KU LEUVEN

Master of Artificial Intelligence

Support Vector Machines Course Report

Name : Keerthitheja Samudrala Chandrasekar
Student Number : r0773368

Prof. Johan Suykens
Artificial intelligence
KU Leuven

Contents

1. Exercise 1 : SVM Classification

1.1 Exercise 1

1.1.1 Two Gaussians :

Obtain a line to classify the data by using what you know about the distributions of the data. In which sense is it optimal ?

There are lots of possibilities to draw a line that can separate these two data sets. But, selecting an optimal line among those possibilities is a challenge. A threshold can be fixed and the line can be drawn by calculating the midpoint between the closest points of the two classes. But, when outliers occur, this threshold based technique can give a lot of misclassifications. Therefore, for our problem we use Support Vector Classifier that can generate an optimally best hyperplane to separate the two data sets. It is also known as soft margin classifier. The margin is given by the closest distance between two observations of different classes. For data of N- dimensions, classification is done using Support Vector Machines and it gives the best hyperplane in N-dimensional space that separates all the classes from one another. Since, our problem has a 2-Dimensional data, our Support Vector Classifier is a line.

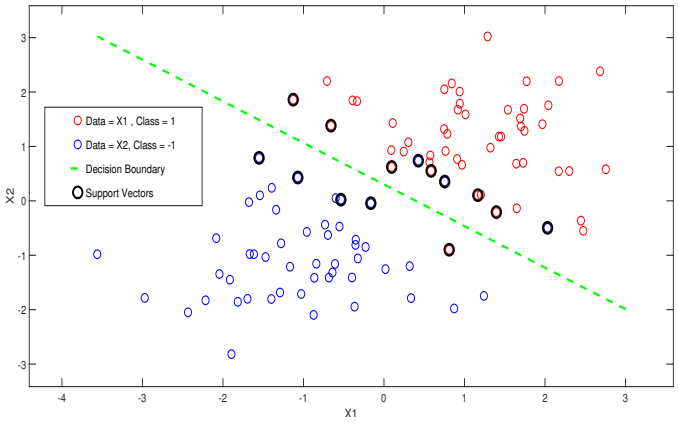


Figure 1.1: Classification

SVM uses few data points from the training data as support vectors that determines the weights and position of hyper planes. The below equation define the hyper plane of a SVM.

$$f(x) = x\beta + b \tag{1.1}$$

where β is the linear predictor coefficient and b is the bias term. These β and b terms are the optimal values of a hyper plane learnt by the classifier as it maximizes the distance between the hyper plane and support vectors. This is done by solving the Lagrange equation using optimization techniques. We train the SVM classifier on our input data set using *fitcsvm* in Matlab. Hence, the separating line (in our case the green line in the figure 1.1) that is drawn using the values of β and b is an optimal one that allows only few misclassifications as shown in the figure 1.1.

1.1.2 Support Vector Machine Classifier:

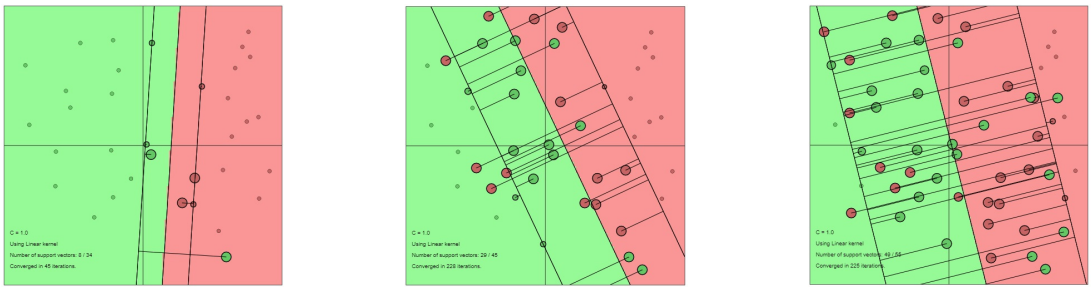
For classes with linearly separable observations, SVM tries to classify the points with a line in such a way that the separation between those classes is maximum. As we know from solving the Lagrangian equation, the solution must satisfy the following the relations

$$w = \sum_{i=1}^{N} \alpha_i y_i x_i, \sum_{i=1}^{N} \alpha_i y_i = 0 \tag{1.2}$$

Solving the above equation 1.2 using dual problem we get α_i.

The below figures 1.2 and 1.3 shows SVM classifier with linear and RBF kernels. In figure 1.2a and 1.3a it can be seen that the data points are on the right side of their respective classes. There

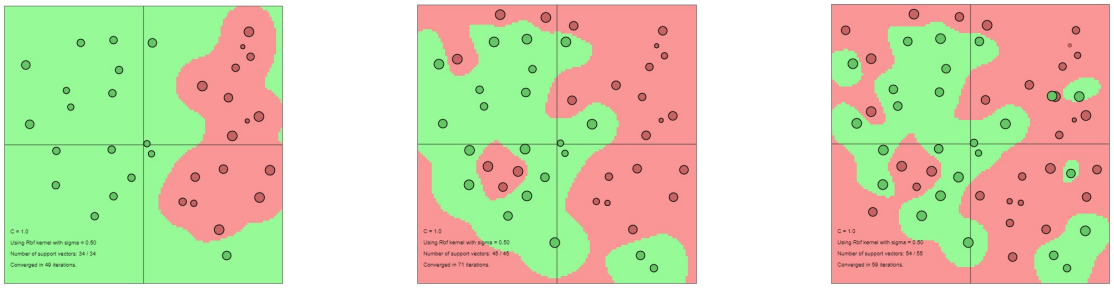
is not much change in the decision boundary even after adding more points on the same side of the class. Both linear and RBF kernels produce same classification output.



(a) Points on right side (b) Points on wrong side (c) More points on wrong side

Figure 1.2: SVM Classification with Linear Kernel

In figures 1.2b and 1.3b few points of one class are added on the wrong sides of hyper plane. In this case, the new points influence the decision boundary and more number of non-zero elements of α_i occur from equation 1.2. Also, the soft margin increases as seen in fig 1.2b. Since, the data points are not linearly separable, the SVM classifier with linear kernel takes more number of support vectors (elements with $\alpha_i \neq 0$) into account while deciding a decision boundary. This accounts for few misclassifications. When it comes to the classifier with RBF kernel in fig 1.3b, it classifies all the data points correctly with no misclassifications.



(a) Points on right side (b) Points on wrong side (c) More points on wrong side

Figure 1.3: SVM Classification with data points on wrong side of a hyperplane

In figures 1.2c and 1.3c more points of one class are added on the wrong sides of the hyper plane. Wrong data points affect the decision boundary to a great extent with linear kernel and the soft margin becomes wide in figure 1.2c. It can be observed in figure 1.2c that even though linear kernel takes almost every point as a support vector, it still fails to give a good decision boundary and a lot of misclassifications occur. This means, changing the support vectors can change the decision boundary. But the classifier with RBF kernel, classifies all the data points correctly with only one misclassification as seen in figure 1.3c.

Try out different values of the regularization hyperparameter C and the kernel parameter sigma. What is the role of the parameters? How do these parameters affect the classification outcome?

The parameter C is responsible for optimization. It tells the SVM classifier about how much misclassification it can allow on a given data set. For some larger value of C, the SVM classifier will give us a hyperplane with smaller margin and classify all the training data points correctly.

Conversely for some smaller value of C, the SVM will give us a hyperplane with wider margin and allows some misclassifications even if the training data is linearly separable.

Sigma determines the reach of a training data point. For a smaller value of sigma, the SVM classifier reaches to the training points that are further away from the margin. This can lead to over-fitting of the model. For larger values of sigma, the model looks for points that are closer and it will be able to generalize better. The effect of these hyper parameters C and sigma is explained in detail in the below question.

Compare classification using the linear kernel with classification using the RBF kernel. Which performs better? Why?

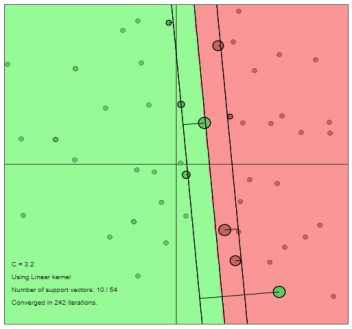
As seen from figures 1.2a and 1.3a, the classifier with RBF and linear kernels gives similar performance as the data points are linearly separable. But from figures 1.2b, 1.3b, 1.2c and 1.3c we can observe that the data is non-linearly separable and RBF out performs the classifier with linear kernel. What RBF does is, it creates a non-linear combination of the data features and project them onto a higher-dimensional space. It later separates the classes with a linear decision boundary(kernel trick) . Training the SVM classifier with linear kernel takes lesser time when compared to RBF kernel. Therefore, linear kernel can do the job for linearly separable data points. But, for non-linearly separable data RBF would be a better choice

What is a support vector? When does a particular datapoint become a support vector? When does the importance of the support vector change? Illustrate visually.

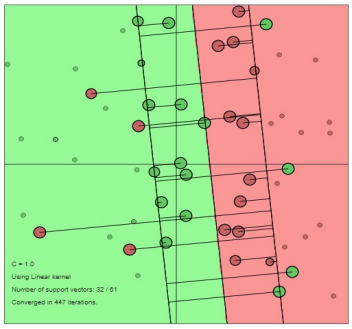
Support vectors are nothing but the training data points which lie on(or inside of) the margin line. SVM tries to find an optimal hyper plane $y(x) = w^T x_i + b$ such that for an unknown value of x, it gives the class of x. Therefore, its goal is to estimate w. As given in equation 1.2 when we solve the Lagrangian dual of SVM, we get $w = \sum_{i=1}^{N} \alpha_i y_i x_i$. Substituting the value of w in $y(x)$ gives the below equation.

$$y(x) = \sum_{i=1}^{N} \alpha_i y_i x_i^T x + b \tag{1.3}$$

Here α_i are the variables of duality. A data point becomes a support vector when the α_i value is non-zero and the point lie on or inside the margin line. Equation 1.3 denotes the sparse representation of SVM. It can be seen from eq 1.3 that our linear classifier is represented using only the points that lie on or inside the margin line. With this we can now transform the SVM into a non-linear classifier using the kernel trick and compute classification hyperplanes in large dimensional spaces as y(x) is now only a dot product of terms. From figure 1.4, it can be seen that in the case of linearly separable points (fig 1.4a), the support vectors are fewer in number and the soft margin has a smaller width. The one misclassified green point is also considered as a support



(a) Linearly Separable Points

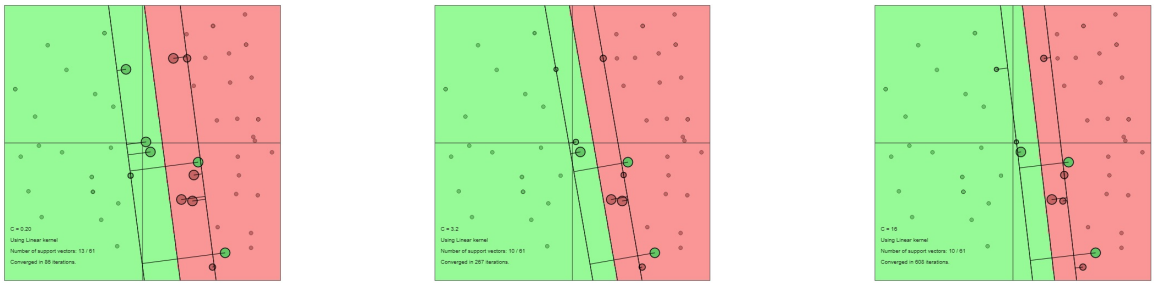


(b) Non-linearly Separable Points

Figure 1.4: Support Vectors during classification

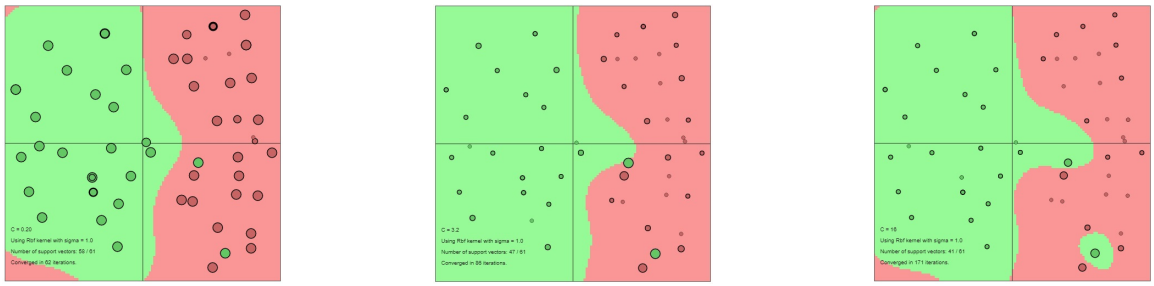
vector. Whereas, in the case of non-linearly separable points (fig 1.4b) every misclassified point is considered as a support vector and SVM takes lot more points in arriving at a decision boundary. Also, the width of the soft margin is huge in figure 1.4a compared to the one in figure 1.4a. Hence, we can say that the support vectors are responsible for the position and orientation of a hyper plane.

What is the role of parameters C and sigma? What happens to the classification boundary if you change these parameters. Illustrate visually.
The regularization parameter C controls the misclassifications done by SVM classifier. The effect of C is illustrated in the following figures.



(a) C = 0.2 (b) C = 3.2 (c) C = 16

Figure 1.5: SVM with Linear Kernel and change in C values



(a) C = 0.2 (b) C = 3.2 (c) C = 16

Figure 1.6: SVM with RBF Kernel and change in C values

From figure 1.5c, it can be seen that for a value of C=16, the decision boundary for a linear kernel has smaller margin and has less number of support vectors. For the same value of C in figure 1.6c, the RBF kernel correctly classifies all the training points. As the C value is reduced to 0.2 and 3.2 in figures 1.5a and 1.5b, the linear kernel uses more support vectors and also accounts to misclassifications. The soft margin becomes wider for lesser values of C. Similarly for the RBF kernel in figures 1.6a and 1.6b, misclassifications occur for smaller value of C. Therefore, from figures 1.5 and 1.6, it can be said that the a larger value of C makes the model over-fit while a smaller value makes it under fit.

