Support Vector Machines Methods and Applications

Angelos Evmorfopoulos

May 26, 2021

1 Exercise 1

1.1 A simple example: two Gaussians

In the toy example for the classification task we see two classes of Gaussians. Since this is a binary classification task, we can use Bayes rule for the examples we have. Specifically, we assign a class i^* to an example x such that $i^* = arg \max_{i=1,...,n_c} P(C_i|x)$ where n_c is equal to 2 and $P(C_i|x)$ corresponds to the posterior class probability of class C_i. In that way, the overlap area between the two distributions $P(x|C_1)P(C_1)$ and $P(x|C_2)P(C_2)$ is also minimized. Considering that the data is constructed by a Gaussian distribution with the same covariance matrices, the decision boundary is a linear separating hyperplane, as shown in Figure 1. This decision boundary is optimal since it is independent of the overlap area and it minimizes the probability of misclassification.

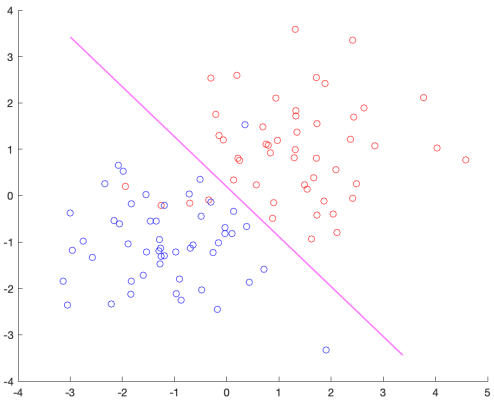
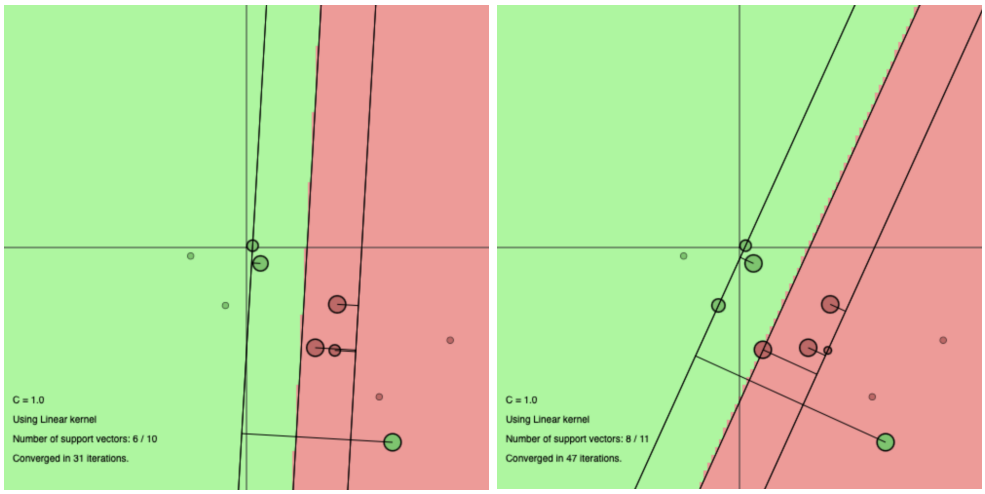


Figure 1: Optimal decision boundary

1.2 Support vector machine classifier

A support vector is the data point that is closest to the hyperplane. Therefore, a data point becomes a support vector when it is closer to the hyperplane. The data points that are closer to the hyperplane influence the hyperplane's orientation and position in the sense that potential modifications of a support vector's position will also affect the orientation and the position of the hyperplane. Their importance changes with regards to the hyperplane, since by removing or changing the position of a support vector, the position of the hyperplane also changes. This is illustrated in Figure 2, where we can see that the addition of extra support vectors changes the position and orientation of the hyperplane.

The addition of more datapoints to the dataset, when on the right side, do not greatly affect the outcome of the decision boundary, as opposed to additions on the wrong side, which have



(a) Initial (b) Addition of support vectors

Figure 2: Addition of new support vectors influence on the hyperplane

a significant effect. This happes due to the fact that additions on the wrong side increase the penalty applied for misclassification errors. This is illustrated in Figure 3.

The regularization hyperparameter C is responsible for handling misclassification errors. For larger values of C we observe that the hyperplane becomes more narrow, whereas for smaller ones it becomes less narrow. This is explained by the fact that the optimization will choose a hyperplane with a smaller margin that classifies all the training points correctly (for larger values of C). Conversely, significantly smaller values lead to larger margins in the hyperplane, even though the hyperplane misclassifies more points. This is illustrated in Figure 4.

The parameter σ is the radius that determines the influence that a single training example has on the classification . Specifically, when the distance from a datapoint exceeds σ, the datapoints will become too similar, which can lead to underfitting. In contrast, for very small values of σ the examples become too dissimilar, thus leading to overfitting. This is depicted in Figure 5.

As we can see in Figure 6 the Gaussian (RBF) kernel performs better than the linear kernel in our classification task. This is due to the fact that it can produce decision boundaries that are non-linear, in contrast to the linear kernel. The result of this is that the Gaussian kernel is able to capture more complex relationships among the datapoints.

1.3 Least-squares support vector machine classifier

1.3.1 Influence of hyperparameters and kernel parameters

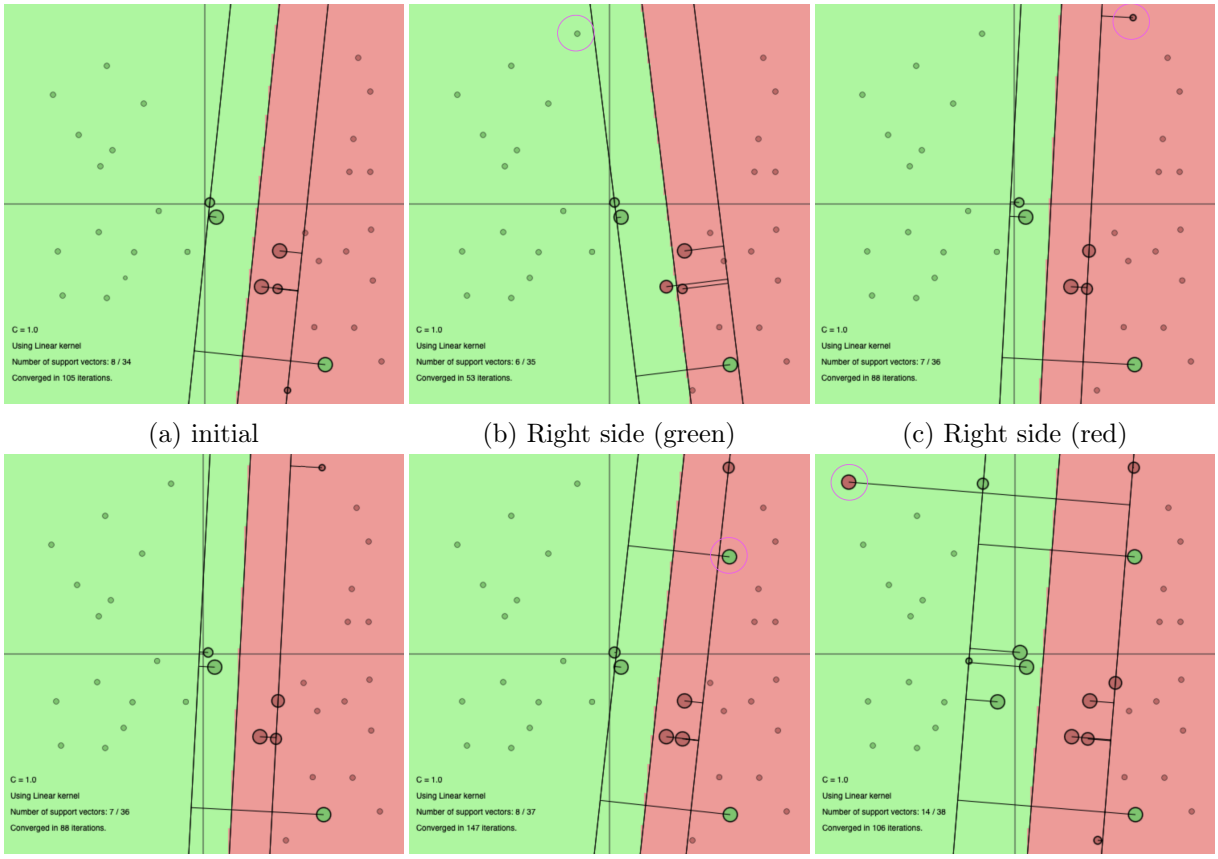
In this section, the objective is to perform classification in the Iris dataset, by using the least squares based variant of the SVM (LS-SVM). Additionally, we want to experiment with different kernel functions and hyperparameters and investigate the effect they have on our classifier. Two different kernel functions were implemented; **polynomial kernel** and **RBF kernel**.

Polynomial kernel For polynomials with degree d, the polynomial kernel is defined as:

$$K(x, x_k) = (x^T x_k + \tau)^d \tag{1}$$

where d corresponds to the degree of the polynomial kernel. The degree has a significant influence on the the decision boundary since it controls its flexibility. Higher degrees provide a more flexible decision boundary but at the same time increase the chance of overfitting.

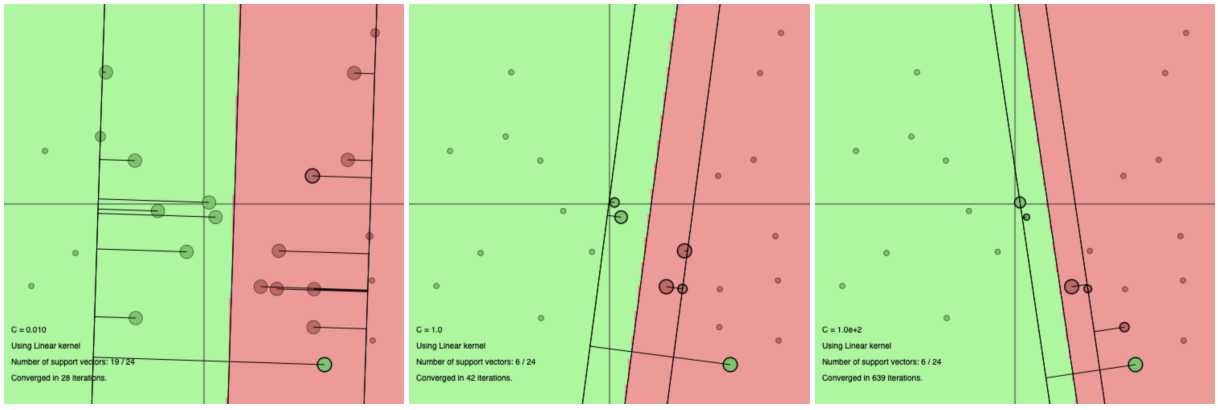
RBF kernel RBF kernels are in general the most widely used kernels. For two datapoints



(a) initial (b) Right side (green) (c) Right side (red)

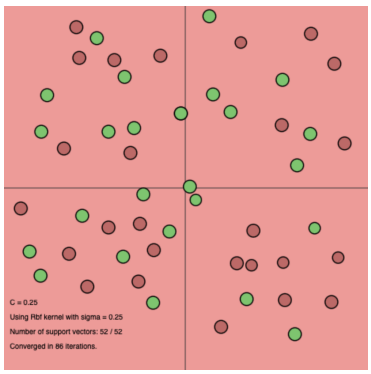
(d) initial (e) Wrong side (green) (f) Wrong side (red)

Figure 3: Illustration of the effect of the addition of new datapoints in the hyperplane

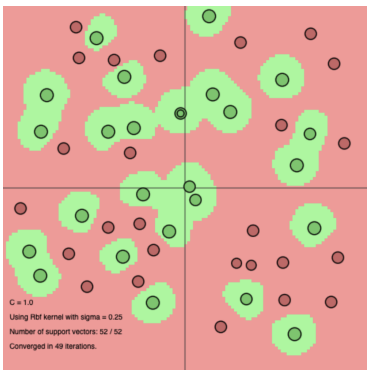


(a) c=0.01 (b) c=1 (c) c=100

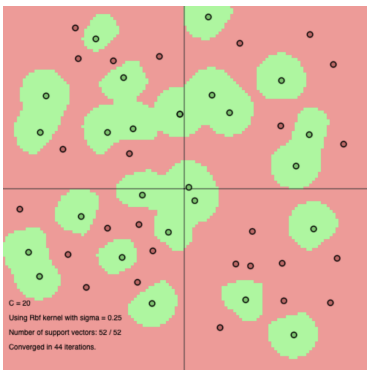
Figure 4: Various values for hyperparameter c in linear kernel



