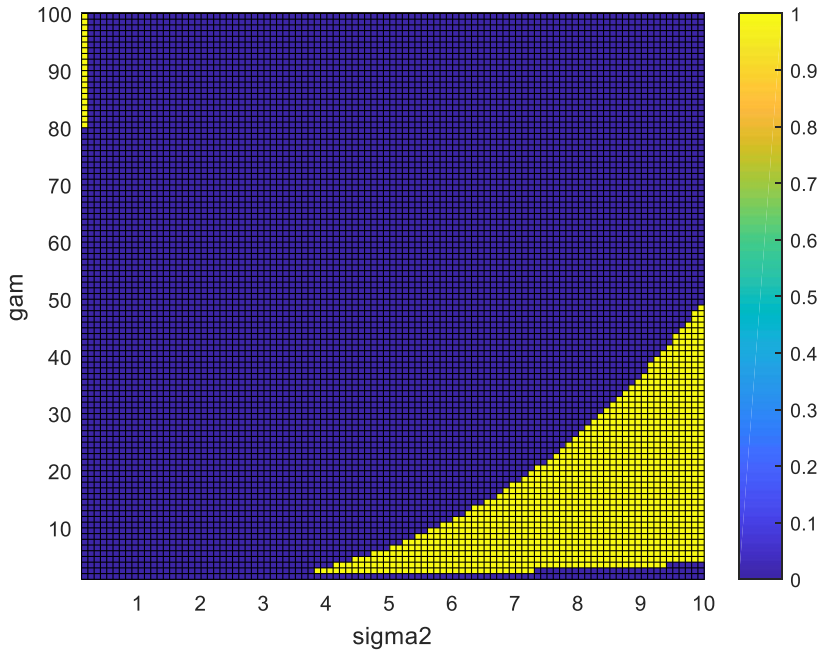


Fig.10. Effect of hyper parameters

- With less value of sigma2 over the range of gamma parameters, the number of misclassified data points are still less.
- When the sigma2 increases above the 4 and for the value of gamma under 50, the there is a misclassified data point(s).

1.1.10. Cross validation:

Cross validation is a method for estimating the predictive accuracy of the Model and a way to ensure that the classifier learns fair in for unseen data. This is done by splitting the dataset by training and validation data in successive combinations keeping validation data as different each time.

For example:

Step 1 : Splitting the total data set into 10 (most times uniform) sets. Let's say K=10

Step 2 : Consider 1 set as validation set with remaining (K-1) sets for training the model

Step 3 : Choose a different set now and keeping the remaining sets for training the model

Step 4 : Step 2 and 3 makes all the sets of data (different samples in the population) to train the model.

The above Cross validation procedure ensures the model is fair for any unknown new dataset.

Optimization algorithms like grid search, is useful when the parameters are not learned within the estimators, so that it can be set by Grid search by considering the given range of parameters.

Simple validation procedure, has the parameters optimized and having been trained for one set of validation set in the whole dataset. So, it might not generalize well to the data which the model had not seen before and might overfit to the given set of data.

In case of Cross validation (10 times here for example), the validation set changes every time so the model has adjusted its hyper parameters with respect to change in the different validation points. In the latter case, the model can be said it is very generalized.

The performance for the crossvalidate and singleoneout are as below;

Method	performance
crossvalidate	0.05
singleoneout	0.06

Table-1. Performance comparison between cross validate and singleoneout

Preference of 'crossvalidate' or singleoneout:

- Leave one out cross validation looks better when there is a small set of training data. In this case of small data set, one can't make 10 folds to make predictions on using the rest of your data to train the model.
- With the large amount of training data on the other hand, 10-fold cross validation would be a better bet, because there will be too many iterations for leave one out cross-validation, and considering these many results to tune your hyperparameters might not be such a good idea.
- There is always a bias-variance trade-off between doing leave one out and K fold cross validation. In leaveoneout cross validation, we get estimates of test error with lower bias, and higher variance because each training set contains n-1 examples, which means that we are using almost the entire training set in each iteration. This leads to higher variance too, because there are chances of overlap between training sets, and thus the test error estimates are highly correlated, which means that the mean value of the test error estimate will have higher variance.
- The opposite is true with k-fold CV, because there is relatively less overlap between training sets, thus the test error estimates are less correlated, and because of which the mean test error value won't have as much variance as leaveoneout cross validation.

1.1.11. Comparison of different techniques for the optimizing the hyperparameters shall be below:

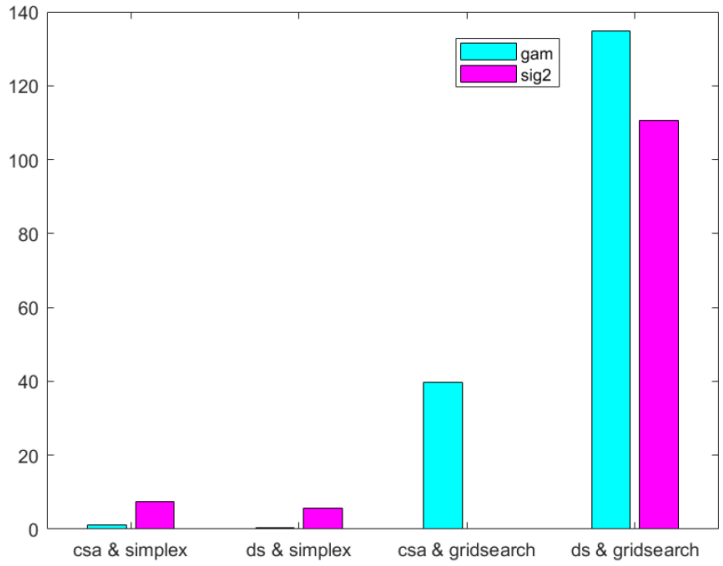


Fig.11. Comparison of optimization techniques for hyperparameters

1.1.12. Receiver Operating Characteristic (ROC):

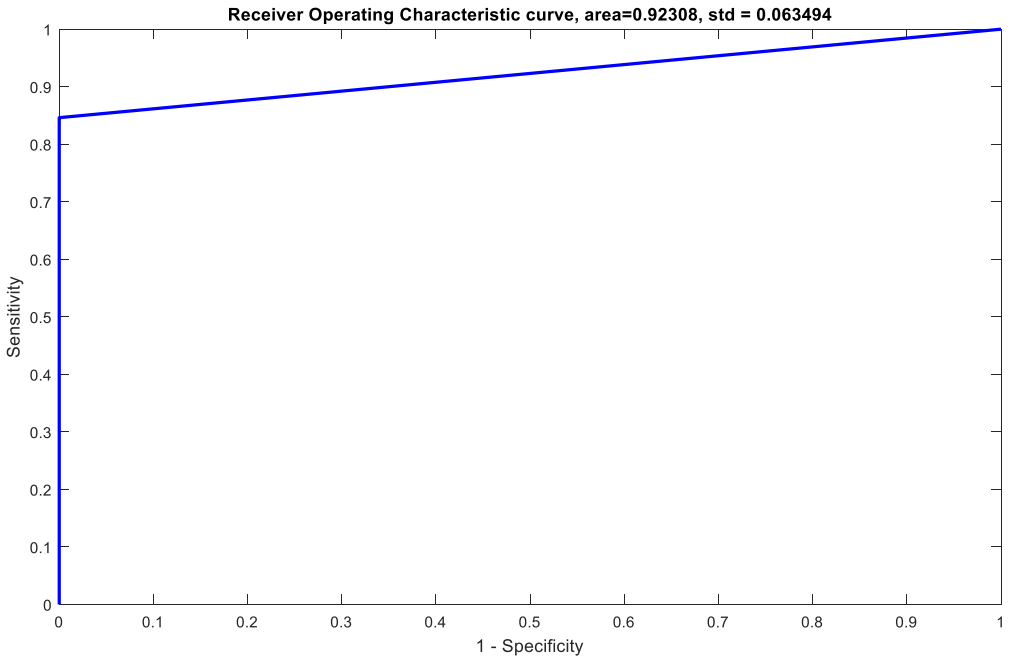


Fig.12. Receiver operating characteristic curve(ROC) curve

The receiver operating characteristic curve(ROC) curves feature true positive rate on the Y axis, and false positive rate on the X axis. This means that the top left corner of the plot is the "ideal" point - a false positive rate of zero, and a true positive rate of one. This is not very realistic, but it does mean that a larger area under the curve (AUC) is usually better. Since the generalization of the model depends upon how the testing set performs with the model, the ROC curve is useful for the testing data set.

1.4 Homework Problems

1.1.13. The Ripley Data-set

1.1.13.1. Plotted data:

Plotted data shows that there are classes which are not clearly separated whereas the data points are located in other side of the classification space.

1.1.13.2. Training with Linear kernel:

For a given dataset with the following spread of positive and negative classes in the vector space, the number of misclassifications and the error rate as follows;

on test: #misclass = 36, error rate = 14.40%

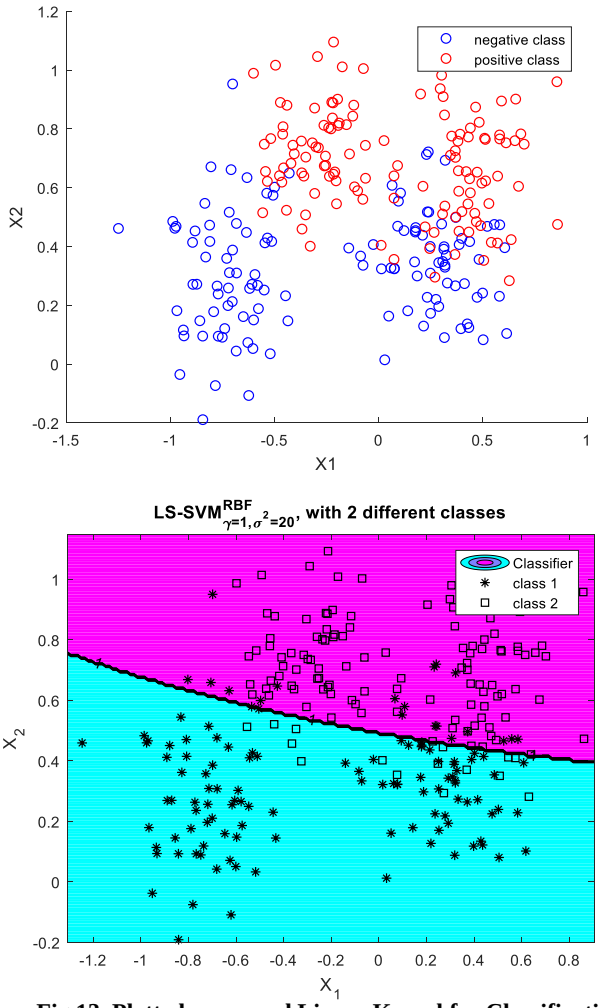


Fig.13. Plotted space and Linear Kernel for Classification

1.1.13.3. **With RBF Kernel;**

With RBF kernel and the parameters of gamma=1, sigma2=20, the classification is not linear but still accounts misclassification errors.

on test: #misclass = 35, error rate = 14.00%

Though there are reduced misclassifications, there are still errors exists.

Tuning the parameters:

With the help of 'tunelssvm' method, the following are the optimized parameters over the 20 different runs.

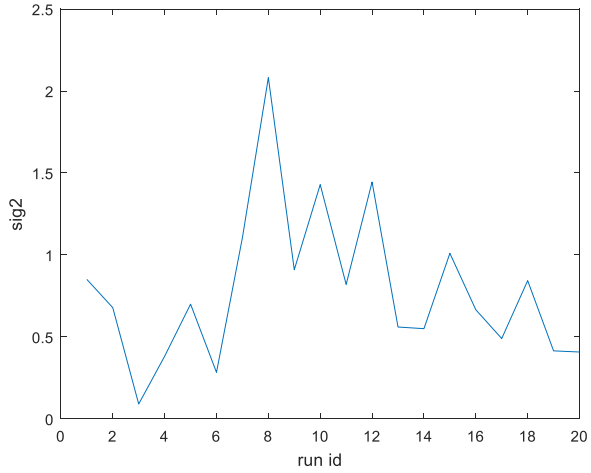


Fig. run id vs gamma

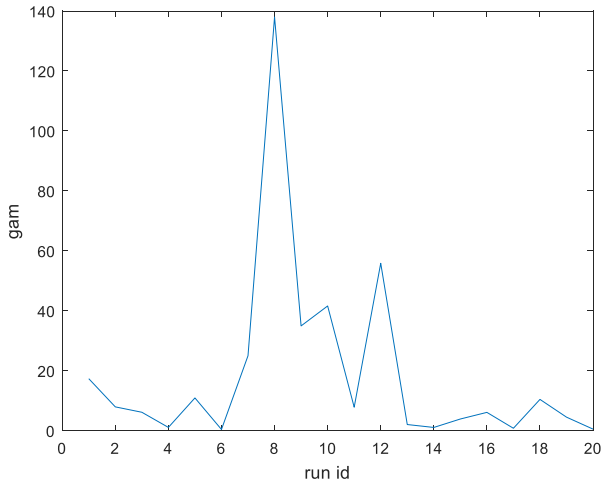


Fig. run id vs sigma2

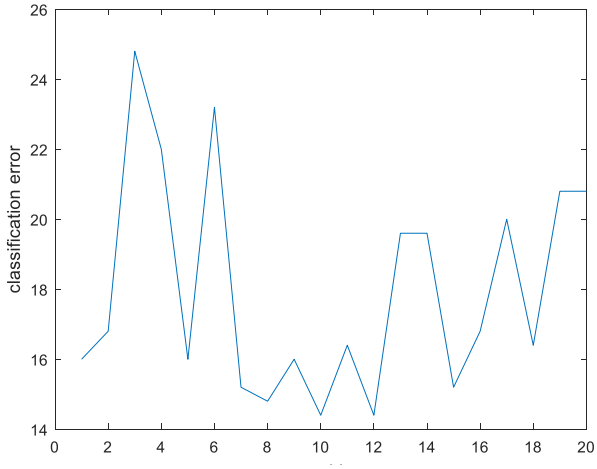


Fig. run id vs classification error

Fig.14.Optimization of parameters over the run

Observations from the RBF kernel optimization:

For a given ripley data set, the optimization parameters vary as below:

- Gamma varies from 0 to 140
- Sigma2 varies from 0 to 2
- Classification errors accordingly vary from 14% to 25%.
- The lowest classification error of 14% comes with gamma value of 40 and sigma2 value of 1.5.
- The highest classification error of 25% comes with gamma value of 5 and sigma2 value of 0.1.
- It is evident from the above observations that, the lowest gamma and sigma2 values misclassify more examples than other combination of these parameters.

1.1.13.4. *Receiver Operating Characteristic Curve:*

As predicted, there is more false positive rate for linear model whereas the RBF kernel shows reasonably more true positives. This can be seen below:

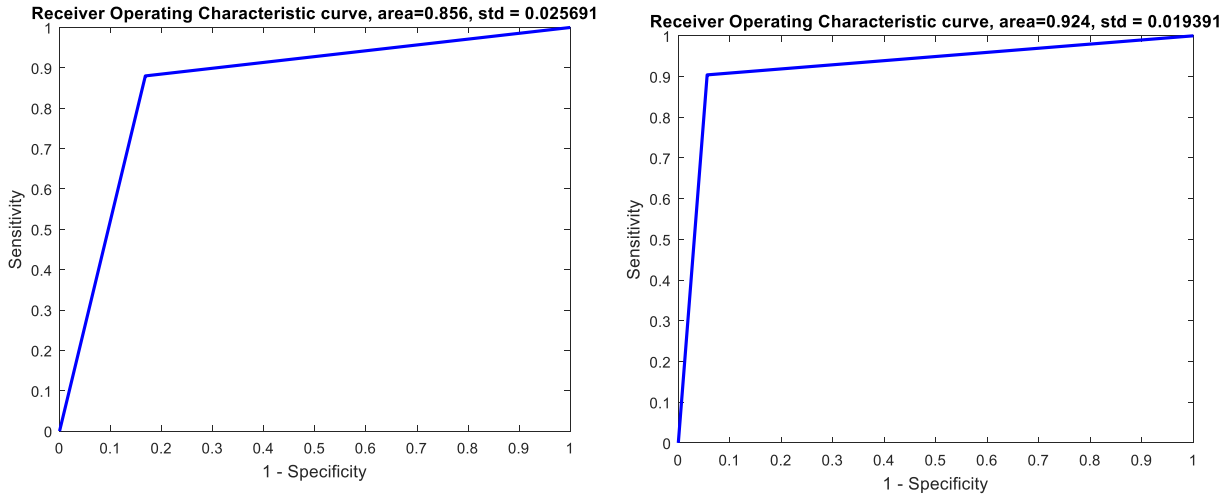


Fig.15 ROC for Linear and RBF kernels

1.1.14. Breast Cancer Dataset

1.1.14.1. Plotting the dataset

Plotting the given variables are done individually as the number of features are 30. Some of the plots are as below;

Matrix of scatter plots by group for some of them:

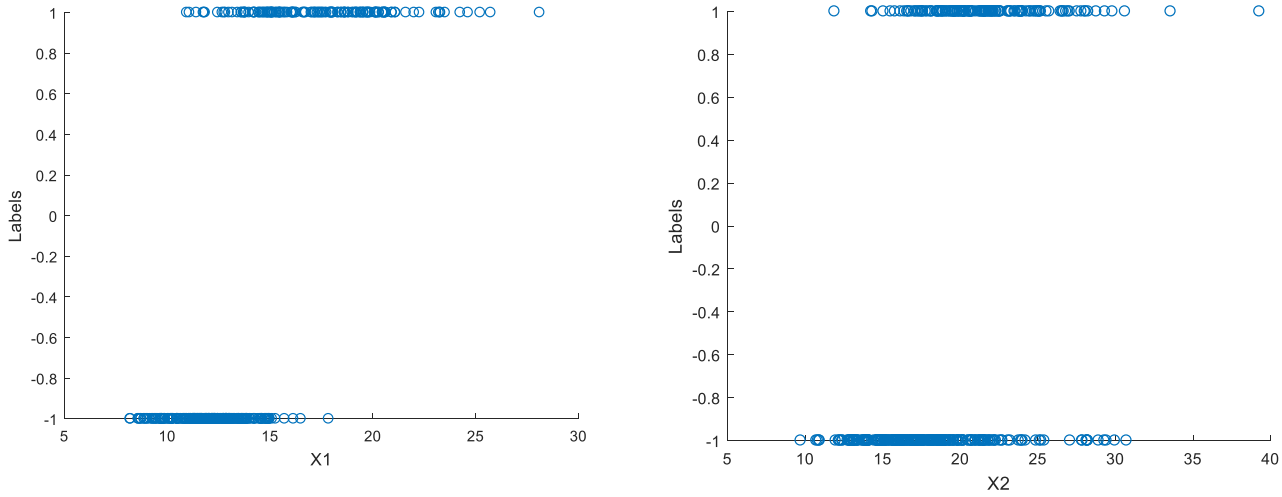


Fig.16 visualization of data points

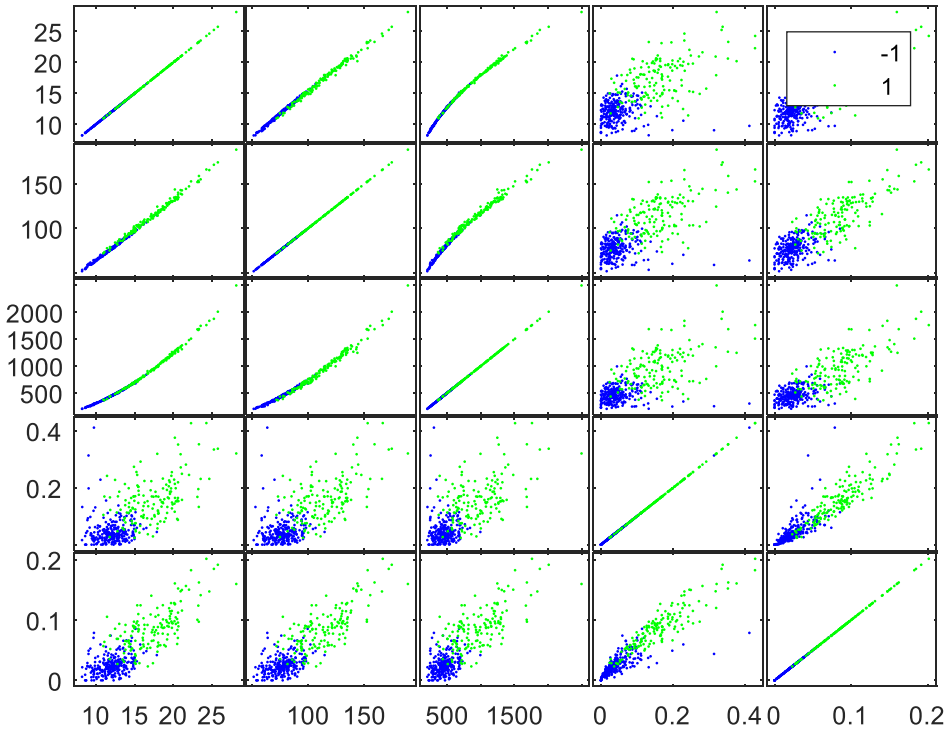


Fig.17 Combination of set wise

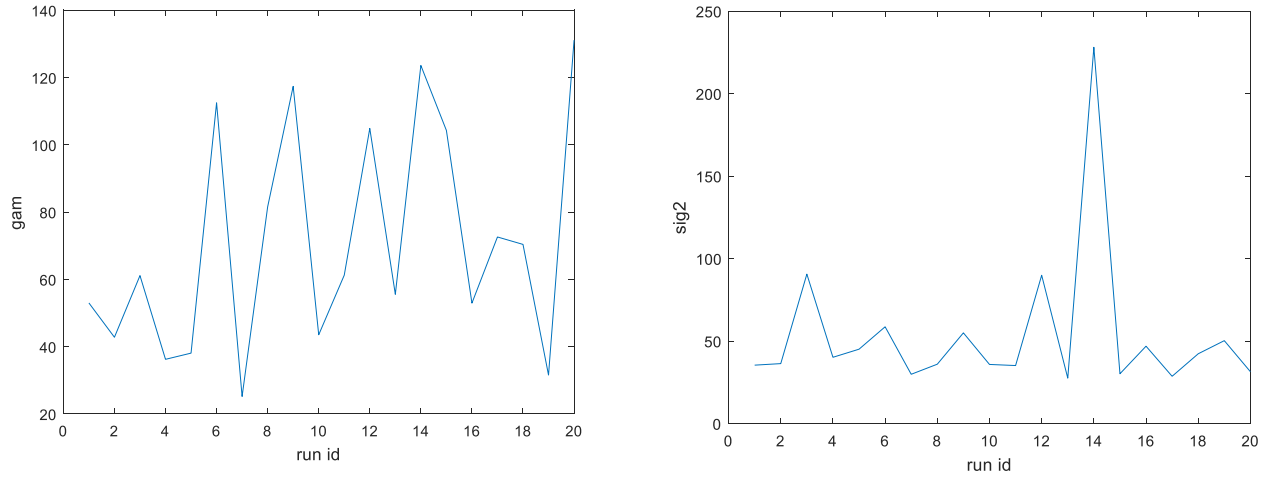
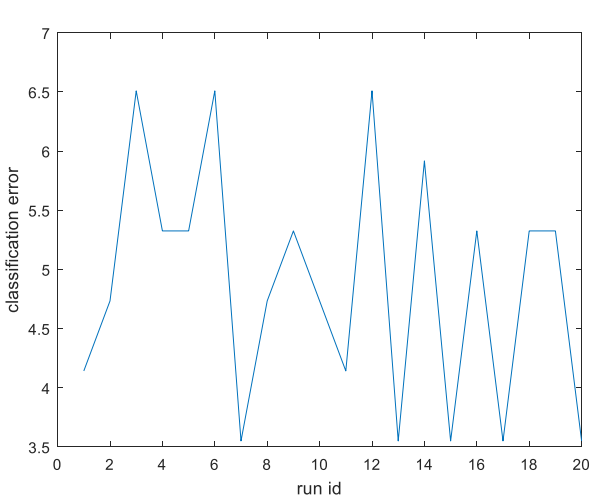
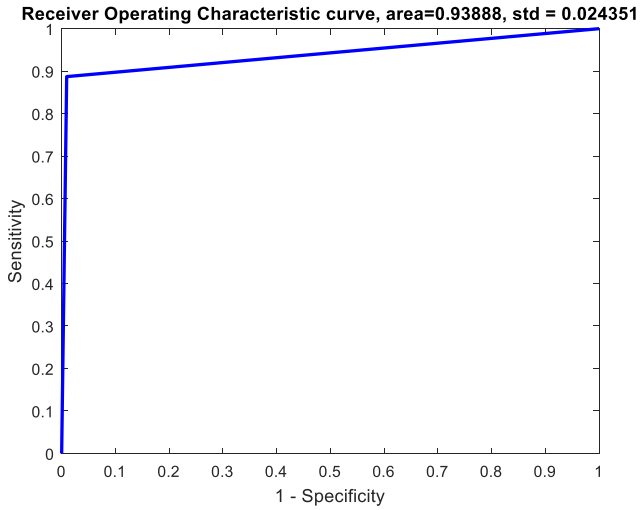


