KU Leuven

Support Vector Machines: Final Assingment

Akshat Dwivedi (Student Number:)

Instructor: Prof. Johann Suykens

September 18, 2016

Contents

The exercises in this report are for the course Support Vector Machines and are programmed in MATLAB and the report is typeset using LaTeX.

1 Exercise Session 1: Classification

1.1 Two Gaussians

A variable X is created as a vector of two sub-variables X_1 and X_2, where $X_1 = 1 + N(0,1)$ and X_2 = -1 + $N(0,1)$ and correspondingly, Y = 1 if X_1 and -1 otherwise. This results in one predictor (with two classes) and one response (class label). The figure is plotted in figure 1.

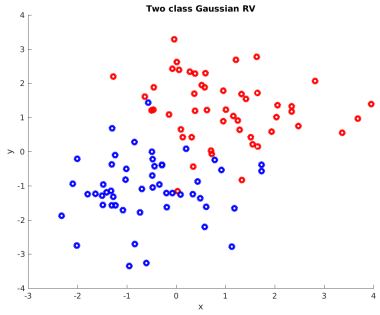


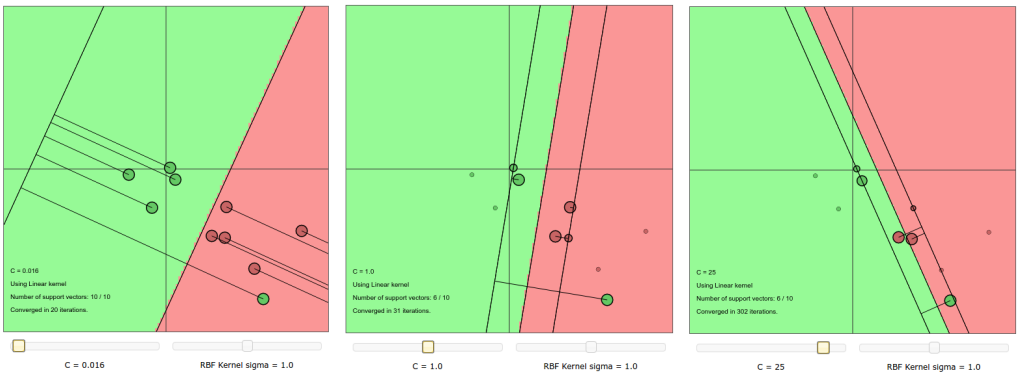
Figure 1: Classification of a two-class variable.

The two classes here cannot be perfectly separated by a linear hyperplane without misclassifying some of the points in the training set that are overlapping with points from the other class. However, a nonlinear boundary can find a separating hyperplane with lesser misclassified observations. Another approach maybe to use the kernel trick where the inputs are mapped to a higher dimension (a mapping from input space to feature space) where they can be linearly separated by a hyperplane.

1.2 The Support Vector Machine

The demo found at `http://cs.stanford.edu/people/karpathy/svmjs/demo/` is explored to visually assess the working of an SVM. Each time a data point is added, the linear decision boundary changes to accommodate the addition of that point. Thus, depending on where the point is added (irrespective of class), the change in the decision boundary can be rather drastic. Adding a point on the opposite side of the decision boundary results again in a drastic shift in the decision boundary so as to accommodate this new point and keep the classification cost minimal by finding a new maximum-margin classifier.

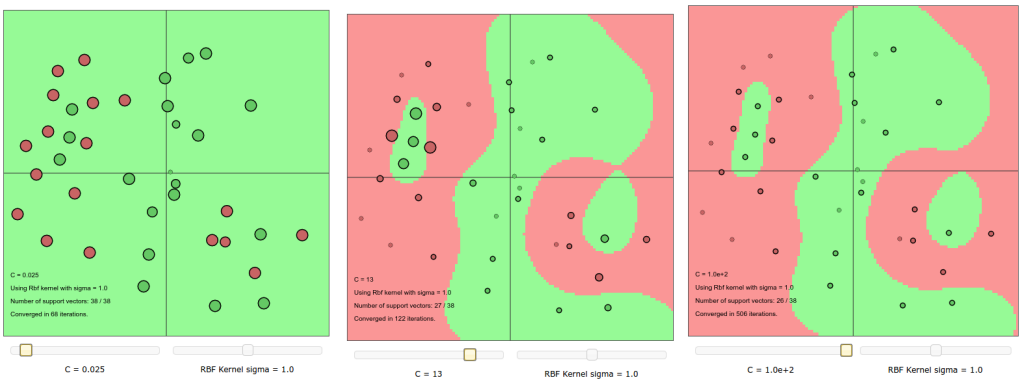
The parameter c is a regularization parameter that controls the number of points that are used as support vectors. A larger value of c uses less support vectors (indicating



(a) c = 0.025 (b) c = 1 (c) c = 25

Figure 2: Linear kernel: exploring the role of regularization parameter c.

more regularization) and a smaller value of c uses a larger fraction of the total points as support vectors (i.e., points closer to the decision boundary). Furthermore, increasing the value of c requires more iterations for the underlying algorithm to converge to the optimal solution. A smaller amount of regularization converges faster.

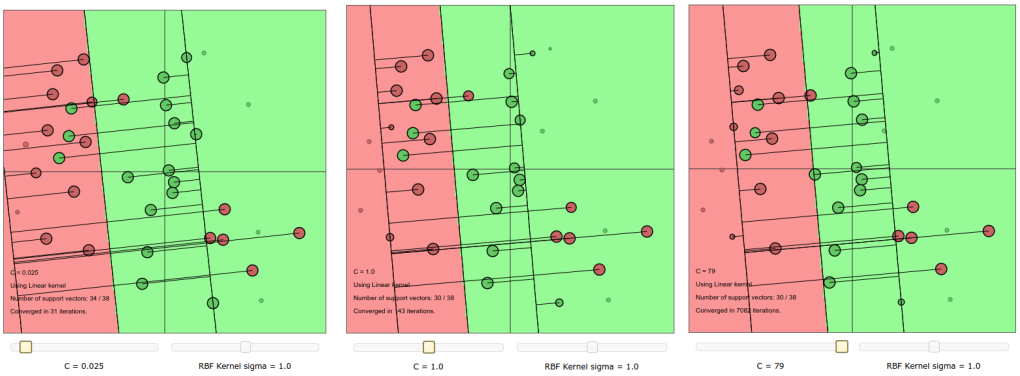


(a) c = 0.025, $\sigma^2 = 1$ (b) c = 13, $\sigma^2 = 1$ (c) c = 100, $\sigma^2 = 1$

Figure 3: Non-separable classes. RBF kernel with σ^2 fixed and c varying.

Switching back to the RBF kernel from the linear kernel, we see that the classification error decreases drastically and the network is highly adaptive to the addition of new data points irrespective of where the points are added. Furthermore, a much larger fraction of data points are used as support vectors compared to the linear case which is the reason for the low misclassification rate. If the regularization hyperparameter is too small, the whole region is assigned one class label (the dominant class) however increasing this value results in separate regions for the two classes. If two points- one from each class are separated from each other, a small region forms around the points. the important point is that the decision boundary is nonlinear, and hence, is capable of

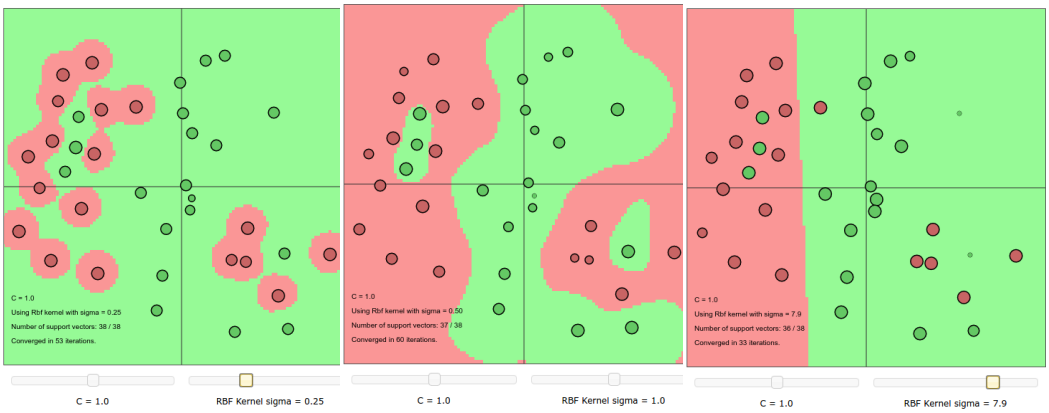
learning from general decision surfaces. The size of the point indicates it's importance in the decision boundary- points which are close to the decision boundary are larger and vice versa. The hyperparameter σ^2 for the RBF kernel controls the smoothness of the decision boundary. A large σ^2 value results in a near linear boundary where as a smaller σ^2 value leads to the decision boundary being more nonlinear (rough).



(a) c = 0.025 (b) c = 1 (c) c = 79

Figure 4: Non-separable classes and linear kernel.

We create a dataset with overlapping regions (thus the two classes are not linearly separable). This is shown in figure 5(a). If a linear kernel is used, about half the points are used as support vectors, as compared to the RBF kernel where nearly all the points are used as support vectors. However, the misclassification in case of linear kernel is much higher where as it is nearly zero in case of the RBF kernel. In case of RBF kernel, a small amount of σ value keeps the size of the Gaussian kernel at the support vectors small.



(a) c = 1, $\sigma^2 = 0.25$ (b) c = 1, $\sigma^2 = 1$ (c) c = 1, $\sigma^2 = 8$

Figure 5: Non-separable classes with RBF kernel and fixed c.

The importance of the support vector changes as each new point is added to the dataset

or if the hyperparameter values are tweaked. A data point becomes a support vector if it happens to be close to the decision boundary. If a new point is added to class A region that is from class B then this new point will become a support vector due to the corresponding slack variable becoming active in the optimization process.

1.3 Using LS-SVMlab

1.3.1 Demo

The demo shows how to use the toolbox for classification. Input matrix is generated as a 30 x 2 matrix $\mathbf{M}$ with $m_{ij} \sim N(0,1)$ entries and the final input matrix $\mathbf{X}$ is created with each entry $x_{ij} = 2m_{ij} - 1$. The response variable $\mathbf{Y}$ is created as $\mathbf{Y} = \text{sign}\{\sin(\mathbf{X}_1 + \mathbf{X}_2)\}$. The regularization ($\gamma$) parameter is chosen to be 10 which selects the trade-off between fitting-error minimization and smoothness. In case of the RBF kernel, σ^2 is the bandwidth (hyper-)parameter and is taken to be 0.2. The model is trained on the training set and can be evaluated on new data points by using the command `simlssvm` and the result for 2 features can be plotted and examined visually as is done in figure 6. `prelssvm` is used to preprocess the input data before training the model. Continuous predictors are scaled to have zero mean and unit variance; binary variables are rescaled to {-1,+1} and categorical predictors are left untouched.

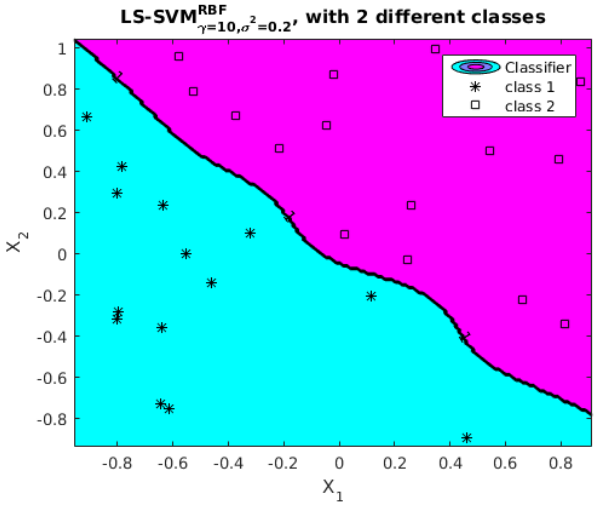
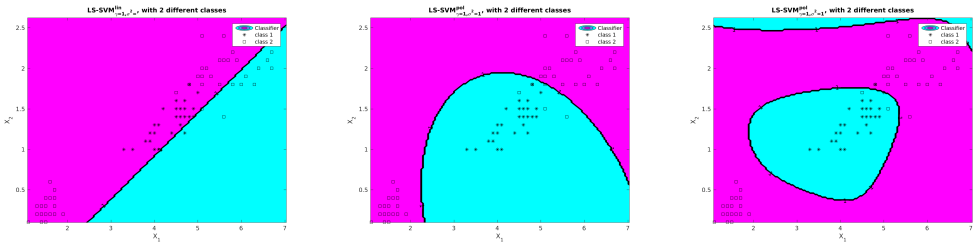


Figure 6: LS-SVM demo: two class classification of a system with 2 features. The decision boundary using an RBF kernel is shown as a solid black line.

1.3.2 Iris dataset

In this section, the classic Iris dataset is used for classification and different kernels (linear, polynomial, RBF) are explored. The dataset is divided into 100 points in the training set and 20 points in the test set. First, a linear LS-SVM is fit to the training set.

