Katholieke
Universiteit
Leuven

**Department of
Computer Science**

SUPPORT VECTOR MACHINES

Assignment Report

Aditya Paliwal (r0732161)

Academic year 2021–2022

Contents

Aditya Paliwal
r0732161

1 Exercise Session 1: Classification

1.1 A simple Example: Two Gaussians

In the given toy example, a variable X is formed out of two different sub variables X1 and X2. Here X1 and X2 are the randomly generated clusters with expected centers specified by [1, 1] for X1 and [-1, -1] for X2. The plot can be seen in figure 1.

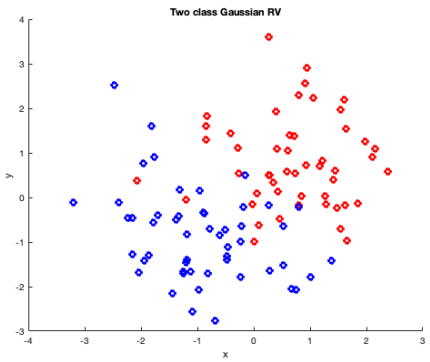


Figure 1: Classification for two class variable

In the above figure, it can be clearly seen that the two classes can not be separated using a linear hyperplane due to partial overlap of two distributions. In order to obtain a linear hyperplane, one approach can be to misclassify the overlapping points in the training set. A non-linear boundary can be obtained as well with lesser number of misclassified observations.

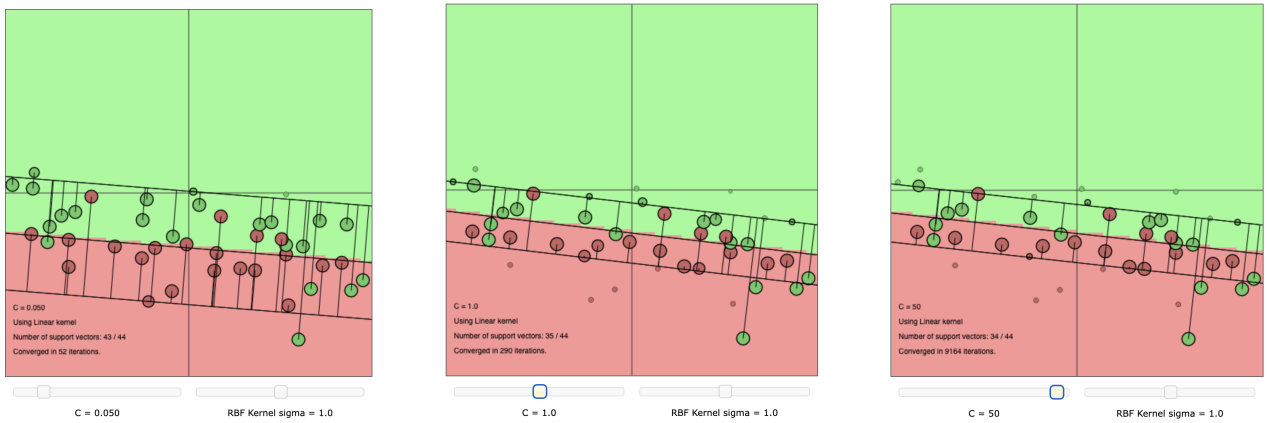
1.2 The Support Vector Machine

The web application at https://cs.stanford.edu/people/karpathy/svmjs/demo/ was used to visualize the working of Support Vector Machines. Different experiments were performed with SVM classification after adding the new data points in the dataset . Based on the experiments, following questions are answered:

1. *What is a support vector? When does a particular data point become a support vector? When does the importance of the support vector change?*
   Answer: Any point which directly influences the location of the decision boundary is a *support vector*. Any data point becomes a support vector if it is close to the decision boundary. The importance of the support vector changes each time either a new point is added to the data set or hyper parameters are changed.

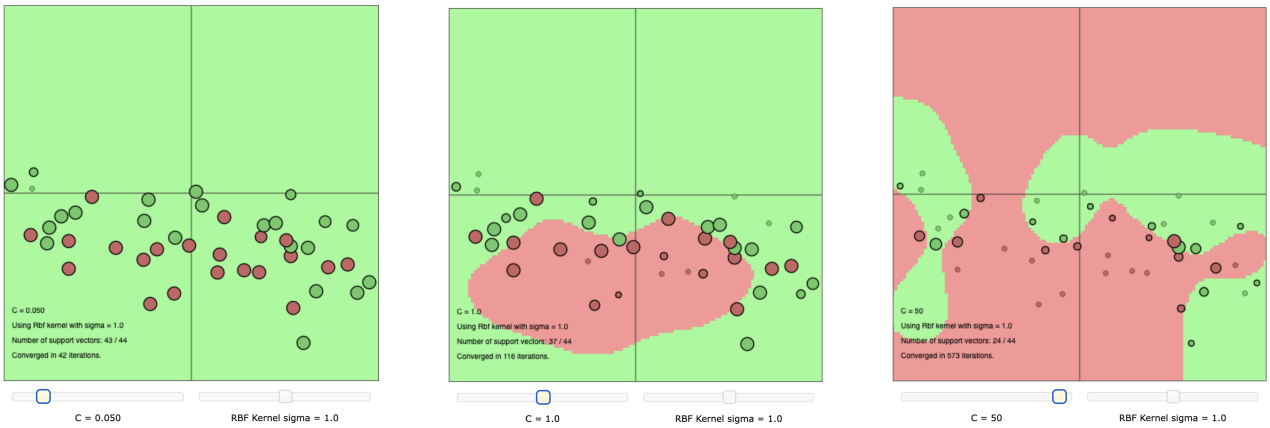
2. *What is the role of parameters c and sigma? What happens to the classification boundary if you change these parameters? Illustrate visually.*
   Answer: The **regularization parameter** c is responsible for controlling the number of points used as support vectors. The experiments were performed with different values of c for both linear and rbf kernels. The results showed that in case of a larger value of c, less number of support vectors were used while a with a smaller value of c larger portion of total points was used as support vectors. The margin size is affected as well with change in value of c where smaller c resulted in larger margins. Additionally, in case of linear kernel, larger value of c required more iterations to converge whereas smaller value for regularization parameter converges faster.

In case of rbf kernel, when the lower value is used for regularization parameter c only the dominant class label is shown for the whole region. With the increased value of c, different regions can be seen for different classes. In comparison to linear kernel, more data points are used as support vectors in case of rbf kernel. Another important observation can be made for both the kernels. The size of the points change in case of different values of c. This indicates the importance of that point in making of the decision boundary where the points closer to the boundary are bigger than the points away from the decision boundary. The visual illustration of these observations can be seen in the plots shown in figures 2 and 3.



(a) $c = 0.050$ (b) $c = 1$ (c) $c = 50$

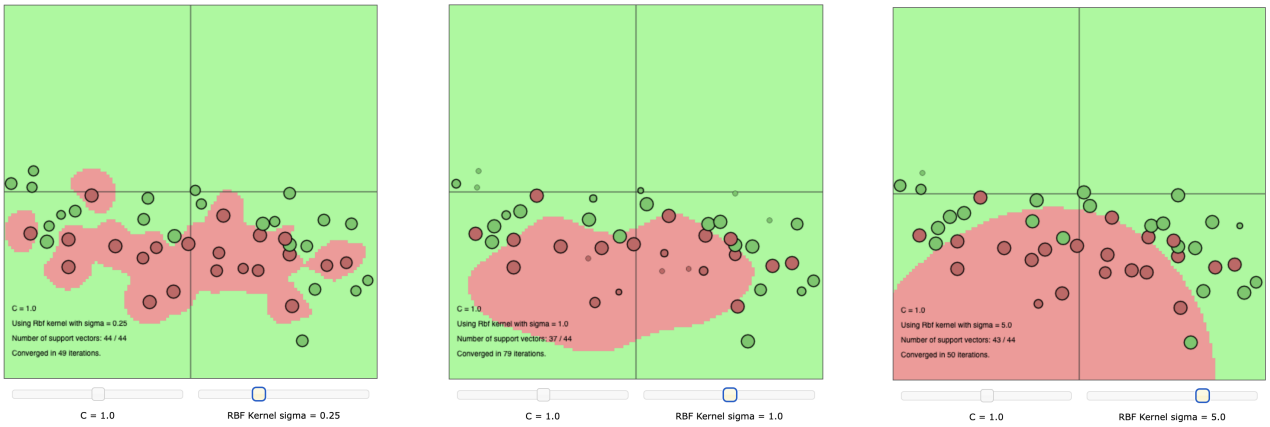
Figure 2: Linear kernel with variable c and fixed *sigma*



(a) $c = 0.050$ (b) $c = 1$ (c) $c = 50$

Figure 3: RBF kernel with variable c and fixed *sigma*

The parameter **sigma** is responsible for controlling the smoothness of the decision boundary. A near linear decision boundary can be observed while using the higher value of sigma where as the decision boundary tends to be more rough in case of a smaller sigma value. This can be see in figure 4 below.



(a) $c = 1$, $sigma = 0.25$ (b) $c = 1$, $sigma = 1$ (c) $c = 1$, $sigma = 5$

Figure 4: RBF kernel with variable *sigma* value and a fixed c

3. *What happens to the classification boundary when sigma is taken very large? Why?*
   Answer: As already mentioned in the previous question, sigma controls the smoothness of the decision boundary. If a very large value is taken for sigma, the decision boundary is near linear because a larger sigma tends to make a much more general classifier where on the other hand a smaller value of sigma tends to make a local classifier.

1.3 Least-squares support vector machine classifier

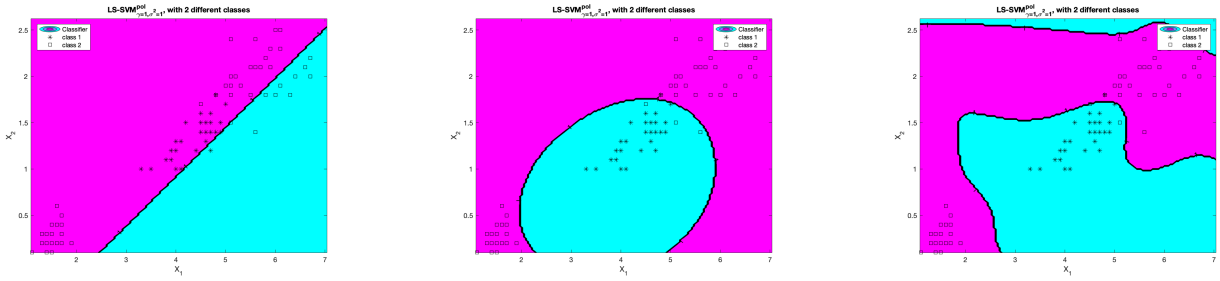
In this exercise multiple experiments were performed, on the available **iris dataset**, under the influence of different kernel parameters and hyper parameters. Results obtained will be discussed in the following sub-sections.

1.3.1 Influence of hyperparameters and kernel parameters

In the first set of experiments, polynomial and RBF kernel were analyzed. The polynomial kernel was used with different degree values ranging from 1 to 20 with a fixed gamma value of 1. In case of experiments using RBF kernel, different values of sigma were explored. Based of the results obtained, following questions were answered.