Fig.18 run-id vs hyperparameters





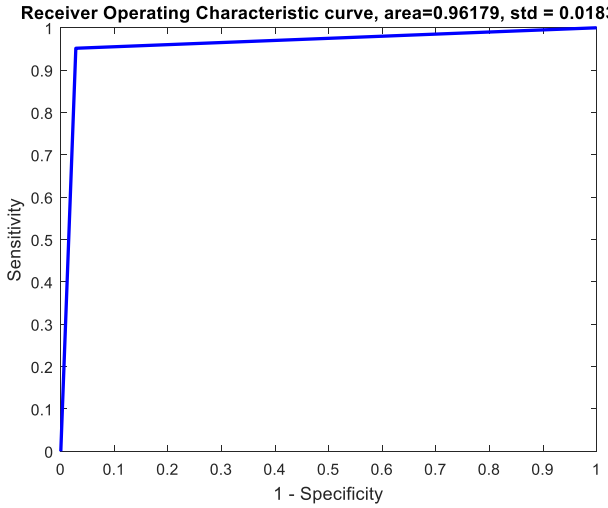


Fig.19 Hyperparameter vs run, ROC's for linear and RBF kernels

1.1.15. Diabetes Database

Similar to the above dataset, this best performance on the highly non-linear vector space,kernel functions perform better than linear kernel.

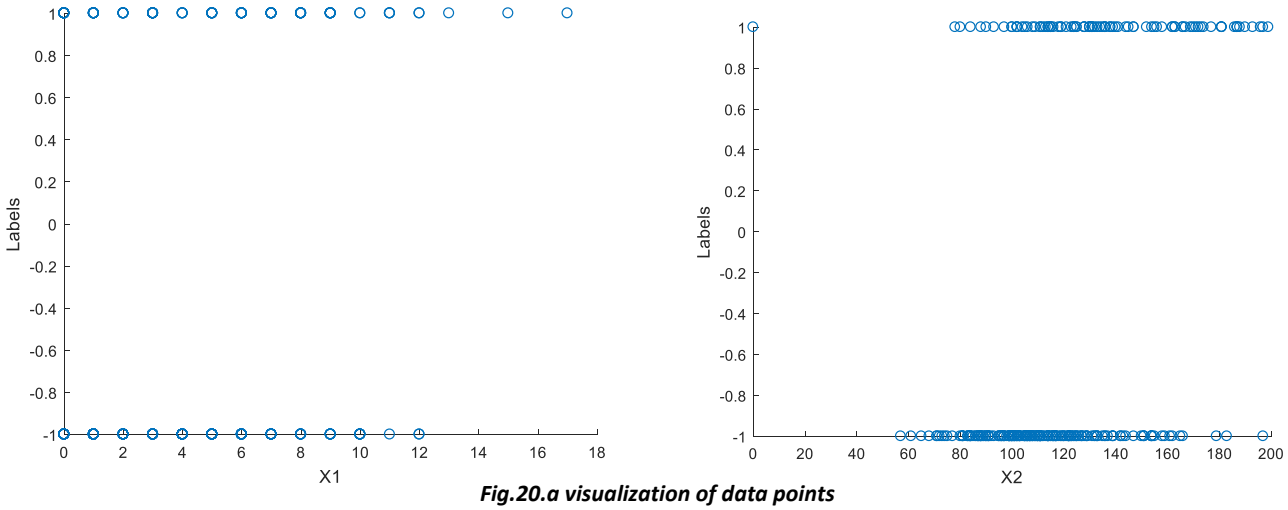
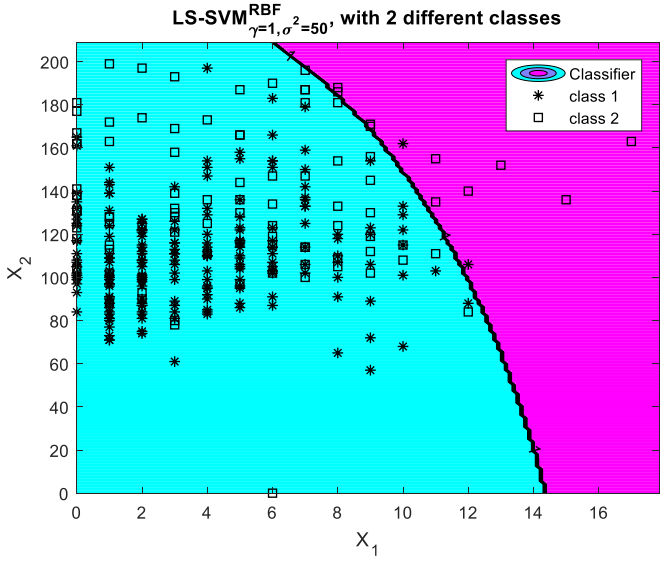
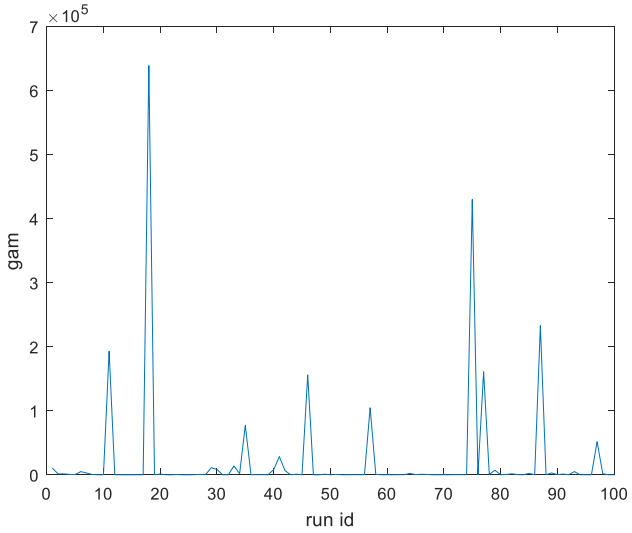
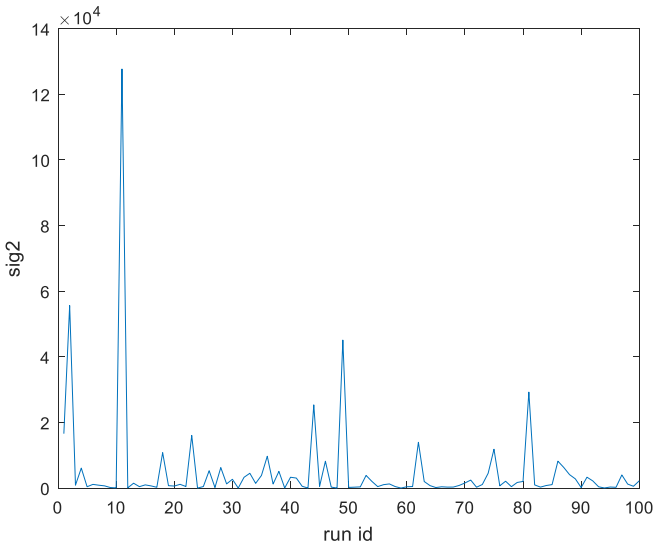
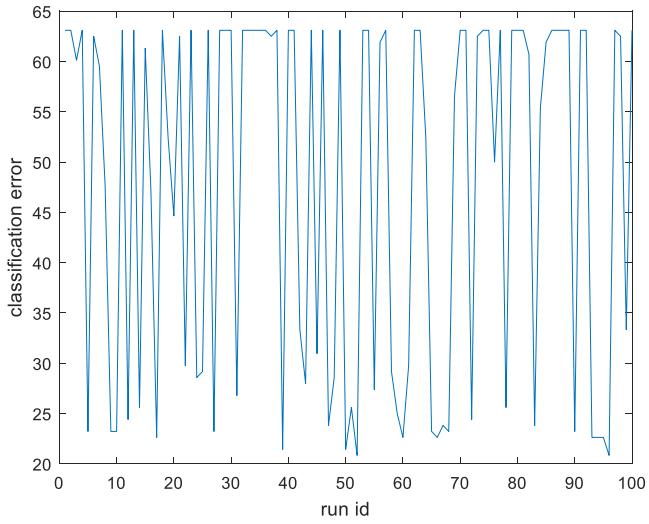


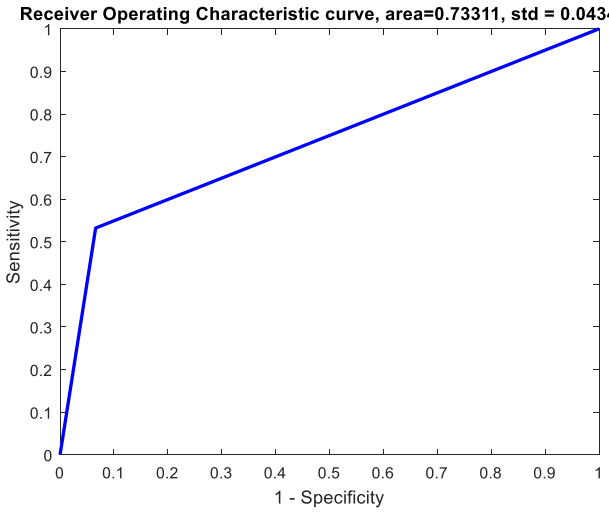
Fig.20.a visualization of data points











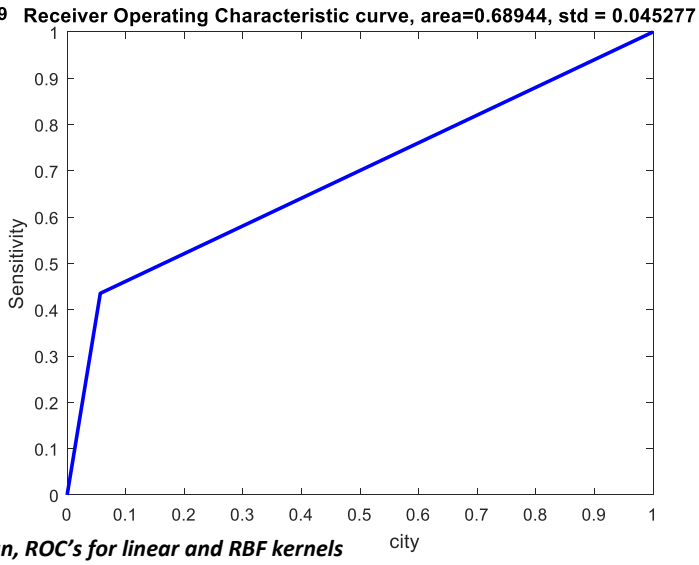


Fig.20b Hyperparameters vs run, ROC's for linear and RBF kernels

2. Function Estimation and Time- series Prediction

For this exercise, the SVM toolbox is used.

2.1 The Support Vector Machine for Regression

A artificial data has been created with the help of 20 different data points using the User interface 'uiregress'. We start by adding 20 points to the user interface. For the chosen data points the added points output the polynomial function of degree 2. And by creating a different dataset using different interface, with different types of kernels.

In the following figures, we display the results for linear, polynomial (degree 2), RBF and linear B-Spline (degree 2). Although theoretically, a polynomial kernel of degree 2 would be the best, we see that other kernels (RBF and linear B-Spline) perform similarly. However, the B-Spline kernel overfits the data.

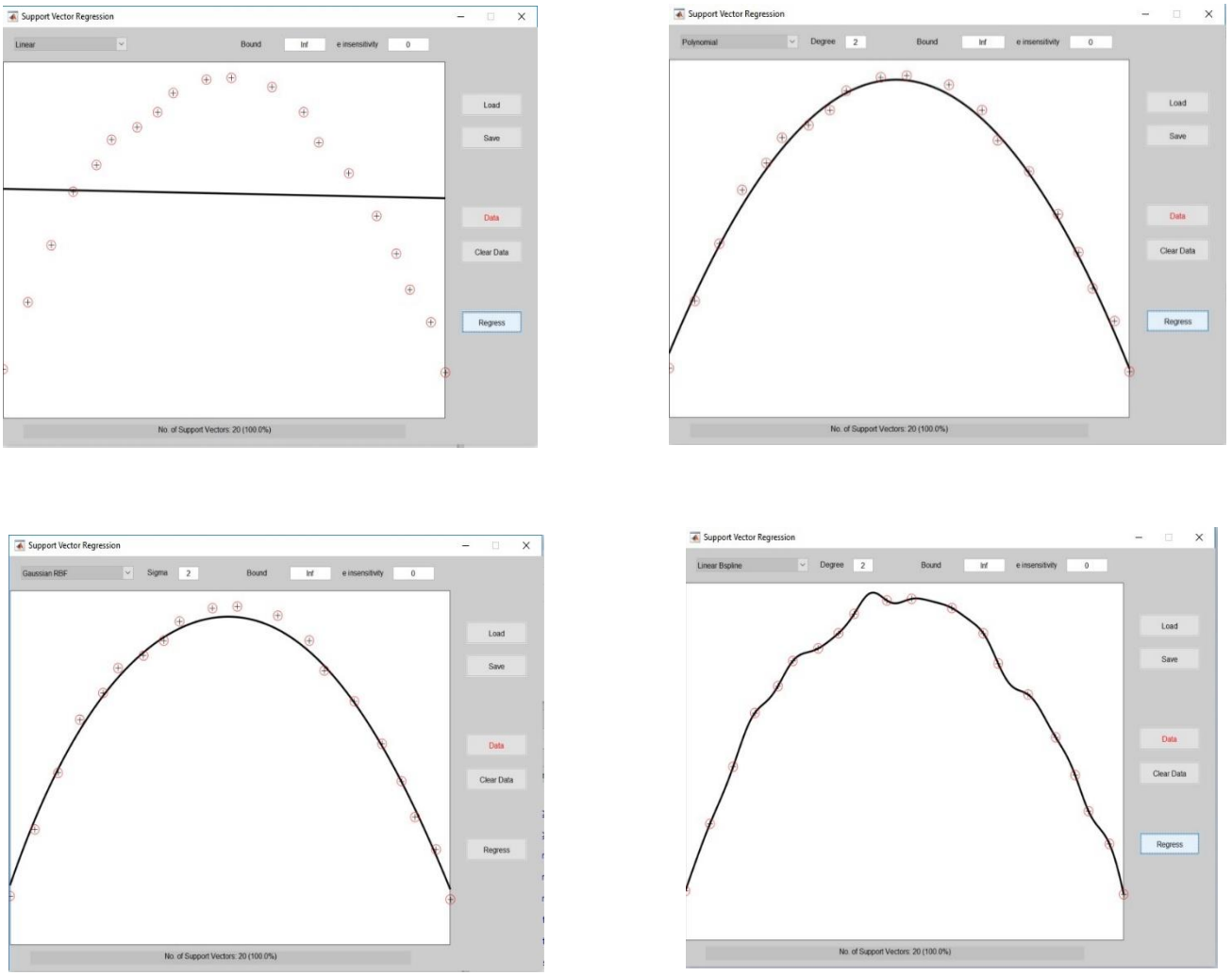
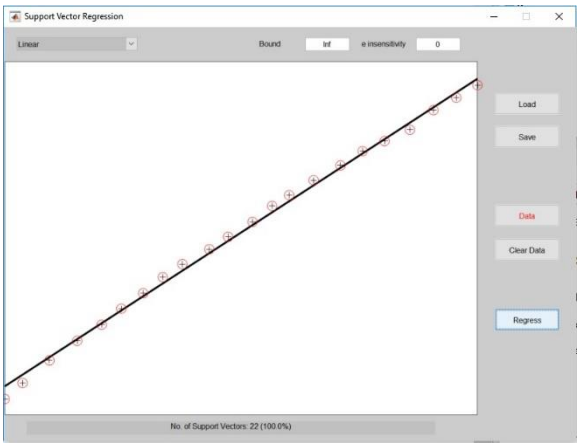


Fig-1- Regression results for linear, polynomial, RBF and linear B-Spline kernels

Figure 2 depicts a comparison of a linear and RBF kernel on approximately linear data. but for linear data with noise, a linear kernel will deliver a better performance on unseen samples, as RBF or more complicated kernels will clearly overfit the data.



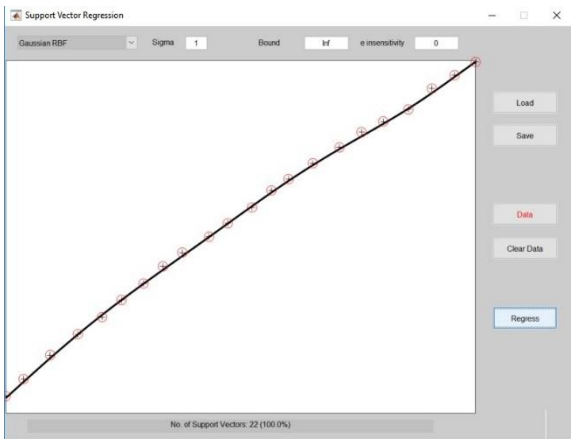
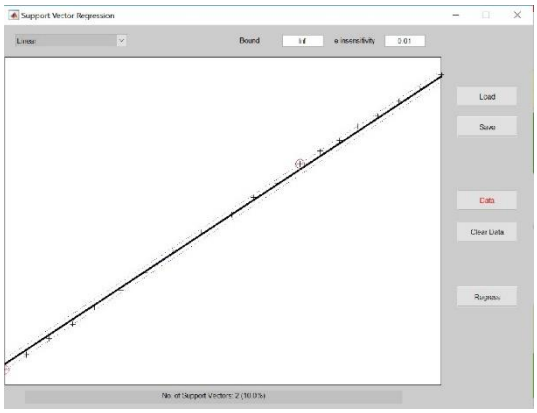
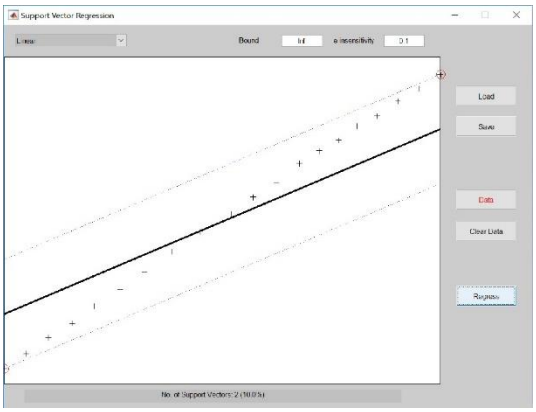
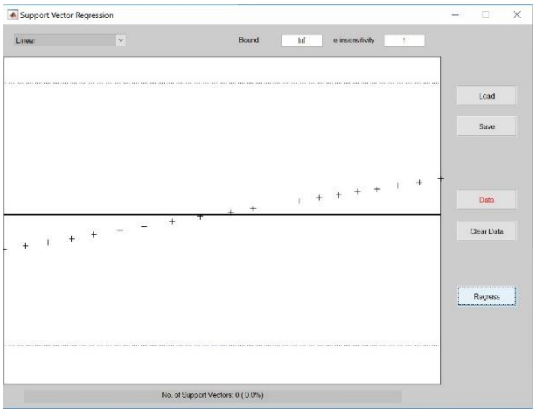


Figure 2: Comparison of linear and RBF kernels on approximately linear data

Both parameters (Bound and e) have an influence on the **flatness** of the regression curve. Figures 3 and 4 show a linear kernel on linear data for increasing values of e and Bound.







