Support Vector Machines: Methods and Applications

Assignments
Session 1: Classification
Session 2: Function Estimation and Time Series Prediction
Session 3: Unsupervised Learning and Large Scale Problems

Alexander J. Lindhardt
Student number: r0826077

KU Leuven
May 25, 2021

Contents

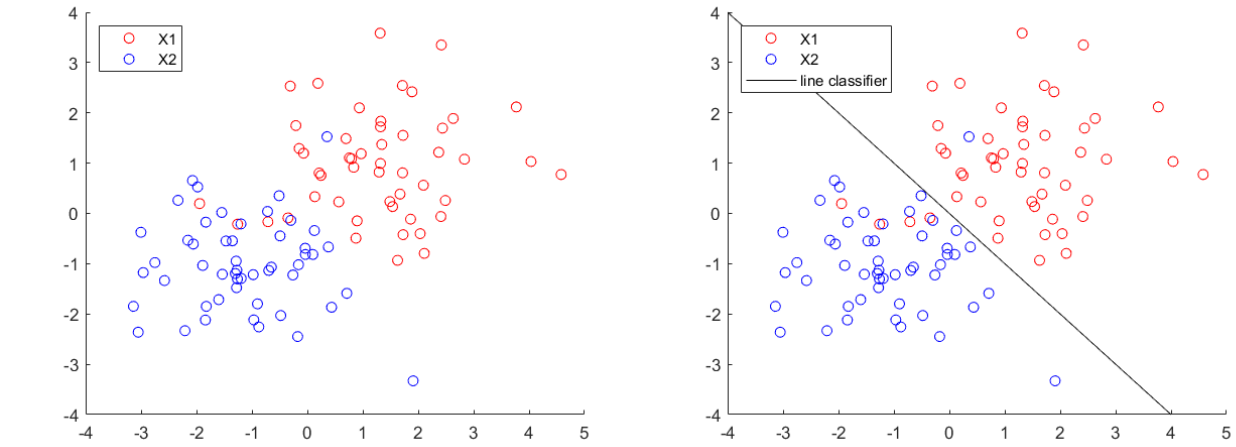
1 Exercise Session 1: Classification

1.1 A simple example: two Gaussians

A dataset consists of two 2-dimensional datasets $X_1 \sim \mathcal{N}(1,1)$ and $X_2 \sim \mathcal{N}(-1,1)$ generated from Gaussian distributions. Note that these have the same covariance matrices. Correspondingly, the class variables Y are defined such as the points from X_1 having $Y = 1$ and the points from X_2 having $Y = -1$. The dataset is visualized in figure 1a.

Q1: *Obtain a line to classify the data by using what you know about the distributions of the data. In which sense is it optimal?*

A1: The optimal line that classifies the data in this case is shown in figure 1b. Since the two classes have the same covariance matrix, we can obtain such a line that minimizes the error rate and allow some overlap.



(a) (b)

Figure 1: (a) The dataset of two classes distinguished by color. (b) The optimal line classifier between the two classes in the dataset.

1.2 Support vector machine classifier

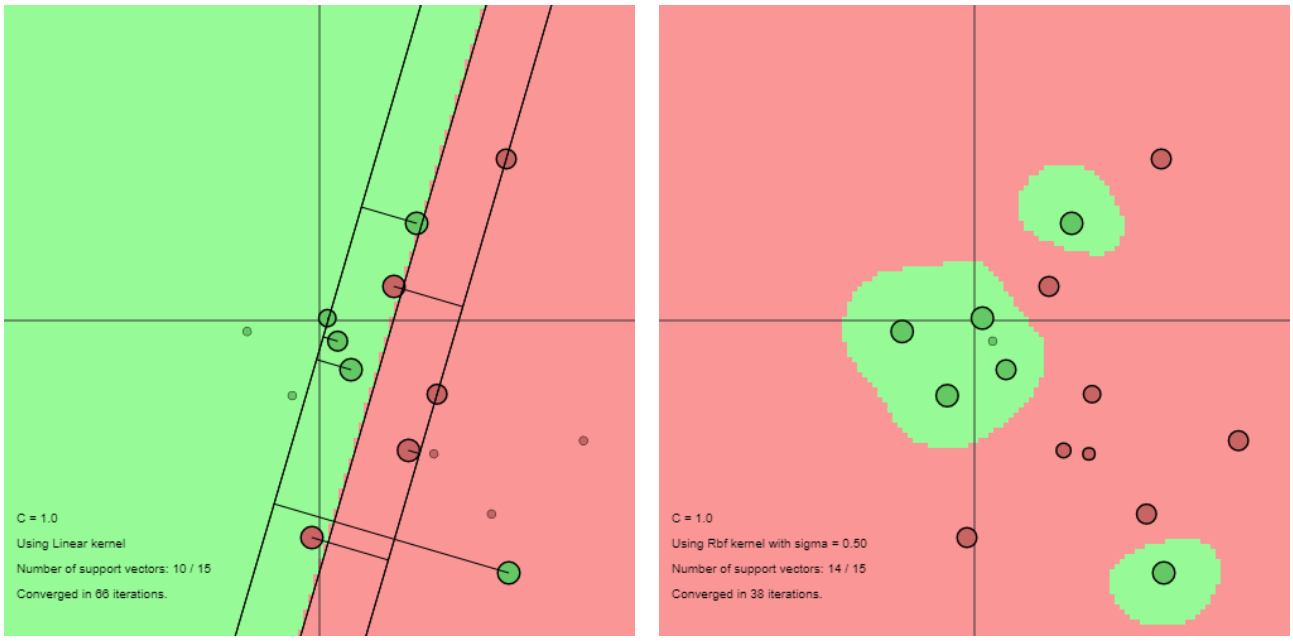
A support vector machine (SVM) classifier is applied using the demo from https://cs.stanford.edu/people/karpathy/svmjs/demo/. Experiments are done to see how the classification depends on the dataset, kernel and hyperparameters.

Q2: *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. Note that a support vector is indicated by a large circle with bold-lined circumference and that its importance is proportional to the size in the online application.*

A2: The support vectors are a subset of the dataset that normally are geometrically closest to the decision boundary, they also directly influence the optimal location on the decision boundary. When using a linear kernel, the support vectors are the points inside or on the margin as well as the misclassified points. Using the radial basis function (RBF) kernel, it usually uses a bigger subset of the data points as support vectors. See the difference between these kernels in figure 2 where we see how the points that are support vectors changes using the same dataset. The importance of a support vector changes if the decision boundary alters. This can happen for example when changing a hyperparameter or adding data points.

Q3: *What is the role of parameters C and σ? What happens to the classification boundary if you change these parameters. Illustrate visually.*

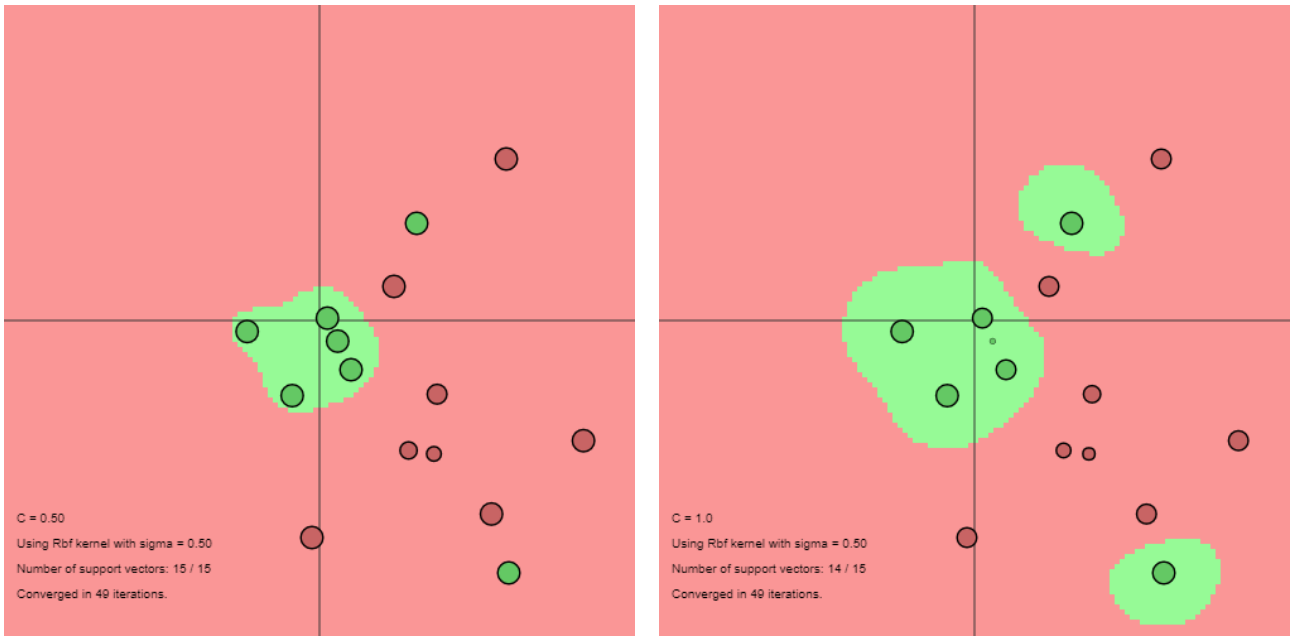
A3: The parameter C is a regularization parameter that is a measure of how much the SVM is allowed to misclassify data points. A smaller value of C allows for more misclassification and therefore also a larger margin, the opposite is true for higher values of C. The points is that there is a trade-off between correctly



(a) (b)

Figure 2: SVM classifier with linear kernel (a) and RBF kernel (b) on the same dataset.

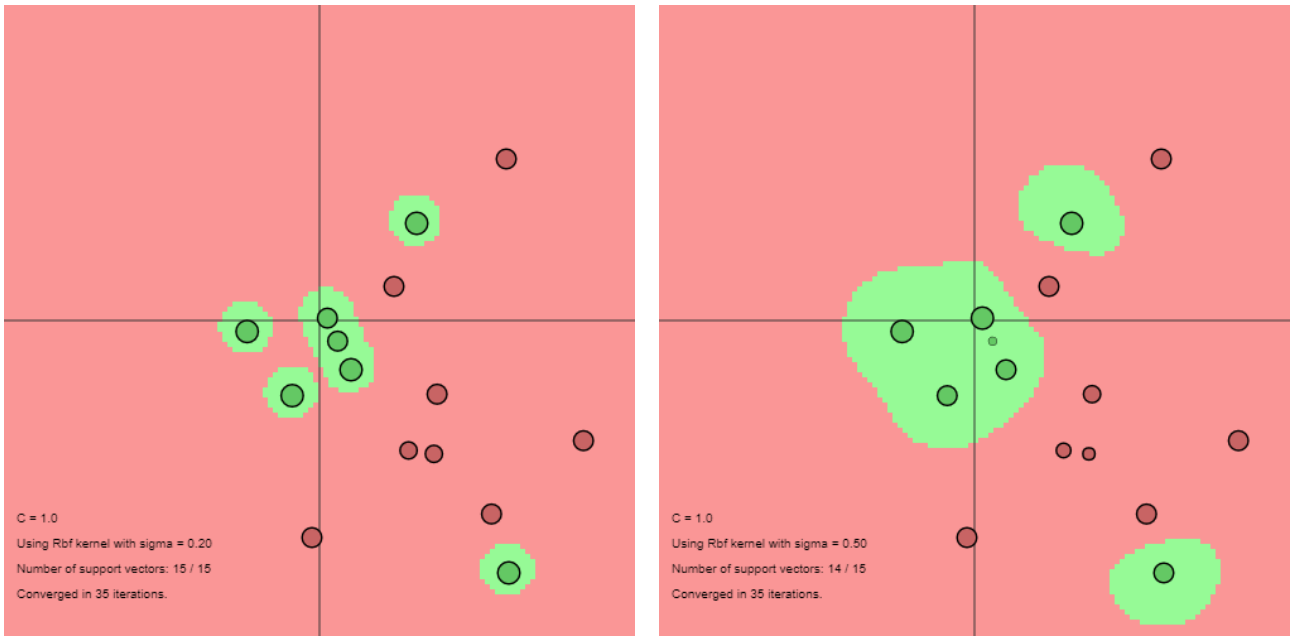
classifying each point in the dataset and having a small margin. An example is shown in figure 3 where an RBF kernel is used with different values of C. When the value of the parameter is set to lower, the SVM will allow for more misclassification and encourage a larger margin and therefore a simpler decision function. In this case, it makes the SVM worse at classifying the dataset.



(a) (b)

Figure 3: SVM classifier with RBF kernel, $\sigma = 0.5$ and $C = 0.5$ (a) and $C = 1$ (b) on the same dataset.

The parameter σ can be seen as a measurement on how much influence a single data point reaches. A small value of σ makes the influence of the support vectors to only include itself which will lead to overfitting. Meanwhile, a larger value of σ will make the decision boundary more linear and the influence of the support vectors would include the whole dataset and therefore lead to underfitting. A visualization of this is shown in figure 4.



(a) (b)

Figure 4: SVM classifier with RBF kernel, $C = 1$ and $\sigma = 0.2$ (a) and $\sigma = 0.5$ (b) on the same dataset.

Q4: *What happens to the classification boundary when sigma is taken very large? Why?*

A4: As mentioned before, a high value of σ makes the decision boundary close to linear when using an SVM with an RBF kernel, see figure 5. This happens since every data point will have strong influence on the boundary and it will therefore separate the centers of high density of the two classes.

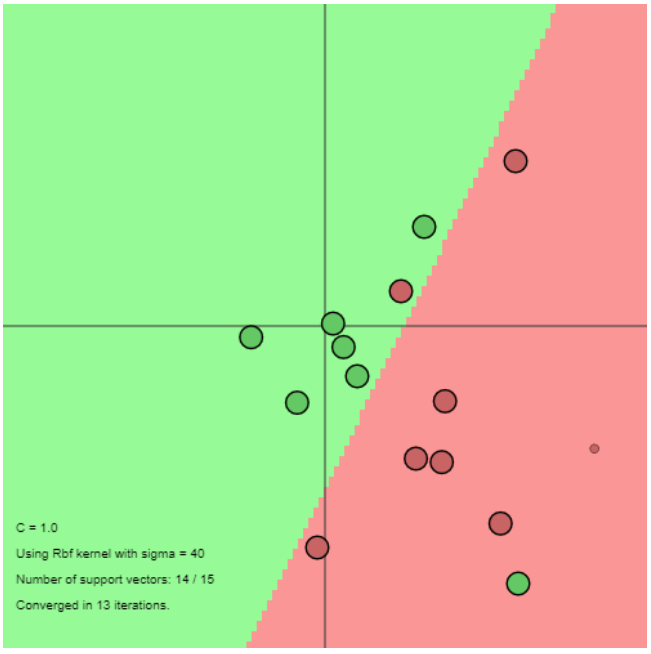


Figure 5: An SVM classifier with RBF kernel and a high value of σ.