1. *Trying out polynomial kernel with different degree values. What happens when you change the degree of the polynomial kernel?*
   Answer: For any degree d, the polynomial kernel K can be formulated as $K(x, y) = (x^T + \gamma)^d$. Using different values of degree d, ranging from 1 to 20, and a fixed gamma value of 1 the test performance of SVM was analyzed. With an increase in value of d, the non-linearity of the decision boundary increased. This trend can be seen in figure 5 below. In case of very high values of d, a warning was issued for matrix becoming singular due to which results might be inaccurate. In such cases, all the data instances were classified as of same class.

   Talking about the classification error, the worst results were obtained with *degree = 1*. The classification error in that case was 55%. It was reduced to 5% when degree value was set to 2. The degree values = 3 to 10 gave lowest classification error with a perfect error of 0%. Further increase in value of d also led to bad classification results where



(a) $d = 1$ (b) $d = 3$ (c) $d = 10$

Figure 5: Iris Classification: Polynomial Kernel with different values for parameter d

error rate started increasing again. The plot for classification error vs degree can be seen in figure 6 below.

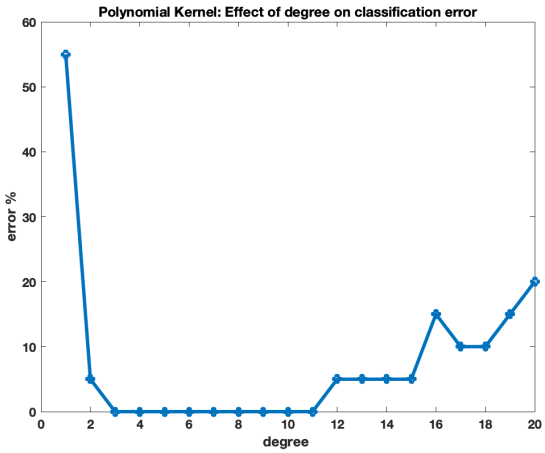
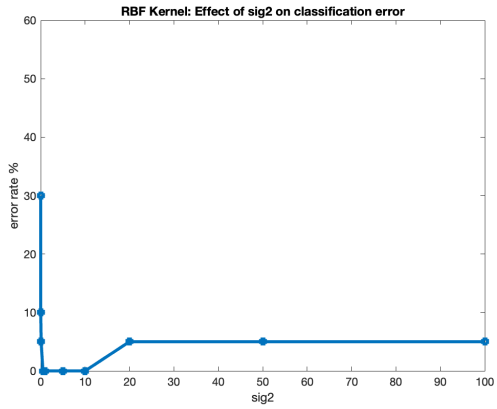


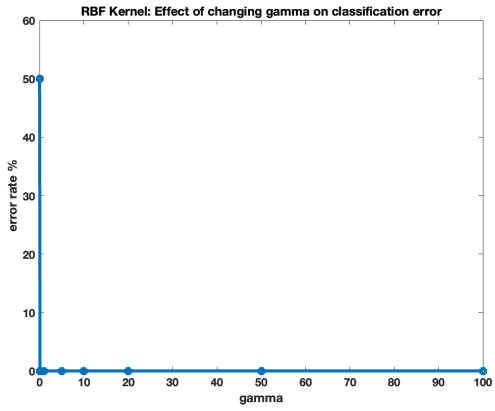
Figure 6: Plot showing the effect of increasing degree value on classification error

2. *RBF Kernel: exploring sig2 and gamma parameters*
Answer: For the first part of this experiment, 10 different values of sig2 were used. These selected values, ranging between 0.001 to 100, were (0.001, 0.01, 0.1, , 0.5, 1, 5, 10, 20, 50 and 100). For all these values, a fixed value of gamma was set to 1. The best results were obtained in case of sig2 = (0.1, 0.5, 1, 5, and 10). Rest of the values gave poor results where in case of very small sig2, i.e 0.001, the error rate was 30% and with values higher than 10 the error rate climbed to 50%.

The second part of this experiment was to observe the effect of changing gamma on classification performance with a fixed sig2 = 1. The different gamma values used for this purpose were (0.001, 0.01, 0.1, , 0.5, 1, 5, 10, 20, 50 and 100). The results were poor for very small values of gamma i.e. 0.001 and 0.01. On all the other values the model gave best results with a perfect error rate of 0%. The visualization of these trends can be seen in the figure 7.



(a) Variable sig2 with $\gamma = 1$



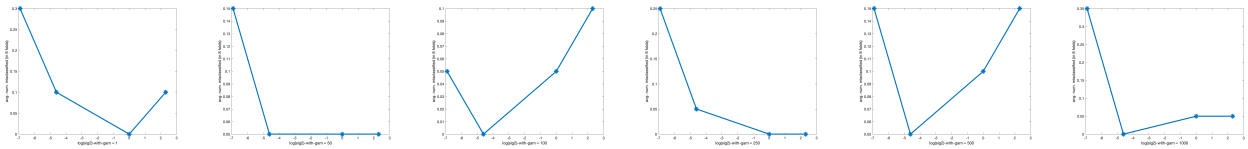
(b) Variable γ with $sig2 = 1$

Figure 7: Iris Classification: RBF Kernel with different values for sig2 and gamma. (a) Effect of change in sig2 on classification error. (b) Effect of changing γ on classification error.

1.3.2 Tuning parameters using validation

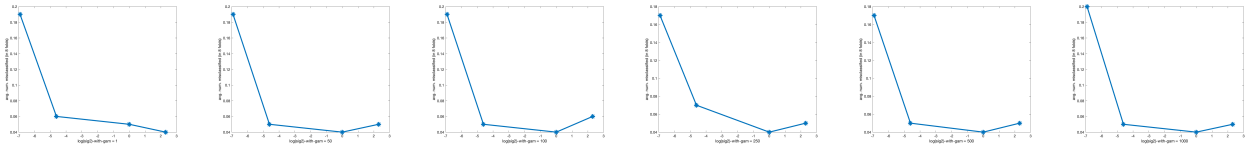
The aim of this exercise was to perform model hyperparameter tuning using multiple techniques such as random split, k-fold crossvalidation and leave-one-out validation. All the experiments were performed on the provided iris dataset and the results obtained after applying the mentioned techniques were used to answer the following questions.

1. *Compute the performance for a range of gam and sig2 values. Use the random split method, 10-fold cross-validation and leave-one-out validation. Visualize the results of each method: do you observe differences? Interpret the results: which values of gam and sig2 would you choose?*
   Answer: For each of the three techniques, wide range of combinations were used for gamma and sigma values. The plots below in figures 8 - 10 shows the effect of different sig2 values on classification performance of the models.



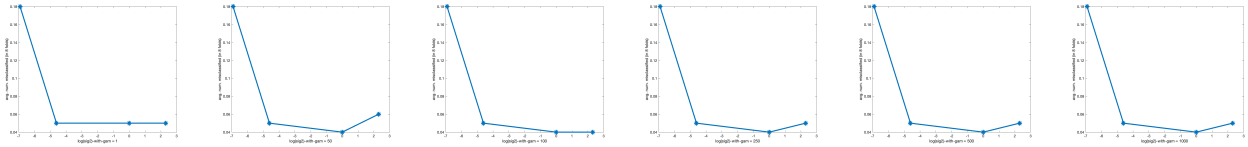
(a) $\gamma = 1$ (b) $\gamma = 50$ (c) $\gamma = 100$ (d) $\gamma = 250$ (e) $\gamma = 500$ (f) $\gamma = 1000$

Figure 8: Iris Classification. Hyperparameter tuning using **random-split validation**. The plots show the effect of sig2 parameter (0.001, 0.01, 1, and 10) on classification error (% of misclassified examples)



(a) $\gamma = 1$ (b) $\gamma = 50$ (c) $\gamma = 100$ (d) $\gamma = 250$ (e) $\gamma = 500$ (f) $\gamma = 1000$

Figure 9: Iris Classification. Hyperparameter tuning using **10-fold crossvalidation**. The plots show the effect of sig2 parameter (0.001, 0.01, 1, and 10) on classification error (% of misclassified examples in validation set)



(a) $\gamma = 1$ (b) $\gamma = 50$ (c) $\gamma = 100$ (d) $\gamma = 250$ (e) $\gamma = 500$ (f) $\gamma = 1000$

Figure 10: Iris Classification. Hyperparameter tuning using **leave-one-out crossvalidation**. The plots show the effect of sig2 parameter (0.001, 0.01, 1, and 10) on classification error (% of misclassified examples)

From the above plots, it is evident that k-fold crossvalidation and LOOCV had almost similar values for % of misclassified examples. From all the possible combinations, gamma = 100 and sig2 = 0.01 gave the minimum error and hence it can be chosen as the most optimal pair of hyperparameters among all possible combinations.

Out of these three, one would definitely prefer k-fold crossvalidation or LOOCV depending upon the size of the dataset. In case of large datasets, k-fold cv should be preferred because, k-fold crossvalidation in such cases is computationally inexpensive. On the other side, if the size of dataset is small, LOOCV should be used as each fold in k-fold cv might have too less instances.

1.3.3 Automatic parameter tuning

Automatic tuning procedure **tunelssvm**, from the LS-SVM toolbox was used for this purpose. Choice of different parameters used in this procedure, such as *'algorithm'*, *'performance_metrics'*, and many more provided the opportunity to try out different combinations and observe the effect on results obtained. Based on the observations, the following question was answered.

1. *Try out the different 'algorithm'. What differences do you observe? Why do the obtained hyperparameters differ a lot in different runs? What about the cost? Computational speed? Explain the results.*
   Answer: The automatic tuning was performed using two different algorithms, namely 'simplex' (Nelder-Mead method) and 'gridsearch' (brute force gridsearch). Model initialization parameters 'csa' (coupled simulated annealing) and 'ds' (randomized directional search) were explored as well. The optimal hyperparameters obtained, along with the computational cost of each combination can be seen in table 1 below.

   The values in the above table shows variance in the optimal parameters obtained. Additionally, the computation times didn't vary much for different combinations but among

Table 1: Table showing optimal gamma and sig2 values for different combinations of algorithms and model parameters 'csa' and 'ds'

Algorithm	**gamma**	**sig2**	**Cost**
simplex + csa	1.2953	0.024	0.0400
simplex + ds	2.0388	6.958	0.0300
gridsearch + csa	5.8274	6.847	0.0200
gridsearch + ds	0.4877	5.168	0.0400

all, (**gridsearch + csa**) took the least time.

1.3.4 Using ROC curves

ROC curve is another performance evaluation technique used to judge any classifier. The criteria here is to calculate the area under the curve and higher the area, better the classifier. Using the ROC curve for this part of the exercise, helped in answering the questions below.

1. *In practice, we compute the ROC curve on the test set, rather than on the training set. Why?*
   Answer: The ROC curve plotted between true positive and false positive rate is generally used for test set rather than training set in order to avoid the biased estimate. The classifier is supposed to see all the data points in the training set and therefore analysing the results on same data points might not be beneficial. The unseen test data set therefore used for accessing performance using ROC curve.

2. *Generate the ROC curve for the iris.mat dataset (use tuned gam and sig2 values). Interpret the result.*
   Answer: The ROC curve for the **iris** dataset is plotted below in figure 11. In the cure, it can be seen that the area under the curve (AUC) is 1 which depicts that the classifier is able to differentiate between the two classes perfectly.