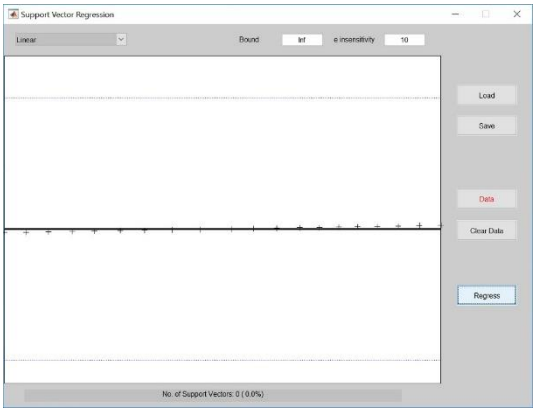


Figure 3: Linear kernel for increasing values of e, e.g. 0.01, 0.1, 1, 10 (Bound=∞)

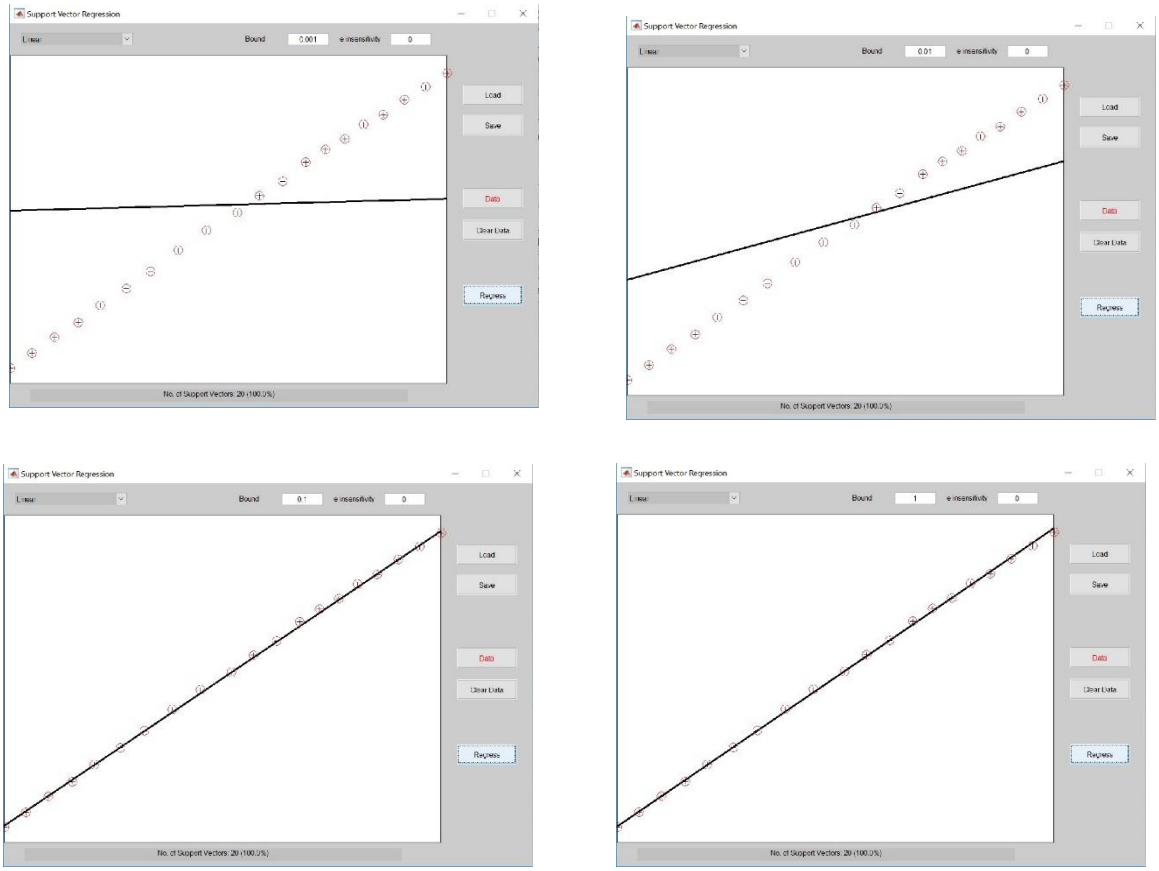


Figure 4: Linear kernel for increasing values of Bound, e.g. 0.001, 0.01, 0.1, 1 (e=0)

Parameter e controls the width of the e-insensitive zone (related to the loss function), used to fit the training data. The higher e, the higher the number of data points that have loss function equal to 0. Therefore, the value of e affects the number of support vectors used to construct the regression function (points for which loss function is not 0). The higher e, the fewer support vectors are selected. High values of e result in more flat estimates. Low values of e result in more accurate estimate on the training set. e must, therefore, be chosen to reflect the data in some way. Choosing a too low e might result in overfitting while choosing a too high e might result in a too general (too flat) model.

Parameter C (Bound) determines the tradeoff between the model complexity and the degree to which deviations larger than e are tolerated in the optimization formulation. For example, if C is too large (infinity), then the objective is to minimise the empirical error only (error on the training set), without regard to model complexity.

Sparsity comes in for non-null values of e (in the equations below). In the formulation of the optimisation problem of SVM regression, the Lagrange multipliers of the dual problem vanish (they have to equal 0 for a solution to exist, linked to the KKT conditions) for the points within the e-insensitive zone. Therefore, sparsity appears as some Lagrange multipliers equal 0 and not all points are needed to characterise the regression function. The points associated with the non-vanishing coefficients are called Support Vectors.

2.2 A Simple Example: The sinc

In this section, We simulate a cardinal sine function with noise and we train using RBF kernel in order to approximate the function. We use a training and a test set. We train the model for arbitrary values of γ (100) and σ2 (1) and visualise the results in the following figure.

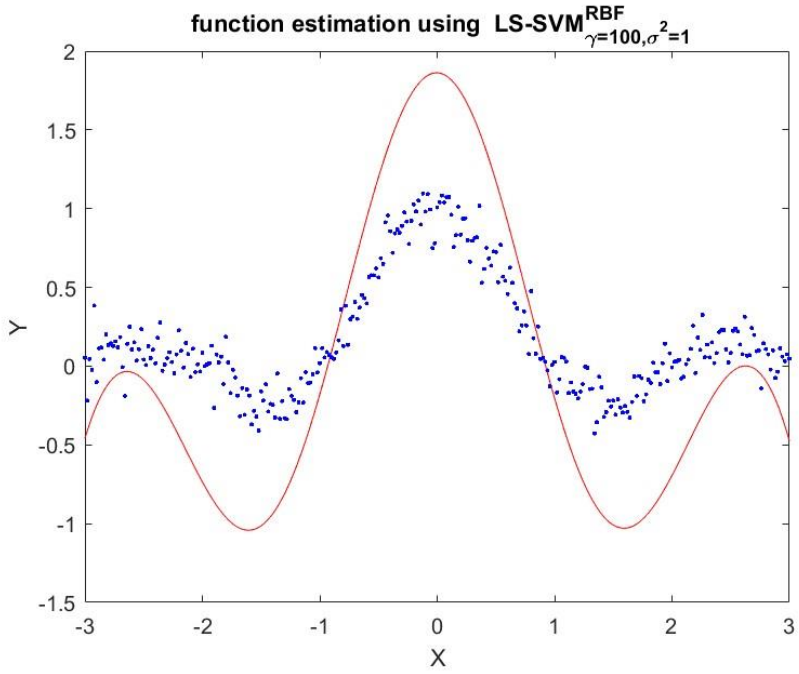


Figure 5: RBF kernel for regression on the training set

The results of applying the trained model on the test set are calculated and visualized on Figure 6.

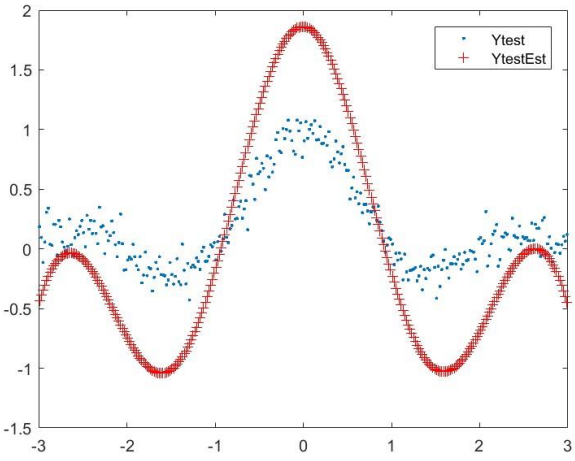
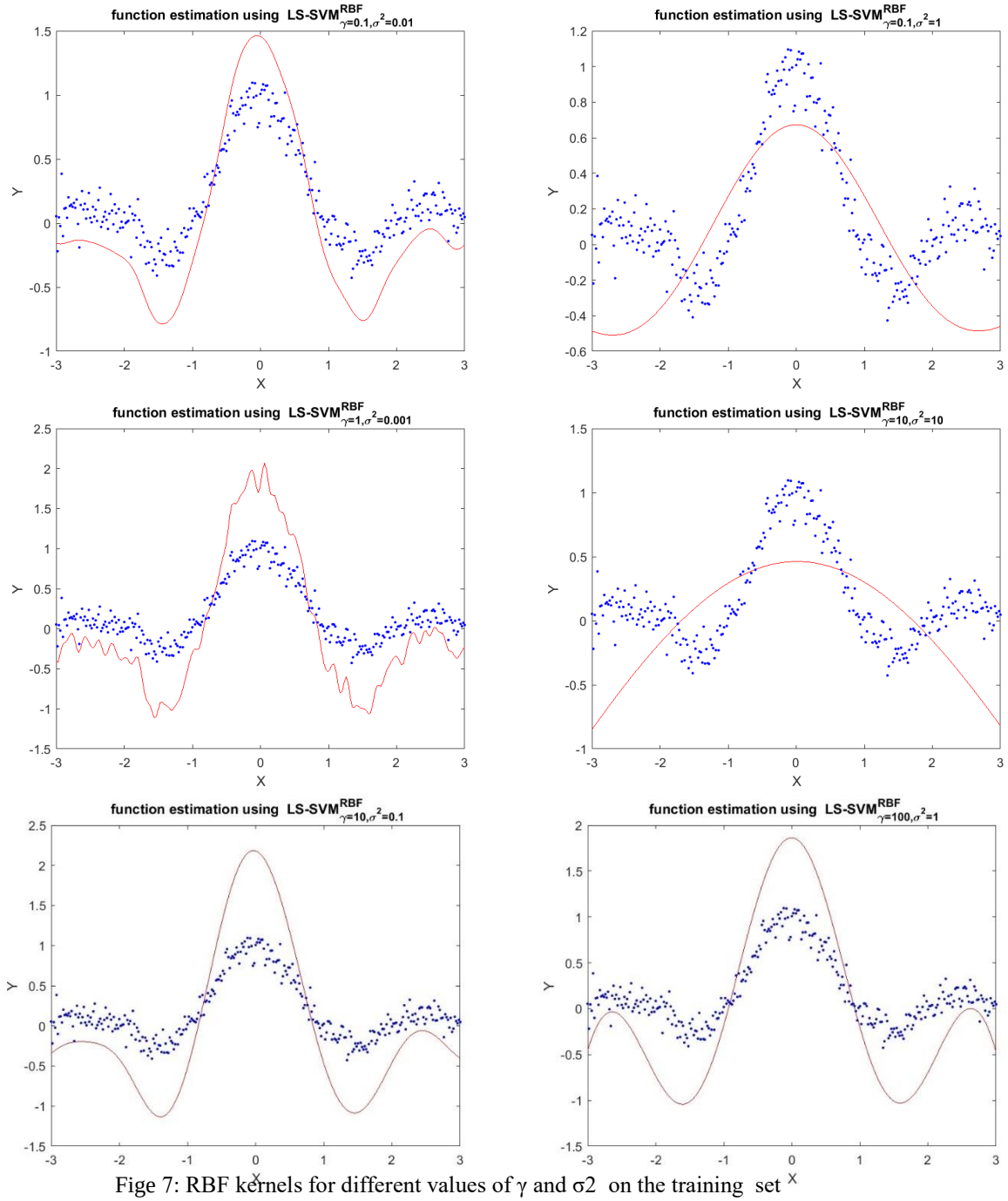


Figure 6: RBF kernel for regression on the test set

We have used arbitrary values of the hyperparameters. We also run a function estimation for a range of values for γ and σ2 and the following Figures highlights the estimated functions for 6 cases on the training and the test sets. It is obvious that it is possible to optimise the parameters according to a certain criteria so that the function estimated is the closest from the true function. However, as the hyper-parameters are continuous, there might be infinite numbers of cases leading to the similar mean squared error.



Fige 7: RBF kernels for different values of γ and σ2 on the training set

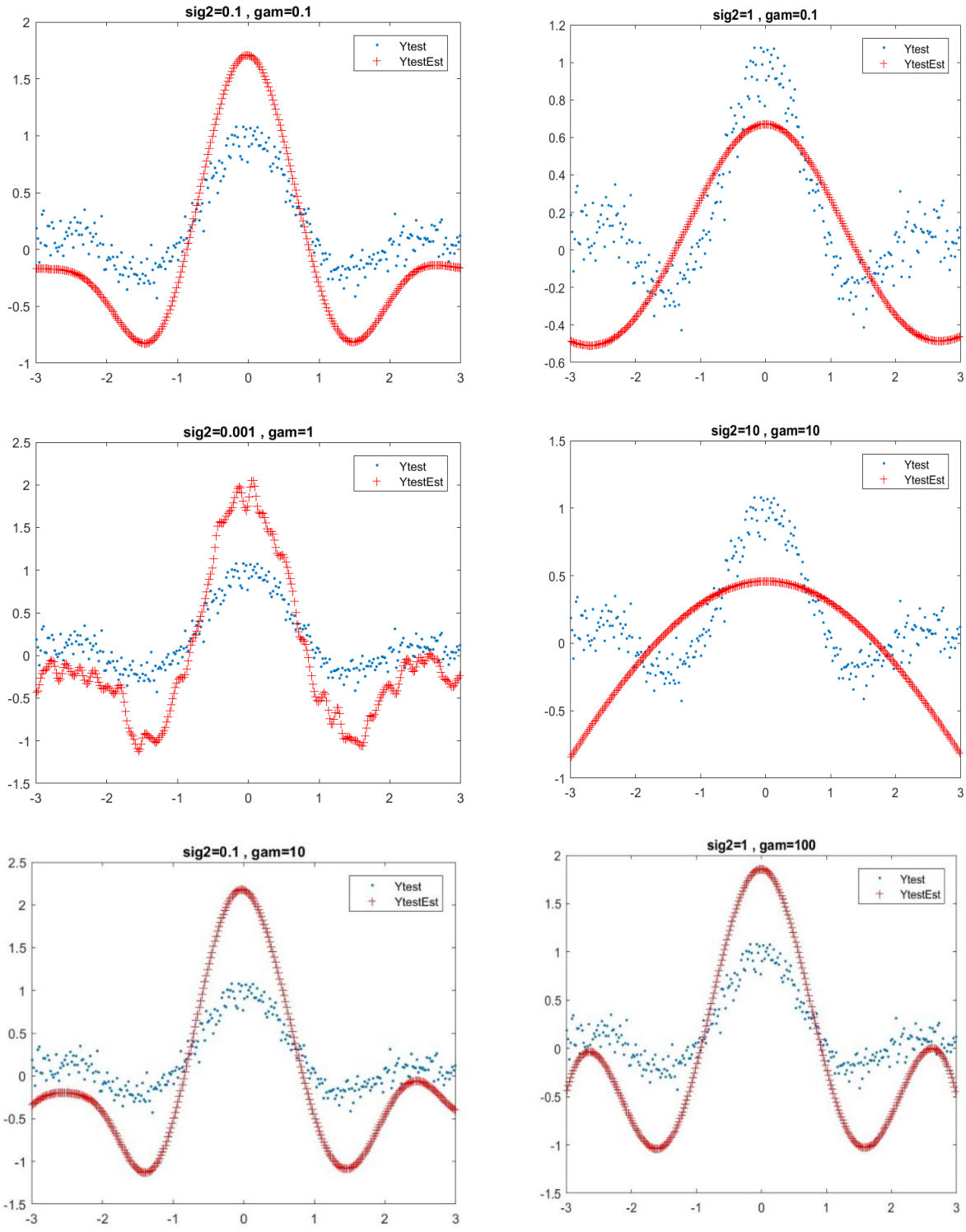


Figure 8: RBF kernels for different values of γ and σ2 on the test set

2.3 Hyper-parameter Tuning

In this section, we study the tuning algorithm used to optimise the parameters of the RBF kernel (Figure 8). We run the tuning algorithm several times and compare the obtained values of the hyper-parameters. This is depicted in Figure 5.

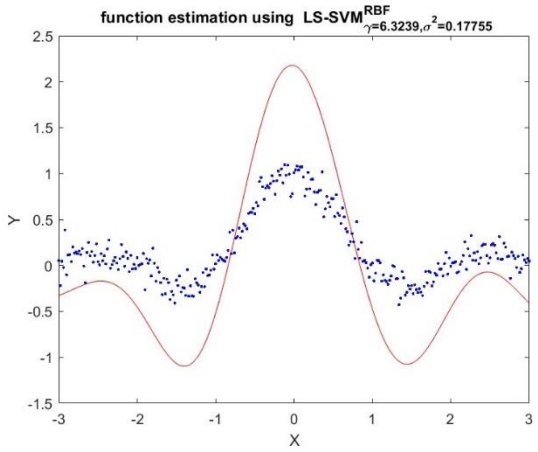


Figure 9: Optimised estimated function (grid search)

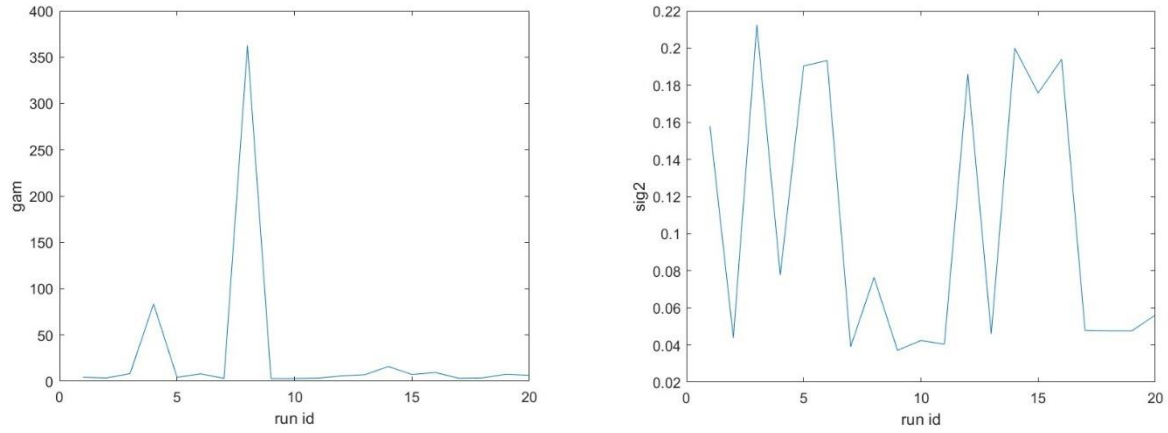


Figure 10: Optimised values of γ and σ2 when running the tuning algorithms several times (grid search)

We observe that the estimated optimal parameters change a lot from one run to another but have little influence on the cost of the hyperparameters (Figure 10). We also observe that although optimal parameters can change, the estimated regression doesn't vary a lot (Figure11).

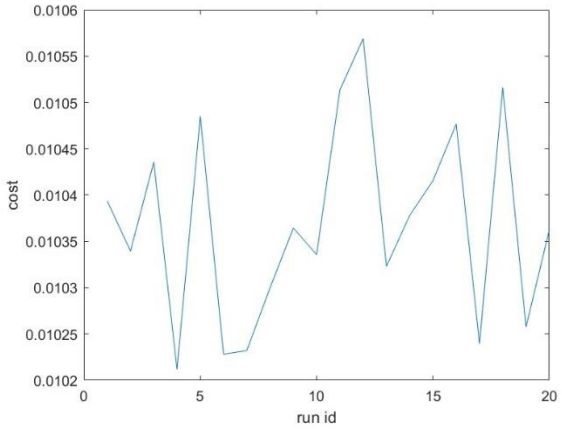


Figure 11: Cost of hyperparameters when running the tuning algorithms several times (grid search)

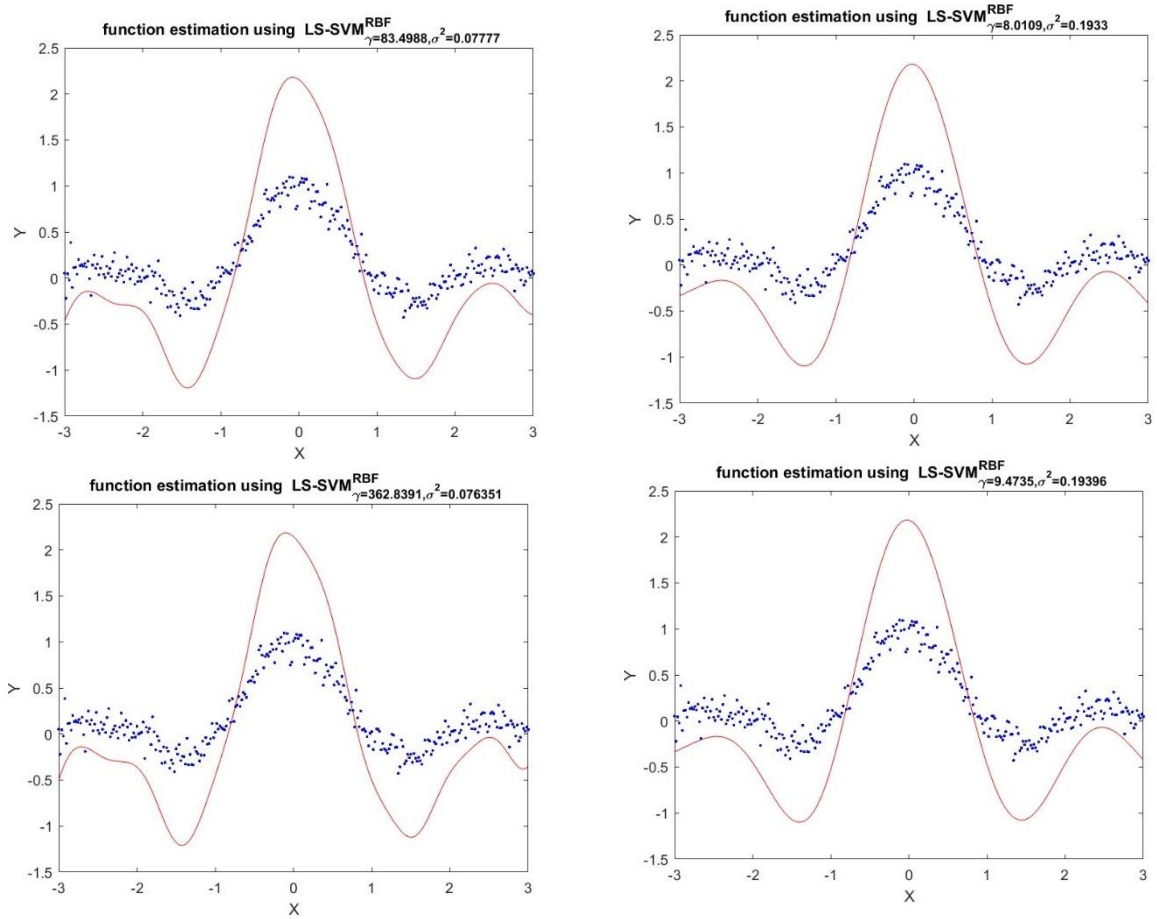


Figure 12: Estimated function when running the tuning algorithm several times (grid search)

When we change the optimisation procedure in the tuning function to simplex and re-estimate the optimal model, we obtain the following result.