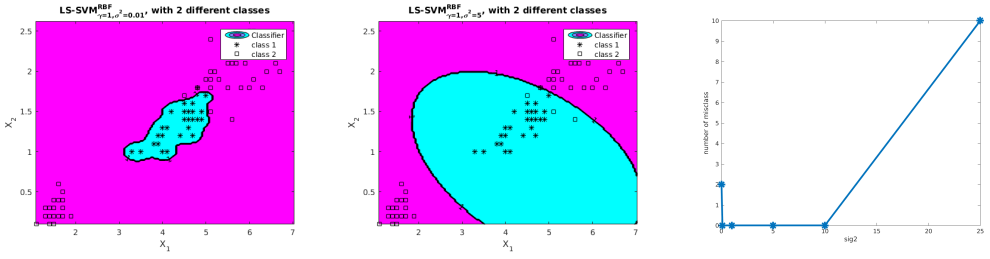
The decision boundary is plotted in figure 7(a). Number of misclassified observations are 11/20 ($\approx 55\%$). Next a polynomial kernel is tried with degree 2 (quadratic kernel) and 5. Note: a polynomial kernel with degree 1 is simply a linear kernel. These are plotted in figure 7. It is observed that for degree ≥ 3 the error rate drops to zero with the error rate at degree $= 2$ equalling 5%. The polynomial kernel is formulated as $K(x, y) = (x^T y + c)^d$ where d denotes the degree of the polynomial. In the middle figure, we see the decision boundary is quadratic which is to be expected from a quadratic polynomial kernel. In case of degree 5, the kernel seems to be fitting erratically towards the edges of the feature space where there are no data points.



(a) $n_{miss} = 11$, err rate: 55% (b) $n_{miss} = 1$, err rate: 5% (c) $n_{miss} = 0$, err rate: 0%

Figure 7: Iris classification: Linear kernel (a), polynomial kernel of degree 2 (b), and polynomial kernel of degree 5 (c). The misclassification percentages are reported on the test set. The error rate drops to zero for degree ≥ 3.

For the RBF kernel case, we keep the regularization parameter (γ) fixed at 1 and vary the values of the RBF kernel parameter ($\sigma^2 \in \{0.01, 0.1, 1, 5, 10, 25\}$). The error rate for the first value (0.01) is 10% (2 misclassified observations) after which the error rate drops to zero except for the last σ^2 value (25) at which point the uncertainty is too large and all points are assigned to one out of the two classes (which results in 50% error rate on the test set). This is plotted in figure 8.

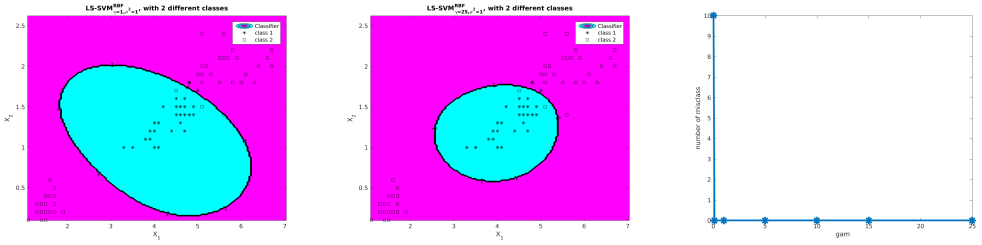


(a) $\sigma^2 = 0.01$ (b) $\sigma^2 = 5$ (c) $n_{misclass}$ vs σ^2

Figure 8: Iris classification: RBF kernel with different values for the hyperparameter σ^2 set to 0.01 (a), 5 (b) and the plot of $n_{misclass}$ as a function of σ^2 on the test set.

From this figure, we see that for small σ^2, the class region is quite small which leads to

non-zero error rate on the test set. However, as σ^2 increases, the uncertainty around the separating margin increases leading to larger regions for each of the classes. Thus, this leads to a reduction in the error rate of the classifier. Figure 8(c) shows the number of misclassified cases on the test set as a function of the hyperparameter σ^2. Next, we do the same with the regularization parameter γ ($\in \{0.01, 0.1, 1, 5, 10, 15, 25\}$) while holding $\sigma^2 = 1$ fixed and the results are shown in figure 9.



(a) $\gamma = 5$ (b) $\gamma = 25$ (c) $n_{misclass}$ vs γ

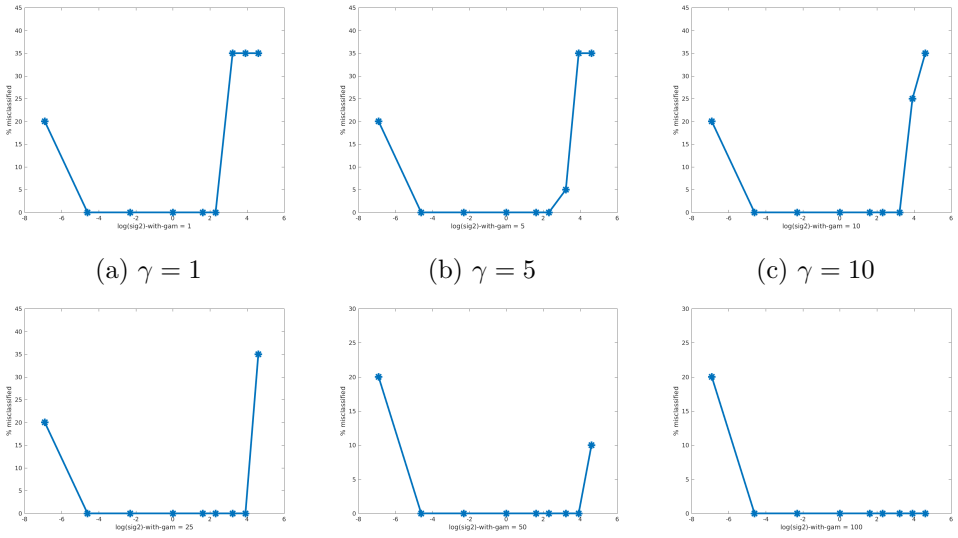
Figure 9: Iris classification: RBF kernel with $\sigma^2 = 1$ (fixed) and different values for the hyperparameter γ set to 1 (a), 25 (b) and the plot of $n_{misclass}$ as a function of γ on the test set.

For the γ parameters and this dataset assuming $\sigma^2 = 1$, a good range seems to be between 0.1 and 10 since beyond that point the region for class 2 kept getting smaller. At smaller γ values, the region is quite large. However in practice this will be different for different datasets. In the next subsection, automated tuning algorithms for hyperparameter optimization are explored.

1.3.3 Choice of hyperparameters

We continue with the Iris dataset in this section. First, the training set from the previous section ($n = 100$) is further divided into a training set ($n = 80$) and a validation set ($n = 20$). The model is trained on the training and set and then evaluated on the validation set. The results are shown below in figure 10. We cannot use this validation set, however, to measure the final model since the error rate obtained will be biased (underestimated) since the model has already "seen" the data, so to speak. In the figures, we see that the error rate shows a large amount of variation for different (γ, σ^2) values and determining an optimal combination that will generalize well to future data remains a mystery.

One problem with this approach, i.e, splitting the training data into a training and a validation set is that although the split was random, the validation set may (by chance) display characteristics either similar or dissimilar to the training set. An alternative approach is performing k-fold CV (say $k = 10$) where the training set is split into k (disjoint) subsets and at each step of training, $k - 1$-subsets are used to build the model



(a) $\gamma = 1$ (b) $\gamma = 5$ (c) $\gamma = 10$

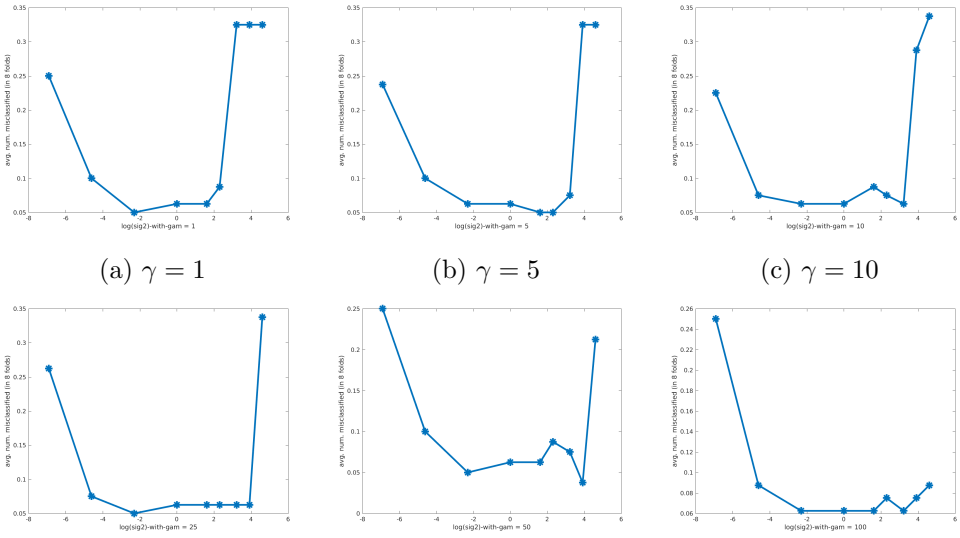
(d) $\gamma = 25$ (e) $\gamma = 50$ (f) $\gamma = 100$

Figure 10: Iris classification (fixed validation set): RBF kernel with different $\log(\sigma^2)$ values at each level of $\gamma \in \{1, 5, 10, 25, 50, 100\}$ values. The error rates (% observations in the validation set that are misclassified) are plotted for the different (γ, σ^2) values.)

and the model is then evaluated on the k^{th} subset that was kept aside as the validation set. We can then select the value of (γ, σ^2) that lead to the smallest average error rate over the k-folds. These figures are plotted in figure 11.

The y-axis does not display the error rate (% misclassified) but the mean (across the folds) of the number of misclassified observations with each fold used as the validation set. It can be seen that in all the cases, a good value of σ^2 is 0.01 and all the γ values tried seems to result in a good fit (although if a single value had to be selected, $\gamma = 100$ seems to be a good choice). Next, we repeat the procedure but instead of k-fold CV, we perform leave-one-out CV (LOOCV) and the results are plotted in figure 12. It is observed that the misclassification rate is nearly the same as the k-fold CV procedure, however, for larger datasets, k-fold cv is less computationally intensive as compared to LOOCV and is to be preferred. For smaller datasets (such as this one), it is more sensible to use LOOCV since splitting up the data into folds results in each fold having too few observations.

Next, the usage of `tunelssvm` function is explored to determine optimal tuning parameters. We see that the girdsearch + ds (randomized directional search) method gives the lowest misclassification rate, however, all the algorithms seem to give varying result. The misclassification rate varied a little across multiple runs of each algorithm but the (γ, σ^2) displayed large variation. This may be a consequence of multiple local minima



(a) $\gamma = 1$ (b) $\gamma = 5$ (c) $\gamma = 10$

(d) $\gamma = 25$ (e) $\gamma = 50$ (f) $\gamma = 100$

Figure 11: Iris classification (10-fold crossvalidation): RBF kernel with different $\log(\sigma^2)$ values at each level of $\gamma \in \{1, 5, 10, 25, 50, 100\}$ values. The average number (across all the folds) of misclassified observations are plotted for the different (γ, σ^2) values.)

and each random start of the algorithm results in a local minima.

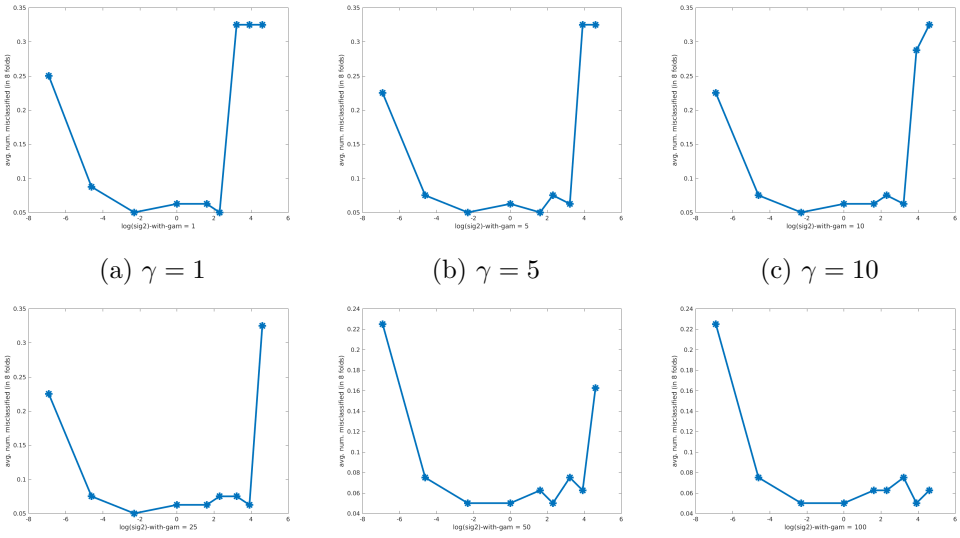
Method	Misclassification rate	γ	σ^2
simplex + csa	0.04	167.82	1.05
simplex + ds	0.05	8.92	2.38
gridsearch + csa	0.04	7.55	0.05
gridsearch + ds	0.02	0.51	8.38

Table 1: Misclassification results and optimal hyperparameter values as chosen by the `tunelssvm` function using different methods.

We can also visually assess the performance of a classifier using ROC (receiver operating characteristic) curve. This curve is usually plotted for the classification accuracy of a classifier on the validation set because using the training set will give a biased estimate of the classification accuracy since the classifier has used the training data to build the model. This is why a separate validation set needs to be used for the ROC curve (and for assessing the model in general).

1.4 Homework Problems

For each of the datasets, we do the following: plot the training and test data, try a linear SVM, try an RBF kernel and tune the parameters and plot the ROC curve.