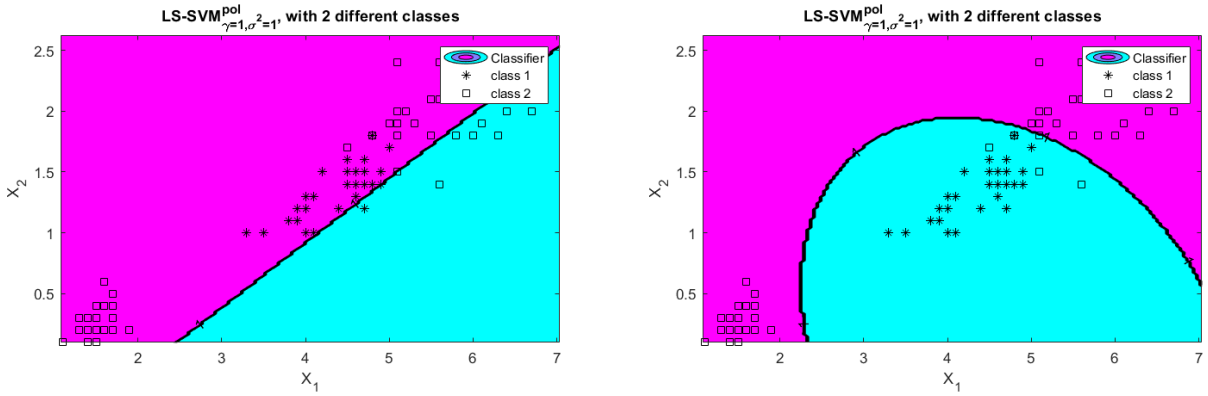
1.3 Least-squares support vector machine classifier

A Least-squares support vector machine (LS-SVM) classifier will be used on the **iris** dataset. Different kernels and hyperparameters will be used to compare performance in the following sections.

1.3.1 Influence of hyperparameters and kernel parameters

Q5: *Try out a polynomial kernel with degree = 1, 2, 3, . . . and t = 1 (fix $\gamma = 1$). Assess the performance on the test set. What happens when you change the degree of the polynomial kernel?*

A5: To see how the performance changed depending on the degree of the polynomial kernel, an LS-SVM was trained for each degree value in the interval $[1, 10]$. They were thereafter evaluated based on the test set and how many wrong classifications were made. We saw that a polynomial kernel with degree 1 had an error rate of 55%. A kernel with degree 2 had 5% and thereafter for $2 < degree \leq 10$ the error rate was 0%. Plots of the LS-SVM results in the environment of the training data with degree 1, and 2 are shown in figure 6.

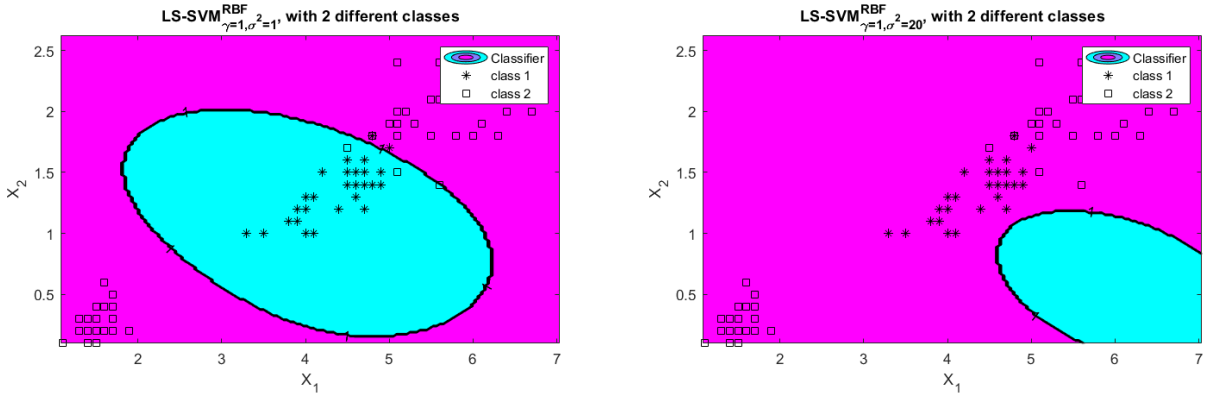


(a) (b)

Figure 6: LS-SVM with polynomial kernel of degree 1 (a) and degree 2 (b).

Q6: *Let's now focus on the RBF kernel with squared kernel bandwidth σ^2. Try out a good range of σ^2 values as kernel parameters (fix $\gamma = 1$). Assess the performance on the test set. What is a good range for σ^2? Fix a reasonable choice for σ^2 and compare the performance using a range of γ. What is a good range for γ?*

A6: Similar as before, an LS-SVM with RBF kernel with fixed $\gamma = 1$ and varying σ^2 over the values $[0.01, 0.1, 1, 5, 10, 25]$ was trained on the training data and then evaluated on the test set. The error rate was 0% for the σ^2-values of $[0.1, 1, 5, 10]$ and thereafter raised to 50% for $\sigma^2 = 25$ and 10% for $\sigma^2 = 0.01$. Figure 7 shows the LS-SVM with $\sigma^2 = 1$ and $\sigma^2 = 20$. So a good range for σ^2 is between 0.1 and 10.



(a) (b)

Figure 7: LS-SVM with RBF kernel with $\sigma^2 = 1$ (a) $\sigma^2 = 20$ (b).

Now fixing $\sigma^2 = 1$, we vary γ in the interval $[0.01, 0.1, 1, 5, 10, 25]$ and decide on a good range for this parameter, same procedure as before is done. Now we see that the error rate is 50% with $\gamma = 0.01$ and 0% for the rest. Looking at the plots of the LS-SVM for the different γ values we decide that a good range for γ when $\sigma^2 = 1$ is between 0.1 and 10.

Q7: *Compare your results with SampleScript_iris.m, which is available on Toledo*

A7: The results and findings are very similar compared to the previous analysis.

1.3.2 Tuning parameters using validation

Instead of using the test set to evaluate the performance of the LS-SVM, we can instead split up the training set such that a part of the training set is used as a validation set.

Q8: *Compute the performance for a range of γ and σ^2 values. Use the random split method, 10-fold crossvalidation and leave-one-out validation. Visualize the results of each method: do you observe differences? Interpret the results: which values of γ and σ^2 would you choose?*

A8: When doing all these validation methods, we use the same value ranges for γ and σ^2 which is $\gamma = [0.01, 0.1, 1, 10, 100, 1000]$ and $\sigma^2 = [0.001, 0.01, 0.1, 1, 10, 100, 1000]$. Each validation is then run on these values and the validation accuracy for these sets of values are plotted against each other. First off all, for random split we use 20% of the data for validation. The validation accuracy for ranges of values for γ and σ^2 is shown in figure 8.

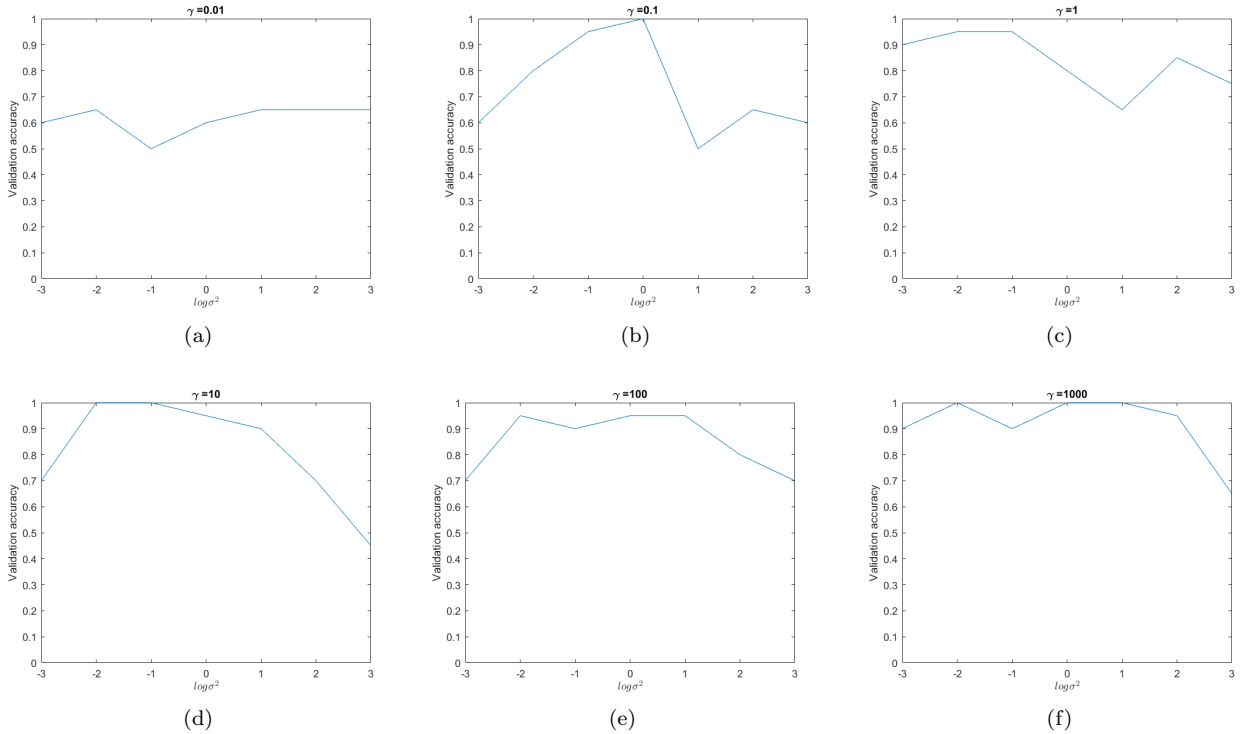


Figure 8: Random split validation with 20% of the training data as validation set. Validation accuracy is shown on the $y-$axis for different values of σ^2 and γ.

Secondly, crossvalidation with 10 folds on the training set is visualized in figure 9 and thirdly leave-one-out validation is shown in figure 10. What we can see is that 10-fold crossvalidation and leave-one-out validation gives almost identical validation accuracies whereas the random split method behaves a bit different although has a similar pattern as the other two. Looking at all three validation methods it looks like $\gamma = 100$ has the best validation accuracy where σ^2 can range between 0.1 and 100.

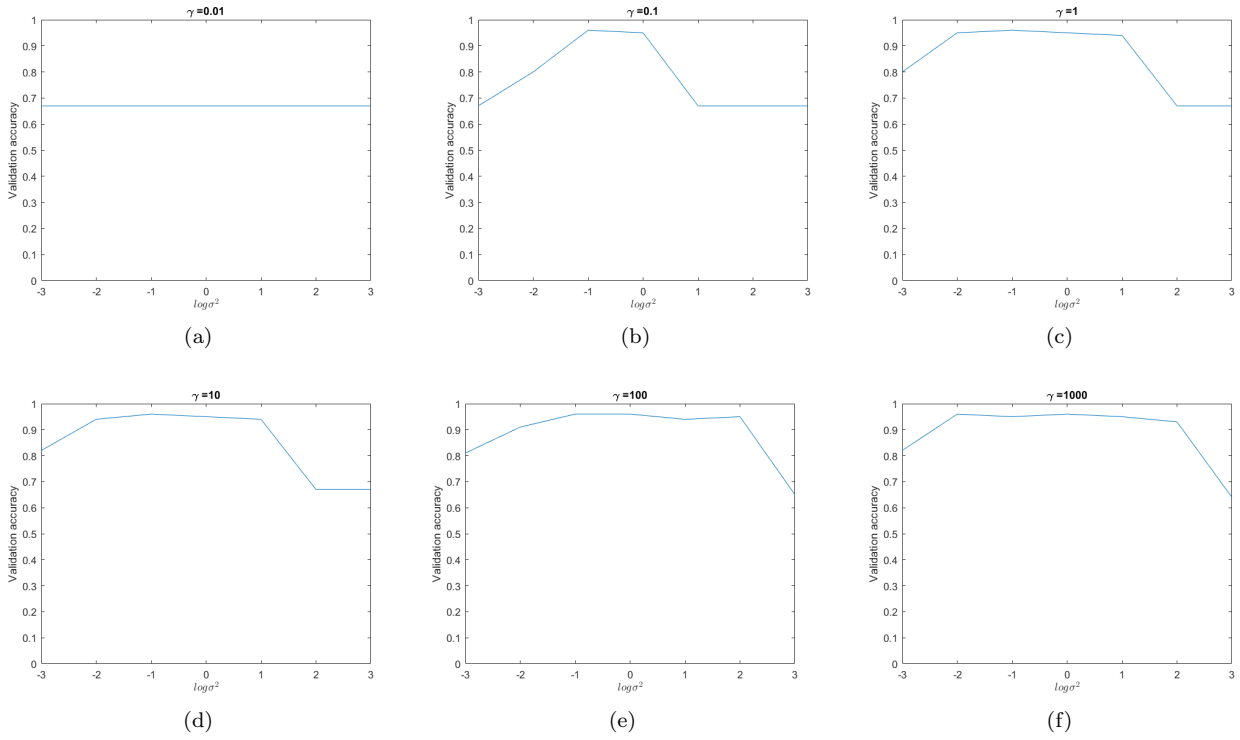


Figure 9: Crossvalidation with 10 folds. Validation accuracy is shown on the $y-$axis for different values of σ^2 and γ.