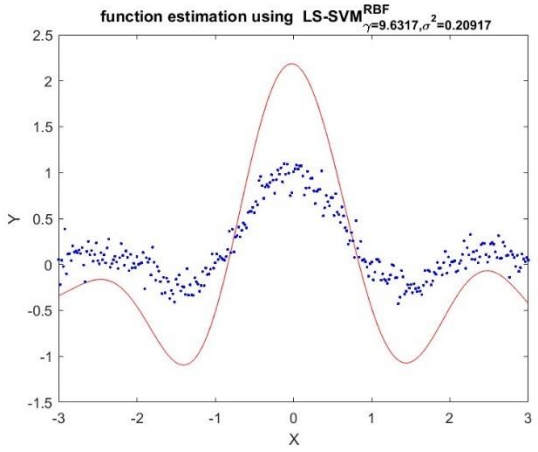


Figure 13: Optimised estimated function (simplex) Again, we run the tuning algorithm and obtain the following pictures.

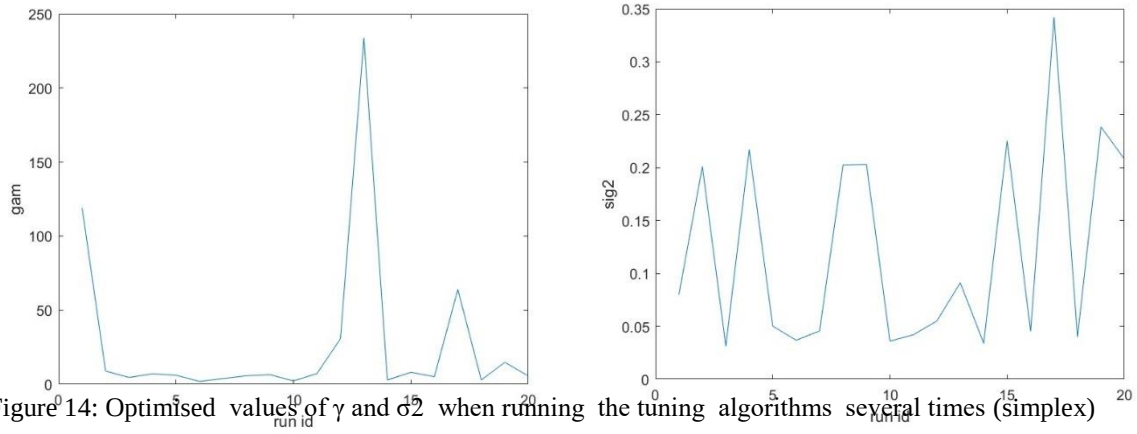


Figure 14: Optimised values of γ and σ2 when running the tuning algorithms several times (simplex)

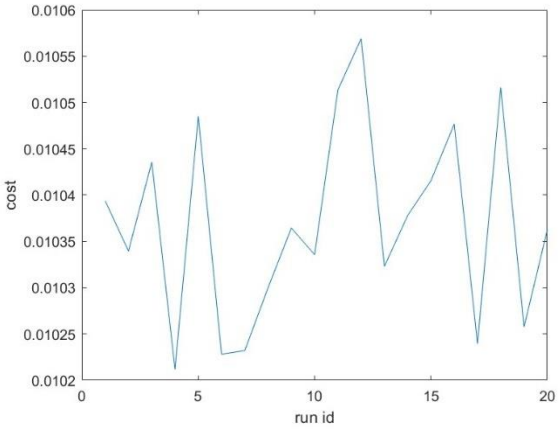


Figure 15: Cost of hyperparameters when running the tuning algorithms several times

(simplex)

Again, optimised values vary heavily, but the estimated functions are very similar. For a given starting point (calculated based on csa), grid search exhaustively considers all parameter combinations for a grid around the starting point. The start-values determine the limits of the grid over parameter space. The Nelder-Mead method or downhill simplex method is a commonly applied numerical method used to find the minimum or maximum of an objective function in a multidimensional space. It is applied to nonlinear optimisation problems for which derivatives may not be known. The method uses the concept of a simplex,

which is a special polytope of n + 1 vertices in n dimensions. The method approximates a local optimum of a problem with n variables when the objective function varies smoothly and is unimodal.

2.4 Application of the Bayesian Framework

The Bayesian framework now can be used to tune and to analyse the LS-SVM regressor. The basic result from the Bayesian framework for the LS-SVM is the derivation of the probability that the data points are generated by the given model. This is called the posterior probability. This probability criterion is expressed as a number. There are 3 variants: the posterior with respect to the model parameters α and b, the posterior with respect to the regularisation constant and the posterior with respect to the choice of the kernel and its parameter.

This three-levels principle can be schematized by the following picture:

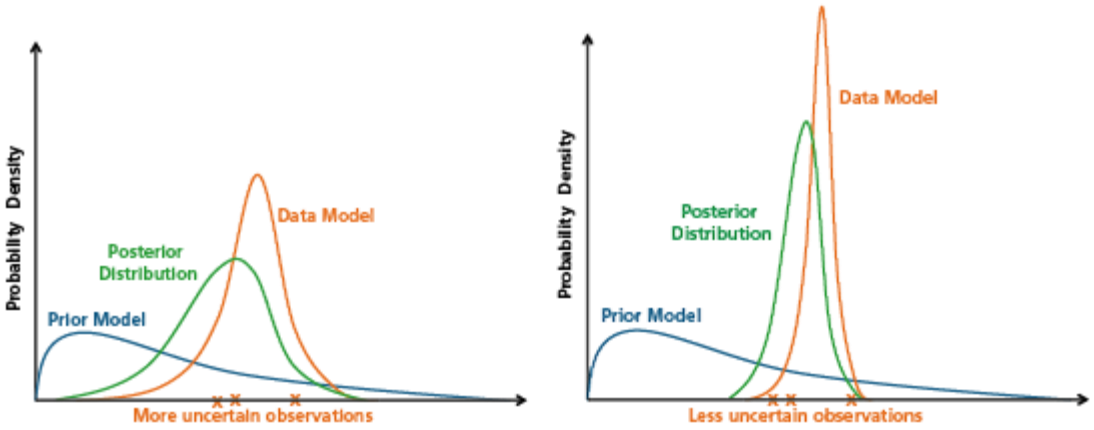


Fig-Schematic of a Bayesian "update" of a prior model from a data model describing the observations as relatively uncertain (left panel) and relatively certain (right panel), resulting in posterior distributions with more and less uncertainty, respectively.

Provided data set D, a model H with parameter vector w and so-called hyper-parameters or regularisation parameters θ, Bayesian inference is constructed with 3 levels of inference:

• In level 1, for a given value of θ, the first level of inference infers the posterior distribution of w by the Bayesian rule:

$$p(w \mid D, \theta, H) \propto p(D \mid w, H)p(w \mid \theta, H) \qquad (5)$$

• The second level of inference determines the value of θ, by maximising

$$p(\theta \mid D, H) \propto p(D \mid \theta, H)p(\theta \mid H) \qquad (6)$$

• The third level of inference in the evidence framework ranks different models by examining their posterior probabilities:

$$p(H \mid D) \propto p(D \mid H)p(H) \qquad (7)$$

For regression, the error bars can be computed using Bayesian inference (see Figure 17)

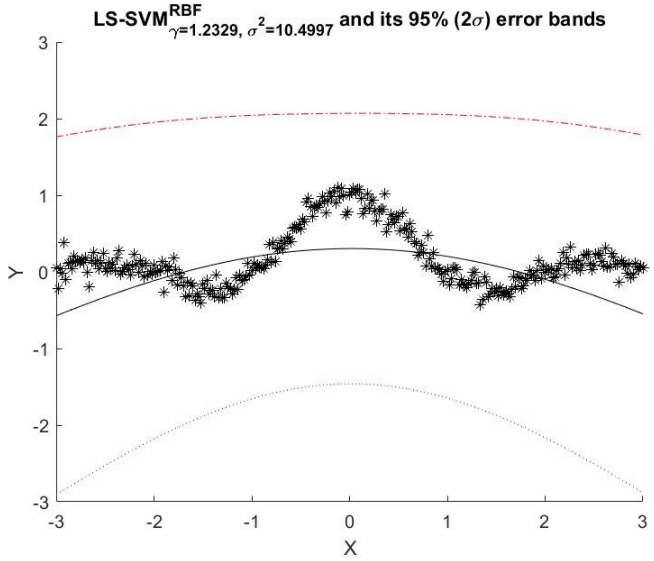


Figure 17: Error bars for Bayesian inference

For classification, it is also possible to get probability estimates. We load the Iris dataset and calculate the posterior class probabilities. Results are displayed in the following figure:

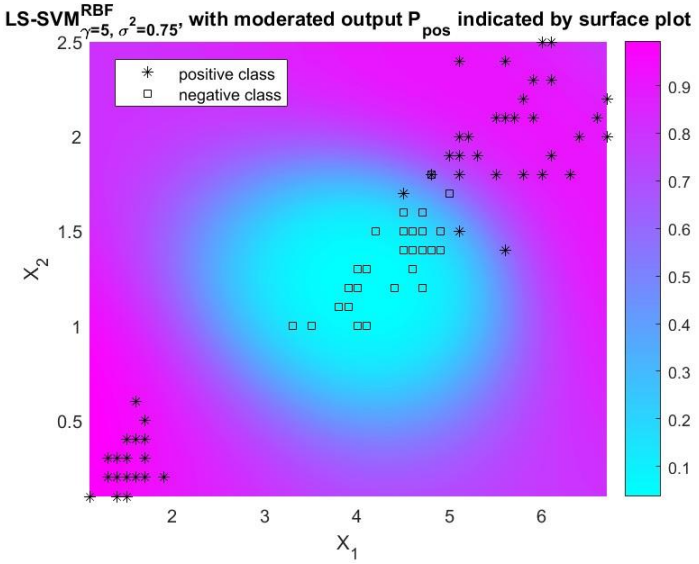
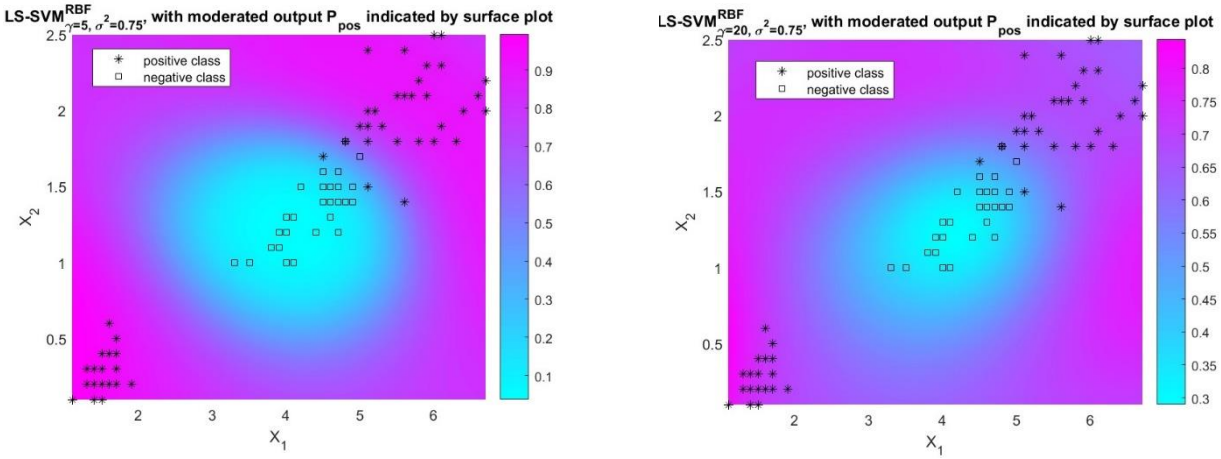


Figure 18: Posterior class probabilities



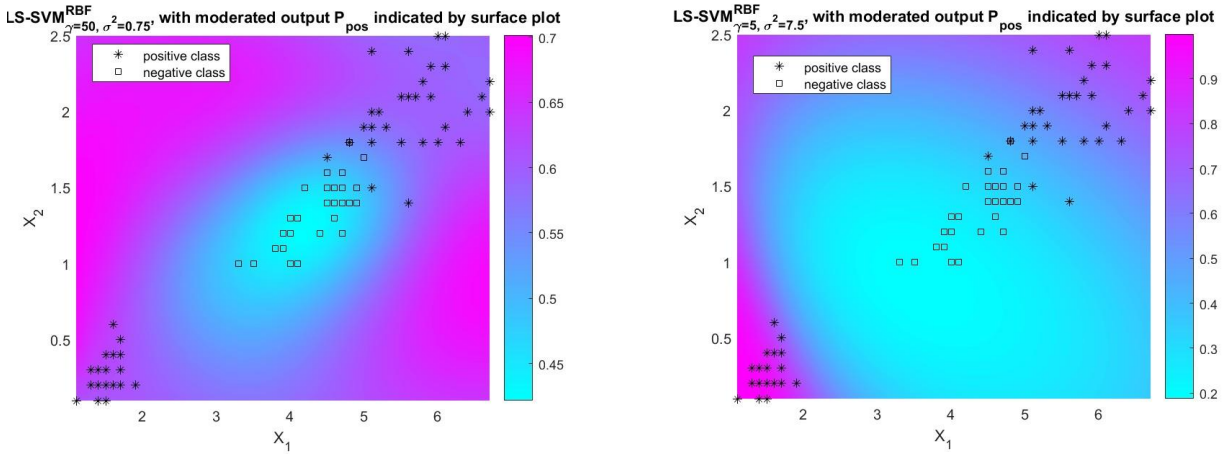


Figure 19: Posterior class probabilities for different values of the hyper-parameters

Colours represent the probability that a point belongs to the positive class. Figure 19 represents the same probabilities when we change the values of the hyper-parameters. Obviously, for different values, the probabilities change. As the values move away from the optimal values, we can observe that the obtained posterior probabilities don't match the empirical labels.

The Bayesian framework can also be used to select the most relevant inputs by Automatic Relevance Determination (ARD). In a backwards variable selection, the third level of inference of the Bayesian framework is used to infer the most relevant inputs of the problem. Only the most explanatory variables will be selected.

ARD assigns a different weighting parameter to each dimension in the kernel and optimises this using the third level of inference. Per the used kernel, one can remove inputs corresponding the larger or smaller kernel parameters. In each step, the input with the largest optimal σ2 is removed (backwards selection). For every step, the generalisation performance is approximated by the cost associated with the third level of Bayesian inference.

An easy way to represent the results is to plot the cost associated with the third level of inference for each feature (Figure 20). Features 1 and 3 have the higher cost (the generalisation performance is approximated by the cost associated with the third level of Bayesian inference.) and are therefore selected.

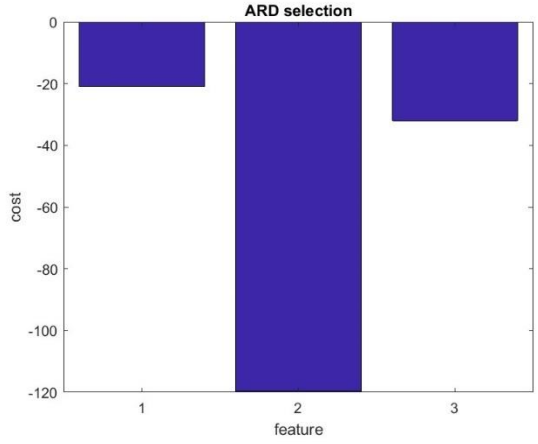


Figure 20: ARD selection

Cross-validation can be used instead of the Automatic Relevance Determination. Selecting each possible combination of variables, tuning the hyper-parameters and calculating the cost function of the cross-validation procedure leads to the same result (selecting the number of variables that correspond to the lowest cost function of the cross-validation procedure). To fully mimic the backwards selection procedure, we can start with all possible variables, exclude sequentially only one variable, select the model that correspond to the lowest cost and start again the backwards procedure.

2.5 Robust Regression

In this section, we study the impact of outliers on the regression. In situations where the data is corrupted with non-Gaussian noise or outliers, it becomes important to incorporate robustness into the estimation.

Figure 21 highlights the impact of outliers on the estimation of the regression.

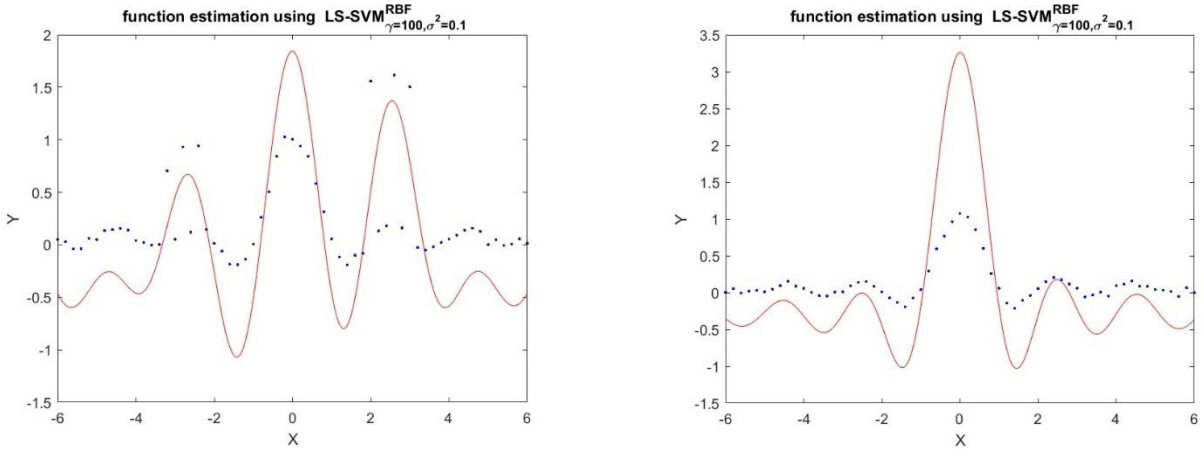


Figure 21: Regression with outliers (left) and without (right)

Outliers impact the regression locally (where outliers are). Learning observations with outliers without awareness may lead to fitting those unwanted data and may corrupt the approximation function. This will result in the loss of generalisation performance in the test phase.

Outliers are dealt with in SVM regression by using robust cross-validation and smoothing the impact of outliers. Figure 22 shows the result of a robust cross-validation and the use of a smoothing function for the outliers. We can see that the outliers don't impact the regression anymore (or at least to a lesser extent).

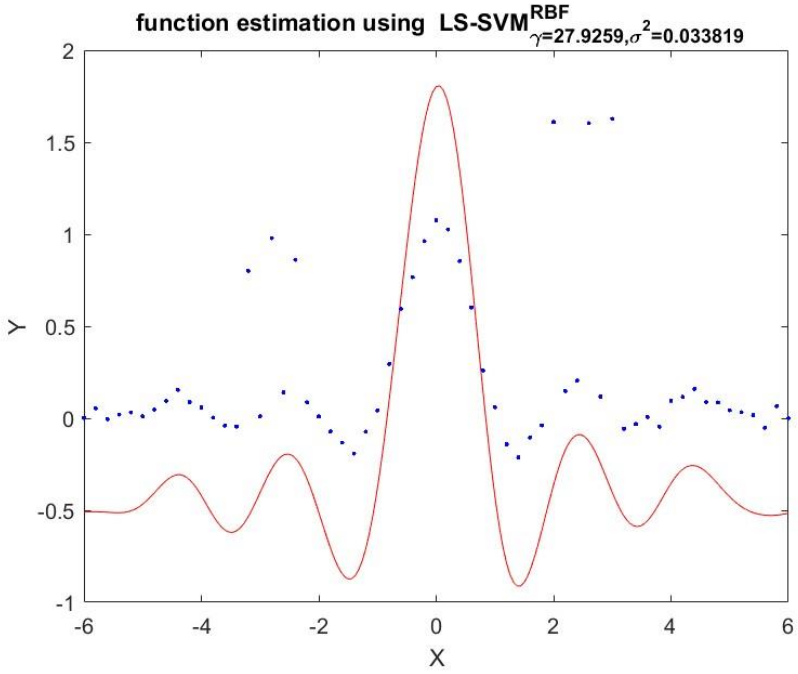


Figure 22: Robust cross-validation and outliers smoothing

Both mean squared error (MSE) and mean absolute error (MAE) are used in predictive modelling. MSE has nice mathematical properties which make it easier to compute the gradient. However, MAE requires more complicated tools such as linear programming to compute the gradient. Because of the square, large errors have a relatively greater influence on MSE than do the smaller error. Therefore, MAE is more robust to outliers since it does not make use of square.

Different weighting functions can be used in the robust regression. Figure 23 describes the four weighting functions available with the tuning algorithm for robust regression. As an example, the weighting function Hampel completely ignores points further than a certain threshold (b2).

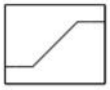
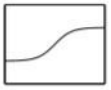
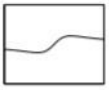
	Huber	Hampel	Logistic	Myriad
$V(r)$	$\begin{cases} 1, & \text{if } \lvert r\rvert < \beta; \\ \frac{\beta}{\lvert r\rvert}, & \text{if } \lvert r\rvert \geq \beta. \end{cases}$	$\begin{cases} 1, & \text{if } \lvert r\rvert < b_1; \\ \frac{b_2 - \lvert r\rvert}{b_2 - b_1}, & \text{if } b_1 \leq \lvert r\rvert \leq b_2; \\ 0, & \text{if } \lvert r\rvert > b_2. \end{cases}$	$\frac{\tanh(r)}{r}$	$\frac{\delta^2}{\delta^2 + r^2}$
$\psi(r)$				
$L(r)$	$\begin{cases} r^2, & \text{if } \lvert r\rvert < \beta; \\ \beta\lvert r\rvert - \frac{1}{2}\beta^2, & \text{if } \lvert r\rvert \geq \beta. \end{cases}$	$\begin{cases} r^2, & \text{if } \lvert r\rvert < b_1; \\ \frac{b_2 r^2 - \lvert r^3\rvert}{b_2 - b_1}, & \text{if } b_1 \leq \lvert r\rvert \leq b_2; \\ 0, & \text{if } \lvert r\rvert > b_2. \end{cases}$	$r\tanh(r)$	$\log(\delta^2 + r^2)$

Figure 23: Weighting functions for robust regression

The following figure represents the results of different weighting functions for robust regression (top-left = Huber, top-right = Hampel, bottom-left = Logistic, bottom-right = Myriad). Comparison of the results of the different weighting functions is made difficult due to the instability of the tuning algorithm (optimal values are different every time).

