KU LEUVEN

Support Vector Machines: Methods and Applications

Exercise Sessions

Meng YUAN (r0601195)

Master of Artificial Intelligence

Academic year 2017-2018

1 Exercise Session 1: Classification

1.1 A Simple Example: Two Gaussians

Figure 1.1.1 shows intuitively the geometric construction of the optimal classifier for linear non-separable data. The artificial data was generated from two classes of Gaussians with the same covariance matrices, and red/blue circles represent the positive/negative classes respectively. The optimal decision boundary is a linear separating hyperplane, which is constructed by maximizing the distance to the nearest points of the two classes (i.e. the margin) and at same time tolerating misclassifications.

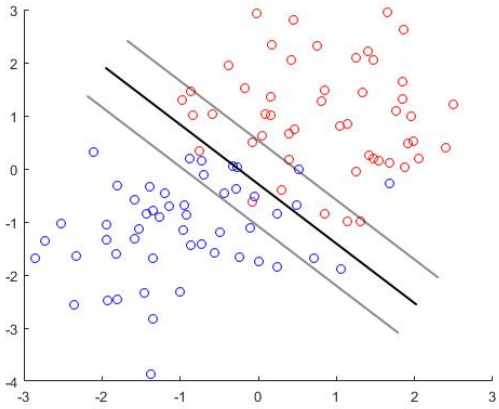


Figure 1.1.1

1.2 The Support Vector Machine

Figure 1.2.1 gives experiment results for a classification problem using the linear kernel. According to figure (2), when we added data points close to the previous decision boundary, the new decision boundary adjusted accordingly because the newly added points would work as support vectors. When added points were far away from the previous decision boundary (but still belong to correct side), they did not have much influence on decision boundary. According to figure (3), when outlying points were added, the classification boundary changed more dramatically if the dataset was still separable; otherwise they would be tolerated as misclassified points. C parameter characterizes how much emphasis is put on the penalty for misclassification. According to figure (4), when C was small, more emphasis was placed on maximizing the margin, the classifier allowed some misclassified data points, the distance between the hyperplanes was larger, there were more support vectors. According to figure (5), when C was large, the classifier became intolerant to misclassified data points and would try very hard to fit all the data points, the margin became smaller. After switching to RBF kernel with C=1 and *sigma*=1 as shown in figure (6), the data set was more properly classified. By applying the nonlinear classifier with RBF kernel, it resulted in a curvy classification boundary with more points used as support vectors and less misclassified cases than linear kernel.

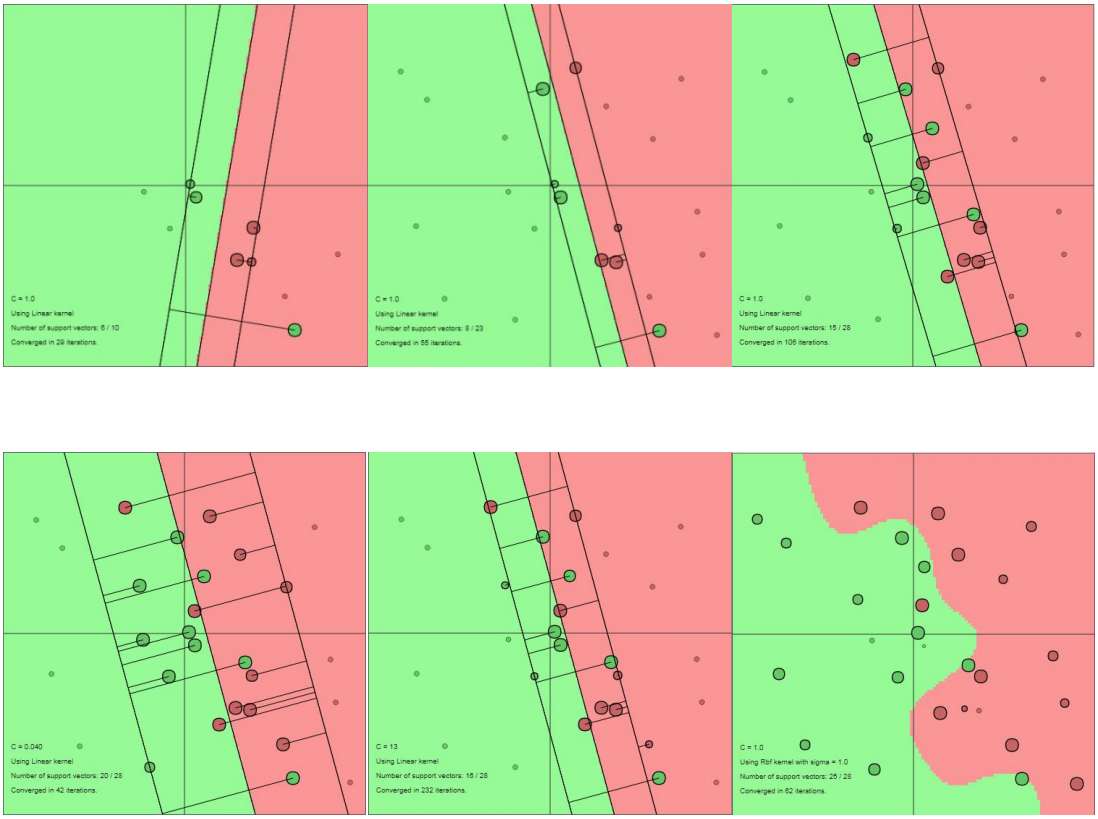


Figure 1.2.1 Classification results (1)original C=1 (2)adding points C=1;

adding outlying points (3) C=1 (4) C=0.04 (5) C=13 (6)RBF kernel, C=1, *sigma*=1

Figure 1.2.2 gives experiment results with varying *sigma* values for a classification problem using the RBF kernel. The *sigma* parameter describes how far the influence of a single training data point reaches. With too small *sigma*, the decision boundary was entirely dependent on individual data points. The resulting "islands" shows that clearly the model overfitted the data. When *sigma* was very large, the radius of the influence of support vectors was too large, the model was too constrained to be able to capture the shape of the data, which might lead to underfitting. The resulting model behaved similarly to a linear classifier. Since the data is almost linearly separable, we do not need to set a large value for C parameter to penalize the misclassification. As shown in figure (2), C=1 and *sigma*=1 could be a good choice.

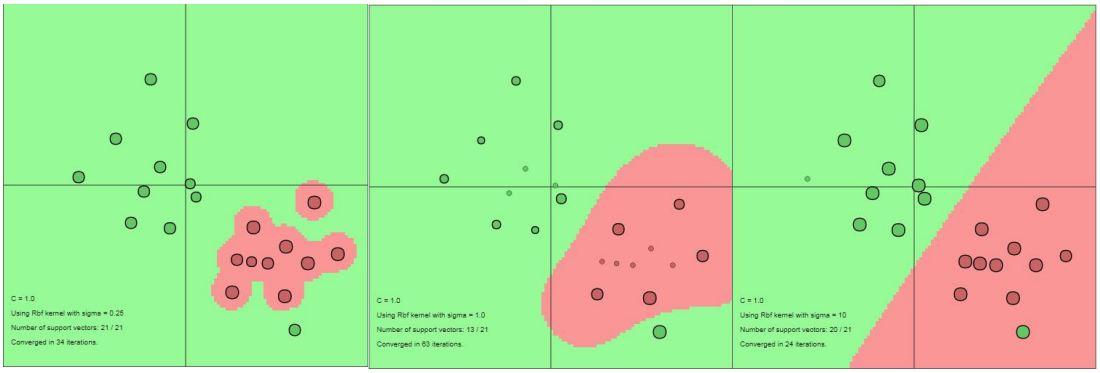


Figure 1.2.2 Classification results C=1, (1) sigma=0.25 (2) sigma=1 (3) sigma=10

The decision boundary is only depended on supports vectors instead of all training data points. Geometrically the support vectors are located close to the decision boundary. The closer to the decision boundary, the more importance of the support vector. Figure 1.2.3 shows that when we added the point near the hdecision boundary, it became a support vector; when we added the point far from the decision boundary, it did not influence the result much.

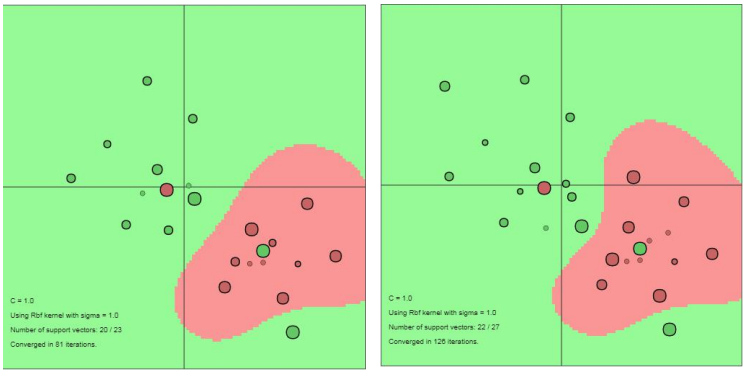


Figure 1.2.3 Classification results C=1 sigma=1, number of support vectors (1)20/23 (2)22/27

For the linearly non-separable data with an overlapping area between two classes, by using RBF kernel we map the input data into a high dimensional feature space, in which the linear separating hyperplane can be constructed. The kernel parameter *sigma* represents how much influence of a single training data point and decides whether it will be selected as support vector. The regularization parameter C characterizes the tradeoff between the misclassification of training examples and complexity of the model. In Figure 1.2.4, as the value of parameter C decreases, the corresponding importance of support vectors increase (the dots becomes larger, which corresponds to the value of Lagrange multipliers).

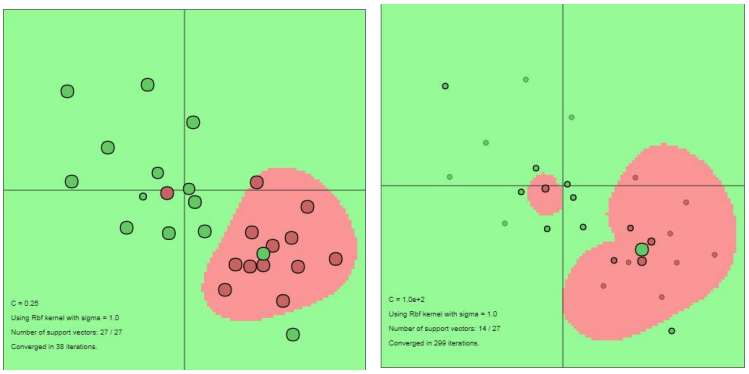


Figure 1.2.4 Classification results sigma=1, (1) C=0.25 (2) C=100

1.3 Using LS-SVMlab for Iris data

In this session, we applied LS-SVM for the Iris data with different choices of kernel functions and model parameters. In Table 1.3.1, it provides the performance on the test set for different settings.

When linear kernel was applied, it resulted in 55% error rate on the test set. When polynomial kernel with degree equal to 1 was applied, it also gave 55% error rate because it worked simply the same as a linear kernel. These two kernel functions are clearly not sufficient to classify the non-linear relationship between features. By increasing the degree, the model becomes more flexible. The error rates equaled 5%, 0 and 0 respectively for polynomial kernels with degree of 2, 3 and 4. When degree rose to 3, the kernel was flexible enough to discriminate between two classes. However, as shown in Figure 1.3.1, the degree-4 polynomial yielded a very curvy decision boundary, indicating overfitting. Like the *sigma* hyperparameter of the RBF kernel, the degree of a polynomial kernel controls overfitting.