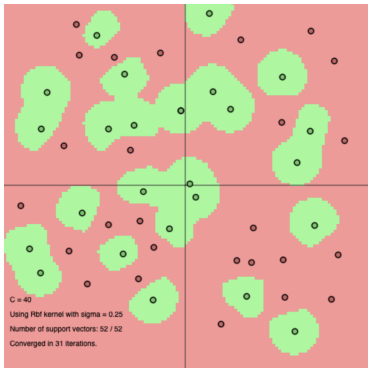
(a) $c = 0.25$, $\sigma = 0.25$



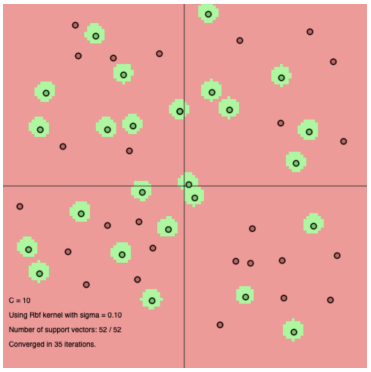
(b) $c = 1$, $\sigma = 0.25$



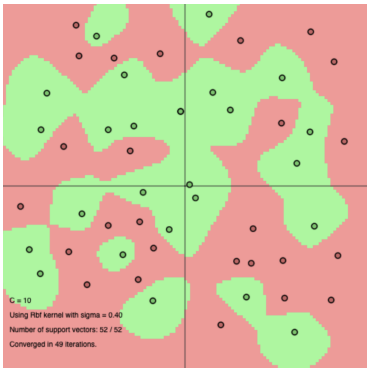
(c) $c = 20$, $\sigma = 0.25$



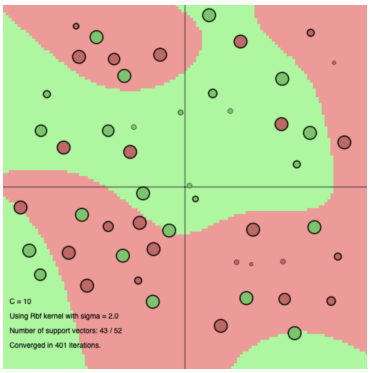
(d) $c = 40$, $\sigma = 0.25$



(e) $c = 10$, $\sigma = 0.1$



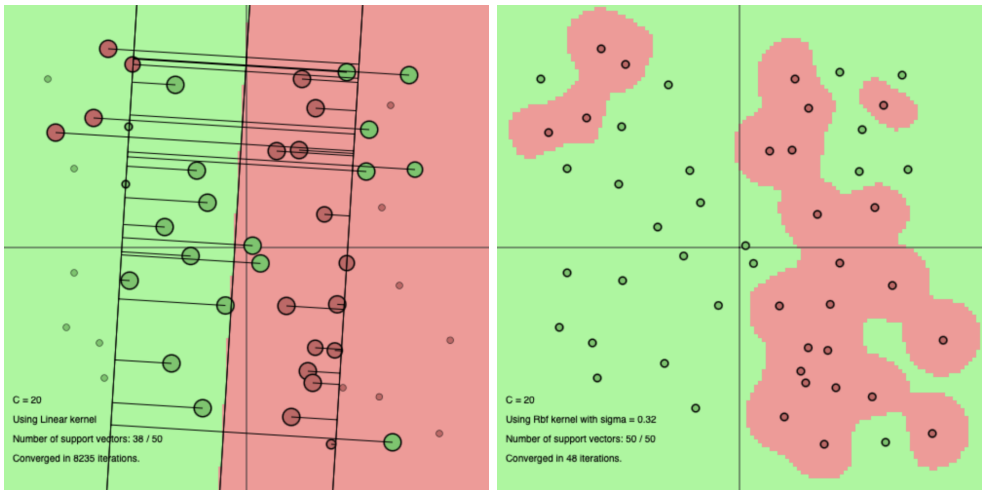
(f) $c = 10$, $\sigma = 0.4$



(g) $c = 10$, $\sigma = 2$

(h) $c = 10$, $\sigma = 13$

Figure 5: Various values for c and σ in the RBF kernel



(a) linear kernel (b) RBF kernel

Figure 6: Comparison between linear and RBF kernel

$X1$ and $X2$, the RBF kernel function computes their similarity and is defined as:

$$K(X1, X2) = exp(-\frac{||X1 - X2||_2^2}{\sigma^2}) \tag{2}$$

where σ is the variance and $||X1 - X2||$ is the Euclidean distance between two points $X1$ and $X2$.

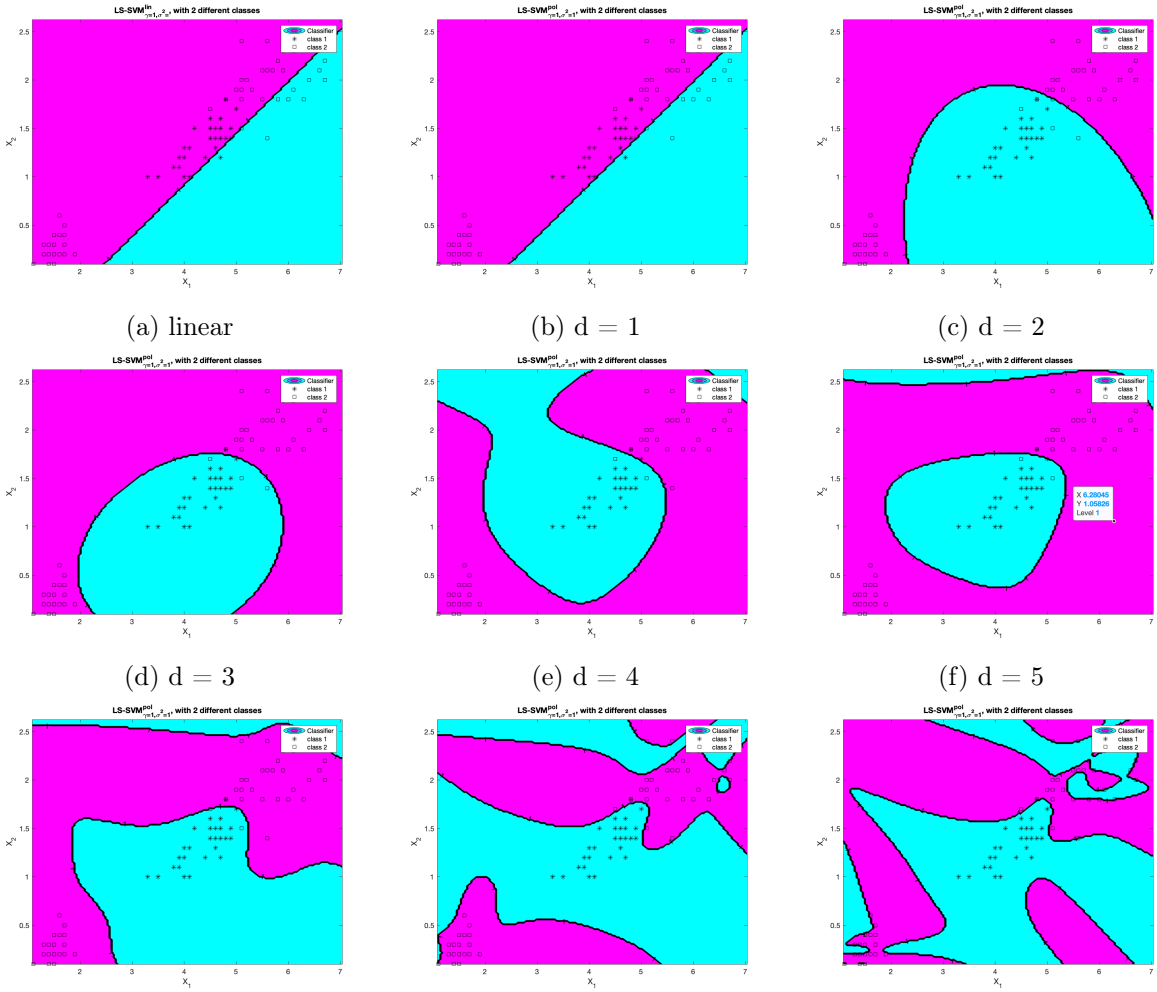
In practice, the first choice in our experiments is a linear kernel, which is then followed by polynomial and RBF kernels, which improve the classification. In Figure 7, we can see the decision boundary for a linear kernel (a). Then, from (b) to (f) we can see the decision boundaries for polynomial kernels with varying degrees. As illustrated, a linear kernel has a high classification error of 55% (11 out of 20 examples misclassified). For the remaining polynomial kernels, we can see a significant improvement in degrees 2 to 4 where the classification error drops to 5% and then to 0%. Finally, as the degree increases (higher than 15), the classification error increases as well. This is visualized in (h) and (i). In Figure 8 we can see the effects of varying γ and σ^2, which was discussed in the previous section. After the experiments, it was observed that the best results were obtained with a σ^2 in the range of 0.05 to 15 and for γ between 0.5 to 50.

1.3.2 Tuning parameters using validation

In this section, the objective is to experiment with three different automated tuning methods and evaluate their performance. These automated tuning methods split the dataset into a training and validation set.

Random Split In random split, the dataset is randomly split into a training and a validation set. In order to evaluate the model the validation set is used. In contrast to a test set, the validation set is not meant to measure the performance of a fully specified model. The validation set is comprised of data independet of the ones used in the training and test sets, therefore it enables the model to be able to generalize more. One major issue of this method is that since the split is random, it may not be able to perform well when the dataset is too small. Additionally, the model is highly susceptible to the choice of data over which it is evaluated.

K-fold cross validation In k-fold cross validation the given dataset is randomly split into k subsets, each having an equal size. The model is trained in $k - 1$ subsets and the remaining subset after the split is used as the validation set. The same process is then repeated k times and every time the training and validation subsets switch. Finally, the created subsets are averaged

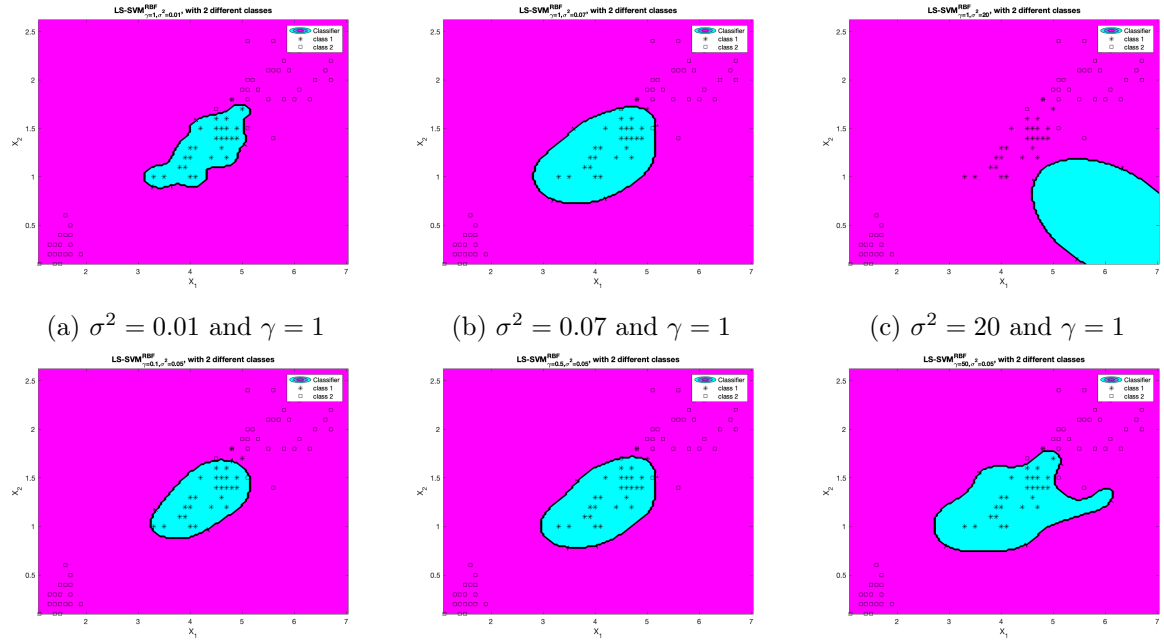


(a) linear (b) d = 1 (c) d = 2

(d) d = 3 (e) d = 4 (f) d = 5

(g) d = 10 (h) d = 15 (i) d = 20

Figure 7: (a) linear kernel (b to i) different degrees on polynomial kernel



(a) $\sigma^2 = 0.01$ and $\gamma = 1$ (b) $\sigma^2 = 0.07$ and $\gamma = 1$ (c) $\sigma^2 = 20$ and $\gamma = 1$

(d) $\sigma^2 = 0.05$ and $\gamma = 0.1$ (e) $\sigma^2 = 0.05$ and $\gamma = 0.5$ (f) $\sigma^2 = 0.05$ and $\gamma = 50$

Figure 8: Varying σ^2 and γ in the RBF kernel.

Simplex			Gridsearch		
γ	σ^2	cost	γ	σ^2	cost
2.4124	0.0658	0.043	0.340	0.9661	0.036

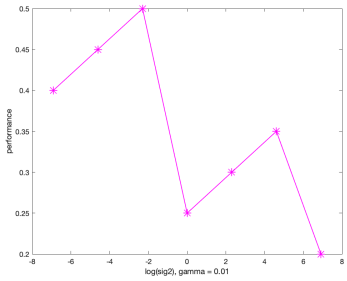
Table 1: Hyperparameters tuned with Nelder-Mead method and Gridsearch for the classification task

and an estimation is obtained in order to evaluate the performance. In general, a specific number which is ideal for k does not exist; however, it has been observed that a relatively low k results in lower variance and higher bias, whereas a higher k increases the variance but reduces the bias.

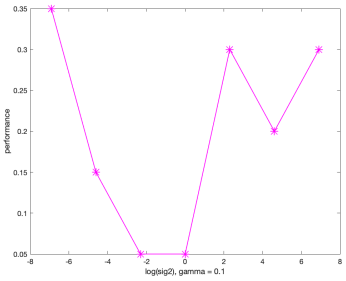
Leave-one-out validation Leave-one-out validation is a special case of k-fold cross validation, in which the number of all training examples X is assigned the number of k. Training of the model is performed on all the available data X times and the errors are averaged in order to measure the performance. A drawback of leave-one-out validation is that in large dataset it can affect the performance since it will require a high number of training data.