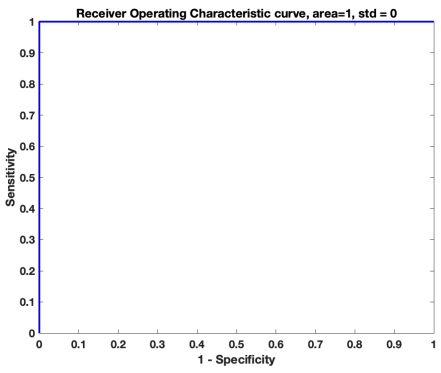


Figure 11: ROC curve for iris dataset

1.3.5 Bayesian framework

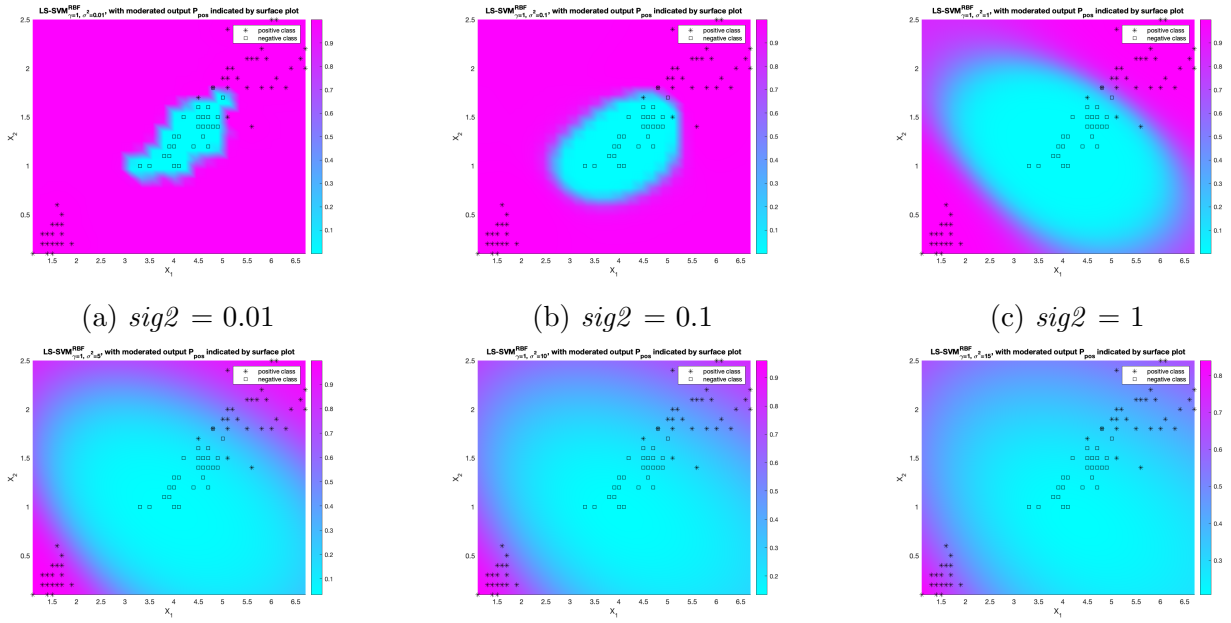
With the help of Bayesian framework, it is possible to get the probability estimates of the data points belonging to a particular class. This framework will be used for next set of experiments

to study the effect of sig2 and gamma on classification results. The experiments, using the same iris dataset, and the results will be discussed below.

1. *How do you interpret the colors of the plot?*
   Answer: The Bayesian framework plots use different colors to differentiate between classes. Here, **blue** and **magenta** colors are used to represent **negative** and **positive** classes respectively. The intensity of the color depicts the probabilities. More intense the color, higher is the probability to belong to a particular class.

2. *Change the values of gam and sig2. Visualize and discuss the influence of the parameters on the figure.*
   Answer: The effect of changing sig2 and gamma was studied in this experiment. The results were as follows:

   Effect of changing *sig2:*
   In order to study the effect of changing sig2 on classification performance, various sig2 values (0.01, 0.1 , 1, 5, 10, and 15) were used. For all these values of sig2, gamma was set to 1. The plots showing the variations can be seen in figure 12. These plots shows that there exists a non linear decision boundary between the two classes. As the value of sig2 increases, the decision boundary begins to expand outwards and becomes more linear. The number of misclassified examples also increase with increase in sig2.

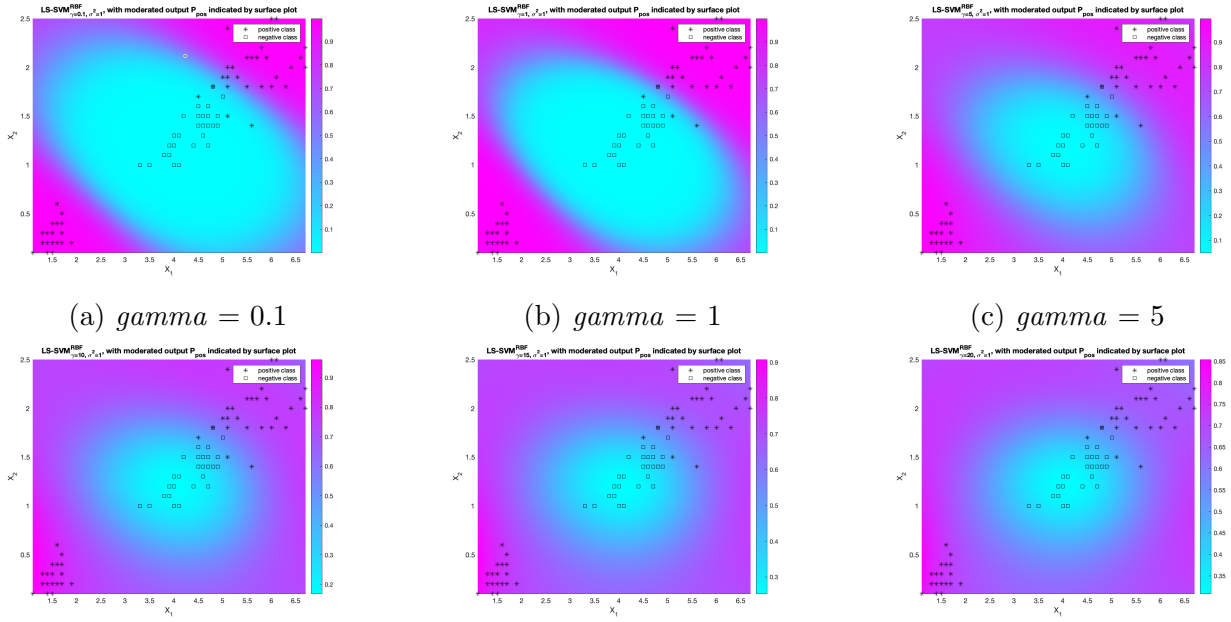


(a) *sig2* = 0.01 (b) *sig2* = 0.1 (c) *sig2* = 1

(d) *sig2* = 5 (e) *sig2* = 10 (f) *sig2* = 15

Figure 12: Bayesian Framework - Effect of changing *sig2*

   Effect of changing *gamma*:
   Similar to previous experiment, once again the aim was to study the change in classification performance but with different gamma values (0.1, 1, 5, 10, 15, and 20). The sig2 value for this purpose was fixed to 1. This effect can be visualized using figure 13 below. With increase in value of sigma, in contrast to sig2 effect, the decision boundary here contracts towards middle making the decision boundary even more non-linear. This validates the theory that higher gamma leads to lower tolerance towards misclassifications.



(a) $gamma = 0.1$ (b) $gamma = 1$ (c) $gamma = 5$

(d) $gamma = 10$ (e) $gamma = 15$ (f) $gamma = 20$

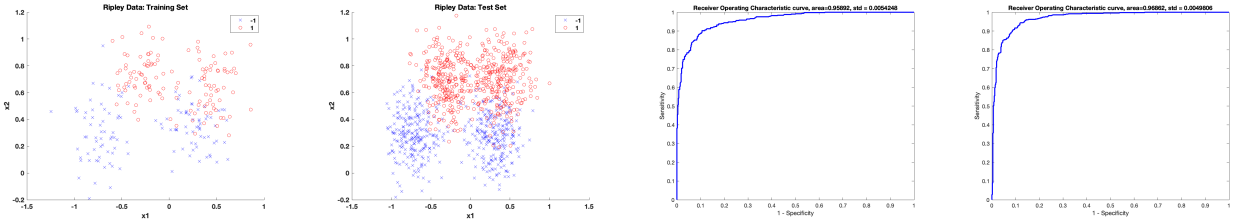
Figure 13: Bayesian Framework - Effect of changing *gamma*

1.4 Homework Problems

The knowledge gained in the above sections will be applied on different datasets and the results will be shared in this section of the report. In order to analyse the datasets models were trained using two different kernels, linear kernel SVM and rbf kernel SVM. The optimal values for sig2 and gamma were obtained using automatic parameter tuning by tunelssvm procedure. 10-fold crossvalidation using misclassification and simplex as algorithm were used as parameters in the tuning procedure. In the end, the performance of the model was evaluated using ROC curve. The results will be discussed in the respective subsections.

1.4.1 Ripley Dataset

Total number of training and test instances were 250 and 1000 respectively. Figure 14 (a) and 14 (b) shows the plots for training set and test sets. The visualisation shows that two classes can be easily separated using linear kernel with few misclassifications. The initial hypothesis was that both linear and RBF kernel should perform well on this data. The results proved that hypothesis was correct as the classification accuracy was 95.8% for linear kernel and 96.8% for RBF kernel. The ROC curves for the same can be seen in figure 14 (c) and 14 (d). The overall performance remains constant even after multiple combination of optimal hyperparameters obtained after every new iteration of tunelssvm procedure. Similar to the previous experiments throughout this exercise, the optimal hyperparameters vary by large amounts.

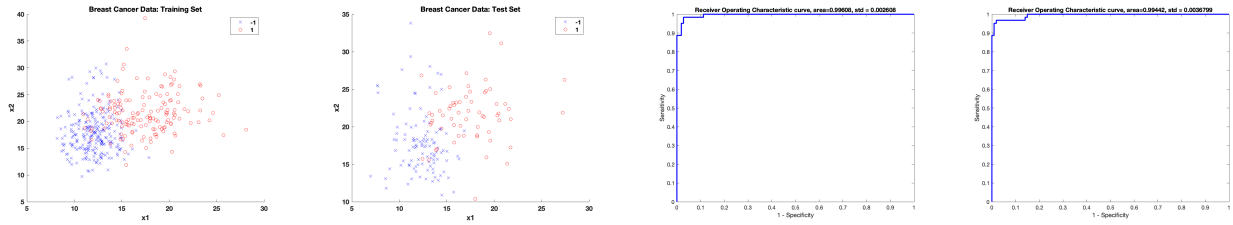


(a) Ripley Training Set (b) Ripley Test Set (c) ROC "lin_kernel" (d) ROC "RBF_kernel"

Figure 14: Visualizations for Ripley dataset

1.4.2 Wisconsin Breast Cancer dataset

The training set with 400 instances, as well as test set with 169 instances can be visualized in figures 15 (a) and 15 (b) respectively. Out of linear or RBF kernel, any of the model can be chosen as the classification accuracy in both the cases was more than 99%. the ROC curves for the same can be seen in figures 15 (c) and 15 (d).