Kernel Type	**Parameter Values**	**Error rate on Testset**
Linear Kernel	gam = 1	55%
Polynomial Kernel	gam=1, t=1, degree=1	55%
	gam=1, t=1, degree=2	5%
	gam=1, t=1, degree=3	0
	gam=1, t=1, degree=4	0
RBF Kernel	gam=1, sig2=0.01	10%
	gam=1, sig2=0.1	0
	gam=1, sig2=1	0
	gam=1, sig2=5	0
	gam=1, sig2=10	0
	gam=1, sig2=25	50%
	gam=0.1, sig2=0.1	10%
	gam=1, sig2=0.1	0
	gam=5, sig2=0.1	0
	gam=10, sig2=0.1	0
	gam=100, sig2=0.1	5%
	gam=1000, sig2=0.1	5%

Table 1.3.1 Performance on Testset for different settings

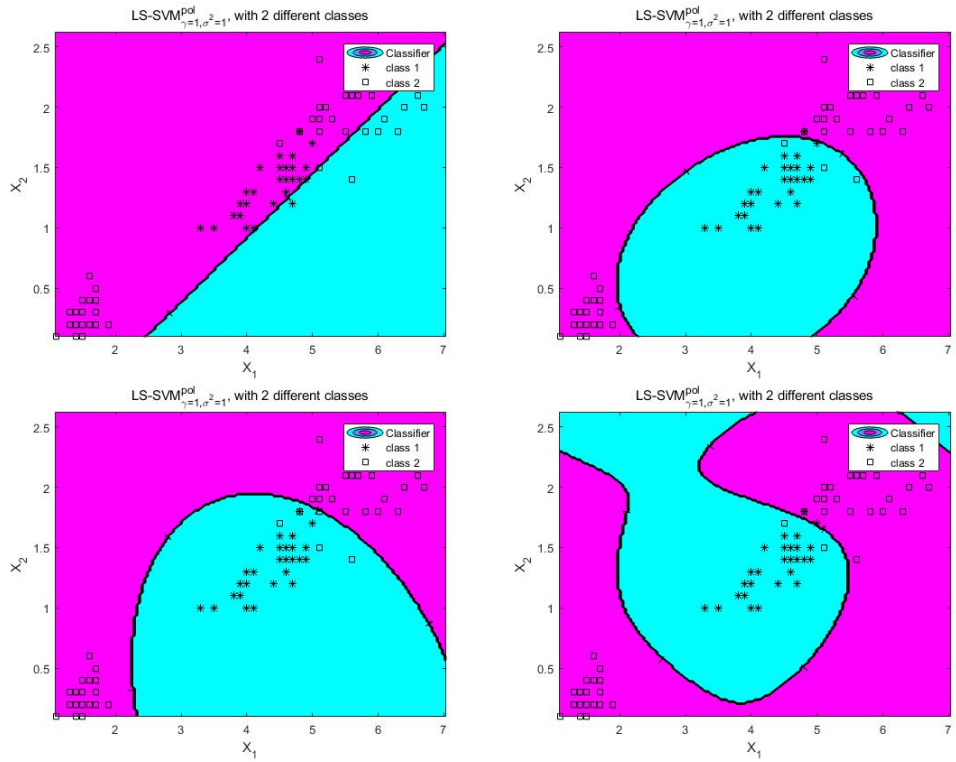
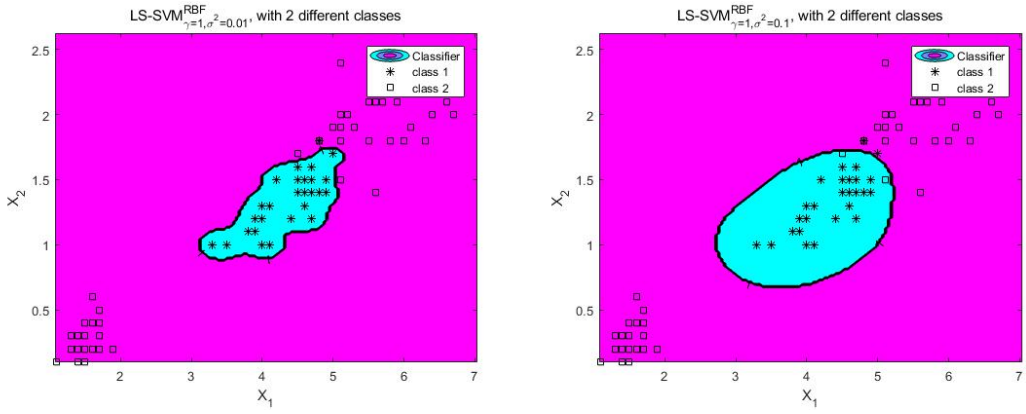


Figure 1.3.1 Classification results for Polynomial Kernel (1) d=1 (2)d=2 (3) d=3 (4) d=4

Figure 1.3.2 gives experiment results with fixed *gamma* and varying *sigma* values using the RBF kernel. The *sigma* parameter describes how far the influence of a single training data point reaches. With too small *sigma* (*sig2*=0.01), the decision boundary was entirely dependent on individual data points. It lead to greater curvature of the decision boundary, indicating overfitting. As *sigma* increases, the decision boundary became smoother. When *sigma* was very large (*sig2*=25), it resulted in only one class output. According to performance plot of different *sigma* values in Figure 1.3.3 and error rate in Table 1.3.1, from 0.1 to 10 can be a good range for *sig2*.



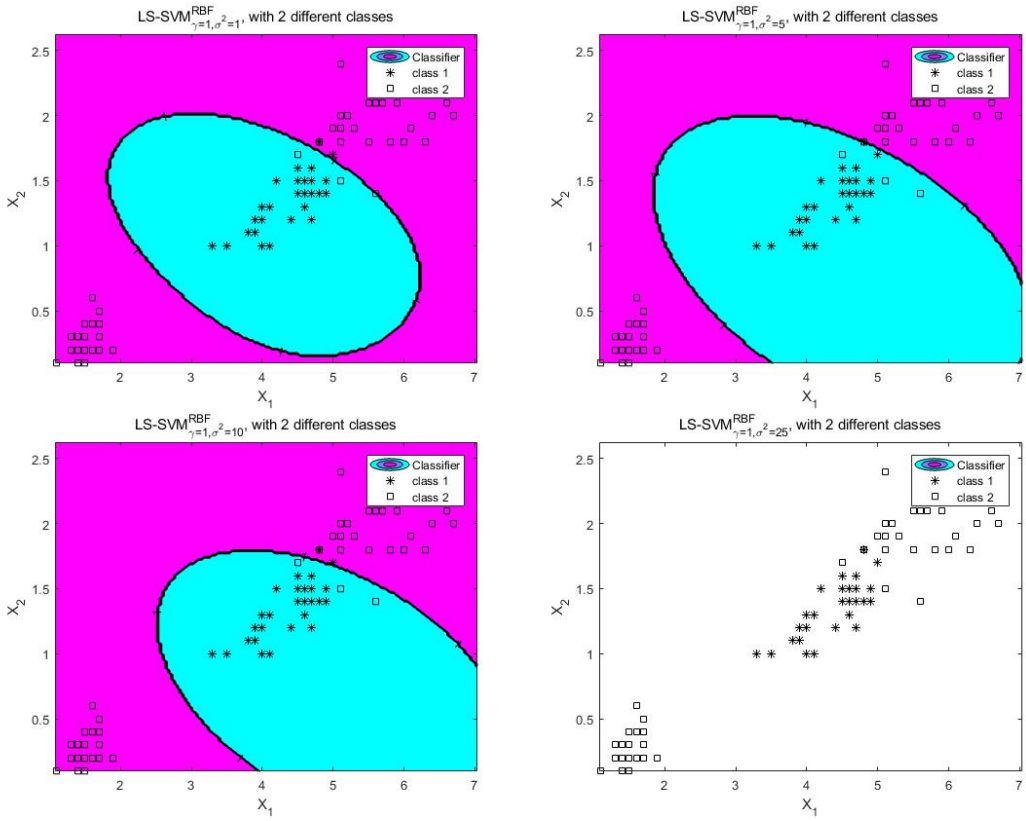


Figure 1.3.2 Classfication results for RBF Kernel gam=1 (1) sig2=0.01 (2) sig2=0.1 (3) sig2=1 (4) sig2=5 (5) sig2=10 (6) sig2=25

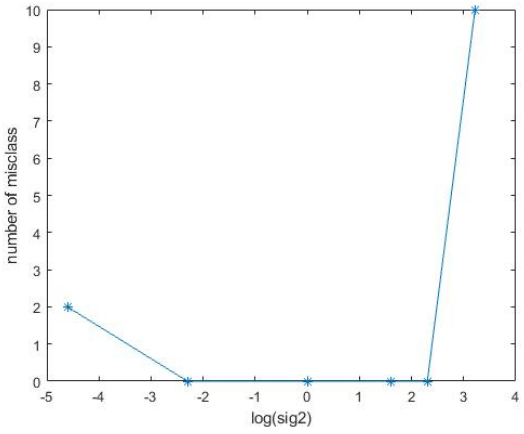


Figure 1.3.3 Performance on Test set with different values of sig2

Figure 1.3.4 gives experiment results with fixed *sigma* (*sig2*=0.1) and varying *gamma* values using the RBF kernel. *Gamma* is the regularization parameter, determining the trade-off between the fitting error minimization and smoothness. A smaller *gamma* means more tolerance on missclassification errors and more emphasis on margin maximization. When *gamma* was too large, the classifier would try very hard to fit all the data points, resulting decision boundary with great curvature, as shown in Figure 1.3.4. According to performance plot of different *gamma* values in Figure 1.3.5 and error rate in Table 1.3.1, from 1 to 10 can be a good range for *gamma*.

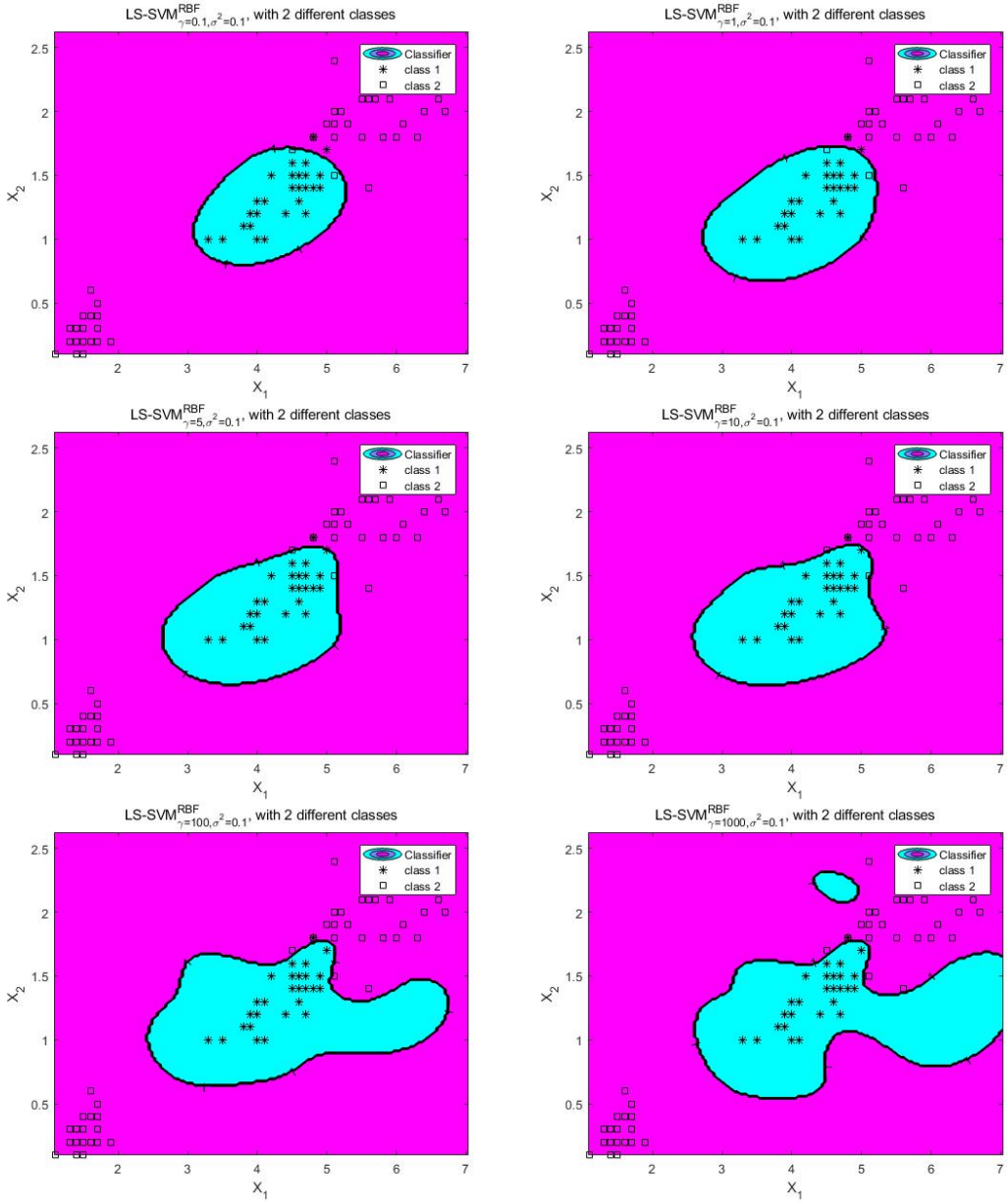


Figure 1.3.4 Classfication results for RBF Kernel sig2=0.1 (1) gam=0.1 (2) gam=1 (3) gam=5 (4) gam=10 (5) gam=100 (6) gam=1000

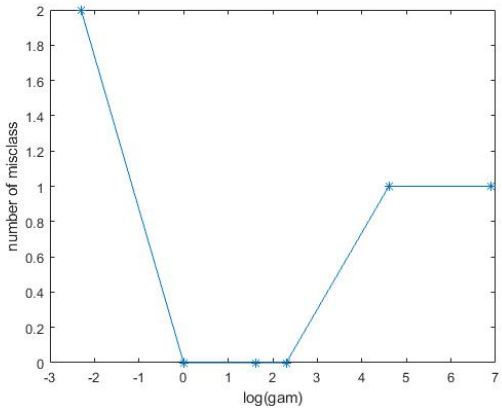


Figure 1.3.5 Performance on Test set with different values of gam

To make better choice of hyperparameters, we can divide the given data into a training set and a validation set. The training set is used to learn the model parameters; the validation set is used to tune the hyperparameters; the test set is used for testing the predictive power of the final model, so it should be brand new data which has not been used in the model building process. In Figure 1.3.5, it provides the performance in terms of misclassification error on the validation set (train:val=8:2) for different values of *gam* and *sig2*. From 0.1 to 10 could be a good range for *sig2* and from 0.1 to 100 can be a good range for *gam*.