1.3.3 Automatic parameter tuning

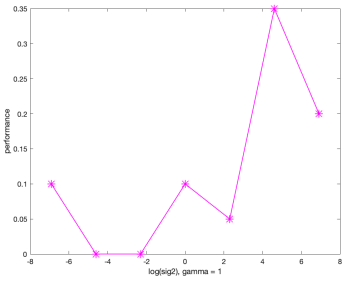
In this section, two different algorithms (gridsearch and simplex) are tested in the task of automatic parameter tuning. These techniques are utilized in order to automatically tune the hyperparameters of a LS-SVM and are implemented in the provided LS-SVM toolbox[1]. Their results are summarized in Table1. The Nelder-Mead method (simplex) is faster than gridsearch, taking almost half the time to complete the computations and obtain its parameters (0.54 and 0.9 seconds respectively). Finally, we can observe a high variation in the hyperparameters everytime due to the fact that multiple minimum solutions may exist.



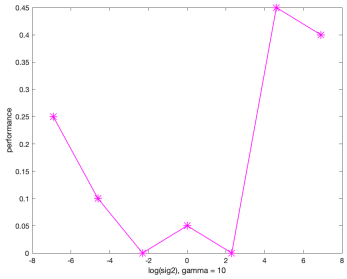
(a) σ and γ equal to 0.1



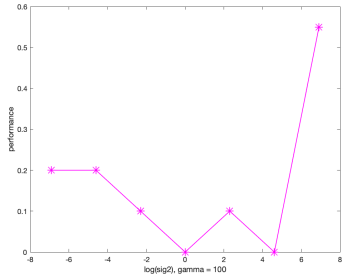
(b) σ and γ equal to 1



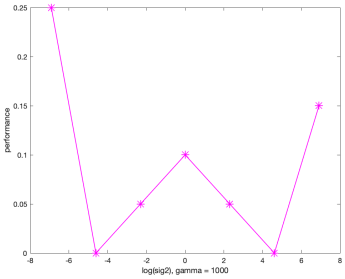
(c) σ and γ equal to 10



(d) σ and γ equal to 100

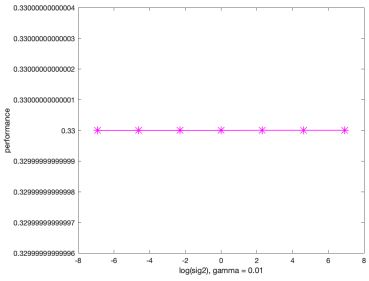


(e) σ and γ equal to 0.01

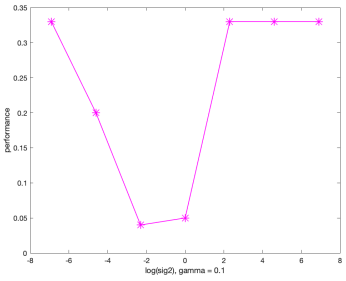


(f) σ and γ equal to 1000

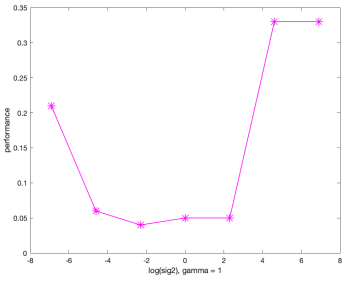
Figure 9: Varying the values of σ and γ in random split



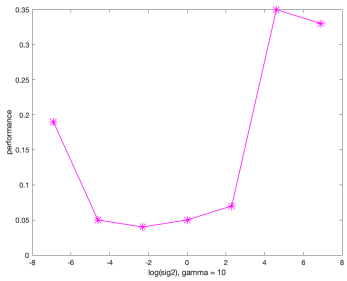
(a) σ and γ equal to 0.01



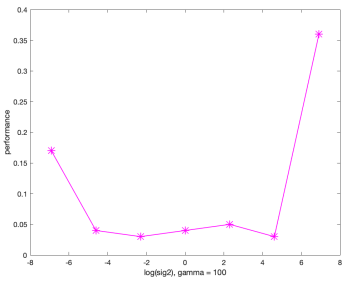
(b) σ and γ equal to 0.1



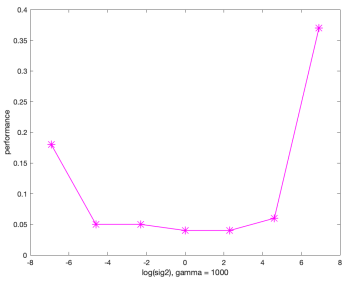
(c) σ and γ equal to 1



(d) σ and γ equal to 10

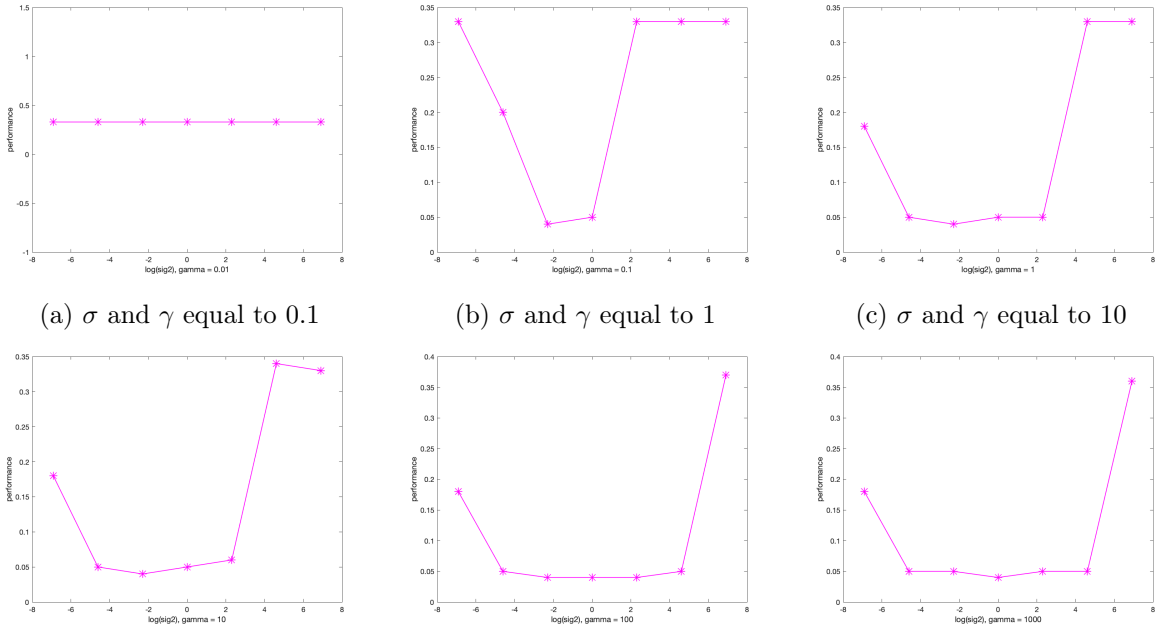


(e) σ and γ equal to 100



(f) σ and γ equal to 1000

Figure 10: Varying the values of σ and γ in 10-fold



(a) σ and γ equal to 0.1 (b) σ and γ equal to 1 (c) σ and γ equal to 10

(d) σ and γ equal to 100 (e) σ and γ equal to 0.01 (f) σ and γ equal to 1000

Figure 11: Varying the values of σ and γ in leave-one-out

1.3.4 Using ROC curves

A Receiver Operating Characteristic (ROC) curve is used to measure the performance of a classifier. Specifically, a ROC curve expresses the relationship between specificity and sensitivity of a classifier. The sensitivity of a classifier is defined as

$$TPR = \frac{TP}{TP + FN} \tag{3}$$

where TP and FN (true positive and false negative) correspond to the number of correctly and incorrectly classified instances as positive. Sensitivity is also named as true positive rate. The specificity of a classifier is defined as

$$FPR = \frac{TN}{TN + FP} \tag{4}$$

where TN and FP correspond to the number of correctly and incorrectly classified instances as negative. The Area Under the Curve (AUC) in a ROC curve describes the performance of the model. If it is close to 0 then the model is able to separate the data perfectly, whereas if it is close to 0.5 the model behaves randomly. In Figure 12 we can see the ROC curve of the Iris dataset.

1.3.5 Bayesian framework

In this section we experiment with using the Bayesian framework[2] to get probability estimates. In Figure 13 we can see the illustration of the bayesian framework. The color represents the probability of an instance to belong to the positive class. Therefore, the blue color can be interpreted as the probability of the instance belonging to the positive class being close to zero, whereas magenta represents the opposite.

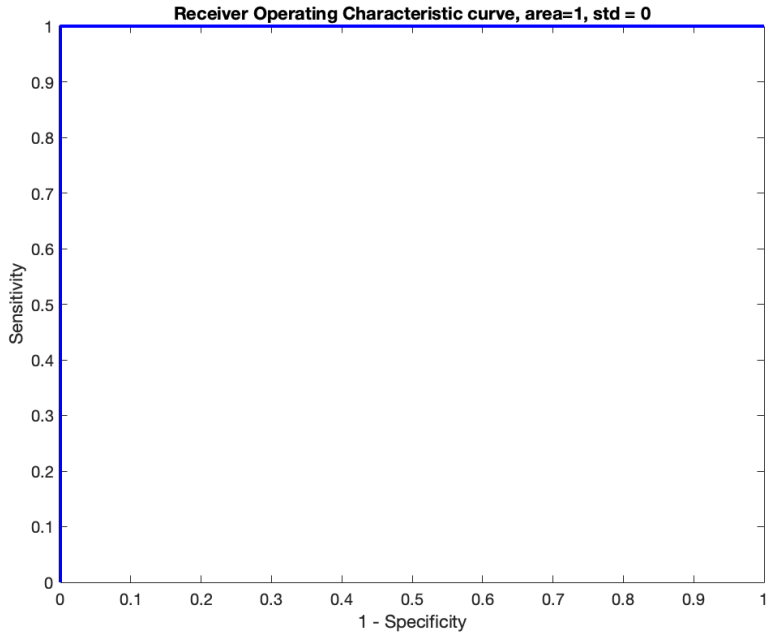


Figure 12: ROC curve of the Iris dataset

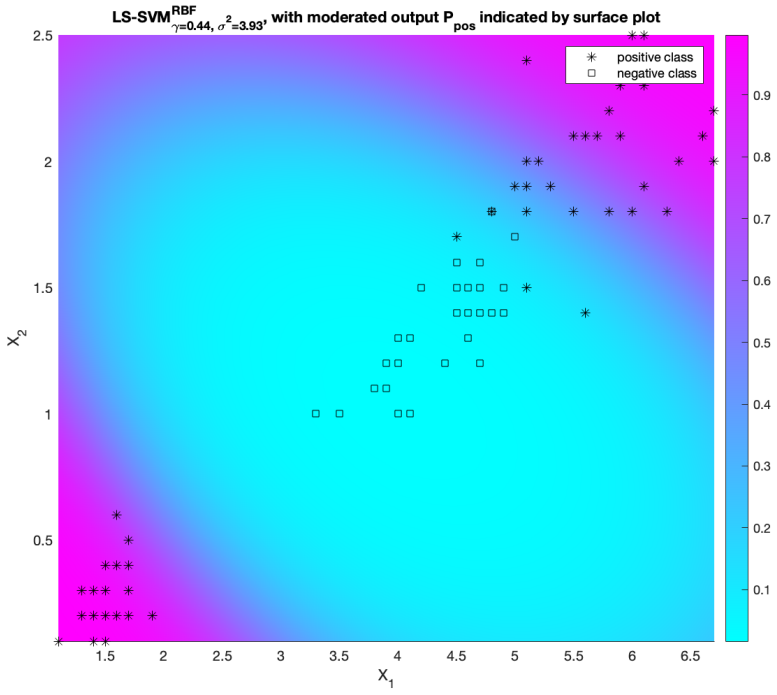
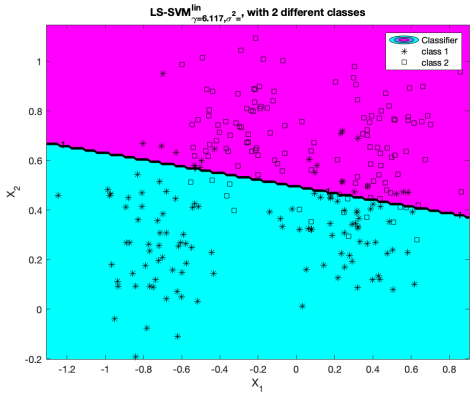


Figure 13: Bayesian framework probabilistic view

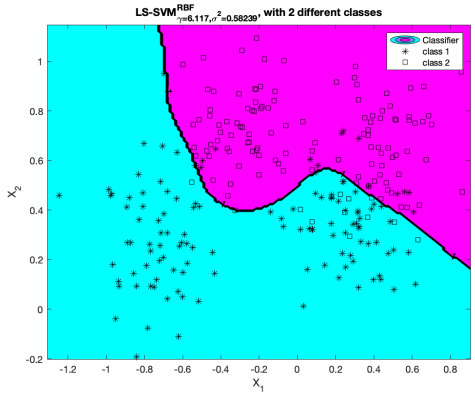
1.4 Homework problems

1.4.1 Ripley dataset

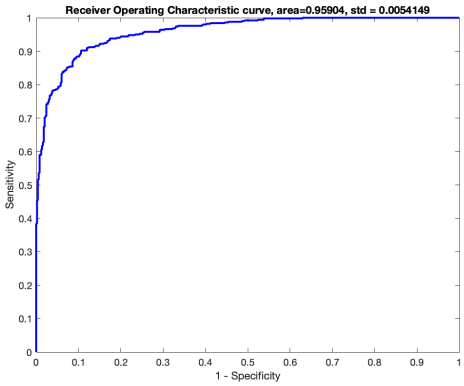
The Ripley dataset contains of a training set and a test set. The training set contains 250 datapoints for training and 1000 datapoints for testing the model. As we can see in Figure 14 (e) the two classes are quite, although not fully, separated from each other. Additionally, the results of applying a linear kernel and an RBF kernel are similar (0.959 and 0.968 respectively) and are visualized in the ROC curves (c) and (d). The tuning of the hyperparameters for the RBF kernel were obtained after using the Nelder-Mead method which was discussed in section 1.3.3.