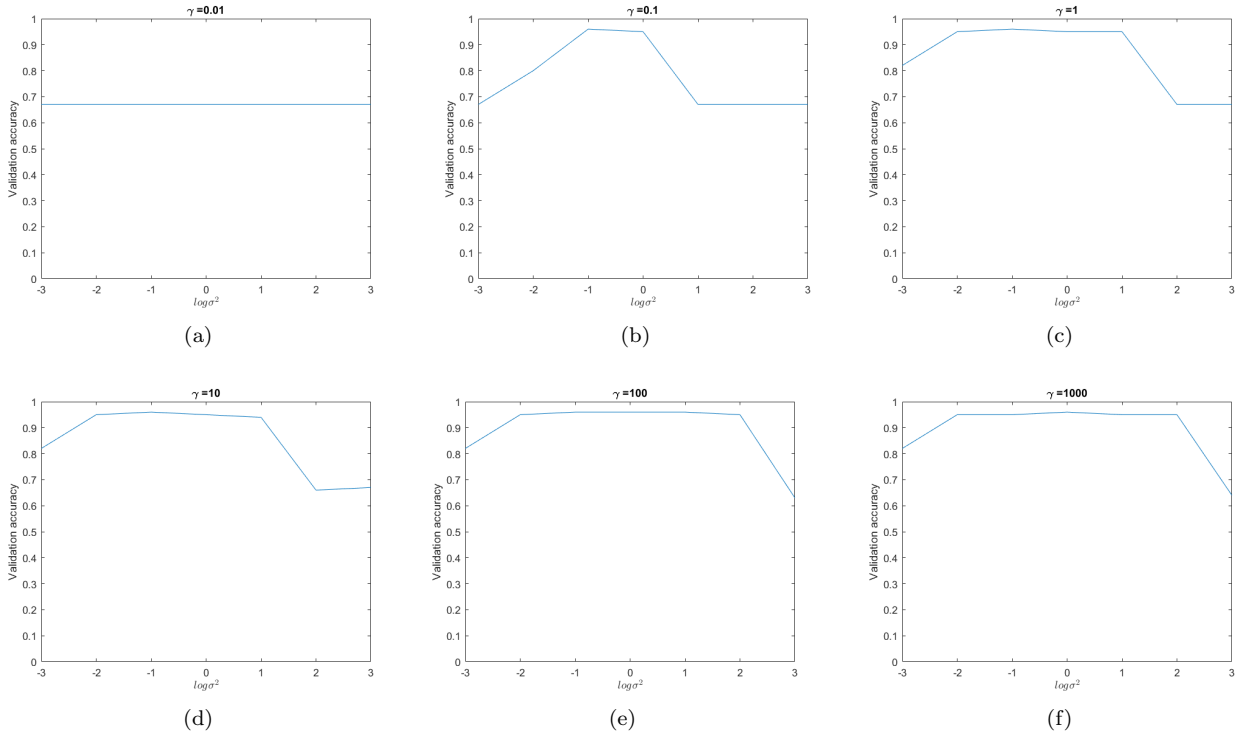


Figure 10: Leave-one-out validation. Validation accuracy is shown on the $y-$axis for different values of σ^2 and γ.

Q9: *Why should one prefer crossvalidation over simple validation (random split)? How to choose the value of k in k-fold crossvalidation?*

A9: Using random split as validation method means that the model trains on less data than what is available since it's not possible to train on the validation set since that would lead to a bias. However, k-fold crossvalidation trains and validates on all data, just in k iterations, which means that all data is used for training. This can be very important when the training set is of a limited size and also to get a better estimate of the model's actual performance. This is due to the fact that crossvalidation averages the validation accuracy in all k folds. Choosing the k depends on the size of the available training data. You want sufficient variance in the both the training and validation set each time so if the dataset is small, it makes sense to not use too many folds.

1.3.3 Automatic parameter tuning

The hyperparameters can be automatically tuned using LS-SVMlab toolbox.

Q10: *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.*

A10: to compare the two algorithms, we run both 100 times and average the classification rate and track the computation time, see table 1. There we see that the gridsearch algorithm is a tiny bit better but takes more than double the amount of time. For this example, we can use gridsearch without a problem but for a much bigger model, that could take a lot of time compared to the simplex method. The parameter values we retrieve varies a lot from each iteration. As we have seen before, there are a lot of combinations of the parameters that yield a good result, that means that the methods will find different local minimums each time it runs.

algorithm	misclassification rate	time [s]
gridsearch	0.0339	78.24
simplex	0.0357	29.08

Table 1: Table showing the average misclassification error and time elapsed when using different tuning algorithms.

1.3.4 Using ROC curves

Another way to evaluate the performance of a model is to look at the Receiver Operating Characteristic (ROC) curve. What we look at is the area under the curve, a big area implies a good classifier.

Q11: *In practice, we compute the ROC curve on the test set, rather than on the training set. Why?*

A11: Since the training set has been used to train the model, it is not a good idea to test performance on that same dataset. Since this doesn't test the model's generalizability, however a test set that the model has never seen can be used for this.

Q12: *Generate the ROC curve for the iris.mat dataset (use tuned γ and σ^2 values). Interpret the result*

A12: A gridsearch was used to get tuned parameters. Then the ROC curve was plotted, see figure 11, there we see that the area is maximized which has do with the fact that the accuracy on the test set is 100%.

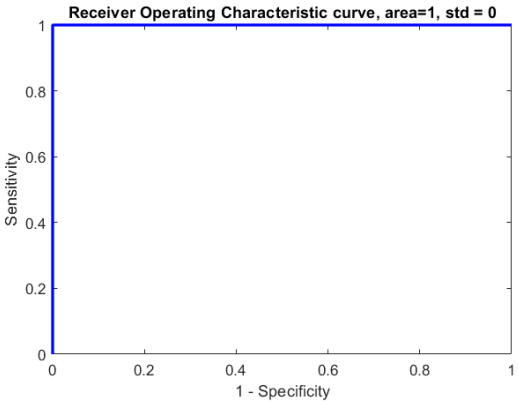


Figure 11: ROC curve on the **iris** test set.

1.3.5 Bayesian framework

Q13: *How do you interpret the colors of the plot?*

A13: The plot can be seen in figure 12 with tuned parameters. The colors can be interpreted as the probability of belonging to the first class. Here we see that pink color implies a high chance of belonging to the first class and the blue color implies a low chance of belonging to that class and therefore a high chance of belonging to class 2.

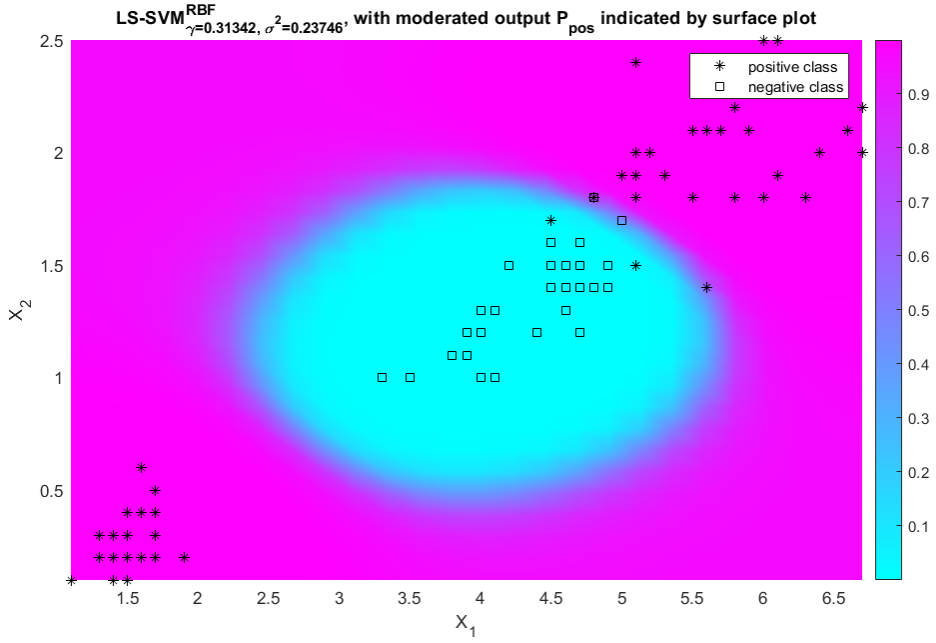


Figure 12: Bayesian framework plot for the **iris** dataset with tuned parameters.

Q14: *Change the values of γ and σ^2. Visualize and discuss the influence of the parameters on the figure.*

A14: We now use parameters that we know perform worse from the previous analysis of parameter tuning. The resulting Bayesian framework is visualized in figure 13 where we now see that some points has a high chance of belonging to the wrong class. So the parameters have a strong influence over how the Bayesian network looks.

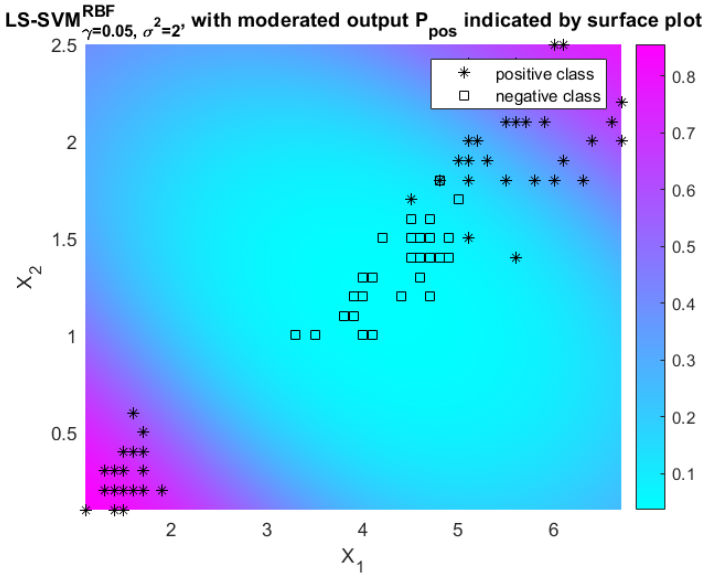


Figure 13: Bayesian framework plot for the **iris** dataset with bad parameters chosen deliberately.

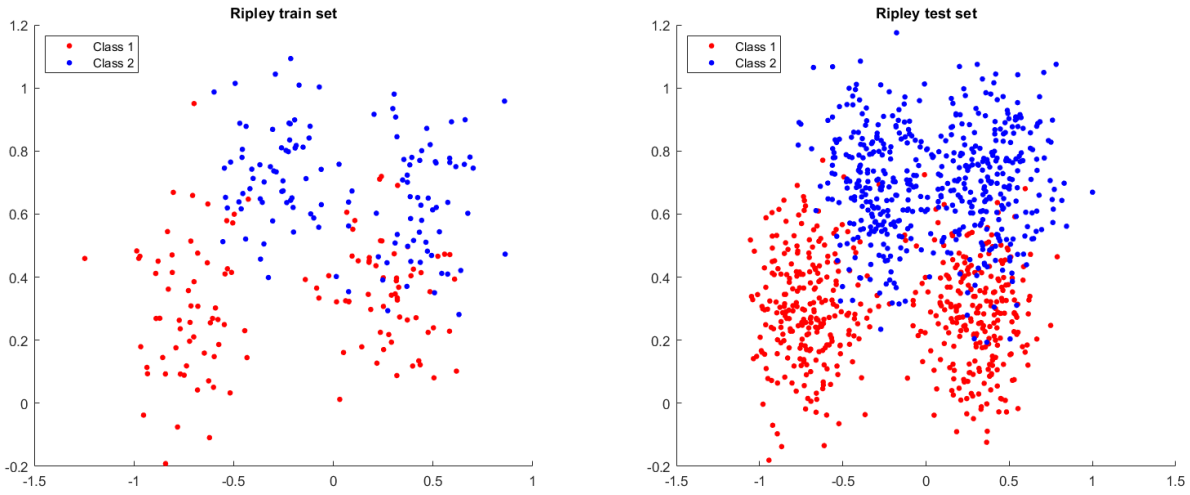
1.4 Homework problems

We are now going to look at three datasets and for each dataset answer and execute the following questions:

- Visualize the data. Inspect the data structure: what seems to be important properties of the data? Which classification model do you think you need, based on the complexity of the data?
- Try out different models (linear, polynomial, RBF kernel) with tuned hyperparameter and kernel parameters. Compute the ROC curves. Which model performs best? Which model would you choose?
- Are you satisfied with the performance of your model? Would you advise another methodology?

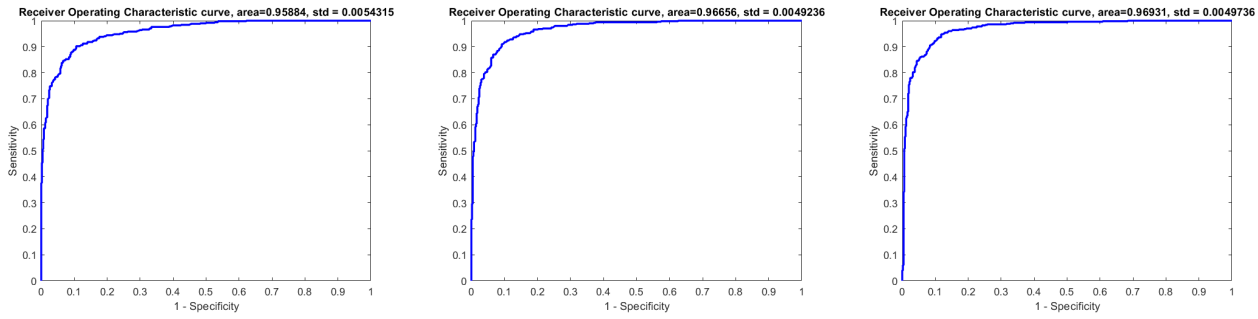
1.4.1 Ripley dataset

The Ripley dataset consists of 2-dimensional data points divided into a training set of size 250 and a test set of 1000 points. These datasets are visualized in figure 14. A notable aspect of the data structure is that the first class lies underneath the second class in the plots, meaning that the second class has in general a higher value on its second variable. This means that even a line would be able to separate the classes quite well. Also, the classes have a few overlapping points but it is quite well clustered within each class. We therefore expect an LS-SVM with an RBF, linear or polynomial kernel to perform quite well on this dataset. The ROC curves for each kernel with tuned hyperparameters are shown in figure 16 where we see that each kernel perform almost equally well and the performance is quite good so this is a suitable methodology for this problem.