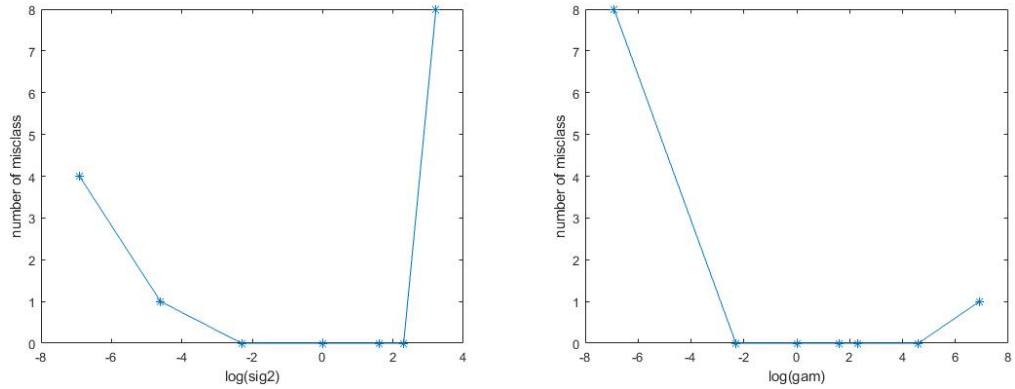


Figure 1.3.5 Performance on validation set with different values of gam and sig2

Besides using a random chosen subset as validation set, we can also apply 10-fold cross-validation or leave-one-out procedure. For 10-fold cross-validation method, the data set is divided into 10 subsets. Each run, one of the 10 subsets is used as the validation set and the other 9 subsets are used as a whole for training. The process is repeated for 10 times and the average error across all 10 trials is computed. For leave-one-out method, each run, the model is trained on all the data except for one point and then validation is performed on that point. The process repeats for N (number of input points) separate times and the average performance is computed. Compared to random choosing validation set, 10-fold cross-validation and leave-one-out method are more structural. Every data point gets to be in validation set exactly once, and gets to be in training set 9 (N-1 for leave-one-out) times. Choosing validation set randomly often gives results that differ a lot since it is influenced a lot by how the data gets divided. However, the disadvantage of 10-fold cross-validation is that the algorithm has to be rerun for 10 times, which means it takes 10 times as much computation to make an evaluation; for leave-one-out method, it is even more expensive to compute. For small data set, leave-one-out method is often preferred because we want to make full use of the information. According to Figure 1.3.6 and Figure 1.3.7, two approaches provide similar results of performance.

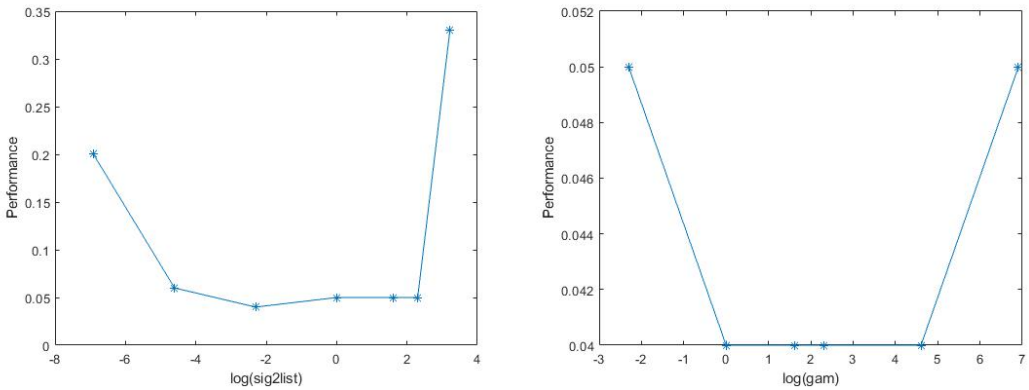


Figure 1.3.6 Performance using 10-fold cross-validation

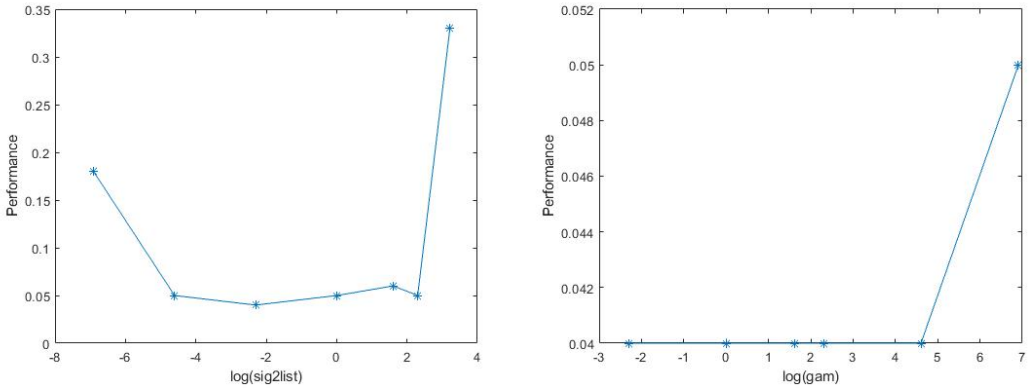


Figure 1.3.7 Performance using leave1out

Next, we applied *tunelssvm* to optimize the hyperparameters. The optimization process consists of two steps: first, determine good initial starting values by global optimization method (e.g. coupled simulated annealing (CSA), Randomized Directional Search (DS)) and then the obtained suitable initials are given to a second optimization procedure (*simplex* or *gridsearch*) to perform a fine-tuning step. In Table 1.3.2, it provides estimated hyperparameters for different choices of optimization algorithms. The resulting *gamma* values differ a lot, varying from 10^{-2} to 10^{5}; the obtained *sig2* values are relatively more stable, varying from 10^{-2} to 10^{2}. When the resulting costs are the same for some trails, it means the obtained pairs of *gam* and *sig2* are equally good. The optimization process is actually solving a non-convex problem, which has multiple local minimums. For both *simplex* and *gridsearch* algorithm, in order to determine the minimum of a cost function with possibly multiple optima, they evaluate sub-spaces over the parameter space and find the minimum iteratively. These two methods differ in how search space is divided, by *simplex* or *grid*. No significant difference of the results from CSA and DS methods are found.

Optimazation Alogorithm	Trial	Gamma	Sig2	Cost
CSA+gridsearch	1	0.15	6.78	0.02
	2	3218.53	0.10	0.02
	3	8094.03	0.08	0.03
	4	29.72	0.17	0.04
	5	0.19	0.17	0.03
CSA+simplex	1	0.06	1.54	0.03
	2	1552.22	0.10	0.03
	3	3.79	19.33	0.04
	4	0.07	1.34	0.03
	5	2.44	0.34	0.04
DS+gridsearch	1	3161.14	0.32	0.02

	2	0.39	0.09	0.03
	3	27880.70	0.05	0.02
	4	1.42	0.03	0.04
	5	0.11	1.90	0.04
DS+simplex	1	3336.94	0.08	0.03
	2	12520.40	0.07	0.02
	3	165.38	0.67	0.04
	4	0.16	1.64	0.04
	5	122.89	3.59	0.04

Table 1.3.2 Hyper-parameters for different optimization algorithms

One alternative way to judge a classifier is by using the Receiver Operating Characteristic (ROC) curve. The higher the area under the curve, the better the performance. For making the ROC curve on the training data is not recommended because it could not represent the generalizability of the model. The ROC plot on the validation set is shown in Figure 1.8.3.

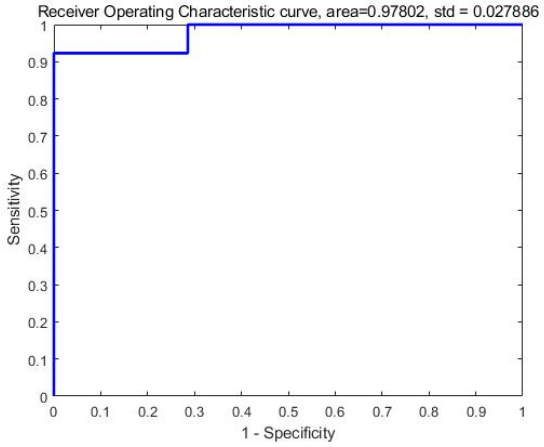


Figure 1.3.8 ROC curve on validation set (gam=1, sig2=2)

1.4 Homework Problems

1.4.1 The Ripley Data-set

In this session, we applied LS-SVM classifier on the Ripley dataset. The Ripley dataset contains 250 training data points and 1000 test data points. The scatter plot of the 250 training data points is given in Figure 1.4.1.1. The dataset consists of two classes where the data points for each class were generated by a mixture of two Gaussian distributions. There exists a high degree of overlap between two classes. Based on visually check, a LS-SVM classifier using linear kernel might be applicable with

some tolerable misclassified points. The RBF kernel is universal applicable so it was also used. In Table 1.4.1.1, it provides a summary of performance for different methods of several runs. The hyper-parameters were chosen by applying 10-fold cross validation (*tunelssvm*).

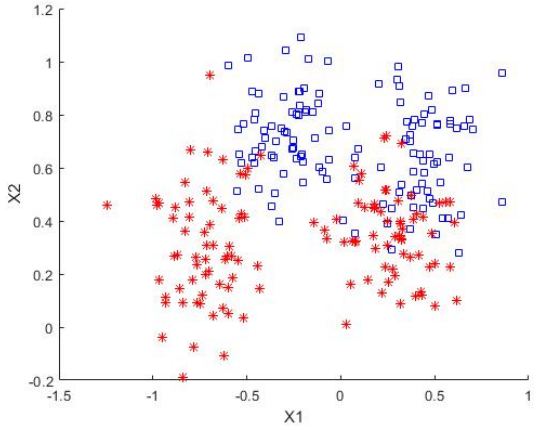


Figure 1.4.1.1 Scatter plot of the Ripley dataset

For linear model, it resulted in 14.8% 10-fold cross validation error rate on validation set, 10.4% misclassification rate on test set , and AUC value of 0.957. It can be concluded that the linear model performs well. For RBF kernel, it resulted in around 12.4% 10-fold cross validation error rate on validation set, 9.4% misclassification rate on test set , and AUC value of 0.968. Compared to linear model, the performance of SVM model with RBF kernel improves a bit. For several runs of tuning RBF hyper-parameters, the resulting *gam* and *sig2* varied; thought they all lead to similar values of 10-fold cross validation error rate. However, for one run (*gam*=9.98, *sig2*=0.05), it resulted in 13.2% misclassification rate on test set , and AUC value of 0.925, which indicates in some cases the tuned parameters are suboptimal. The linear model would be chosen as the final model because the performance by using RBF kernel did not improve much and we would prefer simple and sufficient enough model. Considering the high degree of the overlapping between classes, the SVM model with linear kernel provides about 10% test set error rate and AUC value of 0.957, would be a suitable model.