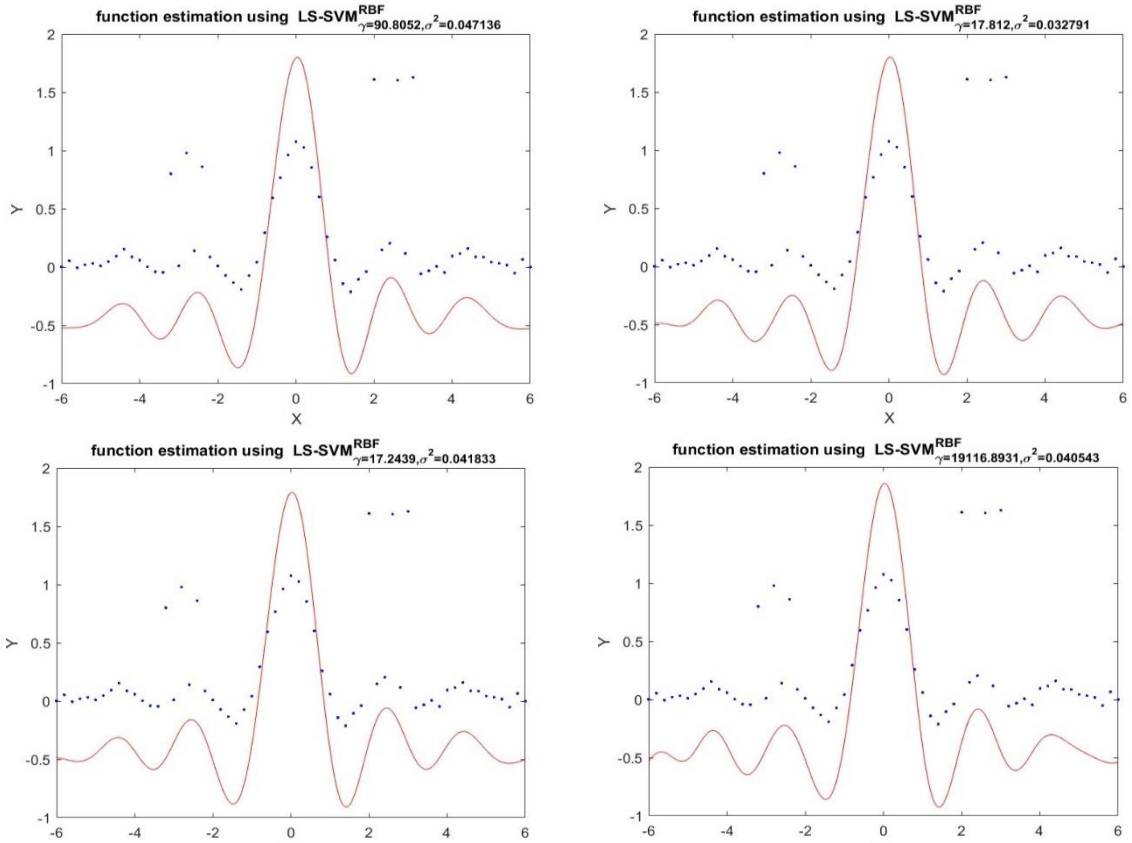


Figure 24:Results of different weighting functions for robust regression

2.6 Homework Problem

2.6.1 Introduction: Time-series Prediction

In this section, we investigate the use of SVM regression for time series model. The linear auto-regressive (AR) model of a process Zt with t = 1, 2, ..., ∞ is given by

$$\tilde{z}t = a1zt-1 + a2zt-2 + ... + anzt-n \quad (8)$$

with ai ∈ R for i = 1, ..., n and n the model order. The nonlinear variant (NAR) is described as:

$$\tilde{z}t = f(zt-1, zt-2, ..., zt-n) \quad (9)$$

The time-series identification can be written as a classical black-box regression modelling problem:

$$\tilde{y}t = f(xt) \quad (10)$$

with yt = zt and xt = [zt−1, zt−2, ..., zt−n].

Any provided sequence Z can be mapped into a regression problem. By using the command window, we can transform the log map data into the required format.

A very important prerequisite is to normalise the data. Support vector machines usually work best with normalised data (like other machine learning techniques). After training an initial model with random values for the hyperparameters, we optimise the hyper-parameters by using the cross-validate approach. Figure 25 displays the result of the prediction for the test set with order = 10. Like the hyper-parameters, the order can be optimised to minimise the mean squared error between the prediction and the test set. We run the above procedure for a range of order from 1 to 15 and calculate the mean squared error for each order. The mean squared error as a function of order and the prediction for order = 7 (minimising MSE) is displayed in Figure 26. It is obvious that the order of the model has an impact on the quality of the prediction (also visually on the prediction itself, not displayed here).

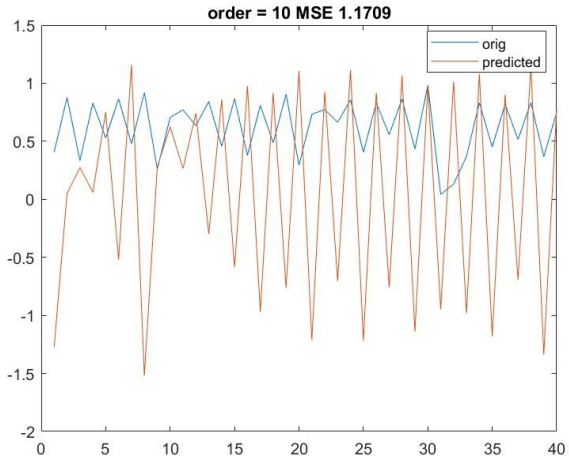


Figure 25: Prediction on the test set for order = 10

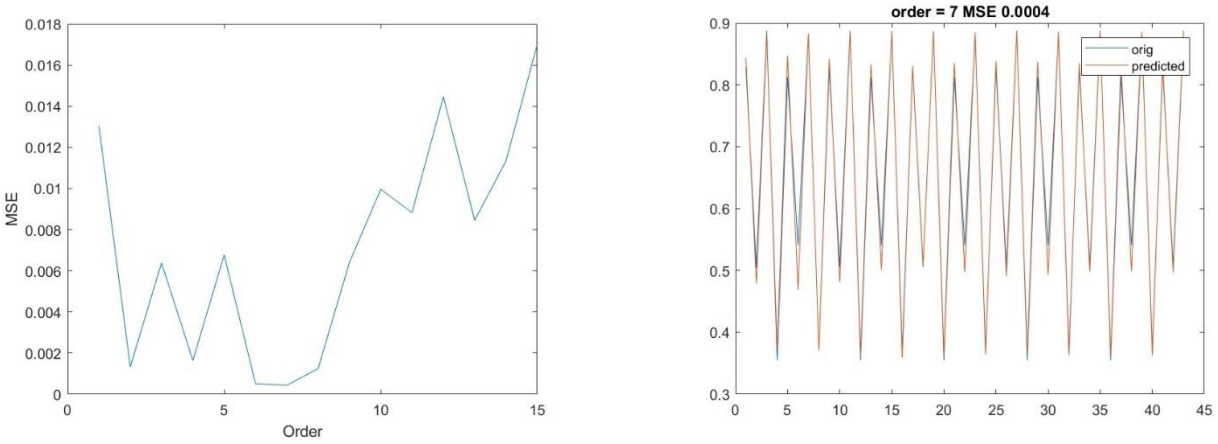


Figure 26: MSE as a function of order and prediction on the test set for order = 7

2.6.2 Application: Santa Fe Laser Dataset

The Santa Fe Laser dataset is shown in Figure 29. The training set has 1000 points and the test set has 200 points.

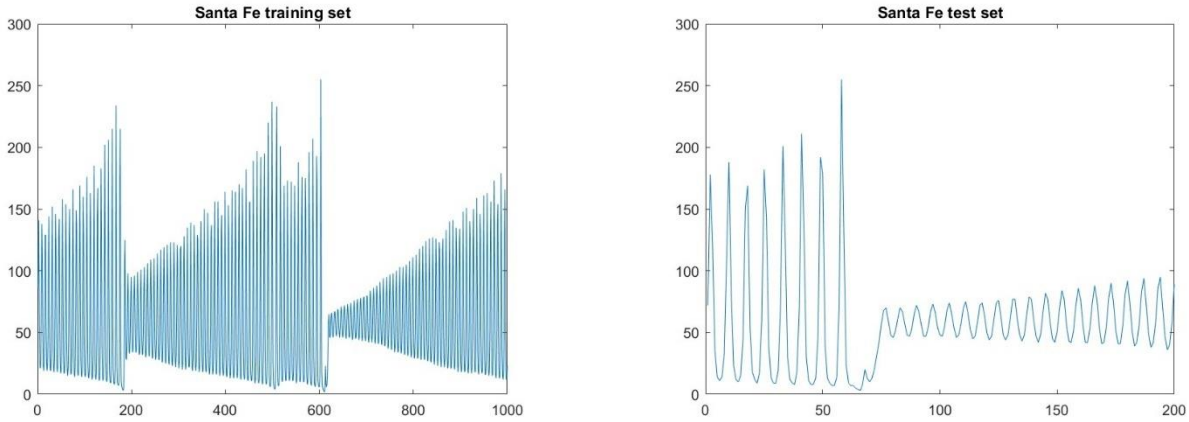


Figure 27: Santa Fe data set

First, we try to fit a model with order = 50. Again, we tune the parameters (γ and $\sigma 2$) on the training set. The results are displayed in Figure 28. Prediction with order = 50 doesn't capture the change of amplitude that occurs around point 15. From point 60, we also see that oscillations of the test set and the prediction are not in phase anymore. Therefore, order = 50 is not a good choice for the model.

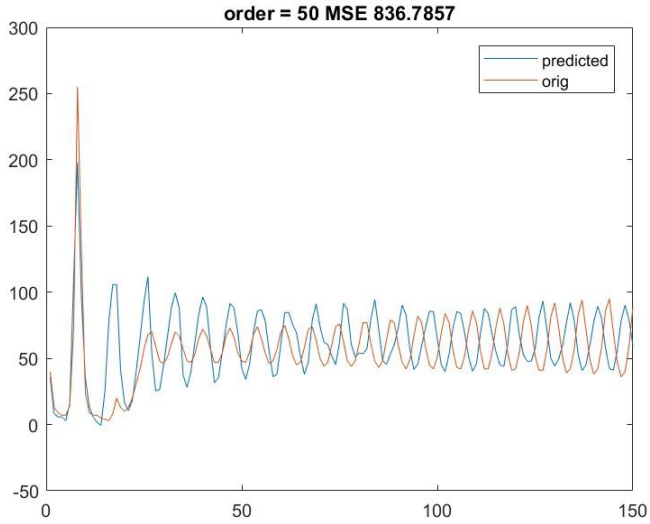


Figure 28: Prediction on the test set for order = 50

In the next step, we try different model orders on the test set and observe the mean squared error (MSE). The results are plotted in Figure 29. The best results are given for an order = 30. The difficulty with the test set is that around point 60, the amplitude of the oscillations change suddenly and it is difficult for the trained model to predict this change. We also noticed that the results of the tuning of the hyper-parameters are not very stable.

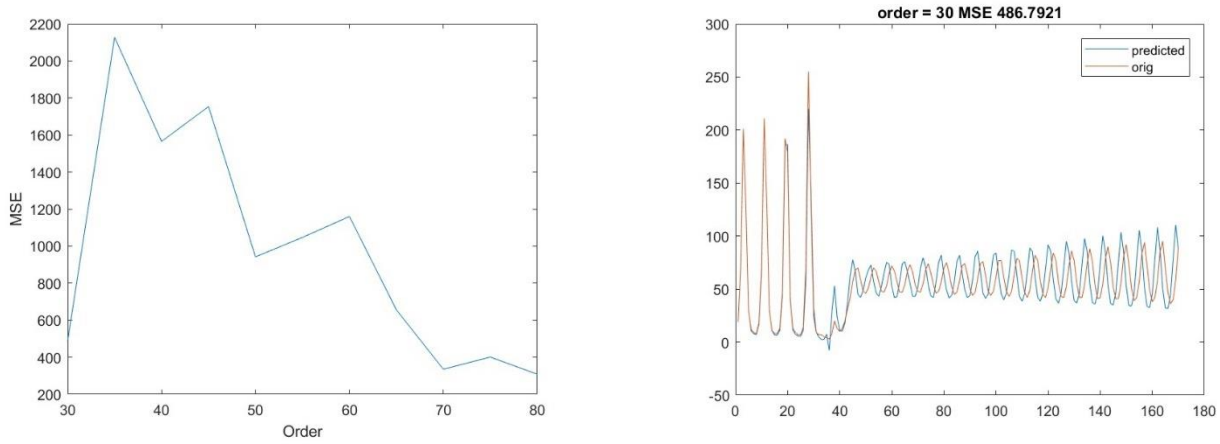


Figure 29: Results of prediction for order = 30 and MSE as a function of order

Other order (70, 80) gave interesting results, but the interpretation is flawed since, for such orders, the sudden change of behaviour lies then in the test sample to predict and is not part of the prediction.

It is not correct to use this test set to choose the model order and report the error on the same set. A separate validation set should be extracted from the training set and could be used to find the optimal model order. Alternatively, Bayesian ARD could be used to determine the order and select the most relevant inputs.

3. Unsupervised learning

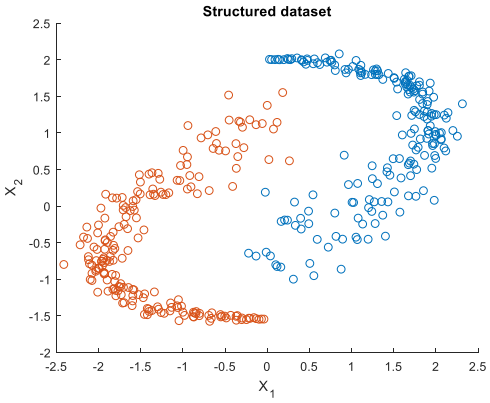
3.1 Kernel Principal Component Analysis:

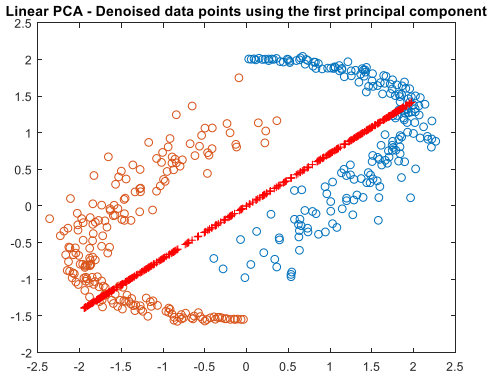
Kernel PCA is a dimensionality reduction technique which corresponds to linear PCA in a kernel-induced feature space and is non-linearly related to the original input space. It does so by mapping the data into a higher-dimensional space. Though it looks contradictory to its purpose, the data points are mapped into a higher-dimensional space, then turn out to lie on a lower dimensional subspace of it. So it increases the dimensionality in order to be able to decrease it. Kernel PCA can be used for feature extraction, denoising, dimensionality reduction and density estimation.

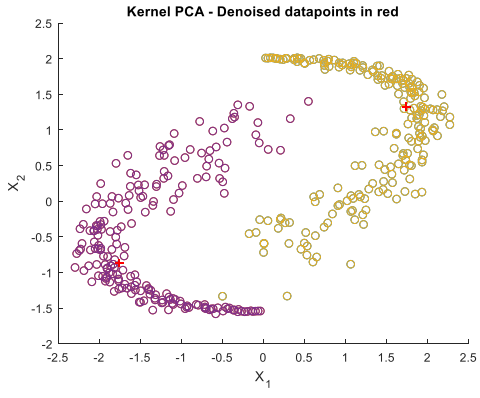
In this section, a toy dataset is used to get insight in the number of components, the choice of the kernel and the kernel hyperparameter. Linear PCA is used to reveal the internal structure of the data in a way that best explains the variance in the data. If a multivariate data set is visualised as a set of coordinates in a high-dimensional data space (1 axis per variable), PCA can supply the user with a lower-dimensional picture, a projection of this object when viewed from its most informative viewpoint. This is done by using only the first few principal components so that the dimensionality of the transformed data is reduced though at decreased information from the original data.

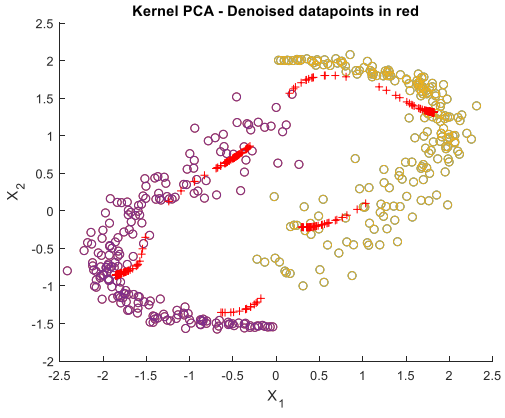
In kernel PCA, the original input space is mapped to a high dimensional features space (*N*-dimensional feature space, where *N* is the number of observations). Therefore, the number of principal components obtained in kernel PCA is higher than in the linear PCA. The difference between both is that in the case of the kernel PCA, the dimension reduction is applied on the kernel matrix (covariance matrix of the data after being transformed into a higher-dimensional space), while in linear PCA, it is applied to the covariance matrix. In linear PCA, the number of principal components is limited to the number of dimensions of the original data. The following figure 1 shows the results of denoising on the toy data set for varying number of components.

As one can note, the more components are kept, the closer the denoised data set will be from the original data set. With threshold (for example 95%) based on which we retain the number of components reaching this threshold in terms of variance is a possibility. Another possibility is to use the concept of scree plot. A scree plot displays the eigenvalues associated with a component in descending order versus the number of the component. The ideal pattern in a scree plot is a steep curve, followed by a bend and then a flat or horizontal line. The idea is then to retain those components in the steep curve before the first point that starts a flat line trend. To tune the hyper-parameter, one can apply cross-validation based on reconstruction error.









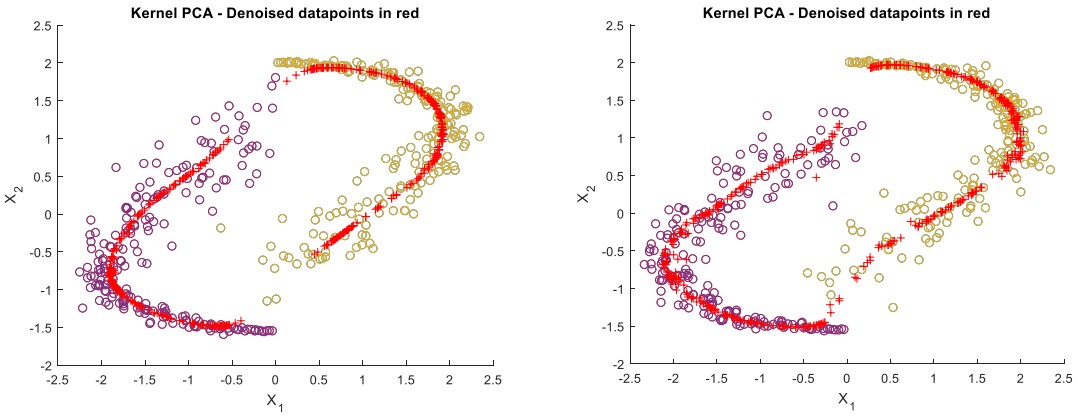


Fig-1.Plot for denoising deigits with different number of principal components (1,2,4,10)

3.2 Handdigits recognition:

In this section, the script '**digitsn'** is used to show how the noise filled data can be retrieved with the help of Linear and Kernel PCA's.Denoising/Noise reduction is the process of removing noise from a signal which is handwritten digit images. Figure 2 shows the results of reconstructing pre-images from noisy images, by applying kernel PCA or linear PCA. Model is first trained on a sample without noise and then applied to noisy images.

Let

$$\{\mathbf{x}j \in \mathrm{R}m, \textit{whereas } j = 1, ..., n\} \text{ be a data set.}$$

To state the result, the principal components are given by,

$$\mathbf{v}_i = \sum_{j=1}^{n} \mathbf{a}_{i,j} \Phi(\mathbf{x}_j)$$

where **a***i* are the eigenvectors of the kernel matrix *K*, given by *Kij* = *k*(**x***i*, **x***j*), sorted in order of decreasing the corresponding eigenvalue.To project onto the principal components, a projection **Pq** is defined by,

$$\mathbf{P}_\mathbf{q}\Phi(\mathbf{x}) = \sum_{j=1}^{q} \beta_i \mathbf{v}_i,$$

$\beta i = \mathbf{v}i.\Phi(\mathbf{x})$ are the projections of $\Phi(\mathbf{x})$ onto the principal components. Denoising requires optimal reconstruction **Pq**_ Φ (**x** in input space R*m*. The idea is to find an approximate pre-image **˜x** in input space that will map closest to the projection **Pq**_ Φ (**x**) in feature space:

$$\tilde{\mathbf{x}} = \arg\min_{\mathbf{x}' \in \mathbb{R}^n} \| \Phi(\mathbf{x}') - \mathbf{P}_\mathbf{q}\Phi(\mathbf{x}) \|^2$$

We can see in Figure 2 that the quality of the denoising increases with the number of principal components. It is also clear that kernel PCA performs much better than linear PCA.