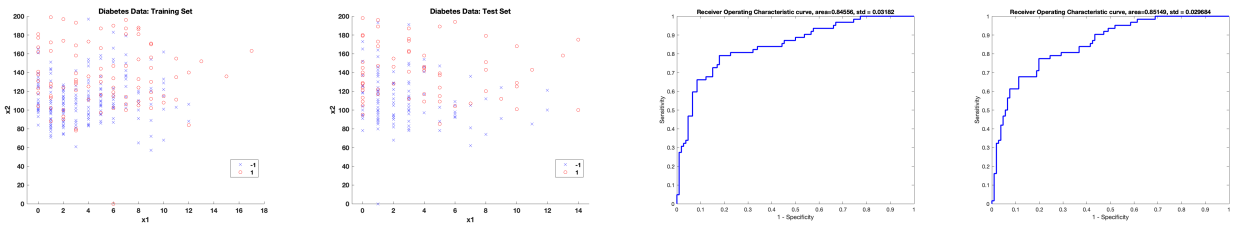


(a) Cancer Training Set (b) Cancer Set (c) ROC "lin_kernel" (d) ROC "RBF_kernel"

Figure 15: Visualizations for Breast Cancer dataset

1.4.3 Diabetes dataset

Diabetes data had 300 training instances and 168 test instances for performance evaluation. This dataset consists of 8 features and therefore difficult to visualize. Only first 2 features of test and training sets were plotted for visualization purpose. These can be seen in figures 16 (a) and 16 (b). This was the most complex data out of all the three with lots of overlaps between instances of different classes. ROC curves for both linear and RBF kernels can be seen in figure 16 (c) and 16 (d). From these curves, it is visible that RBF performed marginally better with 85% accuracy whereas in case of linear kernel accuracy was limited to 84%. This concludes that a model with RBF kernel will be preferred in this scenario.



(a) Diabetes Training (b) Diabetes Test Set (c) ROC "lin_kernel" (d) ROC "RBF_kernel"

Figure 16: Visualizations for Diabetes dataset

2 Exercise Session 2: Function Estimation and Time Series Prediction

2.1 Support vector machine for function estimation

Note: Was not able to run this part of the experiment as the **uiregress** did not work on my MacOS. Followed the instructions and downloaded the latest toolbox from toldeo. The error reads ***"qp.mexmaci64 cannot be opened because the developer cannot be verified."***

2.2 A Simple Example: the sinc function

This part of the exercise was to explore LS-SVM regression model on artificially created dataset using **sinc** function. Different experiments were performed in order to understand the role of different parameters in regression problems. The experiments, and the results obtained will be discussed in the following sub-sections.

2.2.1 Regression of the sinc function

For this part of the experiment, an LS-SVM regressor was constructed with RBF kernel. Different range of gamma and sig2 parameters were explored. Based on the results obtained, following questions were answered.

1. *Try out a range of different gam and sig2 parameter values and visualize the resulting function estimation on the test set data points. Discuss the resulting function estimation. Report the mean squared error for every combination.*
   Answer: The LS-SVM regression model was trained using different combinations of sig2 and gamma values. The values used for this purpose are sig2 $=10^{-3}$, 10^{-2}, 10^{0}, and 10^{2}, and 10^{3} and gamma $= 10^{0}$, 10^{1}, 10^{2}, 10^{3} and 10^{4}. For all the possible combinations, MSE values can be seen in the heat map plotted in figure 17 below.

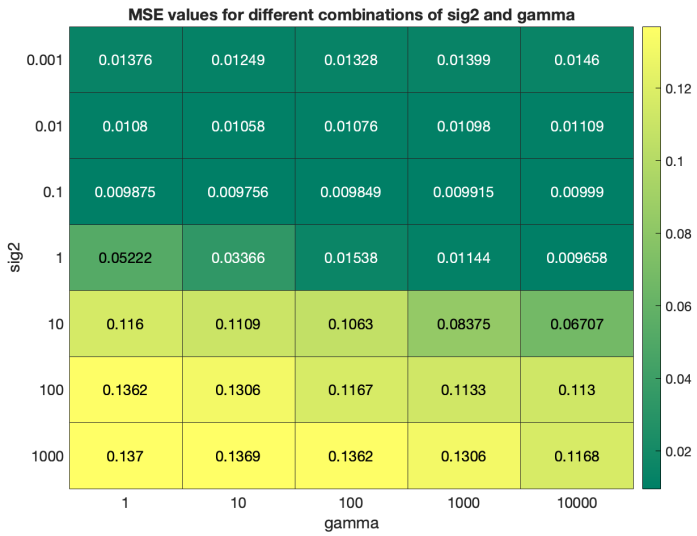
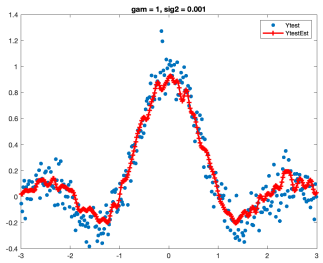
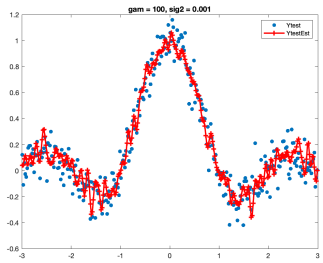


Figure 17: MSEs obtained for different combinations of sig2 and gamma

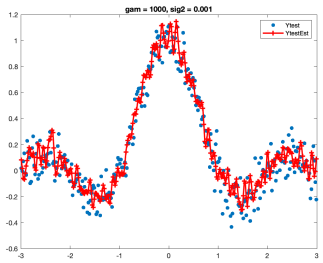
The plots for some of the combinations, showing function estimation on test set data points can be seen in figure 18 below. From these plots it can be clearly seen that in case of gamma, higher value results in overfitting, whereas very low value of gamma under-fits the data. In case of sig2, the higher value resulted in a more linear model.



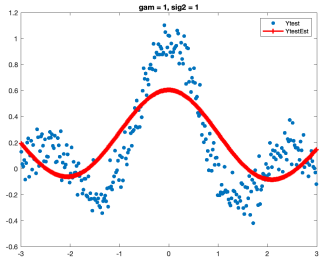
(a) $sig2 = 0.001$, $gamma = 1$



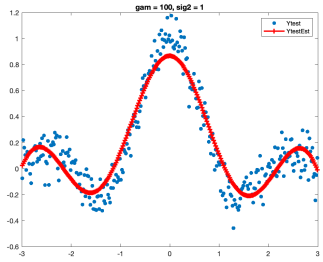
(b) $sig2 = 0.001$, $gamma = 100$



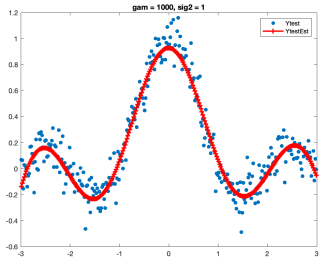
(c) $sig2 = 0.001$, $gamma = 1000$



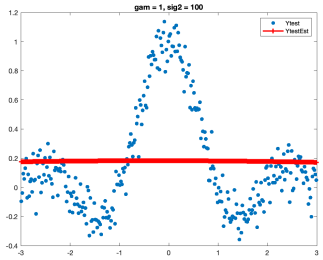
(d) $sig2 = 1$, $gamma = 1$



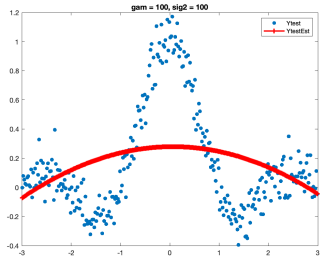
(e) $sig2 = 1$, $gamma = 100$



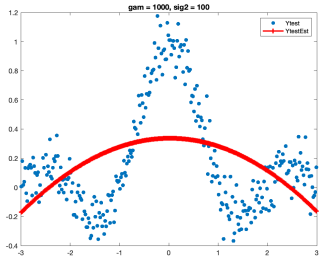
(f) $sig2 = 1$, $gamma = 1000$



(g) $sig2 = 100$, $gamma = 1$



(h) $sig2 = 100$, $gamma = 100$

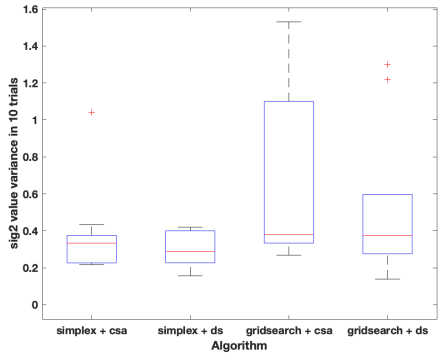


(i) $sig2 = 100$, $gamma = 1000$

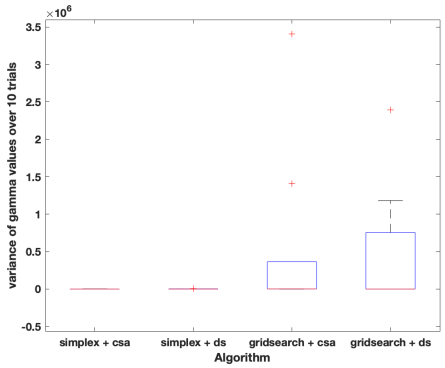
Figure 18: Plots showing the effect of changing sig2 and gamma parameters on LS-SVM regression

2. *Is there one optimal pair of hyperparameters? Why or why not?*
   Answer: The above experiment shows that there can be multiple combinations that might result in lowest MSE value. Therefore, the statement that only one pair of optimal hyperparameters exists is not true.

3. *Tune the gam and sig2 parameters using the* ***tunelssvm*** *procedure. Use multiple runs: what can you say about the hyperparameters and the results? Use both the* ***simplex*** *and* ***gridsearch*** *algorithms and report differences.*
   Answer: 10 trials were performed for tunelssvm in order to get the optimal parameters for the model. The variance in these values was large, this can be seen in the box plots in figure 19.



(a) Variance in sig2 values over 10 trails of tunelssvm



(b) Variance in gamma values over 10 trails of tunelssvm

Figure 19: Variance in hyperparameter values

2.2.2 Application of the Bayesian framework

Bayesian framework was used another method for hyper parameter tuning. The working of this hyper-parameter tuning technique is based on three interconnected levels. Interpreting the calls given in this part of the exercise, it can be seen that the first level estimates the parameters *weight* and *bias*. Level 2 estimates the *gamma* and in the end *sig2* is estimated at level 3.

2.3 Automatic Relevance Determination

The figure 20 below visualize the results obtained after using **Automatic Relevance Determination** for feature selection and therefore training the model with selected features.

This automatic relevance determination technique ranks the input features based on the corresponding sig2 value. Smaller the sig2 value, higher the relevance and vice versa. Input features with the with the largest sig2 values can be dropped.

This input selection can also be performed using cross-validation instead of ARD. This can be done by training multiple models using one feature at a time. The model with lowest MSE should be the one with most relevant feature.

2.4 Robust regression

In this section of the exercise, the effect of outliers on the function estimation was analysed. Based on the experiment, following questions were answered.