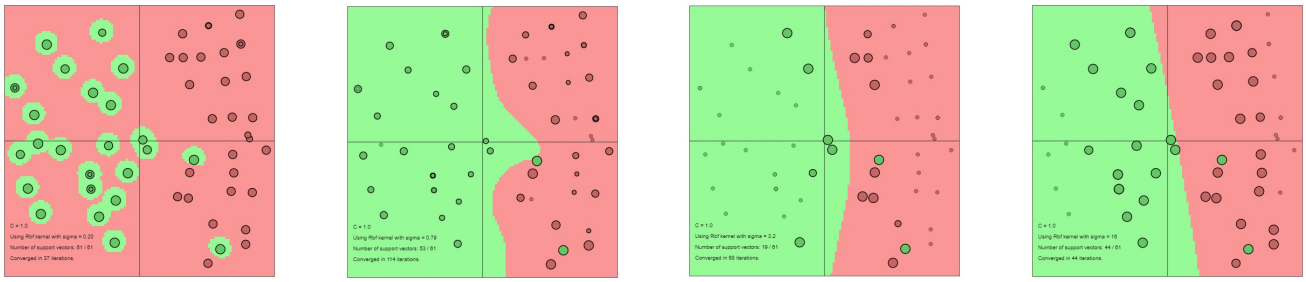
The RBF kernel for SVM classifier in dual space is given as below

$$k(x, x_i) = exp\left(- \frac{||x - x_i||_2^2}{\sigma^2} \right) \tag{1.4}$$

From the above equation 1.4, it can be seen that the value of σ influences the distance between x and x_i. The value of $k(x, x_i)$ tends to zero when σ is much smaller than $||x - x_i||_2^2$. This can

make the classifier reach points that are further away which leads to over fitting. As the value of σ increases, the classifier will look for points that are closer and hence it will generalize better.

In figure 1.7, it can be seen that for a smaller value of $\sigma = 0.2$ the classifier tries to fit all the



(a) Sigma = 0.2 (b) Sigma = 0.79 (c) Sigma = 3.2 (d) Sigma = 16

Figure 1.7: SVM with RBF Kernel and change in sigma values

data points which indicates over fitting. As the value of σ increases to 0.79, 0.32 and 16, the model generalizes better.

What happens to the classification boundary when sigma is taken very large? Why?

As seen from figure 1.7d, for a higher value of $\sigma = 16$ the model generalizes better and gives almost a linear decision boundary. This means that the area of impact of support vectors will include only the support vectors.

1.1.3 Least-Squares Support Vector Machine Classifier:

Influence of hyperparameters and kernel parameters

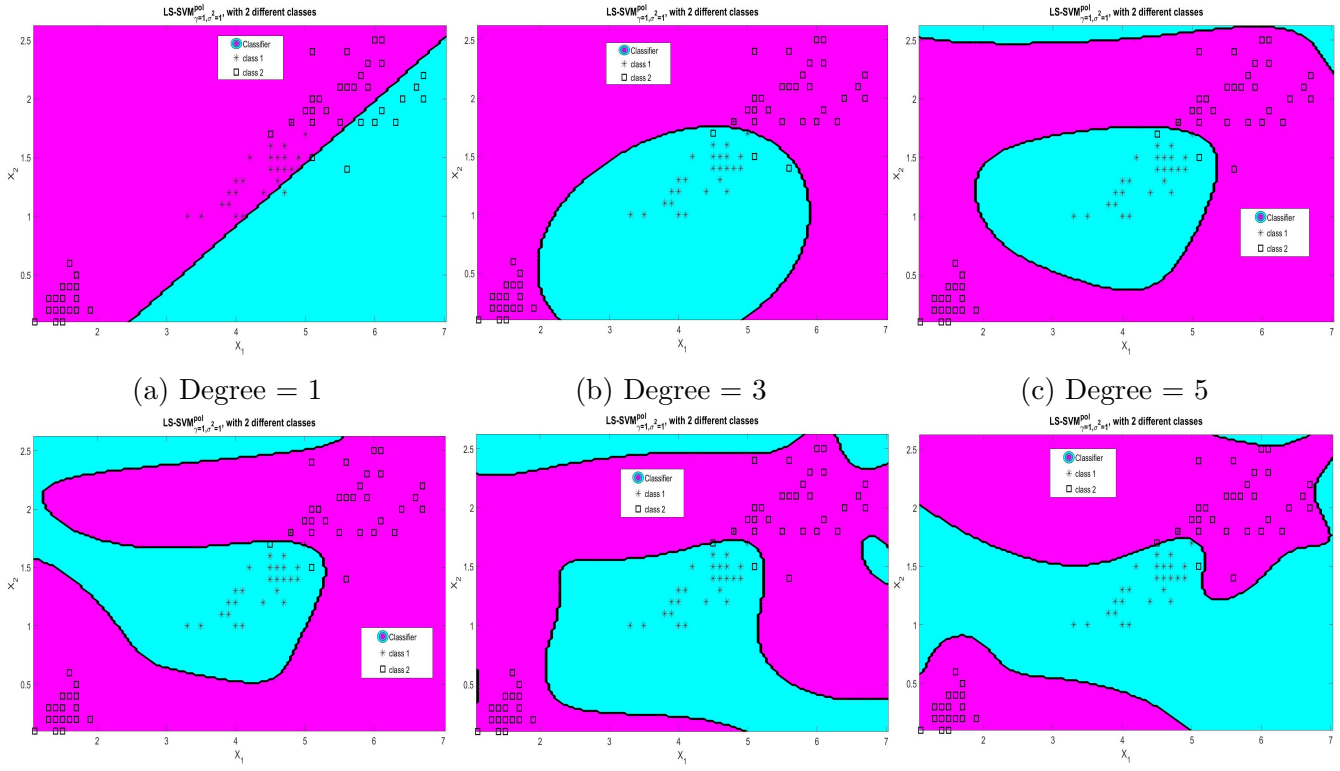
Try out a polynomial kernel with degree = 1; 2; 3; ... and t = 1 (fix gam = 1). Assess the performance on the test set. What happens when you change the degree of the polynomial kernel?

Degree	Misclassifications	Error Rate on test set (%)
1	11	55.00
3	0	0
5	0	0
7	0	0
9	0	0
11	0	0

Table 1.1: Error rate of polynomial kernel on test set

From the figure 1.8 and table 1.1, it can be observed that for degree = 1, the decision boundary is a linear one and a lot of misclassifications occur on the training data points. Even on test set the classifier gives 11 misclassifications with an error rate of 55 %. As the degree value is increased, the model performs better and the number of misclassifications are reduced. The classifier gives exact predictions on the test sets for degree values greater than 1. Also, for the polynomial kernels with degree 9 and 11, the model tries to cover all the training samples with complex decision boundaries which indicates that it is moving towards over-fitting. Therefore, a good range of degree values can be between 3 and 9.

Try out a good range of different sig2 values as kernel parameters (fix gam = 1). Assess the performance on the test set. What is a good range for sigma. Fix a reasonable choice for the sig2 parameter and compare the performance using a range of gam. What is a good range for gam?



(a) Degree = 1 (b) Degree = 3 (c) Degree = 5

(d) Degree = 7 (e) Degree = 9 (f) Degree = 11

Figure 1.8: SVM with polynomial Kernel and change in degree values

As seen from the figure 1.9 and table 1.2, the classifier works good on the test set for a range of σ^2 between 0.1 and 10 when the value of γ is fixed to 1. If σ^2 is increased further, the model performance decreases and gives lot of misclassifications.

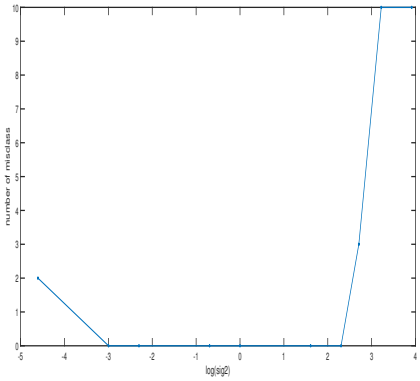


Figure 1.9: $\log(\sigma^2)$ VS misclassifications

Gamma	Sigma	Misclassifications	Error Rate on test set (%)
1	0.01	2	10
1	0.05	0	0
1	0.1	0	0
1	0.5	0	0
1	1	0	0
1	5	0	0
1	10	0	0
1	15	3	15
1	25	10	50
1	50	10	50

Table 1.2: Performance of SVM for various σ^2 values

SVM classifier is sensitive to changes in γ. As, seen from figure 1.10 and table 1.3, the model performs poorly for lower values of γ and gives stable results for higher values. This means that

for γ with very small values, the classifier fails to capture the complexity of the data. For γ with bigger values, even though the model learns complex shapes, it focuses on smoothing the curve rather than minimizing the errors. This could result in poor performance of model without proper fitting of data points. A good range of γ can be between 1 to 5, when σ is fixed to 2.

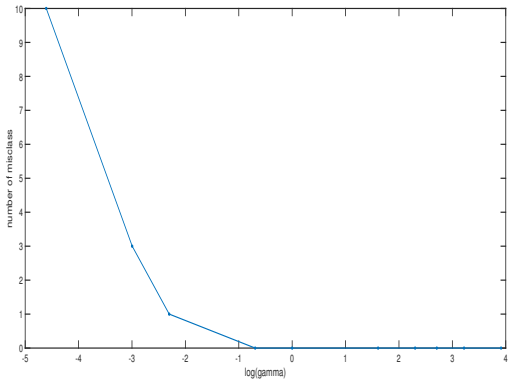


Figure 1.10: log(γ) VS misclassifications

Sigma	Gamma	Mis-classifications	Error Rate on test set (%)
2	0.01	10	50
2	0.05	10	50
2	0.1	3	15
2	0.5	1	5
2	1	0	0
2	5	0	0
2	10	0	0
2	15	0	0
2	25	0	0
2	50	0	0

Table 1.3: Performance of SVM for various γ values

Similar results as described above is seen with the script SampleScript_Iris.m

Tuning parameters using validation

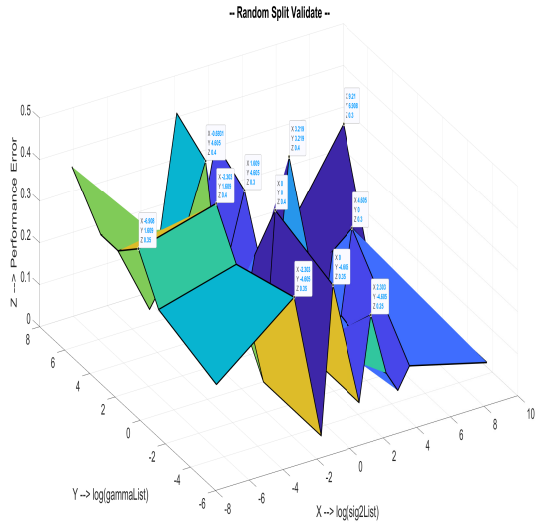
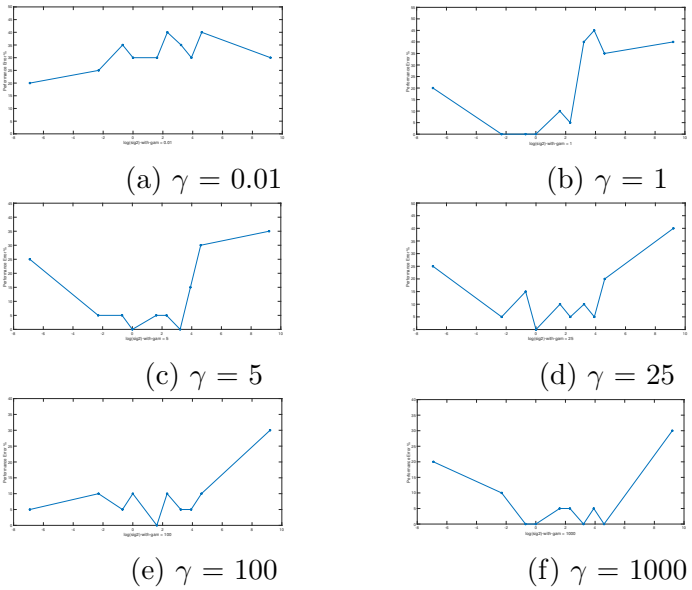


Figure 1.11: Random Split Validation - (log(σ^2) vs log (γ) vs Performane Error)



(a) $\gamma = 0.01$ (b) $\gamma = 1$ (c) $\gamma = 5$ (d) $\gamma = 25$ (e) $\gamma = 100$ (f) $\gamma = 1000$

Figure 1.12: Iris data : Random Split Validation (log(σ^2) values for each level of γ Vs Performance Error) (%) on validation set.

We continue with the Iris data set and tune parameters of γ and σ^2 for a range of values. We use the range of γ as (0.01; 1; 5; 25; 100; 1000) and σ^2 as (0.001; 0.1; 0.5; 1; 5; 10; 25; 50; 100;

10000). Unlike in the previous sections, we now split the training data randomly into 80% and 20% as training and validation sets respectively. The SVM classifier is trained on the training data and the results are shown on the validation set. Figure 1.12 shows the performance error (%) of SVM for a fixed γ and a range of σ^2. Figure 1.11 shows a 3D plot over a range of (γ, σ^2) and performance errors. It is observed from figure 1.12 that the performance error varies between 20 to 40 percent for the given range of (γ, σ^2). Using these results it is difficult to determine a best performing γ and σ values for the random split technique. Another way to determine a good range of (γ, σ^2) is using k-fold cross validation and leave-one-out validation.