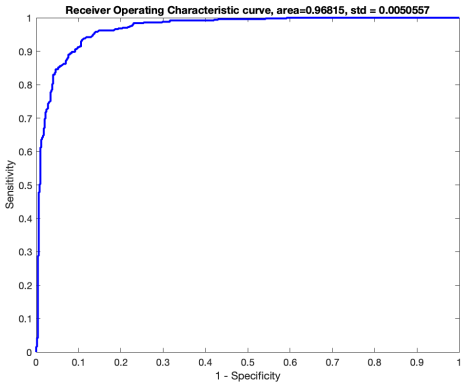
(a) Linear kernel



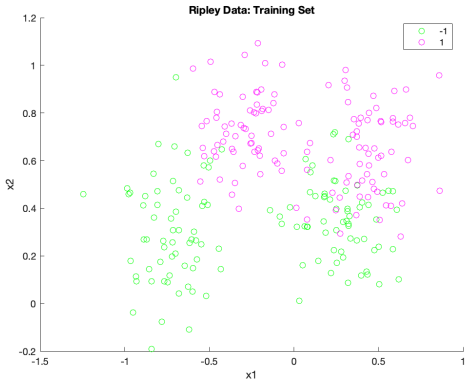
(b) RBF kernel



(c) Linear kernel ROC



(d) RBF kernel ROC

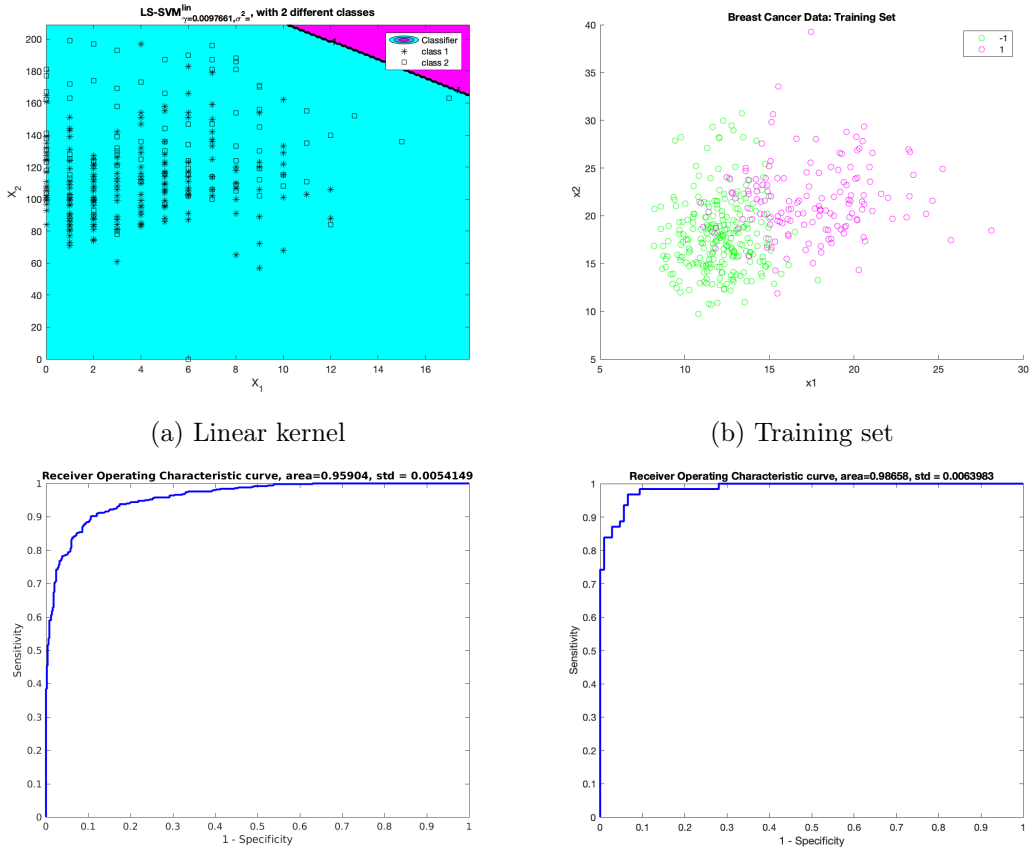


(e) training set

Figure 14: Ripley dataset

1.4.2 Wisconsin Breast Cancer dataset

The Wisconsin Breast Cancer dataset is comprised of 569 examples, 400 for training and 169 for testing purposes. Each datapoint is constructed from 30 features that describe the metrics utilised for classification. The visualization of the dataset is illustrated in Figure 15. As is illustrated in (c) and (d), the accuracy of the linear kernel is 0.9594 and the accuracy of the RBF kernel is 0.98658.



(a) Linear kernel

(b) Training set

(c) TLinear kernel ROC

(d) RBF kernel ROC

Figure 15: Wisconsin Breast Cancer dataset

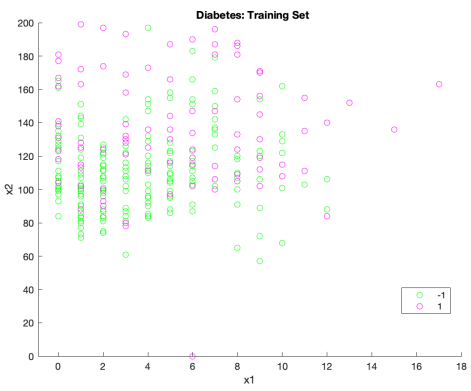
1.4.3 Diabetes dataset

The Diabetes dataset consists of 468 examples in total, out of which 300 are utilized for training and 168 for testing the model. Each datapoint is constructed from 8 features. The two classes have high overlap in their distribution as visualized in Figure 16. The accuracy of the linear kernel is 0.84327.

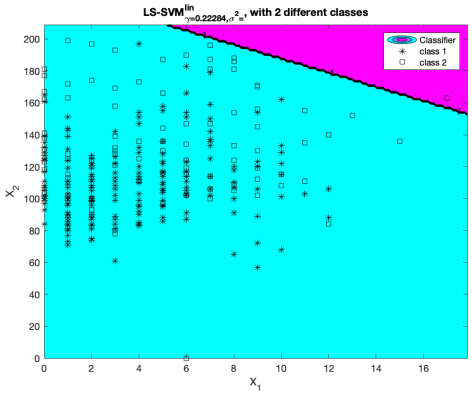
2 Exercise 2

2.1 Support vector machine for function estimation

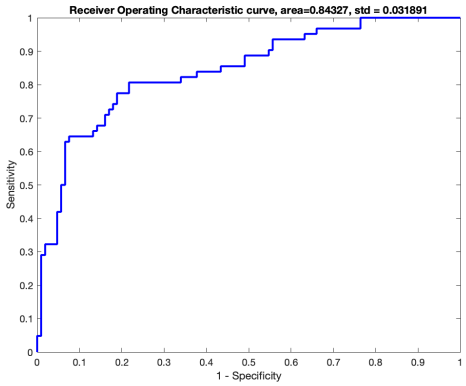
In this exercise we utilize the SVM toolbox in order to perform function estimation. In order to use an SVM to perform such a task, it needs to be extended to be able to solve linear and nonlinear function estimation problems. Specifically, extending an SVM classifier for a regression



(a) Training set



(b) Training set



(c) Linear kernel ROC

Figure 16: Diabetes dataset

task is done as following:

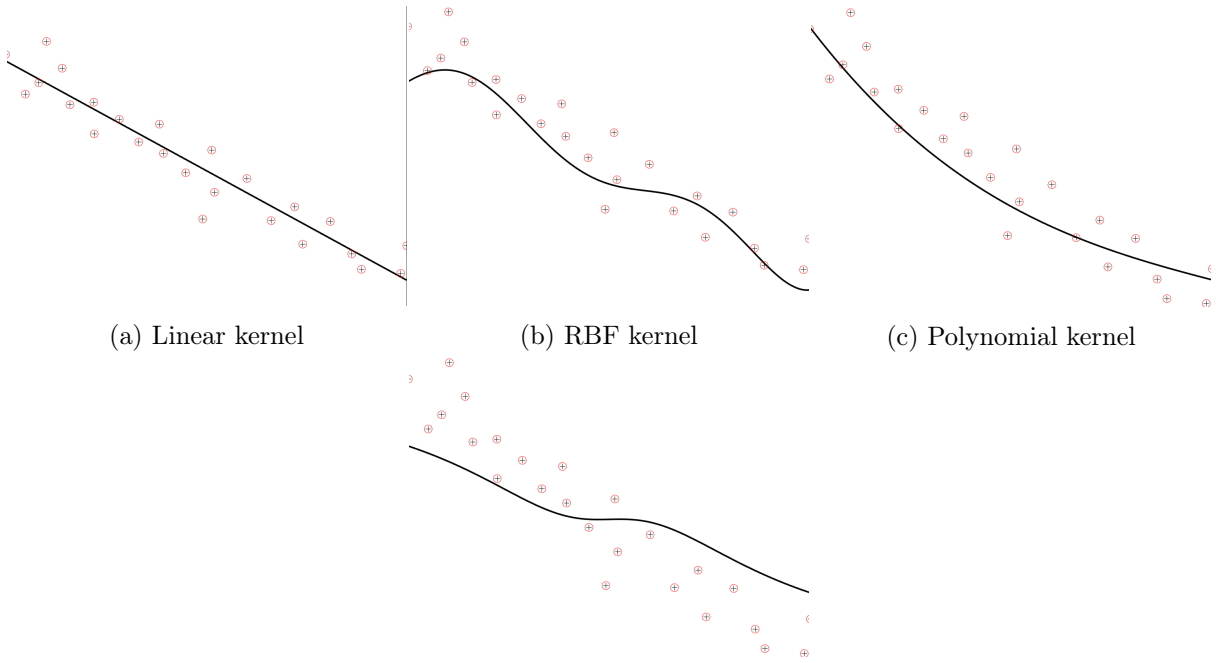
$$\min_{w,b,\xi_i} J(w,b) = \frac{1}{2} w^T w + C \sum_{k=1}^{N} (\xi_i) \tag{5}$$

with

$$|y_i - w_i x_i| \leq \epsilon - \xi_i \tag{6}$$

The regularization parameter C (also known as Bound), as discussed in the previous exercise session, is the parameter that determines the influence of misclassification on the objective function. As C increases we can obtain more datapoints outside the area of ϵ, which is the value responsible for controlling the sparsity (number of support vectors) of the resulting solution. Increasing the value of ϵ to relatively high numbers results in less support vectors, therefore the accuracy of the function approximation is less accurate. The initial dataset and the applied kernels are illustrated in Figure 17. Initially, we construct a dataset that will be utilized for a linear kernel SVM and next we construct a more complex dataset. Both datasets are comprised of around 20 points and different kernels are tested for each dataset.

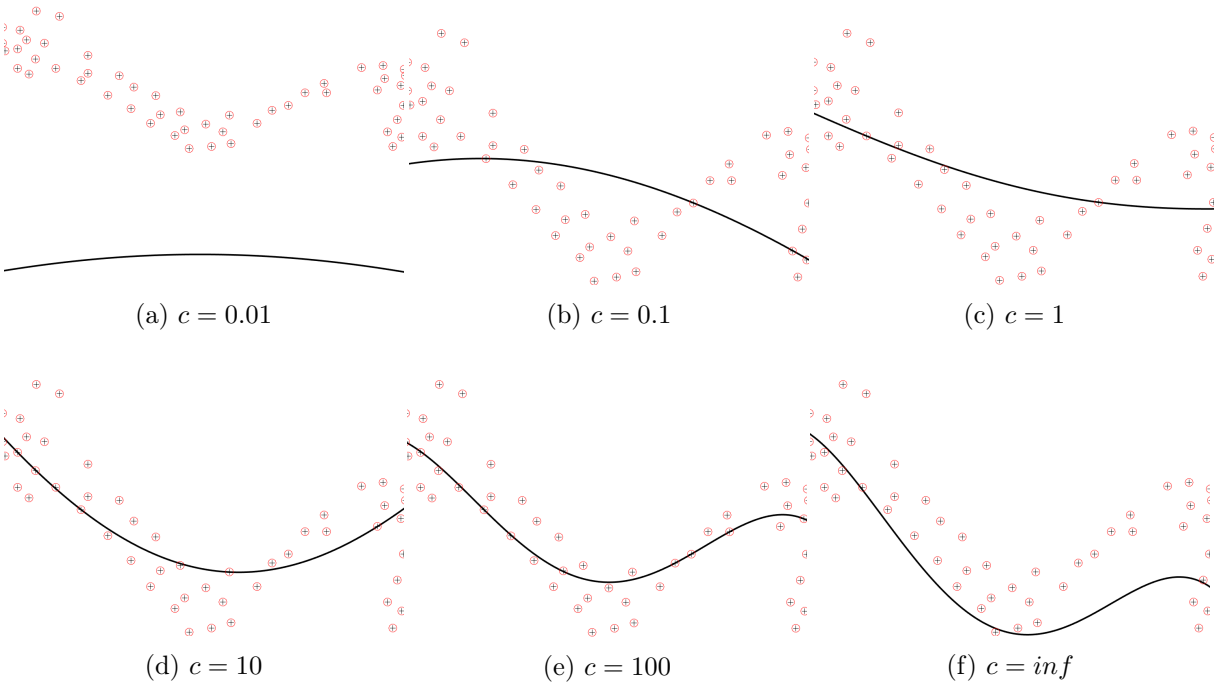
In Figure 18 we can see that while decreasing the bound, the model tends to become more smooth and more sparse. Conversely, when ϵ decreases, the model becomes less sparse and the approximation more smooth. This is illustrated in Figure 19.