1. *Visualize and discuss the results. Compare the non-robust version with the robust version. Do you spot any differences?*
   Answer: After training the LS-SVM on data with outliers, it was identified that standard LS-SVM is not robust against the outliers. In order to handle such cases, there exists different approaches such as use of different loss functions or assigning weights to the data points such that outliers have low weights while training. Huber loss, logistic loss, are the examples of loss functions used in the first approach.
   Figure 21 shown below captures the effect of outliers and the solution using *huber* loss function. In figure 21 (a), a shift in the *sinc* curve can be seen due to the outliers present

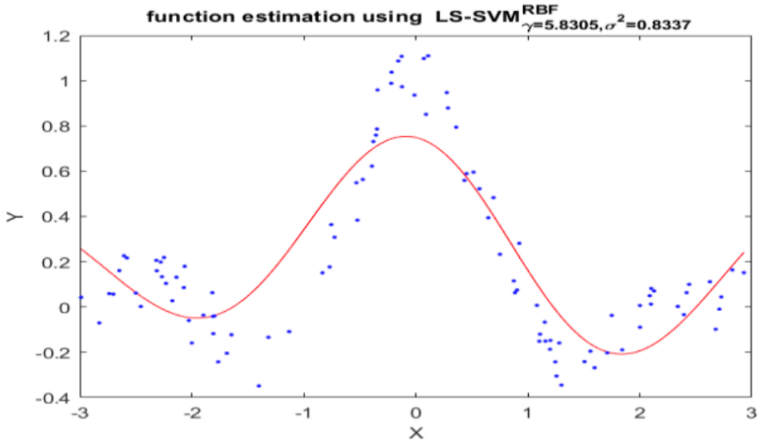
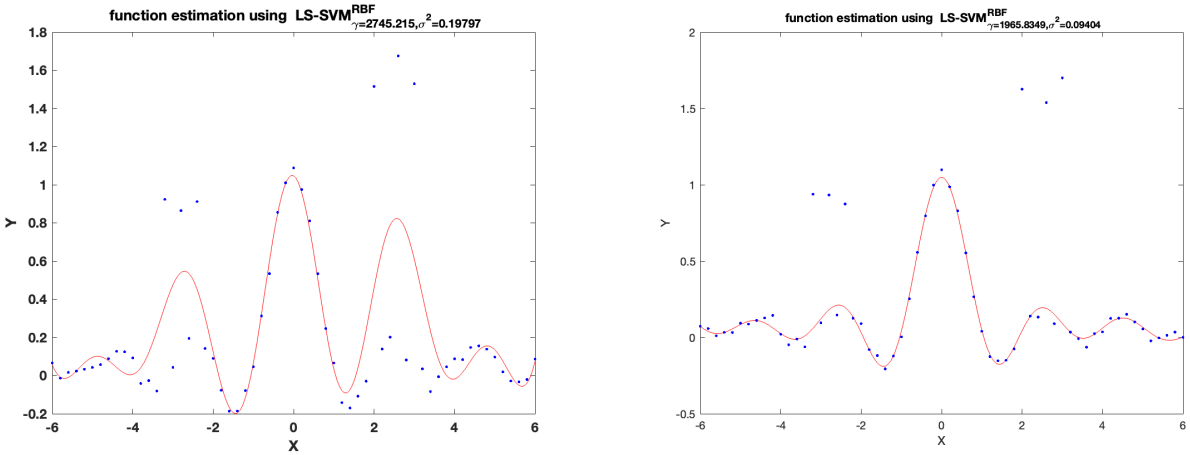


Figure 20: Automatic Relevance Determination

in the data set. While in figure 21 (b), the the impact of outliers is minimised, and a more accurate approximation of the estimation function can be seen.



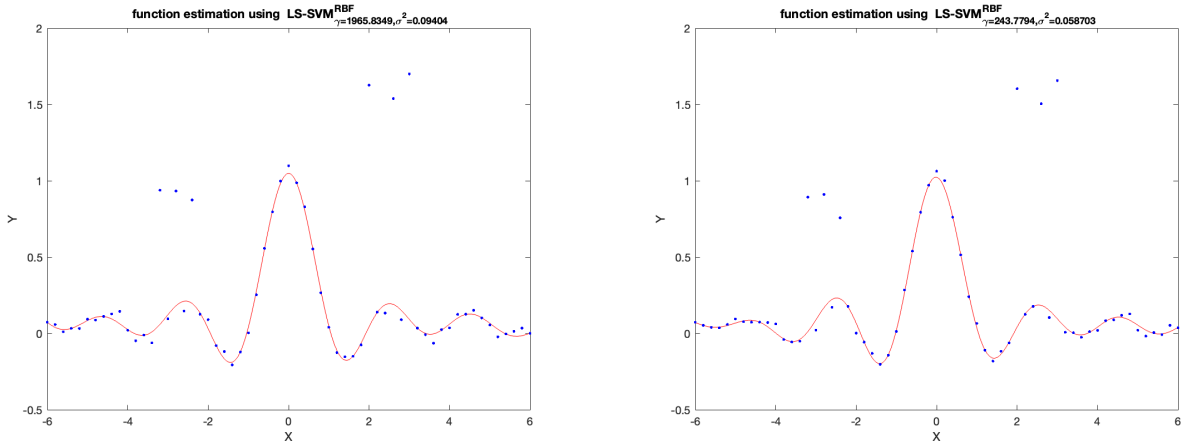
(a) Standard LS-SVM

(b) LS-SVM with huber loss

Figure 21: Handling of outliers using loss function

2. *Why in this case is the mean absolute error ('mae') preferred over the classical mean squared error ('mse')?*
   Answer: MSE(Mean Squared Error) and MAE(Mean Absolute Error) are two different performance metrics used for hyperparameter tuning in k-fold crossvalidation. In case of **MAE**, the mean of all the absolute distances between the predicted and the actual values are used for model evaluation. Whereas, in **MSE** the squared distance is minimised.

   In this case, where the data contains outliers, MAE is preferred over MSE because of the fact that MSE is more sensitive to outliers. This is because it uses the squared distances to minimize the error, which in case of robust regression are higher. The plots comparing the the use of MSE and MAE for a robust regression can be seen below in figure 22.

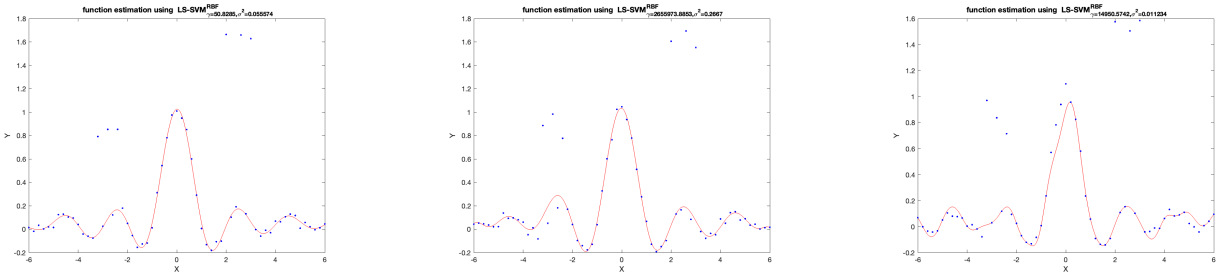


(a) Use of MSE in hyperparameter tuning (b) Use of MAE in hyperparameter tuning

Figure 22: Comparison of using MAE vs MSE in k-fold crossvalidation for hyperparameter tuning

Additionally, it can be noted that the optimal values (gamma, sig2) in case of MAE(1955.8, 0.094) are lower than in case of MSE(243.79, 0.068).

3. *Try different alternatives to the weighting function wFun (e.g., 'whampel', 'wlogistic 25' and 'wmyriad 29'. Report on differences.*
   Answer: In order to identify the optimal training parameters, k-fold cross validation was used. The effect of using different weight functions was experimented in this part of the exercise. Different weight functions used were 'whampel', 'wlogistic', and 'wmyriad 29'. The sinc curves for these can be seen in figure 23 below.



(a) MAE whampel (b) MAE wlogistic (c) MAE wmyriad

Figure 23: Iris Classification: Polynomial Kernel with different values for parameter *d*.

From the above plots, it can be clearly seen that whampel gave the best results followed by wlogistic and wmyriad. Wmyriad was the worst performer where the approximation curve seems to a bit shifted from the actual points. Additionally, the number of iteration taken to converge also varied with change in weight functions. The model took 29 iterations to converge in case of wmyriad function, this number came down to 25 in case of wlogistic. Whampel took least number of iterations, i.e 3 iterations, to converge. Looking at the overall results, it can be concluded that whampel was the most preferable weight function in this case.

Even though, if we compare these above three with the initially used weight function, i.e

whuber, all these still converge quite fast. Whuber took 43 iterations to converge and find the best parameters.

2.5 Homework Problems: Time Series Prediction

The aim of this exercise was to explore the use svm regression on time series data. All the knowledge gained during the previous tasks of this exercise session such as hyperparameter tuning, were applied in context of time series prediction. The datasets used for this purpose were **Logmap** and **Santa Fe** datasets. The results obtained after experiments will be discussed in the respective subsections.

2.5.1 Logmap dataset

Logmap was the first time series dataset provided for this part of the exercise. The goal here was time-series prediction with lowest error possible. For this purpose, the first task was to investigate the optimal hyperparameters gamma, sig2 and order. In case of time series data, the added parameter "order" or "lag size" corresponds to the number of previous instances to be taken into account before making prediction.

Parameter Tuning Strategy: In order to tune the *order* parameter, like sig2 and gamma, k-fold cross validation can be used. Another approach can be the use of Automatic Relevance Determination (ARD) technique. This will work in the following manner [1]:

$inputs = bay\_lssvmARD(Xu(:, 1 : delays), Xu(:, end),' f', gam, sig2,' RBF\_kernel');$
$[alpha, b] = trainlssvm(Xu(:, inputs), Xu(:, end),' f', gam, sig2,' RBF\_kernel');$
$prediction = predict(Xu(:, inputs), Xu(:, end),' f', gam, sig2,' RBF\_kernel', Xt);$

Note: One drawback of this technique is that it works sub-optimally for recurrent models.

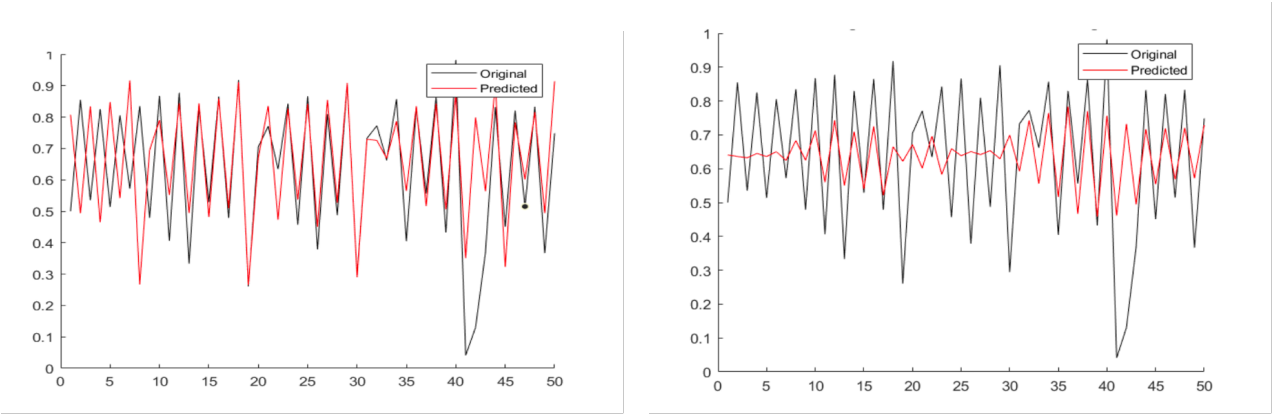
From the previous experience of outliers experiment, both performance metrics MAE and MSE were used to find the optimal parameters. The results obtained showed that performance was better when MAE was used for parameter tuning. This reason for this trend might be the presence of outliers in the dataset. The visualization can be seen in figure below.

2.5.2 Santa Fe dataset

The second dataset available in time series prediction part was santa fe dataset. The dataset consists of 1000 training instances while 200 test instances. The aim, once again, was to predict the future values by training SVM models and evaluate the model performance on test set. A validation set might also be the another choice to evaluate the model performance. The only think that should be considered is that the validation set should be in the time series pattern. If the validation set is shuffled, the predictions will be invalid.