(a) (b)

Figure 14: Ripley dataset. (a) training set with class labels and (b) test set with class labels.

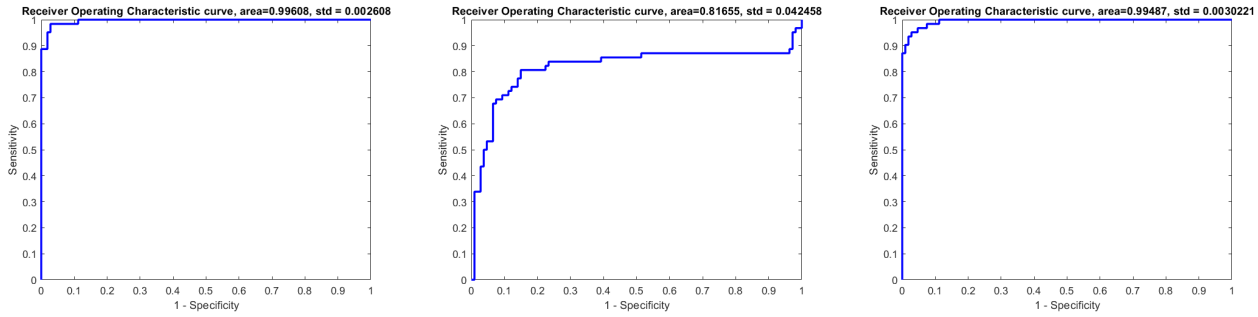


(a) (b) (c)

Figure 15: ROC curves for the Ripley dataset with different kernels with tune hyperparameters. (a) linear kernel (b) polynomial kernel (c) RBF kernel.

1.4.2 Wisconsin Breast Cancer dataset

This dataset consists of two classes with data points of 30 features which makes it difficult to visualize. The training set has 400 data points and the test set contains 169 points. Same as before, we plot the ROC curves for each kernel with tuned hyperparameters, see figure 16. We see that the linear and RBF kernel perform very well meanwhile the polynomial kernel has a worse result.

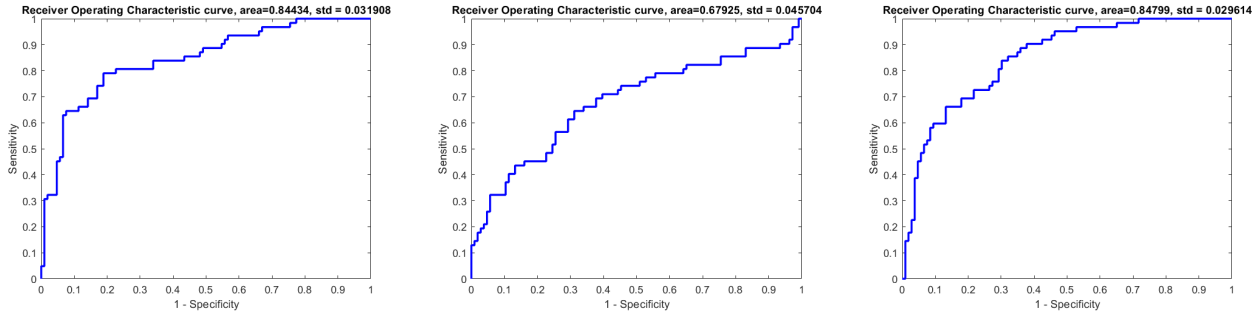


(a) (b) (c)

Figure 16: ROC curves for the Wisconsin Breast Cancer dataset with different kernels with tune hyperparameters. (a) linear kernel (b) polynomial kernel (c) RBF kernel.

1.4.3 Diabetes dataset

This dataset is divided into a training set with 300 points and a test set with 168 points. These data points are from two classes and consists of 8 features which makes it difficult to visualize. But we still look at the performance of a LS-SVM with different kernels on the test set after training on the training set with tuned hyperparameters. The ROC curves are shown in figure 17. As we can see, the performance from all kernels are quite poor although the linear and RBF kernels once again perform the best. It is hard to say why the performance is bad without looking at the dataset, but one can imagine that the classes overlap quite heavily in some of the feature values.



(a) (b) (c)

Figure 17: ROC curves for the Diabetes dataset with different kernels with tune hyperparameters. (a) linear kernel (b) polynomial kernel (c) RBF kernel.

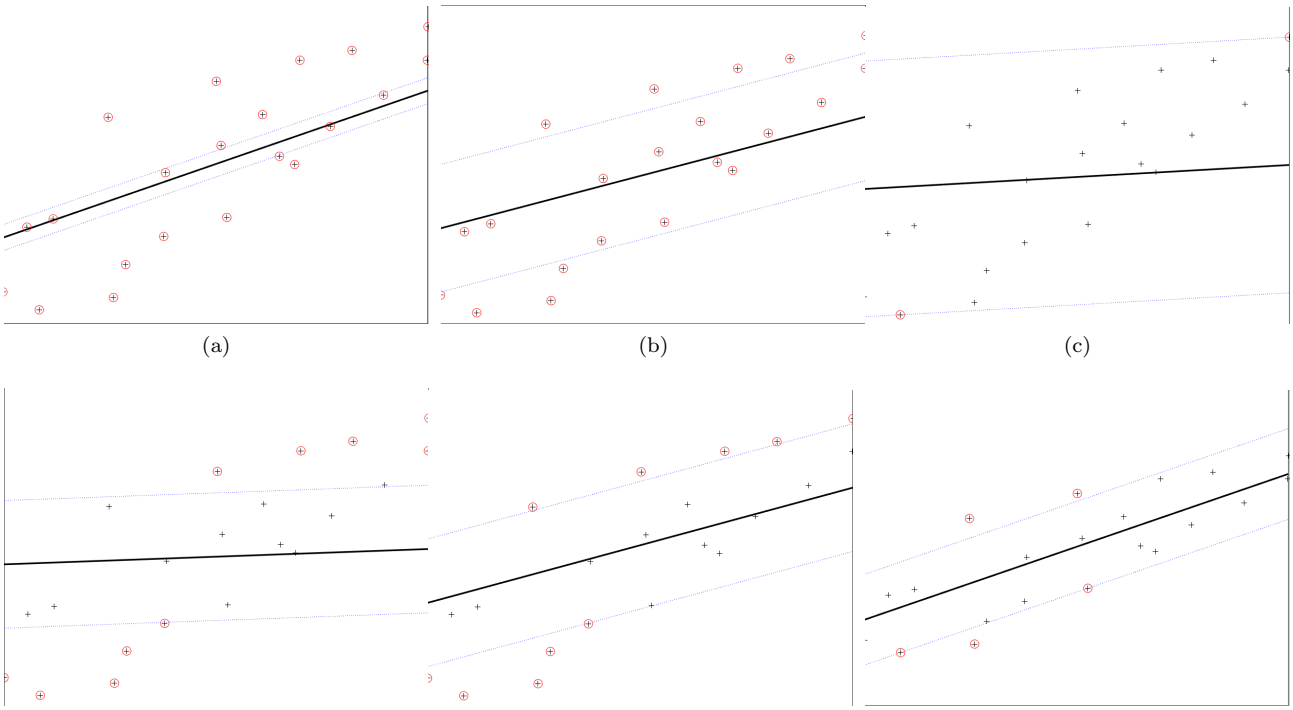
2 Exercise Session 2: Function Estimation and Time Series Prediction

2.1 Support vector machine for function estimation

An SVM will be used for function estimation and to show how this works, we will experiment with the script **uiregress** and tweak the parameters.

Q1: *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?*

A1: A dataset of linear property was constructed and then a SVM with linear kernel was used for regression. The value of e defines the tube where the errors are given no penalty. Changing the parameter e to a higher value means that the model allows larger error and a small value means that each error is penalized resulting in more support vectors. The **bound** parameter is a trade-off parameter between how much the model is allowed to misclassify and the models complexity. It works as a regularization parameter. See figure 18 for the effects of using different parameters on the data with a linear kernel.



(a) (b) (c)

(d) (e) (f)

Figure 18: SVM regression with linear kernel on the same dataset using different hyperparameters. **bound** is fixed for (a)-(c) with e varying from 0.1, 0.5 and 1 respectievly. e is fixed for (d)-(f) with **bound** varying from 0.01, 0.1 and 1 respectievly.

Q2: *Construct a more challenging dataset (around 20 data points). Which kernel is best suited for your dataset? Motivate why.*

A2: Now if we construct a dataset that looks nothing like linear data we expect either the RBF or polynomial kernel to better fit the data. The created dataset can be seen in figure 19 and it is clearly appropriate to use a polynomial kernel with *degree* ≥ 3. The reason being that the constructed dataset looks non-linear and it will therefore not be optimal to use a linear kernel.

Q3: *In what respect is SVM regression different from a classical least squares fit?*

A3: The main difference is that in SVM regression, only a subset of the data points is considered for the

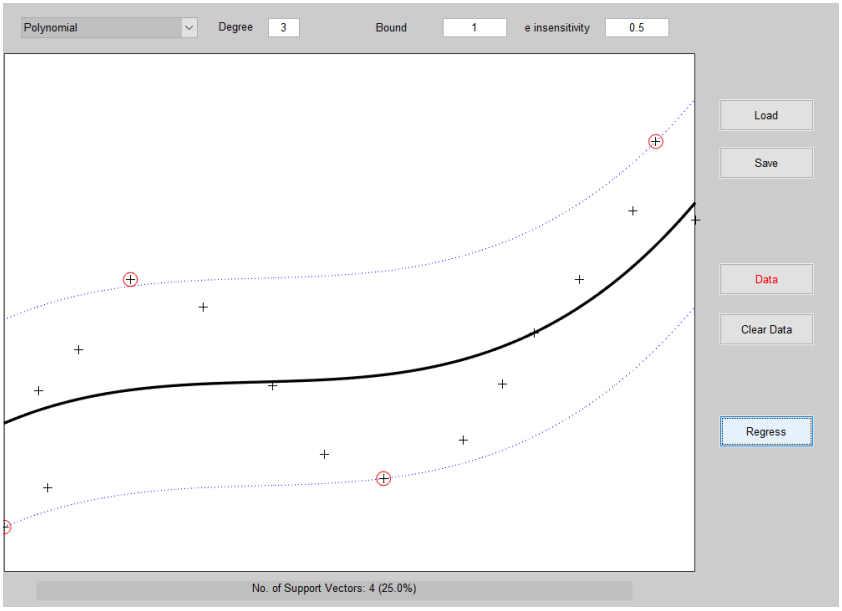


Figure 19: A more challenging dataset with polynomial kernel of three degrees.

function approximation, these are the support vector. Meanwhile in the classical least squares fit, every data point is used for the approximation.

2.2 A simple example: the sinc function

We will continue to work with the least squares based variant of the support vector machine (LS-SVM), using the LS-SVMlab toolbox. The focus will be on the sinc function. An artificial dataset is created from the $y = sinc(x)$ function with added white noise and x is between -3 and 3 with an increment of 0.01. The data points with an odd index is used in the training set and the points with even indices are used as test data.

2.2.1 Regression of the sinc function

An LS-SVM regression model with an RBF kernel will be used in this exercise.

Q4: *Try out a range of different γ and σ^2 parameter values (e.g., $\gamma = 10, 10^3, 10^6$ and $\sigma^2 = 0.01, 1, 100$) 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).*

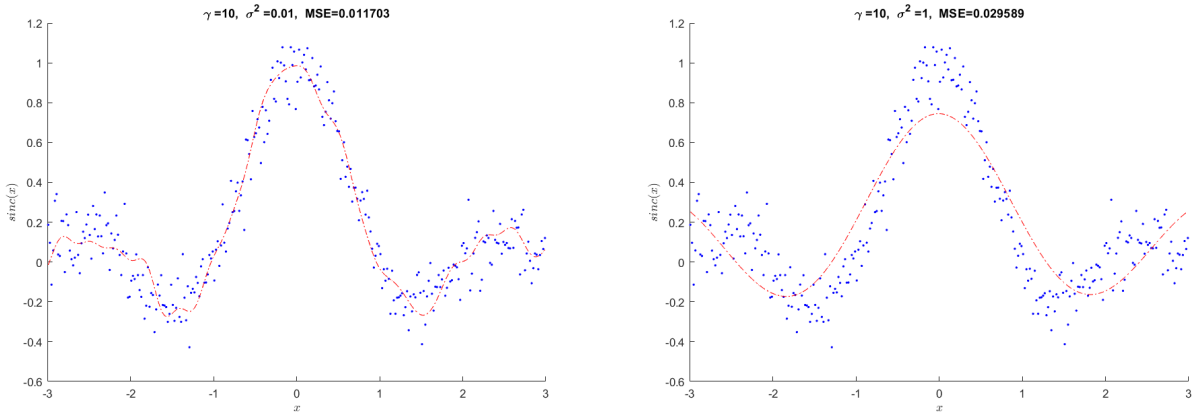
A4: The γ parameter was chosen as 10 and 10^6 and σ^2 as 0.01, 1 and 100. The function estimation together with the test set for these pairs of values are shown in figure 20. Here we can see how the parameters affect the function estimation. For example, when σ^2 is too high the model prioritize a less complex function and can therefore not catch the underlying function. The reverse is true when σ^2 is too low, resulting in a too complex model that overfits on the data points. Increasing the γ parameter from 10 to 10^6 improves the function estimation for each value of γ and we can see that out of all of the selected values that $\gamma = 10^6$ and $\sigma^2 = 1$ has the best performance.

Q5: *Do you think there is one optimal pair of hyperparameters?*

A5: There are most likely many optimal pairs of hyperparameters as we saw in the previous exercise of SVM classification.