(a) Linear kernel (b) RBF kernel (c) Polynomial kernel

(d) MLP kernel

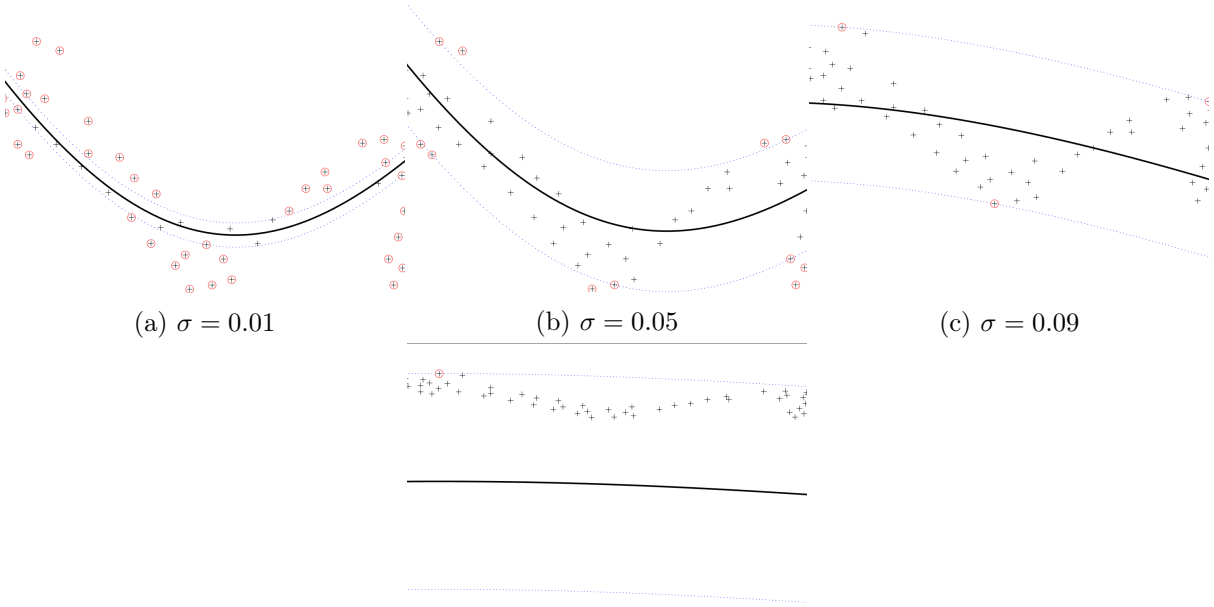
Figure 17: Applying different kernels to the dataset



(a) $c = 0.01$ (b) $c = 0.1$ (c) $c = 1$

(d) $c = 10$ (e) $c = 100$ (f) $c = inf$

Figure 18: Varying c with σ fixed to 0.01



(a) $\sigma = 0.01$ (b) $\sigma = 0.05$ (c) $\sigma = 0.09$

(d) $\sigma = 0.5$

Figure 19: Varying σ with c fixed to 100

2.2 The sinc function

2.2.1 Regression of the sinc function

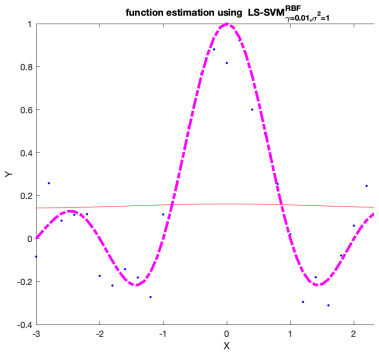
The objective of this exercise is to approximate the function

$$y = sinc(x) + \epsilon \tag{7}$$

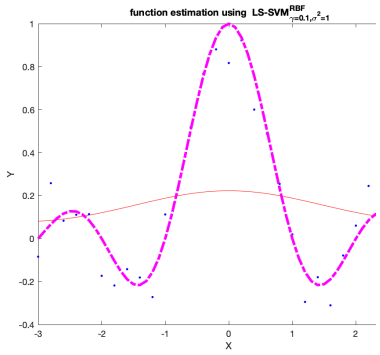
Initially, we construct an artificial dataset from the *sinc* function. Next, we manually test the model for varying values of c and σ^2. The model that achieved the lowest MSE score (0.013) was initialised with $c = 10^6$ and $\sigma^2 = 1$. In Figure 20 we can see various values for c with a fixed $\sigma = 1$ whereas Figure 21 illustrates various values for σ with a fixed $c = 10^6$. In the second part of the exercise, we use autotuning of the hyperparameters with 10-fold crossvalidation. Similarly to the previous exercise session, the methods utilized for automated hyperparameter tuning are the Nelder-Mead method (simplex) and gridsearch. We test the parameters obtained after 50 runs and the results are aggregated in Table 2.

Simplex			Gridsearch		
γ	σ^2	cost	γ	σ^2	cost
197.57	0.1798	0.0196	103.81	0.2661	0.019

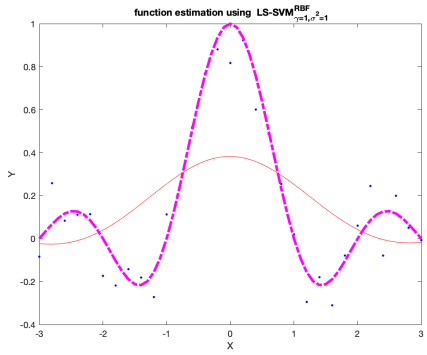
Table 2: Hyperparameters tuned with Nelder-Mead method and Gridsearch for the regression task



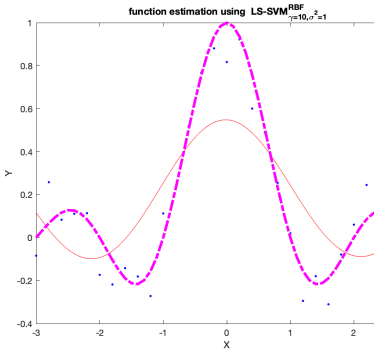
(a) $c = 0.01$



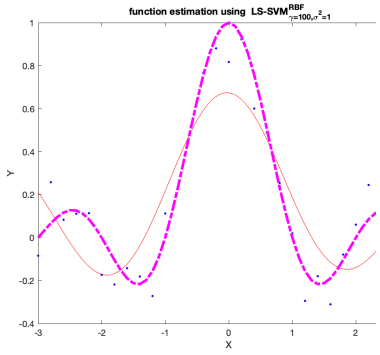
(b) $c = 0.1$



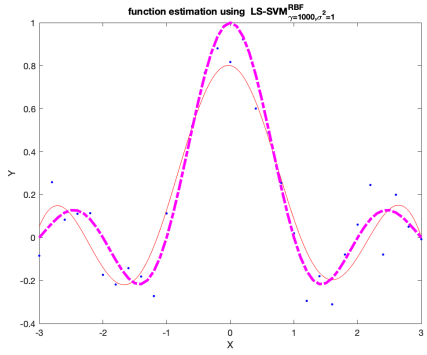
(c) $c = 1$



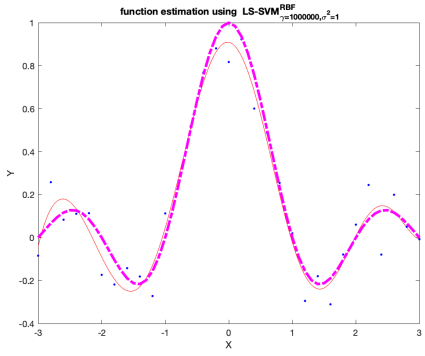
(d) $c = 10$



(e) $c = 100$

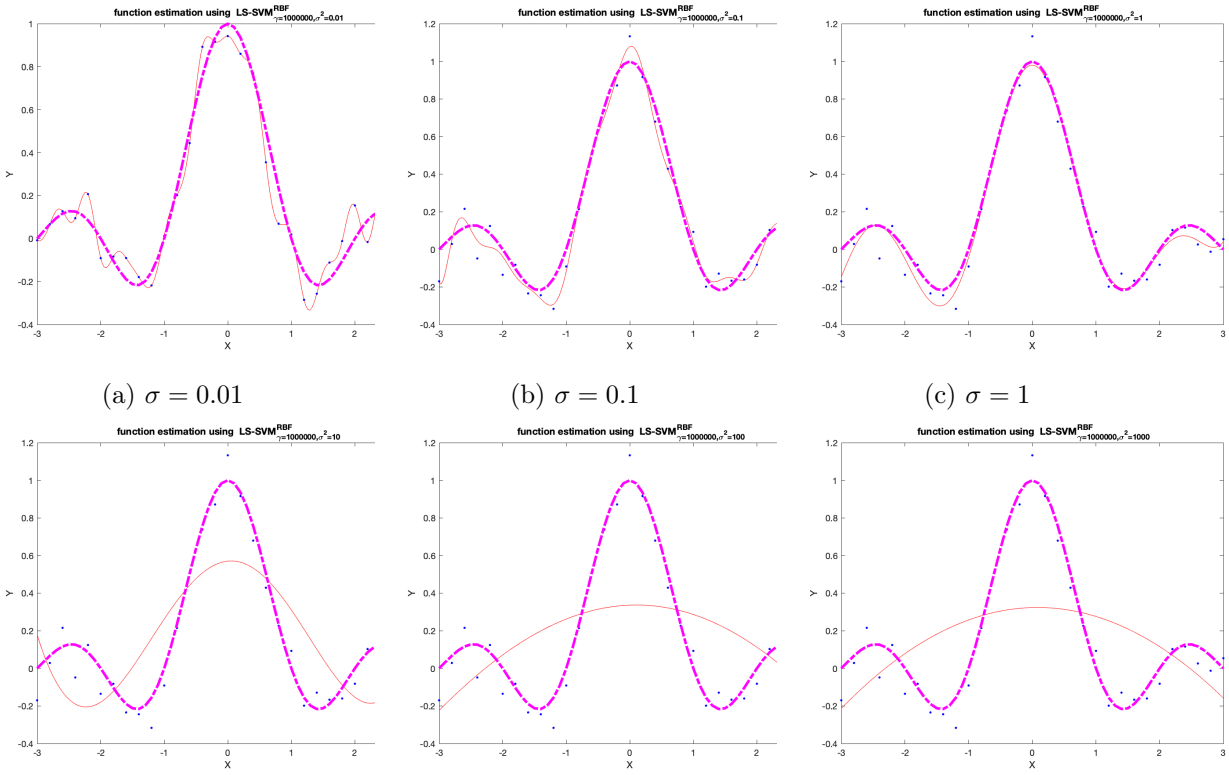


(f) $c = 10^3$



(g) $c = 10^6$

Figure 20: Varying c with σ fixed to 1



(a) $\sigma = 0.01$ (b) $\sigma = 0.1$ (c) $\sigma = 1$

(d) $\sigma = 10$ (e) $\sigma = 100$ (f) $\sigma = 10^3$

Figure 21: Varying σ with c fixed to 10^6

2.2.2 Application of the Bayesian framework

When working in a Bayesian learning setting datapoints are characterized by their probability distribution. In such a setting, we assigned a probability over all the possible values for a datapoint instead of treating it as an individual point. Therefore, we have a prior $p(w)$ distribution over data D and a likelihood distribution $p(D|w)$ which describe the posterior probability distribution $p(w|D)$ as

$$P(w|D) = \frac{P(D|w)P(w)}{P(D} \tag{8}$$

Parameter tuning in a Bayesian network works on three different levels, determined by the Bayesian framework that applies the aforementioned rule.

Level 1 In the first level,the parameters that are inferred are w and b. In this level, we calculate the posterior distribution parameters, given by

$$p(w, b|D, \mu, \zeta, H_\sigma) = \frac{p(D|w, b, \mu, \zeta, H_\sigma)}{p(D|\mu, \zeta, H_\sigma} p(w, b|\mu, \zeta, H_\sigma) \tag{9}$$

In the LSVM toolbox in particular, **crit_L1** corresponds to the cost that is proportional to the posterior $p(w, b|D, \mu, \zeta, H_\sigma$.