Kernel Type	Parameter Values	10-fold CV Error Rate	Error rate on Testset	AUC of ROC curve
Linear Kernel	gam=0.0061	14.8%	10.4%	0.9573
	gam = 0.0102	14%	10.4%	0.9579
	gam=0.0033	15.60%	10.6%	0.9564
RBF Kernel	gam=11.63,sig2=1.04	12.40%	9.4%	0.9684
	gam=26.35, sig2=1.69	12.80%	9.7%	0.9671
	gam=9.98, sig2=0.05	13.20%	13.3%	0.9245

Table 1.4.1.1 Performance for several trials

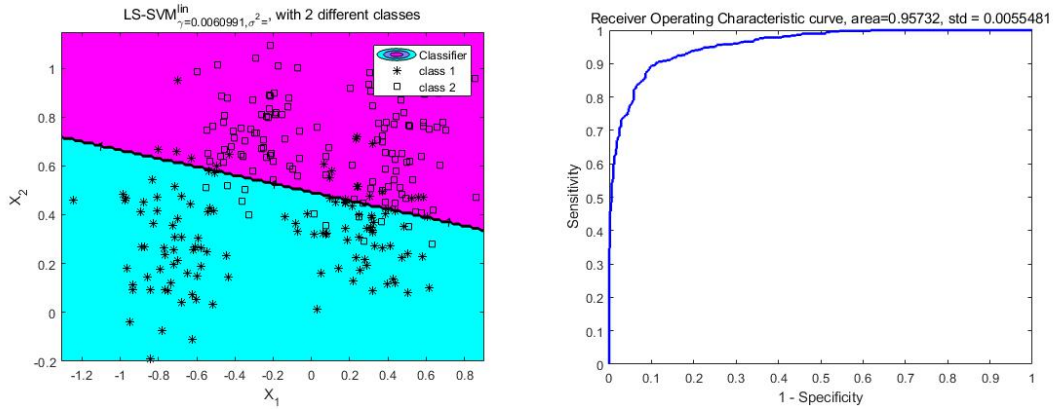


Figure 1.4.1.2 Classification results and ROC curve using linear kernel

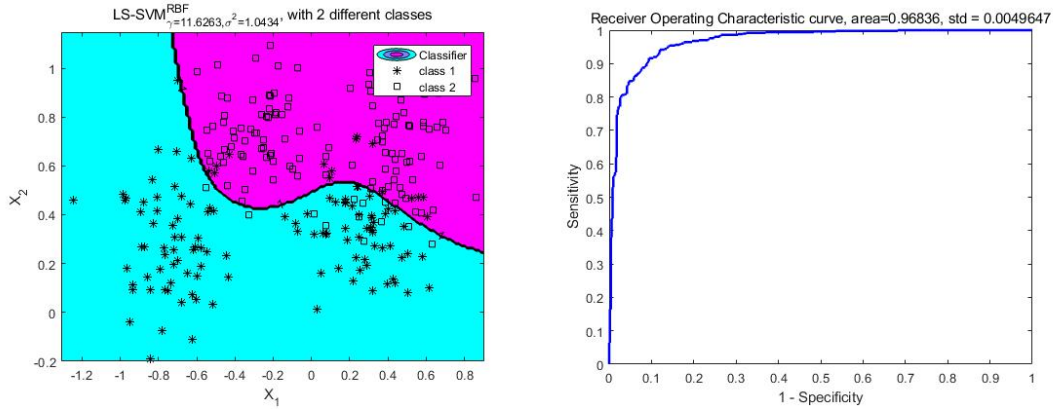


Figure 1.4.1.3 Classification results and ROC curve using RBF kernel

1.4.2 Breast Cancer Data-set

In this session, we applied LS-SVM classifier on the Breast Cancer dataset. The Breast Cancer dataset contains 400 training data points and 169 test data points, of which 357 points are in Benign class and 212 points belong to Malignant class. For both training set and test set, about 63% of the data points belong to the negative class, indicating the imbalance of the dataset. Also, the input dimension is relatively high: each data point has 30 features. First, we performed a PCA analysis to observe which features are most useful in predicting cancer type. The result shows that 98.46% of the variance is explained by the first principal component (PC1). It is also surprising that in PC1, most of the feature weight is related to the 4^{th} and 24^{th} attributes (with weights equal 0.539, 0.839 respectively). This information might be useful for clinicians. In Figure 1.4.2.1, it provides the scatter plot of the dataset on feature 4^{th} and 24^{th}(they appear to be correlated), different classes are indicated by different colors. There exists some overlap between classes. A linear model might be applicable with some tolerable misclassified points. The RBF kernel is universal applicable so it was also used.

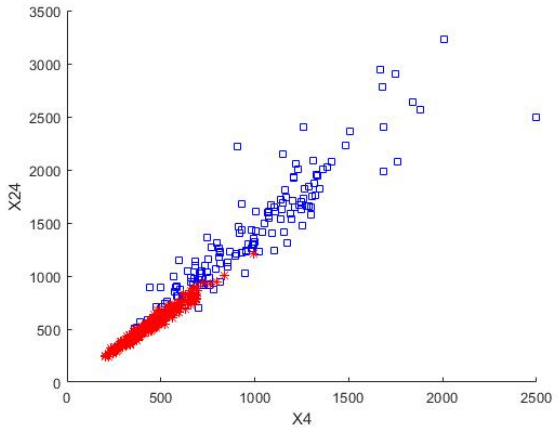


Figure 1.4.2.1 Scatter plot of Breast Cancer dataset on attribute 4th and 24th

Next, we applied both SVM method with linear kernel and RBF kernel. The performance for several runs are presented in Table 1.4.2.1. For several runs of tuning RBF hyper-parameters, the obtained *gam* and *sig2* varied. For one run (*gam*=566.85, *sig2*=119.46), it resulted in relatively high 10-fold cross validation error rate and misclassification rate on test set, and relatively low AUC value, which indicates in some cases the tuned parameters are suboptimal. According to results of several trails, linear model resulted in relatively higher error rate on test set than RBF kernel. However, linear model lead to larger values of AUC. Accuracy is computed at the threshold value of 0.5 while AUC can be seen as an average performance on all possible threshold values. Since the breast cancer dataset is unbalanced, AUC would be a preferable measure for performance. The linear model would be chosen as the final model because it is simple and sufficient enough. The final model provides about 4% test set error rate and AUC value of 0.996, meaning it is a perfectly suitable model.

Kernel Type	Parameter Values	10-fold CV Error Rate	Error rate on Testset	AUC of ROC curve
Linear Kernel	gam=0.1013	3.25%	4.73%	0.9959
	gam=0.0657	4.73%	4.00%	0.9964
	gam=0.1234	4.00%	4.14%	0.9961
RBF Kernel	gam=566.85,sig2=119.46	3.00%	4.73%	0.9855
	gam= 55.17,sig2=35.85	1.25%	2.37%	0.9938
	gam=56.35,sig2=44.96	1.25%	2.37%	0.9947

Table 1.4.2.1 Performance for several trials

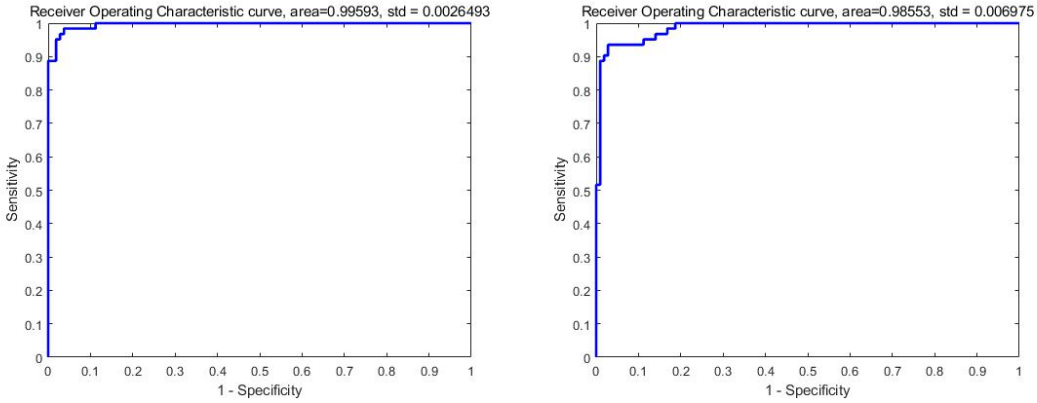


Figure 1.4.2.2 ROC curve for linear kernel (left) and RBF kernel (right)

1.4.3 Diabetes Database

In this session, we applied LS-SVM classifier on the Diabetes dataset. The dataset contains 300 training data points and 168 test data points. The ratio of negative class versus positive class is 205:95 for training set and 106:62 for test set (unbalanced). Each data point has 8 features. According to the correlation matrix graph in Figure 1.4.3.1, 2nd and 6th features are most relevant for class label, and 3rd feature is the least correlated with class label. In Figure 1.4.3.2, it provides the scatter plot of the dataset on feature 4th and 6th, different classes are indicated by different colors. Two classes appear to be heavily overlapped. A linear model might not be sufficient and the SVM method with RBF kernel might be applicable.

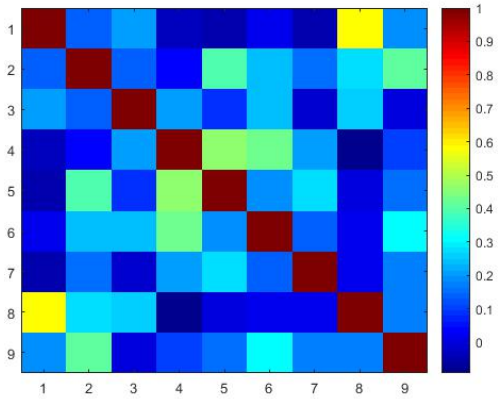


Figure 1.4.3.1 Correlation matrix plot

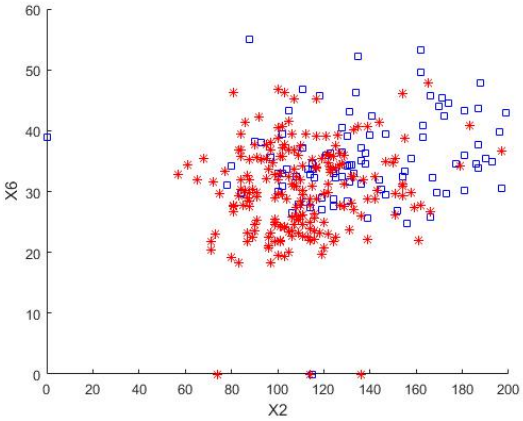


Figure 1.4.3.2 Scatter plot on attribute 4th and 6th

The performance for several runs are presented in Table 1.4.3.1. For several runs of tuning RBF hyper-parameters, the resulting *gam* and *sig2* varied a lot. Some tuned hyperparameters were suboptimal, yet most of them lead to similar performance. According to results of several trails, linear model results in relatively higher error rate on test set and relatively lower values of AUC than RBF kernel. So the SVM model with RBF kernel would be chosen as our final model. Considering the quite high overlapping between classes, the final model, providing 20% test set error rate and AUC

value of 0.85, could be a good enough model yet not perfectly suitable. Other methods could also be tested and after comparison, the best methods can be selected.