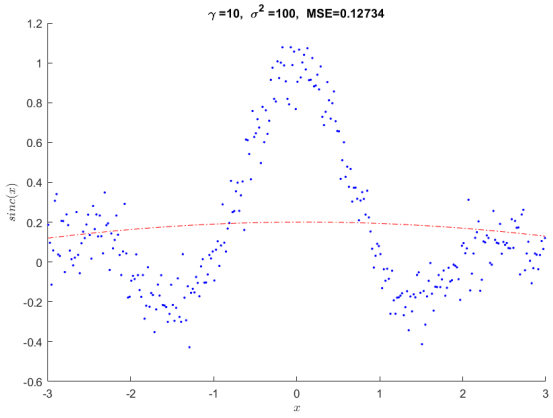
Q6: *Tune the γ and σ^2 parameters using the tunelssvm procedure. Use multiple runs: what can you say about the hyperparameters and the results? Use both the simplex and gridsearch algorithms and report differences.*

A6: Tuning was done using both simplex and gridsearch 50 times each and the mean absolute error was calculated each time and averaged over the runs. The time elapsed for each run was also averaged and the result can be seen in table 2. Both method have similar error although the simplex method is faster. The values for the hyperparameters differed quite a bit between iterations however the σ^2 parameter was quite stable between $0.3 - 0.5$ whereas γ had a bigger range between $1 - 1000$.

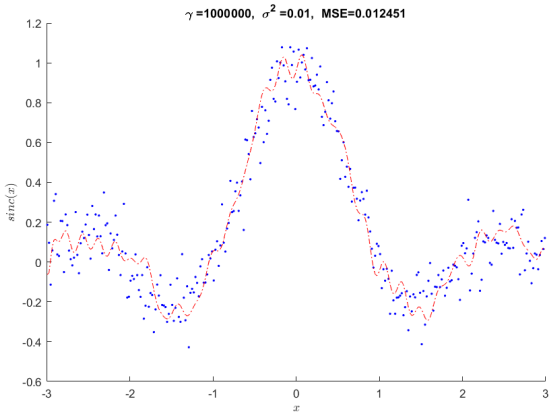


(a)

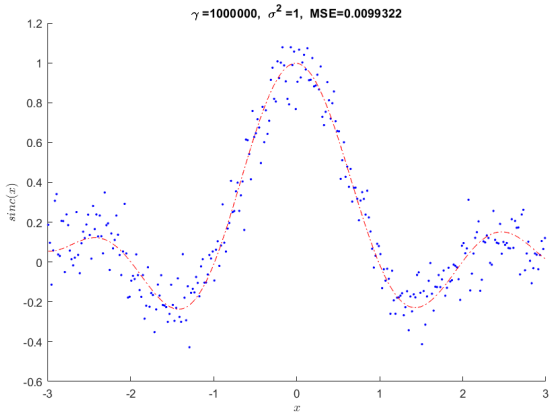
(b)



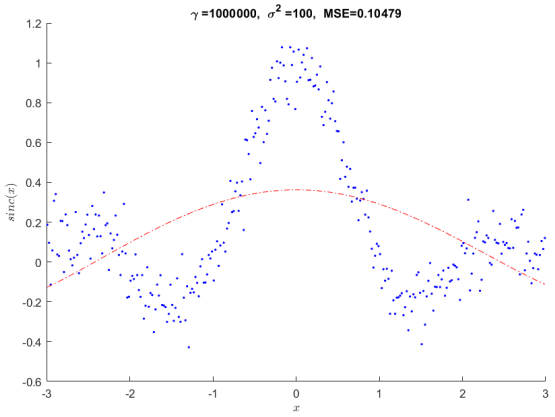
(c)



(d)



(e)



(f)

Figure 20: LS-SVM regression with the RBF kernel for different pairs of parameter values. The function estimation (red lines) together with the test set (blue dots) is shown. The value of the parameters and the MSE are shown for each figure.

algorithm	MAE	time [s]
gridsearch	0.0788	1.2961
simplex	0.0787	0.7961

Table 2: Table showing the average Mean Absolute Error (MAE) and time elapsed when using different tuning algorithms.

2.2.2 Application of the Bayesian framework

Q7: *Discuss in a schematic way how parameter tuning works using the Bayesian framework.*

A7: The parameter tuning using a Bayesian framework has three levels of inference. First level is the inference of the parameters support values α's and bias term b. The second level is the inference of the regularization parameter γ and finally in the third level, it's the inference of the kernel parameter σ^2 in the case of the RBF kernel.

2.3 Automatic Relevance Determination

A three dimensional dataset is generated from a Gaussian distribution, thereafter a function takes one of the dimensions as input. This can be thought of as we have a dataset with three features but only of these features is actually relevant for predicting the data. Automatic Relevance Determination (ARD) which uses the Bayesian framework can be used for selecting the most relevant input.

Q8: *Visualize the results in a simple figure.*

A8: The plots in figure 21 shows the true Y plotted against the generated output using the three dimensions as input respectively. We can see here that it is clear that the first feature was used as input and this is also what ARD returns.

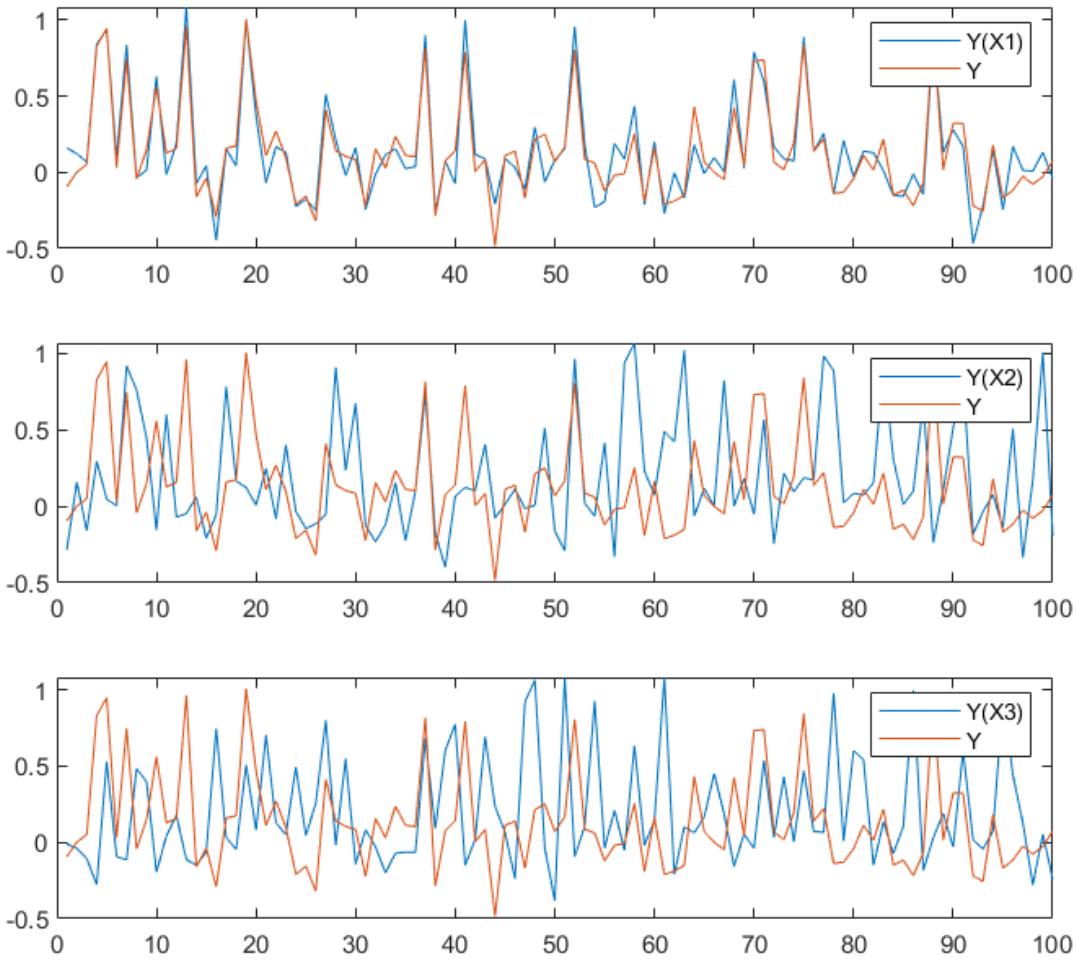


Figure 21: The true function Y and the generated functions using each feature as input.

Q9: *How can you do input selection in a similar way using the crossvalidate function instead of the Bayesian framework?*

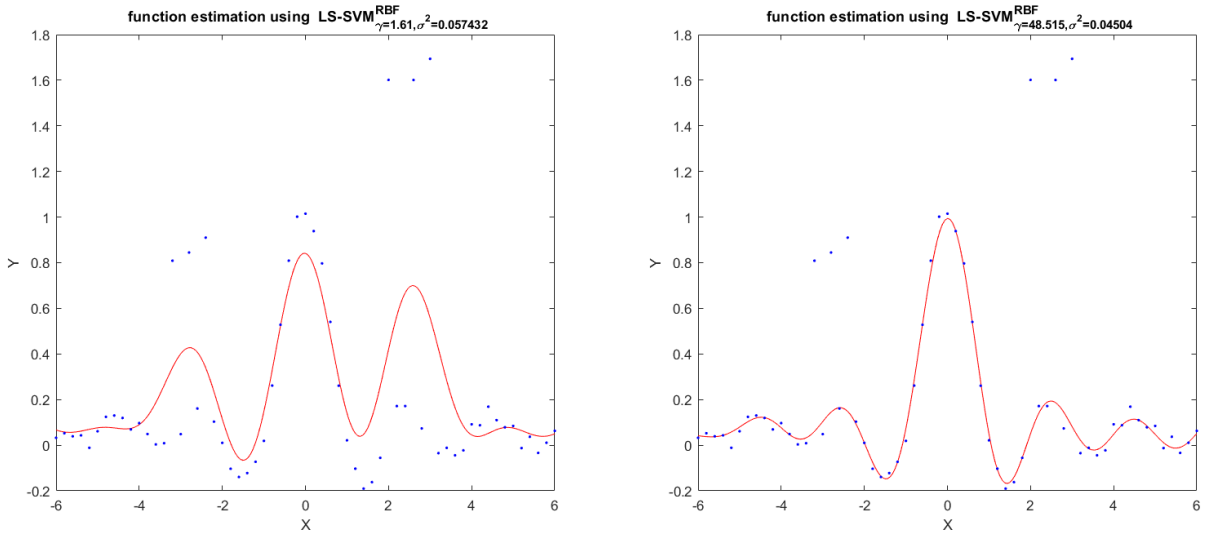
A9: To do this, one could use each dimension as input to the function then use crossvalidation and calculate the error between the true output and the generated output. The dimension with the smallest error is the most relevant input.

2.4 Robust regression

Outliers in the dataset can heavily influence the regression function in a bad way. To deal with this problem, robust regression can be applied.

Q10: *Visualize and discuss the results. Compare the non-robust version with the robust version. Do you spot any differences?*

A10: The plots in figure 22 show the regression model without and with taking outliers into consideration. They very clearly depict how the outliers can affect the function estimation when no robustness is used during regression. We can also see that the robust regression deals with outliers in an impressive way.



(a) (b)

Figure 22: Function estimation with outliers using (a) non-robust SVM regression and (b) using robust SVM regression.

Q11: *Why in this case is the mean absolute error MAE preferred over the classical mean squared error MSE?*

A11: The MSE is more sensitive to outliers than MAE. Meaning that the model will overcompensate because of the outliers if MSE is used, since the error will be greatly affected. An outlier won't affect the MAE as much which makes it more robust when dealing with outliers.

Q12: *Try alternatives to the weighting function wFun (e.g., 'whampel', 'wlogistic' and 'wmyriad'. Report on differences.*

A12: they all handle outliers well, there is not a big difference in the resulting plots.

2.5 Homework problems

Time series prediction on a process Z_t with $t = 1, 2, ..$ can be done using a linear autoregressive (AR) model, then it is given by:

$$\hat{z}_t = a_1 z_{t-1} + a_2 z_{t-2} + ... + a_n z_{t-n},$$

with $a_i \in \mathcal{R}$ for $i = 1, .., n$. Where n is the model order, meaning how many previous time steps are used to predict the following time step. The non-linear variant (NAR) is defined as

$$\hat{z}_t = f(z_{t-1}, z_{t-2}, ... + z_{t-n}).$$

2.5.1 Logmap dataset

Now we use regression to predict on the time series dataset **logmap**. We start by using the hyperparameters $\gamma = \sigma^2 = 10$ and order $n = 10$ to get a reference point. The resulting prediction compared to the true values are depicted in figure 24a, we see that the prediction is quite bad.

Q13: *As indicated numerous times before, the parameters γ and σ^2 can be optimized using crossvalidation. In the same way, one can optimize order as a parameter. Define a strategy to tune these 3 parameters.*

A13: To tune the three parameters the following scheme is used:

- For **order** in the range from 1 to 100
    1. Tune the hyperparameters γ and σ^2
    2. Train the model with these parameters
    3. Make predictions on the following time steps
    4. Compute the MSE between the prediction and the test set
    5. Repeat steps 1-4 five times to get the average MSE for each value of **order**

After this is done, the **order** with the lowest average MSE can be seen as optimal for this dataset. The average MSE for each value of **order** is visualized in figure 23. We can gather from that figure that the optimal value for the **order** parameter is somewhere between 20 and 50.