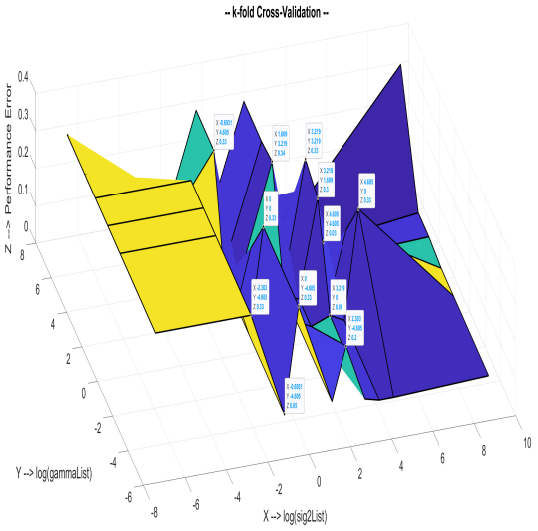
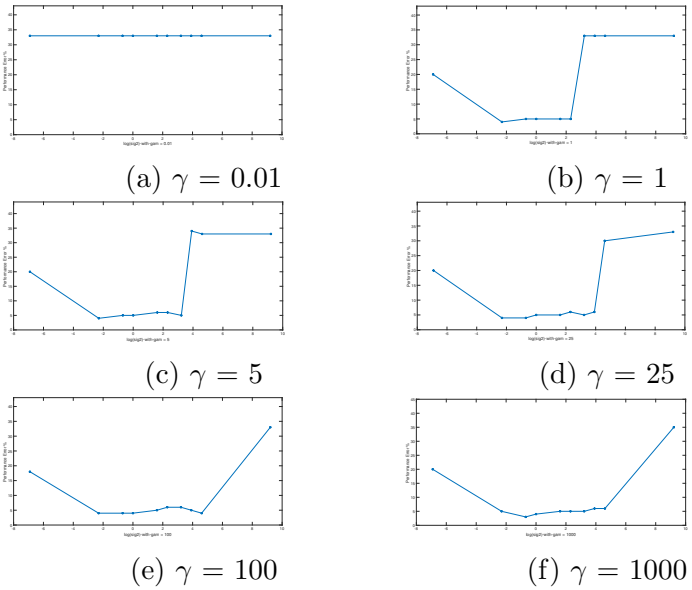


Figure 1.13: k-fold Cross Validation - ($\log(\sigma^2)$ vs $\log(\gamma)$ vs Performane Error)



(a) $\gamma = 0.01$ (b) $\gamma = 1$ (c) $\gamma = 5$ (d) $\gamma = 25$ (e) $\gamma = 100$ (f) $\gamma = 1000$

Figure 1.14: Iris data : k-fold CV ($\log(\sigma^2)$ values for each level of γ Vs Performance Error) (%)

The training data in k-fold cross validation is split into k subsets. Using $k-1$ sets from the k subsets the model is trained and the validated using the k^{th} set. This is repeated over all the k subsets of data in a loop. Next we perform leave-one-out method similar to k-fold cross validation. The result of these methods is not performance rate but rather a average of errors over number of iterations of k-sets. The model performance is shown in figures 1.14 and 1.16. It can be seen that the performance for both k-fold and leave-one-out methods are nearly same. Therefore, in case of larger data sets, k-fold method is preferred as it is computationally faster than leave-one-out and for smaller data sets leave-one-out is preferred. From figure 1.11, it can be seen that random split validation method exhibits high variance compared to k-fold and leave-one-out methods in figures 1.13 and 1.15 giving a clear indication of best (purple color) and least performing (pale yellow and cyan) γ and σ^2 values. This can also be seen from the smoothness of curves in the case of k-fold and leave-one-out.

The optimal values of (γ, σ^2) pair can be chosen from figures 1.14 and 1.16 that gives the least performance error. Some good combinations of (γ, σ^2) can be as shown in table 1.4.

γ	σ^2
25	(0.1, 0.5, 1, 5)
100	(0.1, 0.5, 1, 5)
1000	(0.5, 1, 5)

Table 1.4: Optimal (γ, σ^2) combinations

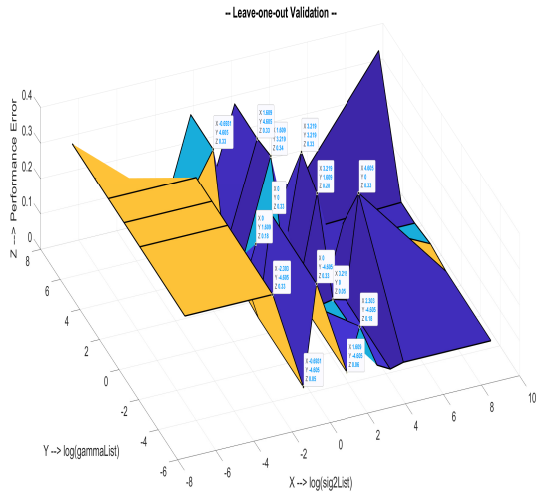
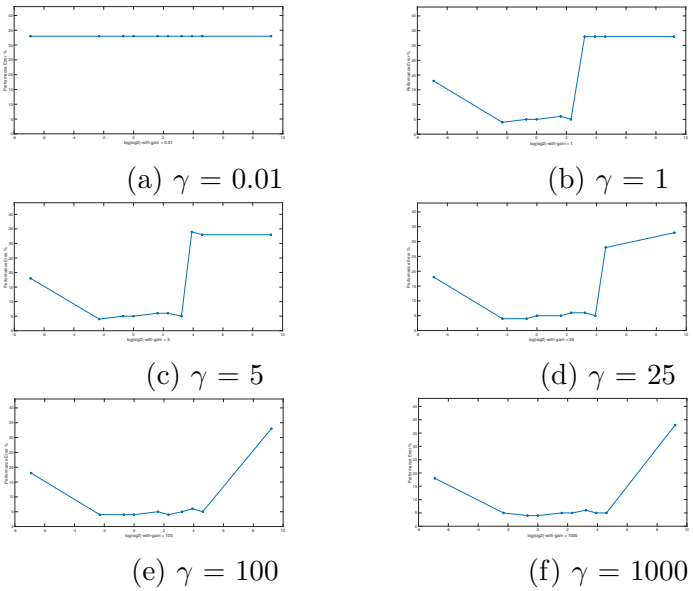


Figure 1.15: Leave-one-out Validation - ($\log(\sigma^2)$ vs log (γ) vs Performane Error)



(a) $\gamma = 0.01$ (b) $\gamma = 1$ (c) $\gamma = 5$ (d) $\gamma = 25$ (e) $\gamma = 100$ (f) $\gamma = 1000$

Figure 1.16: Iris data : LOOCV ($\log(\sigma^2)$ values for each level of γ Vs Performance Error) (%).

Why should one prefer crossvalidation over simple validation (random split)? How to choose the value of k in k-fold crossvalidation?

In a cross validation method the model is trained continuously k times over k-1 subsets of data every time. By doing this the overall error of model is averaged over the errors for each iteration. The overall variance of the end result is small when compared with random split validation. This can be seen in figures 1.11 and 1.13. The value for k is chosen in a such way that the data samples in each set of k splits can statistically represent the original dataset. The choice of k influences the bias-variance trade-off. Factors like size of the data set, computational complexity, etc should also be taken into account while choosing k. For smaller data sets, a higher value of k is chosen so that the training sets have sufficient data samples. For larger data sets, a value of 10 can be chosen. In practice a value of 5 or 10 is chosen as optimal value as these values have proven to give unbiased results.

Automatic parameter tuning

Try out the different 'algorithm'. What differences do you observe? Why do the obtained hyperparameters differ a lot in different runs? What about the cost? Computational speed? Explain the results.

Simplex Algorithm			Grid Search Algorithm		
γ	σ^2	cost	γ	σ^2	cost
3.54	0.018991	0.04	18.165	0.026866	0.04
23.508	0.34958	0.04	133.44	0.024016	0.02
0.4226	0.18471	0.04	98.508	63.077	0.04
6.0573	0.19643	0.04	3944.3	0.32205	0.03
106.55	0.085258	0.03	0.2905	0.020474	0.03
12.893	0.21712	0.04	11.129	0.01182	0.03
382.45	0.28002	0.03	0.36295	0.23467	0.04
1.7285	2.6992	0.04	0.046299	0.41124	0.04

Table 1.5: Optimal (γ, σ^2) pairs using auto tuning methods

We train the SVM model again on Iris data set using tunelssvm for 8 iterations. The function trainlssvm performs 10-fold cross validation on training data and returns optimal values for γ and σ^2. In previous section of tuning hyper-parameters,

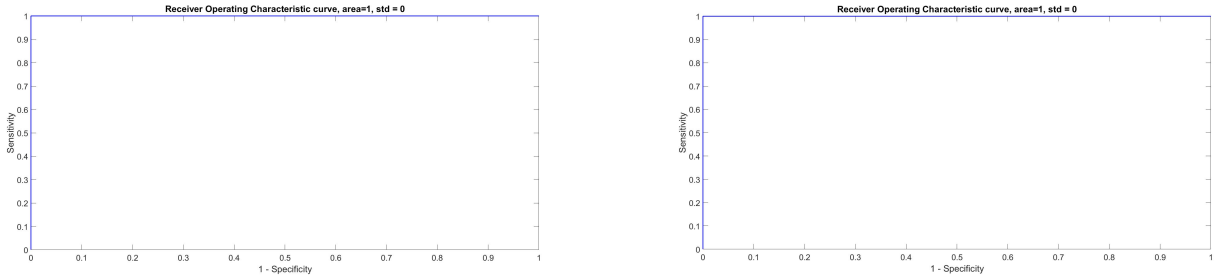
we saw that for a specific value of γ a range of σ works better (shown in table 1.4). How can we decide which pair of (γ, σ^2) is optimal. This is where the optimization algorithms simplex and grid search comes. They solve the convex-optimization problem for (γ,σ^2) and arrive at a local optima. We calculate the average time taken for each algorithm to give optimal values of (γ,σ^2).
Time taken to train SVM for 8 iterations using simplex algorithm is **3.058253 secs**.
Time taken to train SVM for 8 iterations using grid search algorithm is **6.016854 secs**.
We notice that simplex algorithm takes lesser time compared to grid search. This is because simplex tries to converge to the nearest local optima by initializing some random values to γ and σ^2 whereas grid search evaluates over a range of parameters and then looks for the local optima.
From table 1.5 it can be seen that the cost for all (γ,σ^2) pairs is around 0.02 to 0.04 which indicates that these pairs are optimal. As seen earlier in tuning hyper-parameters, the results show that many different (γ,σ^2) optimal pairs are possible. Therefore, the difference in (γ,σ^2) pairs for each run in auto tuning method is expected.

Using ROC Curves

In practice, we compute the ROC curve on the test set, rather than on the training set. Why?

ROC curves is one of the evaluation methods for the performance of a classifier. The classifier model is built using the training data. Evaluating the model for classification accuracy using training set will only give a biased estimate as the model has already seen the data samples. Therefore, the proper way to evaluate the classifier is using unseen data samples i.e., using test sets.

Generate the ROC curve for the iris.mat dataset (use tuned gam and sig2 values). Interpret the result.



(a) ROC Curve with (γ,σ^2)=(6.0573,0.19643) obtained from auto tuning of hyper-parameters using simplex algorithm

(b) ROC Curve with (γ,σ^2)=(18.165,0.026866) obtained from auto tuning of hyper-parameters using grid search algorithm

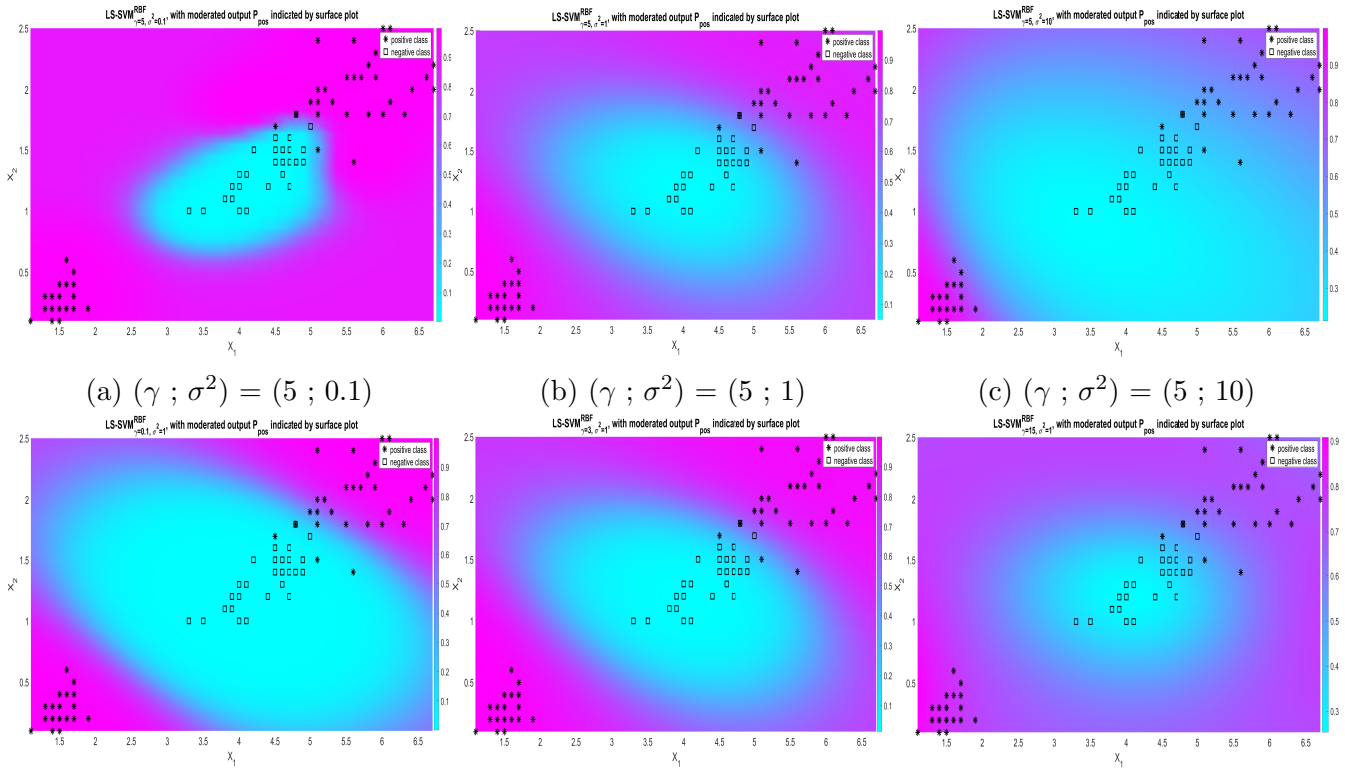
Figure 1.17: ROC Curves for SVM classifier with optimal (γ,σ^2) pairs

The ROC curves in the figure 1.17 tested on IRIS data set resembles an ideal classifier. Therefore, our SVM model built with optimal parameters of (γ,σ^2) pair is a perfect model.