Level 2 In the second level, the parameters that are inferred are μ and ζ, with $\gamma = \frac{\mu}{\zeta}$. They are inferred from

$$p(\mu, \zeta|D, \mu, H_\sigma) = \frac{p(D|\mu, \zeta, H_\sigma)}{p(D|H_\sigma)} p(\mu, \zeta|H_\sigma) \tag{10}$$

In the LSVM toolbox these are calculated by **crit_L2**. In order to optimize the posterior, we first need to find the optimal γ, which is calculated with $\gamma = \frac{\mu}{\zeta}$. This is obtained by using the **bay_optimize** function.

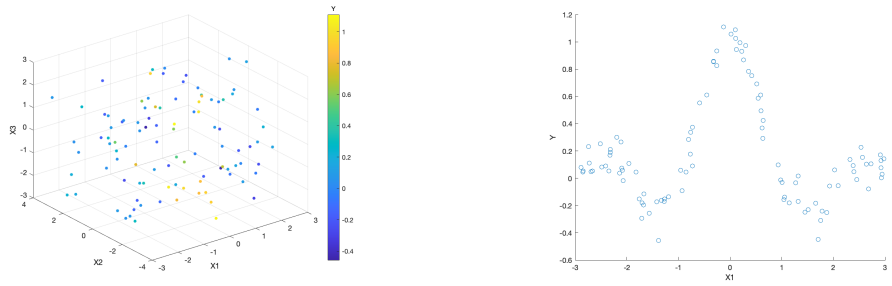
Level 3 In the third level the parameters that are inferred are the kernel parameters. In the case of RBF, the distribution of the kernel parameters is defined as

$$p(H_\sigma|D) = \frac{p(D|H_\sigma)}{p(D)} p(H_\sigma) \tag{11}$$

where H_σ corresponds to an RBF kernel with width σ.

2.3 Automatic relevance determination

Automatic relevance determination (ARD) is a method that is utilized in order to evaluate the relevance of the input features. In this exercise we apply ARD to a three-dimensional input ($X1$, $X2$, $X3$) in order to determine the most significant feature. After executing the program, the result is that $X1$ is the most relevant feature. Figure 22 illustrates (a) the initial situation and (b) the selected feature.

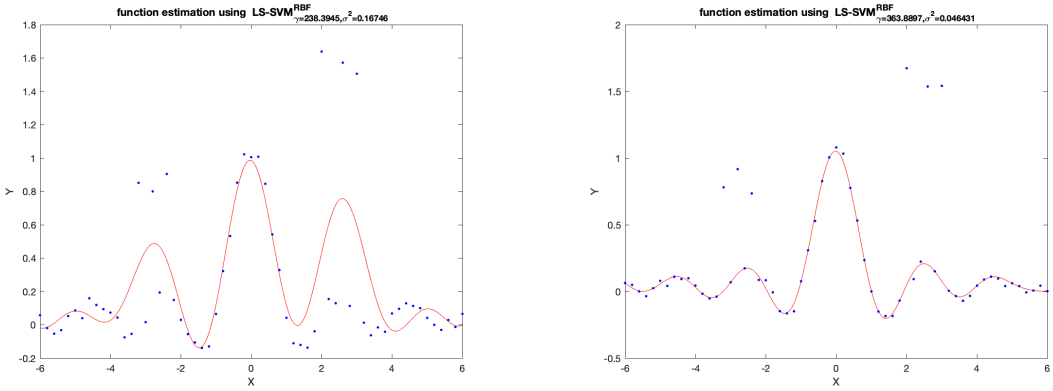


(a) initial (b) selected

Figure 22: (a) Initial 3-dimensional distribution of X1, X2, X3 and (b)Distribution of most important feature X1

2.4 Robust Regression

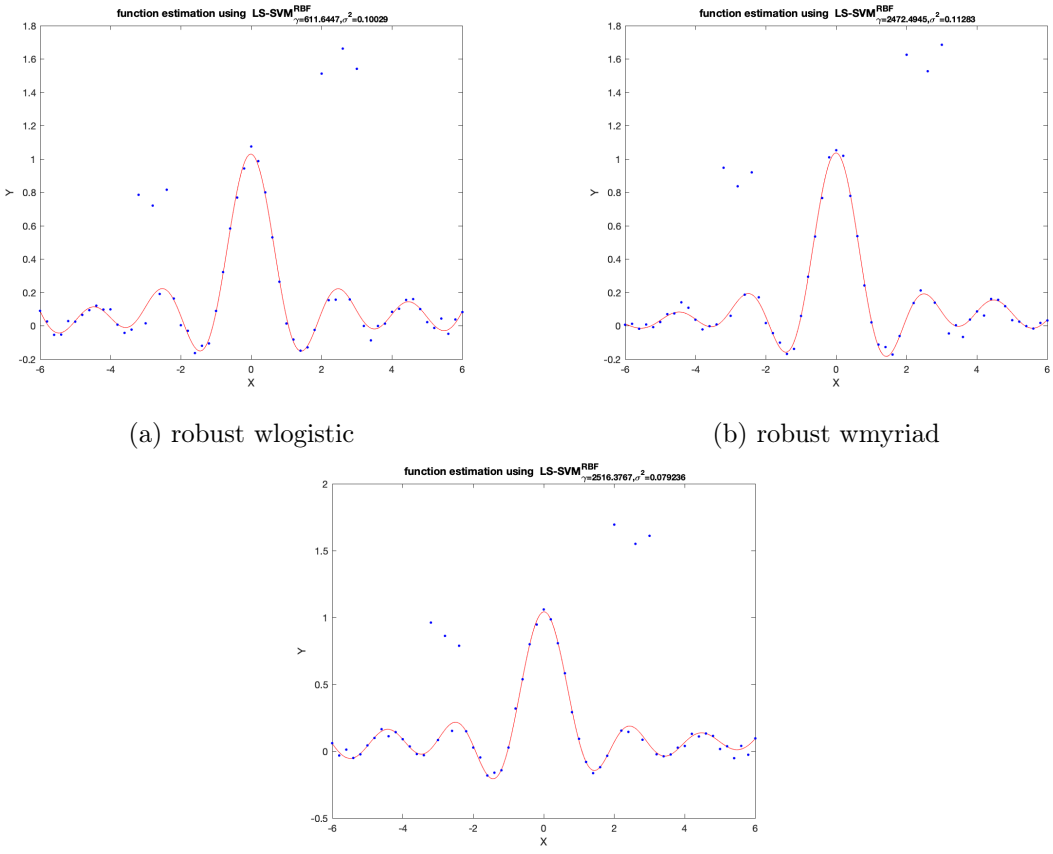
In this section we investigate the effect of outliers on an LS-SVM classifier. In particular, new datapoints are added to Y and then we train two different LS-SVM models: (1) that does not take into account efficient handling of them (non-robust) and (2) a model that takes into account the outliers (robust). The two models behave differently which is to be expected due to the effect the outliers have in such tasks. The non-robust model reacts to the outliers and tries to approximate them instead of the function whereas the robust model is not greatly affected and approximates the function better. This is illustrated in Figure 23.



(a) non-robust model (b) robust model

Figure 23: Illustration of the (a) non-robust and (b) robust models

Different weighting functions were tested for this task. The one used in Figure 23 is 'whampel'. Figure 24 illustrates the remaining weighting functions that were investigated. Finally, MAE is used instead of MSE due to the fact that it can handle outliers more efficiently since minimizing MAE will not greatly affect the estimator.



(a) robust wlogistic (b) robust wmyriad

(c) robust whuber

Figure 24: Illustration of the varying weighting functions

2.5 Homework problems: Time series prediction

2.5.1 Logmap dataset

The objective of this exercise is to perform time series prediction in the logmap dataset. This dataset consists of 150 points and the goal is to predict the next 50. In order to optimize the **order**, we can use crossvalidation (10-fold) in a similar way as we did previously for optimizing *gamma* and σ parameters. Initially we set the value of order to a fixed number and then, after obtaining the values for γ and σ, we compare the results based on their MSE. After several iterations, the optimal number for **order** was 25, with $\gamma = 1.0235e+06$ and $\sigma = 2.22790e+03$, with an MSE of 0.039. This is illustrated in Figure 25.

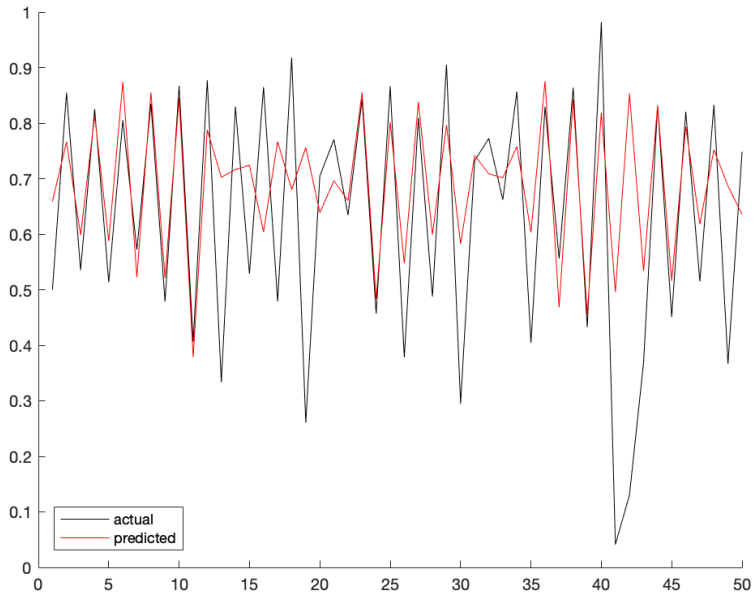


Figure 25: Time series prediction on the logmap dataset

2.5.2 Santa Fe dataset

The goal of this exercise is to perform time series prediction to the Santa Fe dataset [3], which consists of 1000 datapoints. The task is to predict the next 200 datapoints. In such tasks we use the previous instances in order to predict their future values. This approach is called nonlinear autoregressive model since it regresses on itself. Similarly to the previous exercise, we use 10-fold crossvalidation in order to determine the values of the hyperparameters γ and σ. Then we vary the order from 0 to 50 and we compare the MSE scores after each iteration. The optimal values of the hyperparameters were $\gamma = 1.1732e+03$ and $\sigma = 21.28$ and order 47. This is illustrated in Figure 26.

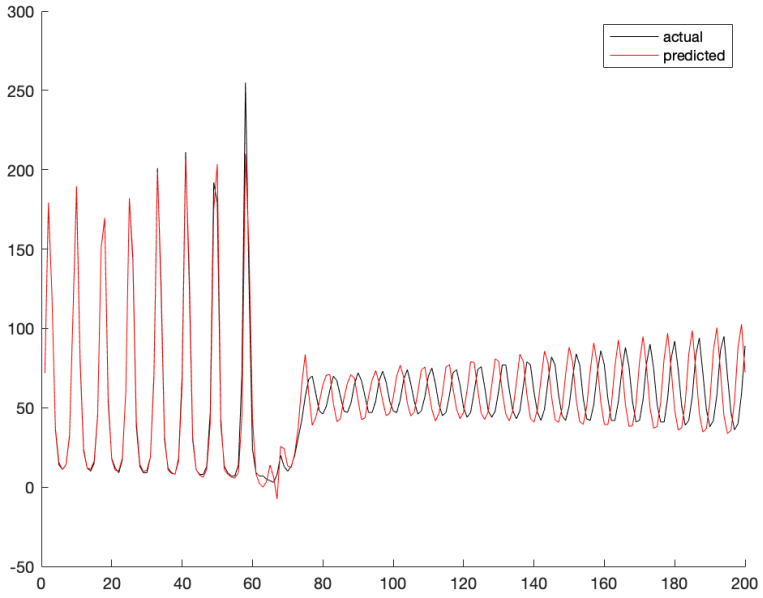


Figure 26: Time series prediction on the Santa Fe dataset

3 Exercise 3

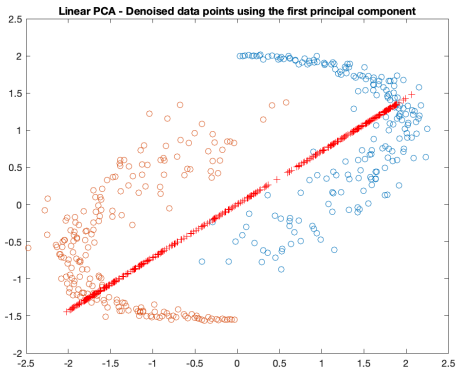
3.1 Kernel principal component analysis

Principal Component Analysis (PCA) is a technique that is utilized in various applications, such as image processing, time series prediction and data compression. It is a method that performs dimensionality reduction of the given data, by mapping them into a lower dimensional space. After performing PCA we can use the extracted features in applications such as the ones mentioned above. Similarly we can use the extracted features for the denoising task in the sense that the extracted features' projection will contain the input's most relevant information. Next, the principal components that are obtained after performing PCA can be used to project the data back to their initial dimension. By doing so, the noise will be reduced since only the relevant principal components are chosen after PCA, therefore the noise is ignored.

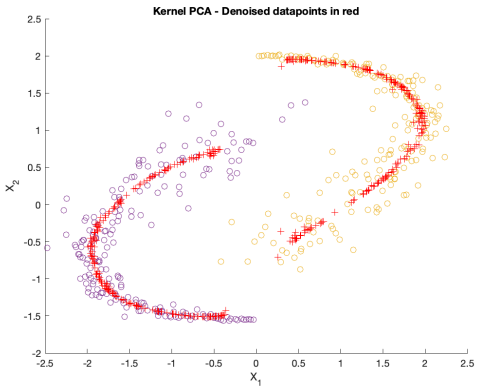
The number of components n_c controls the extent to which denoising can occur since if the reconstruction uses a high number of the first n_c then the data will simply return to their original, non-reduced format and therefore the noise will not be omitted. However, if the n_c is relatively low, then the underlying structure will not be captured accurately. This represents the trade-off between high variance (large n_c) and high bias (low n_c).