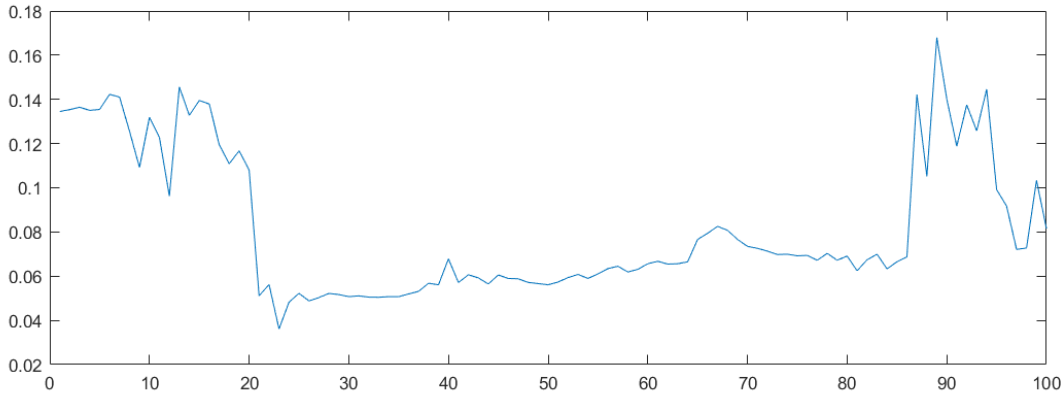
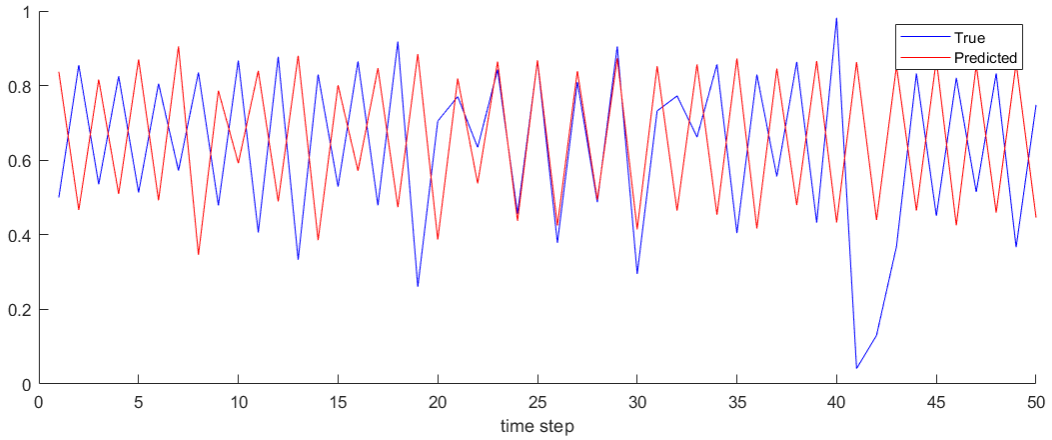


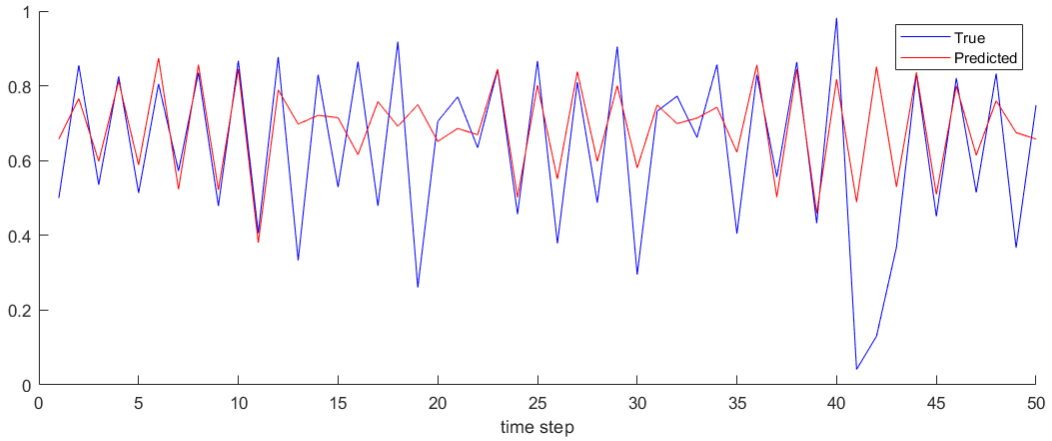
Figure 23: The average MSE for each value of the **order** parameter in the range of 1 and 100.

Q14: *Do time series prediction using the optimized parameter settings. Visualize your results. Discuss.*

A14: Using **order** value $n = 23$ and tuning the hyperparameters yield the result in figure 24b. The optimal parameters in this case was $\gamma \approx 10^6$ and $\sigma^2 \approx 10^3$. We can see that this prediction is better than our reference point although it still has problems with predicting the big drops. The tuned model is more aligned with the oscillation compared to the reference model where the prediction is lagging behind.



(a)



(b)

Figure 24: Time series prediction compared to the true target with (a) as the reference points using $\gamma = \sigma^2 = n = 10$ and (b) with the tuned parameters.

2.5.2 Santa Fe dataset

Now we do a similar task as in the previous section on the Santa Fe dataset. The dataset consists of 1200 data points where the first 1000 points acts as training data and the last 200 as a test set, see figure 25. So the goal is to train a model on the training data and predict the following 200 steps and compare it to the test data.

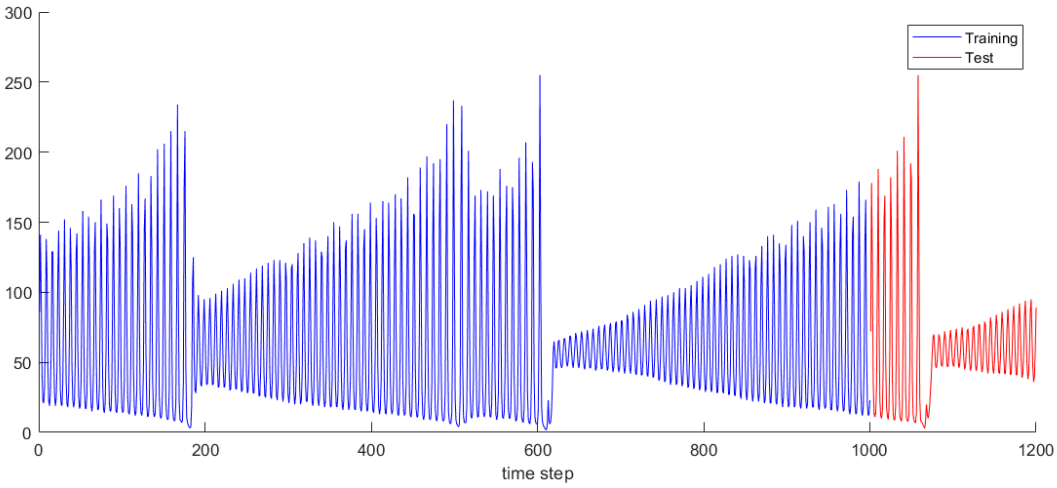


Figure 25: The Santa Fe dataset with the training data colored blue and the test data colored red.

Q15: *Does order* $= 50$ *for the utilized auto-regressive model sound like a good choice?*

A15: Using tuned parameters for *order* $= 50$ yields the predicted time series in figure 26 which looks like a good result. The only issue is that some of the later time steps seem to be lagging a bit.

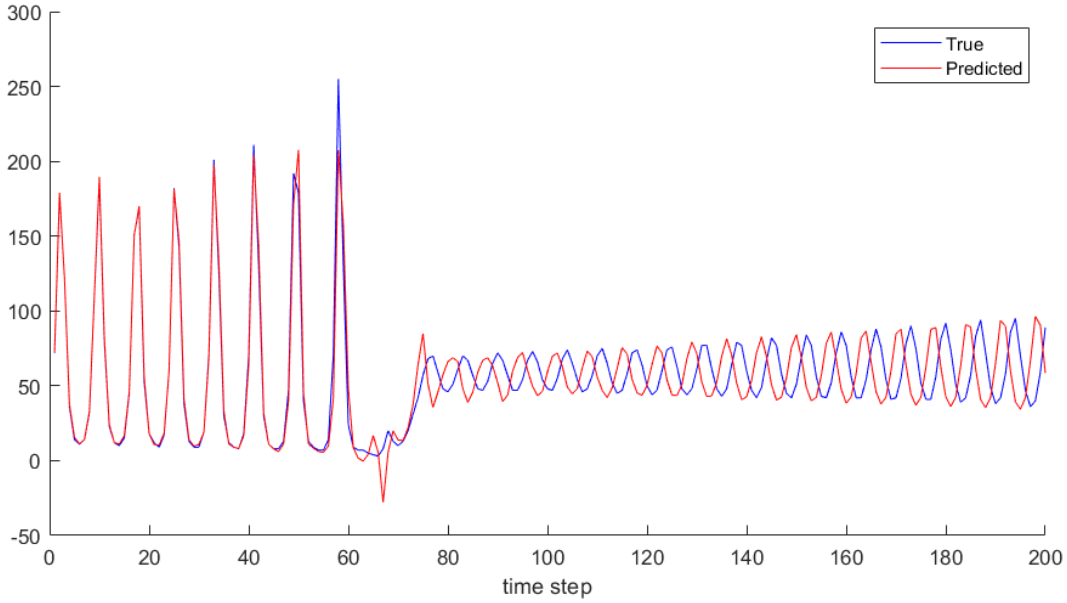


Figure 26: The Santa Fe dataset with the training data colored blue and the test data colored red.

Q16: *Would it be sensible to use the performance of this recurrent prediction on the validation set to optimize hyperparameters and the model order?*

A16: Optimizing on the validation set would make parameters fit the test set and therefore not make the model generalizable. Having a generalizable model is usually what one wants

Q17: *Tune the parameters (order,* γ *and* σ^2*) and do time series prediction. Visualize your results. Discuss.*

A17: The algorithm used in the previous section will again be applied here on the Santa Fe dataset, however since this is so computational heavy, we won't run five iteration for each value and and the value range for **order** will be between 20 and 60. All values in this range yielded a similar MSE so using *order* $= 50$ as before is actually one of the better choice here.

3 Exercise Session 3: Unsupervised Learning and Large Scale Problems

3.1 Kernel principal component analysis

Kernel principal component analysis (KPCA) is here going to be used for the purpose of denoising. To illustrate this, an artificial dataset of 400 points resembling the yin-yang pattern with given point dispersion will be used. This dataset is visualized in figure 27.

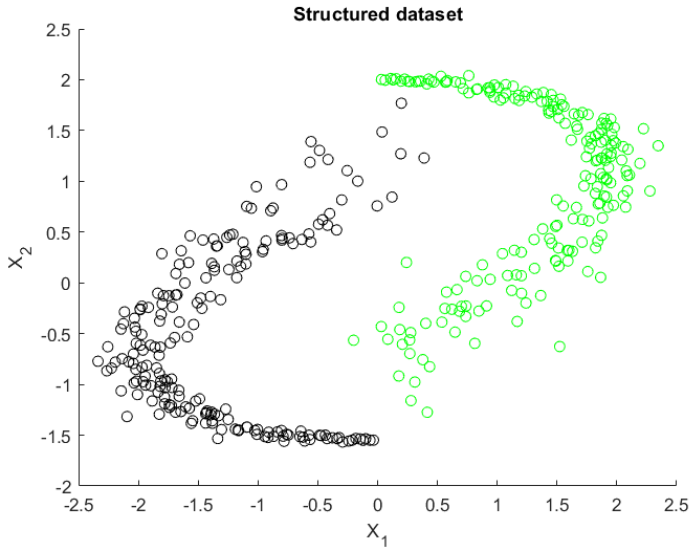


Figure 27: The artificial dataset resembling a yin-yang pattern.

Q1: *Describe how you can do denoising using PCA. Describe what happens with the denoising if you increase the number of principal components.*

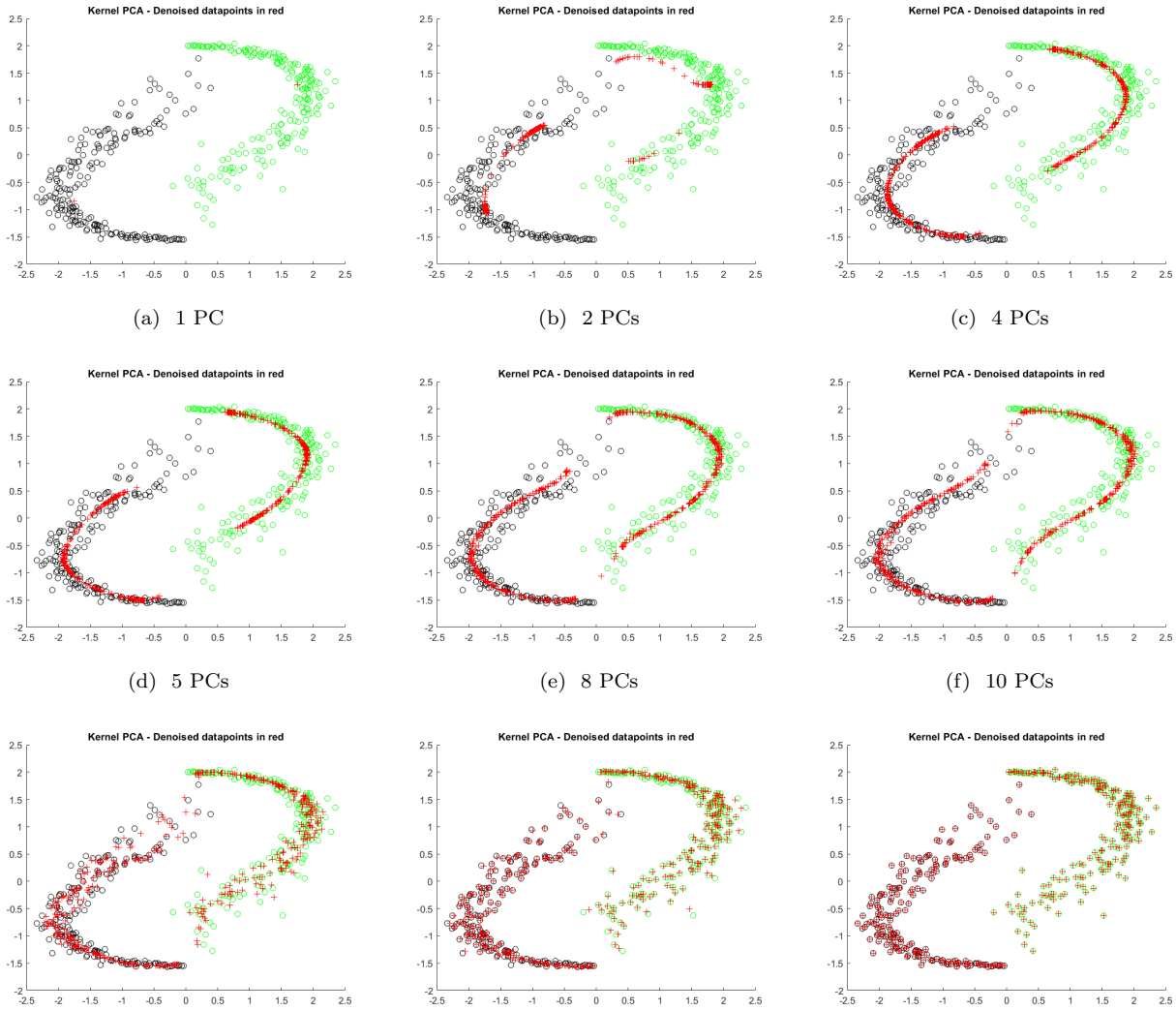
A1: Applying PCA on a noisy dataset means that we are eliminating principal components (PCs) with lower variance and keep those with most information (high variance). This means that we capture the general features of the dataset and ignore the specialized features which in turn will lead to noise being reduced. If the number of PCs is increased too much, more information of the dataset will be retrieved and it will therefore be looking more like the original dataset. So the number of PCs must be chosen small enough such that only the general features are gathered and the noise is ignored.