Bayesian framework

How do you interpret the colors of the plot? Change the values of gam and sig2. Visualize and discuss the influence of the parameters on the figure

The Bayesian framework unlike other methods of SVM cannot give a strong decision boundary. Rather it gives the probability of an observation belonging to a target class based on prior and posterior probabilities of the data distribution. From the figures in fig 1.18, it can be observed that the bayesian method gives two colors for each class (cyan - negative and magenta - positive) that slowly fade into one other indicating a separation of the two classes without a decision boundary. The colors here denote the probability of finding a positive class. As seen from the values of color bars next to each plot, it can be understood that magenta indicates the high probability of data points of positive class whereas cyan indicates the least probability of finding a positive class and vice versa. To get the results as show in figure 1.18, the bayes framework is run first with a fixed value of γ and varying value of σ^2 and then with a fixed value of σ^2 and varying value of γ.



(a) $(\gamma\ ;\ \sigma^2) = (5\ ;\ 0.1)$ (b) $(\gamma\ ;\ \sigma^2) = (5\ ;\ 1)$ (c) $(\gamma\ ;\ \sigma^2) = (5\ ;\ 10)$

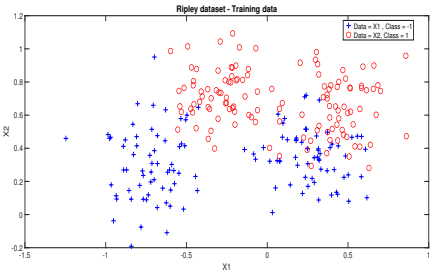
(d) $(\gamma\ ;\ \sigma^2) = (0.1\ ;\ 1)$ (e) $(\gamma\ ;\ \sigma^2) = (3\ ;\ 1)$ (f) $(\gamma\ ;\ \sigma^2) = (15\ ;\ 1)$

Figure 1.18: Bayesian Framework for various $(\gamma\ ;\ \sigma^2)$ values

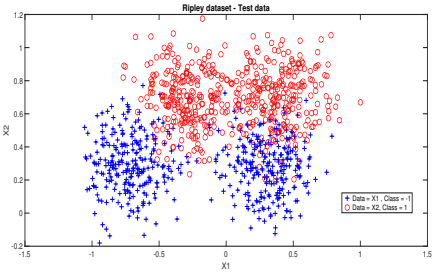
The plots 1.18a, 1.18b. 1.18c is obtained by fixing the γ value to 5 and varying the σ^2 values to (0.1; 1 ; 10). The plots 1.18d, 1.18e. 1.18f is obtained by fixing the σ^2 value to 1 and varying the γ values to (0.1; 3 ; 15). It can be seen from the plots that for a higher value of γ and lower value of σ^2, the model is able to accurately classify the data points of two classes. When σ^2 becomes greater than γ ($\sigma^2 \geq 2\gamma$) in fig 1.18c and 1.18d the cyan region becomes more dominant than magenta indicating a uncertainty in model. This leads to a lot of misclassifications. If the σ^2 values is made too small, the classification region can become too small trying to fit all the data points causing over fitting. Therefore, from these observations it can be said that it is preferred to have a balance between the values of $(\gamma\ ;\ \sigma^2)$ having a σ^2 value lesser than γ for better performance of the model.

1.2 Homework Problems

1.2.1 Ripley Dataset



(a) Training samples of ripley dataset

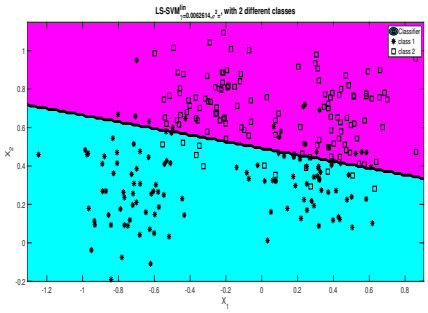


(b) Test samples of ripley dataset

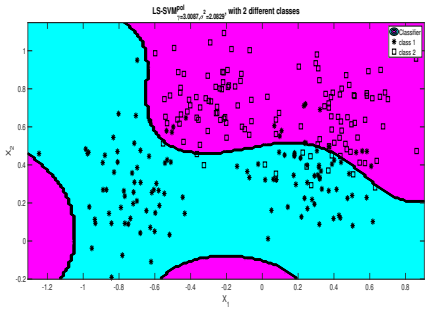
Figure 1.19: Visualizing training and test samples of ripley data

The data set contains 250 training samples and 1000 test samples with only two features. Therefore, it is easy to visualize on a 2D plot. From the plot 1.19 it can be seen that the data not completely linearly separable as there are some overlaps between the two classes. So, a non-linear decision boundary can separate these two classes better than a linear one. However, we train the SVM for linear , polynomial, RBF kernels and evaluate the models using ROC curves. The optimal hyper parameters of (γ, σ^2) for training is obtained using auto tuning method with simplex algorithm as seen in the earlier sections.

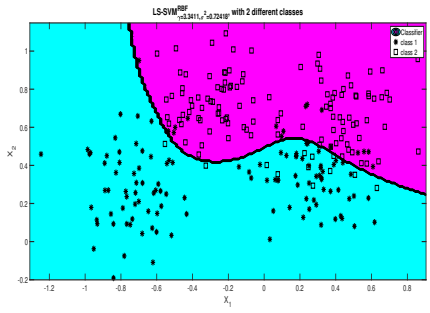
As seen from the figures 1.20b, 1.20c the classifier with polynomial and RBF kernel performs in a similar way allowing only few misclassifications whereas the classifier with linear kernel gives few more misclassifications. In addition to this, the classifier is trained for few iterations with different optimal hyper parameters generated by *tunelssvm* function and some of their results are evaluated and tabulated in 1.6. The results in table 1.6 and figures 1.21a, 1.21b, 1.21c show that, the classifier with RBF kernel gives slightly better accuracy (96.86 % from ROC curve) than polynomial or linear kernels.



(a) SVM with linear kernel- γ = 0.0062614



(b) SVM with pol kernel- γ = 3.0087 and degree = 5

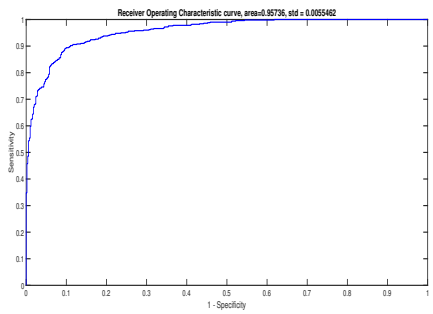


(c) SVM with RBF kernel- (γ, σ^2) =(3.3411, 0.72418)

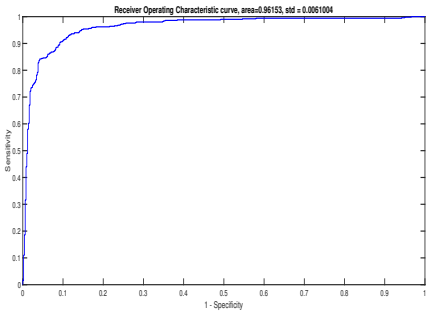
Figure 1.20: SVM Classifier trained on Ripley dataset

Linear Kernel			Polynomial Kernel			RBF Kernel		
γ	σ^2	Error	γ	σ^2	Error	γ	σ^2	Error
0.00291	-	10.6	0.12909	1.109	10.2	0.59161	0.28572	9.3
0.01678	-	10.5	3.0087	2.0829	10.1	8.5381	0.60447	9.5

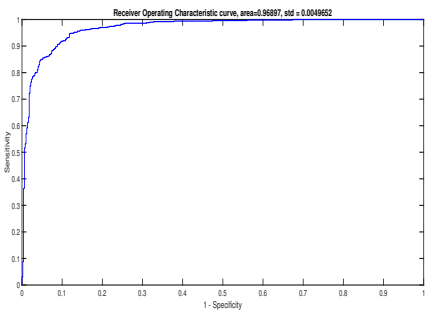
Table 1.6: SVM classifier on Ripley dataset - Performance results with different kernels for auto tuned hyper parameters



(a) ROC curve of SVM classifier with polynomial kernel



(b) ROC curve of SVM classifier with polynomial kernel

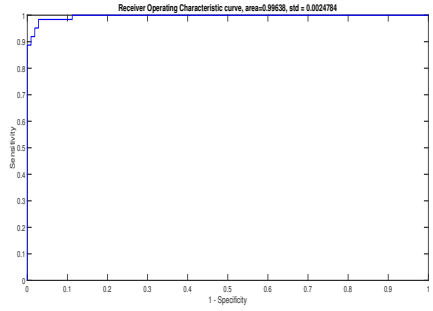


(c) ROC curve of SVM classifier with RBF kernel

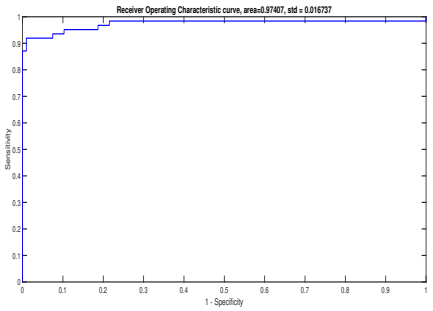
Figure 1.21: ROC curves to evaluate SVM Classifier trained on Ripley dataset

1.2.2 Wisconsin Breast Cancer dataset

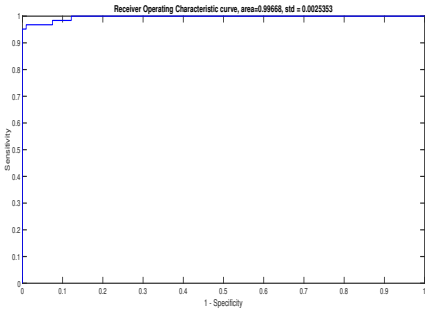
This data set contains 400 training samples and 169 test samples. There are 30 features in this dataset and is difficult to visualize on either 2D or 3D plots. As seen from the ROC curves in figure 1.22, the SVM classifier with linear and RBF kernels gives an accuracy greater than 99%. This could be because of the reason that both the classes are well separated without many overlaps and the model is able to classify the test samples to the right class. Similar to the previous ripley data, the model is trained for few iterations with different optimal hyper parameters generated by *tunelssvm* function and some of their results are evaluated and tabulated in 1.7. It is observed that the model performance remains more or less the same for change in hyper parameters in the case of linear and RBF kernels. For polynomial kernel, the model performs better for a degree of 3 than 5



(a) ROC curve of SVM classifier with polynomial kernel



(b) ROC curve of SVM classifier with polynomial kernel



(c) ROC curve of SVM classifier with RBF kernel

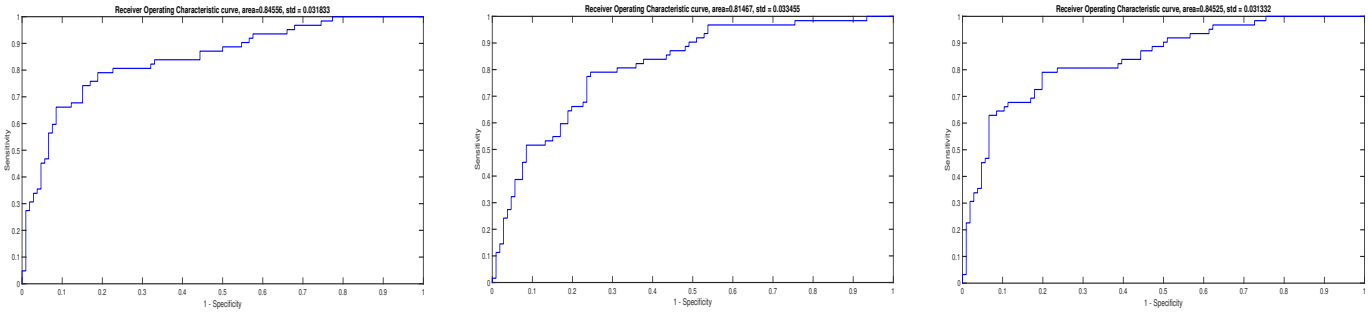
Figure 1.22: ROC curves to evaluate SVM Classifier trained on breast cancer dataset

Linear Kernel			Polynomial Kernel			RBF Kernel		
γ	σ^2	Error	γ	degree	Error	γ	σ^2	Error
0.07253	-	4.7337	1.533e-05	3	4.7337	03.261	37.617	2.3669
8.2338	-	4.7337	2.0799e-07	3	3.5503	22.922	37.755	1.7751
0.47591	-	4.142	182.19	5	15.976	168.12	44.754	4.142

Table 1.7: SVM classifier on Breast Cancer dataset - Performance results with different kernels for auto tuned hyper parameters

1.2.3 Diabetes dataset

This data set contains 300 training samples and 168 test samples. There are 8 features in this dataset and is difficult to visualize on either 2D or 3D plots. As seen from the ROC curves in figure 1.22, the SVM classifier with polynomial kernel(fig 1.23b) performs with lesser accuracy(81.4 %) compared with linear(fig 1.23a) and RBF kernels (1.23c) which gives an accuracy of 84%. To investigate this performance accuracy of the model, two of the features were plotted and it is observed that the two classes have over lapping samples as shown in figure 1.24. Therefore, the model finds it difficult to generate a good decision boundary that separates these two classes with greater accuracy. Similar to the two previous data sets, the model is trained for few iterations with different optimal hyper parameters generated by *tunelssvm* function and some of their results are evaluated and tabulated in 1.8. It is observed that the RBF kernel performs better and the model performances of all three models remain more or less the same despite the change in hyper parameters.