Kernel Type	Parameter Values	10-fold CV Error Rate	Error rate on Testset	AUC of ROC curve
Linear Kernel	gam=0.0221	26.33%	22.62%	0.8454
	gam=1.0098	25.67%	21.43%	0.8430
	gam=0.0113	25.00%	24.40%	0.8442
RBF Kernel	gam=538.78,sig2=1311.13	25.33%	20.24%	0.8539
	gam=2.02,sig2=96.66	24.67%	22.02%	0.8504
	gam=754.12,sig2=1983.37	26.00%	19.64%	0.8524

Table 1.4.3.1 Performance for several trials

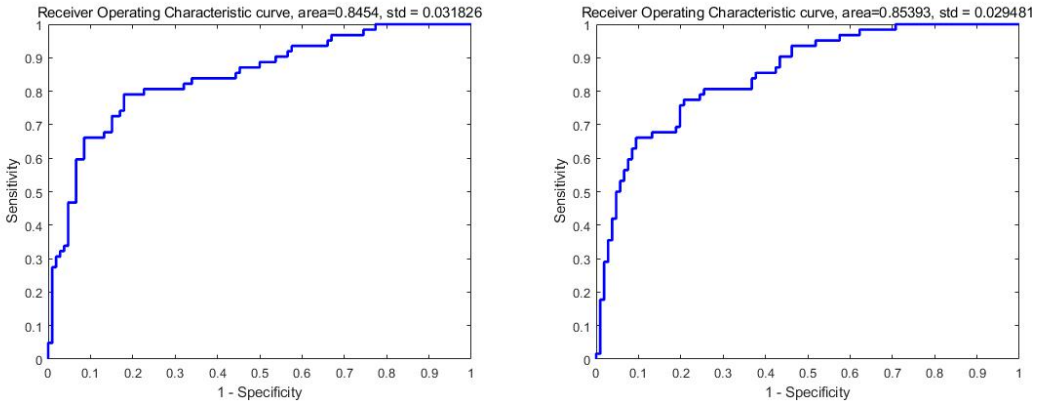


Figure 1.4.3.3 ROC curve for linear kernel (left) and RBF kernel (right)

2 Exercise Session 2: Function Estimation and Time-series Prediction

2.1 The Support Vector Machine for Regression

In this session, we investigated the influence of different values of *e* and *bound* on the support vector regression results. When the constructed data appears to have a linear pattern, a linear kernel is preferred. When non-linearity is observed, a polynomial kernel or RBF kernel can be suitable. In our case, the polynomial kernel is best suited.

The value of *e* defines a margin of tolerance where no penalty is given to errors. The support vectors are the instances outside the tube, i.e. the samples being penalized, which slack variables are non-zero. As shown in Figure 2.1.1, the larger *e* is, the wider tube is, the larger errors are tolerated, the less number of support vectors, the more number of the elements in the solution vector will be equal to zero, which leads to the sparsity property.

Bound is a regularization parameter that controls the trade-off between model complexity and penalty for nonseparable points. As shown in Figure 2.1.2, when the *bound* is too small, we have underfitting. However, if the *bound* is too large, high penalty goes for nonseparable samples and there will be too many support vectors and overfit the data (*bound* goes to infinity, all samples become support vectors).

In a classical Least Squares fit, we try to minimize the sum of the square of the distances between the approximation and the data, making use of all data points. In SVM, we try to maximize the margin by only using a subset of the data, i.e. support vectors. Compared to a classical Least squares fit, the SVM regression is better at dealing with non-linear problem and controlling model overfitting.

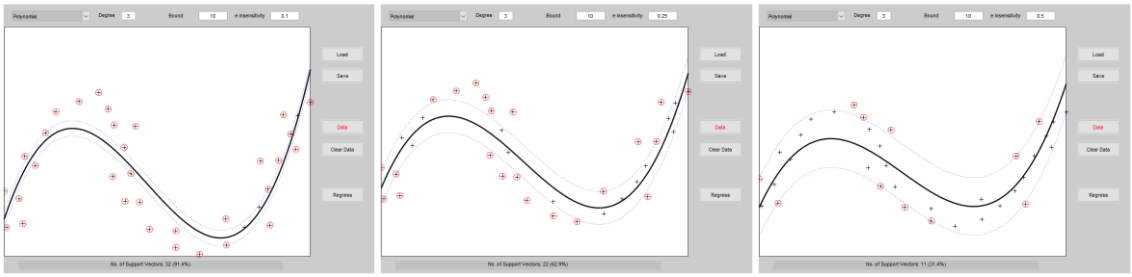


Figure 2.1.1 Results for different values of *e* (0.1,0.25,0.5) with *bound*=10

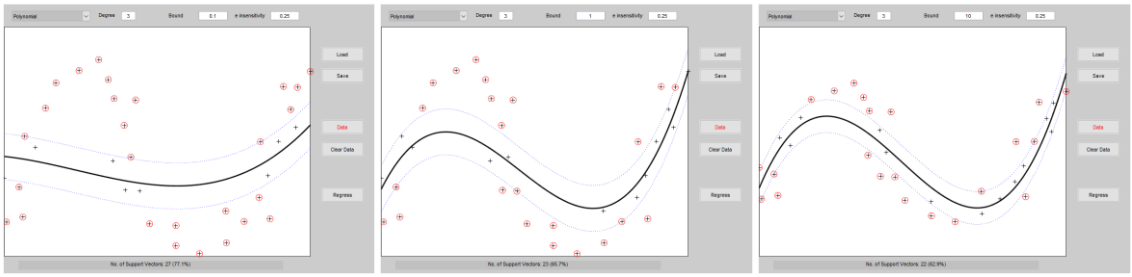


Figure 2.1.2 Results for different values of *bound* (0.1,1,10) with *e*=0.25

2.2 A simple example: Sum of Cosines

In this session, we applied support vector regression for estimating a function constructed from sum of cosines with non-Gaussian noise. In Figure 2.2.1, it gives the results on training set (top) and test set (bottom) for different pairs of hyper-parameters. From left to right, the model unfits, well fit, overfits the data. There exists multiple sets of *gam* and *sig2* that provided comparable good results. For instance, *gam*=10 and *sig2*=0.01 could be a good pair of hyper-parameters.

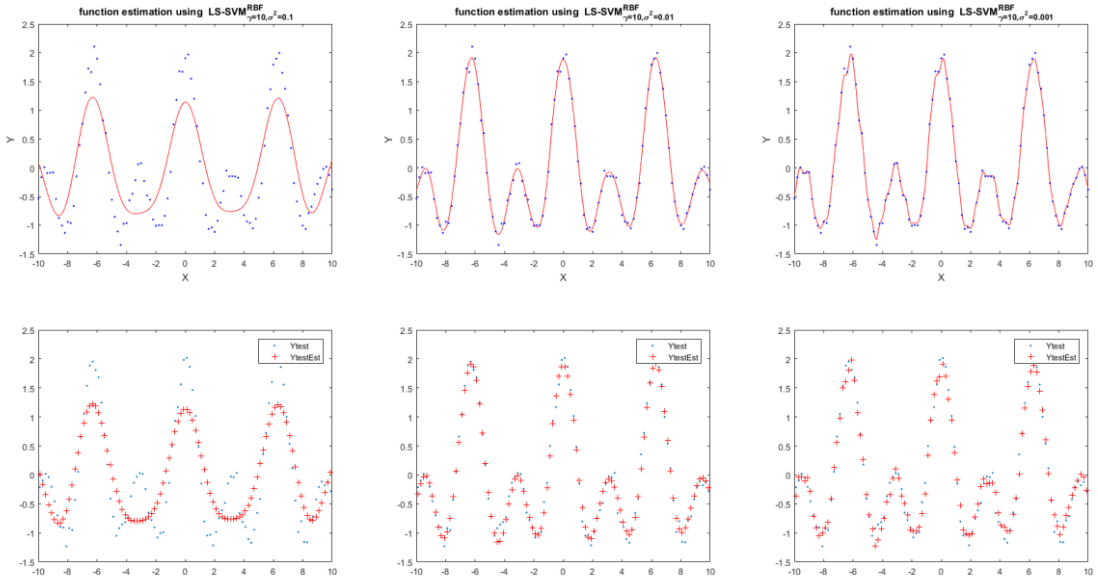


Figure 2.2.1 Results on training set (top) and test set (bottom) for different pairs of hyper-parameters

2.3 Hyper-parameter Tuning

To better tune the hyper-parameters, 10-fold crossvalidation was applied. The optimization process consists of two steps: first, determine good initial starting values by global optimization method (e.g. coupled simulated annealing (CSA), Randomized Directional Search (DS)) and then the obtained suitable initials are given to a second optimization procedure (simplex or gridsearch) to perform a fine-tuning step. The resulting *gamma* values differed a lot, varying from 10^3 to 10^6; the obtained *sig2* values were relatively more stable, varying from 2 to 5. However, for every trail, the resulting costs kept the same, equaling to 0.01 with a bit fluctuation. This means the obtained pairs of gam and sig2 were equally good. The optimization process is actually solving a non-convex problem, which has multiple local minimums.

Optimization algorithms	Trial	Gamma	Sig2	Cost
CSA+gridsearch	1	246131	5.051	0.0107
	2	4131	3.147	0.0110
	3	13109	3.705	0.0114
	4	100354	4.692	0.0103
	5	6061	2.898	0.0105
CSA+simplex	1	4236	3.009	0.0099
	2	2705	3.335	0.0113

	3	2940	3.230	0.0108
	4	2848	3.106	0.0106
	5	857	2.301	0.0110
DS+gridsearch	1	460	2.080	0.0103
	2	113897	5.153	0.0097
	3	5603	3.230	0.0105
	4	2457	2.791	0.0099
	5	29658	4.256	0.0107
DS+simplex	1	3803	3.182	0.0101
	2	2949	3.189	0.0102
	3	1672	2.703	0.0100
	4	6746	3.337	0.0106
	5	3050	3.000	0.0099

Table 2.3.1 Results for different Optimization algorithms

When *simplex* method was applied, the obtained values were more stable and it normally took about 12 iterations to find the solution. When *gridsearch* method was applied, the obtained values were more fluctuating and it took 2 iterations to find the solution. For both algorithms, in order to determine the minimum of a cost function with possibly multiple optima, they evaluate sub-spaces over the parameter space and find the minimum iteratively. These two methods differ in how search space is divided, by *simplex* or *grid*. If l is the length of a vector x, a *simplex* in l-dimensional space is characterized by the l + 1 distinct vectors that are its vertices. In two-space, a *simplex* is a triangle; in three-space, it is a pyramid.[1] *simplex* works for all kernels, *gridsearch* is restricted to 2-dimensional tuning parameter optimization. No significant deviation of results from CSA and DS methods are found.

2.4 Application of the Bayesian Framework

Bayesian framework can be used to optimize the tuning parameters and conduct model comparison. The optimal regularization parameter *gam* and kernel parameter *sig2* can be found by optimizing the cost on the second and the third level of inference, respectively.

For function estimation, by using Bayesian inference, the error bars can computed taking into account model parameter uncertainties. In Figure 2.4.1, it gives the 95% error bars (red dotted and red dashed-dotted line) of the LS-SVM estimate (solid black line) of the sum of cosines function with non-Gaussian noise.

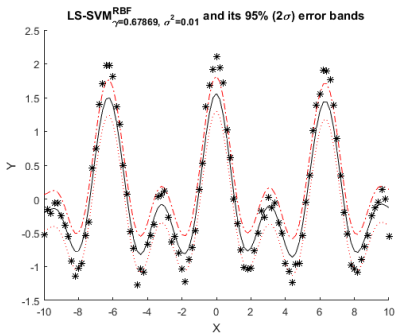


Figure 2.4.1 Result based on Bayesian framework

For classification, by using Bayesian inference, the probability estimates can also be obtained. In Figure 2.4.2, points in blue area have low probability to be classified as positive; points in pink area have high probability to be classified as positive; points in purple area are not sure of their classes with around 0.5 probability to be classified as positive. As the color bar goes from blue to pink, the probability of being classified as positive increases. From left to right of each row, as the value of *sig2* increases, the blue region becomes larger, more points tend to be classified as negative examples. Because *sig2* defines how far the influence of a single training point reaches, with large values meaning far. When *sig2* is too large, the whole training set is selected as support vectors, the resulting model will behave similarly to a linear model. From top to bottom of each column, as the value of *gamma* increases, the blue and pink area become smaller and the purple area becomes larger, meaning more points tend to be uncertain of their classes. Because *gamma* parameter controls the trade-off between model complexity and penalty for nonseparable points. When *gamma* is large, more emphasis is placed on penalizing misclassification, the model tries to classify all points correctly, leading to more points being uncertain of their classes.