(a) $\gamma = 1$ (b) $\gamma = 5$ (c) $\gamma = 10$

(d) $\gamma = 25$ (e) $\gamma = 50$ (f) $\gamma = 100$

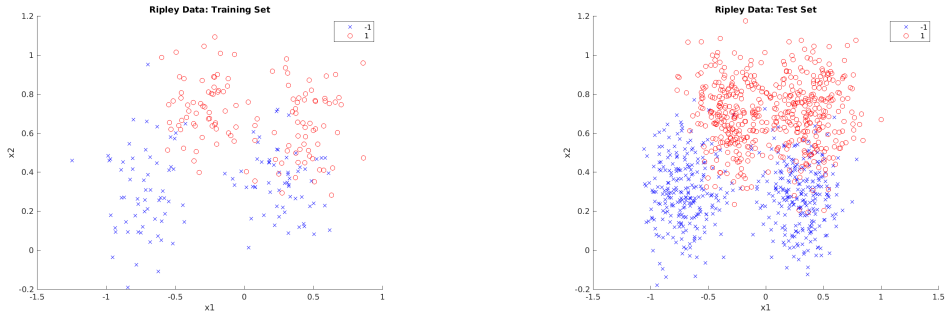
Figure 12: Iris classification (leave-one-out crossvalidation (LOOCV)): RBF kernel with different $\log(\sigma^2)$ values at each level of $\gamma \in \{1, 5, 10, 25, 50, 100\}$ values. The average number (across all the folds) of misclassified observations are plotted for the different (γ, σ^2) values.)

1.4.1 Ripley Dataset

The training ($n = 250$) and test ($n = 1000$) sets are plotted in figure 13. This dataset is easy to visualize since it only has 2 features. From the training and test sets we see that the datasets are to a large extent quite easily separable with few overlapping points. Both the linear and RBF kernel can be expected to perform quite well here. The classification accuracy (as seen in the ROC curves in figure 14) for the linear kernel is 95.9% and 96.9% for the RBF kernel. Both have similar performance but RBF leads to a higher accuracy, albeit by a small margin. Furthermore, similar to the previous cases, the performance of the classifiers remains the same despite the combination of hyperparameter values varying by a large amount at each iteration of `tunelssvm`.

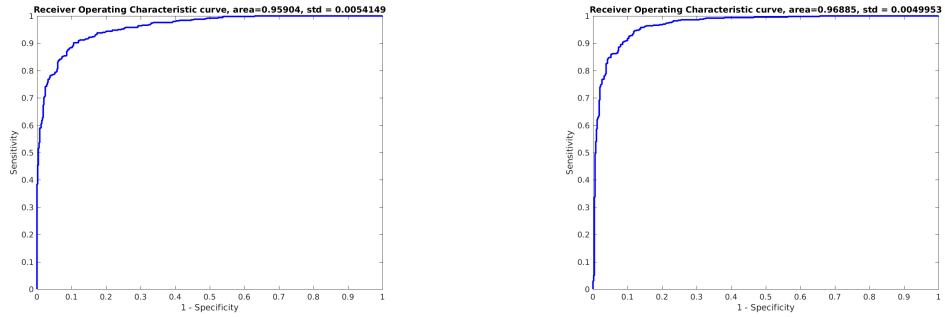
1.4.2 Breast Cancer Dataset

The training ($n = 400$) and test ($n = 169$) sets. This dataset is difficult to visualize since it has 30 features. The classification accuracy (as seen in the ROC curves in figure 15) for the both kernels is 99.56% which indicates that either kernel can be used but hyperparameter optimization must be performed to obtain the same results. The high accuracy of this method probably indicates that the two classes are quite separated which is why it is easy to assign them the correct class label. Furthermore, similar to the previous cases, the performance of the classifiers remains the same despite the combination of hyperparameter values varying by a large amount at each iteration of



(a) Training set (b) Test set

Figure 13: Ripley classification: Training and test data.



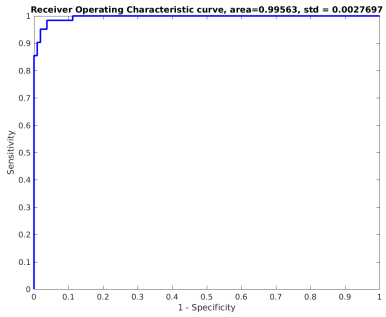
(a) Linear Kernel (b) RBF Kernel

Figure 14: Ripley classification: ROC curve for assessing classification accuracy on the test set.

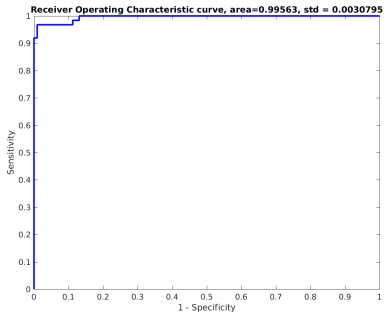
`tunelssvm`.

1.4.3 Diabetes Dataset

The training ($n = 300$) and test ($n = 168$) sets. This dataset is difficult to visualize since it has 8 features. We plotted (not shown) the first two features. It was seen from the figures that there was a lot of overlap between the two classes (at least in the subspace of first two features) and this may make it challenging for the classifier to find a separating margin with high accuracy. The classification accuracy (as seen in the ROC curves in figure 16) for the linear kernel is 84.4% and 85.3% for the RBF kernel which indicates that the RBF kernel is marginally better. Furthermore, similar to the previous cases, the performance of the classifiers remains the same despite the combination of hyperparameter values varying by a large amount at each iteration of `tunelssvm`.

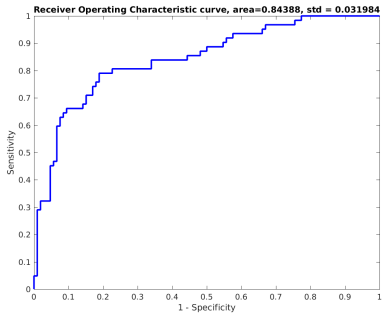


(a) Linear Kernel

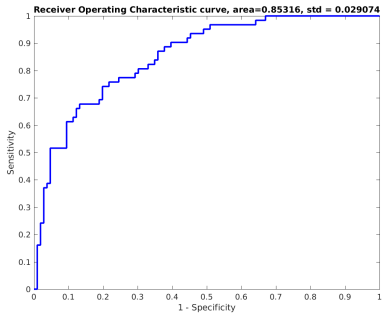


(b) RBF Kernel

Figure 15: Breast cancer classification: ROC for the linear and RBF kernels.



(a) Linear Kernel



(b) RBF Kernel

Figure 16: PIMA Indians Diabetes classification: ROC curves for the linear and RBF kernels.

2 Exercise Session 2: Function Estimation

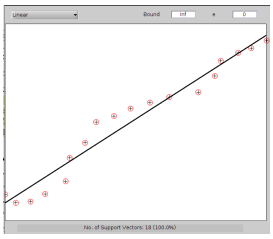
The main goal in this exercise is that of function estimation where a SVM is trained for regression (continuous response), not classification (finite response).

2.1 SVM for Regression

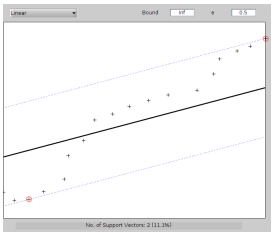
SVM can not only be used for classification but also regression. We explored the demo `uiregress` with different settings. Some of the results are shown below in figure 17.

Parameter C determines the trade off between the model complexity (flatness) and the degree to which deviations larger than ε are tolerated in optimization formulation for example, if C is too large (infinity), then the objective is to minimize the empirical risk only, without regard to model complexity part in the optimization formulation. Parameter ε on the other hand controls the width of the ε-insensitive zone, used to fit the training data. The value of ε can affect the number of support vectors used

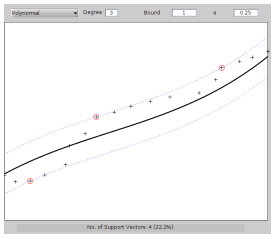
to construct the regression function. The bigger the ε, the fewer support vectors are selected. On the other hand, bigger ε-values results in more 'flat' estimates. Hence, both C and ε-values affect model complexity (but in a different way).



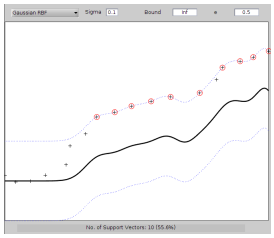
(a) Linear; $\epsilon = 0$



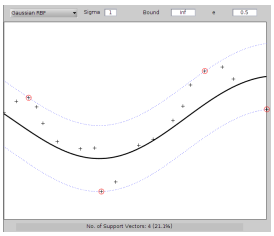
(b) Linear; $\epsilon = 0.5$



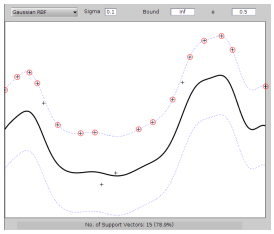
(c) Poly (3) kernel; $\epsilon = 0.25$



(d) RBF; $\epsilon = 0.5$, $\sigma^2 = 1$



(e) RBF; $\epsilon = 0.5$, $\sigma^2 = 1$



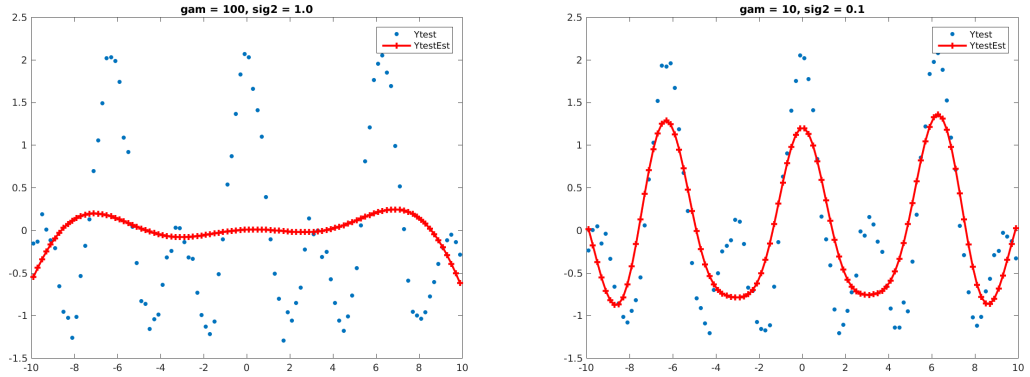
(f) RBF; $\epsilon = 0.5$, $\sigma^2 = 0.01$

Figure 17: SVM for Regression: demo `uiregress`. Figures (a)-(d) have the same points but different kernels/hyperparameter values. Figures (e) and (f) are on a different nonlinear dataset and performance of the RBF kernel with $\epsilon = 0.5$ (fixed) and σ^2 values for the hyperparameter are explored.

2.2 Sum of Cosines

The data in this section is generated as follows: $X \in [-10, 10]$ is a sequence of 201 points starting at -10 and ending at 10 with an increment of 0.1. The response vector $Y = cos(X) + cos(2X) + \varepsilon$, $\varepsilon \sim N(0, 0.1^2)$ is the added noise. The dataset is divided into (fixed) training and test sets with the odd indices of (X, Y) taken as the training set and the even indices taken as the test set. We try two (γ, σ^2) values, namely (100, 1.0) and (10, 0.1). An LS-SVM model is trained on the training data with an RBF kernel and evaluated on the test set. The true and predicted values from the test set for the two (γ, σ^2) values are plotted in figure 18.

The first set of values (100, 1.0) result in a poor performance on the test set, however the second set of parameter values (10, 0.1) result in a much better performance. However, the question remains: which parameter values result in the optimal performance of the SVM? This is explored in a methodical manner in the next section using the `tunelssvm` function. Also, experience from the previous session indicates that there is most likely not a single pair of hyperparamter values that is optimal.



(a) $\gamma = 100$, $\sigma^2 = 1.0$ (b) $\gamma = 10$, $\sigma^2 = 0.1$

Figure 18: Sum of Cosines: Function estimation and evaluation on the test set for the (γ, σ^2) values of the RBF kernel. The LS-SVM model is evaluated on the test set and the known (blue points) and predicted values (red line) are plotted.

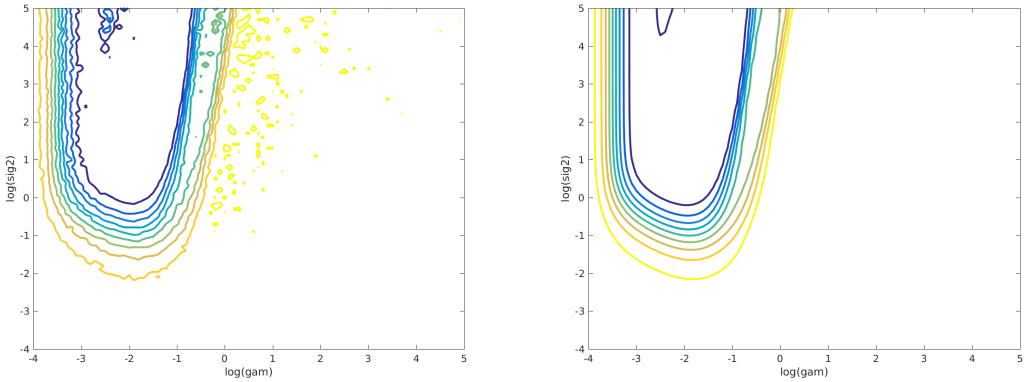
2.3 Hyper-parameter Tuning

We use the same dataset from the previous subsection. For tuning the model, we try 10-fold crossvalidation (k-fold CV) and leave-one-out crossvalidation (LOOCV). In the first part, we create a grid of (γ, σ^2) values and assess the models and fit the (γ, σ^2) values that result in the lowest error on the training set from 10-fold CV and LOO-CV. In the latter half of this section, we use the `tunelssvm` function to automatically determine the best (γ, σ^2) values and compare different methods within `tunelssvm`.