The performance of linear PCA and kernel PCA have substantial differences. Specifically, kernel PCA has a significant advantage over linear PCA since it is able to capture nonlinear structures. The comparison between the two is illustrated in Figure 27. For linear PCA the direction where the variance is maximal does not describe the characteristics of the dataset efficiently, in contrast to kernel PCA which describes the underlying structure very efficiently. An important difference between linear and kernel PCA is that linear PCA is limited by the input space whereas kernel PCA is not and can effectively use the number of components n_c, where n_c is higher than the input space. Effectively, in our case, this means that the linear PCA can only obtain up to 2 principal components while kernel PCA can obtain up to 400. We can see the results obtained after varying the number of principal components for kernel PCA in Figure 28 Finally, we can tune and optimize the hyperparameters and the number of principal components by using crossvalidation (or other tuning mechanisms) as discussed in the previous

sections. MSE can be utilized to measure and minimize the error in reconstruction of features in our dataset.

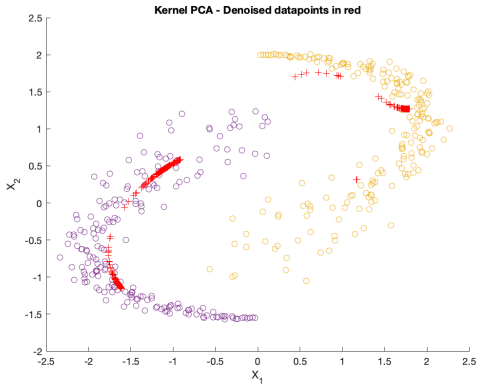


(a) linear PCA with $n_c = 8$

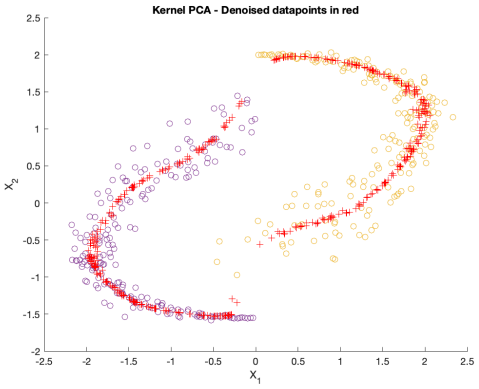


(b) kernel PCA with $n_c = 8$

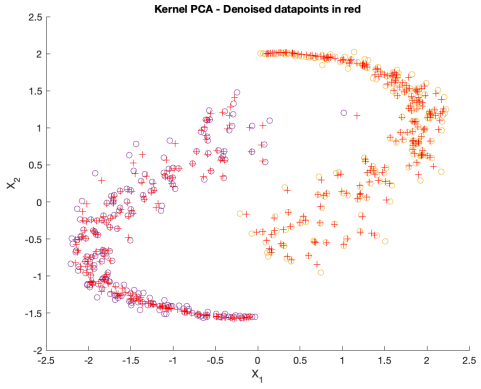
Figure 27: Illustration of the (a) linear and (b) kernel PCA. Kernel PCA is able to capture the structure while denoising in contrast to linear PCA.



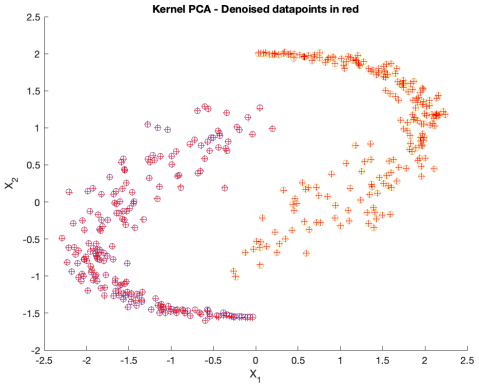
(a) kernel PCA with $n_c = 2$



(b) kernel PCA with $n_c = 10$



(c) kernel PCA with $n_c = 20$



(d) kernel PCA with $n_c = 40$

Figure 28: Varying n_c for kernel PCA

3.2 Spectral clustering

In this exercise we experiment with spectral clustering methods and investigate their functionality. Spectral clustering is a form PCA since we perform dimensionality reduction in the feature

space. It utilizes the eigenvectors of the Laplacian matrix in order to group similar elements together. Similarity metrics for the datapoints consist of distance metrics, the most frequently-used one being Euclidean distance. However, in this exercise we utilize the kernel to measure the similarity. In this case, a degree matrix, that consists of the sum of the columns of the kernel matrix, is constructed. The resulting matrix is diagonal and it contains all the information on the instances.

Classification is a supervised technique that aims to predict the labels of given instances. In contrast, spectral clustering is an unsupervised method that does not aim to "classify" the observations, i.e. does not require any class labels, as opposed to classification that requires a labeled training dataset. It follows that spectral clustering also does not require a train-test split of the dataset. To sum up, a classification task aims to label the observations based on the labels that are provided while spectral clustering groups the observations together to form clusters, based on their similarity.

As mentioned previously, σ^2 is the parameter that measures when two points are (dis)similar. Therefore, decreasing or increasing its value affects the clustering results in the following way: (1) a very low value for σ^2 (e.g. eqaual to 0.001) will result in two points to be considered too dissimilar. Conversely, a higher value results in higher similarity between two points. This is illustrated in Figure 29, where the value for σ^2 is varied between 0.001 and 0.02. As we can see in the left column of the figure, for $\sigma^2 = 0.001$ the resulting cluster is one since the instances are considered too disiimilar and can not be clustered together, thus forming only one cluster. In the projection on the feature space, we see that most values are close to zero and the matrix consists mostly of black spaces. In contrast, in the rightmost column we can see that for a value of $\sigma^2 = 0.02$ there exist two distinct clusters and the corresponding matrix contains more values closer to 1.

3.3 Fixed-size LS-SVM

The LS-SVM model usd for function approxmation in the primal representation, as mentioned before, is defined by

$$y(x) = w^T \phi(x) + b \tag{12}$$

In order to solve the problem in the dual representation, the model is described by

$$y(x) = \sum_{k=1}^{N} a_k K(x, x_k) + b \tag{13}$$

where N corresponds to the number of training points and $K(x, x_k)$ describes a kernel function. The primal representation, supposing that x is in a d-dimensional, would consist of $|w| + |b| = d + 1$ variables (unknown) considering that w would also be n-dimensional. In the dual representation there would be $|a| + |b| = N + 1$ unknown variables. Given a problem with N number of training points in an d dimensional space, we distinct the preferred space into two different cases: (1) *primal* if $d < N$ and (2)*dual* if $N < d$.

In the first case (primal) we have applications with large training sets (N) but not a large number of features for each datapoint d, therefore the amount of unkonw variables is $d+1 < N$. Conversely, in the second case (dual), we have $N+1 < d$ unkonw variables, therefore the training set N is smaller and features d belong to a higher dimensional space.

As discussed previously, the parameter σ^2 affects the RBF kernel in the sense that lower values will consider datapoints as less similar whereas higher values will do the opposite. This is illustrated in Figure 30. Low values for σ^2 will lead to lower entropy thus making the entropy criterion easier to be satisfied. Higher values will accordingly lead to higher entropy.

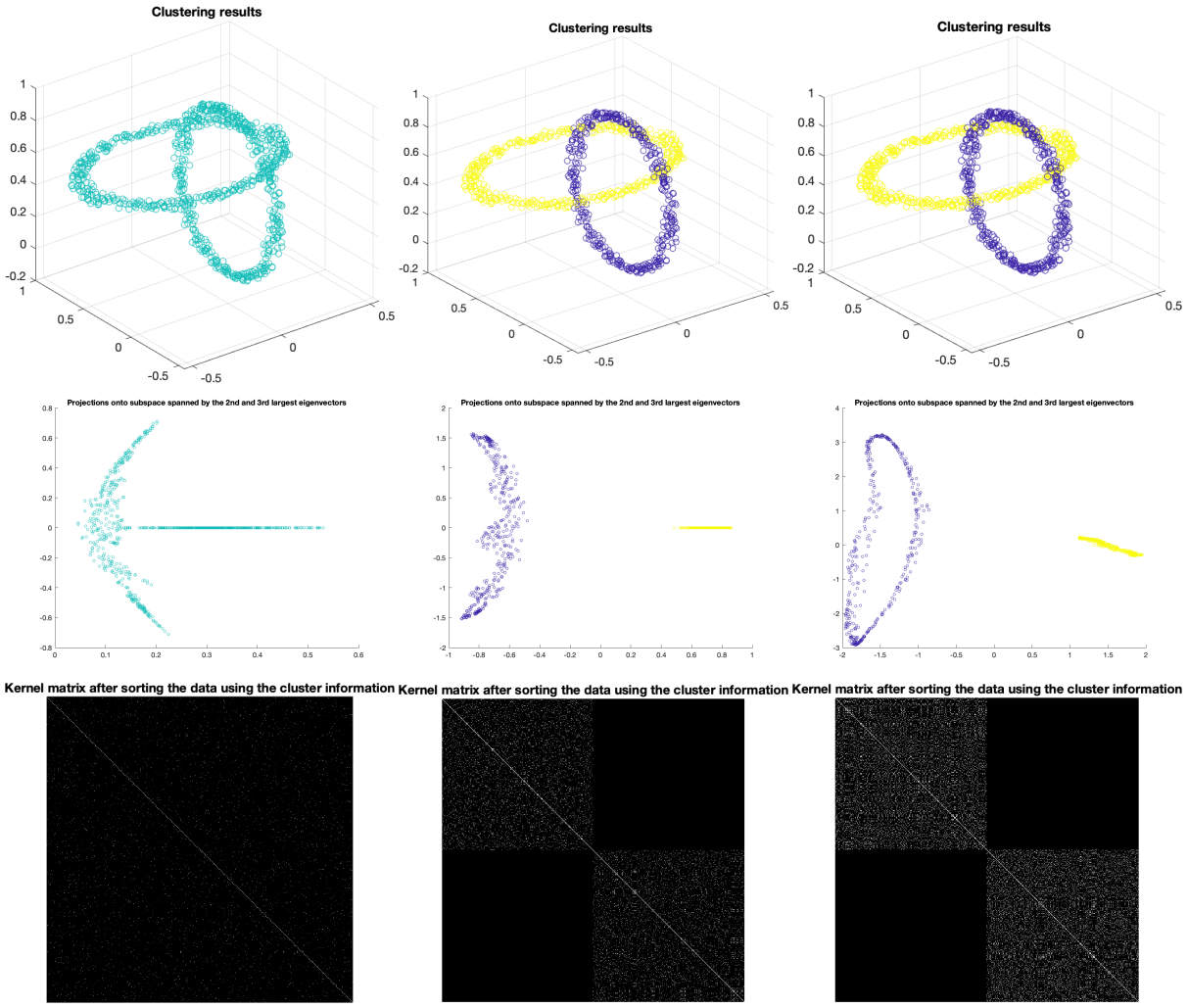
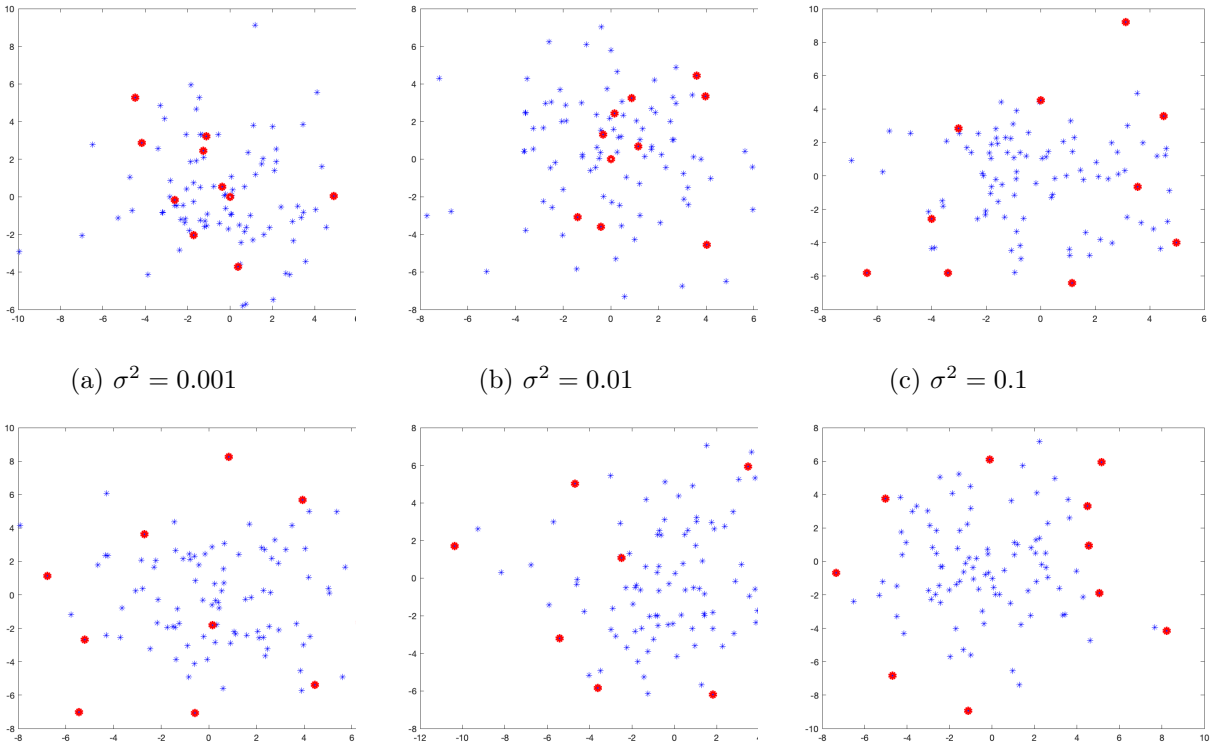


Figure 29: Varying the value of σ in spectral clustering



(a) $\sigma^2 = 0.001$ (b) $\sigma^2 = 0.01$ (c) $\sigma^2 = 0.1$

(d) $\sigma^2 = 1$ (e) $\sigma^2 = 10$ (f) $\sigma^2 = 100$

Figure 30: Varying the value of σ^2 in fixed LS-SVM

In the final part of this exercise we compare the results of a fixed-size LS-SVM to λ_0 model. This comparison is performed on the Breast Cancer Dataset. The results are presented in Figure 31 for $k = 10$ and Figure 32 for $k = 20$. As we can observe, both perform similarly in terms of time in both cases. Additionally, we can see that in both cases the number of support vectors is lower for the λ_0 therefore the solution obtained is more sparse than the one that LS-SVM produces. Finally, for $k = 10$ the error rate of λ_0 is lower whereas for $k = 20$ it is larger than LS-SVM.