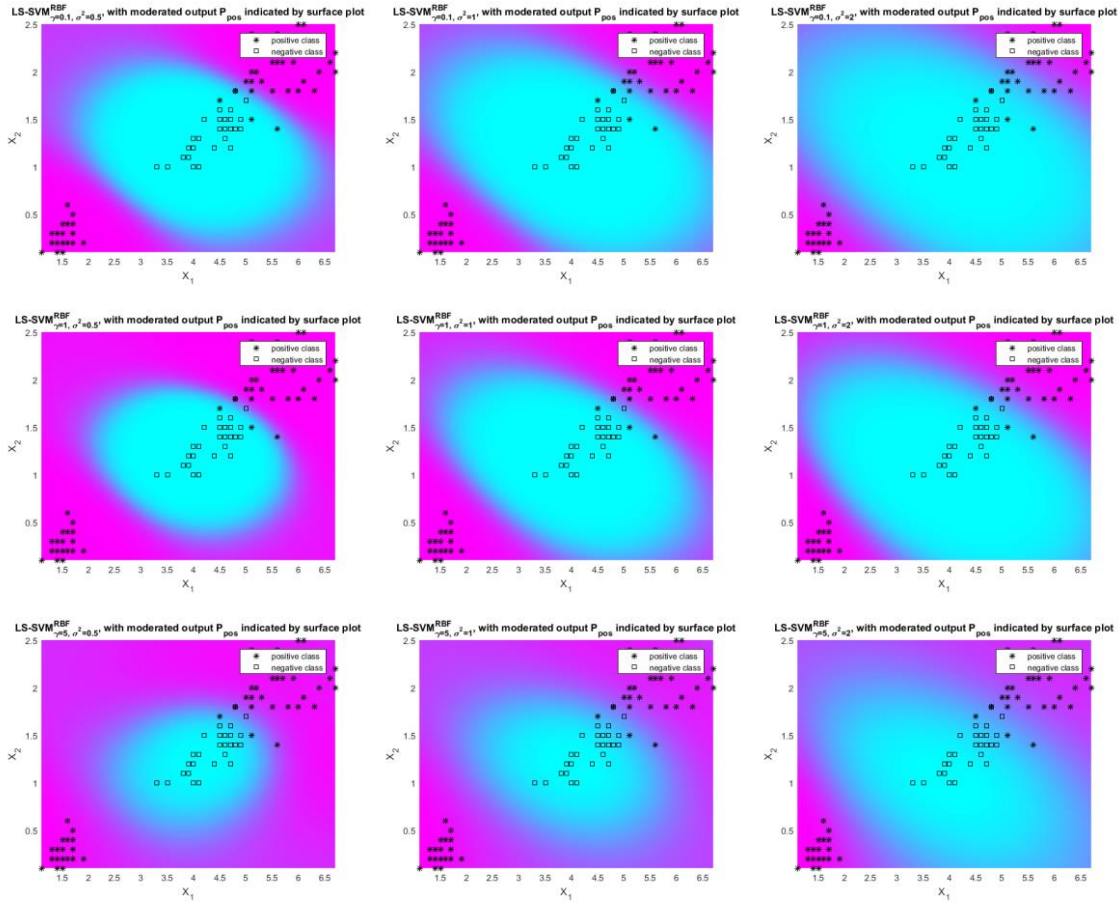


Figure 2.4.2 Result for different combinations of gamma and sig2

Input selection can be done by Automatic Relevance Determination. For the backward selection, the third level of inference is used to infer the most relevant inputs of the problem. As the results shows that the ranking of relevance is 1, 3, 2 and the first column of the input data is selected to be most relevant. This matches how we constructed the data, output data is actually a function of the first column of the input data. Instead of using the Bayesian framework, we can also apply 10-fold cross validation to realize input selection. Each time, a single input feature is considered, we evaluate the MSE on the validation set and then rank all features accordingly.

2.5 Robust Regression

In this part, we worked with the data with outliers. According to Figure 2.5.1, the result of a classical LS-SVM was affected by outliers, for some region, it failed to model the real underlying structure of the data due to making comprise for outliers. In this situation, a robust LS-SVM model is useful. Also, for cost function, MAE is preferred over MSE. When computing MSE, the errors are squared before averaging, so the MSE gives a relatively high weight to large errors. Therefore, MAE is more robust to outliers since it makes use of absolute difference instead of squared difference. As shown in Figure 2.5.1, the robust LS-SVM model provided good result, regardless of the influence of the outliers. Different weighting functions *wFun* resulted in similar results.

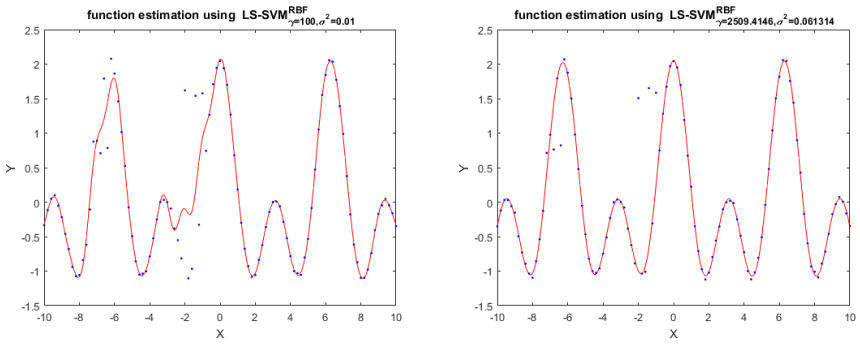


Figure 2.5.1 Results of classical LS-SVM and Robust SVM

2.6 Homework Problem

2.6.1 Introduction: Time-series Prediction

In this session, we applied LS-SVM to make time series prediction for *logmap* dataset. In Figure 2.6.1.1, it provides the prediction results on the test set (*order*=10, *gam*=10, *sig2*=10). The performance of the the prediction can be assessed by MSE on the test set, and for this setting, MSE equals 0.1303.

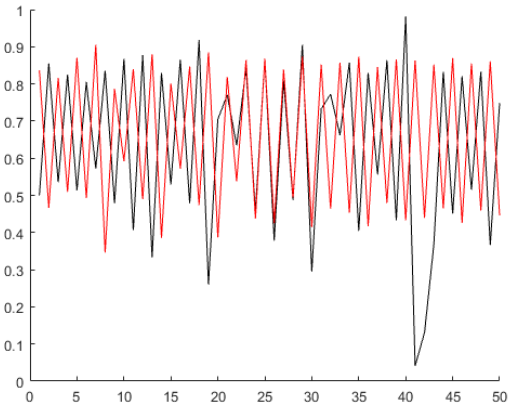


Figure 2.6.1.1 Prediction results on the test set (gam=10, sig2=10)

To improve the performance, we can apply 10-fold cross validation to optimize *order* parameter of the linear auto-regressive model. In the range of *order* (1 to 50), for each value of *order*, we applied *tunelssvm* to find the optimal combination of *gamma* and *sig2*, then computed the MSE of the model using the optimal hyperparameters. In Figure 2.6.1.2, it gives the change of MSE on the test set with different values of *order*. Though some fluctuations exist, in general, with the increase of *order*, the MSE decreases rapidly, after a certain optimal range, with the increase of *order*, the MSE increases a bit and then keeps at a relatively low value(around 0.05). When *order* equals 23, the resulting MSE reaches its smallest value (0.0387). So 23 is the best choice for *order* parameter.

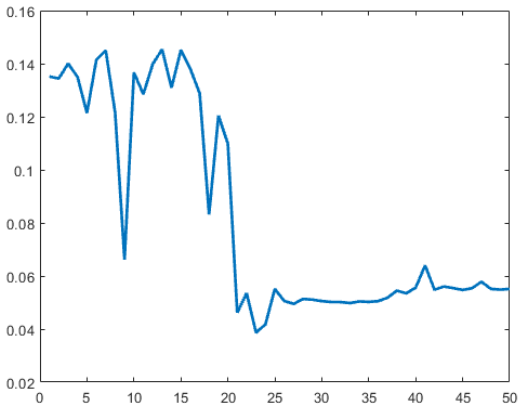


Figure 2.6.1.2 MSEs of different orders under optimal combination of gam and sig2

2.6.2 Application: Santa Fe laser dataset

In this session, we applied LS-SVM to make time series prediction for the *Santa Fe laser* dataset. In the left graph of Figure 2.6.2.1, it gives the prediction result on test set for *oder=50* and *gam* and *sig2* was chosen by *tunelssvm*. The *Santa Fe laser* dataset contains 1000 points for training set and 200 points for test set, so it would be a reasonable choice that *order=50*. Also, as shown in Figure 2.6.2.1, for the first 50 points, the model almost fitted perfectly and afterwards the model could fit the basic patterns with some deviations.

Like in the previous session, to improve the performance, we can apply 10-fold cross validation to optimize *order* parameter. However, there exists some problem using this method. Because it uses the training set to find the optimal *hyper-meters,* and then uses the test set to tune the *model order.* It is important that the test set should only be used for assessing model performance and should not be used in model building process. So we adapted the method by dividing the training set into two subsets, one for tuning *hyper-meters* by 10-fold cross validation for each reasonable value of *order,* another for tuning *model order* based on MSE*,* and the test set is only be used to check performance. In Figure 2.6.2.2, it gives the change of MSE on test set with different values of *order*. The MSE kept fluctuating around 400 in the interval [20,50]. There existed several small MSE values from 50 to 60. After that, it appeared to be an increasing trend and the curve became unstable with large fluctuations. It can be seen that 56 would be a good choice for *order* with the smallest MSE value of test set (MSE=249.35). In the right graph of Figure 2.6.2.1, when *order* equaled 56, the model also fitted perfectly for the first 50 points and afterwards demonstrated better performance than the previous model.

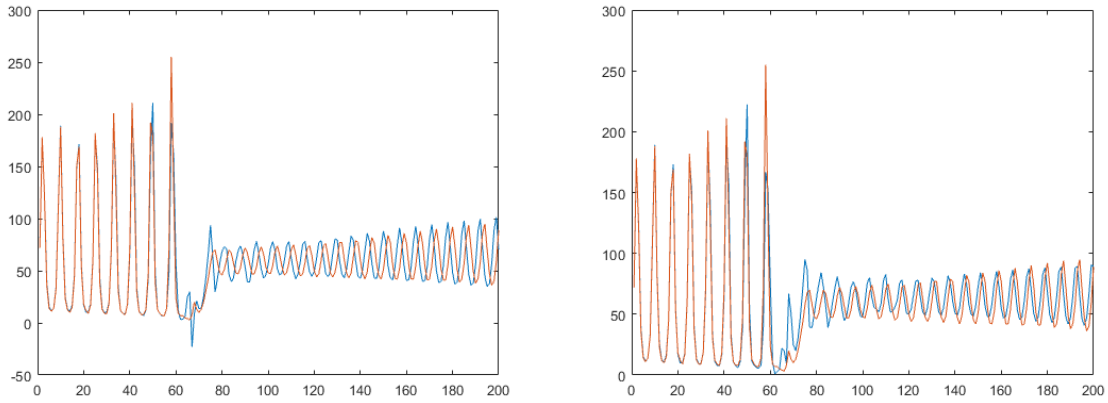


Figure 2.6.2.1 Prediction results on the test set (left order=50, right order=56)

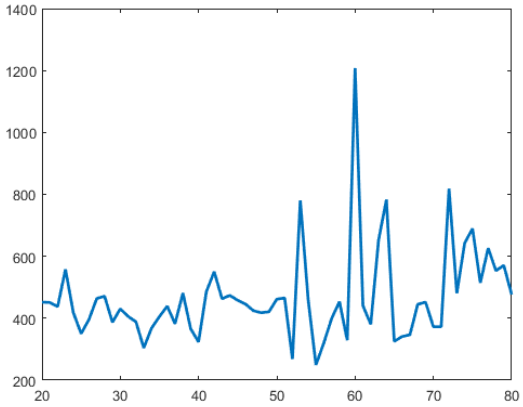


Figure 2.6.2.2 MSE on test set vs Order

3 Exercise Session: Unsupervised Learning

3.1 Kernel Principal Component Analysis

This example shows the idea of denoising in the input space by means of kernel PCA. The constructed noisy data contains 400 data points in 2 dimensions. According to Figure 3.1.1, with the increasing of the number of principal components (PCs), the model becomes better at capturing the intrinsic dimensionality and denoising the noisy input data. However, the chosen number of PCs should not be too large. When 15 PCs is used, the model overfits and is highly influenced by noise.