For the grid search, a subset of values on [-4,5] at increments of 0.1 are taken, which are then used as exponents with base 10 to give the corresponding γ and σ^2 values. Mean squared prediction error (across the folds in CV) is calculated for an SVM fit with each of these (γ, σ^2) values. The (γ, σ^2) value that results in the lowest error is extracted and an SVM model is trained with these values using the training set from the previous section and the performance is evaluated on the test set. These plots are given in figure 20. It can be useful to visualize the MSE of prediction from the CV procedure as a function of (γ, σ^2) values. These are plotted in figure 19.

The contour plots indicate that there plenty of (γ, σ^2) values that can give (nearly) optimal results. Although we chose the (γ, σ^2) value that had the best performance, there could be other (γ, σ^2) values that would lead to nearly the same performance. We observe this from figure 20. Both the fits are very good, however, the γ parameters are vastly different from the two CV procedures whereas the σ^2 differs by an insignificant margin in comparison.

Next, we use the in-built `tunelssvm` function to obtain (γ, σ^2) values. The results are



(a) 10-fold CV (b) LOO-CV

Figure 19: Sum of Cosines: Contour plots of the average prediction error from the CV procedure for different (γ, σ^2) values (which are plotted on $\log_{10}$ scale.

shown in table 2 below. The results on the test set are not shown since the fit appears to be as good as the fit above. Furthermore, a lot of variability was observed in (γ, σ^2) values each time the procedure was run, however the result was the same. All methods give similar performance across all the metrics.

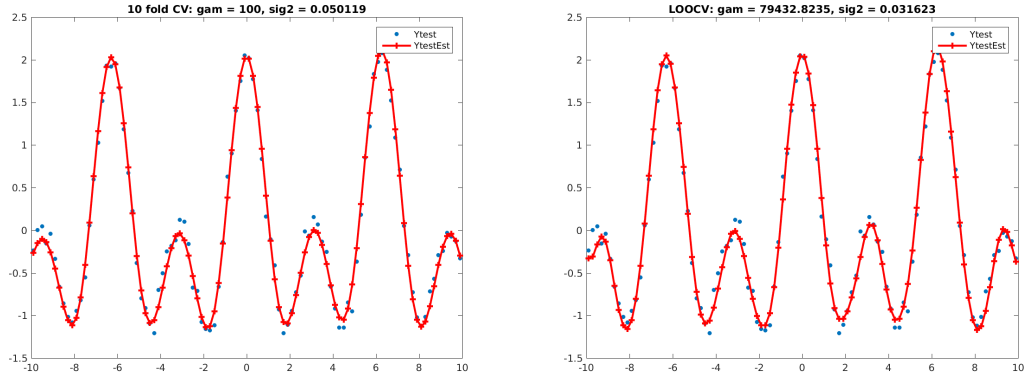
Method	avg. MSE (from CV)	γ	σ^2	runtime (secs)
gridsearch + csa	0.01	117.6	0.05	0.84
simplex + csa	0.0096	233.5	0.05	0.77
simplex + ds	0.0094	78	0.04	0.6
gridsearch + ds	0.0091	151460	0.12	0.74

Table 2: Sum of Cosines: Hyperparameter values as chosen by the `tunelssvm` function using different methods.

2.4 Using Bayesian Framework

In this section we use LS-SVM for classification and function estimation in the Bayesian framework. We continue with the sum of cosines example above. Initial values for (γ, σ^2) are chosen as (10, 0.5). The posterior parameter values as obtained from the Bayesian framework is (13.44, 0.76) and the corresponding figure shows a plot of the estimated function along with error bars. The MSE on the test set is 0.94 which is rather high. Here the initial (γ, σ^2) values were rather arbitrary, perhaps we can get improved performance using the (γ, σ^2) values from the previous section.

(γ, σ^2) values taken as (100, 0.05) shows a slight improvement in performance with an $\text{MSE}_{test} = 0.78$ and the updated posterior values for (γ, σ^2) as (5.38, 0.17). This



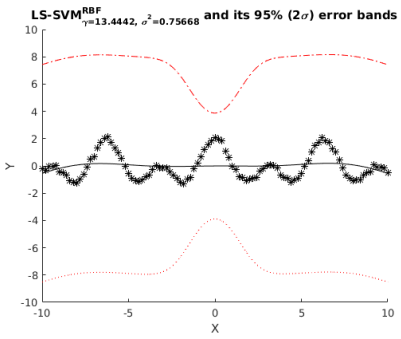
(a) $(\gamma, \sigma^2) = (100, 0.05)$ (b) $(\gamma, \sigma^2) = (79432, 0.03)$

Figure 20: Sum of Cosines: Observed (red) and predicted (blue) values for the test set using the best (γ, σ^2) values obtained from the 10-fold CV (left) and LOOCV (right) procedures.

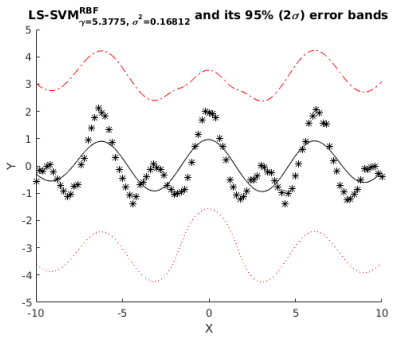
result is somewhat surprising because one would expect the performance to coincide (or at least be similar) with the non-Bayesian method for optimal (γ, σ^2) values. Two more sets of values were tried: (150, 0.05) (posterior: (5.37, 0.09); $\text{MSE}_{test} = 0.52$) and (150, 0.001) (posterior: (10552, 0.001); $\text{MSE}_{test} = 0.02$). These are visualized in figure 21 below. Other (γ, σ^2) values were attempted and it was observed that the result was rather sensitive to the choice of (γ, σ^2) values. Furthermore, $(\gamma, \sigma^2) = (150, 0.001)$ seemd to give the best fit to the training set and as a result had the smallest error on the test set.

Next, we use the Bayesian framework for classification on the iris data. (γ, σ^2) values of (5,0.75) were tried initially which gave perfect accuracy on the test data. The result is visualized in figure 22(a). The posterior (γ, σ^2) values were the same. Next, we tried (γ, σ^2) values obtained from `tunelssvm` (using 10-fold CV) on the training set. These (135.5, 0.34) were chosen as initial (prior) values for the hyperparameters. This was fed into the Bayesian route which gave the following posterior values (14.49, 0.35) and perfect classification accuracy on the test set as well. Since the dataset resides in two dimensional feature space, the decision boundary can be visualized easily and is presented in figure 22(b).

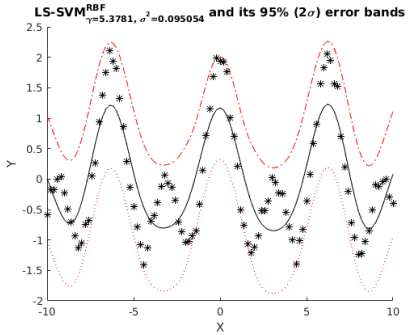
Compared to classification problems from exercise 1, the decision boundary is not sharp but fuzzy and different shades which indicates that the probability of a point belonging to a certain class decreases with an increasing distance of the point to that class. For example, positive class (pink) is taken as the reference class so the shade closest to this indicates high probability of belonging in positive class and the blue region contains points which have a low probability of being in positive class (hence more likely to be in negative class).



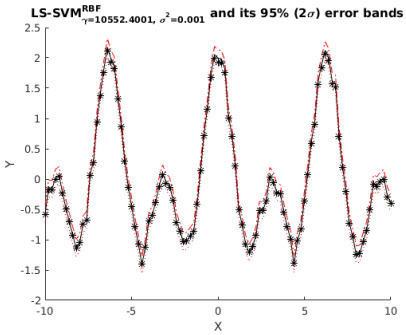
(a) prior: (10, 0.05), post: (13.44, 0.76)



(b) prior: (100, 0.05), post: (5.38, 0.17)



(c) prior: (150, 0.05), post: (5.37, 0.09)



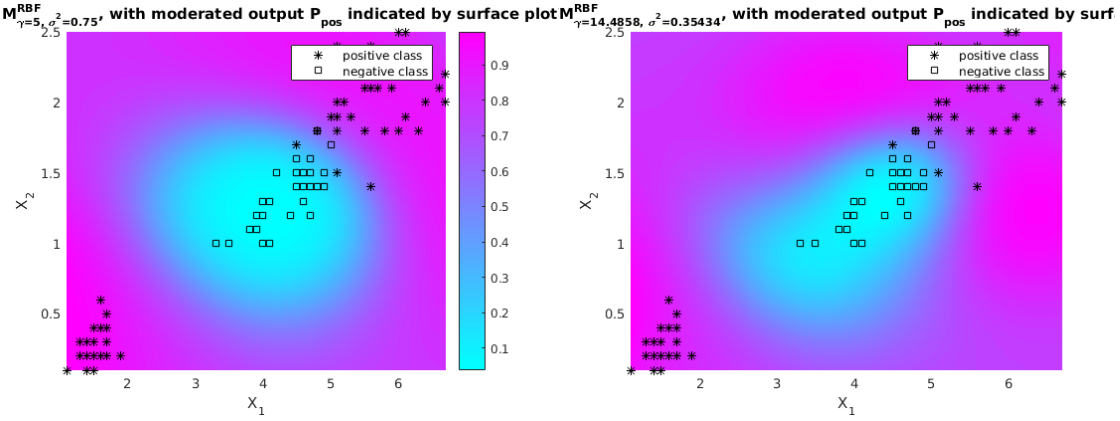
(d) prior: (150, 0.001), post: (10552, 0.001)

Figure 21: Bayesian Framework for LS-SVM: (sum of cosines) Observed (stars) and predicted (solid line) (with 95% intervals) values for the training set using the best (γ, σ^2) values obtained from the Bayesian formulation.

Finally, we use the same approach but for feature selection by using ARD (automatic relevance determination) to detect the most important features. A sum of cosines similar to the previous case is employed with 3 features being created but only 1 feature being used to generate the response. Thus, we have 3 features but only one should be relevant in predicting the response. It is observed that the ARD method is able to select the 1st feature as the most relevant feature. The results can be visualized using 2d plots of the feature vs the response variable (however this would not be feasible for problems with a lot of features). We could also use crossvalidation to determine the most important features by training multiple models (using one feature at a time) and get the avg. MSE score from CV to determine the most relevant inputs resulting in the lowest CV error rates.

2.5 Robust Regression

In this section, we explore the effect of outliers on the SVM classifier. Similar to the sum of cosines example, $X \in [-10, 10]$ at increments of 0.2 and $Y = cos(X) + cos(2X) + \varepsilon$,



(a) prior: (5, 0.75), post: (5, 0.75) (b) prior: (135, 0.34), post: (14.49, 0.35)

Figure 22: Bayesian Framework for LS-SVM: (iris classification) decision boundary in the two cases. The decision boundary is not solid like the previous cases in exercise 1 but fuzzy which indicates the uncertainty around the constructed decision boundary.

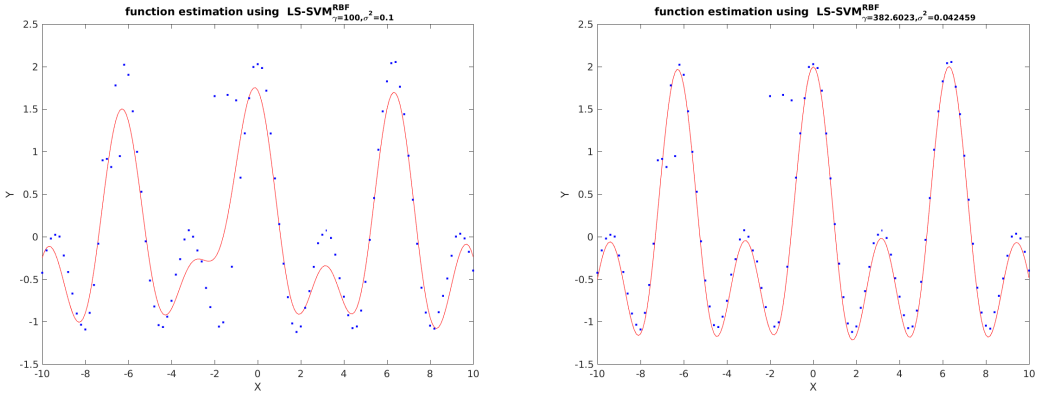
$\varepsilon \sim N(0, 0.1^2)$. However, 3 outliers are added to Y where $Y = 0.7 + \varepsilon$, $\varepsilon \sim N(0, 0.3^2)$ and 3 outliers are added to Y where $Y = 1.5 + \varepsilon$, $\varepsilon \sim N(0, 0.2^2)$. Thus, these outliers are said to come from different populations than the bulk of the data.