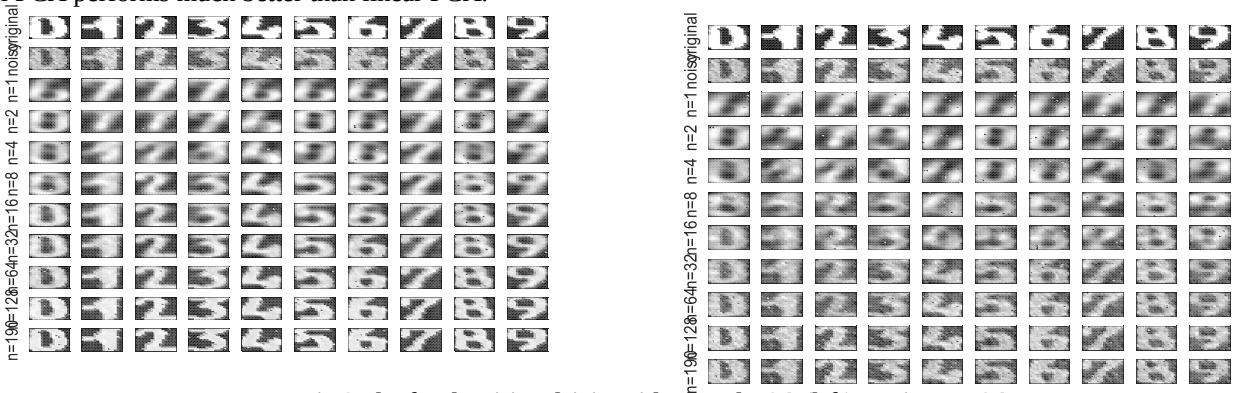


Fig-2.Plot for denoising deigits with Kernel PCA (left) vs Linear PCA

3.3 Spectral Clustering

Spectral clustering techniques make use of the spectrum (eigenvalues) of the similarity matrix of the data to perform dimensionality reduction before clustering in fewer dimensions. Spectral clustering techniques make use of the eigenvectors of a Laplacian matrix derived from the data to create groups of data points that are similar. In this context, the kernel function acts as a similarity measure between two data points. The Laplacian matrix is then obtained by re-scaling the kernel matrix.

We run the script *sclusteringscript* and obtain the following figures: They differ with respect to the different chosen sigma factors.

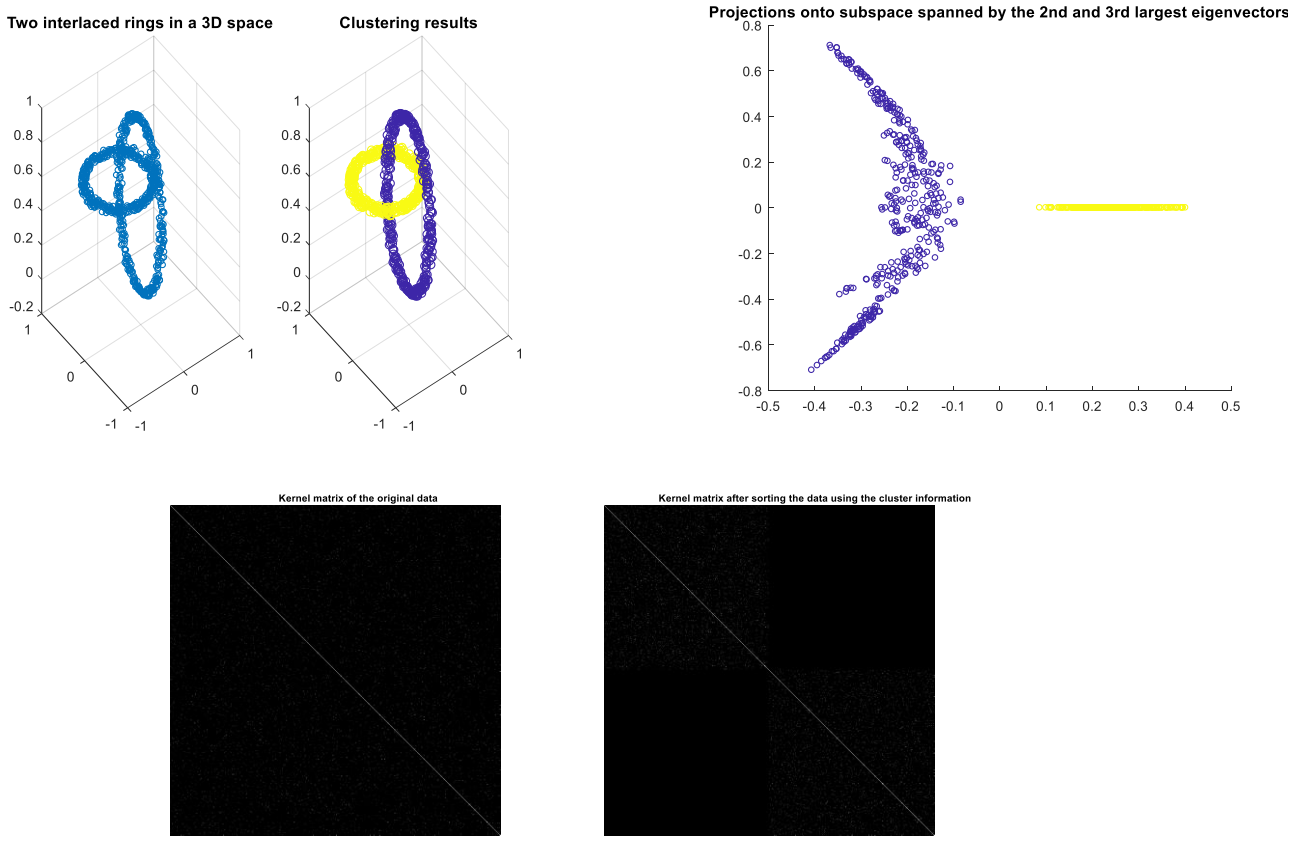
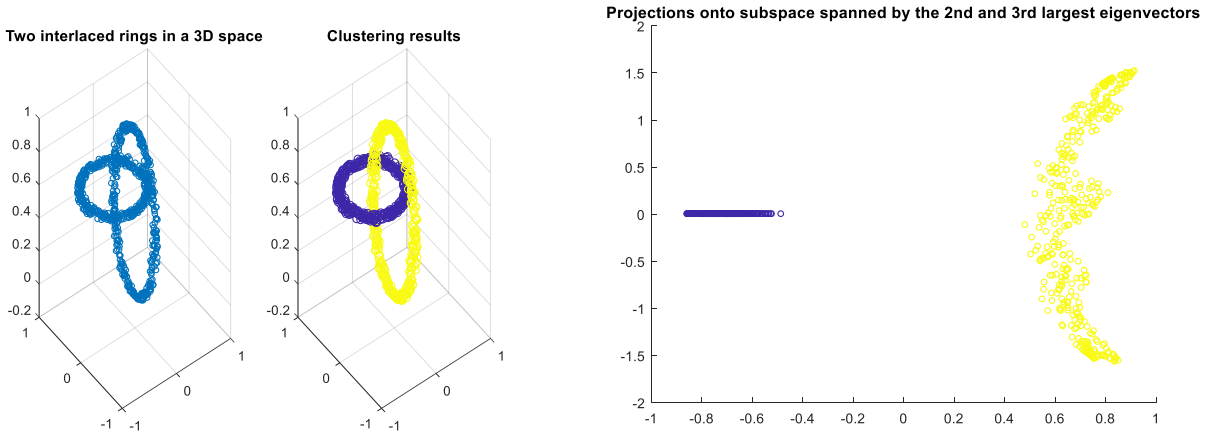


Fig-3. Plots for Sigma factor 0.001



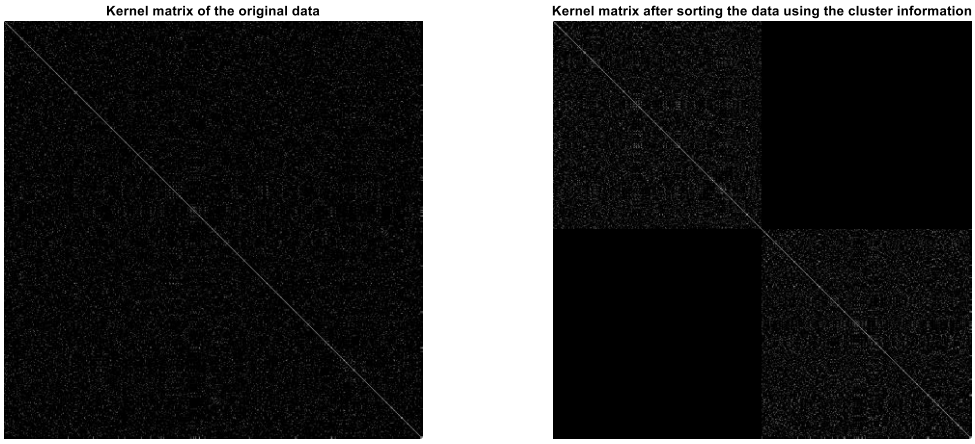
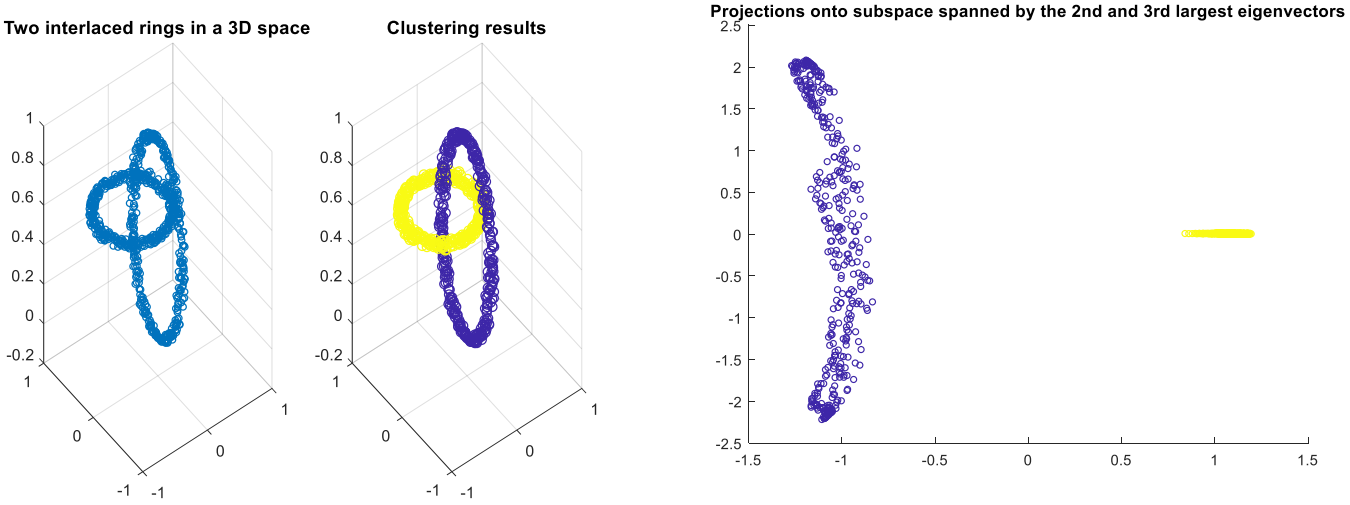


Fig-4. Plots for Sigma factor 0.005



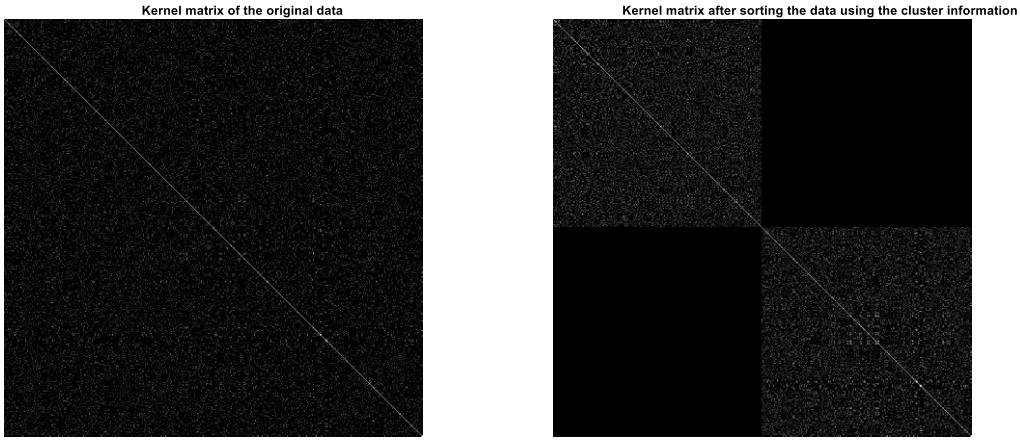


Fig-5. Plots for Sigma factor 0.01

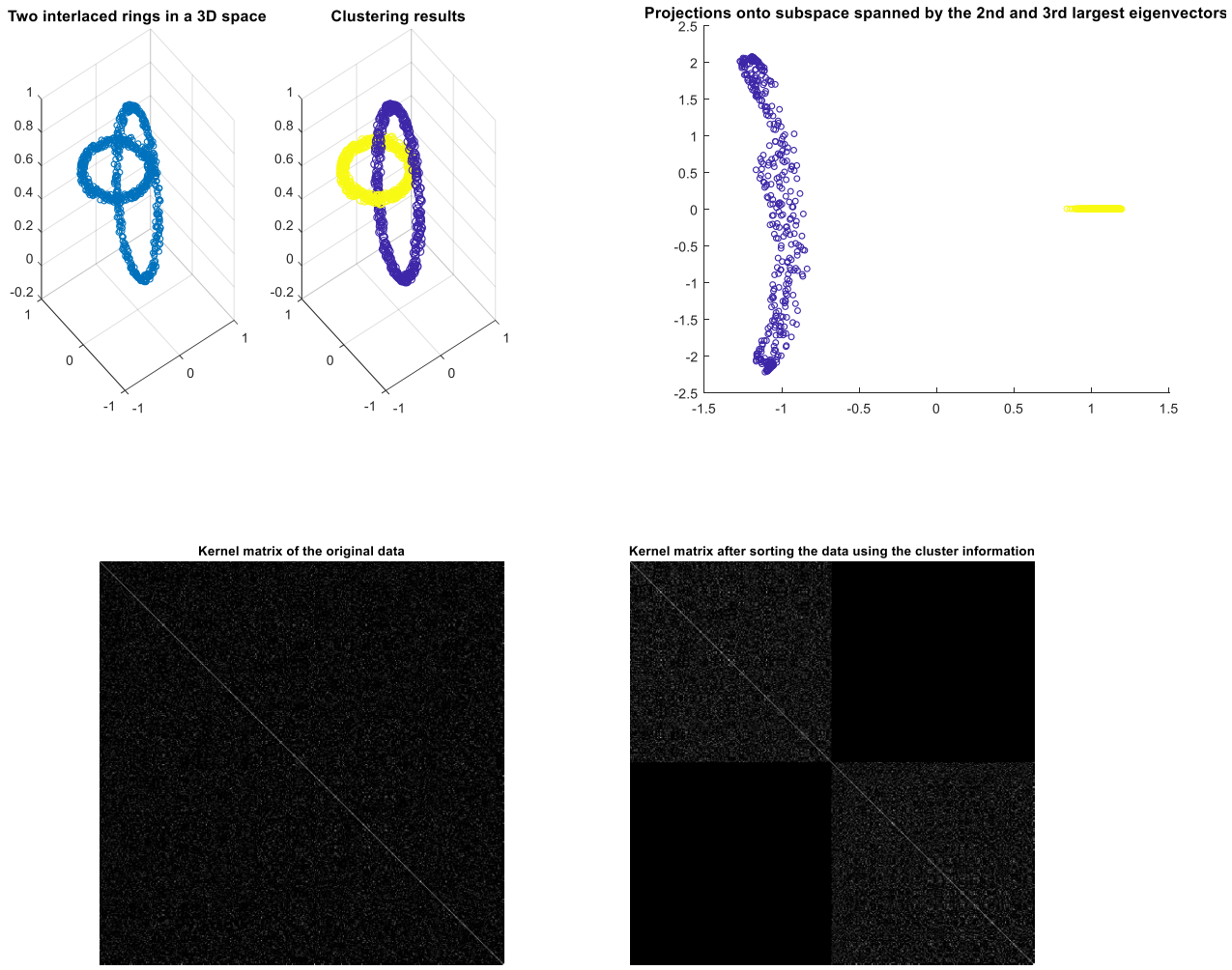


Fig-6. Plots for Sigma factor 0.2

Clustering, an unsupervised learning technique differs with classification by not having/using labels. The different with standard clustering is that, as mentioned above, spectral clustering makes use of the eigenvectors of a similarity matrix instead of the similarity matrix itself (k-means for example).

In spectral clustering with RBF kernel, σ denotes the bandwidth of the kernel. As in other kernel applications, the tuning of the parameters of the kernel is an important component to the performance of the model. A cross-validation approach can be used here.

A model is trained for each combination of parameters and a chosen criterion is evaluated on the partitioning predicted for the validation data. Finally, the parameters yielding the maximum value of the criterion are selected. In the kernel, spectral clustering, a criterion based on the amount of collinearity between validation points belonging to the same cluster, in the space of the projections performs well. It reaches its maximum value 1 in the case of well-separated clusters.

3.4. Fixed-Size LS-SVM

The LS-SVM formulation solves a linear system under a least squares cost function. Although the LS-SVM dual formulation is quite advantageous when working with large dimensional input spaces, or when the dimension of the input space is larger than the sample size, it requires the resolution of a linear system with as many unknowns as the number of data points, N. This situation has an obvious drawback when N is too large, and in such case the direct application of this method becomes prohibitive. However, the primal-dual structure of the LS-SVM can be exploited further. It is possible to compute an approximation of the nonlinear mapping _ to perform the estimations directly in primal space; furthermore, it is possible to compute a sparse approximation by using only a subsample of selected Support Vectors from the dataset. Explicit expressions for _ can be obtained by means of an eigenvalue decomposition of the kernel matrix.

The approximation of the feature space is based on a fixed subset of data points. One way to select this fixed-size set is to optimise the entropy criterion of the subset. As an example, we run the *fixedsizescript*1 script and obtain the Figure 13. The script chooses an optimal subset of points to be used for fixed size ls-svm according to the quadratic Renyi entropy for a kernel-based estimator. The script is run for a specific value of the hyperparameter of the RBF kernel (_2 = 1). Figure 14 compares the same results for varying _2 (0.01, 0.1, 0.5, 10).

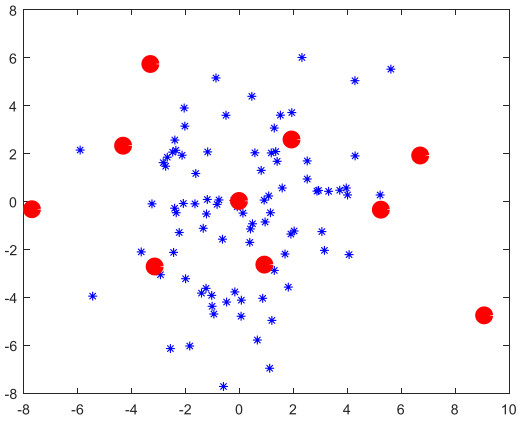


Fig-7.1. Plots for Sigma2 0.05

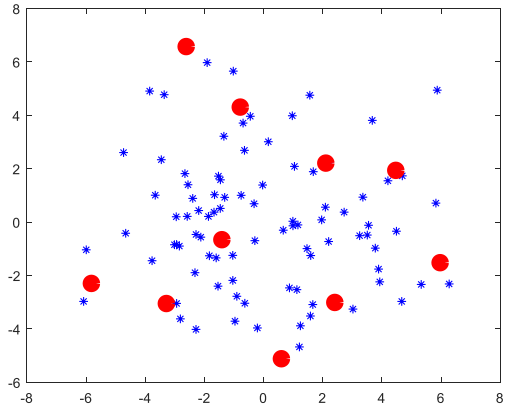


Fig-7.2. Plots for Sigma2 0.1

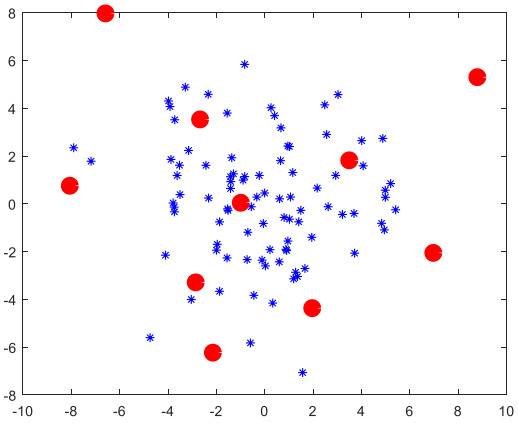


Fig-7.3. Plots for Sigma2 1.0

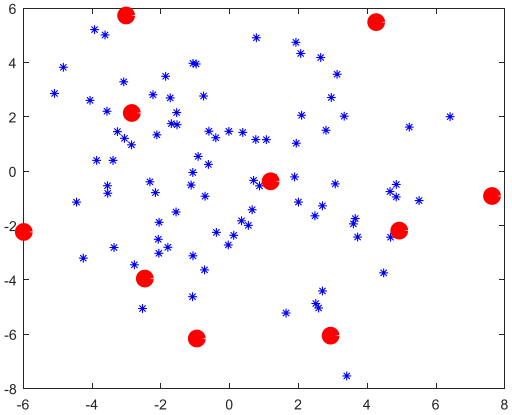


Fig-7.4. Plots for Sigma2 5.0

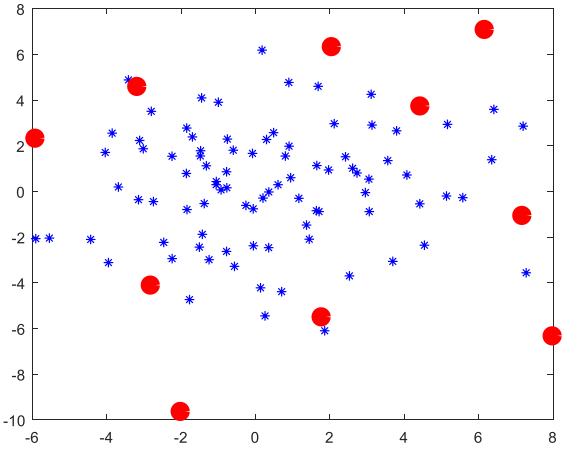


Fig-7.5. Plots for Sigma2 10.0

We clearly see that the hyper-parameter σ2 has an influence on chosen (optimal points) as follows;

- As σ 2 increases, the optimal support vectors are more spread across the entire cloud of points and less concentrated in the centre.
- For higher values of σ 2, the chosen points are more situated at the boundaries of the cloud defined by all points.