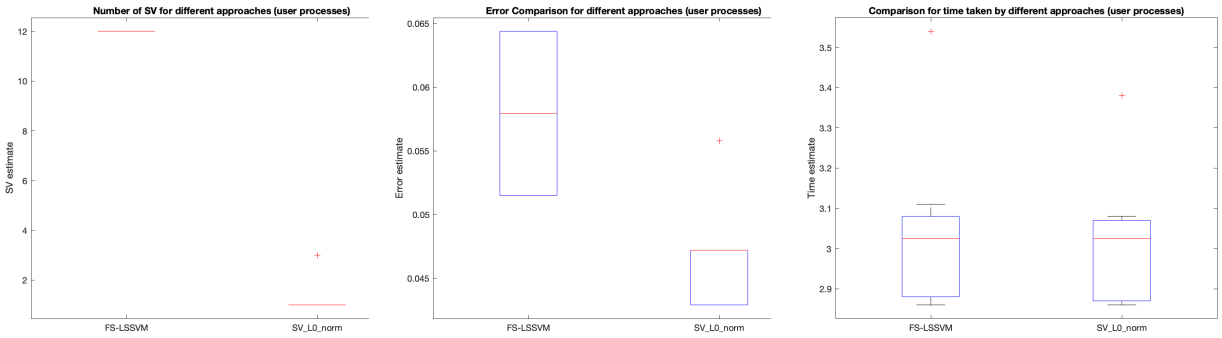


Figure 31: $k = 10$

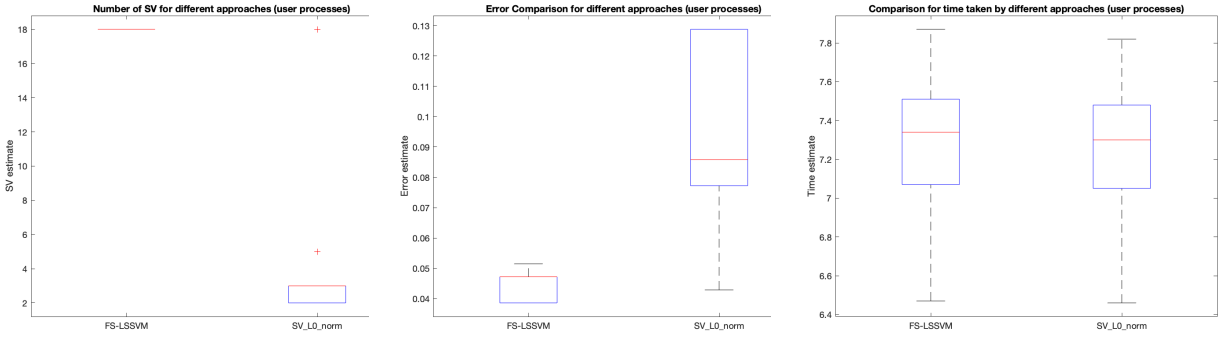
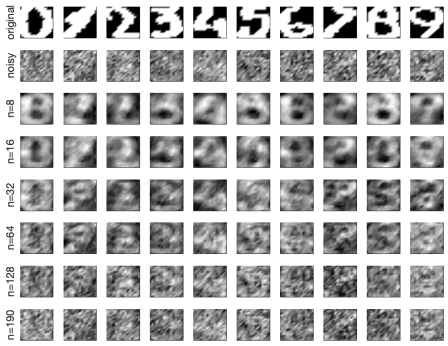


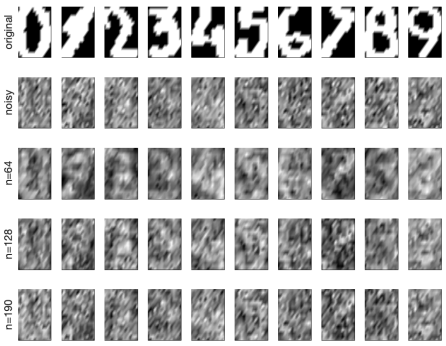
Figure 32: $k = 20$

3.4 Homework problems: Kernel principal component analysis

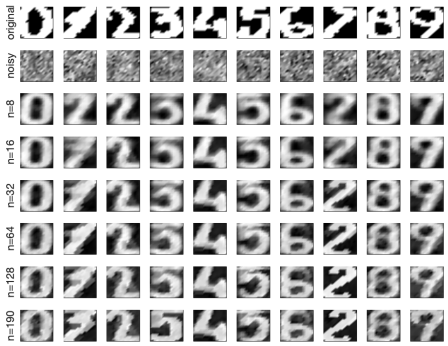
The objective of this exercise is to use kernel PCA to perform analysis on the Digits dataset. This dataset consists of 198 images of handwritten digits, each one having a size of 16×15. The test set and the validation set consist of 20 images each, which are represented as one-dimensional vectors with 240 features each. In this experiment, linear and kernel PCA are trained on the data with a noise factor of 1. Figure 33 illustrates the performance of linear and kernel PCA with number of components n_c equal to 8 and 64. As we can observe, linear PCA performs slightly better for smaller n_c but for larger values it is not able to denoise. On the other hand, kernel PCA can perform significantly better than linear for a larger n_c.



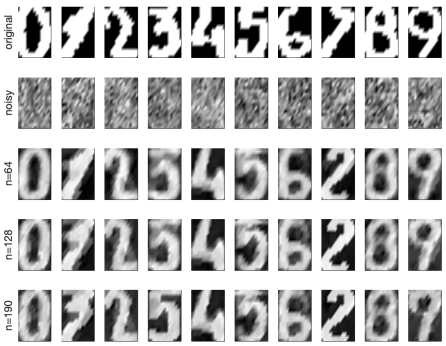
(a) linear PCA with $n_c = 8$



(b) linear PCA with $n_c = 64$



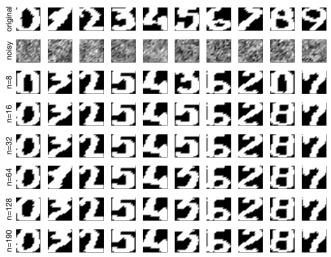
(c) kernel PCA with $n_c = 8$



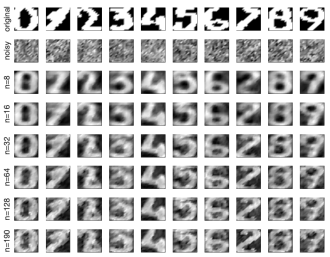
(d) kernel PCA with $n_c = 64$

Figure 33: Varying n_c for linear (left) and kernel PCA (right)

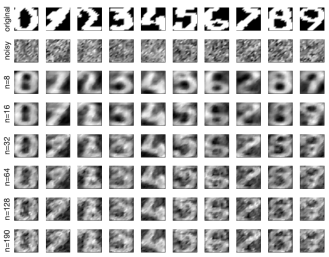
Varying the value of σ^2 has a significant effect in denoising the images. Specifically, we observe that very low values the principal components seem to not contain meaningful informations and that the images are considered to be more dissimilar.This results in the noise being mapped to the wrong image since they principal components contain information that is too specific for the task of denoising to be efficiently performed. In contrast, very high values yield too much information on the principal components, therefore the model is not able to generalize well. In the case of very small values we tend to have high bias whereas very large values lead to high variance. Figure 34 illustrates this situation.



(a) $\sigma^2 = 0.3$



(b) $\sigma^2 = 70$



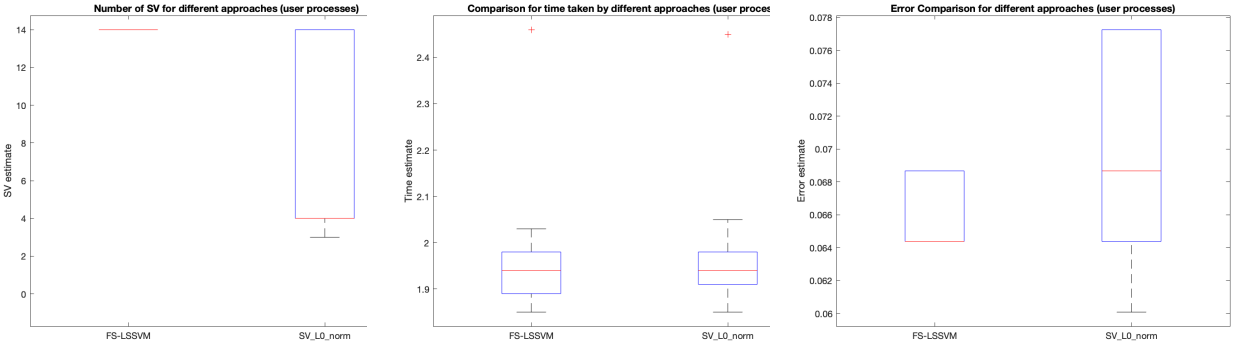
(c) $\sigma^2 = 150$

Figure 34: Varying σ^2 for kernel PCA

3.5 Homework problems: Fixed-size LS-SVM

3.5.1 Shuttle dataset

The objective of this exercise is to perform classification on the Shuttle dataset. This dataset consists of approximately 58000 datapoints each of which has 9 features and 6 different classes. It is an imbalanced since the vast majority of the datapoints belong to the first class (approximately 80%) while only 10 instances belong to class 6. After performing classification on the dataset, we can see the results in Figure 35.

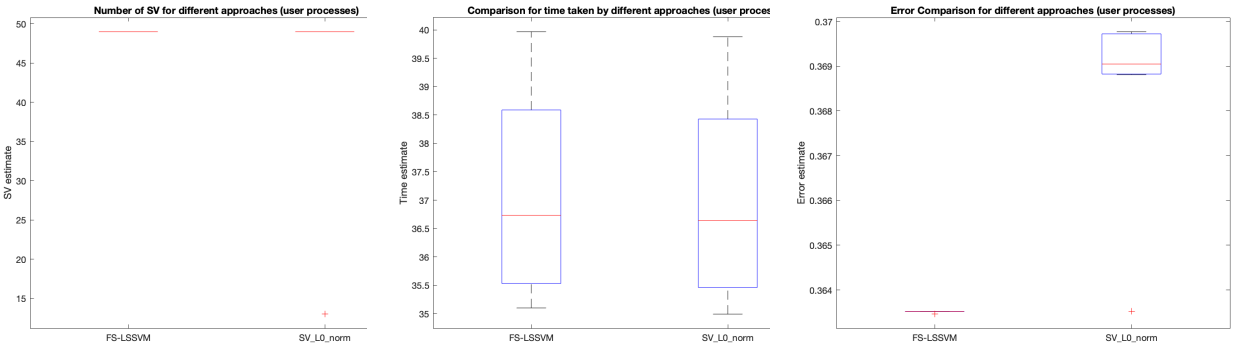


(a) Support vectors (b) Time (c) Error

Figure 35: Resuls of classification on Shuttle dataset

3.5.2 California dataset

The objective of this exercise is to perform regression on the California dataset, which consists of 20640 instances with 9 feature variables available for each. After experimenting with different values of k, which is the size of the sampling that will be applied over the whole data, it was observed that the RBF kernel was significantly slow, even for the minimum value. Therefore, the linear kernel is utilized and the value of k is 5. The results are illustrated in Figure 36.



(a) Support vectors (b) Time (c) Error

Figure 36: Resuls of classification on the California dataset

References

[1] K. D. Brabanter et al. "LS-SVMlab Toolbox User's Guide version 1.7". In: 2003.

[2] Tony Van Gestel et al. "Bayesian framework for least-squares support vector machine classifiers, Gaussian processes, and kernel Fisher discriminant analysis". eng. In: *Neural computation* 15.5 (2002), pp. 1115–1148. ISSN: 0899-7667.

[3] Marcelo Espinoza et al. "Time series prediction using ls-svms". In: (Jan. 2008).