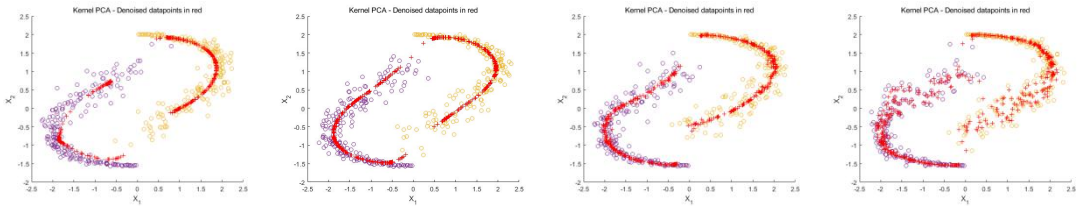


Figure 3.1.1 Denoised data (+) obtained by reconstructing the data-points (o) using

3, 6, 10, 15 kernel principal components with the RBF kernel

Linear PCA is applied to find linear PCs to represent the data in lower dimension. So the number of PCs can only be equal to or smaller than the dimension of the input space (in this case, 2). For Kernel PCA, it uses the Kernel trick to map the input space to a high dimensional feature space. This nonlinear mapping makes the method better at dealing with non-linearity of the data. It also allows the number of PCs be larger than the dimension of the input space. Technically we can choose the number of PCs up to the number of training data (in this case, 400). However, we try to select as few PCs as possible and minimize the reconstruction error. According to Figure3.1.2, the kernel PCA method is able to capture the intrinsic dimensionality and denoise the noisy input data while linear PCA cannot.

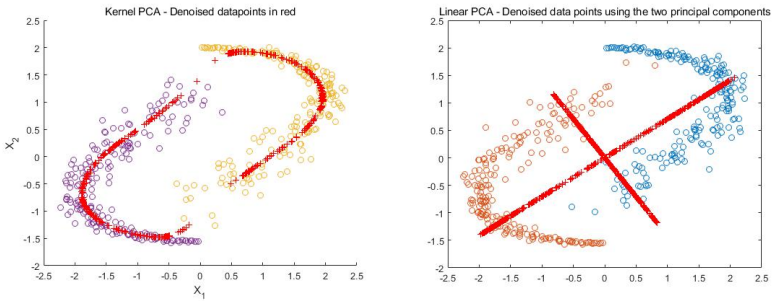


Figure 3.1.2 Denoised data (+) obtained by reconstructing the data-points (o) using

1)6 kernel principal components with the RBF kernel; 2) linear PCA with 2 principal components

To decide the number of PCs, we can use a scree plot as shown in Figure 3.1.3. There is some sort of an "elbow" around 6 components, and afterwards the line becomes very monotonic. So 6 components seems like a reasonable number on the basis of eigenvalues only. Besides visually checking, to better select the number of components and tune the kernel hyper-parameters also, we can apply cross validation using the reconstruction error as the

evaluation criterion. For kernel PCA, there exists a problem that the reconstruction errors are not comparable between different kernels. Because different kernels correspond to different target spaces, so reconstruction error is measured in different space. Alam, M. A., & Fukumizu, K. (2014) proposed to find a 'pre-image' in the original space which is as close as possible to the reconstruction point, and then measure the distance between the test point and this pre-image as reconstruction error.[2]

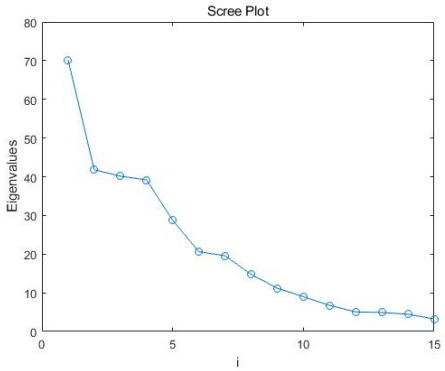


Figure 3.1.3 Eigenvalues v.s. Number of principal components

3.2 Handwritten Digit Denoising

In this session, we conduct some experiments on the handwritten digit data set using kPCA and linear PCA for reconstruction and denoising. In Figure 3.2.1, the first row is the original image of each digit in the test set, the second row is the image with additive noise, following 9 rows are the reconstruction for Kernel PCA (left) and linear PCA (right) using n = 1, 2, 4, 8, 16, 32, 64, 128, 190 components. For identical numbers of components in both algorithms, kernel PCA performs much better than linear PCA.

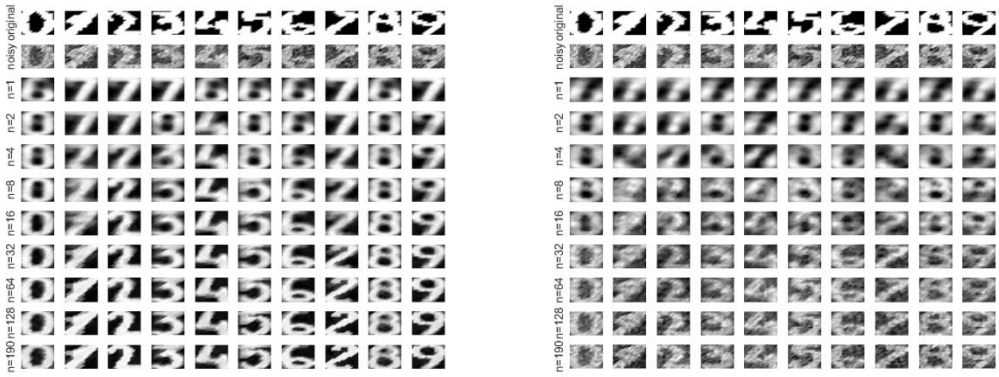


Figure 3.2.1 Denoising of handwritten digit data set

3.3 Spectral Clustering

The spectral clustering is a kind of unsupervised learning, while classification is supervised. It makes use of the kernel function as a similarity measure. In Figure 3.3.1, it gives the clustering results for two 3D rings with different values of *sig2*. When the value of *sig2* is too small, the distance which defines distinct classes is set to be too small, so the algorithm would regard every point as different, then it cannot correctly make classification as shown in the first graph in Figure 3.3.1. When the value of *sig2* is too large, the similarity range is too large, then the distance between points on inner part of horizontal ring and points on inner part of vertical ring is within the similarity range, that is the reason they are falsely classified in the same class as shown in the fourth graph in Figure 3.3.1. It is important to choose a good value of *sig2*, so that the distance defining the same class is able to classify only the points close to each other and on the same ring into the same class.

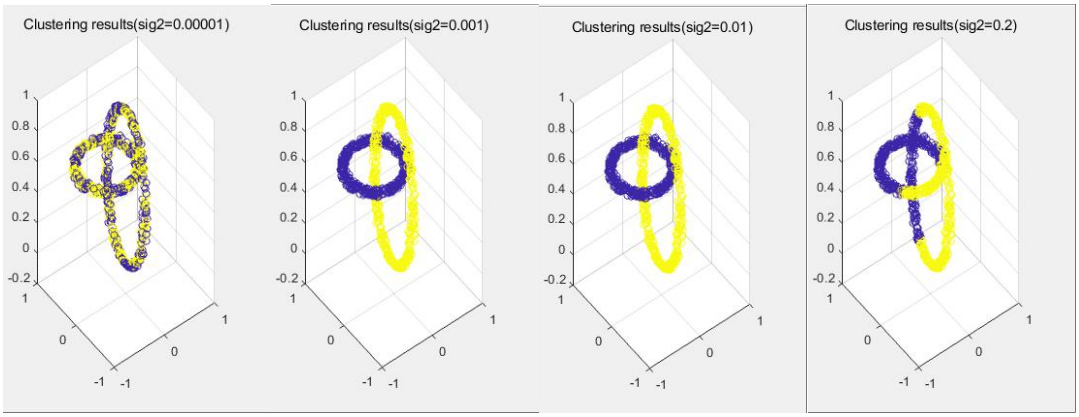


Figure 3.3.1 Clustering results with *sig2*=0.00001, 0.001, 0.01, 0.2

3.4 Fixed-size LS-SVM

For fixed-size LS-SVM, the number of support vectors is pre-defined. In the working set of support vectors a point X* is randomly selected and replaced by a randomly selected point Xt* from the training data if Xt* improves the entropy, otherwise X* stays in the working set. This procedure is conducted iteratively. The algorithm converges to an optimized subset when the change in entropy value is small enough or the number of iterations is exceeded.

To investigate the influence of the chosen *sig2*, a toy example containing 100 data points in a 2 dimension space was constructed and we intended to find 10 support vectors. According to Figure 3.4.1, when *sig2* is too small, no much update of the working set is observed, the support vectors are just selected randomly and concentrated in the central area of the dataset. When *sig2* is too large, the defined similarity measure is too large, the selected support vectors are on outer ring of the data. When *sig2* equals 1, we obtained good result: uniform distributed support vectors over the input space, which can take account the full range of the data points.

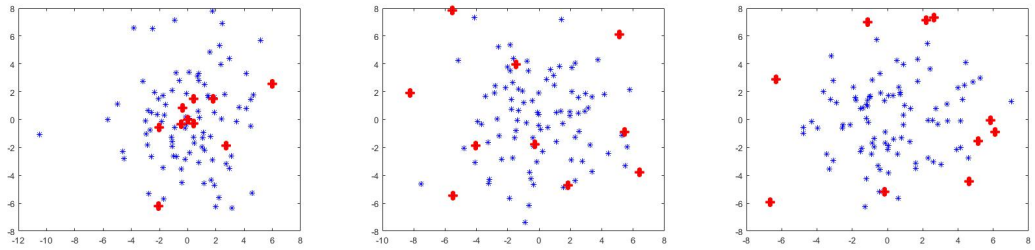


Figure 3.4.1 Selected support vectors in original space for sig2=0.001, 1, 100

By applying l_0-norm minimization, sparser model can be obtained compared to a normal FS-LSSVM. Figure 3.4.2 shows the error estimate, variations in the number of SVs and computation time for two approaches based on the first 700 data points of shuttle data set. FS-LSSVM and SV_L0_norm resulted in the same prediction errors. Some variations in the number of SVs is observed for SV_L0_norm method: minimal number of SVs equals 5, maximal number of SVs equals 8 and medium is 6. While the number of SVs for FS-LSSVM method stays at 14. Compared to FS-LSSVM, the computation time can vary more for SV_L0_norm method. However, the difference in time between FS-LSSVM and SV_L0_norm method are insignificant. It can be concluded that SV_L0_norm method always results in a sparser model than FS-LSSVM without significant tradeoff in test error and time and thus making it preferable.

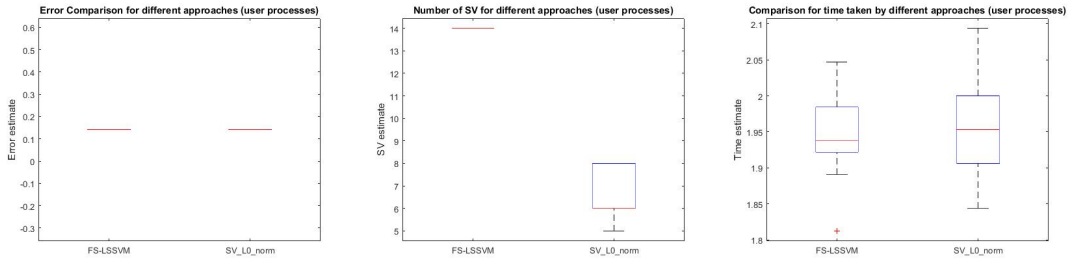


Figure 3.4.2 Comparison of FS-LSSVM and SV_L0_norm (linear kernel)

3.5 Homework Problems

3.5.1 Handwritten Digit Denoising

In this session, we applied kernel PCA on handwritten digits reconstruction and denoising. A rule of thumb for the value of hyper-parameter *sig2* is the mean of the variance of each dimension multiples the dimension of the training data. In Figure 3.5.1.1, it shows the change of the result with the change of the value of *sig2*. When suggested estimate of *sig2* is applied, the performance of reconstruction and denoising becomes better as the number of components increases up to a certain range. By checking visually, 16 could be a choice for the number of component. When the *sig2* value is 10 times of the suggested value, the reconstructed image becomes more noisy. When the *sig2* is 100 times of the suggested value, the results are very noisy, for many digits, the method is not able to reconstruct.