(a) ROC curve of SVM classifier with polynomial kernel

(b) ROC curve of SVM classifier with polynomial kernel

(c) ROC curve of SVM classifier with RBF kernel

Figure 1.23: ROC curves to evaluate SVM Classifier trained on diabetes dataset

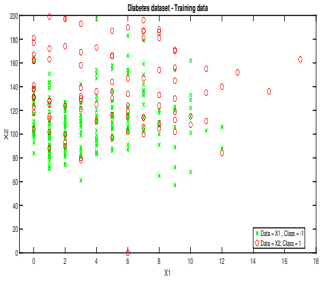


Figure 1.24: Diabetes dataset

Linear Kernel		Polynomial Kernel			RBF Kernel		
γ	Error	γ	degree	Error	γ	σ^2	Error
0.0136	25	0.0020	3	29.167	70.514	253.93	21.429
0.0176	23.214	1.2677	3	37.5	191.73	94556.	22.619
0.0090	25	0.0942	3	32.143	4.149	255.93	22.619

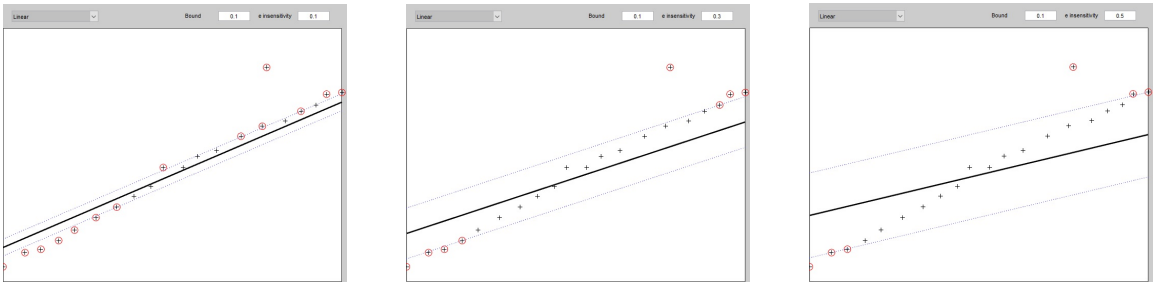
Table 1.8: SVM classifier on Diabetes dataset - Performance results with different kernels for auto tuned hyper parameters

2. Exercise 2: Function Estimation and Time Series Prediction

2.1 Support vector machine for function estimation

Construct a dataset where a linear kernel is better than any other kernel (around 20 data points). What is the influence of e (try small values such as 0:10; 0:25; 0:50;..) and of Bound (try larger increments such as 0:01; 0:10; 1; 10; 100). Where does the sparsity property come in?

The parameter ϵ controls the number of support vectors that the model takes to build the regression function. This in-turn determines width of the margin to fit the training data. Lower the value of ϵ, higher the number of support vectors taken by that the model and vice versa. If ϵ values keeps increasing, the regressor fails to fit all the training data. This can be observed from the figures 2.1a, 2.1b and 2.1c for a fixed bound value. On the other hand, we fix the value of ϵ and explore the regressor's performance for various values of bound. From figures 2.1d, 2.1e and 2.1fm it can seen that the model performs poorly for lower value of bound and fits all the training data for higher value of bound. Further increasing the bound value can cause the model to over-fit. It can be said that the bound is a trade-off between the error and model confidence. Sparsity comes into the picture when the model estimates better with less data points on unseen data. There won't be any sparseness in the model when it uses all the data points as support vectors. It is only possible for some optimal bound and ϵ say (0.15 and 0.1) as seen from figure 2.1e.



(a) $\epsilon = 0.1$, Bound = 0.1 (b) $\epsilon = 0.3$, Bound = 0.1 (c) $\epsilon = 0.5$, Bound = 0.1

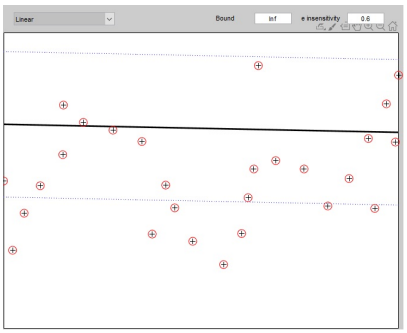
(d) $\epsilon = 0.15$, Bound = 0.01 (e) $\epsilon = 0.15$, Bound = 0.1 (f) $\epsilon = 0.15$, Bound = 10

Figure 2.1: SVM for Regression using Linear Kernel: demo `uiregress`. All the figures have same data points and performance of the regressor for different values of ϵ and bound are observed

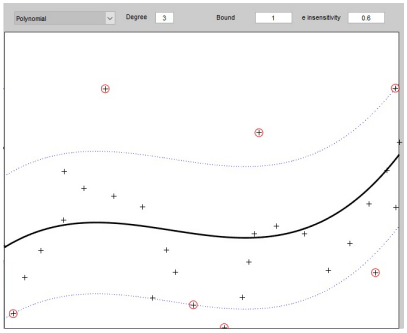
Construct a more challenging dataset (around 20 data points). Which kernel is best suited for your dataset? Motivate why.

Figure 2.2 shows the performance of the SVM regressor with different kernels on a challenging dataset for fixed values of ϵ and bound. Linear kernel as seen in figure 2.2a doesn't perform well. Polynomial kernels as seen from figures 2.2b and 2.2c, for degree = 3 doesn't perform well but for degree = 5 generalizes better. If the polynomial degree is increase the model over-fits. Figures 2.2d,

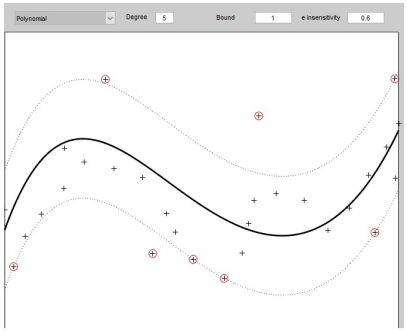
2.2e and 2.2f shows the regressor with RBF kernel for various σ^2 values. For very lower values of σ^2, the model over-fits but for higher values the model gives a linear estimate. Therefore, for a value of $\sigma^2 = 0.1$, the model generalizes better. Overall the regressor with RBF kernel performs betters as it learns complex patterns of data better compared to a polynomial kernel.



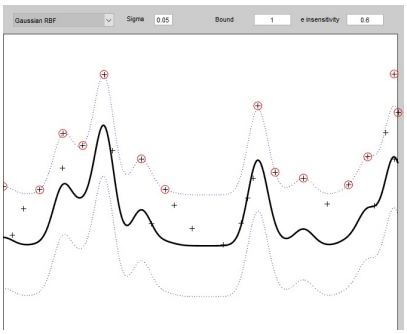
(a) Linear : $\epsilon = 0.6$, Bound = inf



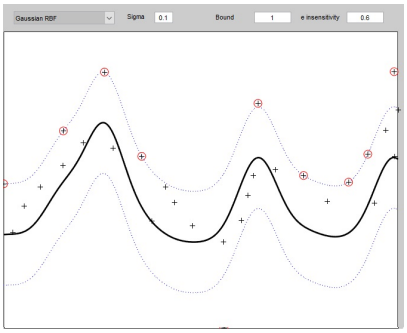
(b) Poly : degree =3, $\epsilon = 1$, Bound = 0.6



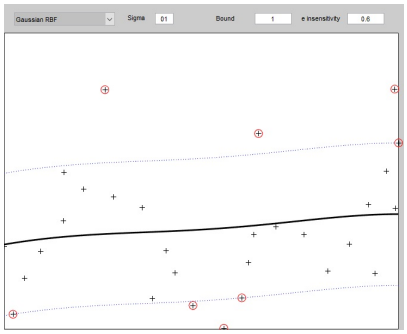
(c) Poly : degree =5, $\epsilon = 1$, Bound = 0.6



(d) RBF : $\sigma^2 = 0.05$ $\epsilon = 0.15$, Bound = 0.01



(e) RBF : $\sigma^2 = 0.1$ $\epsilon = 0.15$, Bound = 0.01



(f) RBF : $\sigma^2 = 1$ $\epsilon = 0.15$, Bound = 0.01

Figure 2.2: SVM for Regression using Linear, Polynomial and RBF Kernels: demo `uiregress`.

In what respect is SVM regression different from a classical least squares fit?

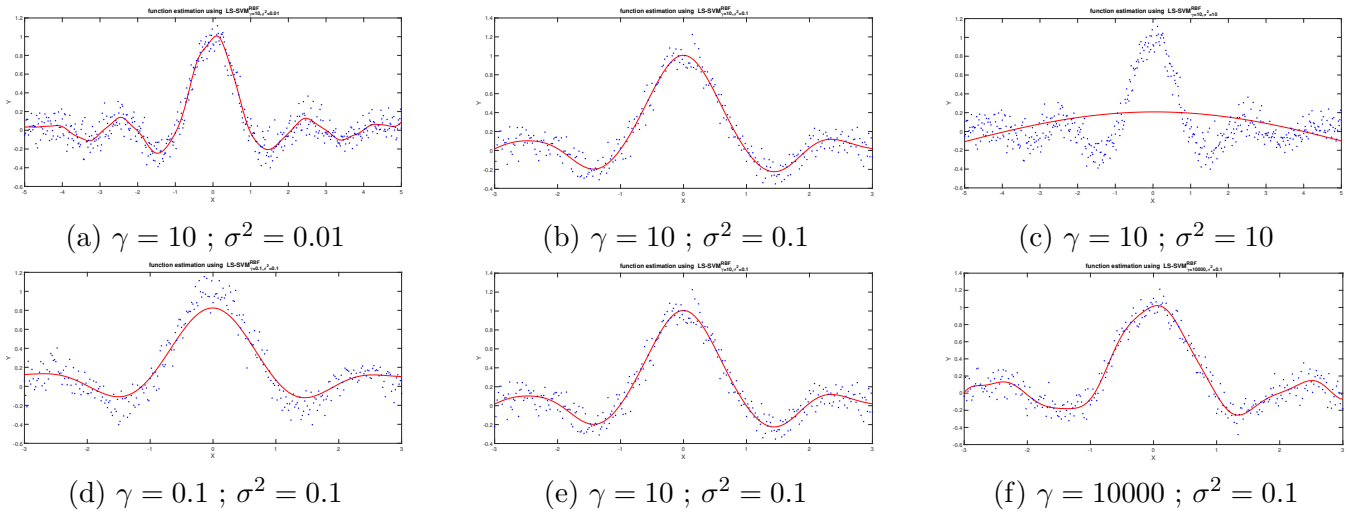
The difference in SVM regression and classical least squares is seen in loss function. Classical least squares use a L2 (squared) loss where as SVR uses a L1 loss with a threshold ϵ. The observations outside this ϵ threshold is considered as points contributing to error.

2.2 The sinc function

2.2.1 Regression of the sinc function

Try out a range of different gam and sig2 parameter values (e.g., gam = 10; 10^3; 10^6 and sig2 = 0:01; 1; 100) and visualize the resulting function estimation on the test set data points. Discuss the resulting function estimation. Report the mean squared error for every combination (γ, σ^2).

The SVM regressor model is trained using *trainlssvm* function for various combinations of (γ : σ^2) values and the results are plotted in figure 2.3. Also, MSE between the estimated and true results for every (γ ; σ^2) pairs are tabulated in the table 2.1. From the figure and table it can be seen that the model performs well for (γ ; σ^2) values (1 ; 0.01), (1 ; 0.1), (10 ; 0.1). For larger values of σ^2, the model under performs on the dataset and gives poor estimates whereas for very low values it tries to overfit the data points. Therefore, a optimal pair of (γ, σ^2) should be such that $\gamma >> \sigma^2$ to get a small MSE between the ground truth and model estimates for the data points.



(a) $\gamma = 10$; $\sigma^2 = 0.01$ (b) $\gamma = 10$; $\sigma^2 = 0.1$ (c) $\gamma = 10$; $\sigma^2 = 10$

(d) $\gamma = 0.1$; $\sigma^2 = 0.1$ (e) $\gamma = 10$; $\sigma^2 = 0.1$ (f) $\gamma = 10000$; $\sigma^2 = 0.1$

Figure 2.3: SVM Regressor : Function estimation for various (γ; σ^2) values of the RBF kernel

Mean Squared Error (MSE) (%)						
(γ ; σ^2)	$\sigma^2 = 0.01$	$\sigma^2 = 0.1$	$\sigma^2 = 1$	$\sigma^2 = 10$	$\sigma^2 = 10^2$	$\sigma^2 = 10^3$
$\gamma = 0.1$	2.2498	1.5692	8.3871	12.988	13.631	13.641
$\gamma = 1$	1.1784	1.1024	5.1397	11.392	13.553	13.639
$\gamma = 10$	1.2326	1.121	3.261	10.967	12.937	13.631
$\gamma = 10^2$	1.2621	1.1367	1.481	10.526	11.474	13.552
$\gamma = 10^3$	1.2758	1.1503	1.2254	8.2999	11.205	12.932
$\gamma = 10^4$	1.2791	1.1612	1.1065	6.659	11.201	11.483

Table 2.1: MSE (%) between SVR estimations and ground truth data for various (γ ; σ^2)

Do you think there is one optimal pair of hyperparameters?