Initial values for $\gamma = 100$ and for $\sigma^2 = 0.1$. An LS-SVM is fit to the data and the results are shown in figure 23(a). It is not so clear that outliers exert a pull on the SVM fit, however `tunelssvm` was run a few times (not shown) and that led to wildly different fits at each run providing some evidence for the effect of outliers on the SVM. In figure 23(b) the fit is nearly perfect (MAE = 0.11 from 10-fold CV) and the effects of outliers is reduced by using the Huber loss function. However, in this case, the parameters were chosen using `tunelssvm` and displayed the same variability observed in the previous sections.

Hampel, logistic and myriad (robust) loss functions were also tried and led to similar results: Hampel: MAE = 0.11 from 10-fold CV; logistic: MAE = 0.11 from 10-fold CV and myriad: MAE = 0.11 from 10-fold CV. The results have been rounded but were almost the same. Since the results are almost the same, the plots for the other three robust loss functions are omitted.

Furthermore, mean absolute error (MAE) is used for crossvalidation instead of mean squared error (MSE) because the MSE is more sensitive to outliers. Minimising MSE can lead to the estimator being grossly affected by the outliers (since the aim is to minimize the squared deviations between the fitted and observed values), whereas minimising MAE leads to a dampened effect of outliers on the estimator (as compared to MSE)

since an attempt is made to minimize the absolute deviations $\varepsilon_i = |y_i - \hat{y_i}|$ instead of squared deviations $\varepsilon_i = (y_i - \hat{y_i})^2$.



(a) Non-robust fit

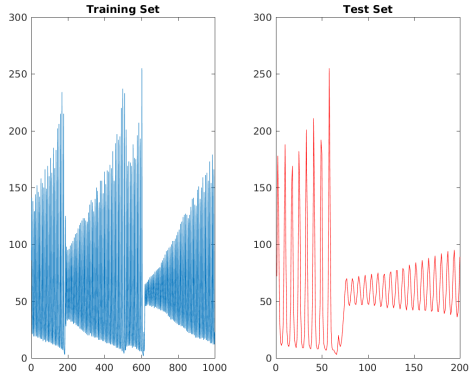
(b) Robust fit using Huber loss

Figure 23: Sum of Cosines: comparison between non-robust and robust fit in the presence of outliers

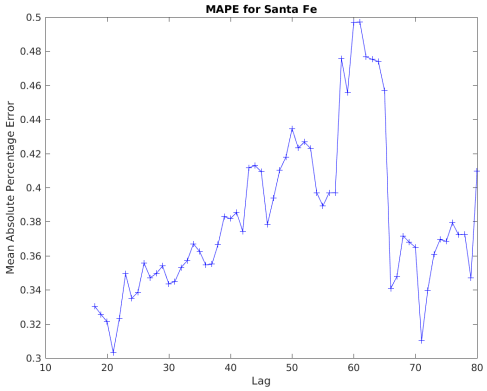
2.6 Homework Problems: Santa Fe Prediction

This part involves time series prediction task. The training set consists of 1000 points and test set consists of 200 points. The training and test sets are plotted below in figure 24. The aim is to train an SVM to predict the future values, the performance of which can be assessed on the test data. In the time series case, our goal is function estimation but using the past values of a series to predict the future values. Such an approach is called (nonlinear) autoregressive model, since the series is being regressed on itself. The aim is to figure out how many time points in the past have an influence on the current value. Modelling this serial dependence in the series can allow us to predict future values.

We initially fit a NAR model with 50 lags for the AR component. The hyperparameters were chosen using `tunelssvm` using 10-fold crossvalidation and MAE as the error criterion although the data were not preprocessed since we have time series data. The predicted series and test series for lag 50 are shown below in figure 25. One might be inclined to consider that there might be other lags that might give an equally good if not better fit. To this extent, we considered lags = {5, 6, 7, 8, . . . , 78, 79, 80} and recorded the MAPE at each iteration. The minimum MAPE was 0.3032 at lag 21 and the MAPE at lag 50 was 0.4233. The predictions for the two lags as well as plot of lag vs MAPE is given below as well. Order 50 was a decent choice, but not the one with the lowest error.



(a) Training (left) and test (right) series



(b) Lag (15-80) vs MAPE

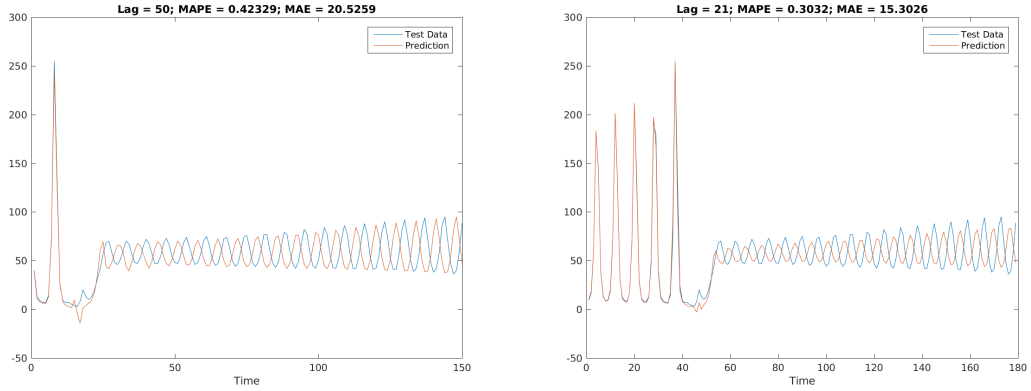
Figure 24: Time Series Prediction: Santa Fe dataset. Series plotted on the left. Right shows the MAPE on the test series as a function of lags.

3 Exercise Session 3: Unsupervised Learning

3.1 Kernel PCA

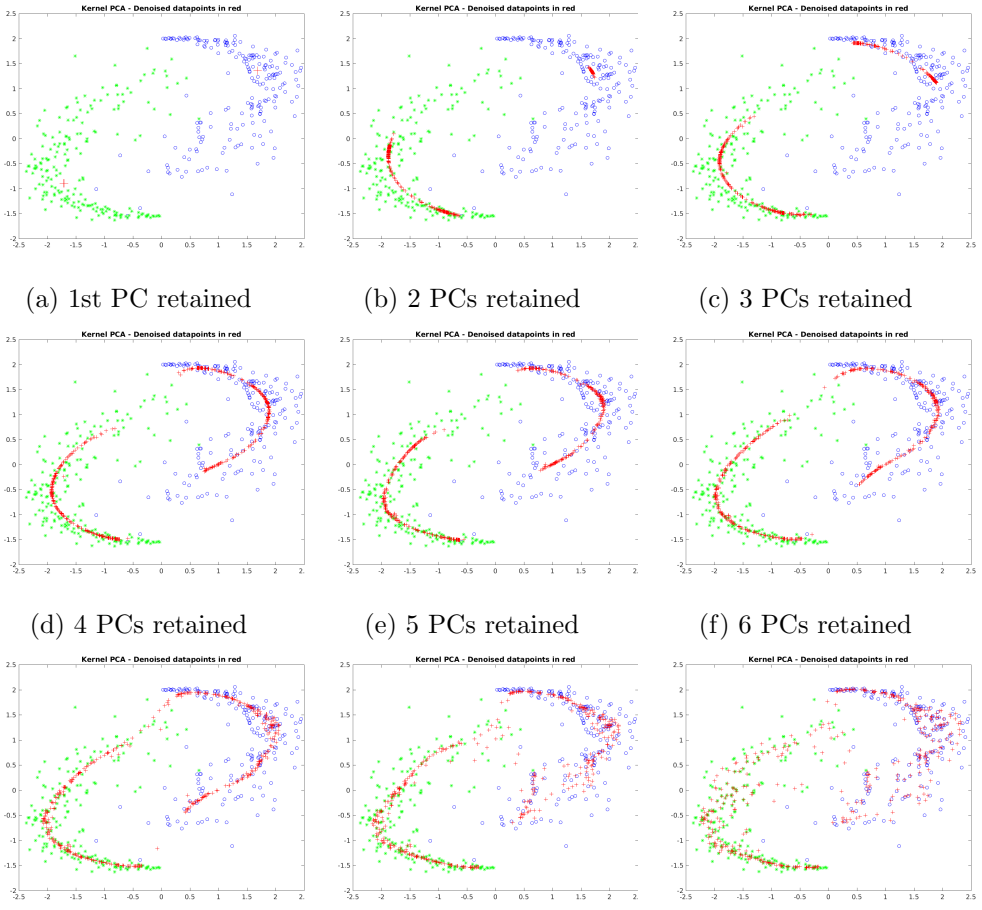
In this section we explore kernel PCA (which is a PCA in the kernel-induced feature space) as a technique for denoising. This toy example is used to study the choice of: number of components, choice of the kernel and the kernel hyperparameter. In each example, we consider 400 points which are divided into two clusters of size 200 each. Noise was added to the generated data (fixed at 0.5) since that is a relatively large amount of noise. We tried reconstructing the original dataset using {1,2,3,4,5,6,8,10,15} principal components (PCs) and observing the results. Since the amount of noise is not small, one would normally expect an improvement in reconstruction with increasing components but this improvement will become marginal after a certain number of PCs. Furthermore, the value of the hyperparameter σ^2 for the RBF kernel is fixed at 0.4. With kernel PCA, one has as many principal components as data points N as compared to linear PCA where the maximum is the dimension of the feature space p.

Furthermore, in case of linear PCA, the data points are projected onto linear axes with maximal variance. In the toy example, the maximum variation in the data is better explained by projection onto nonlinear space as opposed to a linear one. Thus, in figure 26 we see that the first PC is unable to capture the shape of the data. However, 4-6 PCs appear to give a good reconstruction of the underlying function. By the time we get to the 10th and 15th component, we see that those components are largely learning the noise in the data, not to mention the separation between the two clusters becomes blurred.



(a) Lag 50; MAPE = 0.4233 (b) Lag = 21; MAPE = 0.3032

Figure 25: Time Series Prediction: Santa Fe dataset, prediction on the test set using lag 50 and lag 21 for the NAR model trained using LSSVM.



(a) 1st PC retained (b) 2 PCs retained (c) 3 PCs retained

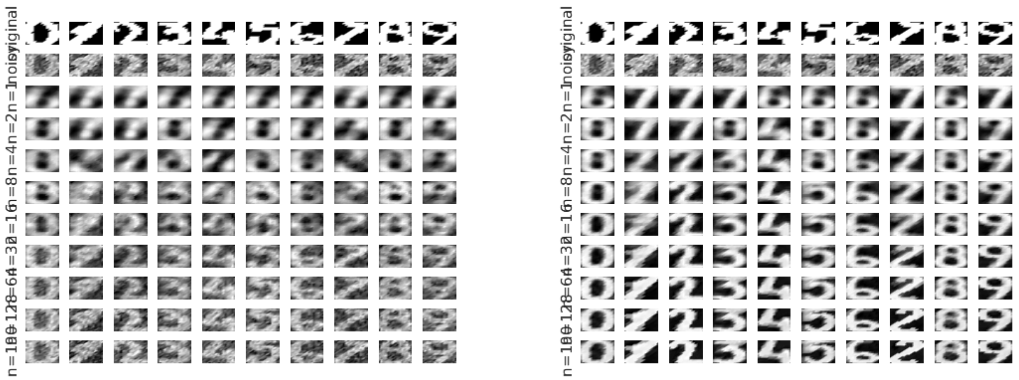
(d) 4 PCs retained (e) 5 PCs retained (f) 6 PCs retained

(g) 8 PCs retained (h) 10 PCs retained (i) 15 PCs retained

Figure 26: Kernel PCA: Denoising dataset with 2 features and 2 clusters. Kernel PCA is run to learn the shape of the clusters and different principal components are retained. Reconstructed principal components are plotted in red with the underlying data in blue and green (2 clusters) respectively.

3.2 Handwritten Digit Denoising

Here kernel PCA and linear PCA are used for handwriting (digit) recognition in the presence of noise. The results from both methods are visualized in figure 27. We can see that the linear PCA even with a large number of components is unable to learn in a noiseless manner the true digit. As for the kernel PCA, the results are much better where the black (noise) and white (digit) are well separated and sharp with an increasing number of principal components (each row). This highlights the success of kernel PCA at learning the underlying structure of the data when this form is nonlinear.



(a) Linear PCA (b) Kernel PCA

Figure 27: Denoising digits using linear and kernel PCA.

3.3 Spectral Clustering

In this section the aim is to perform spectral clustering (2 rings so 2 clusters). The figures are visualized in figure 28. It is seen that σ^2 value of 0.01 results in the best clustering result where the two rings are learned completely. The projection onto the subspace of 2nd and 3rd eigenvectors shows the separation between the two clusters.

3.4 Fixed-size LS-SVM

In this section, we study the Fixed Size LS-SVM method where the number of target support vectors is fixed in advance and iterative methods are used to select the subset that best learns the decision boundary. 4 values of the hyperparameter σ^2 are tried: 0.01, 0.1, 0.5 and 1. The figures are plotted below in figure 29. From the figures, we see that as σ^2 increases, the distance between any two sets of points tends to increase. Furthermore, with larger σ^2 values, the chosen support vectors tend towards a more evenly spaced set of points.