To test this, we apply KPCA on the yin-yang dataset by trying different amounts of PCs. Figures in 28 shows the result by applying KPCA with $1, 2, 4, 5, 8, 10, 15, 25$ and 100 PCs respectively. We can see that for example when using less than four PCs that it is not able to fully capture the pattern with so few components. Also when the number of PCs are more than ten we see that it is no longer a general model, but instead has so many components that it captures the noise as well. When using 100 components we basically see how the denoised data points coincides with the original dataset. A good amount of components seem to be in the range between four and ten, there it captures the general pattern and ignores the noise.

Q2: *Compare linear PCA with kernel PCA. What are the main differences? How many principal components can you obtain?*

A2: The goal of linear PCA is to project the data to a linear subspace in lower dimensions, KPCA however can find a non-linear subspace. KPCA does this by first mapping the data into a higher dimension using the kernel trick and thereafter projects it to a lower dimensional space. The amount of principal components one can obtain when using linear PCA is the same as the amount of features. For KPCA, the maximum amount of components that can be obtained is the same as the amount of data points.

Q3: *For the dataset at hand, propose a technique to tune the number of components, the hyperparameter and the kernel parameters.*



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

(d) 5 PCs (e) 8 PCs (f) 10 PCs

(g) 15 PCs (h) 25 PCs (i) 100 PCs

Figure 28: Applying KPCA on the noisy yin-yang dataset using different amount of principle components.

A3: One could use a gridsearch where different values are tested for each parameter and the objective to be minimized is the mean squared error between the true pattern and the denoised data points.

3.2 Spectral clustering

A dataset consisting of two rings in three-dimensional space is used in this example of spectral clustering.

Q4: *Explain briefly how spectral clustering works*

A4: The goal of spectral clustering is to reduce dimensionality of a dataset before clustering. This is done by using the similarity matrix of the data. One takes the Laplacian matrix of the similarity matrix and by using the eigenvectors of the Laplacian one can create groups of data points that are similar. Finally a clustering method (e.g. k-means clustering) can be used to divide the points into clusters.

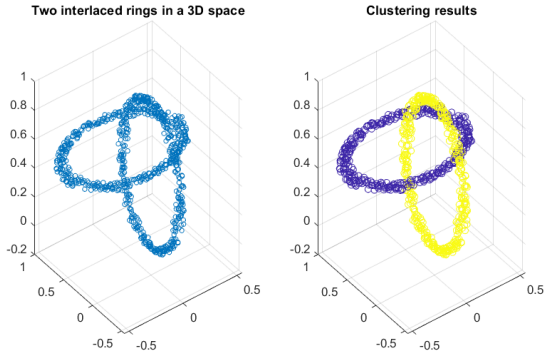
Q5: *What are the differences between spectral clustering and classification?*

A5: Clustering methods is a type of unsupervised learning which means that there is no prior knowledge about class labels and such. The clustering is based on the similarities between points and each cluster contains data points that are similar and objects from different clusters are dissimilar. Classification on the other hand uses class labels to learn a model to categorize data points in a supervised way.

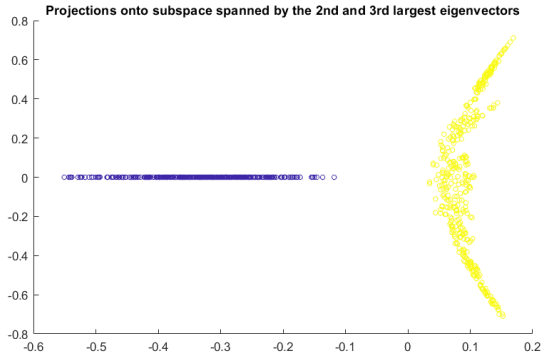
Q6: *Edit the script and try different values of σ^2. What is the influence of the σ^2 parameter on the clus-*

tering results?

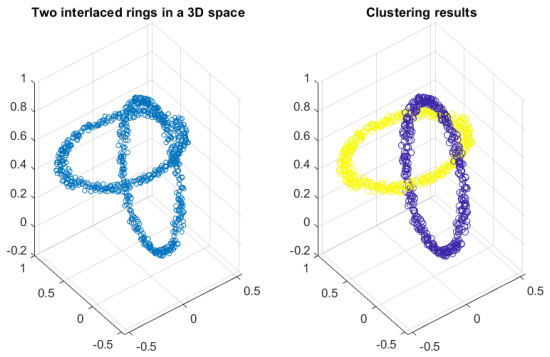
A6: The figures in 29 show the spectral clustering using σ^2 as 0.001, 0.01 and 0.1 both in their original dimension as well as in the projected subspace spanned by the two largest eigenvectors. We can see how the rings are in separate cluster when σ^2 is both 0.001 and 0.01, in the spanned subspace we can further see that the clusters are more separated in the case where $\sigma^2 = 0.01$. The rings are not separated when $\sigma^2 = 0.1$, when σ^2 is this big, the allowance for misclassification is too big to get a good result.



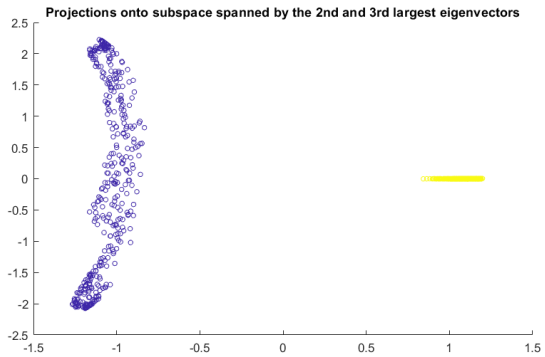
(a) $\sigma^2 = 0.001$



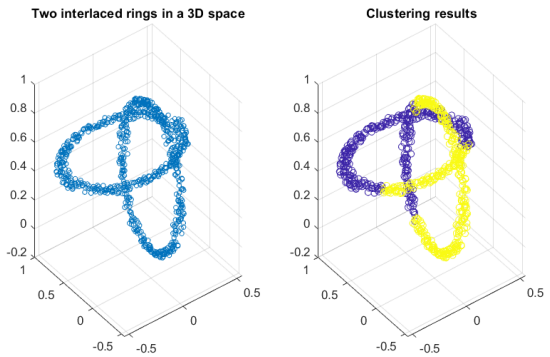
(b) $\sigma^2 = 0.001$



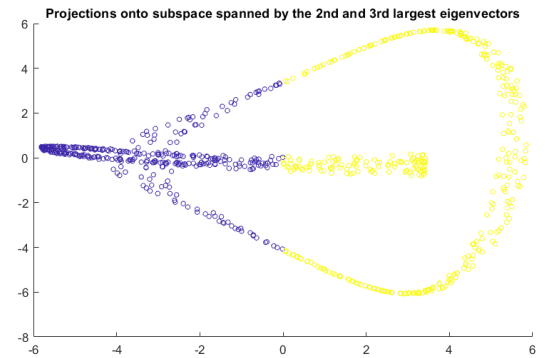
(c) $\sigma^2 = 0.01$



(d) $\sigma^2 = 0.01$



(e) $\sigma^2 = 0.1$



(f) $\sigma^2 = 0.1$

Figure 29: Spectral clustering with (a)-(b): $\sigma^2 = 0.001$, (c)-(d): $\sigma^2 = 0.01$ and (e)-(f): $\sigma^2 = 0.1$. The left column shows the original data and the data after clustering visualized with color. The right column show the projection onto the subspace spanned by the two largest eigenvectors, also visualized with color.

3.3 Fixed-size LS-SVM

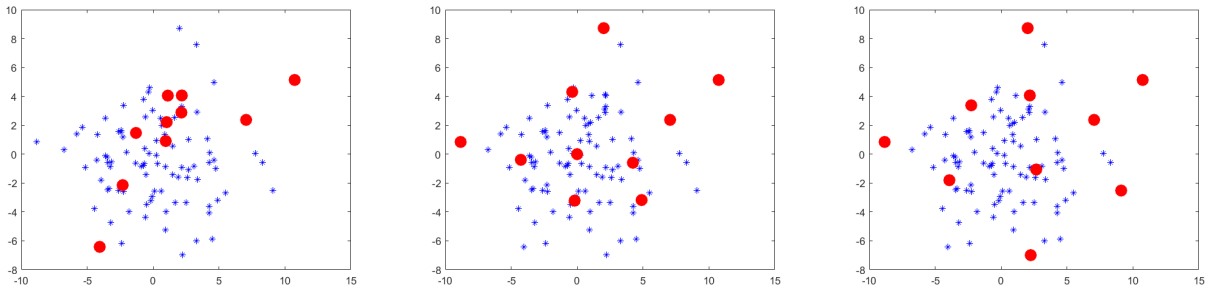
The fixed-sized LS-SVM is convenient when dealing with large-scale datasets. The amount of support vectors are fixed in advance and selected from the training data based upon the entropy criterion.

Q7: *In which setting would one be interested in solving a model in the primal? In which cases is a solution in the dual more advantageous?*

A7: Solving a model in the dual space means that the amount of unknown parameters is proportional to the data points. When dealing with very large datasets, this can become a big problem. The amount of unknown parameters when solving a model in primal space is instead proportional to the feature dimensions, so this is interesting when dealing with large-scale datasets. However the solution in the dual is needed to obtain non-linearity.

Q8: *What is the effect of the chosen kernel parameter σ^2 on the resulting fixed-size subset of data points (see fixedsize_script1.m)? Can you intuitively describe to what subset the algorithm converges?*

A8: The selected subset when using a fixed-size LS-SVM using different values of σ^2 is visualized in figure 30. We can see that when σ^2 is increased, the distance between the points of the subset increases. The algorithm converges to the subset that describes the decision boundary based on the hyperparameter [2].



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

Figure 30: Using a fixed-size LS-SVM with different values for the σ^2 parameter where the red dots represent the support vectors.

Q9: *Run fslssvm_script.m. Compare the results of fixed-size LS-SVM to ℓ_0-approximation in terms of test errors, number of support vectors and computational time.*

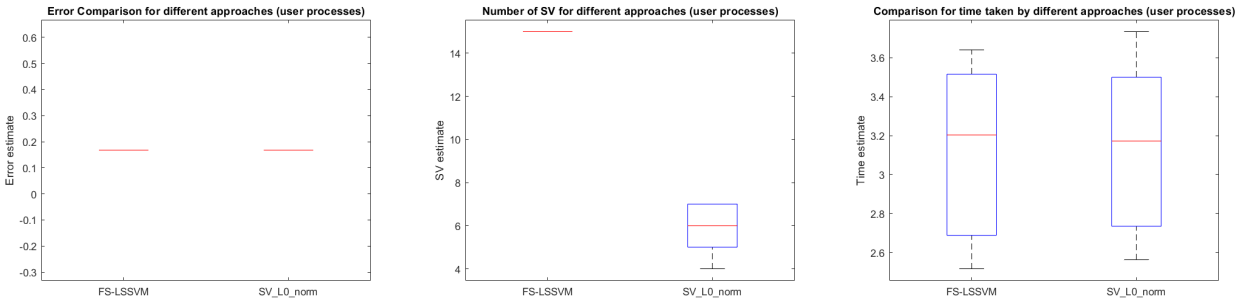
A9: The script was executed without changing any parameters and the resulting box-plots are shown in figure 31. We see that error for both methods are always the same, the computational time taken is also very similar in both methods. The big difference is the number of support vectors that are selected. The fixed-sized LS-SVM of course uses the same amount every time. When using ℓ_0-approximation we instead see a smaller amount used and a bit of variance between the iterations.

3.4 Homework problems

3.4.1 Kernel principal component analysis

Kernel PCA will be used to perform denoising on a dataset consisting of images of handwritten digits. The dataset has approximately 20 images per digit (0-9) where each image is a binary image. The parameter σ^2 is suggested to be calculated as the mean of the variances of each dimension multiplied with the dimension of the training data.

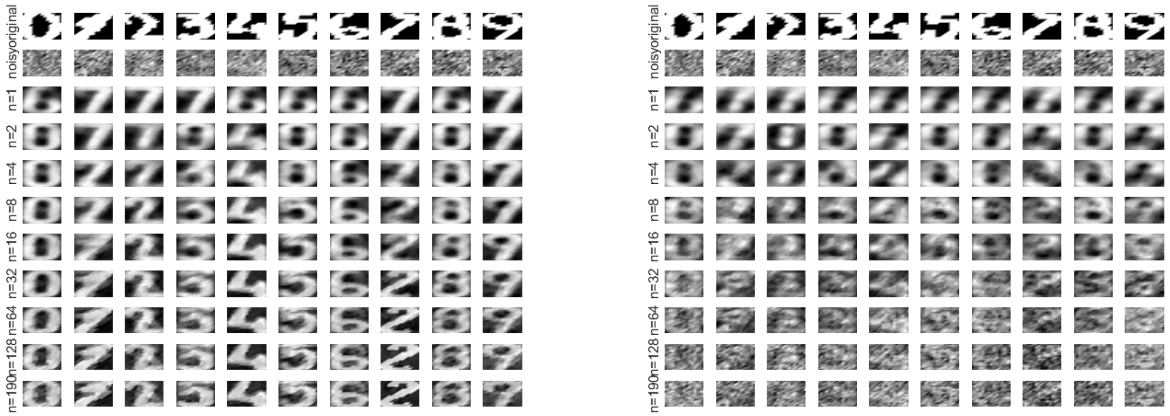
Q10: *Execute the script:* ***digitsdn.m****. Illustrate the difference between linear and kernel PCA by giving an example of digit denoising for noisefactor = 1.0. Give your comments on the results (based on visual inspection).*



(a) Error comparison (b) Number of support vectors comparison (c) Computational time comparison

Figure 31: The box-plots comparing the fixed-size LS-SVM with ℓ_0-approximation.