As seen from figure 2.3 and table 2.1, it can be inferred that the model performs better and have least MSE for lower values of σ^2 such as 0.01 and 0.1 for γ in range of 1 to 10^4). One such optimal pair can be (γ, σ^2) = (1 ; 0.1) and (10;0.1)

Tune the gam and sig2 parameters using the tunelssvm procedure. Use multiple runs: what can you say about the hyperparameters and the results? Use both the simplex and gridsearch algorithms and report differences.

Simplex Algorithm			Grid Search Algorithm		
γ	σ^2	cost	γ	σ^2	cost
44.681	0.36072	0.0099951	222.33	0.44703	0.0098997
390.57	0.48974	0.010855	21.733	0.33129	0.010041
46.424	0.38045	0.010003	44.117	0.3663	0.0099097
76.299	0.39569	0.010843	23.237	0.34383	0.0099472
57.125	0.39352	0.0099651	16.346	0.31191	0.010876

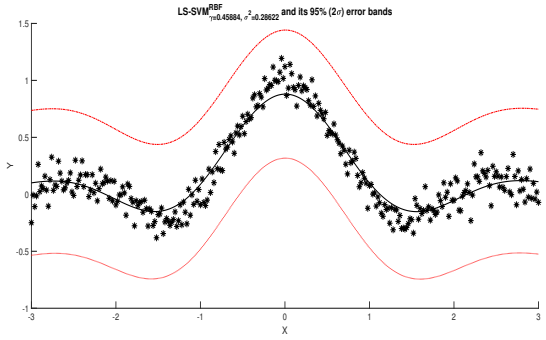
Table 2.2: Optimal (γ, σ^2) pairs using auto tuning methods

From table 2.2 it can seen that both the algorithms produce different optimal (γ;σ^2) for each run but the error approximately remains the same. Also, a lot of variability is seen only in γ values whereas σ^2 has much lesser

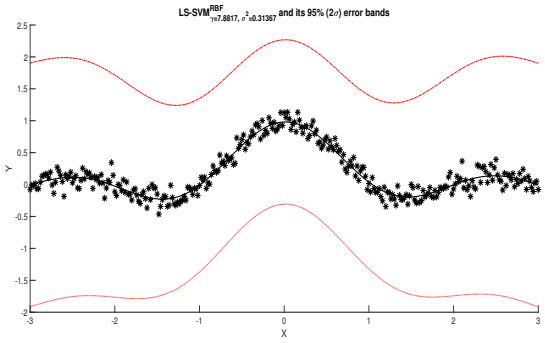
variance. Overall, it can be said that $\gamma >> \sigma^2$. Also, simplex algorithm executes faster with 0.85 secs compared to grid search method which takes 1.2 sec approximately.

2.2.2 Application of the Bayesian framework

Discuss in a schematic way how parameter tuning works using the Bayesian framework. Illustrate this scheme by interpreting the function calls denoted above.



(a) Bayes Framework: LS-SVM with (γ,σ^2)=(10,0.4)



(b) Bayes Framework: LS-SVM with (γ,σ^2) obtained from auto tune method

Figure 2.4: Bayesian Framework : LS-SVM

In this section we analyze the function estimation in Bayesian Framework. We first initialize (γ,σ^2) values as (10,0.4). We obtain the corresponding posterior parameters as (0.41251, 0.26488) and MSE as 0.1957. Instead of these random (γ;σ^2), we obtain the optimal hyperparameter values using *tunelssvm* with simplex algorithm as (25.6337 ; 0.3815). With these settings we get the posterior parameters as (7.8817; 0.31367) and MSE 0.0423 after bayesian optimization. The corresponding plots are given in figure 2.4.

The optimization in bayesian learning happens in three levels. We assign a probability over all the values of the weight vector w instead of single point of the vector space. For this we need to a prior $p(w)$ and a likelihood probabilities $p(D|w)$ over the data distribution D. The posterior probability distribution can be obtained from prior and likelihood as $P(w|D) = \frac{P(D|w)P(w)}{P(D}$.

Level 1: The main objective here is to minimize the weight space parameters w and b. The LS-SVM also includes the regularization hyperparameters μ and ζ in its objective function as shown in the below equation 2.1. Here the optimization is done for w and the bias b.

$$\min_{w,b,e_c} J_P(w, e_c) = \mu \frac{1}{2} w^T w + \zeta \sum_{k=1}^{N} e_{c,k}^2 \tag{2.1}$$

In the above equation the first term corresponds to prior distribution and the second term corresponds to the likelihood. The posterior distribution and the model H_σ is given by:

$$p(w, b|D, \mu, \zeta, H_\sigma) = \frac{p(D|w, b, \mu, \zeta, H_\sigma)}{p(D|\mu, \zeta, H_\sigma)} p(w, b|\mu, \zeta, H_\sigma) \tag{2.2}$$

Minimization of equation 2.1 for μ, w and ζ gives the maximization of equation 2.2.

Level 2: The regularization parameters μ and ζ are optimized in second level.

Level 3: In level three the optimization is done for the kernel parameters. It can used for model selection.

Therefore, as shown in figure 2.4b, it results in a better performance for the bayesian framework when tuned with optimal hyperparameters and optimization as MSE given by that model is very less.

2.3 Automatic Relevance Determination

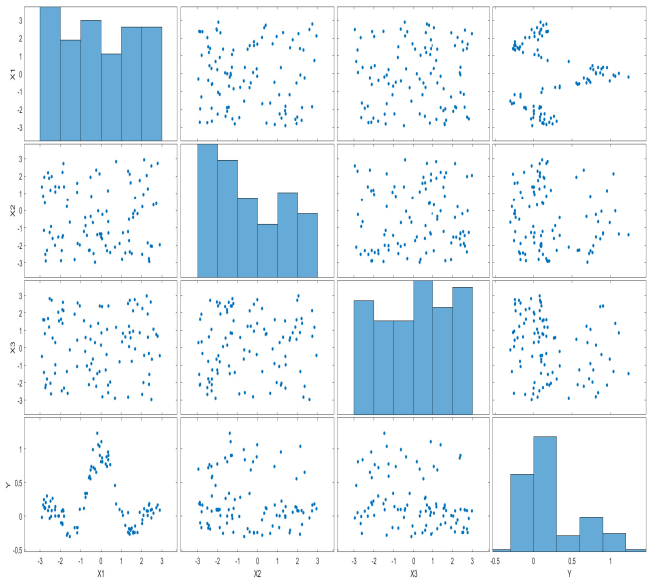
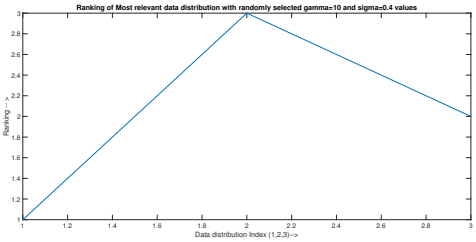
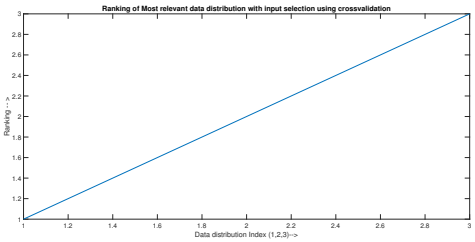


Figure 2.5: Data distribution



(a) Bayes ARD ranking for (γ;σ^2)=(10,0.4)



(b) Bayes ARD ranking for (γ;σ^2) selected from cross validation

Figure 2.6: Bayes : ARD

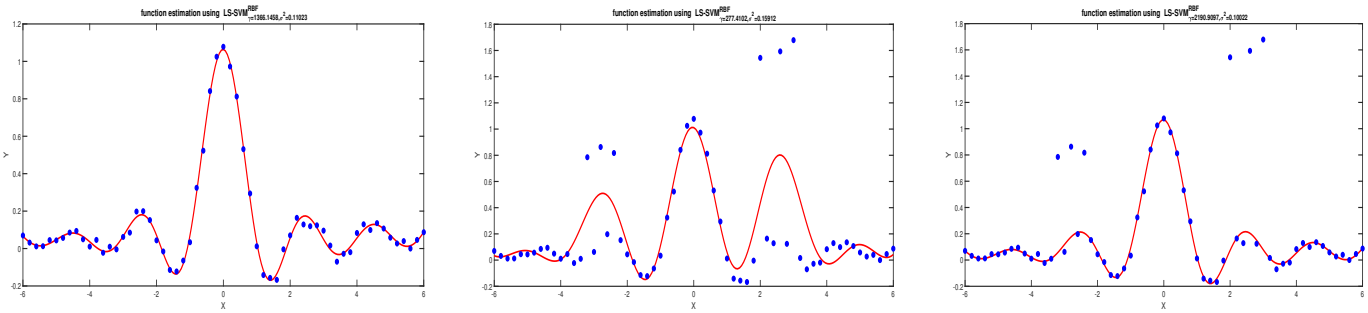
The Automatic Relevance Distribution ranks the best input data distribution that fits the target by estimating the hyper parameters. To illustrate this, we build three random data distributions X1, X2 and X3.The target label Y is built from X1. The result of figure 2.6a, is plotted for the model built by randomly choosing (γ;σ^2) pair. In figure 2.6b, the model is built by selecting the inputs using cross validation. As seen from figures 2.5 and 2.6, the ARD algorithm in both cases gives the data X1 and the most relevant one which is obvious as Y corresponds to X1. Similar results are seen when changing the target Y corresponding to X2 and X3 distributions.

2.4 Robust regression

Visualize and discuss the results. Compare the non-robust version with the robust version. Do you spot any differences?

The figure 2.7 shows robust vs non-robust LS-SVM estimations for data points with and without outliers. The non-robust regressor with outliers fail to perform well as the estimations try to take the outliers also into account. This results in a poor fit over the data points as seen in figure 2.7b. On the other hand, the robust regressor with cross validation gives good performance with better function estimations ignoring the outliers as seen in figure 2.7c

Why in this case is the mean absolute error ('mae') preferred over the classical mean squared error ('mse')?



(a) LS-SVM Regressor : Without Outliers

(b) LS-SVM Regressor : With Outliers

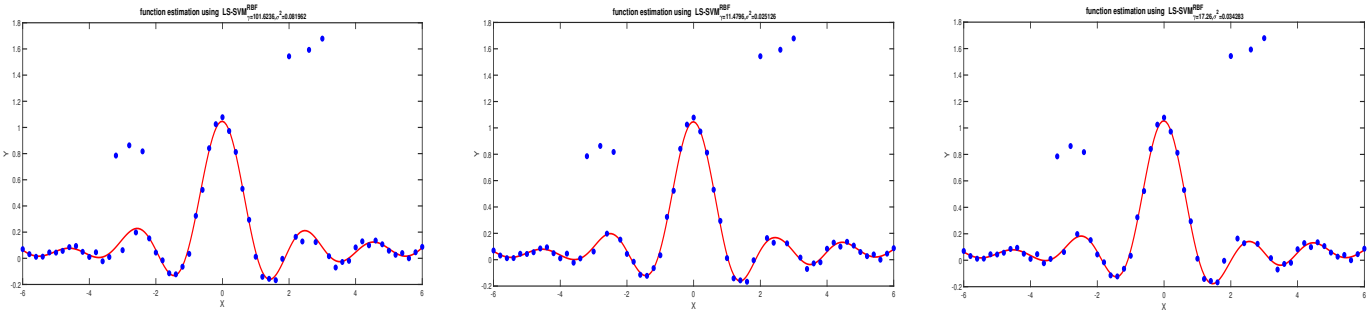
(c) LS-SVM Regressor : Robust Cross Validation

Figure 2.7: LS-SVM : Robust Vs Non-Robust versions

MAE is the measure of average magnitude of errors between true and estimated data points. Whereas MSE is the measure of squared average magnitude of errors. Since, MSE is the squared error, it adds more weight to outliers where the errors are high.This makes the model perform poorly when trying to minimize the errors. This is not the case with MAE as it only takes the absolute average of errors and that makes it more robust to outliers.

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \qquad MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

Try alternatives to the weighting function wFun (e.g., 'whampel', 'wlogistic' and 'wmyriad'. Report on differences. Check the user's guide of LS-SVMlab for more information.



(a) Robust LS-SVM Regressor : Logistic

(b) Robust LS-SVM Regressor : Myriad

(c) Robust LS-SVM Regressor : Hampel

Figure 2.8: LS-SVM : Robust versions with different weighting functions

The estimations of robust LS-SVM for various weighting functions are plotted in figure 2.8 and the results are given in the table 2.3. From the table 2.3, it can be seen that Hampel is faster as it achieves the convergence with only 4 iterations and Huber is the slowest. But, as seen from cost, the overall performance of all the weighting functions remain same. Every model gives approximately same estimations over the training data ignoring the outliers.