The algorithm converges to the subset with maximum entropy, i.e., the subset of a fixed

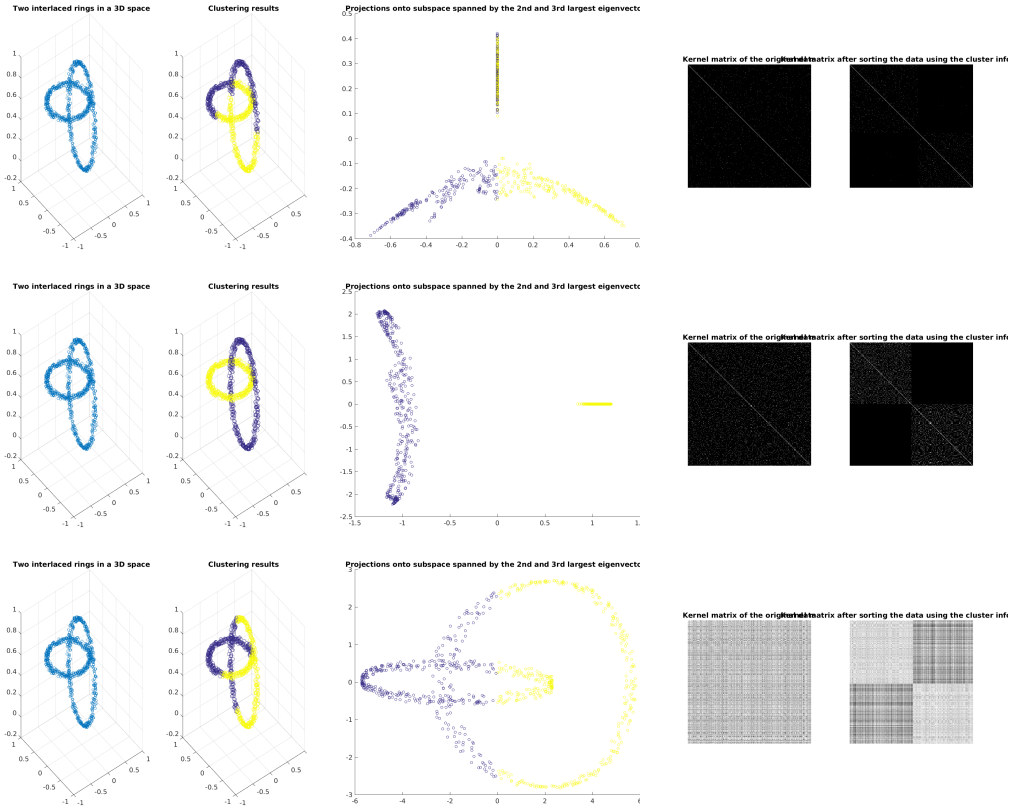
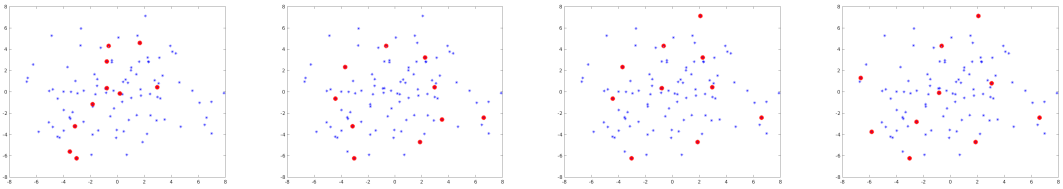


Figure 28: Spectral Clustering: Top row with $\sigma^2 = 0.001$; middle row with $\sigma^2 = 0.01$; bottom row with $\sigma^2 = 1$. First column: shows Original Data (left) and results from the clustering (right). Middle column: Projection onto subspace spanned by 2nd and 3rd eigenvectors. Right column: kernel matrix of the original data (left) and sorted data after clustering (right).

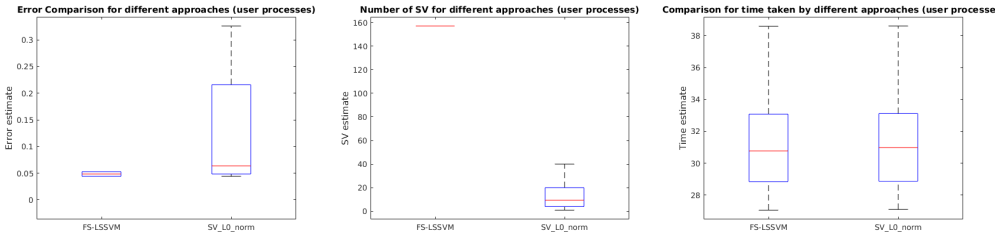
size that best describes the structure of the data.

Next, we compare a fixed-size LS-SVM to an ℓ_0-type approximation. The figures are given in figure 30. From the figures we see that the median error rate is quite same for both methods, however, the fixed size solution shows much less variability as compared to the ℓ_0 solution. The number of support vectors are fixed around 160 for the fixed-size case however the median number of support vectors in the ℓ_0 case is somewhere between 5-10% of the SVs in the FS case.



(a) $\sigma^2 = 0.01$ (b) $\sigma^2 = 0.1$ (c) $\sigma^2 = 0.5$ (d) $\sigma^2 = 1$

Figure 29: 3.4.1: Illustration of fixed-size LS-SVM for a two dimensional problem using the RBF kernel.



(a) Error Rate (b) No. of support vectors (c) Computation time

Figure 30: 3.4.3: FS-LSSVM vs ℓ_0-type approximation

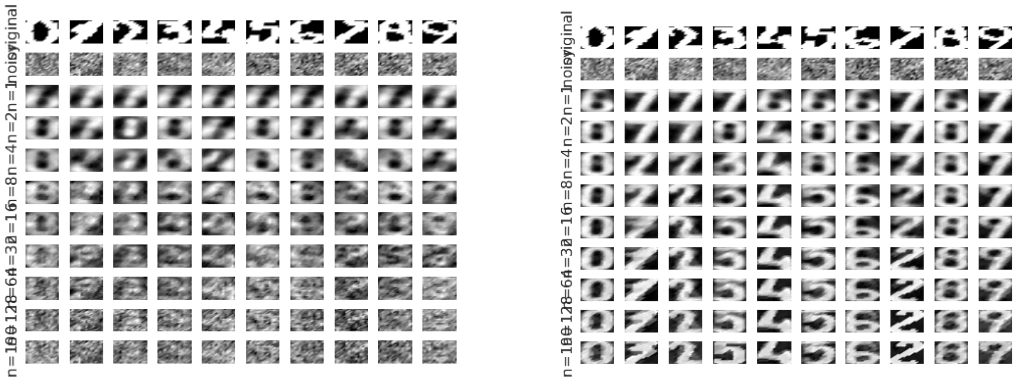
3.5 Homework Problems

3.5.1 Handwritten Digit Denoising

A rule of thumb for the σ^2 parameter in the kernel PCA is taken as $\sigma^2 = p * \mathbb{E}[var(x_i)]$ where $i = 1, \ldots, p$ is the number of features in the data. In our dataset, $\sigma^2 = 35.91$. Increasing the σ^2 value results in a degradation in the performance of the method. Furthermore, values of *sigmafactor* were tried on the logarithmic scale and small values led to the model being unable to learn the digits in either case although the performance in the kernel PCA was much worse than linear PCA. At a sigmafactor value of 0.01, it was able to learn some of the digits in the kernel PCA case but not in the linear PCA case. At sigmafactor = 10, the network was also unable to learn the digits irrespective of the PCA method. This indicates the importance of selecting the right sigmafactor value. Next, we set the $noisefactor$ to 1 and display the results from the linear and kernel PCA in figure 31. We can see that the performance of the linear PCA degrades completely but the Kernel PCA is still able to learn some of the digits for different values of the n which is the number of components.

3.5.2 FS-SVM: Classification

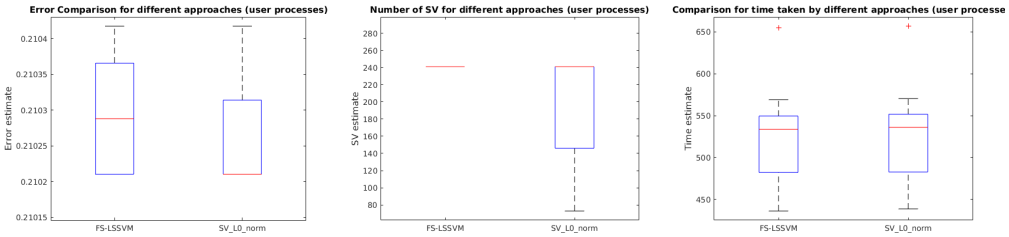
FS-LSSVM is used for classification on the shuttle dataset from the UCI ML repository. The results (for $k = 1$ to reduce computational time) are given below in figure 32. From the figures we see that the distribution of error rates is nearly the same for both



(a) Linear PCA (b) Kernel PCA

Figure 31: Difference between linear and kernel PCA with $noisefactor = 1$ on the test set for handwriting recognition.

the methods, however the FS-LSSVM has a higher median error rate compared to the sparser solution using the ℓ_0 penalty. From the middle figure, we see that in case of FS-LSSVM, the number of support vectors used in constructing the decision boundary is constant as compared to the $\ell_0 - norm$ solution which at it's lowest point uses only a fourth of the support vectors compared to the FS-LSSVM procedure. Finally the last figure shows that the computation time is the same for both approaches.



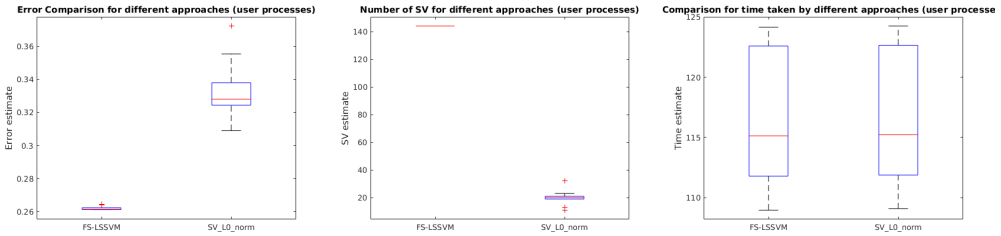
(a) Error Rate (b) No. of support vectors (c) Computation time

Figure 32: FS-LSSVM for classification on the shuttle dataset.

3.5.3 FS-SVM: Regression

Compared to the previous section, here FS-LSSVM is used for regression on the california (housing) dataset from the UCI ML repository. The results (for $k = 1$ to reduce computational time) are given below in figure 33. From the figures we see that the distribution of error rates is much higher for the sparser solution using the ℓ_0 penalty as compared to the FS-LSSVM which has a much lower error rate. From the middle figure, we see that in case of FS-LSSVM, the number of support vectors used in constructing the decision

boundary is constant as compared to the $\ell_0 - norm$ solution which at it's median uses less than 10% of the support vectors compared to the FS-LSSVM procedure. Finally the last figure shows that the computation time is the same for both approaches.



(a) Error Rate (b) No. of support vectors (c) Computation time

Figure 33: FS-LSSVM for regression on the california housing dataset.

4 Appendix (MATLAB Code)

Session 1

```
%% adding the path with the SVM and LSSVM toolboxes
addpath('/home/ad/Desktop/KUL Course Material/SVM (support vector machines)
    /Exercise Session/svm/');
addpath('/home/ad/Desktop/KUL Course Material/SVM (support vector machines)
    /Exercise Session/LSSVMlab/');
addpath('/home/ad/Desktop/KUL Course Material/SVM (support vector machines)
    /Exercise Session/fixed-size/');
clear;
clc;
%% Classification
%% (1.1) Classifying two Gaussians
X1 = 1 + randn(50,2);
X2 = -1 + randn(51,2);
Y1 = ones(50,1);
Y2 = -ones(51,1);
X  = [X1;X2];
Y = [Y1;Y2];

%% Plotting
figure;
hold on;
plot(X1(:,1), X1(:,2), 'ro', 'LineWidth', 3);
plot(X2(:,1), X2(:,2), 'bo', 'LineWidth', 3);
hold off;
title('Two class Gaussian RV');
xlabel('x');
ylabel('y');
print('/home/ad/Desktop/KUL Course Material/SVM (support vector machines)/
    Exercise Session/images/plot1-1', '-dpng');