Kernel density estimation is a non-parametric way to estimate the probability density function of a random variable (here, RBF kernel is used). For fixed size ls-svm, the selection of a subsample of size *M* (*M* has here chosen arbitrarily), the initial support vectors, is done prior to the estimation of the model, and the final performance of the model can depend on the quality of the initial selection. It is possible to take a random selection of *M* data points and use them to build the approximation of the nonlinear mapping Φ, or it is possible to use a more optimal selection. External criteria such as entropy maximisation can be applied for an optimal selection of the subsample. In this case, given a fixed-size *M*, the aim is to select the support vectors that maximise the quadratic Renyi entropy.

$$H_R = -log \int p(x)^2 dx$$

that can be approximated by,

$$\int \tilde{p}(x)^2 dx = \frac{1}{N^2} 1^T \Omega 1$$

Where Ω is the kernel matrix.

Starting from a random sample of size *M*, it is possible to replace elements of the selected sample by elements of the remaining sample if the entropy is minimised, and iterate this procedure until convergence. In this way, it is possible to obtain a selection of those *M* points that converge to a minimum value of the quadratic entropy.
Therefore, the algorithm used in the *fixedsizescript*1 script converges to the points that 'summarises' the best the kernel density estimation (i.e. have the highest probability in the KDE). It is important to choose an appropriate value of the hyperparameter. This can be done using cross-validation based on quadratic Renyi entropy.
Given this optimised subset, the feature space mapping can be reconstructed. Figure 8 shows an example of such reconstruction for a random variable of dimension for which three points have been extracted.

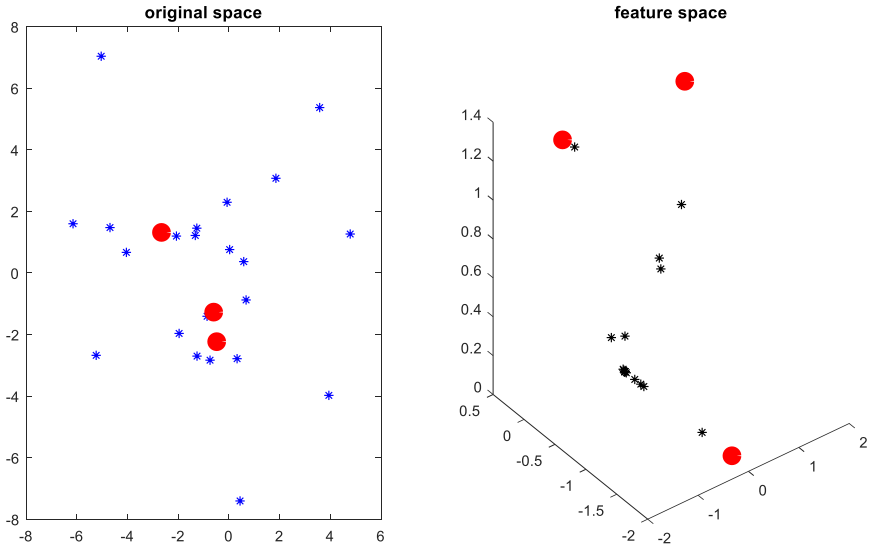


Fig-8. Original and Feature space

A drawback of LS-SVM models is the lack of sparseness, as nearly all patterns become support vectors. A solution to this problem is To use *L*0-norm. The *L*0-norm counts the number of non-zero elements of a vector. Therefore, minimising it leads to very sparse models. Figure 9 compares Fixed-Size LS-SVM with the *L*0 approximation LS-SVM.

Comparison of FS LS-SVM and *L*0 approximation LS-SVM:

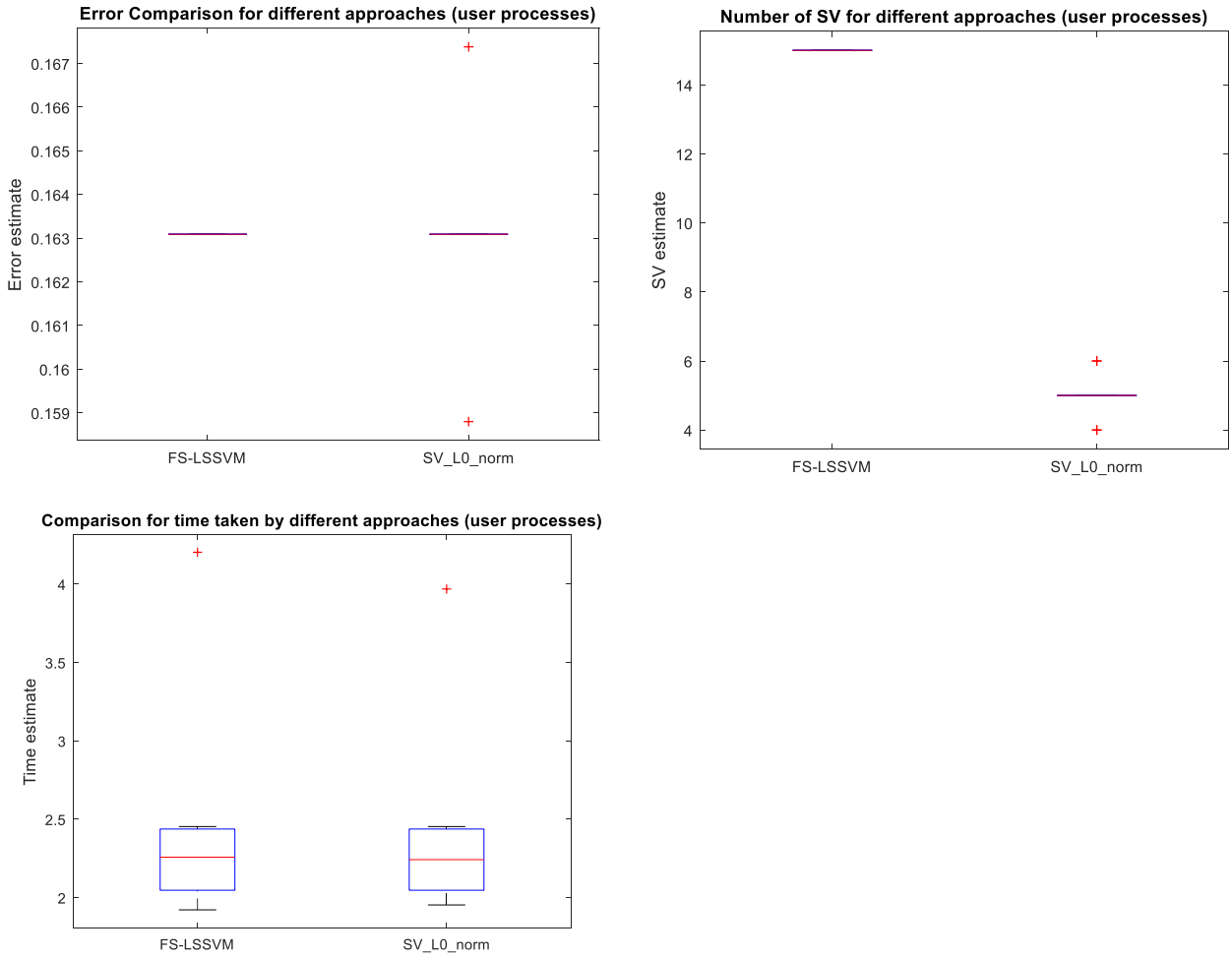


Fig-9. FS-LSSVM vs SV_L0_nom Comparison

The performance (in terms of error) of *L*0 approximation is comparable to the Fixed-Size LS-SVM. This is very good, as the number of support vectors is generally much lower than the Fixed-Size case. The computing time is comparable to both models but obviously much lower than the LS-SVM case.

3.5 Homework Problems:

3.5.1 Handwritten Digit Denoising

This work demonstrates how we can use the kernel methods for denoising the handwritten digits with varying details of different parameters. Usually, a rule of thumb for σ2 has calculated as the mean of the variances of each dimension times the dimension (number of features) of the training data. The σ2 hyper-parameter plays a major role in the performance of the denoising and should be tuned for meeting the desired performances. When we change the value of σ2 to a much bigger value compared to the previous rule of thumb with an increase in the noise factor, we obtain various results which all are presented below. We also observe that if the value if the value of σ2 is too high, denoising with kernel PCA deteriorates. As noise is associated with darker colours, higher values of σ2 imply that each point will be influenced by more adjacent points in the denoising process. Therefore, higher values of σ2 lead to a less efficient denoising algorithm which can be seen below.

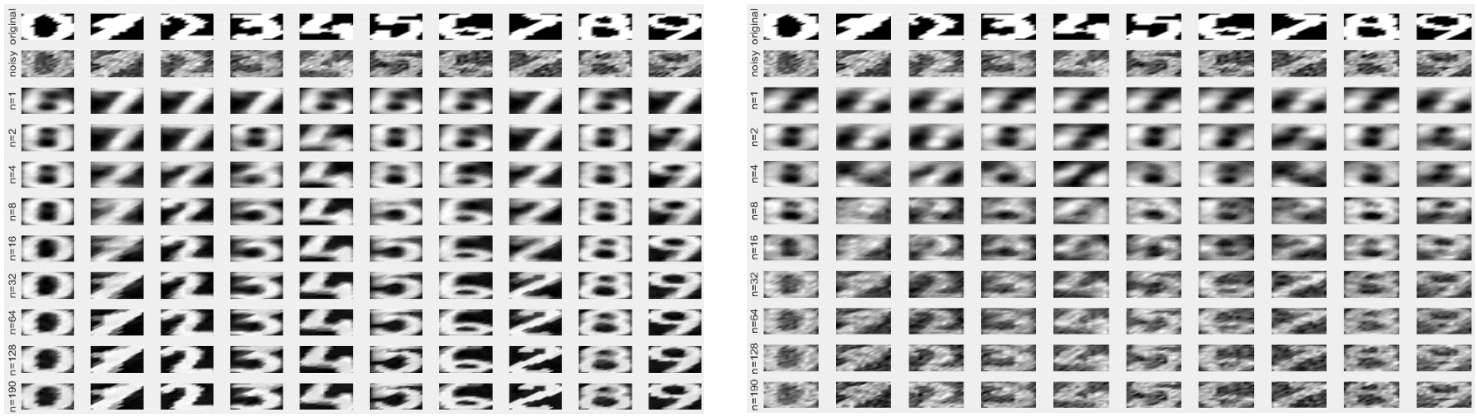


Fig-10.1 Denoising based on Kernel PCA and Linear PCA with sigma2=0.7 and noise factor 0.3

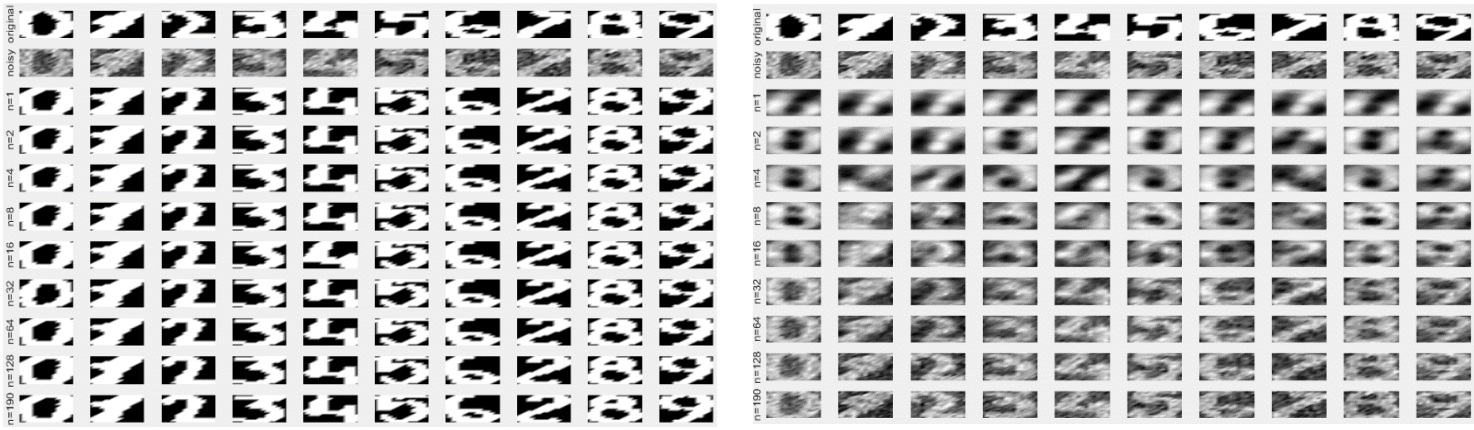


Fig-10.2 Denoising based on Kernel PCA and Linear PCA with sigma2=0.01 and noise factor 0.3

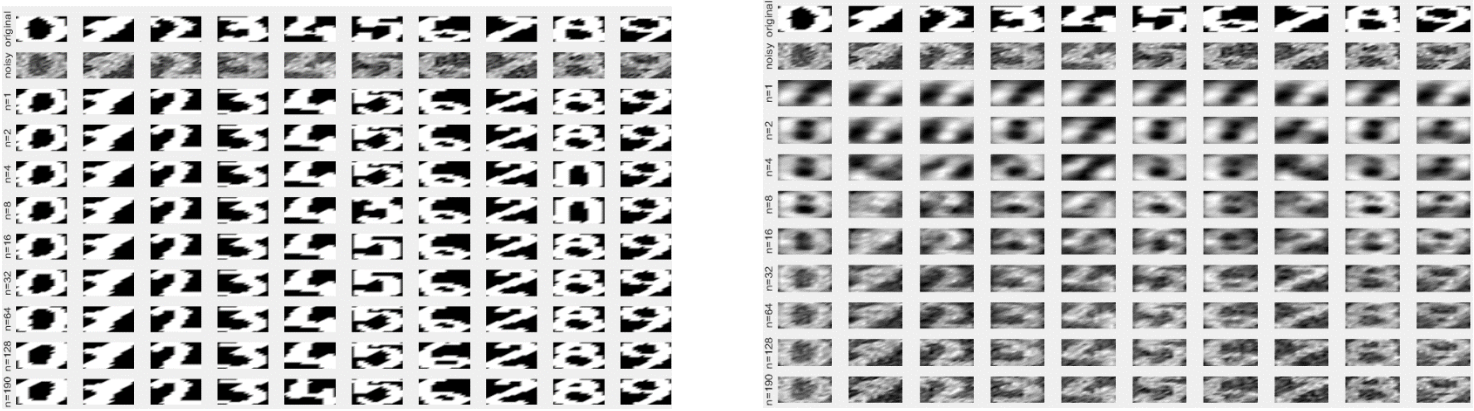


Fig-10.3 Denoising based on Kernel PCA and Linear PCA with sigma2=1 and noise factor 0.3

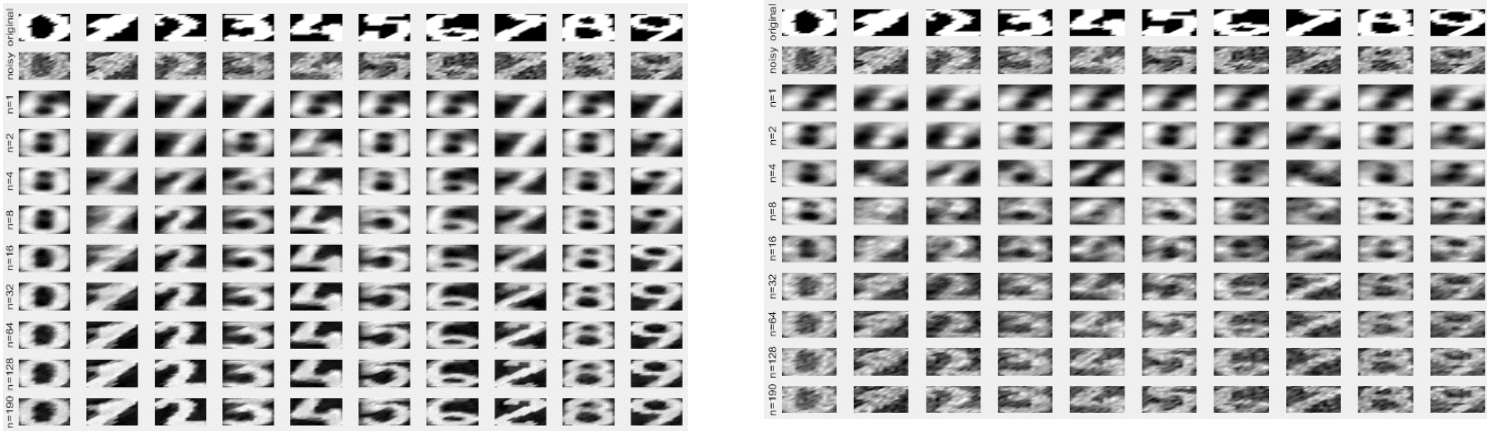


Fig-10.3 Denoising based on Kernel PCA and Linear PCA with sigma2=1 and noise factor 0.3

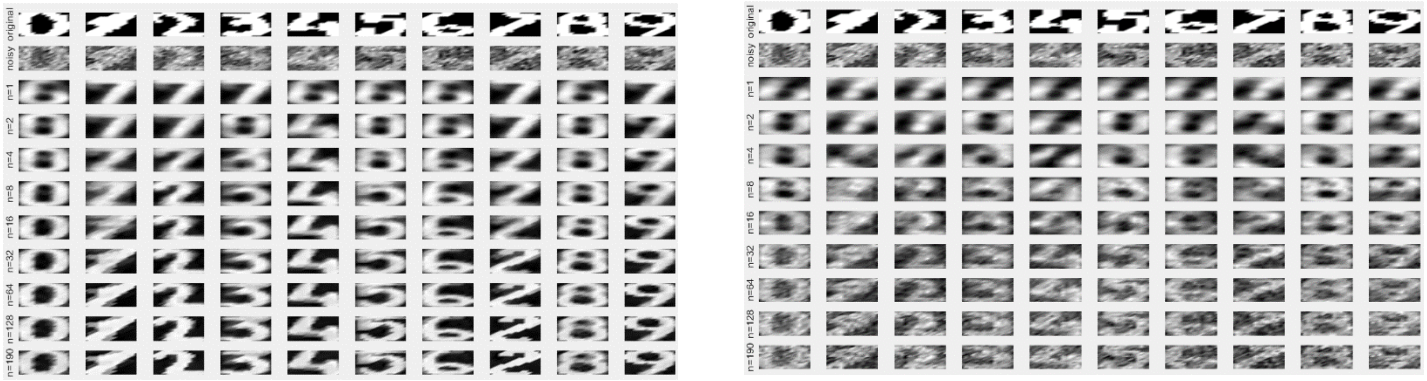


Fig-10.3 Denoising based on Kernel PCA and Linear PCA with sigma2=0.7 and noise factor 0.4

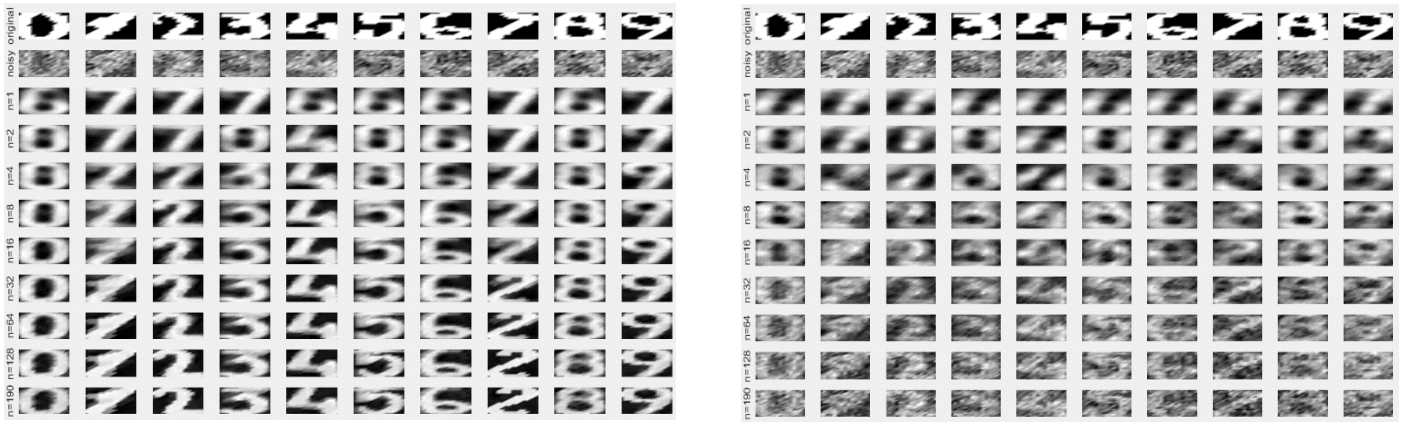


Fig-10.3 Denoising based on Kernel PCA and Linear PCA with sigma2=0.7 and noise factor 0.5

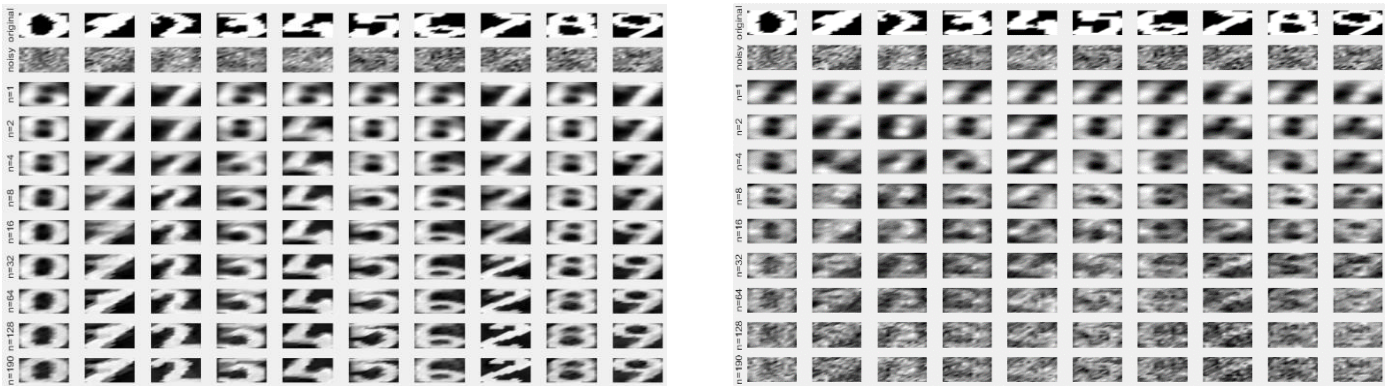


Fig-10.3 Denoising based on Kernel PCA and Linear PCA with sigma2=0.7 and noise factor 0.7