Weighting Function	γ	σ^2	Iterations for convergence	Cost
wHuber	12190.9097	0.10022	112	1.368683e-01
wLogistic	101.6236	0.081962	62	1.392045e-01
wMyriad	11.4796	0.025126	13	1.339679e-01
wHampel	17.26	0.034283	4	1.360631e-01

Table 2.3: LS-SVM Robust Regressor with different weighting functions and their optimal hyper parameters (γ;σ^2)

2.5 Homework Problems

2.5.1 Logmap dataset

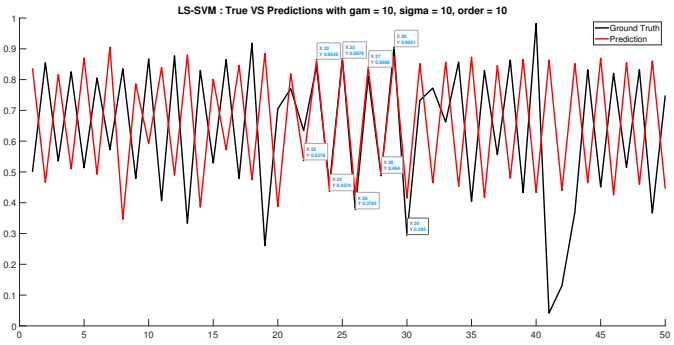
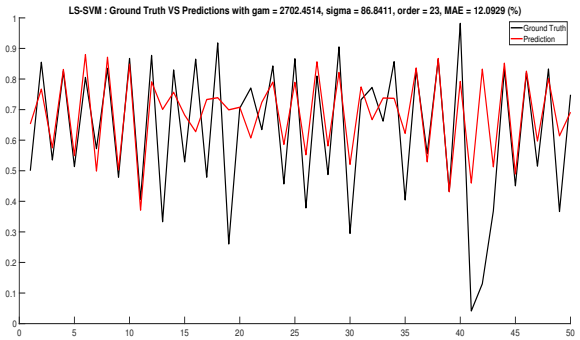
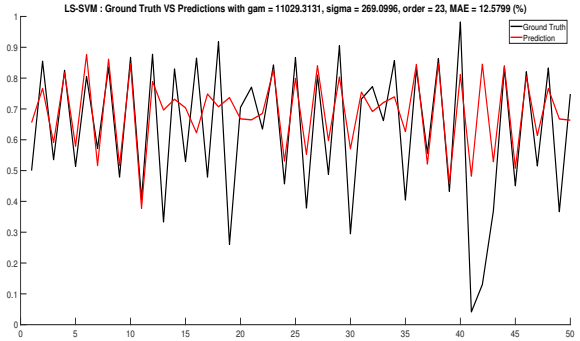


Figure 2.9: LS-SVM: Logmap Dataset with (γ;σ^2)=(10;10)

In this section, we follow the instructions given in the exercise to obtain a model fit on the logmap dataset. From figure 2.9 it can be noticed that for initial (γ;σ^2) values of (10;10), the model estimates are very poor. The predictions fit the ground truth data only between the range 22 to 30. To get better model performance we use auto tune methods and cross validation method to obtain the optimal hyper parameters (γ;σ^2). For the lag parameter, we try out a range from 1 to 50 with each of the obtained optimal (γ;σ^2) pairs. The best fit obtained from the model is plotted in the figure 2.10. The optimal values of (γ;σ^2) and the lag is chosen by calculating MAE of the model on the test set for which the MAE $\leq$ 15. The model estimates are not completely perfect as it fails to fit at the huge peaks of the data set. The plots shown in figure 2.10 are the ones with the least MAE values.



(a) LogMap: LS-SVM Regressor with optimal (γ;σ^2)=(2702.4514 ; 86.8411), lag = 23 and MAE = 12.09(%)



(b) LogMap : LS-SVM Regressor with optimal (γ;σ^2)=(11029.3131 ; 269.0996), lag = 23 and MAE = 12.5799(%)

Figure 2.10: LS-SVM: Logmap Dataset with optimally tuned hyper parameters

2.5.2 Santa Fe dataset

Does order = 50 for the utilized auto-reressive model sounds like a good choice?

The data set contains 1000 training and 200 test points. In time-series prediction, the future values are predicted based on the past values during the function estimation. This is known as a auto-regressive model. The prder/lag in this model determines the numbers of samples/period away from which the prediction is to be made which means the prediction starts after lag number of points. With the setting of lag = 50, the model gives a decent performance initially but later, a small shift is seen between the actual and predicted data set in figure 2.11. The order 50 seems like a good choice, but the model performance can still be improved by reducing its value.

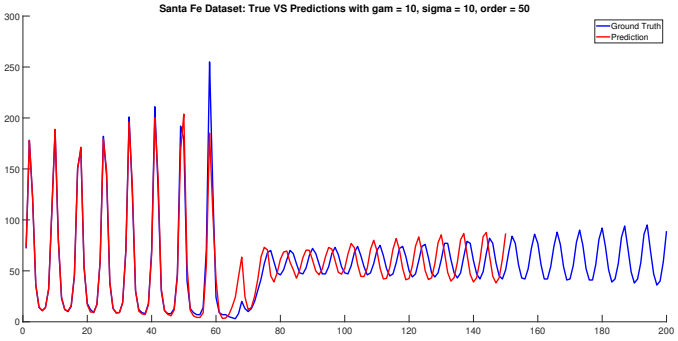


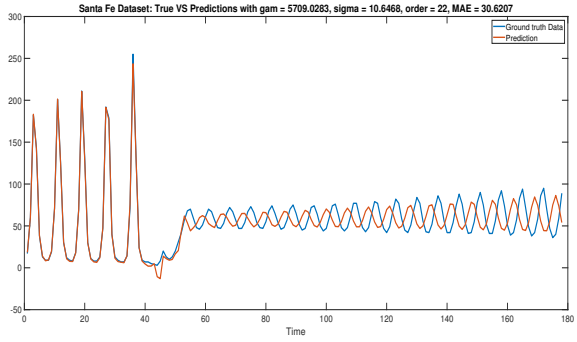
Figure 2.11: Santa Fe Dataset: $(\gamma;\sigma^2)$=(10,10) and Lag = 50

Would it be sensible to use the performance of this recurrent prediction on the validation set to optimize hyperparameters and the model order?

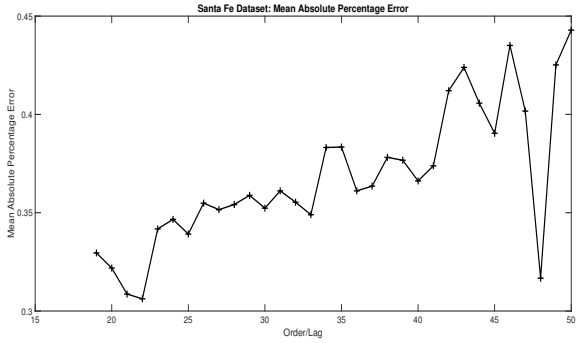
Yes, we can use the performance of this model to optimize the hyper parameters. We can divide the training set of 1000 points into say 700 training and 300 validation points. The validation set can be used for fine tuning the model parameters so that the model can perform better on the test set.

Tune the parameters (order, gam and sig2) and do time series prediction.

The model is tuned with optimal hyper parameters obtained from *tunelssvm* over a range of order/lag values 1-50. MAE values are calculated for the corresponding order and $(\gamma;\sigma^2)$ pairs. The plot 2.12a shows the model with least MAE value which is obtained for a lag of 22. It can be seen that the model gives good estimate for the larger peaks of the dataset and poor estimates in the later part. Figure 2.12b shows MAE's calculated over the range of order/lag.



(a) Santa Fe Dataset: $(\gamma;\sigma^2)$ = (5709.0283 ; 10.6468) and MAE = 30.6207(%)



(b) Santa Fe Dataset : MAPE for optimal $(\gamma;\sigma^2)$ over a range of order/lag values

Figure 2.12: Santa Fe Dataset: Best Fit with optimal hyper parameters and corresponding MAPE for the tuned $(\gamma;\sigma^2)$ pairs

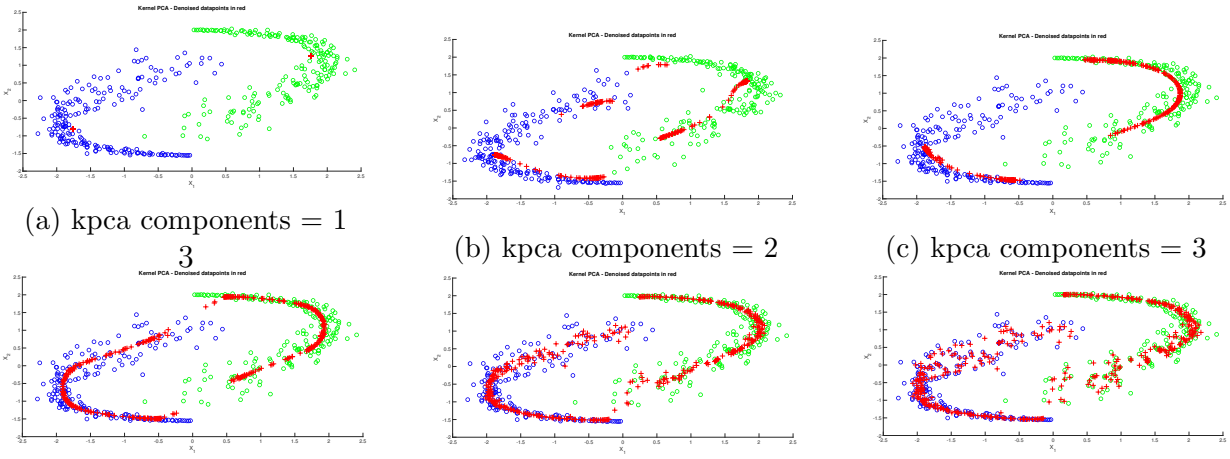
3. Unsupervised Learning and Large Scale Problems

3.1 Exercises

3.1.1 Kernel principal component analysis

Describe how you can do denoising using PCA. Describe what happens with the denoising if you increase the number of principal components.

PCA is a technique that transforms and maps the observations to a lower dimension space such that the greatest variance by some scalar projection of the data lies on the first coordinate (called the first principal component), the second greatest variance on the second coordinate, and so on. The projections or the principal components is meant to have only the relevant features of the data. Therefore, PCA can be used for the purpose of denoising where the components are expected to have only the data features and not the noise. PCA does a linear orthogonal transformation and therefore cannot handle complex or non-linear data patterns. This motivates us to use a kernel PCA which can handle non-linear features. In the provided kpca script we use an example dataset with 400 points and noise has been added. To denoise the original data from the noisy one, we use (1;2;3;7;11;15) principal components. The model performance is shown in figure 3.1. It can be observed from the figure 3.1 that as the number of principal components increases, more information is retained during the denoising and it becomes close to the actual data.



(a) kpca components = 1 (b) kpca components = 2 (c) kpca components = 3

(d) kpca components = 7 (e) kpca components = 11 (f) kpca components = 15

Figure 3.1: Kernal PCA: Denoising the dataset with different PCA components

Compare linear PCA with kernel PCA. What are the main differences? How many principal components can you obtain?

A linear PCA always needs linear principal components to map the data to lower dimensions. But our data set in non-linear and when linear PCA is used for denoising this data set, the principal components cannot retrieve all the information. This lead to huge information loss in linear PCA but it is not he case with kernal PCA. Linear PCA finds new projects based on covariance matrix of original variables. If the features are N dimensional then linear PCA can extract a maximum of N eigen values whereas kernel PCA can extract as many eigenvalues as the number of data points/observations. For linear PCA the number of principal components that can be obtained are equal to the feature dimensions whereas for kernel pca it is equal to the number of data points/observations.

For the dataset at hand, propose a technique to tune the number of components, the hyperparameter and the kernel parameters.

The kernel hyper parameters like σ^2 can be tuned using cross-validation techniques. The number of principal components can be chosen by plotting the eigen vlaues divided by their cumulative sum over the number of eigen values. From the plot, the principle component can be selected looking at the region where the graph shows diminishing values. Other way could be by selecting a value from a range of values that gives less denoising error over few iterations.

3.1.2 Spectral Clustering

Spectral clustering is a graph based technique with improved performance than k-means methods. It involves three main steps: A similarity graph is constructed between the N objects that are to be clustered ; Grouping more similar features based on the first k eigenvectors computations; Finally applying k-means method on these features to separate objects into k classes.

What are the differences between spectral clustering and classification?

Spectral clustering is an unsupervised technique where the data samples with similar features are grouped into a cluster. Classification is a supervised learning method, where the model learns to associate the data features to a class label. On unseen datasets, spectral clustering can only say if the data point belongs to a cluster whereas classification can say to which class the data point belongs to.

What is the influence of the sig2 parameter on the clustering results?

We run the spectral clustering algorithm for various σ^2 values (0.001;0.005;0.01;0.02;0.5;1) and some of the plots are given in the figure 3.2. It can be noticed that for smaller values of σ^2, the results are the best as there is good separation between the two rings as well as the 2nd and 3rd eigen vectors projections on the subspace. For $\sigma^2 > 0.5$, the rings overlaps and the clustering gives poor results. Also, the projections of 2nd and 3rd eigen vectors gets overlapped.

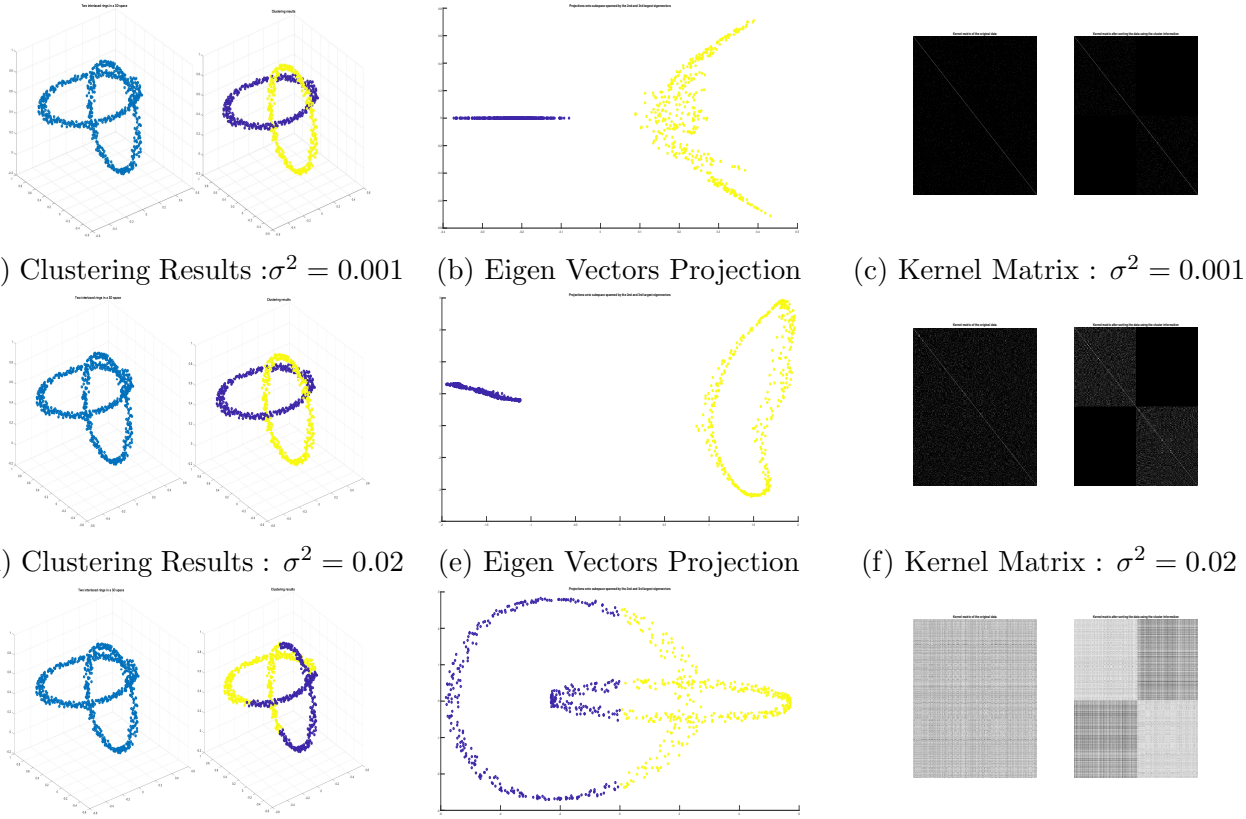
3.1.3 Fixed-size LS-SVM

Solving a model in primal space would be useful for large datasets whereas dual space would be suitable for large dimensional inputs.