```

```
%% (1.2) Support Vector Machine
% http://cs.stanford.edu/people/karpathy/svmjs/demo/
%% (1.3) LS-SVMlab
%% Demo
democlass;
%%
help prelssvm
%%
clear;
clc;

%% Iris data
load('/home/ad/Desktop/KUL Course Material/SVM (support vector machines)/Exercise Session/Session 1/iris.mat')
%% Fit the model
gam = 1.0;
type = 'lin_kernel';

[alpha, b] = trainlssvm({X, Y, 'c', gam, [], type});
plotlssvm({X, Y, 'c', gam, [], type, 'preprocess'}, {alpha, b});
print('/home/ad/Desktop/KUL Course Material/SVM (support vector machines)/Exercise Session/images/iris-linear', '-dpng');

%% performance on test set
[Yht, Zt] = simlssvm({X,Y,type,gam,[],'lin_kernel'}, {alpha,b}, Xt);
err = sum(Yht ~= Yt);
fprintf('\n on test: #misclass = %d, error rate = %.2f%%\n', err, err/length(Yt)*100)

%% 1.3.1 (tuning hyperparameters) LS-SVM
% set the parameters to some value
gam = 0.1;
sig2 = 20;

% generate random indices
idx = randperm(size(X,1));

% create the training and validation sets
% using the randomized indices
Xtrain = X(idx(1:80),:);
Ytrain = Y(idx(1:80));
Xval = X(idx(81:100),:);
Yval = Y(idx(81:100));

% train the model
[alpha,b] = trainlssvm({Xtrain,Ytrain,'c',gam,sig2, 'RBF_kernel'});

% evaluate it on Xval:
estYval = simlssvm({Xtrain,Ytrain,'c',gam,sig2,'RBF_kernel'},{alpha,b},Xval);

%% model evaluation (fixed validation set)
% generate random indices
idx = randperm(size(X,1));
```

```
77
78  % create the training and validation sets
79  % using the randomized indices
80  Xtrain = X(idx(1:80),:);
81  Ytrain = Y(idx(1:80));
82  Xval = X(idx(81:100),:);
83  Yval = Y(idx(81:100));
84
85  gamlist=[1, 5, 10, 25, 50, 100];
86  sig2list=[0.001, 0.01, 0.1, 1, 5, 10, 25, 50, 100];
87
88  for gam=gamlist,
89      errlist=[];
90
91      for sig2=sig2list,
92          % train the model
93          [alpha,b] = trainlssvm({Xtrain,Ytrain,'c',gam,sig2, 'RBF_kernel'});
94          % evaluate it on Xval:
95          estYval = simlssvm({Xtrain,Ytrain,'c',gam,sig2,'RBF_kernel'},{alpha
                ,b},Xval);
96          % calculate the error
97          err = sum(estYval ~= Yval); errlist = [errlist; err];
98          fprintf('\n on test: # misclass = %d, error rate = %.2f%% \n', err,
                err/length(Yval)*100)
99      end
100
101     % make a plot of the % misclassification wrt. sig2 for each gam val
102     figure;
103     plot(log(sig2list), ((errlist)/(length(Yval))*100), '*-', 'LineWidth',
            3, 'MarkerSize', 12),
104     ylim([0 max(((errlist)/(length(Yval))*100))+10]),
105     xlabel(sprintf('log(sig2)-with-gam = %s', num2str(gam))), ylabel('%
            misclassified'),
106     file_string = sprintf('/home/ad/Desktop/KUL Course Material/SVM (
            support vector machines)/Exercise Session/images/iris-rbf-fixed-val
            -gam-%s', num2str(gam));
107     print(file_string, '-dpng');
108 end
109
110 %%
111 clc;
112
113 %% model evaluation (10-fold CV)
114 gamlist=[1, 5, 10, 25, 50, 100];
115 sig2list=[0.001, 0.01, 0.1, 1, 5, 10, 25, 50, 100];
116
117 for gam=gamlist,
118     errlist=[];
119
120     for sig2=sig2list,
121         % calculate the prediction error
122         err = crossvalidate({Xtrain, Ytrain, 'c', gam, sig2, 'RBF_kernel'},
                10, 'misclass', 'mean');
123         errlist = [errlist; err];
```

```
124         %fprintf('\n on test: #misclass = %d, error rate = %.2f%% \n', err,
                (err/10)*100)
125     end
126
127     % make a plot of the % misclassification wrt. sig2 for each gam val
128     figure;
129     plot(log(sig2list), errlist, '*-', 'LineWidth', 3, 'MarkerSize', 12),
130     xlabel(sprintf('log(sig2)-with-gam = %s', num2str(gam))), ylabel('avg. num. misclassified (in 8 folds)'),
131     file_string = sprintf('/home/ad/Desktop/KUL Course Material/SVM (support vector machines)/Exercise Session/images/iris-rbf-cv-gam-%s', num2str(gam));
132     print(file_string, '-dpng');
133 end
134
135 %% model evaluation (LOO-CV)
136 gamlist=[1, 5, 10, 25, 50, 100];
137 sig2list=[0.001, 0.01, 0.1, 1, 5, 10, 25, 50, 100];
138
139 for gam=gamlist,
140     errlist=[];
141
142     for sig2=sig2list,
143         % calculate the prediction error
144         err = leaveoneout({Xtrain, Ytrain, 'c', gam, sig2, 'RBF_kernel'}, 'misclass', 'mean');
145         errlist = [errlist; err];
146         %fprintf('\n on test: #misclass = %d, error rate = %.2f%% \n', err,
                (err/10)*100)
147     end
148
149     % make a plot of the % misclassification wrt. sig2 for each gam val
150     figure;
151     plot(log(sig2list), errlist, '*-', 'LineWidth', 3, 'MarkerSize', 12),
152     xlabel(sprintf('log(sig2)-with-gam = %s', num2str(gam))), ylabel('avg. num. misclassified (in 8 folds)'),
153     file_string = sprintf('/home/ad/Desktop/KUL Course Material/SVM (support vector machines)/Exercise Session/images/iris-rbf-loocv-gam-%s', num2str(gam));
154     print(file_string, '-dpng');
155 end
156
157 %% Using tunelssvm
158 clc;
159 model = {X,Y,'c',[],[],'RBF_kernel','csa'}; % csa vs ds
160 [gam,sig2,cost] = tunelssvm(model,'gridsearch', 'crossvalidatelssvm', {10,'misclass'});
161 gam
162 sig2
163 cost
164 % simplex vs gridsearch
165
166 %% ROC curve
167 gam = 0.1;
```

```
168 sig2 = 2;
169 [alpha,b] = trainlssvm({Xtrain,Ytrain,'c',gam,sig2,'RBF_kernel'});
170 [Ysim,Ylatent] = simlssvm({Xtrain,Ytrain,'c',gam,sig2,'RBF_kernel'}, {alpha
        ,b},Xval);
171 roc(Ylatent,Yval);
172
173 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
174 %%-------------------------------------------------------------------
175 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
176 %% Ripley dataset
177 clear;
178 clc;
179 load('/home/ad/Desktop/KUL Course Material/SVM (support vector machines)/
        Exercise Session/Session 1/ripley.mat')
180
181 %% plot the data
182 figure;
183 hold on;
184 % plot the train set
185 gscatter(X(:,1), X(:,2), Y, 'br', 'xo');
186 % plot the test set
187 %gscatter(Xt(:,1), Xt(:,2), Yt, 'br', 'xo');
188 hold off;
189 title('Ripley Data: Training Set');
190 %title('Ripley Data: Test Set');
191 xlabel('x1');
192 ylabel('x2');
193 % plot the train set
194 print('/home/ad/Desktop/KUL Course Material/SVM (support vector machines)/
        Exercise Session/images/ripley-train', '-dpng');
195 % plot the test set
196 %print('/home/ad/Desktop/KUL Course Material/SVM (support vector machines)/
        Exercise Session/images/ripley-test', '-dpng');
197
198 %% Model
199 kernel = 'RBF_kernel'; % 'RBF_kernel' or 'lin_kernel'
200 model = {X,Y,'c',[],[],kernel,'csa'};
201 [gam,sig2,cost] = tunelssvm(model,'simplex', 'crossvalidatelssvm', {10,'
        misclass'});
202 [alpha,b] = trainlssvm({X,Y,'c',gam,sig2, kernel});
203 plotlssvm({X, Y, 'c', gam, sig2, kernel, 'preprocess'}, {alpha, b});
204 print('/home/ad/Desktop/KUL Course Material/SVM (support vector machines)/
        Exercise Session/images/ripley-class-rbf', '-dpng');
205 % evaluate it on Xval:
206 [Ysim, Ylatent] = simlssvm({X,Y,'c',gam,sig2,kernel},{alpha,b},Xt);
207 roc(Ylatent,Yt);
208 print('/home/ad/Desktop/KUL Course Material/SVM (support vector machines)/
        Exercise Session/images/ripley-roc-rbf', '-dpng');
209
210 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
211 %%-------------------------------------------------------------------
212 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
213 %% Breast Cancer dataset
214 clear;
```

```
215 clc;
216 load('/home/ad/Desktop/KUL Course Material/SVM (support vector machines)/
    Exercise Session/Session 1/breast.mat')
217 X = trainset;
218 Y = labels_train;
219 Xt = testset;
220 Yt = labels_test;
221
222 %% plot the training data
223 figure;
224 hold on;
225 % plot the train set
226 gscatter(X(:,1), X(:,2), Y, 'br', 'xo');
227 hold off;
228 title('Breast Cancer Data: Training Set');
229 xlabel('x1');
230 ylabel('x2');
231 % plot the train set
232 print('/home/ad/Desktop/KUL Course Material/SVM (support vector machines)/
    Exercise Session/images/breast-train', '-dpng');
233
234 %% plot the test data
235 figure;
236 hold on;
237 %plot the test set
238 gscatter(Xt(:,1), Xt(:,2), Yt, 'br', 'xo');
239 hold off;
240 title('Breast Cancer Data: Test Set');
241 xlabel('x1');
242 ylabel('x2');
243 % plot the test set
244 print('/home/ad/Desktop/KUL Course Material/SVM (support vector machines)/
    Exercise Session/images/breast-test', '-dpng');
245
246 %% Model
247 kernel = 'lin_kernel'; % 'RBF_kernel' or 'lin_kernel'
248 model = {X,Y,'c',[],[],kernel,'csa'};
249 [gam,sig2,cost] = tunelssvm(model,'simplex', 'crossvalidatelssvm', {10,'
    misclass'});
250 [alpha,b] = trainlssvm({X,Y,'c',gam,sig2, kernel});
251 plotlssvm({X, Y, 'c', gam, sig2, kernel, 'preprocess'}, {alpha, b});
252 print('/home/ad/Desktop/KUL Course Material/SVM (support vector machines)/
    Exercise Session/images/breast-class-linear', '-dpng');
253 % evaluate it on Xval:
254 [Ysim, Ylatent] = simlssvm({X,Y,'c',gam,sig2,kernel},{alpha,b},Xt);
255 roc(Ylatent,Yt);
256 print('/home/ad/Desktop/KUL Course Material/SVM (support vector machines)/
    Exercise Session/images/breast-roc-linear', '-dpng');
257
258 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
259 %%-------------------------------------------------------------------
260 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
261 %% Diabetes dataset
262 clear;
```

```
clc;
load('/home/ad/Desktop/KUL Course Material/SVM (support vector machines)/Exercise Session/Session 1/diabetes.mat')
X = trainset;
Y = labels_train;
Xt = testset;
Yt = labels_test;

%% plot the training data
figure;
hold on;
% plot the train set
gscatter(X(:,1), X(:,2), Y, 'br', 'xo');
hold off;
title('Breast Cancer Data: Training Set');
xlabel('x1');
ylabel('x2');
% plot the train set
print('/home/ad/Desktop/KUL Course Material/SVM (support vector machines)/Exercise Session/images/diabetes-train', '-dpng');

%% plot the test data
figure;
hold on;
%plot the test set
gscatter(Xt(:,1), Xt(:,2), Yt, 'br', 'xo');
hold off;
title('Breast Cancer Data: Test Set');
xlabel('x1');
ylabel('x2');
% plot the test set
print('/home/ad/Desktop/KUL Course Material/SVM (support vector machines)/Exercise Session/images/diabetes-test', '-dpng');

%% Model
kernel = 'RBF_kernel'; % 'RBF_kernel' or 'lin_kernel'
model = {X,Y,'c',[],[],kernel,'csa'};
[gam,sig2,cost] = tunelssvm(model,'simplex', 'crossvalidatelssvm', {10,'misclass'});
[alpha,b] = trainlssvm({X,Y,'c',gam,sig2, kernel});
plotlssvm({X, Y, 'c', gam, sig2, kernel, 'preprocess'}, {alpha, b});
print('/home/ad/Desktop/KUL Course Material/SVM (support vector machines)/Exercise Session/images/diabetes-class-rbf', '-dpng');
% evaluate it on Xval:
[Ysim, Ylatent] = simlssvm({X,Y,'c',gam,sig2,kernel},{alpha,b},Xt);
roc(Ylatent,Yt);
print('/home/ad/Desktop/KUL Course Material/SVM (support vector machines)/Exercise Session/images/diabetes-roc-rbf', '-dpng');
```