For training the model, the order or lag size was initialized to 50 in order to verify if it is a good choice or not. The hyperparameter tuning for (sig2, gamma) was done using the same technique used in previous logmap dataset using **tunelssvm** procedure. In the next

[1] https://www.esat.kuleuven.be/sista/lssvmlab/tutorial/node23.html



(a) Optimal Results using MAE, (sig2, gamma, order) = (2, 5, 10)

(b) Optimal Results using MSE, (sig2, gamma, order) = (3, 90, 30)

Figure 24: Time Series Prediction with tuned parameters under different evaluation criteria MAE(left) and MSE(right)

step, different lag sizes (15, 16, 17, ..., 78, 79, 80) were compared for lowest MAE values. The graph showing the effect of different lag sizes on MAPE error can be seen in figure 25 below:

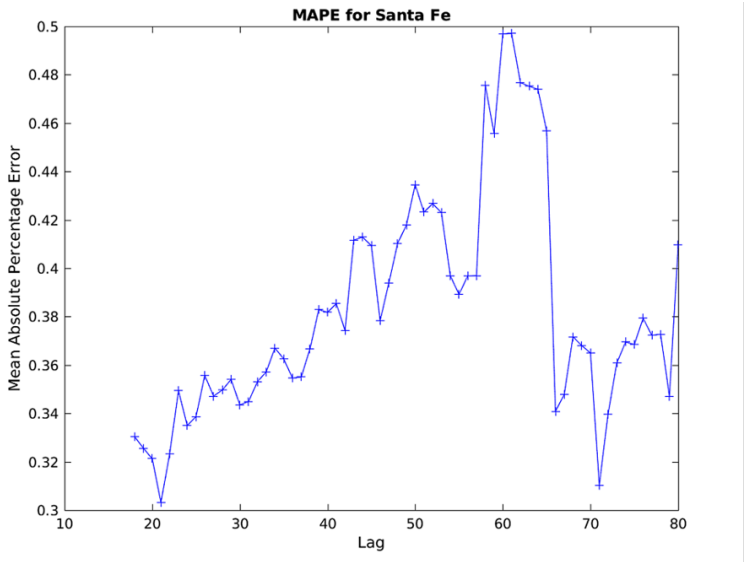


Figure 25: Lag-size vs MAPE for Santa Fe dataset

From the above graph, it can be clearly seen that order = 50 performed decently well but there were other lag sizes as well which gave even better results. The most optimal value for order was observed as 21 where the lowest MAPE [2] = 0.303 was obtained.

[2]Besides from MAE, MAPE values were also calculated.
More info about MAPE can be found at: https://en.wikipedia.org/wiki/Mean_absolute_percentage_error

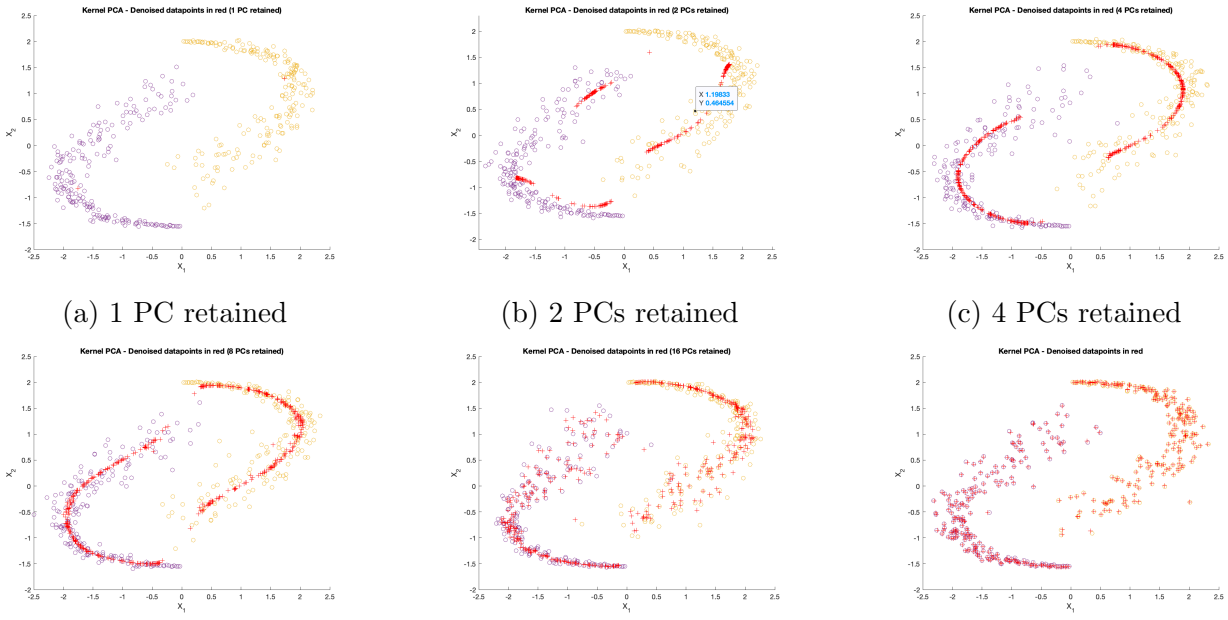
3 Exercise Session 3: Unsupervised Learning and Large Scale Problems

3.1 Kernel principal component analysis

In this part of the exercise, KPCA (Kernel Principal Component Analysis) will be explored. Different experiments will be carried out in order to explore the role of different parameters involved in the process. This section will be discussing about these experiments and their results on detail.

As a part of this exercise, script **kpca_script.m** available from toledo was used to illustrate the power of PCA for denoising. For this purpose, logmap dataset was used with 400 data points, divided into two clusters of 200 points each. The original dataset was reconstructed using (1, 2, 4, 8, 16, and 32) principal components. In each round of experiment, the value of sig2 was fixed to 0.4. Based on these experiments performed, following questions were answered.

1. *Describe how you can do denoising using PCA. Describe what happens with the de-noising if you increase the number of principal components.*
   Answer: While performing the process of denoising, two major steps are followed. Firstly, principle components are limited and then PCA is performed by projecting the data along the chosen components. Secondly, the original dataset is reconstructed using the projected data in step 1. When the number of principle components are increased, the de-noising is not achieved as expected because reconstructed data replicates the original dataset. This effect of increasing the number of principal components can be seen in figure 26 below.



(a) 1 PC retained (b) 2 PCs retained (c) 4 PCs retained

(d) 8 PCs retained (e) 16 PCs retained (f) 32 PCs retained

Figure 26: Effect of changing number of principle components for de-noising. Here, the reconstructed principle components are plotted in red while the data points are plotted in two clusters, blue and green.

2. *Compare linear PCA with kernel PCA. What are the main differences? How many principal components can you obtain?*
   Answer: The major difference between linear PCA and the kernel PCA is that in case of linear PCA, the components extracted are linear combinations of the original dataset. On the contrary, the components in case of kernel PCA can be non-linear. This projection of data points on a linear axis, in case of linear PCA can be seen in figure 27 below.

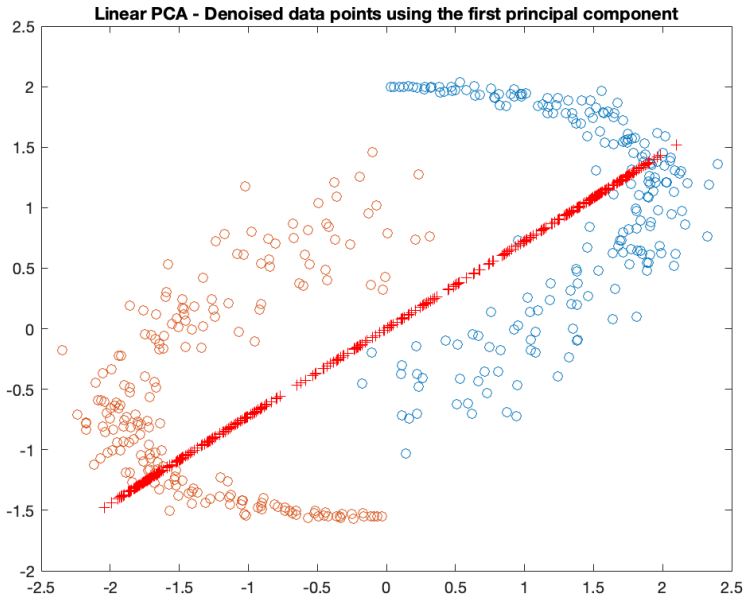


Figure 27: Linear Projection of components in case of linear PCA

   Another difference between linear and kernel PCA is in number of principle components extracted. In case of linear PCA, the maximum number of components that can be extracted is limited to dimension of feature space p. Whereas in case of kernel PCA, the maximum principle components can be as many as data points N.

3. *For the dataset at hand, propose a technique to tune the number of components, the hyperparameter and the kernel parameters.*
   Answer: In order to tune the number of components, the hyperparameter and the kernel parameters, **k-fold crossvalidation** technique can be used with MSE as the evaluation metric.

3.2 Spectral clustering

This part of the exercise deals with spectral clustering, which is another form of kernel PCA. In these techniques, eigenvectors of a laplacian matrix derived from the data are used to group similar data points. The script **sclustering_script.m** from toledo, along with **two3drings** dataset were used for the experiments. Based on the results obtained, following questions were answered.

1. *Explain briefly how spectral clustering works and how is it different from classification?*
   Answer: Spectral clustering is a type of unsupervised learning technique used to group similar data together.

It is different from classification as in case of classification the aim is to label the data not grouping it, as in case of spectral clustering.

2. *Try out different values of sig2. What is the influence of the sig2 parameter on the clustering results?*
   Answer: The experiments were performed for sig2 = 0.0001, 0.01, 1, and 100. Different plots for each value of sig2 can be seen in figures 28 to 31 below. In these figures, the original data is plotted left, followed by kernel matrix of original and stored data after clustering in middle and Projection onto subspace plotted on the right.