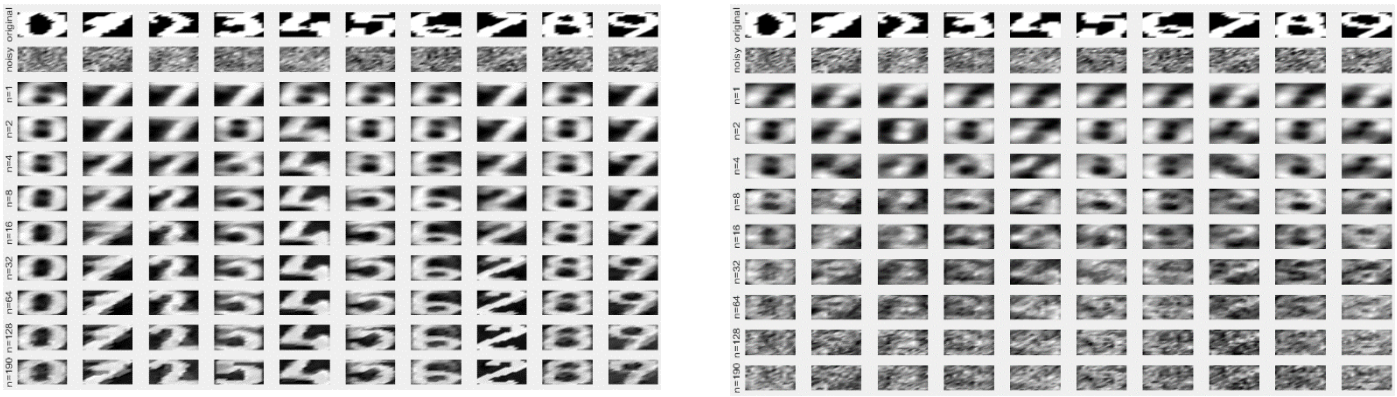


Fig-10.3 Denoising based on Kernel PCA and Linear PCA with sigma2=0.7 and noise factor 0.9

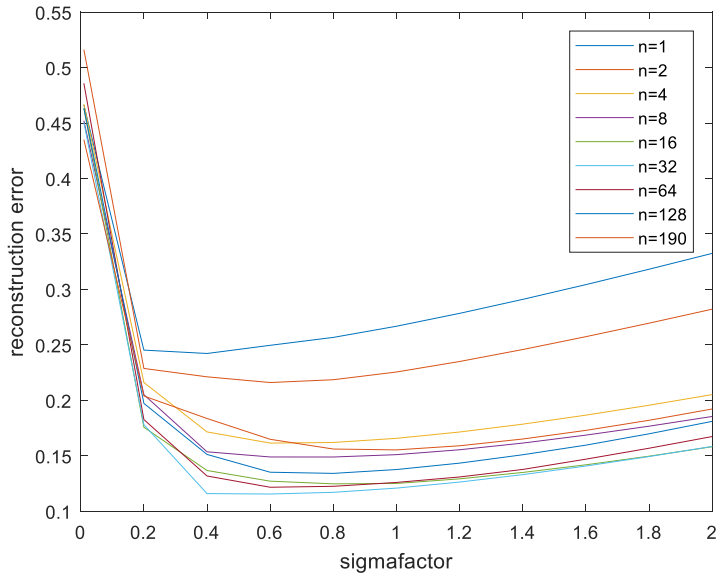


Fig-11 Reconstruction error as a function of sigma factor

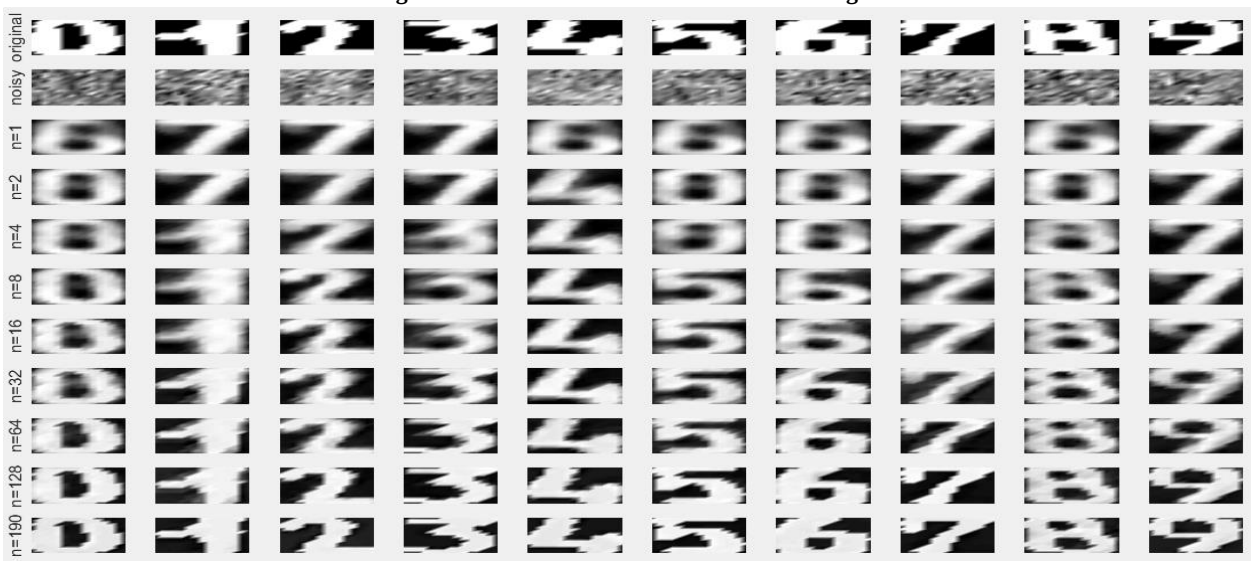


Fig-12 Plot for denoising on the test set using Kernel PCA

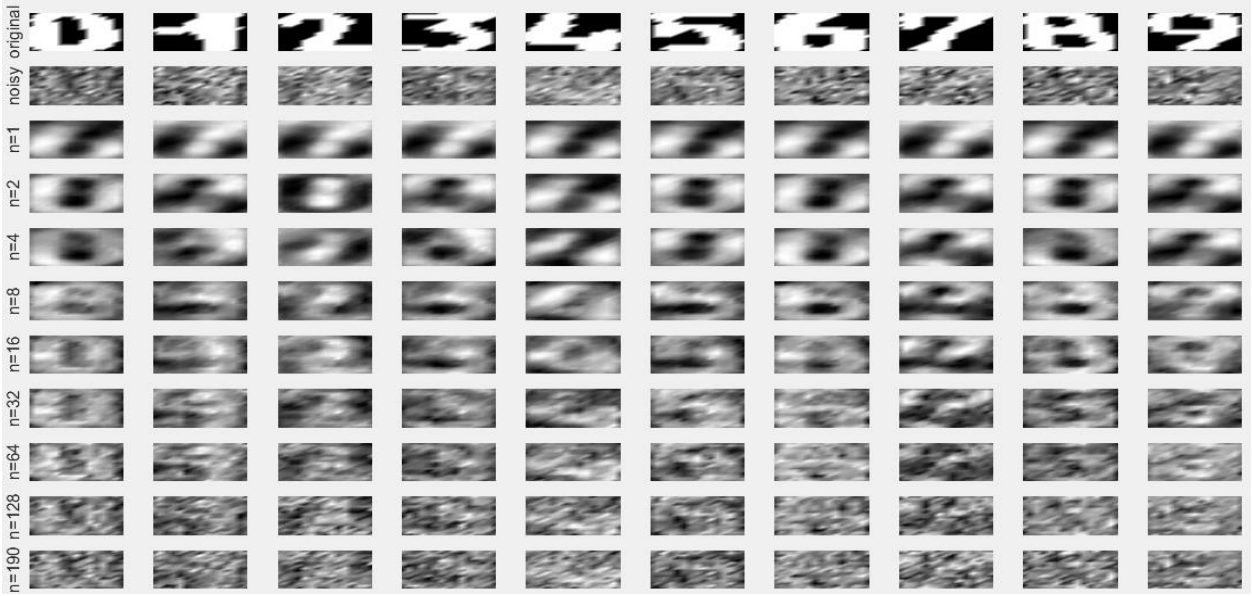


Fig-13 Plot for denoising on the test for using Linear PCA

We chose σ *factor* that minimised the reconstruction error on the validation set (Xtest1). The exact value minimising reconstruction error depends on the number of principal components we keep for the denoising. However, we can observe from these Figures that the reconstruction error is always minimised around a σ *factor* value of 0.5. Therefore, we use this same σ *factor* on the test set(Xtest2). We observe that again, kernel PCA (left) gives much better results than linear PCA.

3.5.2 Shuttle Dataset:

In this section, the methods of Fixed-size LS-SVM and 'l_0' approximation are compared for the classification on the Shuttle data set. For both fixed size LS-SVM and 'l0' approximation method, Nystrom approximated support vectors are used as training data. For the latter case, a 'l0' norm is first performed on these support vectors. Thus the latter case can be seen as a pruned version of fixed size LS-SVM. The experiment is performed with a random sample of 1000 points with replacement. Simplex optimisation and CSA are used for optimal hyperparameters tuning. Both RBF and linear kernel are tried for comparison and plotted on the sample data.

Fig.14 shows the results for RBF kernel and the following shall be observed;

- It can be observed that the mean of errors for both cases is almost the same.
- Support vectors for the former are very large compared to the latter.
- Also, the computation times for both the methods are quite large and comparable. Thus, the 'l0' approximated version performs slightly better than the unpruned version.

For the linear case (Fig.15), the following are the observations;

- The mean error is same.
- This could also be due to the smaller sampling size of the dataset considered.
- The number of support vectors decreases for the linear model. The time taken by the linear method is also significantly smaller due to reduced computational efforts.
- In both case, 'l0' approximation produces a sparser solution and can be preferred over the standard FS LSSVM for this dataset.

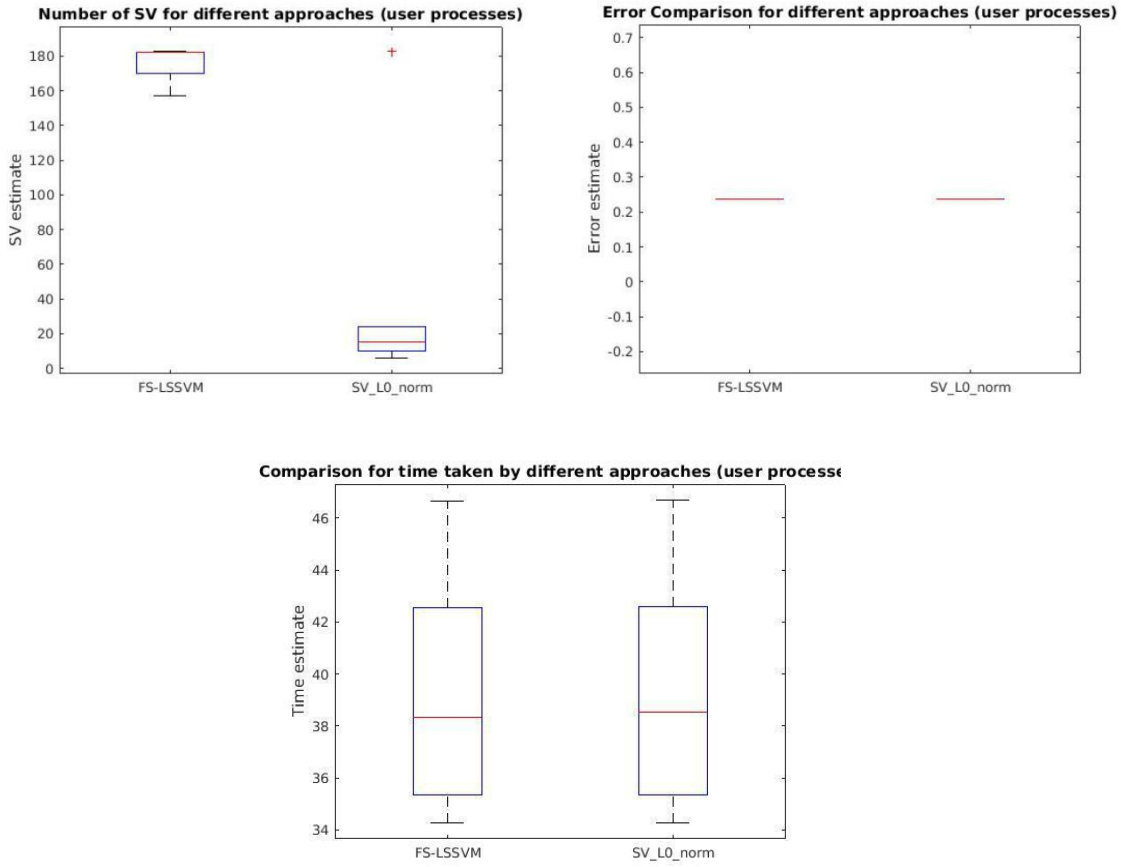


Fig.14 RBF Kernel

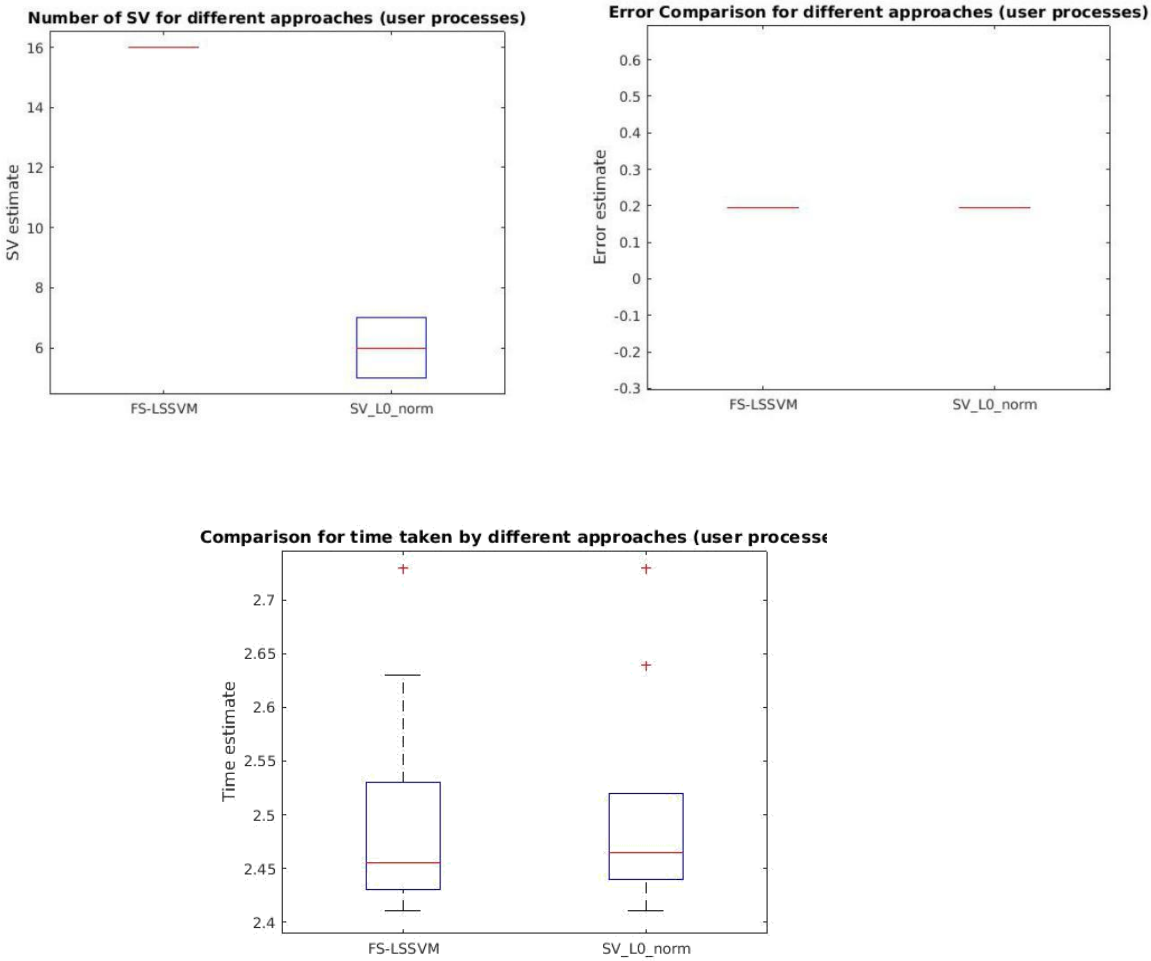


Fig-15 Linear kernel

3.5.3 California Dataset:

The fslssvm script is adjusted to be used with California dataset with given number of following features and apply regression. With different types of kernels, the comparison has been made between FS-LSSVM and SV_L0_norm.

Polynomial plots:

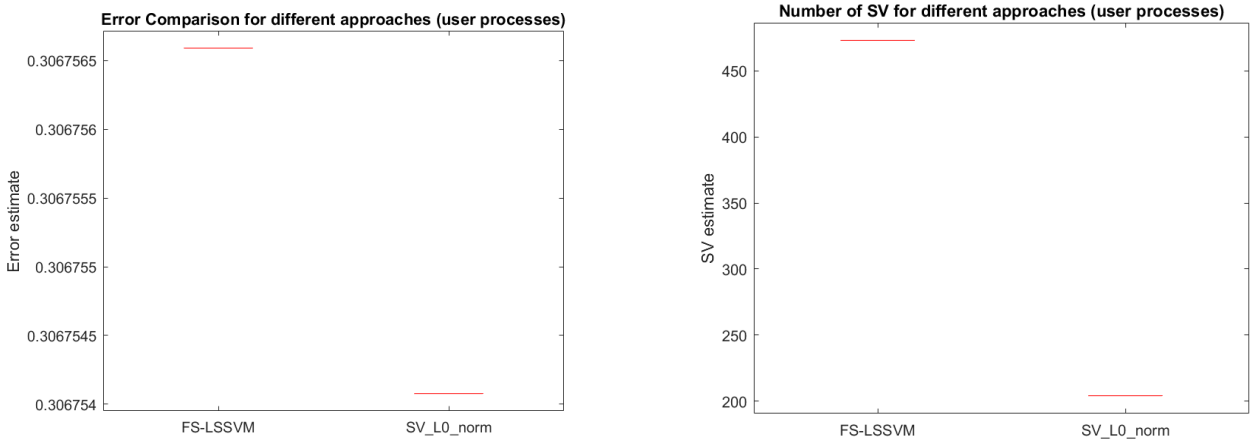


Fig-16.a Plots for polynomial kernel

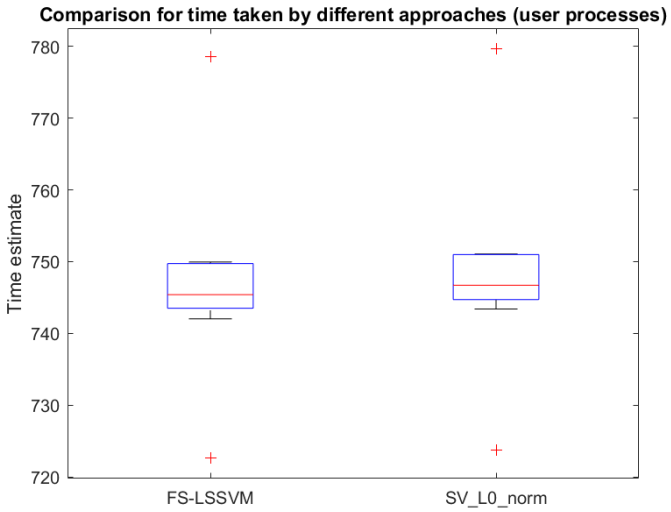
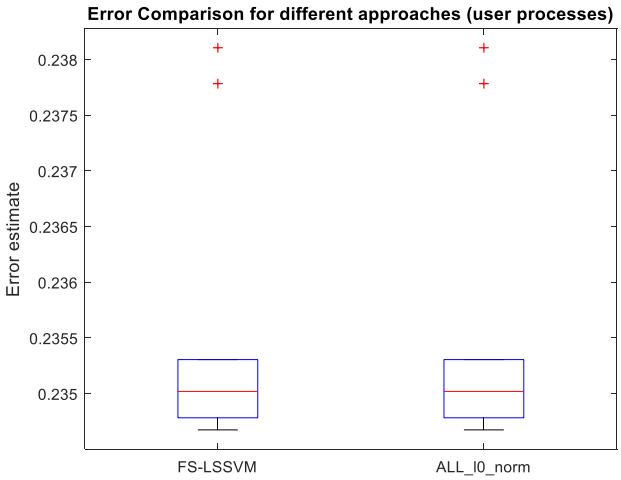
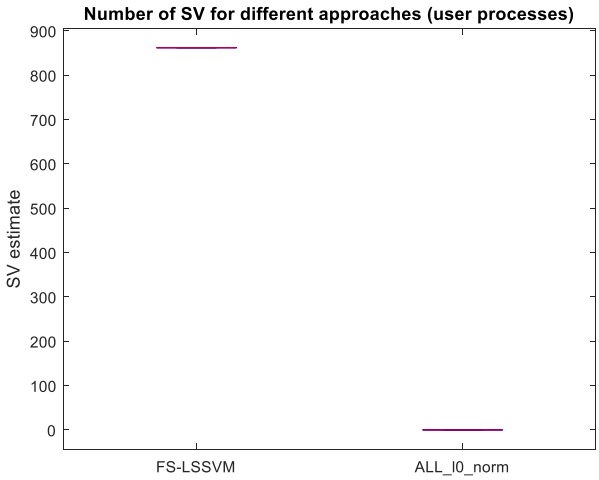


Fig-16.b Plots for polynomial kernel

Using RBF kernel:





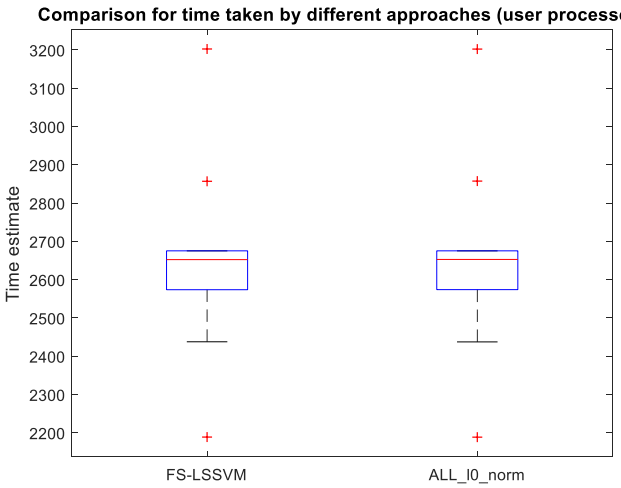


Fig-17 Plots for Comparison on California dataset using RBF Kernel