Session 2

```
%% adding the path with the SVM and LSSVM toolboxes
```

```
addpath('/home/ad/Desktop/KUL Course Material/SVM (support vector machines)/Exercise Session/svm/');
addpath('/home/ad/Desktop/KUL Course Material/SVM (support vector machines)/Exercise Session/LSSVMlab/');
%addpath('/home/ad/Desktop/KUL Course Material/SVM (support vector machines)/Exercise Session/fixed-size/');
clear;
clc;
%% Function estimation and time series prediction
%% SVM for regression
uiregress

%% Sum of cosines

%% Demo
% code taken from demofun
%demofun
X = (-3:0.2:3)';
Y = sinc(X)+0.1.*randn(length(X),1);
gam = 10;
sig2 = 0.3;
type = 'function estimation';
[alpha,b] = trainlssvm({X,Y,type,gam,sig2,'RBF_kernel'});

% evaluate
Xt = 3.*randn(10,1);
Yt = simlssvm({X,Y,type,gam,sig2,'RBF_kernel','preprocess'},{alpha,b},Xt);

plotlssvm({X,Y,type,gam,sig2,'RBF_kernel','preprocess'},{alpha,b});
hold on; plot(min(X):.1:max(X),sinc(min(X):.1:max(X)),'g-.', 'LineWidth', 3);

%% Synthetic Example
clear; clc;
X = (-10:0.1:10)';
Y = cos(X) + cos(2*X) + 0.1.*randn(length(X),1);
% training/validation and test sets are created:
Xtrain = X(1:2:length(X));
Ytrain = Y(1:2:length(Y));
Xtest = X(2:2:length(X));
Ytest = Y(2:2:length(Y));

gam = 10; % 10, 100
sig2 = 0.1; % 0.1, 1
[alpha,b] = trainlssvm({Xtrain,Ytrain,'f',gam,sig2,'RBF_kernel'});

plotlssvm({Xtrain,Ytrain,'f',gam,sig2,'RBF_kernel'},{alpha,b});

YtestEst = simlssvm({Xtrain,Ytrain,'f',gam,sig2,'RBF_kernel'}, {alpha,b}, Xtest);

plot(Xtest,Ytest,'.', 'MarkerSize', 15);
hold on;
plot(Xtest,YtestEst,'r-+', 'LineWidth', 2);
```

```
51 legend('Ytest','YtestEst');
52 title('gam = 10, sig2 = 0.1');
53 %print('/home/ad/Desktop/KUL Course Material/SVM (support vector machines)/
       Exercise Session/images/cos-reg-2', '-dpng');
54
55 %% Tuning the hyperparameters
56 seq = (-4:0.1:5);
57 N = length(seq);
58
59 cost_crossval = zeros(N);
60 cost_loo = zeros(N);
61
62 for i = 1:N,
63     gam = 10^seq(i);
64     for j = 1:N,
65         sig2 = 10^seq(j);
66         cost_crossval(i,j) = crossvalidate({Xtrain,Ytrain,'f', gam, sig2
               },10);
67         cost_loo(i,j) = leaveoneout({Xtrain,Ytrain,'f', gam, sig2});
68     end
69 end
70
71 %% plot the results for 10 fold cv
72 figure;
73 contour(seq, seq, cost_crossval, 'LineWidth', 2)
74 xlabel('log(gam)');
75 ylabel('log(sig2)');
76 zlabel('10-fold CV Error Rate');
77 %print('/home/ad/Desktop/KUL Course Material/SVM (support vector machines)/
       Exercise Session/images/tune-grid-err-cv', '-dpng');
78
79 %% evaluating the performance of the model on the test set for 10 fold CV
80 [gam_idx,sig2_idx] = find(cost_crossval == min(cost_crossval(:)));
81 gam = 10.^seq(gam_idx);
82 sig2 = 10.^seq(sig2_idx);
83
84 [alpha,b] = trainlssvm({Xtrain,Ytrain,'f',gam,sig2,'RBF_kernel'});
85
86 YtestEst = simlssvm({Xtrain,Ytrain,'f',gam,sig2,'RBF_kernel'}, {alpha,b},
       Xtest);
87
88 plot(Xtest,Ytest,'.', 'MarkerSize', 15);
89 hold on;
90 plot(Xtest,YtestEst,'r+', 'LineWidth', 2);
91 legend('Ytest','YtestEst');
92 title(sprintf('10 fold CV: gam = %s, sig2 = %s', num2str(gam), num2str(sig2
       )));
93 %print('/home/ad/Desktop/KUL Course Material/SVM (support vector machines)/
       Exercise Session/images/tune-grid-err-cv-test', '-dpng');
94
95 %% plot the results for loocv
96 figure;
97 contour(seq, seq, cost_loo, 'LineWidth', 2)
98 xlabel('log(gam)');
```

```
99  ylabel('log(sig2)');
100 zlabel('LOOCV Error Rate');
101 %print('/home/ad/Desktop/KUL Course Material/SVM (support vector machines)/
        Exercise Session/images/tune-grid-err-loo', '-dpng');
102
103 %% evaluating the performance of the model on the test set for loocv
104 [gam_idx,sig2_idx] = find(cost_loo == min(cost_loo(:)));
105 gam = 10.^seq(gam_idx);
106 sig2 = 10.^seq(sig2_idx);
107
108 [alpha,b] = trainlssvm({Xtrain,Ytrain,'f',gam,sig2,'RBF_kernel'});
109
110 YtestEst = simlssvm({Xtrain,Ytrain,'f',gam,sig2,'RBF_kernel'}, {alpha,b},
        Xtest);
111
112 plot(Xtest,Ytest,'.', 'MarkerSize', 15);
113 hold on;
114 plot(Xtest,YtestEst,'r+', 'LineWidth', 2);
115 legend('Ytest','YtestEst');
116 title(sprintf('LOOCV: gam = %s, sig2 = %s', num2str(gam), num2str(sig2)));
117 %print('/home/ad/Desktop/KUL Course Material/SVM (support vector machines)/
        Exercise Session/images/tune-grid-err-loocv-test', '-dpng');
118
119 %% Using tunelssvm
120 optFun = 'gridsearch'; % gridsearch and simplex
121 globalOptFun = 'ds'; % csa and ds
122
123 tic
124 [gam,sig2,cost] = tunelssvm({X,Y,'f',[],[],'RBF_kernel', ...
        globalOptFun},optFun,'crossvalidatelssvm',{10,'mse'})
125
126 toc
127
128 gam
129 sig2
130 cost
131
132 %% Evaluate
133 [alpha,b] = trainlssvm({Xtrain,Ytrain,'f',gam,sig2});
134 %plotlssvm({Xtrain,Ytrain,'f',gam,sig2},{alpha,b});
135
136 YtestEst = simlssvm({Xtrain,Ytrain,'f',gam,sig2,'RBF_kernel'}, {alpha,b},
        Xtest);
137
138 plot(Xtest,Ytest,'.', 'MarkerSize', 15);
139 hold on;
140 plot(Xtest,YtestEst,'r+', 'LineWidth', 2);
141 legend('Ytest','YtestEst');
142 %title(sprintf('LOOCV: gam = %s, sig2 = %s', num2str(gam), num2str(sig2)));
143
144 %% Bayes
145 clear; clc;
146 X = (-10:0.1:10)';
147 Y = cos(X) + cos(2*X) + 0.1.*randn(length(X),1);
148 % training/validation and test sets are created:
```

```
149 Xtrain = X(1:2:length(X));
150 Ytrain = Y(1:2:length(Y));
151 Xtest = X(2:2:length(X));
152 Ytest = Y(2:2:length(Y));
153
154 %%
155 %sig2 = 0.5; gam = 10;
156 sig2 = 0.001; gam = 100;
157 criterion_L1 = bay_lssvm({Xtrain,Ytrain,'f',gam,sig2},1)
158 criterion_L2 = bay_lssvm({Xtrain,Ytrain,'f',gam,sig2},2)
159 criterion_L3 = bay_lssvm({Xtrain,Ytrain,'f',gam,sig2},3)
160 [~,alpha,b] = bay_optimize({Xtrain,Ytrain,'f',gam,sig2},1);
161 [~,gam] = bay_optimize({Xtrain,Ytrain,'f',gam,sig2},2);
162 [~,sig2] = bay_optimize({Xtrain,Ytrain,'f',gam,sig2},3);
163
164 % compute error bars
165 sig2e = bay_errorbar({Xtrain,Ytrain,'f',gam,sig2},'figure');
166
167 Yest = simlssvm({Xtrain, Ytrain, 'f', gam, sig2, 'RBF_kernel'}, {alpha, b},
        Xtest);
168 mse_test = mse(Yest-Ytest);
169 mse_test
170
171 %% Bayes to iris
172 clear;
173 load iris;
174 gam = 5; sig2 = 0.75;
175 %[gam, sig2] = tunelssvm({X,Y,'c',[],[],'RBF_kernel', 'csa'}, 'simplex', '
        crossvalidatelssvm', {10, 'misclass'});
176 [~,alpha,b] = bay_optimize({X,Y,'c',gam,sig2},1);
177 [~,gam] = bay_optimize({X,Y,'c',gam,sig2},2);
178 [~,sig2] = bay_optimize({X,Y,'c',gam,sig2},3);
179 bay_modoutClass({X,Y,'c',gam,sig2},'figure');
180
181 Yest = simlssvm({X,Y,'c',gam,sig2}, {alpha, b}, Xt);
182 err = sum(Yest ~= Yt)/length(Yt)
183 roc(Yt, Yest)
184
185 %% Bayes regression
186 X = 10.*rand(100,3)-3;
187 Y = cos(X(:,1)) + cos(2*(X(:,1))) + 0.3.*randn(100,1);
188 [gam, sig2, cost] = tunelssvm({X,Y,'f',[],[],'RBF_kernel', 'csa'}, 'simplex
        ', ...
189     'crossvalidatelssvm', {10, 'mse'});
190 [selected, ranking] = bay_lssvmARD({X,Y,'f',gam,sig2});
191
192 %% Robust regression
193 clear;clc;
194 X = (-10:0.2:10)';
195 Y = cos(X) + cos(2*X) + 0.1.*rand(size(X));
196
197 % adding outliers
198 out = [15 17 19];
199 Y(out) = 0.7+0.3*rand(size(out));
```

```
out = [41 44 46];
Y(out) = 1.5+0.2*rand(size(out));

gam = 100; sig2 = 0.1;

[alpha,b] = trainlssvm({X,Y,'f',gam,sig2});
%[gam, sig2, cost] = tunelssvm({X,Y,'f',[],[],'RBF_kernel'},'simplex','crossvalidatelssvm', {10,'mse'});
plotlssvm({X,Y,'f',gam,sig2},{alpha,b});
%print('/home/ad/Desktop/KUL Course Material/SVM (support vector machines)/Exercise Session/images/non-rob-fit', '-dpng');

%% model robust
model = initlssvm(X,Y,'f',[],[],'RBF_kernel');
costFun = 'rcrossvalidatelssvm';
wFun = 'wmyriad'; % whampel, wlogistic, wmyriad
model = tunelssvm(model,'simplex',costFun,{10,'mae'},wFun);
model.costCV
model = robustlssvm(model);
plotlssvm(model);
%print(sprintf('/home/ad/Desktop/KUL Course Material/SVM (support vector machines)/Exercise Session/images/rob-%s', wFun), '-dpng');

%% Homework Problem
%% Time series prediction
clear; clc;
load('/home/ad/Desktop/KUL Course Material/SVM (support vector machines)/Exercise Session/Session 2/santafe.mat')

figure
subplot(1,2,1)
plot(Z, '-');
title('Training Set');
subplot(1,2,2)
plot(Ztest, 'r-');
title('Test Set');
%print('/home/ad/Desktop/KUL Course Material/SVM (support vector machines)/Exercise Session/images/laser', '-dpng');

%%
lag = 50; % lag of the series; lag 21 had the minimum MAPE
X = windowize(Z,1:(lag+1));
Y = X(:,end);
X = X(:,1:lag);
horizon = length(Ztest)-lag;

[gam,sig2] = tunelssvm({X,Y,'f',[],[],'RBF_kernel','csa','original'}, ...
    'simplex','crossvalidatelssvm', {10,'mae'});
% model = bay_optimize({X,Y,'f',gam,sig2,'RBF_kernel','csa','original'},1);
% model = bay_optimize({X,Y,'f',gam,sig2,'RBF_kernel','csa','original'},2);
% model = bay_optimize({X,Y,'f',gam,sig2,'RBF_kernel','csa','original'},3);
model = trainlssvm({X,Y,'f',gam,sig2, 'RBF_kernel', 'csa', 'original'});
Zpt = predict(model,Ztest(1:lag),horizon);

```

```
249 mape = mean(abs(Zpt-Ztest(lag+1:end))./abs(Ztest(lag+1:end)));
250 mae = mean(abs(Zpt-Ztest(lag+1:end)));
251 figure;
252 plot([Ztest(lag+1:end) Zpt]);
253 xlabel('Time');
254 legend('Test Data','Prediction');
255 title(sprintf('Lag = %s; MAPE = %s; MAE = %s', num2str(lag), num2str(mape),
        num2str(mae)));
256 mape
257 mae
258 %print('/home/ad/Desktop/KUL Course Material/SVM (support vector machines)/
    Exercise Session/images/laser -50', '-dpng');
259
260 %% Can see which lag had the lowest error
261 lag_all = 5:1:80;
262 mae_all = ones(1, length(lag_all));
263 mape_all = ones(1, length(lag_all));
264
265 for i = 1:length(lag_all),
266
267     lag = lag_all(i); % lag of the series
268     X = windowize(Z,1:(lag+1));
269     Y = X(:,end);
270     X = X(:,1:lag);
271     horizon = length(Ztest)-lag;
272
273     [gam,sig2] = tunelssvm({X,Y,'f',[],[],'RBF_kernel','csa','original'},
          ...
274     'simplex','crossvalidatelssvm', {10,'mae'});
275
276     model = trainlssvm({X,Y,'f',gam,sig2, 'RBF_kernel', 'csa', 'original'})
          ;
277     Zpt = predict(model,Ztest(1:lag),horizon);
278
279     mape_all(i) = mean(abs(Zpt-Ztest(lag+1:end))./abs(Ztest(lag+1:end)));
280     mae_all(i) = mean(abs(Zpt-Ztest(lag+1:end)));
281
282     figure;
283     plot([Ztest(lag+1:end) Zpt]);
284     xlabel('Time');
285     legend('Test Data','Prediction');
286     title(sprintf('Lag = %s; MAPE = %s; MAE = %s', num2str(lag), num2str(
          mape_all(i)), num2str(mae_all(i))));
287     filename = sprintf('/home/ad/Desktop/KUL Course Material/SVM (support
          vector machines)/Exercise Session/images/Time Series/lag-%s',
          num2str(lag));
288     print(filename, '-dpng');
289     close;
290
291 end
292
293 %% which lag had the smallest error
294 [lag_idx] = find(mape_all == min(mape_all(:)));
295 mape_all(lag_idx)
```

```
296 plot(lag_all(14:end), mape_all(14:end), 'b+-');
297 xlabel('Lag');
298 ylabel('Mean Absolute Percentage Error');
299 title('MAPE for Santa Fe');
300 %print('/home/ad/Desktop/KUL Course Material/SVM (support vector machines)/
    Exercise Session/images/lag-vs-error', '-dpng');
```