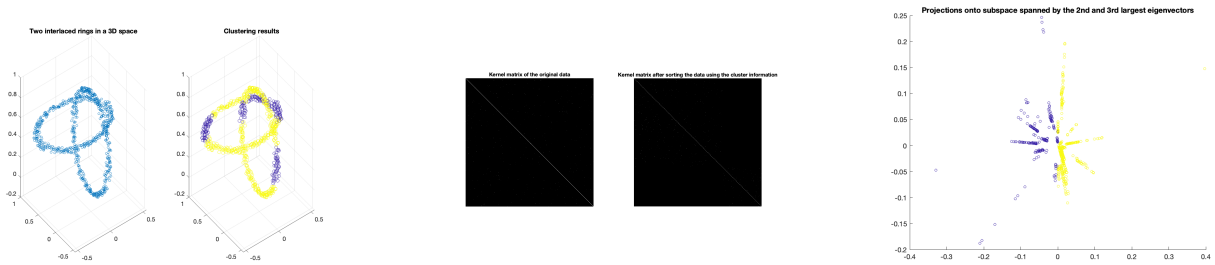


Figure 28: Spectral clustering with sig2 = 0.0001

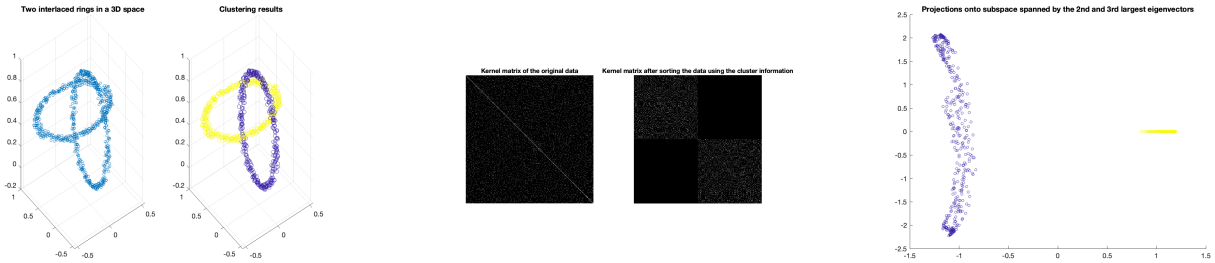


Figure 29: Spectral clustering with sig2 = 0.01

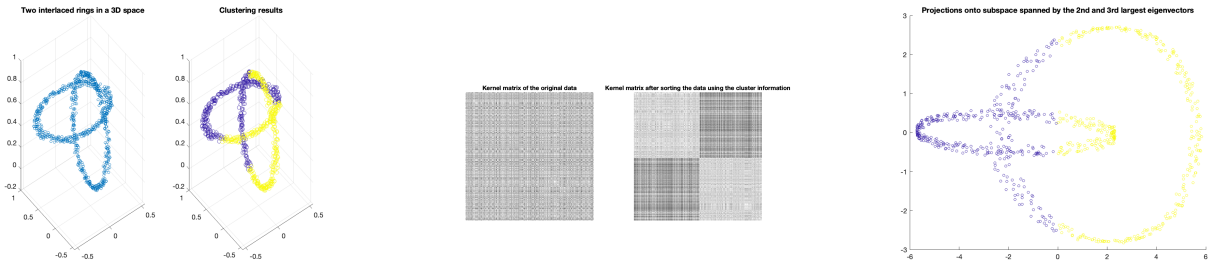


Figure 30: Spectral clustering with sig2 = 1

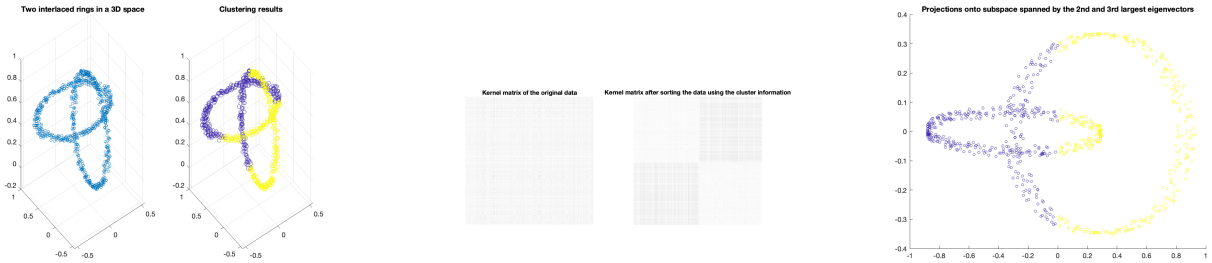


Figure 31: Spectral clustering with sig2 = 10

From the above plots, the role of sig2 parameter can be seen such that, for a very low value like 0.001 all the training points were treated as similar thus resulting in overlaps. After increasing the sig2 value to 0.01, the results were best as the points closer in original input space were treated similar leading to both the rings learnt perfectly. Further increasing sig2 to 1, resulted in overlaps and these remained same for sig2 = 10. This means that the model converged somewhere around these values of sig2.

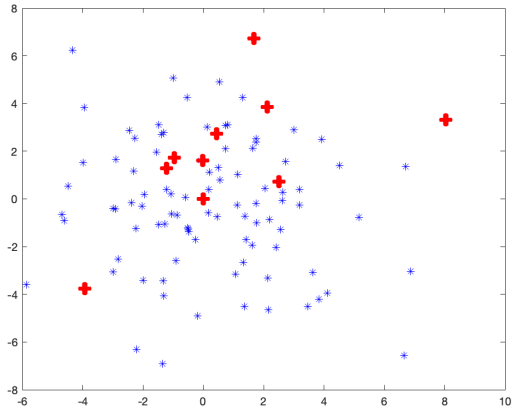
3.3 Fixed-size LS-SVM

This exercise aims to study the working of fixed size LS-SVM. This technique works by fixing the number of support vectors in advance and then using the iterative methods to select the best decision boundary learning subset. Various experiments were performed in order to investigate the role of hyper-parameters such as sig2, kernel, number of support vectors and many more. The results obtained are discussed in the answers below.

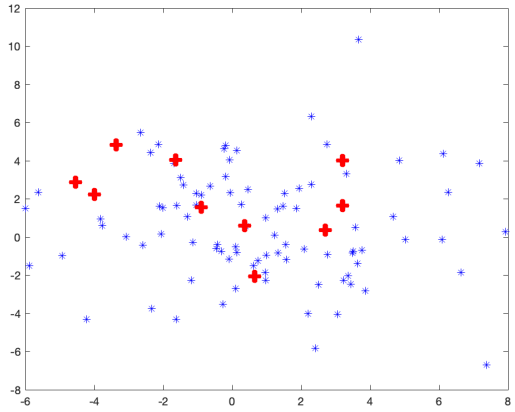
1. *In which setting would one be interested in solving a model in the primal? In which cases is a solution in the dual more advantageous?*
   Answer: Number of training data points and dimensions of the input space are the major parameters which helps to decide solving the model in primal or dual form. If a dataset contains the large number of training data points, primal space is preferred as dual space for such cases may lead to expensive computations. Dual space may be handy for small and high dimensional dataset.

2. *What is the effect of the chosen kernel parameter sig2 on the resulting fixed-size subset of data points. Can you intuitively describe to what subset the algorithm converges?*
   Answer: For the experiment, four different values of sig2 were selected i.e. 0.0001, 0.01, 1 and 10. The plots can be seen in figure 32. The major observation which can be made form these figures is that increase in value of *sig2* leads to increase in distance between any two set of points.

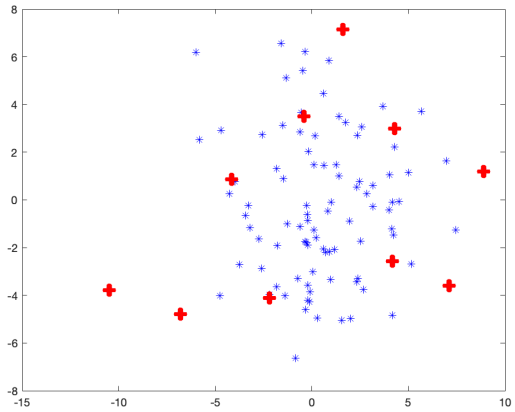
   The algorithm converges to subset of fixed size that best describes the structure of the data i.e subset with maximum entropy.



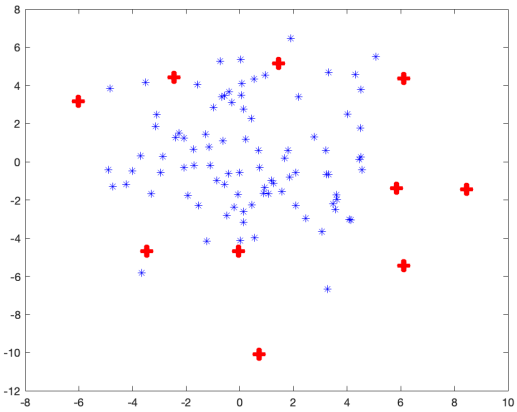
(a) *sig2* = 0.0001



(b) *sig2* = 0.01



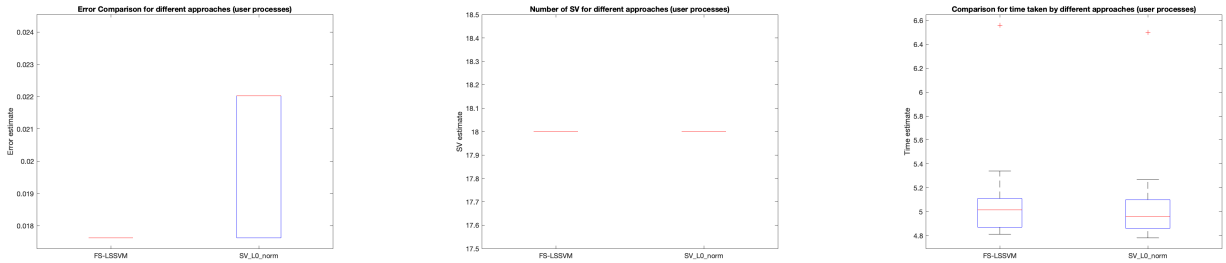
(c) *sig2* = 1



(d) *sig2* = 10

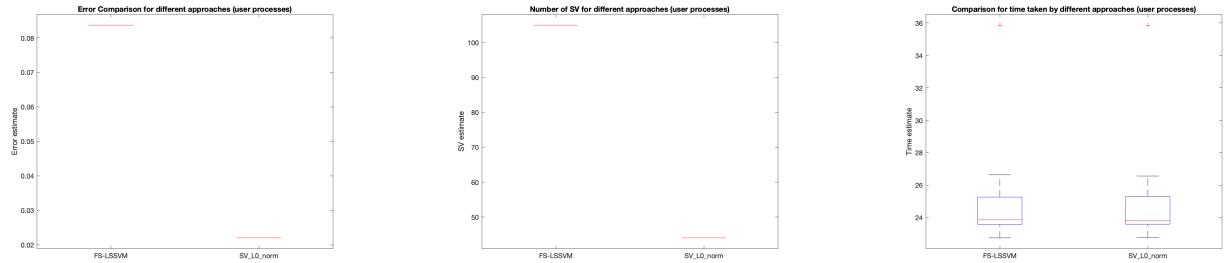
Figure 32: Effect of changing *sig2* on Fixed Size LS-SVM

3. *Compare the results of fixed-size LS-SVM to ℓ_0 - approximation in terms of test errors, number of support vectors and computational time.*
Answer: The comparison was made between LS-SVM and ℓ_0 - approximation based on test error, number of support vectors and the computational time taken. The dataset used for this purpose was Breast cancer data (classification). Both kernel options i.e. using linear kernel and polynomial kernel were used. The box-plots showing the results can be seen in figure 33 and figure 34. The box plots in the figure shows that FS-LSSVM performed better in case of linear kernel whereas ℓ_0 - approximation had the upper hand while using polynomial kernel. Time taken by both the techniques is quite similar with both of them taking higher time in case of polynomial kernel. Talking about the number of support vectors, the value was similar for both the techniques in case of linear kernel. However in case of polynomial kernel, FS-LSSVM took way more support vectors, more than 100 ,as compared to ℓ_0 - approximation in which the number was limited to 25.