When the *sig2* value is smaller than the suggested value, as shown in the first and second image (sigmafactor=0.01, 0.1), the denoising effect is more obvious, the reconstructed

image becomes sharper. However, for some digits, the method provides wrong reconstructed image. For example, digit 7 can be incorrectly reconstructed as 2. Visually, there is no significant improvement of the performance as the number of components increases.

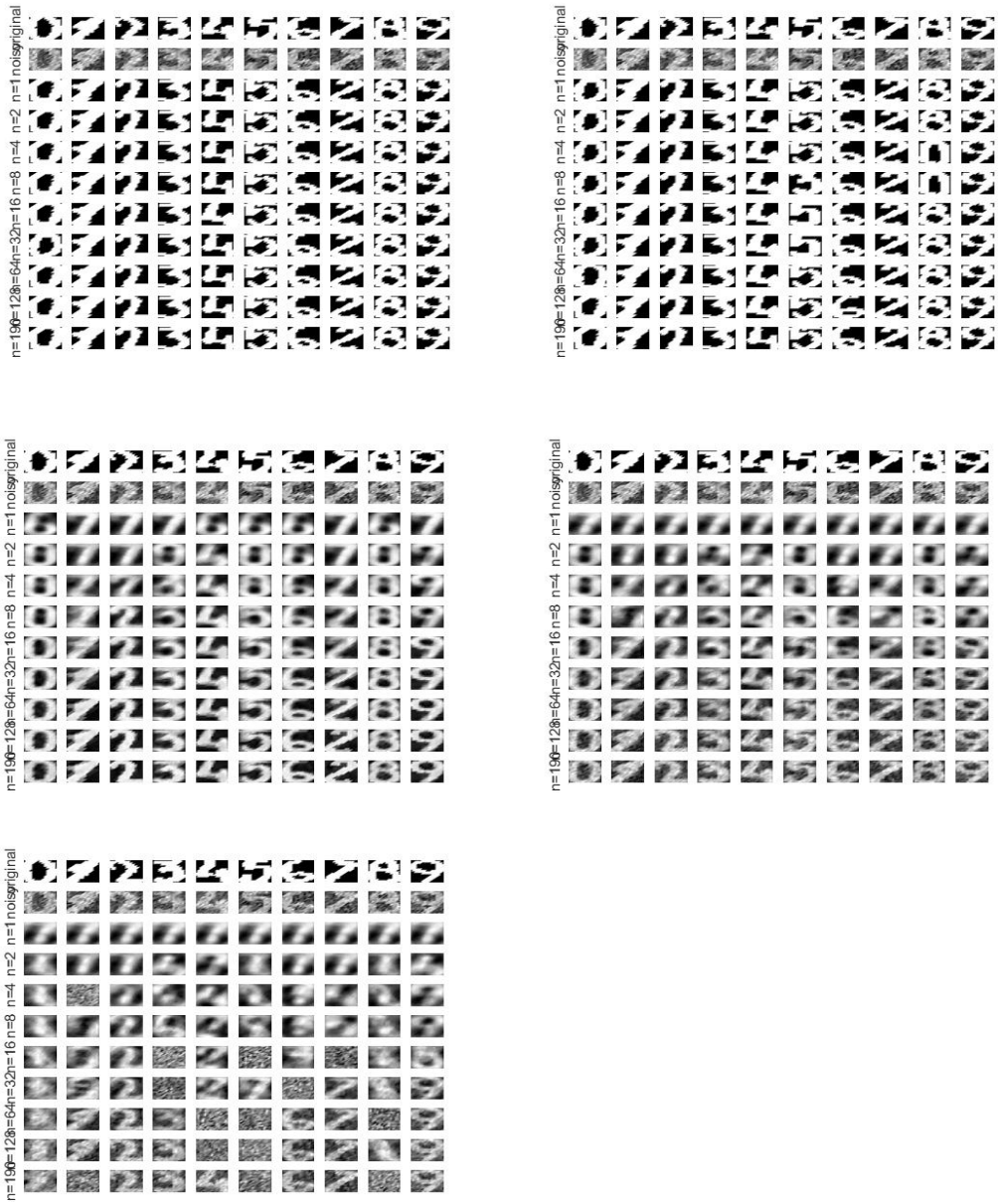


Figure 3.5.1.1 Denoising of handwritten digits for kPCA (*noisefactor*=0.3)

with *sigmafactor*=0.01, 0.1, 1, 10, 100

In Figure 3.5.1.2, it provides the reconstruction and denoising results on Xtest2 for Kernel PCA (left) and linear PCA (right) with *noisefactor*=1. The linear PCA can be considered of being not able to reconstruct and denoise the original digits of Xtest2. For identical numbers of components in both algorithms, kernel PCA performs much better than linear PCA.

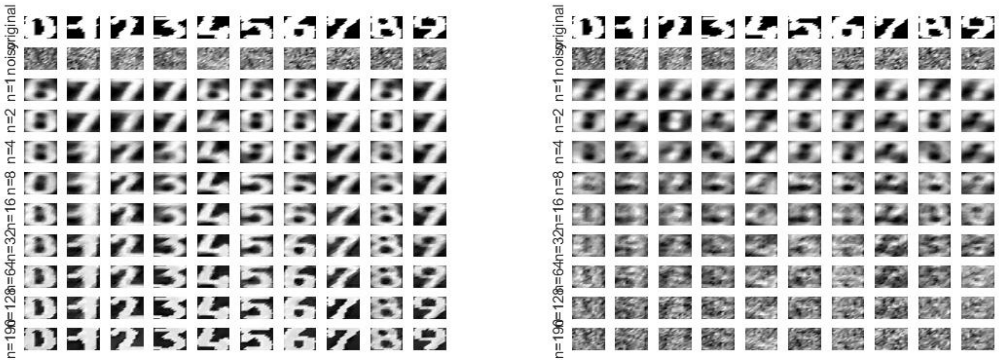


Figure 3.5.1.2 Denoising of handwritten digits on Xtest2

for kPCA (left) and linear PCA(right) with *noisefactor*=1

Next, we tried to select the optimal value of *sig2* by minimizing the reconstruction error on the validation set. A reasonable range of *sigmafactor*[0.5, 1.5] of suggested value was tested. As shown in Figure 3.5.1.3, the reconstruction error firstly decreases as the *sig2* increases; when the *sig2* equals 61.56, the reconstruction error reaches minimal value 0.1538; afterwards the reconstruction error increases as the *sig2* increases. In Figure 3.5.1.4, it provides the results when original and optimal *sig2* were used. By visually checking, not much difference is observed.

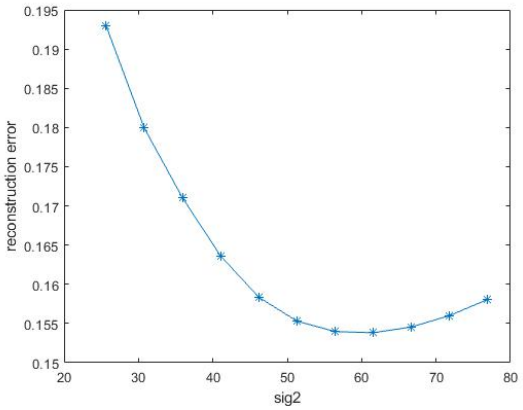


Figure 3.5.1.3 Reconstruction error on validation set VS *sig2* value

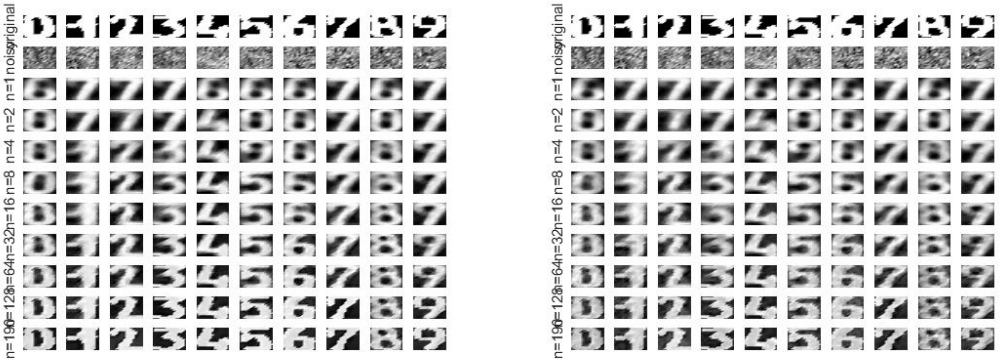


Figure 3.5.1.4 Denoising of handwritten digits on Xtest2

Using kPCA for sigmafactor = 0.7(left) and 1.2(right) (*noisefactor*=1)

3.5.2 Shuttle Dataset

Figure 3.5.2.1 shows the error estimate, variations in the number of SVs and computation time of a classification problem for two methods based on the shuttle data. FS-LSSVM and SV_L0_norm resulted in the same prediction errors. Some variations in the number of SVs is observed for SV_L0_norm method: minimal number of SVs equals 6, maximal number of SVs equals 10 and mean is 7. While the number of SVs for FS-LSSVM method stays at 43. The variations of computation time for both methods are similar and there is no insignificant difference between medium. It can be concluded that SV_L0_norm method always results in a sparser model than FS-LSSVM without significant tradeoff in test error and time and thus making it preferable.

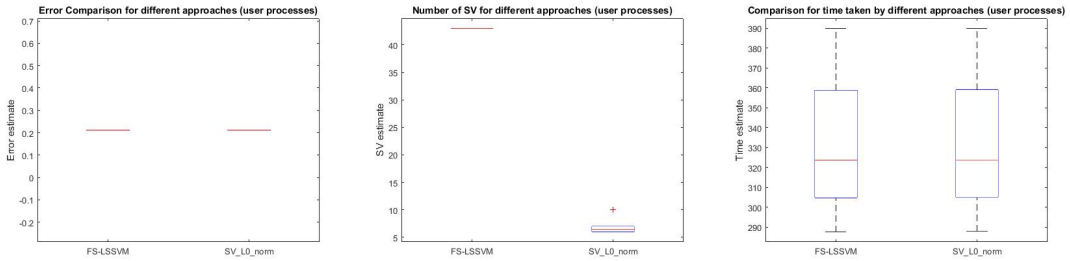


Figure 3.5.2.1 Comparison of FS-LSSVM and SV_L0_norm (linear kernel)

3.5.3 California Dataset

Figure 3.5.3.1 shows the error estimate, variations in the number of SVs and computation time of a function regression problem for two methods based on the california data. Compared to FS-LSSVM, SV_L0_norm resulted in larger value of prediction error. Some variations in prediction error is observed for SV_L0_norm method: minimal prediction error equals 0.378, maximal prediction error equals 0.508 and medium is 0.482. While the prediction error for FS-LSSVM method stays at 0.368. Some variations in the number of SVs is observed for SV_L0_norm method: minimal number of SVs equals 4, maximal number of SVs equals 44 and medium is 5. While the number of SVs for FS-LSSVM method stays at 44. The variations of computation time for both methods are similar and there is no insignificant difference between medium. It can be concluded that SV_L0_norm method provides a sparser model than FS-LSSVM without significant tradeoff time; yet costs higher prediction error.

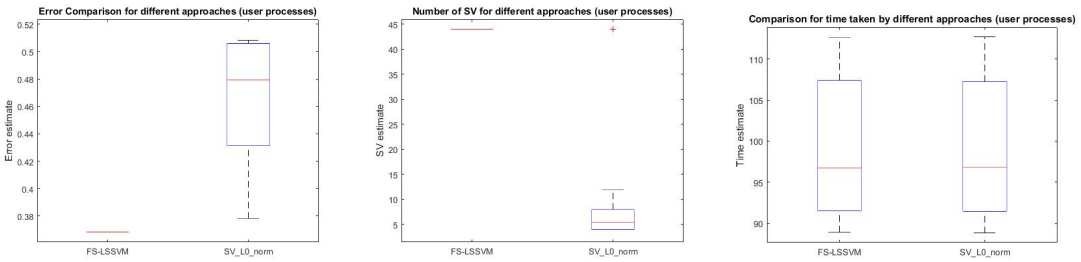


Figure 3.5.3.1 Comparison of FS-LSSVM and SV_L0_norm (linear kernel)