A10: The difference between linear PCA and kernel PCA applied on some noisy images of handwritten digits is visualized in figure 32. There we can see first the original image, the image with added noise and the reconstruction using different amounts of principal components. It is quite obvious that kernel PCA outperforms linear PCA in this case.



(a) Kernel PCA (b) Linear PCA

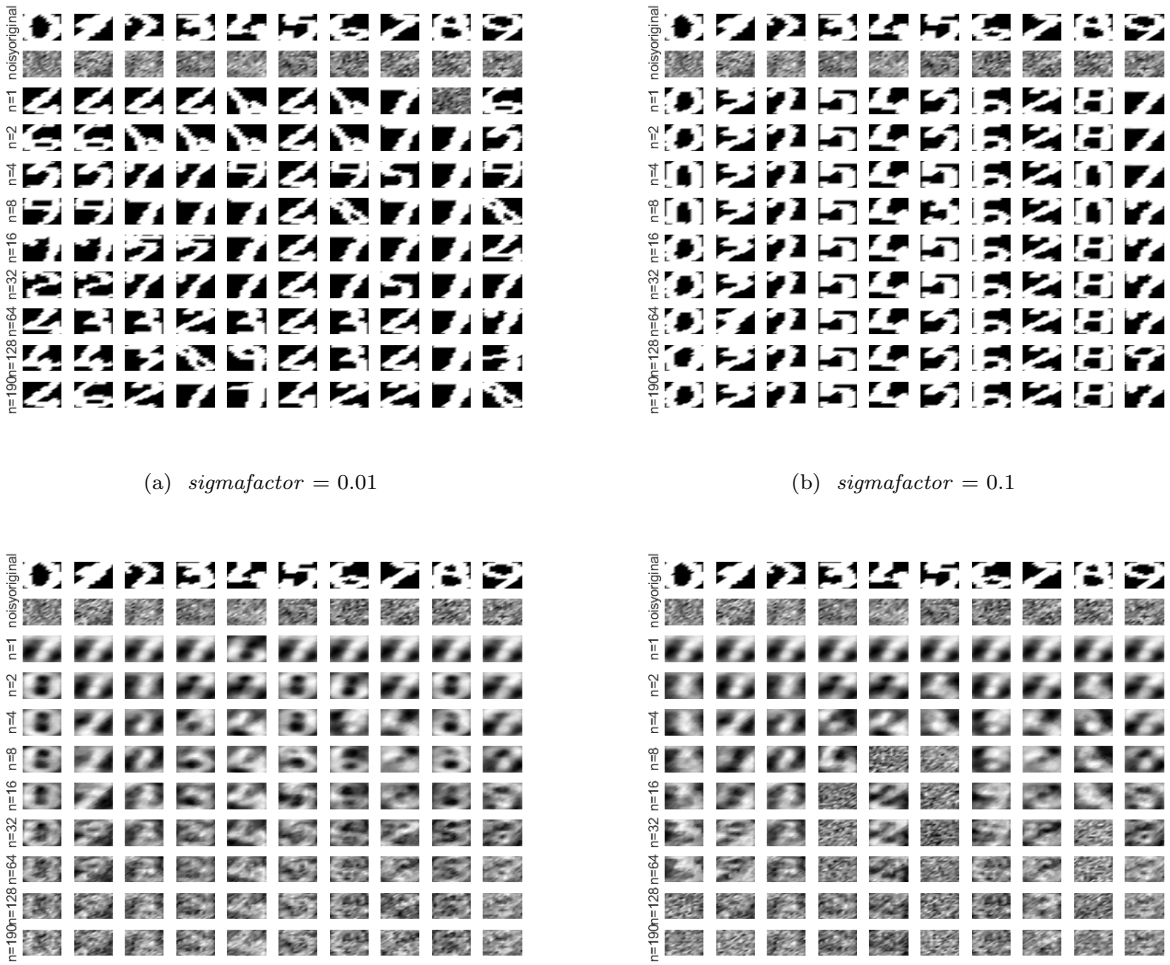
Figure 32: The denoising of images using different methods of PCA.

Q11: *What happens when the sig2 parameter is much bigger than the suggested estimate? What if the parameter value is much smaller? In order to investigate this, change the sigmafactor parameter for equispaced values in logarithmic scale.*

A11: To test this, we set the value of *sigmafactor* in the script to $0.01, 0.1, 10$ and 100 respectively. The denoised images for each *sigmafactor*-value is shown in figure 33. We notice that when sig2 is much smaller than the suggested estimate the noise in the images vanishes, however the model doesn't output the correct digits. When sig2 is much bigger than the suggestion, we see that using kernel PCA doesn't denoise the images well enough, and using too many components makes the images more noisy. These facts tells us that the choice of the sig2 parameter is very important to get a good result when denoising images.

Q12: *Investigate the reconstruction error on training (Xtest) and validation sets (Xtest1 and Xtest2), as a function of the kernel PCA denoising parameters. Select parameter values such that the error on the validation sets is minimal. Can you observe any improvements in denoising using these optimized parameter settings?*

A12: The reconstruction error was computed by taking the mean squared error between the reconstructed images and the true images in the validation sets. To get the parameter values when this error was mini-



(a) *sigmafactor* = 0.01

(b) *sigmafactor* = 0.1

(c) *sigmafactor* = 10

(d) *sigmafactor* = 100

Figure 33: The denoising of images using Kernel PCA with different values for the parameter *sigmafactor*.

mal we tested a grid of values for both the *sigmafactor* parameter and the amount of extracted components. The grid chosen for *sigmafactor* was based upon the previous result that the performance was bad both when $sigmafactor \leq 0.1$ and $sigmafactor \geq 10$. So the grid was chosen as equidistant values in between these values. The grid for the amount of extracted components was chosen as $[1, 2, 4, 8, 16, 32, 64, 128, 190]$, the same as before. Finally after iterating over all these combinations of values we found that the parameters that gave the best reconstruction error was when $sigmafactor = 0.61$ and extracting 128 principal components. The result with these values on the validation sets are shown in figure 34 and we see an improvement compared with the initial values used in figure 32.

3.4.2 Fixed-size LS-SVM

The fixed-size LS-SVM will be further investigated here on two other datasets: the Shuttle dataset and the California housing dataset.

3.4.2.1 The Shuttle dataset

Q13: *Explore and visualize (part of) the dataset. How many data points? How many and meaning of attributes? How many classes? What is to be expected about classification performance?*

A13: This dataset consists of 58000 data points with nine numerical features and one class label. There is in total seven classes that a data point can belong to. We know that 80% of the data points belong to the first class so we should expect to get at least 80% accuracy.

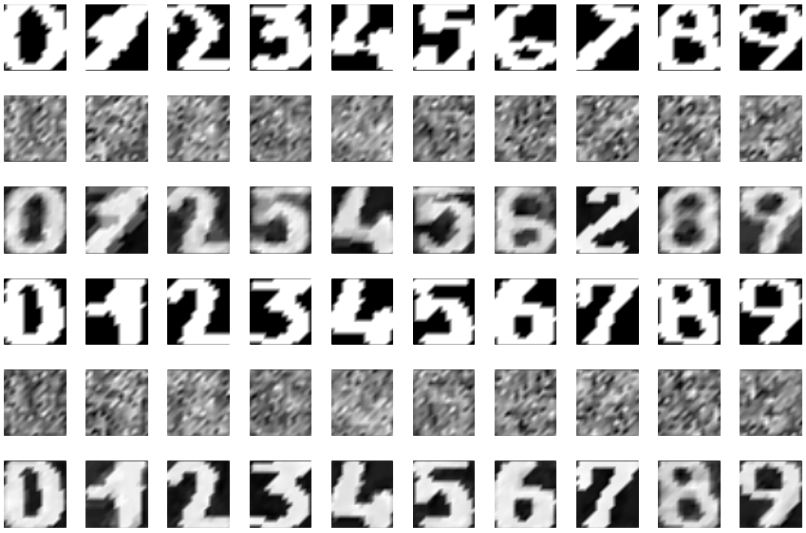
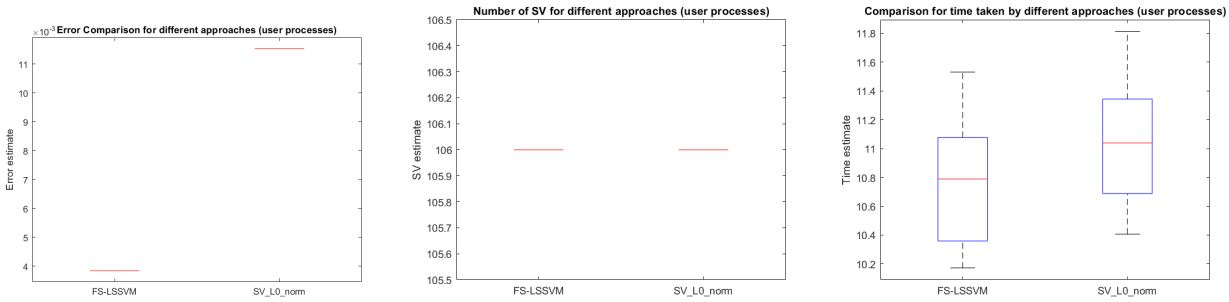


Figure 34: The denoising of the images in the validation sets. First row shows the original images in the first validation set, second row is the same images with added noise, third row shows the denoised images with optimal parameters. Rows 4-6 show the same for the second validation set.

Q14: *Visualize and explain the obtained results.*

A14: The script **fslssvm_script.m** is executed using the first 780 points of the shuttle dataset, since the computational time would be too large if the whole dataset was used. A polynomial kernel is used and the resulting box-plots can be seen in figure 35 showing the comparison between the fixed-size LS-SVM and using ℓ_0-approximation. The compared metrics are the error, the number of support vectors used and the computational time for both methods. We can see that the error rate for the fixed-size LS-SVM is lower and is only about 0.4% whereas using the ℓ_0-approximation we get an error rate of around 1.2%. The amount of support vectors are the same for both methods and the computational time is slightly faster for the fixed-size LS-SVM



(a) Error comparison (b) Number of support vectors comparison (c) Computational time comparison

Figure 35: The box-plots comparing the fixed-size LS-SVM with ℓ_0-approximation on the Shuttle dataset.

3.4.2.2 The California housing dataset

Q15: *Explore and visualize (part of) the dataset. How many data points? How many and meaning of attributes?*

A15: This dataset consists of 20640 data points of 9 features where all refer to information about the households in a neighborhood and its inhabitants. The last variable is the median house value in a neighborhood which

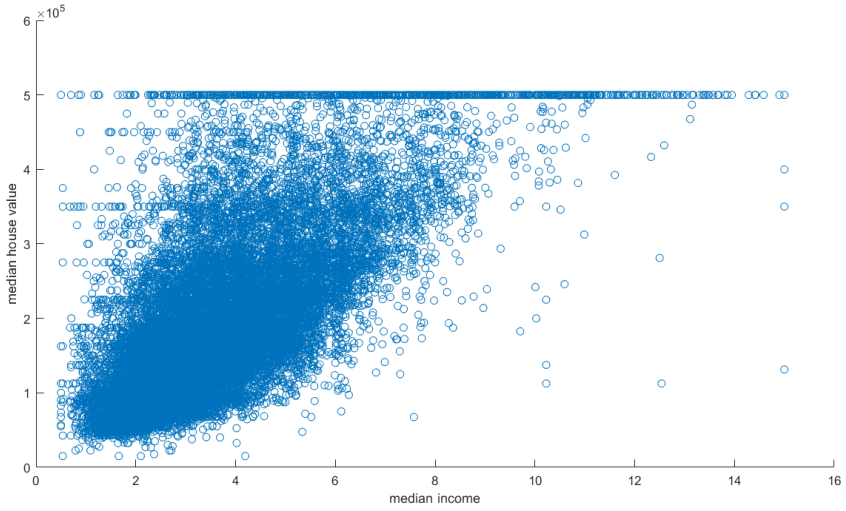
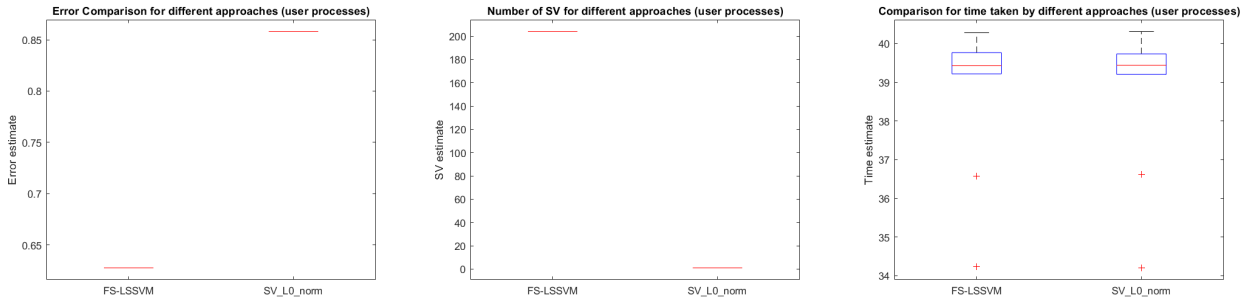


Figure 36: A plot showing how the median housing price in a neighbourhood depends on the median income in that same neighbourhood.

depends on all other variables. The dependency between the median income and the median house value in a neighborhood is shown in figure 36.

Q16: *Visualize and explain the obtained results.*

A16: The script is executed using regression this time with the first 1000 data points of the California dataset. The resulting box-plots are visualized in figure 37. Now with this dataset, we see that the fixed-sized LS-SVM has a lower error than the case with ℓ_0-approximation, also the amount of support vectors in that is higher. The computational time is about equal.



(a) Error comparison (b) Number of support vectors comparison (c) Computational time comparison

Figure 37: The box-plots comparing the fixed-size LS-SVM with ℓ_0-approximation on the California housing dataset.