(a) Error Comparison (b) Number of SVs (c) Run Time Comparison

Figure 33: Comparison between LS-SVM and ℓ_0 - approximation with **linear_kernel**. Error comparison (Left), Number of Support Vectors used in each case (Middle) and Computational time taken by each technique (Right)



(a) Error Comparison (b) Number of SVs (c) Run Time Comparison

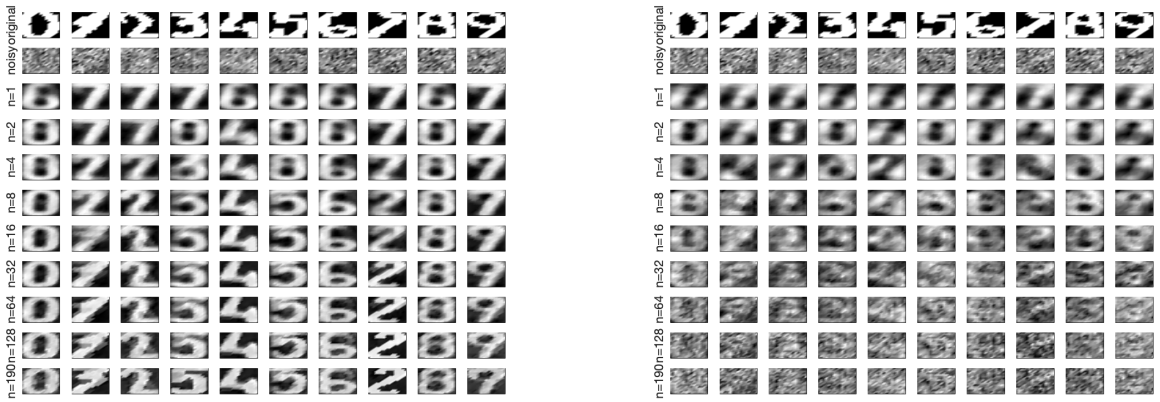
Figure 34: Comparison between LS-SVM and ℓ_0 - approximation with **poly_kernel**. Error comparison (Left), Number of Support Vectors used in each case (Middle) and Computational time taken by each technique (Right)

3.4 Homework Problems

As the final part of this exercise, different datasets were provided to experiment with. The goal was to apply all the knowledge gained using previous experiments on these datasets. **Handwritten digits** dataset was provided to explore Kernel PCA whereas **Shuttle** and **California** datasets were made available to further explore Fixed size LS-SVM. The observations for the same will be discussed in the respective subsections.

3.4.1 Kernel principal component analysis

De-noising the data using kernel PCA technique was further experimented on the **digits** dataset. First part of the experiment was to identify the difference between linear and kernel PCA with noise factor set to 1. The sigmafactor in this case was set to 0.7. The plots for the same can be seen below in figure 35. From this figure it is well evident that kernel PCA performed well in denoising the data and learning the digits, in comparison to linear PCA. Even after increasing the number of components to 190, kernel PCA performed better in learning the digits.



(a) Kernel PCA

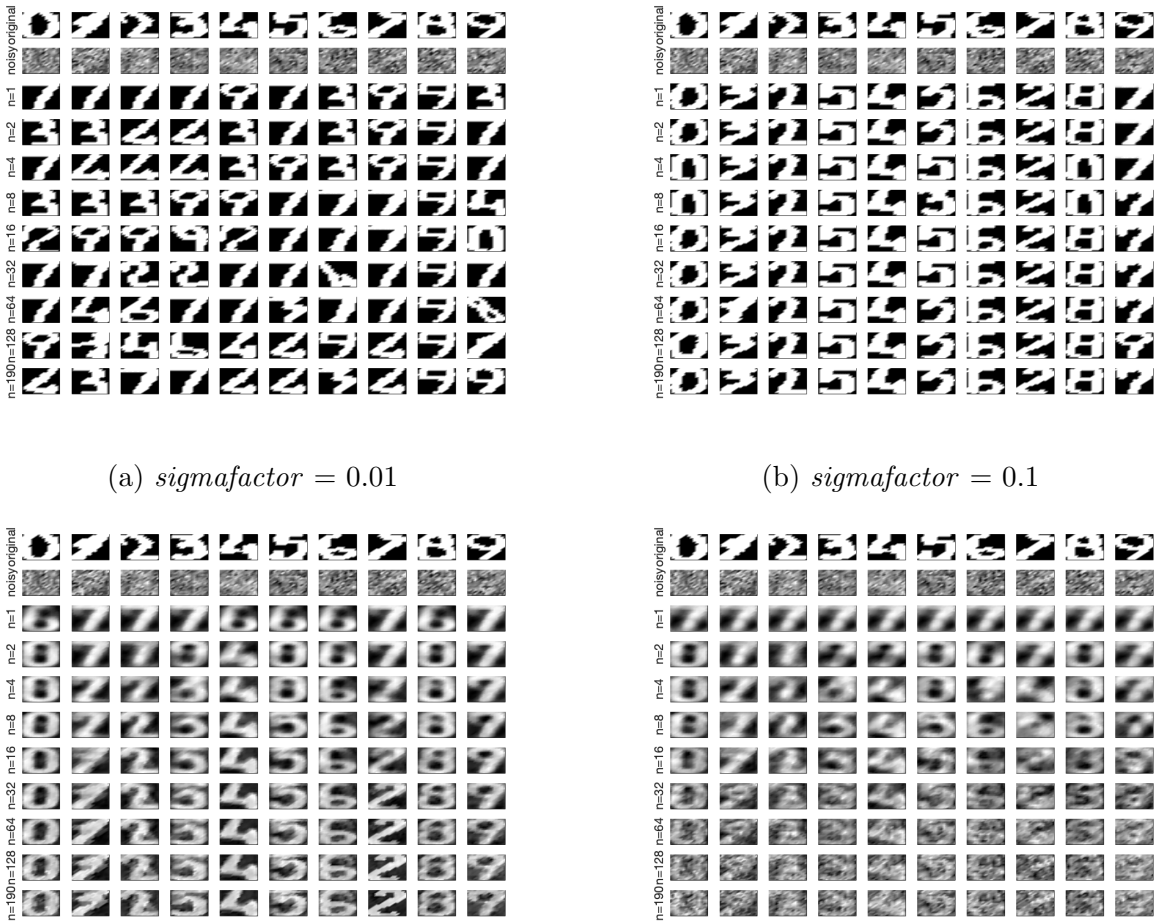
(b) Linear PCA

Figure 35: Difference between kernel and linear PCA with noise factor = 1 and sig2 = 35.91

It is a well evident fact, from the experience of all the exercises performed for SVM, that the value for sig2 parameter plays a significant role in the performance of the SVM models. As a rule of thumb for kernel PCA, sig2 $= p * E[var(x_i)]$ for $i = 1..p$, where p is the number of features in the dataset. According to this equation, the sig2 for handwritten dataset becomes 35.91. Changing the value of sig2 might degrade the performance of the model.

The next task was to illustrate the effect of change in value of *sigmafactor*. Inspired form the better performance of the kernel PCA in the above experiment, it was decided to use this in further experiments with sigmafactor. The values for sigmafactor were chosen on the logarithmic scale. For different values of sigmafactor (on logarithmic scale) = 0.01, 0.1 , 1 and 10. The performances for these values can be visualized using figure 36. Note that the noise in both the cases remains 1, similar to previous experiment.

From the figure it can be seen that for sigmafactor = 0.01, the model wasn't able to learn the digits properly. Only one out 10 digits was learned correctly. The performance improved a little while using sigmafactor = 0.1, where the model was able to learn couple of digits correctly. The best performance was observed at sigmafactor = 1, where model learned most of the digits correctly. At even higher value of 10, the model was again not able to learn anything. The optimal value of sigma can once again be computed using k-fold crossvalidation with reconstruction error as performance evaluation criteria.



(a) *sigmafactor* = 0.01

(b) *sigmafactor* = 0.1

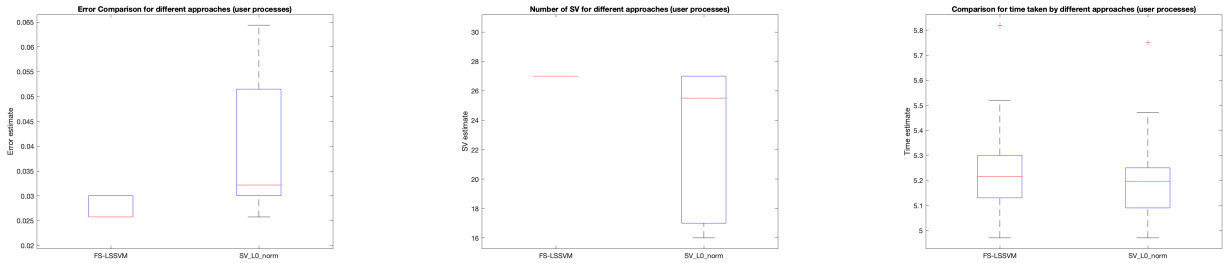
(c) *sigmafactor* = 1

(d) *sigmafactor* = 10

Figure 36: Effect of changing *sigmafactor* on learning handwritten digits using kernel PCA

3.4.2 Fixed-size LS-SVM

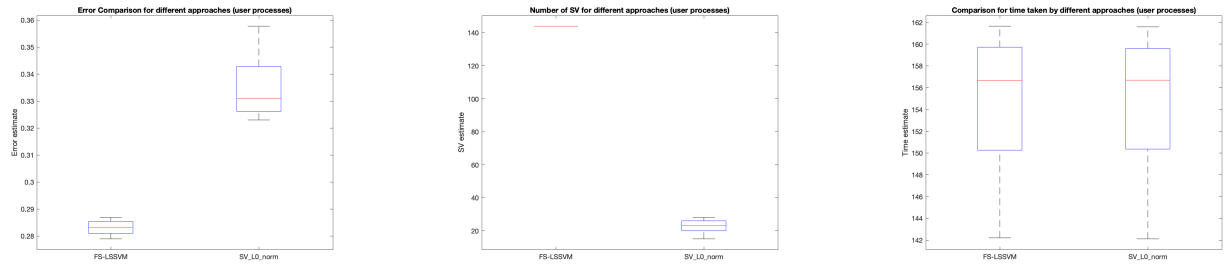
1. **Shuttle Data** was provided as one of the two datasets to explore Fixed-size LS-SVM. This is a classification dataset with 58,000 data points distributed within 7 different classes. The data points in the dataset were not linearly separable giving the intuition that linear kernel might not be able to properly differentiate between the classes. Hence RBF kernel was used for both FS-LSSVM and SV_LO_NORM. The experiments were performed with (k = 1) in order to reduce the computation time. The plots showing the comparison of two techniques, based on error, number of support vectors and the computation time taken can be seen in figure 37. The figure shows that FS-LSSVM performed better for shuttle dataset where the error estimate was between 0.025 to 0.030. This in case of SV_LO_NORM was much higher. Not much difference was recorded in computational time taken by both these techniques.



(a) Error Comparison (b) Number of SVs (c) Run Time Comparison

Figure 37: **Shuttle Dataset:** Comparison between LS-SVM and ℓ_0 - approximation. Error comparison (Left), Number of Support Vectors used in each case (Middle) and Computational time taken by each technique (Right)

2. **California Data** was the second dataset made available for this part of the exercise. This dataset was a regression dataset with 20,640 9-dimensional data instances. With k set to 1 for fast computation, once again FS-LSSVM and SV_LO_NORM were compared for function estimation. The kernel used was RBF, based on past experience of better results on RBF kernel. The plots showing the comparison of two techniques, based on error, number of support vectors and the computation time taken can be seen in figure 38. The figure shows that once again FS-LSSVM performed better in terms of error estimate. There was not much difference observed in computational time taken by two techniques. However this time was much larger if we compare it with shuttle dataset.



(a) Error Comparison (b) Number of SVs (c) Run Time Comparison

Figure 38: **California Dataset:** Comparison between LS-SVM and ℓ_0 - approximation. Error comparison (Left), Number of Support Vectors used in each case (Middle) and Computational time taken by each technique (Right)