What is the effect of the chosen kernel parameter sig2 on the resulting fixed-size subset of data points (see fixedsize script1.m)? Can you intuitively describe to what subset the algorithm converges?

The function *kentropy* uses Quadratic Renyi Entropy in the selection of the subset. The algorithm selects the subset when it converges to the subset with maximum entropy and the subset gives a better representation of the original data. The algorithm is run with 6 different hyper parameters of σ^2 (0.001 ; 0.01 ; 0.1 ; 1 ; 10 ; 100). Some of the results are plotted in the figure 3.3. For lower values of σ^2, the chosen support vectors are very close and as the σ^2 increases, the support vectors spread out evenly away from the data point. From the figure 3.3d, it can be seen that all the support vectors are mostly outliers. For $\sigma^2 = 0.1$ and 1, the chosen candidates have better entropy and a gives a good representation of the data set.

Run fslssvm script.m. Use the Wisconsin Breast Cancer dataset for this exercise. Compare the results of fixed-size LS-SVM to l_0-approximation in terms of test errors, number of support vectors and computational time.

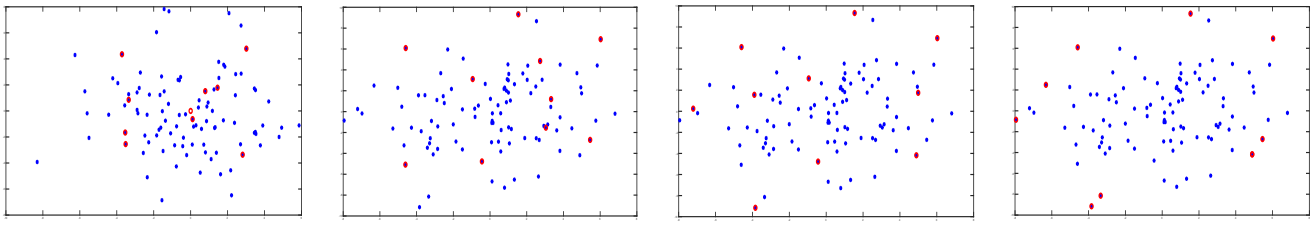


(a) Clustering Results :$\sigma^2 = 0.001$ (b) Eigen Vectors Projection (c) Kernel Matrix : $\sigma^2 = 0.001$

(d) Clustering Results : $\sigma^2 = 0.02$ (e) Eigen Vectors Projection (f) Kernel Matrix : $\sigma^2 = 0.02$

(g) Clustering Results : $\sigma^2 = 1$ (h) Eigen Vectors Projection (i) Kernel Matrix : $\sigma^2 = 1$

Figure 3.2: Spectral Clustering Results; $(\sigma^2) = (0.001;0.02;1)$



(a) $\sigma^2 = 0.001$ (b) $\sigma^2 = 0.1$ (c) $\sigma^2 = 1$ (d) $\sigma^2 = 100$

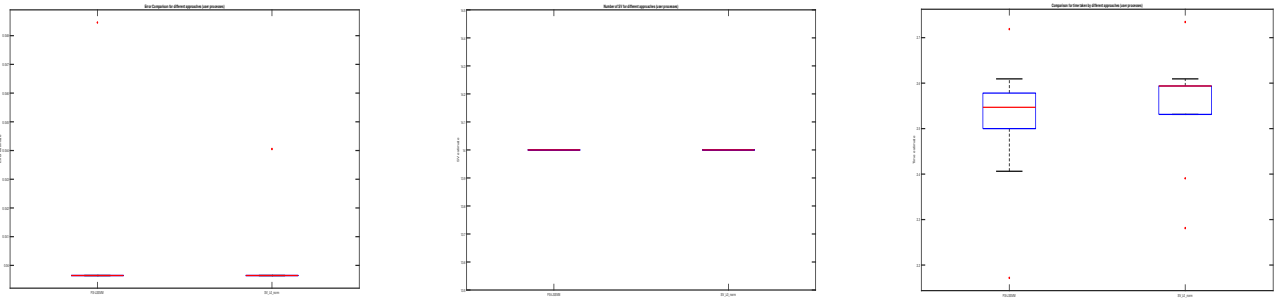
Figure 3.3: Fixed-size LS-SVM for a two dimensional problem using RBF kernel

In this exercise we compare the results of FS-LSSVM with l_0 approximation using breast cancer dataset. The error estimates for both the methods remains same without any variability as seen from the figure 3.4a. The number of support vectors also remain the same. From figure 3.4c, it can seen that FS-LSSVM takes lesser computations compared to l_0 approximation, but FS-LSSVM has higher variability than l_0.

3.2 Homework Problems

3.2.1 Kernel principal component analysis

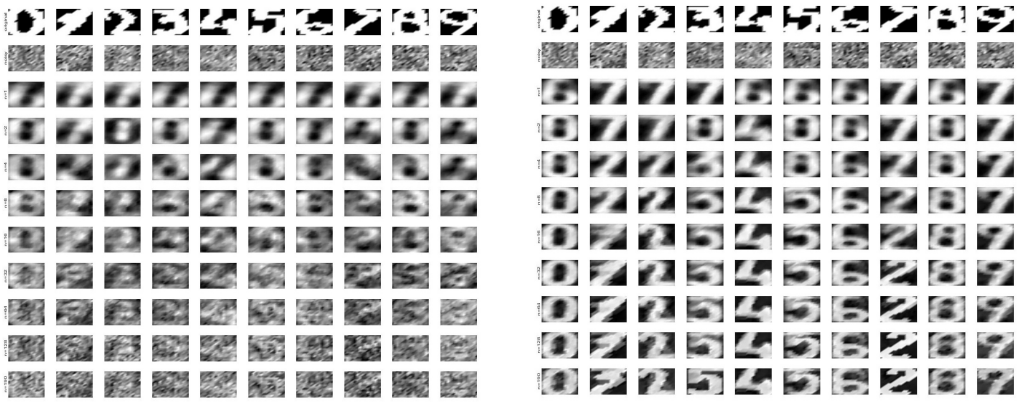
We run the provided script for denoising the digit images with a noise level of 1.0 and the results are plotted in the figure 3.5 for the models with linear and RBF kernels. It can be seen that the model with RBF kernel does a good job in denoising the noisy images. For number of PC's = 16



(a) Error estimates (b) Number of Support Vectors (c) Computation Time

Figure 3.4: Breast Cancer Dataset; Fixed Size - LSSVM vs l_0 - type approximation

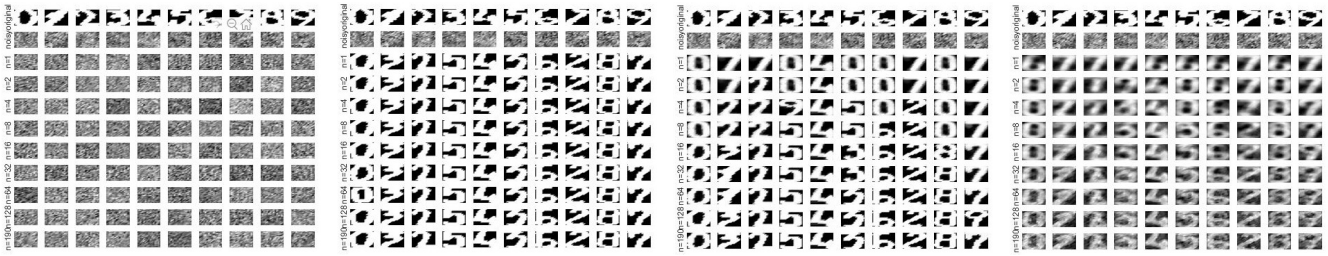
and above, the model denoises the original images from noisy ones except for the the digit 7 where it decodes it as digit 2. Linear kernel on the other hand does a very poor job for higher principal components and the denoised images are more or less noisy. In the script the default settings has



(a) Noise Level: 1.0 - Linear Kernel (b) Noise Level: 1.0 - RBF Kernel

Figure 3.5: Digit denoising using RBF and Linear Kernels

a σ^2 value of 35. To analyze the effect of change in σ^2 we run the algorithm with multiples values such as (0.001;0.01;0.1;1;10;100) and some of the results are shown in the figure 3.6. From the figure it can be seen that for values 0.1 and 1, the denoising of images from noisy data is clear with only few wrong images (digit 3, 5, 9). As the σ^2 increases, the denoised images tend to be more nosier as the reconstruction error is high and the output resembles the one similar to a linear kernel.



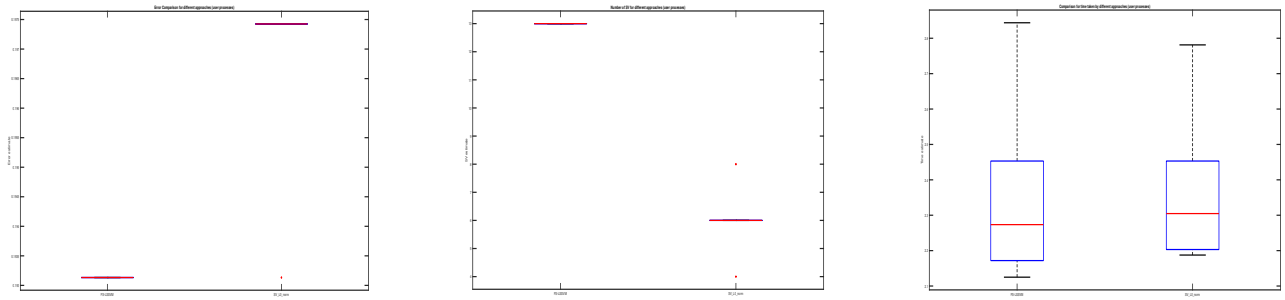
(a) $\sigma^2 = 0.001$ (b) $\sigma^2 = 0.1$ (c) $\sigma^2 = 1$ (d) $\sigma^2 = 100$

Figure 3.6: Digit Denoising with RBF kernel for various σ^2 values

3.2.2 Fixed-size LS-SVM

Shuttle (statlog)

The shuttle log dataset contains 700 data points with 9 features and 1 class. Fixed-size LS-SVM algorithm is run on this dataset and the results are plotted in the below figure 3.7. The error estimates is almost same for both the methods but Fixed-size LSSVM is has slight lower error by a value of 0.003. FS-LSSVM takes around 13 support vectors whereas l_0 -type method takes only 6 about one-third of FS-LSSVM method. Regarding the computational complexity, FS-LSSVM is faster and can be seen from figure 3.7c.

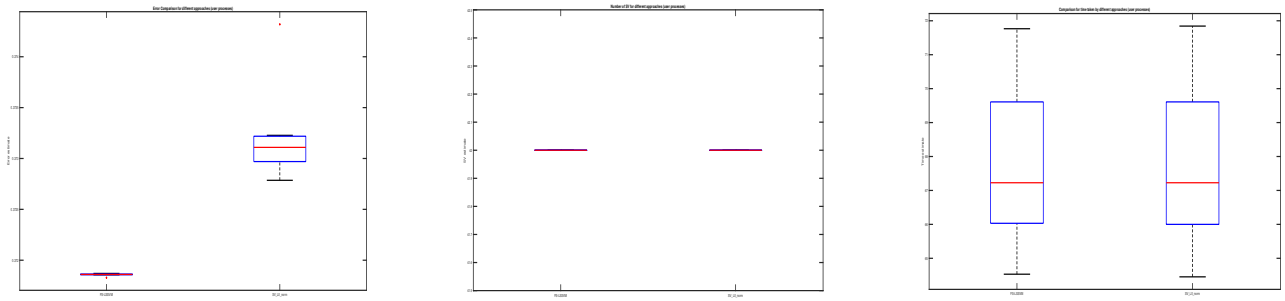


(a) Error estimates (b) Number of Support Vectors (c) Computation Time

Figure 3.7: Shuttle Dataset; Fixed Size - LSSVM vs l_0 - type approximation

California dataset

The California log dataset contains 20.640 data points with 9 different features. Fixed-size LS-SVM algorithm is run on this dataset and the results are plotted in the below figure 3.8. The error estimates is almost same for both the methods but Fixed-size LSSVM is has slight lower error by a value of 0.001 with no variability. Both FS-LSSVM and l_0 type methods takes equal number of support vectors around 42 as seen from figure 3.8b. Also both methods takes same computational time as seen from figure 3.8c.



(a) Error estimates (b) Number of Support Vectors (c) Computation Time

Figure 3.8: California Dataset; Fixed Size - LSSVM vs l_0 - type approximation